403-RDBMS (Relational Database Management System)

M.Com (CA) Semester-IV

Introduction data – information – Advantages of DBMS – Data Modeling using the ER Model – The Relational Model – Relational Database Language – Transaction Processing Concepts.

M.C.Mouli, Department of Commerce & Computer

MASTEROFCOMMERCE (COMPUTERAPPLICATIONS)-FOURTHSEMESTER
403- RELATIONAL DATABASEMANAGEMENTSYSTEMS
(For M.Com-Computer Applications- under CBCS)
Class Hours: 4 ppw Credits: 4

UNIT-I: Introduction: Data Vs Information , Database and Database Users, Characteristics of the Database Approach, Implications of Database Approach, Advantages of using DBMS, Database System Concepts, Data Models, Schemas, and Instances. DBMS Architecture and Data Independence. The Database System Environment, Classification of DBMS.

UNIT-II: Data Modeling Using the Entity-Relationship Model: Entity types, Entity sets, attributes, and Keys, ER Model Concepts, Notation for ER Diagrams, Proper naming of Schema Constructs, Relationship types, degrees and cardinalities . Business Rules, Enhanced ER Model– Representing Super type Sub types, Representing Generalization and Specialization, Specifying constraints.

UNIT-III: The Relational Model: Integrity constraints, Relational tables, transforming EER diagrams into relations, Functional Dependencies and Normalization for Relational Database: Functional Dependencies, Normal Forms Based on Primary Keys, General Definitions of Second and Third Normal Forms Based on Primary Keys, Boyce Codd Normal Form, DE normalization Relational Algebra, File Organization techniques.

UNIT-IV: Relational Database Language: Data definition in SQL, Queries in SQL, Insert, Delete and Update Statements in SQL, Views, joins in SQL, sorting and grouping in SQL, specifying indexes, Query optimization - strategies – Query decomposition.

UNIT-V: **Transaction Processing Concepts:** Introduction, Transaction and System Concepts, Desirable properties of transaction, Schedules and Recoverability, Serializability of Schedules, Transaction Support in SQL, Locking Techniques for Concurrency Control, Concurrency Control based on time stamp ordering, Overview of distributed data bases Overview of access control mechanism.

Suggested Readings:

1. Jeffrey A Hoffer, Mary B. Prescott and Fred R McFadden, Modem Database Management, Pearson Education, New Delhi, 2010.
2. Abrahamsi. Silberschatz, Henry. F. Korth, S. Sudarshan, Database System Concepts, McGraw Hill, New Delhi, 2013.

References:

1. Gerald V. Post, Data Base Management Systems- Designing and Business Application, Tata McGraw Hill Company Limited, New Delhi, 2002.
2. Bipin C Desai, An Introduction to Database System, Galgotia Publications Private Limited, New Delhi, 2010.
3. CJ Date, An Introduction to Database Systems, Pearson/Addison Wesley Publishers, New Delhi, 2004.

*****

403-LAB: RELATIONAL DATABASEMANAGEMENTSYSTEMS

(For M.Com-Computer Applications- under CBCS)

Lab: 2 PPW: One Period Lab means 2 hours of Lab Session with SQL.

*****

1. The STUDENTDET AIL databases have a table with the following attributes.

STUDENT (regno: int, name: string, dob: date, marks: int)

i) Create the above table.

ii) Remove the existing attributes from the table.

iii) Change the data type of regno from integer to string.

iv) Add a new attribute phone number to the existing table.

v) Enter five tuples into the table.

vi) Display all the tuples of student table.

2. A LIBRARY database has a table with the following attributes.

LIBRARY (bookid:int, title:string, author:string, publication:string, yearpub:int, price:real)

i) Create the above table. Enter five tuples into the table

ii) Display all the tuples in student table.

iii) Display the different publishers from the list.

iv) Arrange the tuples in the alphabetical order of the book titles.

v) List the details of all the books whose price ranges between Rs. 100 and Rs. 300

3. The SALARY database of an organization has a table with the following attributes.

EMPSALARY (empcode: int, empname: string, dob: date, department: string, salary:real)

i) Create the above table. Enter five tuples into the table

ii) Display all the number of employees working in each department.

iii) Find the sum of the salaries of all employees.

iv) Find the sum and average of the salaries of employees of a particular department.

v) Find the least and highest salaries that an employee draws.

4. Consider the insurance database given below.

PERSON(driver-id-no: string, name: string, address: string)

CAR(regno: string, model: string, year: int)

ACCIDENT(report-no: int, date: date, location: String)

OWNS(driver-id-no: string, regno: string)

PARTICIPATED (driver-id-no: string, regno: string, report-no: int, damage-amount: int)

i) Create the above tables by properly specifying the primary keys and the foreign keys. Enter at least five tuples f or each relation.

ii) Demonstrate how you:

a) Update the damage amount for the car with a specific regno in the accident with Report no 12 to 25000.

b) Add a new accident to the database.

iii) Find total number of people who owned cars that were involved in accidents in 2012

iv) Find the number of accidents in which cars belonging to a specific model were involved.

5. Consider the following database of student's enrollment in courses and books adopted for each course.

STUDENT(regno: string, name: string, major: string, bdate: date)

COURSE(course-no: int cname: string, dept: string)

ENROLL(reg-no: string, course-no: int, sem: int, marks: int)

BOOK-ADOPTION (course-no: int, sem: int, book-isbn: int)

TEXT(book-isbn: int, book-title: string, publisher: string, author: string.

i) Create the above tables by properly specifying the primary keys and the foreign keys, Enter at least five tuples for each relation.
ii) Demonstrate how you add a new text book to the database and make this book be adopted by some department.
iii) Produce a list of text books (include Course-no, book-isbn, book-title) in the alphabetical order for Courses offered by the 'Computer Science' department that use more than two books.
iv) List any department that has all its adopted books published by a specific publisher.

6. The following tables are maintained by a book dealer

AUTHOR (author-id: int, name: string, city: string, country: string)
PUBLISHER (publisher-id: int name: string, city: string, country: string)
CATLOG (book-id: int, title : string, author-id: int, publisher-id: int, category: int, year: int, price: int)
CATEGORY (category-id: int, description: string)
ORDER-DETAILS (order-no: int, book-id: int, quantity: int)

i) Create above tables by properly specifying the primary keys and the foreign keys. Enter at least five tuples for each relation.
ii) Give the details of the authors who have 2 or more books in the catalog and the price of the books is greater than the average price of the books in the catalog and the year of publication is after 2010.
iii) Find the author of the book which has maximum sales.
iv) Demonstrate how to increase price of books published by specific publisher by 10%

7. Consider the following database for BANK.

BRANCH (branch-name: string, branch-city: string, assets: real)
ACCOUNT (accno: int, branch-name: string, balance: real)
DEPOSITOR (customer-name: string, accno: int)
CUSTOMER (customer-name: string, customer-street: string, customer-city: string)
LOAN (loan-no: int, branch-name: string, amount: real)
BORROWER (customer-name: string, loan-no: int)

i) Create the above tables by properly specifying the primary keys and foreign keys. Enter at least five tuples for each relation.
ii) Find all the customers who have at least two accounts at the main branch.
iii) Find all customers who have an account at all the branches located in a specific city.
iv) Demonstrate how to delete all account tuples at every branch located in specific city.

8. Consider the following database for ORDER PROCEESING.

CUSTOMER (cust-no: int, cname: string, city: string)
ORDER (orderno: int, odate: date, ord-amt: real)
ORDER_ITEM (orderno: int, itemno:int, qty: int)
ITEM (itemno: int, unit price: real)
SHIPMENT (orderno: int, warehouseno: int, ship-date: date)
WAREHOUSE (warehouseno: int, city: string)

i) Create the above tables by properly specifying the primary keys and the foreign keys. Enter at least five tuples for each relation.
ii) List the order number and ship date for all orders shipped from particular warehouse.
iii) Produce a listing: customer name, no of orders, average order amount.
iv) List the orders that were not shipped within 30 days of ordering

*****

1. Introduction: Data Vs Information:

Introduction: Databases are used to store, manipulate and retrieve data in all most all types of organizations such as education, business, health care, government and libraries. Database technology is used by individuals on personal computers and by work groups on network computers.

Databases will certainly continue to grow, as there is highly competitive environment among organizations. Managers are seeking to use knowledge derived from databases to face the competition. For example, detailed sales database can be used as the basis for advertising and marketing. Many organizations are building separate databases, called "data ware houses", for this type of decision support application.

Databases can be used in Customer relationship management, On-line shopping, On-line reservation system i.e. Air ticket, Railway ticket reservation, Employee relationship management, Personal digital assistants, Information appliances, Enterprise wide information systems, Client/Server Applications, Computerization of: Electricity Billing System, Water Billing System, Inventory Management system and library Management system etc.
Although the future of database is assured, much work remains to be done by organizations. New skills are required to design data warehouses. Those skills include database analysis, database design, and database administration.

Data refers to raw, unorganized facts or figures. It can be in the form of text, numbers, images, sounds, or any other format. Data, on its own, lacks context and meaning. For example, a list of numbers or a series of characters in a database is data.

Information, on the other hand, is the result of organizing, analyzing, and interpreting data to give it context and meaning. Information provides insights, answers questions, or helps make decisions. It is data that has been processed and structured in a way that is meaningful to the user. For instance, a report summarizing sales figures for the past quarter or a graph showing trends over time is information derived from data.

Metadata: Data that describe the properties or characteristics of other data.

Data	Metadata	Information
Data is a raw collection of characters or text, facts etc.	Metadata is data about data.	Information is meaningful desired in an organized form.
Data exists with the user.	Metadata is required for entering data	Information is required to the user.
It requires process.	It requires defining using languages such as SQL.	This is processed data

Ravi	33	88	74	87
Kiran	44	98	87	65

Data →

MetaData

Name (Varchar2)	Rno (Number)	M1 (Number)	M2 (Number)	M3 (Number)	Total (Number)
Ravi	33	88	74	87	249
Kiran	44	98	87	65	250

Information

2. Database and Database Users:

Database: A database is a structured collection of data that is organized and stored electronically in a computer system. It typically consists of tables, each containing rows and columns where data is stored. Databases are designed to efficiently manage and retrieve data, providing mechanisms for adding, modifying, deleting, and querying information.

There are various types of databases, including relational databases, NoSQL databases, object-oriented databases, and more. Each type has its own strengths and is suitable for different types of data and applications.

Database Users: Database users are individuals or entities who interact with the database in various ways. They can be broadly categorized into different roles based on their interactions with the database:

1. **Database Administrators (DBAs):** DBAs are responsible for managing and maintaining the database system. Their tasks include installation, configuration, monitoring performance, ensuring data security, backup and recovery, and troubleshooting issues.
2. **Database Developers:** Database developers design, implement, and maintain the database schema, which includes creating tables, defining relationships, and optimizing database performance. They also write queries, stored procedures, and other database code to manipulate and retrieve data.
3. **End Users:** End users are the individuals who interact with the database to perform their daily tasks. They may include employees of an organization, customers, or other stakeholders. End users typically use applications or interfaces that interact with the database to input, retrieve, and manipulate data.
4. **Casual Users:** These users have limited interaction with the database and may only perform simple tasks such as viewing reports or entering data through forms.
5. **Power Users:** Power users have more extensive knowledge of the database and its capabilities. They may create ad-hoc queries, generate custom reports, or perform data analysis using tools provided by the database system.
6. **Executive Users:** Executive users are typically decision-makers within an organization who rely on database-derived information for strategic planning and decision-making.

Understanding the different types of database users and their roles is essential for designing database systems that meet the needs of the organization and its stakeholders. It ensures that the database is accessible, secure, and optimized for performance.

3. Characteristics of the Database Approach:

The database approach to managing data offers several characteristics that distinguish it from other methods of data management:

1. **Data Independence:** The database approach provides both logical and physical data independence. Logical data independence means that changes to the logical structure of the database (like adding or modifying tables) do not require changes to the application programs that access the data. Physical data independence means that changes to the physical storage of the data (like moving to a different storage system) do not affect the logical structure or the application programs.
2. **Data Integration:** Databases allow for the integration of data from various sources into a single, unified repository. This integration enables users to access and analyze data from different parts of an organization without the need for redundant data storage or data inconsistency.
3. **Data Integrity:** Databases enforce data integrity constraints to ensure the accuracy, consistency, and reliability of the data. This includes mechanisms such as primary key

constraints, foreign key constraints, unique constraints, and check constraints to maintain the quality of the data.

4. **Data Security:** Database systems provide robust security features to protect data from unauthorized access, modification, or deletion. This includes user authentication, access control, encryption, and auditing capabilities to safeguard sensitive information.
5. **Concurrent Access and Transactions:** Database systems support concurrent access to data by multiple users or applications while maintaining data consistency and integrity. Transactions ensure that a series of database operations are executed atomically, consistently, and durably, even in the presence of failures.
6. **Data Abstraction:** The database approach allows for data abstraction, which hides the complex implementation details of the data storage and retrieval mechanisms from the users and applications. This abstraction is achieved through various levels, including the conceptual, logical, and physical levels.
7. **Data Query Language:** Database systems provide a data query language (e.g., SQL - Structured Query Language) that allows users to retrieve, manipulate, and manage data stored in the database. This standardized language simplifies data access and manipulation tasks.
8. **Data Redundancy Reduction:** By storing data in a centralized repository and minimizing redundancy, the database approach reduces the likelihood of inconsistencies and data anomalies that can occur in systems where data is duplicated across multiple files or applications.
9. **Scalability and Performance:** Database systems are designed to handle large volumes of data and support scalability to accommodate growing data needs. Additionally, they provide features such as indexing, query optimization, and caching to improve performance and responsiveness.

Overall, the database approach provides a structured and efficient way to manage data, offering benefits in terms of data integrity, security, integration, and accessibility. These characteristics make it a cornerstone of modern information systems across various industries.

4. Implications of Database Approach

The database approach has several implications across various aspects of data management, information systems, and organizational operations:

1. Data Centralization: With the database approach, organizations centralize their data in a single repository. This allows for better control, management, and security of data. However, it also means that any disruptions or failures in the database system can have significant impacts on the entire organization's operations.

2. Improved Data Quality and Consistency: By enforcing data integrity constraints and reducing data redundancy, the database approach helps improve data quality and consistency. This ensures that users can rely on the accuracy and reliability of the information stored in the database.

