DATABASE MANAGEMENT SYSTEMS

UNIT - I

Overview of Database Systems

Dr. Shaik Hussain Shaik Ibrahim

School of Engineering
Department of Computer Science & Engineering
Malla Reddy University
Hyderabad

Contents

Syllabus

UNIT-I: Overview of Database Systems

Overview of Database Systems: Managing Data, File system vs DBMS, Advantages of DBMS, Describing and storing data in DBMS, Queries in DBMS, Transaction Management, Structur of DBMS, Role based Access.

Introduction to Databse design: E–R Diagrams, Entities, Attributes, Entity Sets, Relationships and Relationship sets, Additional Features of ER Model, Conceptual Databse Design with ER Model.

A Database Management System (DBMS) is a software application that enables organizations and individuals to efficiently store, manage, and manipulate vast amounts of data. It provides an organized and structured approach to handling data, allowing users to interact with the information effectively. A DBMS acts as an intermediary between users and the physical data storage, shielding users from the complexities of data storage and retrieval.

As data continues to grow exponentially, the need for a DBMS becomes increasingly evident. Traditional file-based systems often fall short in managing large volumes of data and face challenges like data redundancy, inconsistency, and difficulty in data sharing.

A **database** serves as a comprehensive repository of data, offering insights into the activities of one or more related organizations. For instance, a university database that encompasses important information regarding:

Entities like students, faculty, courses, and classrooms.

Relationships among these entities, such as students' course enrollments, faculty members teaching courses, and room allocations for conducting courses.

A **Database Management System (DBMS)** is a software system that facilitates the creation, maintenance, and retrieval of structured data stored in a database. It offers a set of tools and functions to manage data efficiently while ensuring data security, integrity, and concurrency control. DBMS provides an abstract view of data, presenting it in a logical and organized manner to users and applications.

DBMS is prevalent across various domains, and its usage is pervasive in modern computing. Here are some common examples:

1. Enterprise Applications: In businesses, DBMS is used to store and manage customer data, employee records, inventory, financial information, and more.
2. Online Retail and E-Commerce: E-commerce platforms use DBMS to store product catalogs, user profiles, order history, and transaction records.
3. Social Media: Social media platforms rely on DBMS to handle vast amounts of user-generated content, profiles, and social connections.
4. Healthcare Systems: Hospitals and healthcare institutions utilize DBMS to manage patient records, medical history, prescriptions, and treatment data.
5. Banking and Finance: Banks employ DBMS to store customer account details, transaction history, loans, and financial data.
6. Educational Institutions: Educational databases manage student information, grades, course catalogs, and academic records.
7. Logistics and Supply Chain: DBMS is used to track inventory, shipments, and logistics data in supply chain management systems.

1 Managing Data

Managing data, in the context of Database Management Systems (DBMS), refers to the process of efficiently and securely organizing, storing, retrieving, and manipulating data in a structured and systematic manner. It involves handling various aspects of data, such as creating databases, defining data structures, enforcing data integrity rules, ensuring data security, and optimizing data access and storage.

Key aspects of managing data in a DBMS include:

Data Organization: DBMS organizes data in a logical structure, typically using tables in a relational database or other data structures in different database models. Data is divided into meaningful categories to represent real-world entities and relationships.

Data Storage: DBMS manages the physical storage of data on disk or in-memory to ensure efficient access and retrieval. It optimizes storage space and reduces data redundancy through normalization.

Data Retrieval: Users can query the database to retrieve specific information using structured query languages like SQL. The DBMS processes these queries and returns the requested data.

Data Manipulation: DBMS allows users to add, modify, or delete data in the database through data manipulation operations (INSERT, UPDATE, DELETE).

Data Integrity: DBMS enforces data integrity constraints to maintain the accuracy and consistency of data. Constraints prevent invalid or inconsistent data from entering the database.

Data Security: DBMS ensures that data is protected from unauthorized access and maintains data privacy and confidentiality. It implements user authentication and authorization mechanisms.