3. Enhanced Data Accessibility: Databases provide mechanisms for efficient data retrieval and manipulation, allowing authorized users to access and analyze data quickly and easily. This accessibility facilitates decision-making, reporting, and other data-driven tasks within the organization.

4. Increased Productivity: The database approach streamlines data management processes, reducing the time and effort required to access, update, and analyze data. This increased productivity allows employees to focus on higher-value tasks rather than dealing with manual data handling.

5. Support for Decision-Making: Databases provide timely and accurate information that supports decision-making at all levels of the organization. Decision-makers can rely on data-driven insights to make informed choices, leading to better outcomes and strategic planning.

6.Scalability and Flexibility: Database systems are designed to scale with the organization's data needs. They can handle growing volumes of data and adapt to changing requirements, ensuring that the organization's data infrastructure remains robust and flexible over time.

7.Costs and Investments: While the database approach offers numerous benefits, implementing and maintaining a database system requires significant investments in terms of infrastructure, software, personnel, and training. Organizations must carefully consider the costs and benefits to ensure a positive return on investment.

8. Data Security and Privacy: Centralizing data in a database requires robust security measures to protect sensitive information from unauthorized access, breaches, or data loss. Organizations must implement security protocols, access controls, encryption, and regular audits to safeguard their data.

9. Interoperability and Integration: Databases facilitate data integration from various sources, enabling interoperability between different systems and applications. This integration enhances collaboration, data sharing, and communication across departments or external partners.

10. Regulatory Compliance: Many industries are subject to regulatory requirements regarding data management, privacy, and security. The database approach helps organizations comply with these regulations by providing mechanisms for data governance, audit trails, and compliance reporting.

In summary, the database approach offers numerous advantages for organizations in terms of data management, accessibility, and decision-making. However, it also entails challenges such as cost, security, and regulatory compliance that must be addressed to fully realize its benefits.

5. Advantages of using DBMS

Using a Database Management System (DBMS) offers several advantages for organizations:

- **Data Centralization:** DBMS centralizes data in a single repository, making it easier to manage, access, and secure. This eliminates the need for redundant data storage and ensures data consistency and integrity.
- **Data Integrity and Consistency:** DBMS enforces data integrity constraints such as primary keys, foreign keys, and data validation rules, ensuring that data remains accurate, consistent, and reliable throughout its lifecycle.
- **Improved Data Security:** DBMS provides robust security features, including user authentication, access control, encryption, and audit trails, to protect sensitive information from unauthorized access, breaches, or data loss.
- **Concurrent Access and Transactions:** DBMS supports concurrent access to data by multiple users or applications while ensuring data consistency and integrity through transaction management. This allows for efficient collaboration and avoids data conflicts.
- **Data Independence:** DBMS provides both logical and physical data independence. Logical data independence allows changes to the database schema without affecting application programs, while physical data independence enables changes to the physical storage without impacting the logical structure or applications.
- **Data Abstraction:** DBMS hides the complex implementation details of data storage and retrieval from users and applications through data abstraction. This allows users to interact with the database using high-level languages and concepts without needing to understand the underlying technical complexities.
- **Data Query Language:** DBMS provides a standardized data query language, such as SQL (Structured Query Language), which allows users to retrieve, manipulate, and manage data efficiently. SQL simplifies data access and manipulation tasks, making it easier to work with databases.
- **Scalability and Performance:** DBMS is designed to handle large volumes of data and support scalability to accommodate growing data needs. It also provides features such as indexing, query optimization, and caching to improve performance and responsiveness.

- **Data Backup and Recovery:** DBMS includes mechanisms for data backup and recovery, allowing organizations to recover data in the event of hardware failures, human errors, or disasters. Regular backups ensure data availability and minimize downtime.
- **Reduced Data Redundancy:** DBMS minimizes data redundancy by storing data in a centralized repository and eliminating duplicate copies. This reduces storage requirements and the likelihood of inconsistencies or data anomalies.

Overall, using a DBMS offers numerous benefits, including improved data management, security, performance, and scalability, making it an essential component of modern information systems.

6. Database System Concepts:

Database system concepts encompass the fundamental principles and components that constitute a database management system (DBMS). Here are the key concepts:

- **Data Model:** A data model is a conceptual representation of the data and its relationships within the database. Common data models include the relational model, hierarchical model, network model, and object-oriented model.
- **Schema:** The database schema defines the structure of the database, including tables, columns, constraints, and relationships. It provides a blueprint for organizing and storing data.
- **Data Independence:** Data independence refers to the separation of data from the application programs and the physical storage details. It includes logical data independence (ability to change the schema without affecting applications) and physical data independence (ability to change storage structures without affecting applications).
- **Query Language:** A query language, such as SQL (Structured Query Language), allows users to interact with the database by querying, updating, and managing data. SQL provides a standardized way to communicate with the DBMS.
- **Transactions:** A transaction is a unit of work that consists of a sequence of database operations (e.g., read, write, update) that must be executed atomically, consistently, and durably. Transactions ensure data integrity and consistency.
- **Concurrency Control:** Concurrency control mechanisms manage simultaneous access to data by multiple users or transactions. They prevent data anomalies, such as lost updates and uncommitted dependencies, by coordinating the execution of transactions.
- **Data Integrity:** Data integrity ensures that data is accurate, consistent, and reliable. It is enforced through constraints, such as primary key constraints, foreign key constraints, and check constraints, which maintain the quality of the data.
- **Security:** Database security features protect data from unauthorized access, modification, or deletion. This includes user authentication, access control, encryption, and auditing to safeguard sensitive information.
- **Indexing and Optimization:** Indexing improves the performance of data retrieval operations by creating data structures (indexes) that allow for faster access to records. Query optimization techniques enhance query performance by selecting efficient execution plans.
- **Backup and Recovery:** Backup and recovery mechanisms ensure data availability and minimize downtime in case of hardware failures, disasters, or human errors. Regular backups are taken, and recovery procedures are in place to restore the database to a consistent state.
- **Data Dictionary:** The data dictionary or metadata repository stores information about the database structure, schema, constraints, and other database objects. It provides a centralized repository for managing and documenting database metadata.
- **Data Warehousing and OLAP:** Data warehousing involves the collection, storage, and management of large volumes of data from multiple sources for analysis and decision-

making. Online Analytical Processing (OLAP) tools facilitate multidimensional analysis of data stored in data warehouses.

Understanding these database system concepts is essential for designing, implementing, and managing efficient and reliable database systems that meet the needs of organizations and their users.

7. Data Models, Schemas, and Instances:

Understanding data models, schemas, and instances is fundamental to comprehending database system concepts:

a) Data Model: A data model is a conceptual representation of the data and its relationships within a database. It defines the structure, constraints, and rules for organizing and manipulating data. There are various types of data models, including:

i) **Relational Model:** Represents data as tables (relations) consisting of rows (tuples) and columns (attributes), with relationships defined by foreign keys.
ii) **Hierarchical Model:** Organizes data in a tree-like structure, where each record has a single parent record and may have multiple child records.
iii) **Network Model:** Extends the hierarchical model by allowing each record to have multiple parent and child records, forming complex networks of relationships.
iv) **Object-Oriented Model:** Treats data as objects, with attributes and methods, allowing for encapsulation, inheritance, and polymorphism.

b) Schema: A schema is a logical blueprint that defines the structure of the database, including tables, columns, data types, constraints, and relationships. It provides a formal description of the database's organization and ensures data consistency and integrity. There are several types of schemas:

i) **Conceptual Schema:** Represents the overall logical structure of the database, independent of any specific DBMS implementation. It describes entities, their attributes, and relationships.
ii) **Logical Schema:** Specifies the logical organization of data in terms of tables, columns, keys, and constraints. It defines how data is stored and accessed within the database.
iii) **Physical Schema:** Describes the physical storage structures and access methods used by the database system. It includes details such as file organization, indexing, and storage allocation.

c) Instance: An instance refers to the actual data stored in a database at a particular moment in time. It represents a snapshot of the database's contents, including all records, values, and relationships.

For example, in a relational database, an instance would be the specific rows and columns of data stored in the tables. Each row represents a record, and each column represents an attribute.

Instances can change over time as data is inserted, updated, or deleted. They provide a real-world representation of the data model defined by the schema.

In summary, data models define the structure and relationships of data, schemas provide a blueprint for organizing and storing data, and instances represent the actual data stored in the database at a given moment. Understanding these concepts is essential for designing, implementing, and managing database systems effectively.

8. DBMS Architecture and Data Independence:

DBMS Architecture: The architecture of a Database Management System (DBMS) typically consists of several layers, each responsible for different aspects of data management:

1. **External Level (View Level):** This layer is closest to the end-users and defines how data is presented and accessed. It includes user views or schemas, which provide a customized and

simplified view of the database for specific user groups or applications. Users interact with the database through queries and transactions formulated in a high-level language like SQL.

2. **Conceptual Level (Logical Level):** The conceptual level defines the logical structure of the entire database independent of any specific DBMS implementation. It includes the conceptual schema, which describes the organization of data using entities, attributes, relationships, and constraints. The conceptual schema is designed to represent the underlying data model, such as the relational model or the hierarchical model.
3. **Internal Level (Physical Level):** The internal level describes how data is physically stored and organized within the database system. It includes the physical schema, which specifies the storage structures, access methods, indexing techniques, and optimization strategies used by the DBMS. This level deals with details such as file organization, data storage formats, and buffer management.
4. **Storage Level:** The storage level represents the actual storage media and mechanisms used to store data on disk or in memory. It includes file systems, storage devices, caching mechanisms, and I/O operations. The DBMS interacts with the storage level to read and write data efficiently.
5. **Data Independence:** Data independence refers to the ability to make changes to the database schema or physical storage without affecting the application programs or the external view of the data. There are two types of data independence:
6. **Logical Data Independence:** Logical data independence allows changes to the conceptual schema (the logical structure of the database) without affecting the external schemas or application programs. It means that modifications to the database schema, such as adding or modifying tables or relationships, do not require changes to the user views or application code.
7. **Physical Data Independence:** Physical data independence allows changes to the physical schema (the physical storage and organization of data) without affecting the conceptual schema or external schemas. It means that modifications to the storage structures, such as reorganizing files, adding indexes, or changing storage devices, do not impact the logical structure or the way data is accessed by users or applications.

Data independence is essential for database systems because it allows for flexibility, scalability, and maintenance without disrupting the functionality or usability of the system. It enables applications to remain unaffected by changes to the underlying database structure or storage mechanisms, thereby reducing complexity and improving adaptability.

9. The Database System Environment, Classification of DBMS.

A) Database System Environment: The database system environment refers to the various components and interactions within a database system. It encompasses hardware, software, users, and data, as well as the interactions and processes that occur within the system. Here are the main components of the database system environment:

- **Hardware:** The physical equipment that hosts the database system, including servers, storage devices, and networking infrastructure.
- **Software:** The software components of the database system, including the DBMS software itself, operating systems, middleware, and any additional software applications or tools used for database management and development.

Database Management System (DBMS):

The software that facilitates the creation, management, and manipulation of databases. It includes components for data storage, retrieval, security, concurrency control, and query processing.

- **Users:** Individuals or entities who interact with the database system, including database administrators (DBAs), developers, end-users, and application programs.

- **Data:** The information stored in the database, organized according to the database schema. This includes data records, attributes, relationships, and constraints.
- **Database Applications:** Software applications or systems that use the database to perform specific tasks, such as online transaction processing (OLTP), data analytics, reporting, and decision support.
- **Database Operations:** The various operations performed on the database, such as querying, updating, inserting, deleting, and indexing data. These operations are carried out by users or application programs through the DBMS.
- **Database Administration:** The process of managing and maintaining the database system, including tasks such as database design, configuration, monitoring, backup and recovery, security management, and performance tuning.

B) Classification of DBMS: Database Management Systems (DBMS) can be classified based on several criteria, including data model, architecture, usage, and functionality. Here are some common classifications:

a)Based on Data Model:

i) **Relational DBMS (RDBMS):** Organizes data into tables with rows and columns, and establishes relationships between tables using keys.
ii) **Hierarchical DBMS:** Organizes data in a tree-like structure, where each record has a single parent record and may have multiple child records.
iii) **Network DBMS:** Extends the hierarchical model by allowing records to have multiple parent and child records, forming complex networks of relationships.
iv) **Object-Oriented DBMS (OODBMS):** Treats data as objects with attributes and methods, supporting encapsulation, inheritance, and polymorphism.

b) Based on Architecture:

i) **Centralized DBMS:** All database processing is performed on a single computer system.
ii) **Distributed DBMS (DDBMS):** Data is distributed across multiple computer systems or locations, with processing distributed among them.
iii) **Client-Server DBMS:** Data processing is divided between a client application and a server that manages the database.

c) Based on Usage:

i) **Operational DBMS (OLTP):** Designed for transaction processing and day-to-day operations, supporting high-volume, real-time transactions.
ii) **Analytical DBMS (OLAP):** Designed for data analysis and decision support, supporting complex queries and aggregations on large volumes of historical data.

d)Based on Functionality:

i) **Document-oriented DBMS:** Designed for storing, retrieving, and managing document-oriented data, such as JSON or XML documents.
ii) **In-memory DBMS:** Stores data primarily in memory for faster access and processing, suitable for high-performance applications.
iii) **Graph DBMS:** Designed for managing and querying graph-structured data, such as social networks or network topologies.

These classifications help in understanding the characteristics, capabilities, and suitability of different types of DBMS for various applications and environments.

*****

UNIT-II

1. Entity-Entity Type and Entity Set:

i) ENTITY: An entity is a person, place, object, event or concept in the user environment about which an organization maintains the data.

ii) ENTITY TYPE: A collection of entities that share common properties or characteristics.

Some examples are shown below.

Type	Entity
Person	EMPLOYEE, STUDENT
Place	STOREHOUSE, LIBRARY
Object	COMPUTER, BOOK
Event	SALES, ADMISSION
Concept	ACCOUNT, COURSE

Single occurrence of an entity type is *Entity Instance.* In simple terms, it can be treated as a record. Where as, *Entity Type* is a collection of entities that share common properties or characteristics. There are basically three types of entities:

Strong Entity Type: Strong entity is an entity that exists independently of other entity types. Ex: Include STUDENT & EMPLOYEE. Instances of Strong Entity have unique characteristics.

1. **Weak Entity Type:** Weak Entity is an entity type whose existence depends on some other entity type. This has no meaning without the entity on which it depends. The entity type on which weak entity depends is called as "identifying owner". The relationship between weak entity and its owner is called as "identifying relationship". If an EMPLOYEE has a DEPENDENT then DEPENDENT is weak entity, and it exists only when EMPLOYEE exists.
2. **Associative Entity Type:** These are formed from many to many relationships between entity types. It is an entity type that associates the instances of one or more entity types and contains attributes that are peculiar to the relationship between those entity instances.

iii) ENTITY SET: An entity set is a named collection of related data. The data within an entity set are related through their classification. For example, the data in an EMPLOYEE entity set are related by the fact that only employee data are stored in this set. In other words, the EMPLOYEE entity set contains a collection on EMPLOYEE entities. You should not expect to find product entities in an EMPLOYEE entity set.

Guidelines for Naming Entity Types.

- Entity Type should be a singular noun.
- It should be written in capital letters.
- It should be specific to the organization.
- It should be as short as possible.
- It should not describe the event but result.
- It should be same in all ERDs.

2. Attribute and Attribute Types:

Attribute: Attribute is a property or characteristic of an entity type that is of interest to the organization. It can also be called as "field". Attributes start with capital letters followed by small case letters. Every attribute has a value for each record. Following table shows attributes for few entity types:

Entity	Attributes / Fields
STUDENT	Name, Rno, Marks, Address
EMPLOYEE	Ename, Job, Salary, Department

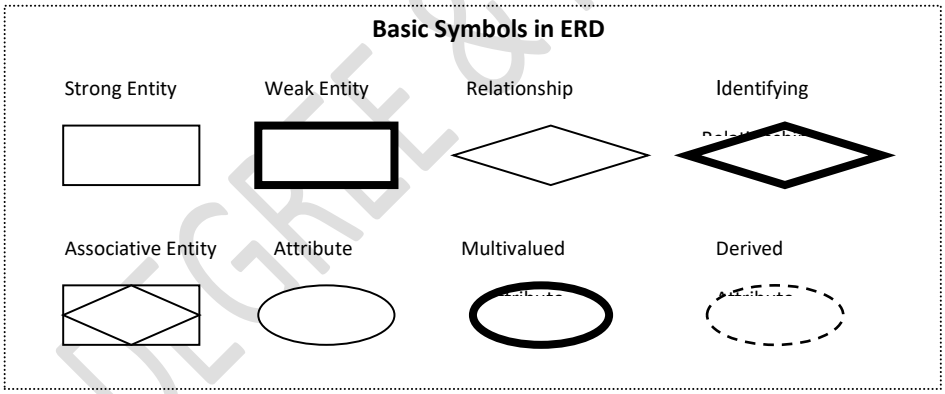
Attribute Types: There are several types of attributes

1. **Composite attribute:** Composite attribute is an attribute that can be broken down into two or more parts. *Example:* "Name" attribute can be broken into "First_name" and "Last_name".
2. **Simple (**or **atomic) attribute:** Simple attribute is an attribute that cannot be broken down into smaller components. *Example:* "Age" attribute can be broken into parts i.e other attributes.
3. **Single valued attribute:** It is an attribute that takes only one value for a given entity instance. *Ex:* "Rno" attribute takes only one value for each record. i.e. every student has only one "Rno"
4. **Multi-valued attribute:** It is an attribute that may take on more than one value for a given entity instance. Ex: "Languages_known" attribute may take multiple values for each record. i.e. a student may know more than one language.
5. **Derived attribute:** It is an attribute whose values can be calculated from related attribute values. *Example:* "Average" attribute is a derived attribute, which can be calculated from the other attributes "Marks1, Marks2 and Marks3".
6. **Stored attribute:** It is an attribute stored in a database to supply values for the derived attributes. *Example:* "Marks1, Marks2 and Marks3" are stored attributes that supply values to calculate value for the "Average" attribute.
7. **Identifier attribute:** It is an attribute used to identify a row uniquely. *Example:* "Rno" is an identifier attribute that identifies a row uniquely. It can be referred to as "primary key attribute"

3. ER-Model:

E-R Model: ER model is a logical representation of the data for an organization or for a business area. The ER model is expressed in terms of entities, attributes and relationships ER model is generally expressed as Entity Relationship Diagram (ERD).

Entity-Relationship Diagram (ERD): ERD is a graphical representation of an entity-relationship model. This represents attributes, relationships and entities using different symbols. The symbols are shown below.



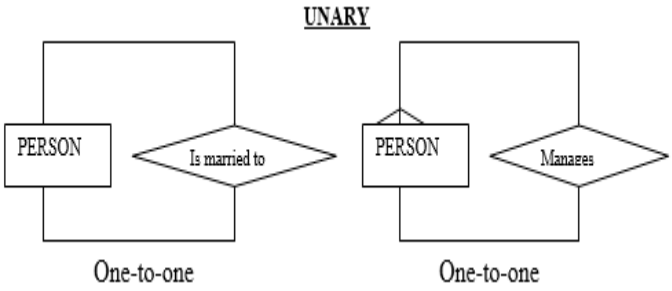
RELATIONSHIP

Relationship is an association among the instances of one or more entities of an organization's database. Degree of relationship is the number of entity types that participate in that relationship. Based on the degree of relationship, they can be categorized as

1) Unary Relationship 2. Binary relationship 3. Ternary relationship.

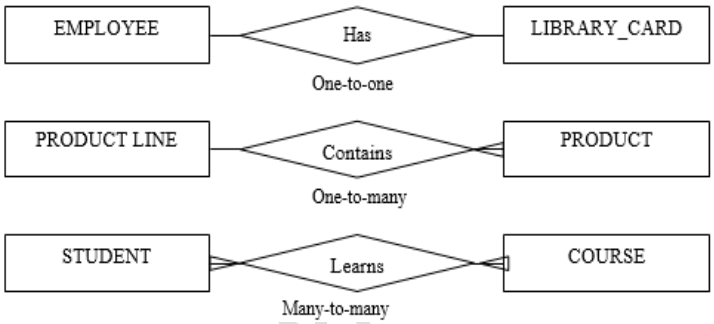
i) Unary Relationship: Unary relationship is a relationship between the instances of a single entity type. Unary relationship is also called as *recursive relationship*. Here the degree of the relationship is '1'. For example, in the following ERDs:

- "Is married to" is *unary one-to-one* relationship between instances of the PERSON type i.e. one person gets married to another person.
- "Manages" is *unary one-to-many* relationship between instances of the EMPLOYEE type i.e. an employee manages another employee.

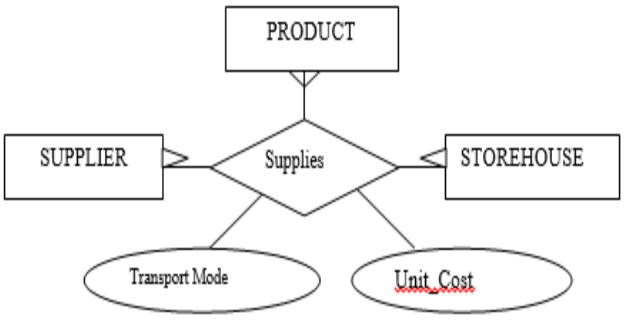


2) Binary Relationship: Binary relationship is a relationship between the instances of two entity types. This is the most common type of relationship encountered in data modeling. Here the degree of the relationship is '2'. For example, in the following ERDs:

- "Has" is a *binary one-to-one* relationship between instances of the EMPLOYEE and LIBRARY_CARD types. It indicates that an employee has one library card.
- "Contains" is a *binary one-to-many* relationship between instances of the PRODUCT_LINE and PRODUCT types i.e. product line has the information of several products
- "Learns" is a *binary many-to-many* relationship between instances of the STUDENT and COURSE types i.e. several students learn several courses.



3) Ternary Relationship: Ternary relationship is a simultaneous relationship among the instances of three entity types. It is recommended to convert ternary relationships to associative entities. Here the degree of the relationship is '3'. For example, in the following ERD:
"Supplies" is a *ternary* relationship among the instances of the SUPPLIER, PRODUCT and STORE_HOUSE types. It indicates that supplier supplies various products to store house and product records the details of supply. And, this leads to ternary relationship.



4. Relational Data Model: E.F.Codd first introduced the relational data model in 1970. The relational data model represents data in the form of tables. The relational model is based on mathematical theory and therefore has a solid theoretical foundation. The relational model consists of the following three models.

1. **Structure:** Data is an organized in the form of tables with rows and columns.
2. **Manipulation***:* Powerful operations (using the SQL language) are used to manipulate data stored in the relations.
3. **Integrity***:* Facilities are included to specify business rules that maintain the integrity of data when they are manipulated.

Relation: Relation is a named two-dimensional table of data. In simple terms it can be called as a table.

a) Properties of Relational Data Model: We have defined relations as two-dimensional table of data. However, not all tables are relations. Relations have several properties that distinguish them from non-relational tables. The properties are shown below.

1. Each relation (or table) in a database has a unique name.
2. An entry at the intersection of each row and column is atomic (or single valued). There can be no multi-valued attributes in a relation.
3. Each row is unique; no two rows in a relation are identical.
4. Each attribute (or column) within a table has a unique name
5. The sequence of columns is not important. The columns of a relation can be interchanged without changing the meaning of the relation.
6. The sequence of rows is not important. The rows of a relation may be interchanged or stored in any sequence.

b) Types of Relational Keys:

We must be able to store and retrieve a row of data in a relation based on the data values stored in that row properly. For this purpose, we define few attributes as some of the following keys.

1. **Primary key:** Primary Key is an attribute that uniquely identifies each row in a relation. *Example:* "Roll_Number" in a STUDENT table.
2. **Composite key:** This is a primary key that consists of more than one attribute. *Example:* "College_Code" and "Roll_Number" in EXAMS table.
3. **Candidate Key:** Candidate key is an attribute, or combination of attributes that uniquely identifies a row in a relation. *Example:* All Primary keys and Composite keys come under this category.
4. **Foreign key:** Foreign key is an attribute in a relation of a database that serves as the primary key of another relation in the same database. *Example:* A matching "Dept_Number" in EMP and DEPT tables. In DEPT table, it is a Primary Key and in EMP table, it is a Foreign Key.
5. **Recursive Foreign key:** This is a foreign key in a relation that references the primary key values of the same relation. *Example:* "Manager_ID" (of EMP table) is a recursive foreign key for "Emp_Num" of the same EMP table.
6. **Enterprise key:** Enterprise Key is a primary key whose value is unique across all relations. *Example:* "Roll_Num" in STUDENT supertype that apples to all other tables in a database.
7. **Secondary or Non-Unique key:** Secondary Key (Non-Unique Key) is a field or a combination of fields for which more than one record may have the same combination of values. *Example:* "Name" in STUDENT table.

Properties of Candidate key: Candidate key is an attribute, or combination of attributes that uniquely identifies a row in a relation. All Primary keys and Composite keys comes under this category. A candidate key must satisfy the following properties. These two properties are essential for a candidate key.

1. **Unique identification:** For every row, the value of the key must uniquely identity that row. This property implies that each non-key attribute is functionally dependent on that key.
2. **Non-redundancy:** No attribute in the key can be deleted without destroying the property of unique identification.

5. Proper naming of Schema Constructs, Relationship types, degrees and cardinalities

Proper naming of schema constructs, relationship types, degrees, and cardinalities is essential for creating clear and understandable database designs. Here's how each should be named:

1. Proper Naming of Schema Constructs:

- **Entities:** Use singular nouns to represent entities. Names should be descriptive and meaningful.
    - Example: **Customer**, **Product**, **Order**.

- **Attributes:** Use descriptive names that represent the property or characteristic being described. Example: **CustomerID**, **ProductName**, **OrderDate**.
- **Relationships:** Use descriptive names that reflect the nature of the relationship between entities. Example: **Customer_Order**, **Product_Supplier**.

2. Relationship Types:

- **Binary Relationship:** A relationship between two entities.
  - Example: **Customer_Order**, **Product_Supplier**.
- **Ternary Relationship:** A relationship involving three entities.
  - Example: **Customer_Order_Product**.
- **Recursive Relationship:** A relationship where an entity is related to itself.
  - Example: **Employee_Manager**.

3. Degrees and Cardinalities:

- **Degree of Relationship:**
  - **Unary Relationship:** A relationship involving only one entity.
    - Example: **Employee_ReportsTo**.
  - **Binary Relationship:** A relationship involving two entities.
    - Example: **Customer_Order**.
  - **Ternary Relationship:** A relationship involving three entities.
    - Example: **Customer_Order_Product**.
- **Cardinality:**
  - **One-to-One (1:1):** Each entity in one set is related to exactly one entity in the other set.
    - Example: Each employee has exactly one manager.
  - **One-to-Many (1:N):** Each entity in one set is related to one or more entities in the other set.
    - Example: One customer can place many orders.
  - **Many-to-One (N:1):** Many entities in one set are related to exactly one entity in the other set.
    - Example: Many orders are placed by one customer.
  - **Many-to-Many (N:M):** Many entities in one set can be related to many entities in the other set.
    - Example: Many students can enroll in many courses.

4. Proper Naming of Cardinality Constraints:

- **Minimum Cardinality:** Describes the minimum number of occurrences of an entity that must participate in a relationship.
  - Example: "1" for mandatory participation, "0" for optional participation.
- **Maximum Cardinality:** Describes the maximum number of occurrences of an entity that can participate in a relationship.
  - Example: "1" for one occurrence, "N" for many occurrences.

Example:

Consider a university database with the following constructs:

- **Entities: Student**, **Course**, **Professor**.
- **Attributes: StudentID**, **CourseID**, **ProfessorID**, **Name**, **Title**.
- **Relationships: Enrollment**, **Teaches**.
- **Relationship Types: Student_Course**, **Course_Professor**.
- **Degrees:** Binary relationships (**Student_Course**, **Course_Professor**).
- **Cardinalities: Student_Course** (1:N), **Course_Professor** (N:1).
- **Minimum/Maximum Cardinalities: Student_Course** (Minimum: 1, Maximum: N), **Course_Professor** (Minimum: 1, Maximum: 1).