Concurrency Control: DBMS handles simultaneous access to data by multiple users to maintain data consistency. It prevents conflicts and ensures that transactions are executed correctly.

Backup and Recovery: DBMS provides mechanisms for backing up data regularly and recovering data in case of system failures or data corruption.

2 File System vs DBMS

Both file systems and Database Management Systems (DBMS) are essential components in managing data, but they serve distinct purposes and offer varying levels of functionality.

2.1 File System

A file system is a method or structure used by operating systems to organize, store, and manage files and data on a storage medium, such as a hard drive, solid-state

drive, or other storage devices. It provides a hierarchical framework for arranging files and directories (folders) in a way that facilitates efficient storage, retrieval, and manipulation of data by both users and applications.

Key features of a file system include:

1. Data Storage: File systems manage the physical allocation of data on storage devices, ensuring efficient utilization of space. They also handle issues like data fragmentation and defragmentation to optimize storage.

2. File Organization: A file system organizes files into directories or folders, creating a structured hierarchy for data storage. This hierarchical arrangement helps users locate and manage their files more effectively.

3. File Naming: Each file within a file system is identified by a unique name, which helps users distinguish and access individual files.

4. File Metadata: File systems store metadata associated with each file, including attributes like file size, creation date, modification date, permissions, and more.

5. Data Retrieval: Users and applications can search for and retrieve files from the file system using file names or directory paths.

6. Access Control: File systems provide mechanisms to control access to files and directories, allowing users to set permissions to restrict or grant access to specific users or groups.

7. File Manipulation: File systems support operations such as creating, reading, writing, updating, and deleting files. These operations are essential for data manipulation.

8. Directory Management: File systems enable the creation, deletion, and organization of directories to group related files together.

9. File System Types: Various types of file systems exist, each with its own characteristics and optimizations. Common file system types include FAT32, NTFS, HFS+, ext4, and more.

2.1.1 Disadvantages of File Systems

1. **Data Redundancy and Inconsistency:** Information Duplication: The same data may be stored in multiple files, resulting in redundant data entries. This leads to higher storage requirements and increased costs. Inconsistent Data: Due to redundancy, changes made to data might not be consistently reflected across all instances, resulting in data inconsistency. For example, an updated address might be correct in one file but outdated in another.
2. **Difficulty in Accessing Data:** Inefficient Retrieval: Conventional file systems lack efficient methods for retrieving data based on user preferences. Users might struggle to retrieve specific information in a user-friendly manner.
3. **Data Isolation:** Fragmented Data: Data is dispersed across various files, often using different formats. This makes it challenging for new applications to access the appropriate data seamlessly.
4. **Integrity Problems:** Data Validation Complexity: Maintaining data validation by modifying application programs can be cumbersome. Adding new constraints requires altering numerous programs, leading to potential inconsistencies. Integrity Constraint Enforcement: Ensuring data adheres to integrity constraints becomes intricate and error-prone due to the distributed nature of data.
5. **Atomicity:** Transaction Reliability: Guaranteeing transaction atomicity—where all operations are either fully completed or fully undone—poses challenges. Failures like power outages can lead to incomplete transactions, compromising data integrity.
6. **Concurrent Access:** Limited Concurrent Usage: File systems lack proper mechanisms to manage concurrent access by multiple users or processes to the same file simultaneously, potentially leading to conflicts and data corruption.
7. **Security Problems:** Lack of Data Security: File systems typically lack robust security measures to control unauthorized access to data. Sensitive data is vulnerable to unauthorized viewing or manipulation.

2.2 Database Management System (DBMS)

Database

A database is a structured collection of organized and interconnected data that is stored, managed, and accessed electronically. It serves as a centralized repository for storing various types of information, such as text, numbers, images, and more, in a way that allows for efficient data retrieval, manipulation, and analysis. Databases are designed to provide a structured framework for storing and managing data to meet the needs of organizations, applications, and users.