With proper naming and clear cardinality constraints, the database schema becomes easier to understand and maintain.

6. Business Rules Enhanced ER Model:

a) Business Rules: Business rules are constraints or guidelines that govern the structure, behavior, and operations of an organization. In the context of a database, business rules define the policies, procedures, and constraints that must be enforced to ensure data integrity, consistency, and correctness. These rules are derived from the business requirements and are implemented in the database schema to ensure that the data remains accurate and valid. Here are some examples of business rules:

1. **Entity Integrity Constraints:**
   - Every entity must have a primary key attribute, and the primary key attribute cannot have a null value.
2. **Referential Integrity Constraints:**
   - In a relationship between two entities, the foreign key attribute(s) in one entity must match the primary key attribute(s) in the related entity, or they must be null.
3. **Attribute Constraints:**
   - Certain attributes may have specific constraints, such as a maximum length, a range of valid values, or a format requirement (e.g., email addresses must follow a specific pattern).
4. **Business Logic Rules:**
   - Rules that enforce specific business processes or logic, such as validation rules for data entry, calculation rules for derived attributes, or rules for triggering certain actions based on data changes.
5. **Security Rules:**
   - Rules that enforce access control policies, specifying which users or roles have permission to access, modify, or delete certain data.

Business rules should be documented and implemented as part of the database design process to ensure that the database accurately reflects the requirements of the organization and supports its business operations effectively.

b) Enhanced Entity-Relationship (EER) Model:

The Enhanced Entity-Relationship (EER) model extends the basic concepts of the Entity-Relationship (ER) model to include additional features for representing complex relationships and constraints. Some key elements of the EER model include:

1. **Subtypes and Supertypes:**
   - Entities can be organized into hierarchies, where a supertype represents a generalization of several subtypes. This allows for modeling of inheritance relationships.
2. **Specialization and Generalization:**
   - Specialization represents the process of defining subtypes based on a supertype, while generalization represents the reverse process of grouping similar entities into a supertype.
3. **Category and Union Types:**
   - Category types represent entities that can belong to multiple entity types simultaneously, while union types represent entities that can belong to one of several entity types.
4. **Aggregation:**
   - Aggregation allows for the modeling of relationships between entities and relationships themselves. It represents a "whole-part" relationship, where the whole entity is composed of several parts.
5. **Attributes Inheritance:**
   - Attributes can be inherited from supertypes to subtypes, allowing subtypes to inherit attributes and associated constraints.

6. **Ternary and Higher-Degree Relationships:**
    - EER models support relationships of higher degrees, such as ternary relationships involving three entities.

The EER model provides a more expressive and flexible way to represent complex data relationships and constraints, making it suitable for modeling advanced database designs that go beyond the capabilities of the basic ER model. It is particularly useful for modeling real-world scenarios with complex business rules and requirements.

*****

UNIT-III

1. Integrity constraints, Relational tables, transforming EER diagrams into relations,

Integrity constraints, relational tables, and transforming Enhanced Entity-Relationship (EER) diagrams into relations are all fundamental concepts in database design. Let's break down each of them:

a) Integrity Constraints:

Integrity constraints ensure the accuracy and consistency of data within a database. They are rules that restrict the data values allowed in a database and maintain the integrity of the data. Common types of integrity constraints include:

- **Entity Integrity Constraint**: Ensures that a primary key attribute in a table is not null.
- **Referential Integrity Constraint**: Ensures that relationships between tables remain consistent. For example, a foreign key in one table must match a primary key value in another table.
- **Domain Integrity Constraint**: Defines the permissible values for a column, such as data type, range, or format.
- **Check Constraint**: Enforces a condition on the values allowed in a column.

b) Relational Tables:

Relational tables are the building blocks of a relational database. Each table represents an entity or a relationship between entities. A table consists of rows and columns, where each row represents a record and each column represents an attribute.

For example, consider a simple table for storing information about employees:

EmployeeID	Name	Department	Salary
1	John	HR	50000
2	Alice	Sales	60000
3	Bob	IT	55000

c) Transforming EER Diagrams into Relations:

Enhanced Entity-Relationship (EER) diagrams extend the capabilities of traditional ER diagrams by including additional concepts like subclasses, superclasses, specialization, and generalization.

To transform an EER diagram into relations (tables), you typically follow these steps:

- **Identify Entities**: Each entity in the EER diagram becomes a table. Include all attributes associated with each entity as columns in the corresponding table.
- **Resolve Subclasses**: If the EER diagram contains subclasses, each subclass typically becomes its own table. Attributes specific to the subclass are included in the subclass table, along with a foreign key referencing the superclass table.
- **Resolve Relationships**: Relationships between entities become foreign keys in the corresponding tables. For one-to-many relationships, the foreign key is placed on the "many" side.

- **Resolve Specialization and Generalization**: Specialization and generalization hierarchies are represented using separate tables or columns, depending on the approach taken during the design.

By following these steps, you can effectively translate the structure and relationships depicted in an EER diagram into relational tables.

2. Functional Dependencies and Normalization for Relational Database:

Functional dependencies (FDs) and normalization are crucial concepts in database design for ensuring data integrity and reducing redundancy. Here's a brief overview:

Functional Dependencies (FDs):

Functional dependencies describe the relationship between attributes in a relation (table). An FD states that the value of one attribute (or a set of attributes) uniquely determines the value of another attribute within the same table.

For example, let's say we have a table **Student** with attributes **StudentID**, **Name**, and **Email**. We might have the functional dependency:

StudentID → Name, Email

This means that given a **StudentID**, we can uniquely determine the **Name** and **Email**.

Normalization: Normalization is the process of organizing the attributes and tables of a relational database to minimize redundancy and dependency. It ensures that the database schema is free from anomalies and adheres to a set of normal forms.

Normal Forms:

1. First Normal Form (1NF):

Eliminate repeating groups within rows.

Ensure each attribute contains atomic (indivisible) values.

2. Second Normal Form (2NF):

Must be in 1NF.

Eliminate partial dependencies.

Move attributes dependent on part of a candidate key to a separate table.

3. Third Normal Form (3NF):

Must be in 2NF.

Eliminate transitive dependencies.

Move attributes not dependent on the primary key to a separate table.

4. Boyce-Codd Normal Form (BCNF):

A stricter form of 3NF where every determinant is a candidate key.

5. Fourth Normal Form (4NF):

Eliminate multivalued dependencies.

Split tables with multivalued attributes into separate tables.

6. Fifth Normal Form (5NF):

Further eliminate redundancy by decomposing tables based on join dependencies.

Process of Normalization:

- **Identify Functional Dependencies**: Determine the functional dependencies within each relation.
- **Decompose Relations**: Decompose the relations to achieve a higher normal form, while preserving the dependencies.
- **Verify Normal Form**: Ensure that each relation meets the requirements of the target normal form.
- **Repeat if Necessary**: If a higher normal form is desired or required, repeat the process until the database schema is appropriately normalized.

Example:
Consider a table **StudentCourse** with attributes **StudentID**, **CourseID**, **CourseName**, and **Instructor**. Suppose we have the following functional dependencies:
StudentID, CourseID → CourseName, Instructor
CourseID → CourseName
This table is not in 3NF because **Instructor** depends on **CourseID**, which is not a candidate key. To normalize it, we can decompose it into two tables:
StudentCourse: **StudentID, CourseID**
Course: **CourseID, CourseName, Instructor**
This way, we eliminate the transitive dependency and achieve 3NF.

3. Normalization:
Normalization is a process of decomposing relations with anomalies to produce smaller well-structured relations. It is a formal process for deciding which attributes should be grouped together in a relation.

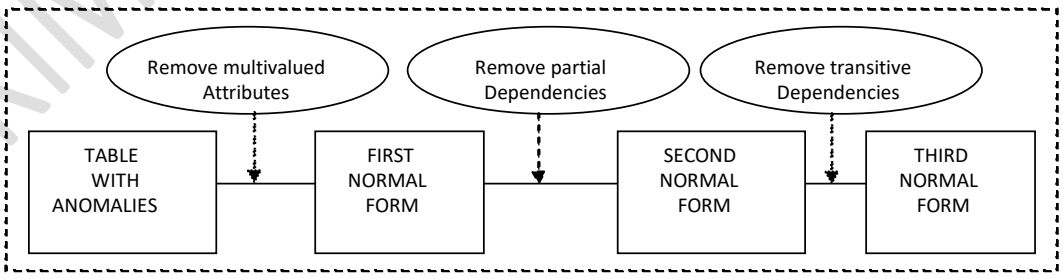
Normal Forms: It is a state of relation that results from applying simple rules regarding functional dependencies to that relation. *Functional Dependency* is a constraint between two attributes or two sets of attributes.

1. **First Normal Form:** A relation is in first normal form (1NF) if it *contains no multivalued attributes*. According to the properties of relational model, the value at the intersection of each row and column must be a single value.

2. **Second Normal Form:** A relation is in second normal form (2NF) if *it is in first normal form and every non-key attribute is fully functionally dependent on the primary key*. Thus no non-key attribute is functionally dependent on part of the primary key.

A relation that is in first normal form will be in second normal form if any one of the following conditions applies.
- ✓ The primary key consists of only one attribute.
- ✓ No non-key attributes exist in the relation (thus all of the attributes in the relation are components of the primary key).
- ✓ Every non-key attribute is functionally dependent on the full set of primary key attributes.

3. **Third Normal Form:** A relation is in third normal form (3NF) if *it is in second normal form and has no transitive dependencies present.* A Transitive Dependency is a functional dependency between two or more non-key attributes.



Example: How do you use normalization to decompose a relation with anomalies into well-structured relations?
Consider the following table with anomalies. The table has multi-valued attribute "Skills".
Let us decompose the table into structured table's step by step using normalization. Table: **STUDENT**

Rno	Sname	Group	Fee	Skills	Faculty
101	Ravi	BSc	7000	C Oracle C++	Prasad Karthik Vinod
102	Ramu	BCom	6000	C VB	Prasad Sridhar
103	Kiran	BSc	7000	Java ASP	Kishor Sathyam
104	Srinivas	BTech	12000	Java	Kishor

First Normal Form: The following relation is in first normal form (1NF) because it *contains no multivalued attributes*. But, it is not in 2NF because it has partial dependency i.e. Sname, Group and Fee are functionally dependent on part (but not all) of the primary key (Rno).

Table: **STUDENT**

Rno	Sname	Group	Fee	Skills	Faculty
101	Ravi	BSc	7000	C	Prasad
101	Ravi	BSc	7000	Oracle	Karthik
101	Ravi	BSc	7000	C++	Vinod
102	Ramu	BCom	6000	C	Prasad
102	Ramu	BCom	6000	VB	Sridhar
103	Kiran	BSc	7000	Java	Kishor
103	Kiran	BSc	7000	ASP	Sathya m
104	Srinivas	BTech	12000	Java	Kishor

Second Normal Form: The following relation is in second normal form (2NF) because *it is in first normal form and every non-key attribute is fully functionally dependent on the primary key*. But, it is not in 3NF because it has transitive dependency i.e. dependency between "Fee" and "Group" attributes.

Table: **STUDENT**

Rno	Sname	Group	Fee
101	Ravi	BSc	7000
102	Ramu	BCom	6000
103	Kiran	BSc	7000
104	Srinivas	BTech	12000

Table: **SKILL**

Rno	Skills	Faculty
101	C	Prasad
101	Oracle	Karthik
101	C++	Vinod
102	C	Prasad
102	VB	Sridhar
103	Java	Kishor
103	ASP	Sathyam
104	Java	Kishor

Third Normal Form: The following relation is in third normal form (3NF) because *it is in second normal form and has no transitive dependencies.*

Table: **GROUPS**

Group	Fee
BSc	7000
BCom	6000
BTech	12000

Table: **STUDENT**

Rno	Sname	Group
101	Ravi	BSc
102	Ramu	BCom
103	Kiran	BSc
104	Srinivas	BTech

Table: **SKILL**

Rno	Skills	Faculty
101	C	Prasad
101	Oracle	Karthik
101	C++	Vinod
102	C	Prasad
102	VB	Sridhar
103	Java	Kishor
103	ASP	Sathyam
104	Java	Kishor

4. Explain Boyce Codd Normal Form with example

Boyce-Codd Normal Form (BCNF) is a stricter form of normalization than Third Normal Form (3NF). It deals with the problem of functional dependencies and ensures that a relation is free from all redundancy and anomalies related to functional dependencies.

Boyce-Codd Normal Form (BCNF):

A relation is in BCNF if and only if for every non-trivial functional dependency $X \rightarrow Y$X→Y, where XX is a superkey, YY is a candidate key.

In simpler terms, BCNF states that every determinant (the attribute or set of attributes on the left side of the functional dependency) must be a candidate key. If any non-trivial dependency exists where the determinant is not a candidate key, the relation violates BCNF.

Example:

Let's consider a relation **Employee_Project** with the following attributes:

EmployeeID
ProjectID
EmployeeName
ProjectName
HoursWorked

And the following functional dependencies:

EmployeeID → EmployeeName
ProjectID → ProjectName
EmployeeID, ProjectID → HoursWorked

Here, (EmployeeID, ProjectID) is a candidate key since it uniquely identifies each combination of employee and project. So, the relation is not in BCNF because the third functional dependency violates the rule.

To bring the relation into BCNF, we decompose it:

Employee:

EmployeeID→EmployeeNameEmployeeID→EmployeeName

Project:

ProjectID→ProjectNameProjectID→ProjectName

Employee_Project_Hours:

EmployeeID,ProjectID→HoursWorkedEmployeeID,ProjectID→HoursWorked

Now, each relation satisfies BCNF because all the determinants (the attributes on the left side of the functional dependency) are candidate keys in their respective relations.

In summary, BCNF ensures that all functional dependencies within a relation are fully dependent on the candidate keys, eliminating any possibility of redundancy and anomalies related to functional dependencies.

5. DE normalization Relational Algebra.

Denormalization, relational algebra, and normalization are essential concepts in database management. Let's dive into each:

a) Denormalization: Denormalization is the process of intentionally adding redundancy to a database for performance reasons. It involves combining tables and relaxing normalization rules to improve query performance by reducing the need for joins and aggregations.

Benefits of Denormalization:

Improved query performance: Since data is stored redundantly, complex queries can be simplified, and the need for joins may be reduced.
Reduced computational overhead: Aggregations and calculations can be precomputed and stored.
Enhanced read performance: Denormalized structures can be optimized for specific read patterns.

Examples of Denormalization:

- **Flattening Tables**: Combining multiple related tables into a single table.
- **Introducing Redundant Data**: Storing aggregated or calculated values alongside raw data.
- **Adding Duplicate Columns**: Storing a copy of a frequently accessed column in multiple tables to avoid joins.

b)Relational Algebra:

Relational algebra is a theoretical language used to manipulate relations (tables) in relational databases. It provides a set of operations to perform queries on the database.

Basic Operations in Relational Algebra:

- **Selection ($\sigma\sigma$)**: Selects rows from a relation that satisfy a given predicate.
- **Projection ($\pi\pi$)**: Selects certain columns (attributes) from a relation.
- **Union (∪∪)**: Combines tuples from two relations, removing duplicates.
- **Intersection (∩∩)**: Returns tuples that appear in both relations.
- **Difference (−−)**: Returns tuples that appear in one relation but not in the other.
- **Cartesian Product (××)**: Combines every tuple in one relation with every tuple in another relation.
- **Join (⋈⋈)**: Combines tuples from two relations based on a common attribute.
- **Division (÷÷)**: Finds tuples that, when combined with every tuple from one relation, cover every tuple in another relation.

Denormalization and Relational Algebra:

Denormalization affects the relational algebra operations in the following ways:

- **Projection and Selection**: Denormalization may impact which attributes are selected or which predicates are used in queries, as redundant data may provide more options.
- **Join**: Denormalization can reduce the need for joins by storing related data together, making certain joins unnecessary for some queries.
- **Union, Intersection, and Difference**: These operations remain largely unchanged by denormalization unless the denormalization introduces inconsistencies.

Conclusion: Denormalization can improve query performance in certain scenarios, but it comes with the trade-off of increased storage requirements and potential data inconsistency. Relational algebra provides the theoretical foundation for querying and manipulating relational databases, including those that have been denormalized.

6. File Organization techniques:

File organization techniques are methods used to arrange data in files to optimize storage, access, and retrieval. Several file organization techniques are commonly used in database systems:

1. Sequential File Organization:

- Data is stored in sequential order based on a primary key.
- Suitable for applications where data is accessed sequentially or requires frequent range queries.
- Examples: Text files, log files.

2. Indexed Sequential Access Method (ISAM):

- Combines sequential and indexed access methods.
- Data is stored sequentially, but an index structure is maintained to facilitate direct access to records.
- Suitable for applications where both sequential and random access are required.
- Examples: Indexed files in filesystems, some database systems.

3. Hashed File Organization:

- Records are stored at locations determined by applying a hash function to the record's key.
- Provides fast access to records based on their key.
- Suitable for applications with a high volume of random access.
- Examples: Hash tables, some filesystems.

4. B-Tree and B+Tree Organization:

- Balanced tree structures that maintain sorted data for efficient retrieval.
- B-Tree: Each node contains both keys and pointers to child nodes.
- B+Tree: Similar to B-Tree but with additional pointers at leaf nodes for efficient range queries.
- Suitable for large datasets and frequently updated data.
- Widely used in database indexes and filesystems.

5. Heap File Organization:

- Records are stored in no particular order.
- Suitable for applications where records are inserted and deleted frequently.
- Overhead for searching and retrieving records can be high.
- Examples: Some database systems for temporary storage.

6. Clustered File Organization:

- Groups related records together physically on disk.
- Typically used when records are frequently accessed together.
- Reduces disk I/O by minimizing seeks for related data.
- Examples: Filesystems supporting clustering, some database systems for index-organized tables.

7. Inverted File Organization:

- Maintains an index of attribute values to records where they appear.
- Suitable for applications where searches are primarily based on attributes rather than record identifiers.
- Examples: Full-text search engines, some database systems for text retrieval.

8. Multi-Level Indexing:

Uses multiple levels of indexing to improve access speed.
Suitable for large datasets where a single-level index would be impractical.
Examples: Multi-level B-Trees, some filesystems.

Conclusion: Each file organization technique has its advantages and disadvantages, and the choice depends on factors such as the nature of the data, access patterns, and performance requirements. Effective file organization is crucial for efficient data management in both file-based and database systems.

*****

UNIT-IV

ROLE OF SQL IN DATABASE DEVELOPMENT

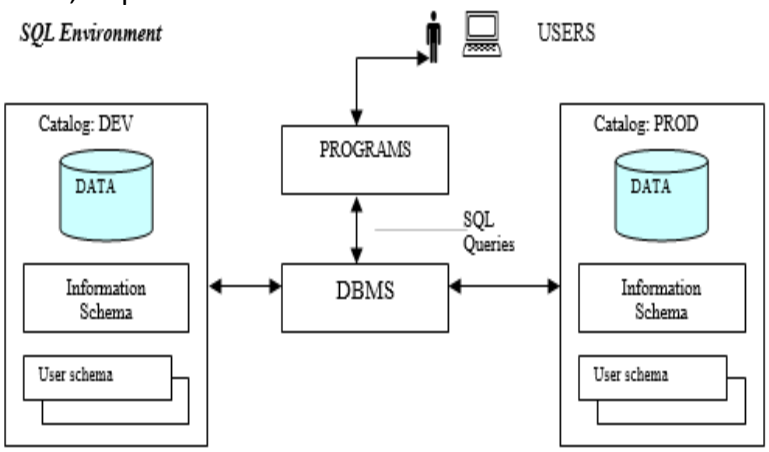
1. To specify the syntax and semantics of SQL data definition and manipulation languages.
2. To define the data structures and basic operations for designing, accessing, maintaining, controlling, and protecting an SQL database.
3. To establish portability of database definition and application-modules between compatible DBMS.
4. To specify both minimal (Level 1) and complete (Level 2) standards.
5. To provide an initial standard that will be enhanced later to include specifications for handling referential integrity, transaction management, and user-defined functions.

SQL ENVIRONMENT

The SQL environment includes the following:

- An instance of SQL DBMS
- Programs that use the DBMS to access database.
- A catalog which consists database
- Users who work with the database

Most companies keep at least two versions of any database they are using.
One is Product version and the other is Development version.
Production version must be tightly controlled and monitored
Development version is not tightly controlled and monitored
Each database will have a named schema associated with a catalog.
Users can access the information from a database using programs, which include SQL queries.
Users can browse the catalog contents by using queries.
Catalog is a set of schemas that, when put together, constitute a description of a database.
Schema is a structure, which contains descriptions of objects created by a user, such as base tables, views, and constraints, as part of a database.



SQL: SQL stands for Structured Query Language: SQL used to write queries that store, alter and retrieve data and metadata of a relational database. This is a Non-Procedural language. This is a unified language i.e. common for many databases. This is a Fourth Generation Language (4GL) in which programmer concentrates on what rather than how.
SQL commands can be classified into three types: 1.Data Definition Language (DDL)
2. Data Manipulation Language (DML) 3.Data Control Language (DCL)

1. DDL: Data Definition Language commands are used to define a database, including creating, altering, and dropping tables and establishing constraints. DDL deals with *metadata.*

Create	To create Oracle database objects such as tables, views, indexes, clusters, etc.
Alter	To alter the structure of a database object
Drop	To delete a database object
Truncate	To delete rows with auto commit
Rename	To rename the database object

2. DML: Data Manipulation Language commands are used to maintain and access a database, including updating, inserting, modifying, and querying data. It deals with *data.*

Select	To retrieve data from a table. Allows filtering.
Insert	To inert data into a table
Update	To update the values of attributes
Delete	To delete rows from a table

3. DCL: Data Control Language commands are used to control a database, including administering privileges and the committing (saving) of data. It deals with *accessibility and transaction changes.* The commands are:

Accessibility Control	**Grant**	To give privileges on database objects to other users
	Revoke	To cancel privileges on database objects to other users
Transaction Control	**Commit**	To make the changes permanent
	Rollback	To reverse the changes
	SavePoint	To set a margin for commit or rollback

2. Data types SQL:

In SQL, data types define the type of data that can be stored in a column of a table. Different database management systems (DBMS) might have slightly different data types, but some common ones include:

1. **INTEGER**: Used to store whole numbers. Sizes can vary depending on the database, like **INT**, **INTEGER**, **SMALLINT**, **BIGINT**.
   Example: **INT**
2. **DECIMAL or NUMERIC**: Used to store fixed-point numbers. You specify the total number of digits and the number of digits after the decimal point.
   Example: **DECIMAL(10, 2)** - This allows 10 digits in total with 2 digits after the decimal point.
3. **FLOAT or REAL**: Used to store floating-point numbers. They're approximate, not exact.
   Example: **FLOAT**
4. **CHAR or VARCHAR**: Used to store character strings. **CHAR** is for fixed-length strings, while **VARCHAR** is for variable-length strings.
   Example: **VARCHAR(255)** - This allows up to 255 characters.
5. **DATE**: Used to store dates without time.
   Example: **DATE**
6. **TIME**: Used to store time without date information.
   Example: **TIME**
7. **DATETIME or TIMESTAMP**: Used to store date and time information.
   Example: **DATETIME**
8. **BOOLEAN**: Used to store boolean values (**TRUE** or **FALSE**).
   Example: **BOOLEAN**
9. **BLOB**: Binary Large Object, used to store binary data like images or files.
   Example: **BLOB**
10. **JSON**: Used to store JSON formatted data.
    Example: **JSON**

These are just some common data types; there are more specific types and variations depending on the DBMS you're using. For example, SQL Server has **BIT** for storing boolean values, and PostgreSQL has **TEXT** for large text values, among others.

3. OPERATORS AND CLAUSES USED IN SQL:

Comparison Operators	=Equal to, != or <> Not Equal to, > Greater than , < Less than, >= Greater than or equal to, <= Less than or equal to
Logical Operators	AND, OR, NOT
Arithmetic Operators	+ - * /
Set Operators	UNION, UNION ALL, INTERSECT,MINUS
Special Operators	IN, BETWEEN AND, LIKE, IS NULL, | | Concatenation operator

4.What are types of CLAUSES:

SELECT	Lists the columns (including expressions) from tables or views.
FROM	Identifies the tables or views from which columns will be chosen
WHERE	Includes the conditions for row selection within a single table or multiple tables or views
DISTINCT	Avoids duplicate rows
GROUP BY	Groups rows according to the category specified
ORDER BY	Sorts the final results rows in ascending or descending order
HAVING	Can only be used with a GROUP BY and acts as a secondary WHERE clause, returns only those groups, which meet a specified condition.
UNION	Combines unique records with matching attributes of multiple tables
UNION ALL	Combines all records with matching attributes of multiple tables including duplicates
INTERSECT	Retrieves common records with matching attributes of multiple tables
MINUS	Retrieves other than common records with matching attributes from the first table

5. Generating SQL Database Definitions:

We can use CREATE command to generate database definitions. The three main SQL Data Definition languages CREATE commands are:

1. CREATE DATABASE: To create a database use creates database command, in this again many tables or views or queries we can create and manipulate the things.

Syntax: CREATE DATABASE <Database Name>; Example: CREATE DATABASE EIT;

2. CREATE TABLE: Defines a new table and its columns. The table may be a base table or a derived table. Tables are dependent on a schema. Executing a query that uses one or more tables or views creates derived tables.

Example: CREATE TABLE *DEPARTMENT* **(** DEPT_NUM INT PRIMARY KEY, DEPT_NAME VARCHAR2(20) **);**

3. CREATE VIEW: Defines a logical table from one or more tables or views. Views may not be indexed. There are limitations on updating data through a view. If views can be updated, those changes can be transferred to the underlying base tables.

Example: CREATE VIEW *MANAGERS* AS SELECT * FROM EMP WHERE JOB='MANAGER'

1) Table: Procedure for Creating Tables:

1. Identify the appropriate data types. Including length and precision for each attribute.
2. Identify those columns that should accept null values.
3. Identify those columns that need to be unique. When a column control of UNIQUE is established for a column, then the data in that column should not take duplicate values.
4. Identify all primary key-foreign key mates.
5. Determine values to be inserted in any columns for which a default value is required.
6. Identify columns for which special constraints are required.
7. Create the table and any desired indexes using the CREATE TABLE and CREATE INDEX statements.

Example: CREATE TABLE *STUDENT* **(RNO** INT PRIMARY KEY, SNAME VARCHAR2 (20) NOT NULL, MARKS NUMBER (3));

2) Views: View is a virtual table based on a table(s) or another view(s). It can be treated as a stored query. The table, on which a view is based, is called a *base table.*
Example: CREATE VIEW ***MANAGERS*** AS SELECT * FROM EMP WHERE JOB='MANAGER'

3) Creating Indexes: We choose to index primary and/or secondary keys to increase the speed of row selection, table joining, and row ordering. If you want to search for a record in a large database, it is time consuming to scan every row of a table. Indexes can speed up the process of searching records.

Indexes are created in most RDBMS to provide rapid random and sequential access to base-table data. In SQL standards, no standard syntax for creating indexes is included.

The users do not directly refer to indexes when writing SQL commands. The DBMS recognizes indexes and improve query performance. Indexes can usually be created for both primary and secondary keys.

Example: To create an **index** on **CNAME** in the **CUSTOMER** table:
CREATE INDEX *CUST_INDEX* ON CUSTOMER (CNAME);
To create a **Composite Index**: CREATE INDEX *DEPT_NAME* ON DEPT (DEPTNO, DNAME);
To **remove the index** on the customer name in the CUSTOMER table: DROP INDEX *CUST_INDEX;*

6. ESTABLISH REFERENTIAL INTEGRITY USING SQL:

Integrity constraints such as Entity Integrity and Referential Integrity can be easily created using SQL commands. Referential integrity is an integrity constraint specifying that the value of an attribute in one relation depend on the value of a primary key in the same or another relation**.**

Example: In the following tables:

- In EMP table, the values of "Dno" must match the values of "Dno" attribute of DEPART table.
- In order to achieve that, we need to write "Dno" attribute of EMP-table as *Foreign Key.* This establishes referential integrity between EMP and DEPART tables. We use "references" key word to make an attribute as foreign key.