DBMS

A Database Management System (DBMS) is a software application that facilitates the creation, organization, manipulation, and retrieval of data from a database. It acts as an intermediary between users, applications, and the physical data storage, providing tools and functionalities to manage data efficiently.

A DBMS offers mechanisms for defining data structures, enforcing data integrity rules, handling data security, optimizing data access, and more. It allows users to interact with the data using query languages, such as SQL (Structured Query Language), and ensures data consistency, security, and reliability through various features and controls. In essence, a DBMS serves as a crucial tool for effectively managing and leveraging the wealth of information stored in databases. The following are some of the key features of DBMS.

2.2.1 Features of DBMS

1. Data Organization: DBMS structures data into tables, each consisting of rows and columns, establishing logical relationships between entities. This structured organization allows for efficient data management and analysis.

2. Data Naming: Each data element within a DBMS has a unique identifier, often a combination of the table and column names. This ensures precision and differentiation among individual data points.

3. Data Storage: DBMS manages the physical storage of data within the constraints of the underlying storage medium. It optimizes storage by minimizing redundancy and fragmentation.

4. Data Integrity: DBMS enforces data integrity through constraints, ensuring that data adheres to predefined rules. This safeguards data accuracy and consistency.

5. Access Control: DBMS offers granular access control, enabling administrators to define permissions for users and roles. This ensures data security and restricts unauthorized access.

6. Data Retrieval: Users employ queries to retrieve specific data from the database. The SQL (Structured Query Language) serves as a powerful tool for formulating and executing queries.

7. Data Manipulation: DBMS supports a range of data manipulation operations, including inserting, updating, and deleting records. These operations are pivotal for maintaining data accuracy.

8. Transaction Management: DBMS provides transaction management capabilities to ensure data integrity in the face of concurrent data manipulations. ACID (Atomicity, Consistency, Isolation, Durability) properties guide this process.

9. Relationship Management: DBMS excels in managing relationships between data entities. Foreign key constraints maintain referential integrity and establish connections between tables.

10. Query Optimization: DBMS employs query optimization techniques to enhance the efficiency of data retrieval and manipulation, optimizing resource utilization.

11. DBMS Types: Various types of DBMS exist, catering to diverse needs. These include relational DBMS (RDBMS), NoSQL DBMS, object-oriented DBMS, and more.

2.2.2 Functions of DBMS

1. **Defining database schema:** DBMS allows us to design how our data will be organized, and it also lets us decide who can access the data. This helps keep everything structured and secure.

2. **Manipulation of the database:** The DBMS needs to have tools for adding new information, updating existing records, deleting data we no longer need, and finding specific data when we ask for it.

3. **Sharing of database:** When many people need to use the same data, the DBMS makes sure everyone gets the right information, and it does this without compromising the data consistency.

4. **Protection of database:** The DBMS assumes the pivotal role of safeguarding the database against unwarranted access. It enforces stringent security protocols to regulate and authenticate user access, mitigating the risks associated with unauthorized access.

5. **Database recovery:** In the event of system failures or unforeseen disruptions, a robust DBMS stands prepared to facilitate the recovery of data. This indispensable feature serves to restore the integrity of the database and maintain uninterrupted data availability.

2.3 File System vs DBMS

Table 1: A comparison between File System and DBMS

	File Systems	DBMS
Structure	It is a software that manages and organizes the files in a storage medium within a computer.	DBMS is a software for managing the database.
Data Redundancy	Redundant data may be present in a file system.	In DBMS there is no redundant data.
Query Processing	There is no efficient query processing in file system.	Efficient query processing is an essential part of DBMS.
Consistency	There is less data consistency in file system.	There is more data consistency because of the process of normalization
Complexity	It is less complex	It has more complexity in handling as compared to file system
Security Constraints	File systems provide less security in comparison to DBMS	DBMS has more security mechanisms as compared to file system
Cost	It is less expensive than DBMS.	It has a comparatively higher cost than a file system.
Back–Up and Recovery	It doesn't provide backup and recovery of data if it is lost	It provides backup and recovery of data even if it is lost

3 Describing and Storing data in DBMS

The DBMS user focuses on a real-world organization, with data describing its elements. For instance, a university DB contains details about students, faculty,

and courses, along with their connections.