Table: **EMP**

Eno (PK)	Ename	Job	Salary	Dno (Foreign Key)
123	Ravi	Manager	12000	10
124	Rajesh	Clerk	2000	20
125	Amar	Clerk	2400	10
126	Kiran	Accountant	4000	30

Table:**DEPART**

Dno (Primary Key)	Dname
10	Finance
20	Accounts
30	Marketing

a) CREATE TABLE *DEPART* **(**DNO INT PRIIMARY KEY, DNAME VARCHAR (20));
b) CREATE TABLE *EMP (*ENO INT PRIMARY KEY, ENAME VARCHAR2 (20),
JOB VARCHAR (20), SALARY INT, DNO INT **REFERENCES** DEPART (DNO));

7. BUILT-IN FUNCTIONS:

SQL provides several built-in functions that can be operated on characters, numbers and dates. The types include: Aggregate (*Group) functions, Number functions, String functions, Date functions, Conversion functions and Miscellaneous functions*

1. AGGREGATE (GROUP) FUNCTIONS: These functions are used to operate on a group of values. They aggregate the group to perform calculations such as finding out sum, max value, min value and etc. These functions can be used with GROUP BY clause to categorize results. Example, find out the sum of the salaries in each department.

Function	Purpose	Example	Output
sum	Finds sum of the values of a column	SELECT SUM(SAL) FROM EMP	Sum of the salaries of all employees
avg	Finds avearge of the values of a column	SELECT AVG(SAL) FROM EMP	Average of the salaries of all employees
max	Finds maximum of the values of a column	SELECT MAX(SAL) FROM EMP WHERE JOB='MANAGER'	Maximum of the salaries of all managers
min	Finds minimum of the values of a column	SELECT MIN(SAL) FROM EMP	Minimum of the salaries of all employees
count	Finds number of records	SELECT COUNT(*) FROM EMP	Number of records in employee table
stdev	Finds standard deviation of the values of a column	SELECT STDEV(SAL) FROM EMP	Standard deviation of the salaries of all employees
var	Finds variance of the values of a column	SELECT VAR(SAL) FROM EMP	Variance of the salaries of all employees

NUMERIC FUNCTIONS

Function	Purpose	Example	Output
abs	Returns the absolute value	Print abs(-20)	20
floor	Greatest integer less than or equal to a value	Print floor(2.6)	2
sqrt	Returns square root of a number	Print sqrt(16)	4
power	Returns power specified by a number	Print power(2,3)	8
%(mod)	Returns the remainder	Print (10%3)	1
sign	Returns –1 if the no. is <0, 1 if the no. is >0 , 0 otherwise	Print sign(-3), sign(5), sign(0)	-1 1 0
round	Rounds: the int part of a no. with -ve precession, decimal part otherwise	Print(99.267,2)	99.27
sin	Returns the sin value	Print SIN((30*3.14)/180)	0.499
cos	Returns the cos value	Print COS((30*3.14)/180	0.866
tan	Returns the tan value	Print TAN((30*3.14)/180)	0.576
log	Returns the logarithmic value as per the base value	Print LOG(10)	2.30259

STRING FUNCTIONS

Function	Purpose	Example	Output
+(concate)	Appends two strings	Print 'Wel'+'come'	Welcome
upper	Converts the words / values of a column into uppercase	Print upper('eit')	EIT
lower	Converts the words / values of a column into lowercase	Print lower('EIT')	eit
ltrim	Removes all occurrences of the	Print ltrim(' eit')	eit

	blank spaces from left side		
Rtrim	Removes all occurrences of the blank spaces from right side	Print ltrim('eit ')	eit
left	Returns left no. of characters	Print left('ERITREA',3)	ERI
right	Returns right no. of characters	Print right('ERITREA',4)	TREA
substring	Subtracts n chars from a given position.	Print substr('ERITRE',3,3)	ITR
reverse	Returns the given string into reverse string	Print reverse('RAMA')	AMAR
len	Returns number of chars of a string	Print len('RAMA')	4
Replace	Replaces the search string, if exists, with the given string	Print replace('abcaabcc','ab','xy'	xycaxycc
char	Returns character corresponding to a no.	Print char(65)	A
ascii	Returns ASCII value corresponding to a char	print ascii('A')	65

DATE FUNCTIONS

Function	Purpose	Example	Output
Getdate()	Returns system's date	Print getdate()	May 11 2011 12:34
dw	Returns system day of the week	Print datename(dw,getdate())	wednesday
dd	Returns day number	Print datename(dd,'12/12/2011'	12
month	Returns month characters	Print datename(month,getdate())	May
year	Returns year number	Print datename(year,getgate())	2011
to_days	Returns no. of days (differences)		

8. LAB SESSION:

EMP: TABLE

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	12/17/80	800		20
7499	ALLEN	SALESMAN	7698	2/20/81	1600	300	30
7521	WARD	SALESMAN	7698	2/22/81	1250	500	10
7566	JONES	MANAGER	7839	4/2/81	2975		20
7654	MARTIN	SALESMAN	7698	9/28/81	1250	1400	30

DEPT: Table

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

1. DATA DEFINITION LANGUAGE (DDL):

DDL commands are used to define a database, including creating, altering, and dropping tables and establishing constraints. DDL deals with metadata. The commands are: CREATE, ALTER, DROP, TRUNCATE and RENAME

1. CREATE **Command:** To create Oracle database objects such as tables, views, indexes, clusters, and synonyms etc.

Syntax: CREATE TABLE table_name (column_name datatype(size)
[CONSTRAINT constraint_name] [constraint_type] . . .);

***Example**: To create a table student with rno, name, marks attributes.*

Create table student (rno int , name varchar(10), marks int)

CREATING TABLES WITH CONSTRAINTS IN SQL:

NOT NULL	Enforces the values of a column to be not null i.e. it has to have value
UNIQUE	Enforces the values of a column to be unique i.e. the column cannot have duplicate values, but it allows null value
PRIMARY KEY	Enforces the values of a column to be unique and to be not null i.e. the column cannot have duplicate values and null values
DEFAULT	Provides a default value for a column when value is not supplied for that.
CHECK	Used for checking the values of column against the conditions specified
FOREIGN KEY (REFERENCES)	Allows only the values matching to the values of another primary key column of another table or the same table.

***Example**: To create a table with primary key, check and not null constraints. Roll number should not allow duplicate values, Name cannot be null and marks must be <=100 in student table.*
CRE ATE TABLE STUDENT (RNO INT PRIMARY KEY, NAME VARCHAR (20) NOT NULL, MARKS INT CHECK (MARKS<= 100));

***Example**: To create a table with foreign key (references) constraint. Values of dno attribute of depart table should match with values of dno attribute of employee table.*
CREATE TABLE *DEPART* (DNO INT PRIMARY KEY, DNAME VARCHAR (20));

CREATE TABLE *EMP* (ENO INT PRIMARY KEY, ENAME VARCHAR (20), JOB VARCHAR (20), SALARY INT DEPTNO INT REFERENCES DEPART (DNO));

***Example**: To create a table with default and unique constraint. Values of dno attribute should be unique and default value for the dname should be "ACCOUNTS" in department table.*
CREATE TABLE *DEPART*
(DNO INT UNIQUE, DNAME VARCHAR (20) DEFAULT 'ACCOUNTS');
Suppose, if you try to insert a record without any value for dname,

INSERT INTO DEPART (DEPTNO) VALUES (10);
The record "*10, ACCOUNTS* " is entered into the table i.e. it takes "ACCOUNTS" as default value for dname even if we do not enter.

***Example**: To create a table with composite unique constraint. Combination of ino and iname attributes should not be duplicated in items table.*
CREATE TABLE *ITEMS* (INO INT, INAME VARCHAR (20) CONSTRAINT C1 UNIQUE (INO, INAME));

***Example**: To create a table Department with dno unique constraint i.e. there should be a dno for unique constraint.*
CREATE TABLE *DEPT* (DNO INT CONSTRAINT C1 UNIQUE, DNAME VARCHAR (20));
***Example**: To create an index on a column. To create alphabetical index on cname attribute of customer table.* CREATE INDEX *CUST_INDEX* ON CUSTOMER (CNAME);
***Example**: To create an index on multiple columns deptno and dname*
CREATE INDEX *DEPT_NAME* ON DEPT (DEPTNO, DNAME);
***Example**: To create a view of managers of emp table*
CREATE VIEW *MANAGERS* AS SELECT * FROM EMP WHERE JOB='MANAGER';

***Example**: To create a view of multiple tables. View should have empno, ename, job attributes from emp table and dname from dept table.*
CREATE VIEW *EMPVIEW* AS SELECT EMPNO, ENAME, JOB, DNAME FROM EMP, DEPT WHERE EMP.DEPTNO=DEPT.DEPTNO

2) ALTER **Command:** To alter the structure of a database object

Syntax: ALTER TABLE table_name
[ADD column_name datatype(size)]
[ALTER COLUMN column_name data type (size)]
[DROP COLUMN column_name];

***Example**: To add a new column to an existing table. The following query adds a new column to student table.* ALTER TABLE STUDENT ADD AVERAGE FLOAT (5, 2));

***Example**: To modify the length of an existing column of a table. The following query modifies the length of the existing attribute rno to '3' in student table.*

ALTER TABLE STUDENT ALTER COLUMN RNO INT (3);

***Example**: To add constraint an existing column of a table. The following query adds constraint primary key to the existing attribute rno of student table.*

ALTER TABLE STUDENT ADD CONSTRAINT C1 PRIMARY KEY (RNO);

***Example**: To remove an existing column of a table. The following query removes the existing attribute "average" of student table.* ALTER TABLE STUDENT DROP COLUMN AVERAGE;

***Example**: To remove constraint on existing column of a table. The following query removes the constraint C1 of student table.* ALTER TABLE STUDENT DROP CONSTRAINT C1;

***Example**: To enable/disable constraint on existing column of a table. The following query enables/disables the constraint C1 of student table.*

ALTER TABLE STUDENT ENABLE CONSTRAINT C1;
ALTER TABLE STUDENT DISABLE CONSTRAINT C1;

3. DROP **Command:** To delete a database object

Syntax: DROP object_type object_name;

***Example**: To delete an existing table "items" DROP* TABLE ITEMS;

***Example**: To delete an index.* DROP INDEX DNAME_IND;

***Example**: To delete a view.* DROP VIEW MANAGER;

4. TRUNCATE **Command:** To delete rows with auto commit

Syntax: TRUNCATE TABLE table_name;

***Example**: To delete complete information from a table.* TRUNCATE TABLE STUDENT;

5. RENAME **Command:** To rename a database object (Table)

Syntax: RENAME old_table_name TO new_table_name;

***Example**: To rename a table (in oracle package).* RENAME STUDENT TO STUD1;

Note: Micro Soft SQL Server: Select old table name
Press Right click mouse button
Select "Rename" option then type new table name "Table Name"

2. DATA MANIPULATION LANGUAGE-DML:

DML commands are used to maintain and access a database, including updating, inserting, modifying, and querying data. It deals with *data.* The commands are: SELECT INSERT, UPDATE and DELETE.

1) SELECT **Command:** To retrieve data from a table, and it allows filtering.

Syntax: SELECT [DISTINCT] column_list FROM table_list
[WHERE condition]
[GROUP_BY grouping columns]
[HAVING group condition]
[ORDER_BY orderby_columns];

***Example**: To select complete information from the table student.* SELECT * FROM STUDENT;

***Example:** To select top 10 records information from the table student.*

*SELECT TOP 10 * FROM STUDENT;*
***Example**: To select records of the employees whose salary >= 2000, from emp table (limited rows)*
SELECT * FROM EMP WHERE SAL>=2000;
***Example**: To select name, job, salary from the emp table (limited columns).*
SELECT ENAME, JOB, SAL FROM EMP;
***Example**: To select names of all mangers from the emp table (limited columns and rows).*
SELECT ENAME FROM EMP WHERE JOB='MANAGER';
Example: To print details of the employees who joined in 'February' and the year 1981.
*SELECT * FROM EMP WHERE DOB='2011-01-01'*

a) Using Operators: Operators are relational, arithmetic, logical and special operator**s.**
***Example**: To calculate total of m1, m2 and m3 attributes of student table. Heading of the result should be "TOTAL" (Column Alias)* SELECT M1+M2+M3 *TOTAL* FROM STUDENT;
***Example**: To select records of the clerks whose salary is <=2000 from emp table.*
SELECT * FROM EMP WHERE JOB='CLERK' AND SAL<=2000;
***Example**: To select records of the employees whose comm is null from emp table.*
SELECT * FROM EMP WHERE COMM IS NULL;
***Example**: To select records of the employees whose salary is between 2000 and 4000 from emp table.* SELECT * FROM EMP WHERE SAL BETWEEN 2000 AND 4000;
***Example**: To select records of Smith, James and Allen from emp table.*
SELECT * FROM EMP WHERE ENAME IN ('JAMES', 'SMITH', 'ALLEN');

b) Using Wildcard Characters (%, _): **Characters are Percent (%) and Underscore (_).**
'%' is used to replace any number of characters, '_' is use d to replace any one character.
***Example**: To select records whose names start with 'S' from emp table.*
SELECT * FROM EMP WHERE ENAME LIKE 'S%';
***Example**: To select records whose names have only '4' characters from emp table.*
SELECT * FROM EMP WHERE ENAME LIKE 'S____';
***Example**: To select records of the employees whose names have ' i ' as second character*
SELECT * FROM EMP WHERE ENAME LIKE '_S%';

c) SORTING: **Using "Order By" Clause**
***Example**: To list the employee details in ascending order according to salary*
SELECT * FROM EMP ORDER BY SAL;
***Example**: To list the employee details in descending order according to names.*
SELECT * FROM EMP ORDER BY ENAME DESC;
***Example**: To list the employee details in ascending order according to names. If there is a matching name, then sort by empno.* SELECT * FROM EMP ORDER BY ENAME, EMPNO;

d) GROUPING**: Using "Group By" and "Having" Clauses:**
***Example**: To list the department numbers and number of employees in each department.*
SELECT DEPTNO, COUNT (*) FROM EMP GROUP BY DEPTNO;
***Example**: To list the total salary, maximum salary, minimum salary and the average salary of employees job-wise.*
SELECT JOB, SUM (SAL), MAX (SAL), MIN (SAL), AVG (SAL) FROM EMP GROUP BY JOB;
***Example**: To list the jobs and number of employees in each job. The result should be in ascending order according to the number of employees.*
SELECT JOB, COUNT (*) FROM EMP GROUP BY JOB ORDER BY 2;
***Example**: To list the jobs, which are done by minimum of 2 persons?*
SELECT JOB FROM EMP GROUP BY JOB HAVING COUNT (*)>=3;
***Example**: To list the department number in which maximum employees work*

SELECT DEPTNO FROM EMP GROUP BY DEPTNO HAVING COUNT (*) = (SELECT MAX (COUNT (*)) FROM EMP GROUP BY DEPTNO);
***Example**: To list the department number and number of clerks in each department*
SELECT DEPTNO, COUNT (*) FROM EMP WHERE JOB='CLERK' GROUP BY DEPTNO
***Example**: To list the unique jobs in emp table.* SELECT JOB FROM EMP GROUP BY JOB;
The same can be written using "DISTINCT" clause as: SELECT DISTINCT JOB FROM EMP;

e) Using SET OPERATORS**:** are union, union all, intersect and minus operators.
Assume STD1 and STD2 are tables with same attributes name, Rno and marks with few common and few unique records.
***Example**: To list combined set of unique records from multiple tables*
SELECT NAME, RNO, MARKS FROM STD1
UNION
SELECT NAME, RNO, MARKS FROM STD2;
***Example**: To list combined set of records from multiple tables including duplicates*
SELECT NAME, RNO, MARKS FROM STD1
UNION ALL
SELECT NAME, RNO, MARKS FROM STD2;
***Example**: To list common records from multiple tables*
SELECT NAME, RNO, MARKS FROM STD1
INTERSECT
SELECT NAME, RNO, MARKS FROM STD2;
***Example**: To list the records (of first table) other than common records of multiple tables*
SELECT NAME, RNO, MARKS FROM STD1
MINUS
SELECT NAME, RNO, MARKS FROM STD2;

2) INSERT **Command:** To enter data into existing table.
Syntax: INSERT INTO table_name [column_list] VALUES (value_list);
***Example**: To insert a record into emp table.*
INSERT INTO EMP VALUES (7777, 'RAVI', 'MANAGER', 7001, '12-JUL-85', 9000, NULL, 20);
***Example**: To insert only emp number, ename, job and salary*
INSERT INTO EMP (EMPNO, ENAME, JOB, SAL) VALUES (8888, 'KIRAN', 'CLERK', 4000);
***Example**: To insert a record through parameter substitution.*
INSERT INTO EMP (EMPNO, ENAME, JOB, SAL)
VALUES (101,'MOULI','LECTURER', 5000), (102,'VASAVI','TEACHER', 3500);
***Example**: To insert the result set of a query in a table.*
INSERT INTO STD2 (RNO, NAME, MARKS) SELECT RNO, NAME, MARKS FROM STD1;
With the above query, records of STD1 will be inserted into STD2

3) UPDATE **Command:** To update the values of attributes
Syntax: UPDATE table_name SET column_name=value [,column_name=value, ..]
[WHERE condition];
***Example**: To set Smith's salary to 8000.* UPDATE EMP SET SAL = 8000 WHERE ENAME='SMITH';
***Example**: To increase the salary of employees by 10%.* UPDATE EMP SET SAL = SAL+SAL*0.1;
***Example**: To update values of multiple attributes at a time*
UPDATE EMP SET SAL = 8000, COMM=100 WHERE JOB='MANAGER';

4) DELETE **Command:** To delete rows from a table
Syntax: DELETE FROM table_name [WHERE condition];
***Example**: To delete the records of clerks.* DELETE FROM EMP WHERE JOB='CLERK'

***Example**: To delete all records of a table:* DELETE FROM STD1; or DELETE STD1;

3. DATA CONTROL LANGUAGE-DCL:

DCL commands are used to control a database, including administering privileges and the committing (saving) of data. It deals with *accessibility and transaction changes.* It includes a set of privileges that can be granted to or revoked from a user.

The commands are: GRANT, REVOKE, COMMIT, ROLLBACK, and SAVEPOINT.

Oracle8i Privileges	**Capability**
SELECT	Query the object
INSERT	Insert records into the table/view. Can be given for specific columns.
UPDATE	Update records in table/view. Can be given for specific columns.
DELETE	Delete records from table/view.
ALTER	Alter the table.
INDEX	Create indexes on the table.
REFERENCES	Create foreign keys that reference the table.
EXECUTE	Execute the procedure, package, or function.
WITH GRANT	Allows to grant privileges further

1) GRANT **Command:** To give privileges on database objects to other users

Syntax: GRANT privilage_list | role ON object TO user_list | public [WITH GRANT OPTION];

***Example**: To grant all permissions on emp table to everybody.* GRANT ALL ON EMP TO PUBLIC;

***Example**: To grant Select, Delete permissions on dept table to ravi*

Use student;

GRANT SELECT, delete ON EMP:: Person.address TO RosaQdm

***Example**: To grant all permissions on emp table to kiran. Allow him to give further grants on emp to other users.* GRANT ALL ON EMP TO KIRAN WITH GRANT OPTION;

2) REVOKE **Command:** To cancel privileges on database-objects to other users

Syntax: REVOKE privilage_list | role ON object FROM user | public;

***Example**: To revoke Update privilage from everybody on emp table.*

Use student

REVOKE UPDATE ON EMP:: Person.address FROM RosaQdm

***Example**: To revoke all privileges on emp from ravi.* REVOKE ALL ON EMP FROM RAVI;

3) COMMIT **Command:** To make the changes permanent

Syntax: COMMIT TRANSACTION;

***Example**: To delete records of clerks and save the change on emp table.*

USE STUDENT

GO

Begin transaction;

Go

DELETE FROM EMP WHERE JOB='CLERK';

go

COMMIT transaction;

go

We will not be able to retrieve the records of clerk back.

4) ROLLBACK **Command:** To cancel the changes

Syntax: ROLLBACK TRANSACTION;

Canceling Changes:

***Example**: To delete records of clerks and retrieve they back*

```
USE STUDENT
GO
BEGIN TRANSACTION;
GO
DELETE FROM EMP WHERE JOB='CLERK';
ROLLBACK TRANSACTION;
GO
```

We will retrieve the records of clerk back.

5) SAVE Command: To set a margin for commit or rollback

Syntax: SAVE TRANSACTION savepoint_name;

Saving or Canceling Changes up to a Point:

```
BEGIN TRANSACTION;
GO
DELETE FROM EMP WHERE JOB='MANAGER';
SAVE TRANSACTION P1;
GO
DELETE FROM EMP WHERE JOB='CLERK';
ROLLBACK TRAN P1;
GO
```

We will retrieve the clerk-records back but not manager-records, because we have cancelled the changes up to P1 only. P1 is a save point set after deleting manager-records.

9. TYPES OF JOIN OPERATIONS:

JOINS: Processing Multiple Tables-Join is a relational operation that causes two or more tables with a common domain to be combined into a single table or view. The types of joins: *Equi-join, Non Equi-join, Natural join, Outer join, Self join, Cartesian (Product) join, Union join* are listed below. These are used to retrieve information from multiple tables.

Consider the following tables for the demonstration of joins:

STD

Rno	Name	CID
33	RAVI	10
44	KIRAN	30
55	AMAR	80

COURSE

CID	Cname
10	C
20	JAVA
30	ORACLE

Equi-join: A join in which the joining condition is based on equality between values in the common columns. Common columns appear (redundantly) in the table.

```
SELECT RNO, NAME, STD.CID, COURSE.CID, CNAME FROM STD JOIN COURSE ON STD.CID=
COURSE.CID;
```

The above query displays the values of rno, name, and course id and course name from student and course tables by joining them.

RNO	NAME	ID	CID	CNAME
33	RAVI	10	10	C
44	KIRAN	30	30	ORACLE

Non Equi-join: A join in which the joining condition is based on non-equality between values in the common columns.

```
SELECT RNO, NAME, CNAME FROM STD, COURSE WHERE STD.CID! =COURSE.CID ORDER BY
NAME
```

Lists the names of the students and subjects not known to them in alphabetical order.

RNO	NAME	CNAME
55	AMAR	C
55	AMAR	JAVA
55	AMAR	ORACLE
44	KIRAN	C
44	KIRAN	JAVA
33	RAVI	JAVA
33	RAVI	ORACLE

Natural join: Same as equi-join except one of the duplicate columns is eliminated in the result table.

SELECT RNO, NAME, STD.CID, CNAME FROM STD JOIN COURSE ON STD.CID=COURSE.CID;

Lists the values of rno, name, course id and course name from std and course tables by joining them. The common column course id is not duplicated.

RNO	NAME	CID	CNAME
33	RAVI	10	C
44	KIRAN	30	ORACLE

Outer join: A join in which rows that do not have matching values in common columns are nevertheless included in the result table. It includes left outer join and right outer join.

Left Outer Join: SELECT RNO, NAME, COURSE.CID, CNAME FROM COURSE LEFT OUTER JOIN STD ON STD.CID = COURSE.CID;

It displays all the records of course table even if there is no matching course id in std table.

Left outer join

RNO	NAME	CID	CNAME
33	RAVI	10	C
NULL	NULL	20	JAVA
44	KIRAN	30	ORACLE

Right outer join

RNO	NAME	CID	CNAME
33	RAVI	10	C
44	KIRAN	30	ORACLE
55	AMAR	NULL	NULL

Right Outer Join: SELECT RNO, NAME, STD.CID, CNAME FROM COURSE RIGHT OUTER JOIN STD ON STD.CID = COURSE.CID;

It displays all the records of student table even if there is no matching course id in course table.

Cartesian product joins: A join in which all-possible combinations of all the rows of first table with each row of the second table appear. SELECT RNO, NAME, STD.CID, CNAME FROM STD, COURSE;

If std table has 3 records and course table has 3 records the result will have a total of 9 (the product of 3*3) records.

RNO	NAME	CID	CNAME
33	RAVI	10	C
44	KIRAN	30	C
55	AMAR	80	C
33	RAVI	10	JAVA
44	KIRAN	30	JAVA
55	AMAR	80	JAVA
33	RAVI	10	ORACLE
44	KIRAN	30	ORACLE
55	AMAR	80	ORACLE

WORKER	MANAGER
VASAVI	RAVI
VASAVI	VASAVI
MOULI	KIRAN
MOULI	MOULI

Self-Join: A join in which a table is joined to itself, where joining condition is based on columns of a same table. SELECT E1.NAME "WORKER", E2.NAME "MANAGER" FROM STD E1, STD E2 WHERE E1.RNO=E2.CID; Lists the names of the employees and their managers from emp table.

B) MULTIPLE JOINS INVOLVING MORE THAN TWO TABLES:

***Example**: To list rno, name, average of marks, city from the tables std, marks and address tables Assume the tables to be:*

STD

RNO	NAME

MARKS

RNO	AVERAGE

ADDRESS

RNO	CITY

SELECT STD.RNO, NAME, AVERAGE, CITY FROM STD, MARKS, ADDRESS WHERE STD.RNO=MARKS.RNO AND MARKS.RNO=ADDRESS.RNO;

Sub Query involves placing a query with in another query. The inner query provides values to the outer query.

Correlated Query is the one in which processing the inner query depends on data from the outer query. Here, inner query must be computed for each outer row.

***Example**: To find the second maximum salary of the employees in emp table.*
SELECT MAX (SAL) FROM EMP WHERE SAL< (SELECT MAX (SAL) FROM EMP);

***Example**: To print the name, job, salary of the employee who draws maximum salary.*
SELECT ENAME, JOB, SALARY FROM EMP WHERE SAL= (SELECT MAX (SAL) FROM EMP);

***Example**: To print the names, which have maximum number of characters?*
SELECT ENAME FROM EMP WHERE LEN (ENAME) = (SELECT MAX (LEN (ENAME)) FROM EMP);

***Example**: To list the names and hiredate of employees who joined on the same day.*
SELECT ENAME, HIREDATE FROM EMP WHERE HIREDATE IN
(SELECT HIREDATE FROM EMP GROUP BY HIREDATE HAVING COUNT (*)>1);

ENAME	HIREDATE
JAMES	03-DEC-81
FORD	03-DEC-81

*****

10. Query optimization strategies - query decomposition:

Query optimization strategies involve various techniques to improve the performance of database queries. Query decomposition is one such strategy, which involves breaking down complex queries into simpler ones to optimize their execution. Here's how it works:

1. Subquery Factoring (Common Table Expressions or Inline Views): Break down complex subqueries into separate, named subqueries using Common Table Expressions (CTEs) or inline views. This can make the query more readable and allow the optimizer to handle each subquery separately, potentially improving performance.

Example:

WITH Subquery1 AS (SELECT ... FROM ... WHERE ...), Subquery2 AS (SELECT ... FROM .. WHERE ...) SELECT ... FROM Subquery1 JOIN Subquery2 ON ...

2. Use of Temporary Tables or Derived Tables: Instead of performing complex calculations or joins within a single query, break them down into separate steps by using temporary tables or derived tables. This can reduce the complexity of the main query and allow the optimizer to handle each step more efficiently.

Example:

CREATE TEMPORARY TABLE TempTable AS SELECT ... FROM ... WHERE ... SELECT ... FROM TempTable JOIN ...

3. Query Rewrite: Rewrite complex queries to simpler forms without changing their semantics. This may involve converting correlated subqueries to JOINs, using EXISTS or NOT EXISTS instead of IN, or restructuring WHERE clauses for better performance.

Example:

SELECT ... FROM Table1 WHERE EXISTS (SELECT 1 FROM Table2 WHERE Table1.ID = Table2.ID

can be rewritten as:

SELECT ... FROM Table1 INNER JOIN Table2 ON Table1.ID = Table2.ID

4. Use of Indexes: Ensure that appropriate indexes are in place for columns involved in joins, filters, and sorting. This can speed up query execution by allowing the database engine to quickly locate relevant rows.

5. Materialized Views: If certain queries are run frequently and involve complex calculations or joins, consider creating materialized views that store the results of these queries. This can reduce the overhead of recalculating the results each time the query is executed.

Example:

CREATE MATERIALIZED VIEW mv_name AS SELECT ... FROM ... WHERE ...

Partitioning: If dealing with large tables, consider partitioning them based on certain criteria (e.g., date range, key range). This can improve query performance by limiting the amount of data that needs to be scanned for each query.

By decomposing complex queries and applying these optimization techniques, you can often achieve significant improvements in query performance.

*****

UNIT-V

1. Transaction and System Concepts:

"Transaction" and "System" are two fundamental concepts in computer science, particularly in the context of databases and information systems.

1. **Transaction:** A transaction is a unit of work performed within a database management system (DBMS) or similar system, characterized by four properties, often abbreviated as ACID:
2. **Atomicity:** A transaction is atomic, meaning it either completes in its entirety or is completely undone if it fails at any point. There is no partial completion.
3. **Consistency:** The database remains in a consistent state before and after a transaction. Constraints, such as integrity constraints, are not violated.
4. **Isolation:** Transactions are executed independently of each other, and the intermediate state of one transaction is not visible to others until it is committed.
5. **Durability:** Once a transaction is committed, its changes persist even in the face of system failures.

   Transactions ensure data integrity and provide a way to maintain consistency in the database despite concurrent access and system failures.

System: In computer science, a system refers to a collection of components or elements that work together to achieve a common goal or function. These components could be hardware, software, processes, or people.

In the context of information systems, a system typically refers to a software system designed to handle some specific task or set of tasks. For example, a transaction processing system (TPS) is a type of information system that processes transactions.

A system could be:

- **Software System:** This includes applications, databases, middleware, and any other software components designed to work together to perform specific functions.
- **Hardware System:** The physical infrastructure supporting the software system, including servers, networks, storage devices, etc.
- **Human System:** This includes the people who interact with the system, such as users, administrators, and other stakeholders.
- **Transaction within a System:** In the context of information systems, transactions are fundamental to ensure data consistency and integrity. A transaction may involve operations like reading data from the system, modifying it, and saving the changes back to the system.

For instance, in banking system, transferring money from one account to another involves a transaction. This transaction ensures that money is deducted from one account and added to another account atomically, maintaining the integrity of account balances.

In summary, transactions ensure the consistency and integrity of data within a system, which is a critical aspect of system reliability and performance.

2. Desirable properties of transaction, Schedules and Recoverability, Serializability of Schedules,

Let's break down each of these concepts:

a) Desirable Properties of Transactions:

1. **Atomicity:** Transactions should be atomic, meaning they are all-or-nothing operations. Either all operations within a transaction are successfully completed, or none of them are. This property ensures that the database remains in a consistent state, even in the event of failures.
2. **Consistency:** Transactions should preserve the consistency of the database. This means that the database should transition from one consistent state to another consistent state after the successful completion of a transaction.

3. **Isolation:** Transactions should be isolated from each other. Each transaction should operate independently of other transactions. This ensures that the outcome of one transaction is not affected by the concurrent execution of other transactions.
4. **Durability:** Once a transaction is committed, its effects should persist even in the event of system failures. This property ensures that committed data is not lost and can be recovered in case of failures.

b) Schedules and Recoverability:

1. **Schedule:** A schedule is an ordered sequence of actions (read and write operations) from a set of transactions. It represents the chronological order in which transactions are executed in the system.
2. **Recoverability:** Recoverability ensures that if a transaction commits, its effects should not be lost due to subsequent failures. A schedule is recoverable if, for every pair of transactions T1 and T2, if T2 reads a data item written by T1 and T1 commits before T2 starts, then T2 can commit without causing a loss of data consistency. Recoverability is essential for maintaining the ACID properties of transactions.

c) Serializability of Schedules:

1. **Serializability:** A schedule is serializable if its outcome is equivalent to some serial execution of the transactions, meaning the transactions appear to have executed one after the other without overlapping in time. Serializable schedules ensure that the database remains in a consistent state, even if transactions are executed concurrently.
   Two common techniques for ensuring serializability are:
2. **Conflict Serializability:** This technique ensures that conflicting operations (e.g., two transactions trying to write to the same data item) are serialized in such a way that their effects are equivalent to some serial execution.
3. **View Serializability:** This technique ensures that the read-write dependencies among transactions are preserved, meaning that the results of executing transactions should be the same as if the transactions were executed one after the other in some order.

By ensuring serializability, the system guarantees that the transactions are executed in a way that maintains consistency and integrity of the database.

3. Transaction Support in SQL

SQL (Structured Query Language) provides support for transactions through a set of commands and features that ensure the ACID properties (Atomicity, Consistency, Isolation, Durability) of transactions. Here are the main components of transaction support in SQL:

Transaction Control Commands: SQL provides commands to control transactions explicitly:

- **BEGIN TRANSACTION** or **START TRANSACTION**: Begins a new transaction.
- **COMMIT**: Ends the current transaction and makes all changes permanent.
- **ROLLBACK**: Ends the current transaction and undoes all changes made since the transaction began.
- **SAVEPOINT**: Sets a named point within a transaction to which you can later roll back.
- **RELEASE SAVEPOINT**: Removes a savepoint.
- **ROLLBACK TO SAVEPOINT**: Rolls back to a savepoint within the current transaction.
- **Isolation Levels:** SQL allows you to specify the isolation level for transactions, which determines the degree to which the transactions are isolated from each other. Common isolation levels include:
  Read Uncommitted
  Read Committed

Repeatable Read
Serializable

- **Locking Mechanisms:** SQL uses locking mechanisms to control concurrent access to data during transactions to ensure isolation. Types of locks include:
- **Shared Locks (Read Locks)**: Allows multiple transactions to read a resource but prevents any transaction from writing to it.
- **Exclusive Locks (Write Locks)**: Prevents other transactions from reading or writing a resource while the lock is held.
- **Update Locks**: A combination of a shared lock and an intention to update.
- **Autocommit:** Many SQL database systems have autocommit enabled by default. With autocommit, each SQL statement is treated as a single transaction, automatically committed upon completion.
- **Transaction Logging and Recovery:** SQL databases typically log changes made during transactions to a transaction log. This log is used for recovery in case of system failure. The log allows the database to roll forward committed transactions and roll back uncommitted transactions to maintain durability and consistency.

Example (MySQL syntax):

-- Begin a transaction START TRANSACTION; -- Perform some operations INSERT INTO table1 (column1) VALUES ('value1'); UPDATE table2 SET column2 = 'new value' WHERE column3 = 'condition'; -- Commit the transaction COMMIT;

-- Begin a transaction START TRANSACTION; -- Perform some operations INSERT INTO table1 (column1) VALUES ('value1'); UPDATE table2 SET column2 = 'new value' WHERE column3 = 'condition'; -- Roll back the transaction due to an error ROLLBACK;

These commands and features provide developers with the tools necessary to ensure data integrity and consistency in SQL databases by effectively managing transactions.

4. Locking Techniques for Concurrency Control:

Locking techniques are crucial for maintaining data integrity and ensuring concurrency control in database systems, especially in multi-user environments where multiple transactions may access and modify the same data simultaneously. Here are some common locking techniques:

Shared and Exclusive Locks:

- **Shared Lock (Read Lock):** Allows multiple transactions to read a resource but prevents any transaction from writing to it. Multiple transactions can hold shared locks simultaneously.
- **Exclusive Lock (Write Lock):** Prevents other transactions from reading or writing a resource while the lock is held. Only one transaction can hold an exclusive lock on a resource at a time.

Two-Phase Locking (2PL):

- **Growing Phase:** Transactions acquire locks as they need them but cannot release any locks.
- **Shrinking Phase:** Transactions release locks but cannot acquire any new locks.
- Once a transaction releases any lock, it cannot acquire any new locks.
- Guarantees serializability and prevents deadlock.

Deadlock Detection and Prevention:

- **Deadlock Detection:** Periodically check for cycles in the lock graph and resolve deadlocks by aborting one of the involved transactions.
- **Deadlock Prevention:** Ensure transactions acquire locks in a predefined order to prevent circular waits.

Timestamp-based Concurrency Control:

- Assign a unique timestamp to each transaction.
- Use timestamps to determine transaction ordering and to resolve conflicts.

- For example, in timestamp ordering, older transactions have precedence over newer ones.

Optimistic Concurrency Control:

- Transactions execute without acquiring locks initially.
- At the commit time, check for conflicts. If conflicts are detected, the transaction may be rolled back and retried.

Multi-Version Concurrency Control (MVCC):

- Maintains multiple versions of a data item to allow concurrent transactions to read the database without locking.
- Each transaction sees a consistent snapshot of the database as of the time the transaction began.

Lock Escalation:

- Automatically converts fine-grained locks to coarse-grained locks if a transaction acquires too many locks.
- Helps reduce lock overhead and contention.

Read and Write Locks (Update Locks):

- **Read Lock:** Allows multiple transactions to read a resource but prevents any transaction from writing to it.
- **Write Lock:** Prevents other transactions from reading or writing a resource while the lock is held.

Intent Locks:

- Indicates the intention of a transaction to acquire a specific type of lock on a resource. It helps in preventing certain types of deadlocks.

Each of these techniques has its advantages and trade-offs, and the choice depends on factors like the database system, workload characteristics, and performance requirements.

5. Concurrency Control based on time stamp ordering

Concurrency control based on timestamp ordering is a method to ensure the consistency and isolation of transactions in a multi-user environment. In this approach, each transaction is assigned a unique timestamp that determines its precedence over other transactions. Transactions are ordered based on their timestamps, and conflicts are resolved according to this order. Here's how it works:

Timestamp Assignment:

- Every transaction is assigned a timestamp when it starts. This timestamp can be based on the system clock or a monotonically increasing counter.

Transaction Ordering:

- Transactions are executed according to their timestamps. Older transactions have precedence over newer ones.
- Reads and writes are allowed based on the timestamp order to maintain consistency.

Read Operation:

- A read operation of a transaction T on an item X is allowed if no transaction that wrote a value to X has a higher timestamp than T's timestamp.
- **Write Operation:**
- A write operation of a transaction T on an item X is allowed if no transaction that wrote a value to X has a higher timestamp than T's timestamp, and no transaction that read a value from X has a higher timestamp than T's timestamp.

Conflict Resolution:

- Conflicts arise when two transactions try to access the same data item concurrently.
- If a transaction T1 tries to write to an item that has been written by another transaction T2 with a higher timestamp, T1 is aborted or rolled back to maintain consistency.

- In the case of a read operation conflict, where T1 tries to read an item that has been written by T2 with a higher timestamp, T1 may be aborted or restarted depending on the system's requirements.
- **Timestamp Update:** After a transaction completes, its timestamp may be updated to ensure that subsequent transactions have higher timestamps.

Concurrency Control Policies:

- Two common concurrency control policies based on timestamp ordering are:
- **Thomas Write Rule:** A transaction that wants to write an item must first obtain the write lock and then check if any transaction with a higher timestamp has read or written that item. If so, it aborts.
- **Wound-Wait:** If a younger transaction requests a lock held by an older transaction, it is aborted and restarted.

Garbage Collection:

- Completed transactions with no influence on future transactions can be removed from the system to free up resources.

Timestamp-based concurrency control ensures serializability of transactions and avoids deadlock by providing a deterministic order for conflict resolution. However, it may introduce some overhead due to timestamp generation and management.

6. Overview of distributed data bases Overview of access control mechanism.

Sure, let's start with an overview of distributed databases followed by an overview of access control mechanisms.

a)Overview of Distributed Databases:

A distributed database is a database in which data is stored across multiple locations or nodes, often in different physical locations or on different servers. Here are some key points about distributed databases:

- **Data Distribution:** Data is distributed across multiple nodes to improve performance, scalability, and fault tolerance.
- **Replication:** Data replication is often used to improve availability and reliability. Copies of data may be stored on multiple nodes to ensure redundancy and fault tolerance.
- **Location Transparency:** Users and applications are generally unaware of the physical location of data. The system provides mechanisms to abstract the distributed nature of the database.
- **Transaction Management:** Ensuring consistency and isolation of transactions across distributed nodes is challenging. Techniques such as two-phase commit and distributed locking are used to manage distributed transactions.
- **Query Processing:** Query optimization and processing in distributed databases involve considerations like minimizing data transfer across nodes, parallel processing, and load balancing.
- **Concurrency Control:** Distributed databases require concurrency control mechanisms to ensure data consistency and isolation in a multi-user environment.
- **Security:** Security mechanisms are crucial in distributed databases to protect data integrity, confidentiality, and availability across distributed environments.

Overview of Access Control Mechanisms:

- Access control mechanisms are used to regulate access to resources in a computer system. They ensure that only authorized users or processes are granted access to resources based on predefined policies. Here's an overview:
- **Authentication:** The process of verifying the identity of users or processes. Common methods include passwords, biometrics, tokens, and certificates.

- **Authorization:** Determines what actions users or processes are allowed to perform on resources once they are authenticated. Authorization is based on roles, permissions, or access control lists (ACLs).

Access Control Models:

- **Discretionary Access Control (DAC):** Users have control over the access permissions of their own resources. Owners can set permissions for other users or groups.
- **Mandatory Access Control (MAC):** Access permissions are determined by a central authority based on security labels and policies.
- **Role-Based Access Control (RBAC):** Access is based on the roles users have within an organization. Users are assigned roles, and permissions are granted to roles rather than individual users.
- **Attribute-Based Access Control (ABAC):** Access decisions are based on attributes of the user, resource, environment, and action.
- **Access Control Lists (ACLs):** Lists associated with a resource that specify which users or groups have permissions to access the resource and what actions they can perform.
- **Security Policies:** Define the rules and guidelines for access control. Policies may include rules for granting and revoking access, handling exceptions, and enforcing security measures.
- **Enforcement Mechanisms:** The mechanisms that enforce access control policies. These can include software, hardware, or a combination of both.
- **Auditing and Logging:** Monitoring and recording access attempts and actions to detect and investigate security breaches or policy violations.
- **Database Access Control:** Specific access control mechanisms for databases include SQL privileges, database roles, and database-level security mechanisms.
- **Network Access Control:** Controlling access to network resources, such as firewalls, VPNs, and network access control lists (NACLs).

Both distributed databases and access control mechanisms are crucial components of modern computing systems, especially in environments where data is distributed across multiple locations and accessed by multiple users or processes. Ensuring data security and integrity while enabling efficient access is a significant challenge in these environments.

*****