A data model simplifies data description, masking storage complexities. DBMS lets users define data in line with a data model. Modern systems often use the relational model.

While the DBMS's data model conceals many details, it is closer to the system's data storage than a user's perception of the enterprise. A semantic data model offers higher-level abstraction, aiding users in formulating initial data descriptions for an enterprise. Such models encompass diverse constructs to depict real application scenarios. The DBMS typically centers around a basic data model, like the relational model, rather than supporting all these constructs directly. Initially designing a database with a semantic model is valuable, then it's translated into the DBMS's supported data model.

An extensively employed semantic data model, the entity-relationship (ER) model, employs graphical representations to illustrate entities and their interrelationships.

3.1 Relational Model

The relational model is a fundamental concept in Database Management Systems (DBMS) that revolutionized how data is organized and accessed. Proposed by Edgar F. Codd in the 1970s, this model offers a structured and efficient way to store and manage data.

In the relational model, data is organized into tables, also known as relations. Each table comprises rows (tuples) and columns, where each column represents a specific attribute or field. The essence of this model lies in establishing relationships between tables through shared attributes, creating a framework for interconnecting data.

The simplicity of the relational model has led to its widespread adoption across various industries. It offers flexibility, data integrity, and powerful querying capabilities through Structured Query Language (SQL). The model's elegance lies in its ability to abstract the underlying complexities of data storage, enabling users to focus on information rather than technical intricacies.

The relational model remains a cornerstone of modern data management, serving as the foundation for many Database Management Systems.

```
-- Students Table Schema
sid INTEGER,
sname VARCHAR,
age INTEGER,
address VARCHAR,
cgpa REAL
```

Listing 1: Students Table Schema

In the context of the relational model the follwoing terms are important.

Relation: A relation refers to a table within a database. It's a fundamental concept where data is organized in rows and columns. Each row represents a record, and each column represents an attribute or field. Relations are key elements of the relational model, allowing data to be structured, stored, and retrieved systematically.

Records: Records, also known as tuples, are individual instances or entries within a relation. They represent a specific data item, combining values for each attribute defined by the table's schema. For example, in a "Students" relation, each record could represent a student's information like name, ID, and birthdate.

Fields: Fields, also known as attributes, are the individual data elements within a relation's columns. Each field holds a specific type of data, such as text, numbers, dates, or other data types. For instance, in a "Books" relation, fields could include attributes like title, author, and publication year.

Schema: A schema outlines the structure and characteristics of a relation. It defines the attributes (fields) that the relation will have and specifies their data types and any constraints. The schema provides a blueprint for how data is organized within the relation. It's like a template that ensures uniformity in how records are stored and accessed.

An example schema for a University database is as follows:

Table 2: Sample Data in Students Table

sid	sname	age	address	cgpa
101	Alice	20	123 Main St	3.75
102	Bob	21	456 Elm St	3.90
103	Carol	19	789 Oak St	3.60
104	David	22	234 Pine St	3.85

The relational model provides a solid foundation for structuring and managing data, and its robustness can be further enhanced through key strategies. Here are two important aspects:

Integrity Constraints – To maintain data accuracy and consistency, integrity constraints play a crucial role. These rules define conditions that data must adhere to, ensuring the data remains reliable. Examples include primary keys (uniquely identifying records), foreign keys (maintaining relationships between tables), and various constraints like checks, unique, and not null. Implementing integrity constraints guarantees that the stored data meets predefined criteria, preventing inconsistencies and errors.

Database Expansion – As organizations grow, their data requirements evolve. Expanding a relational database involves careful consideration of schema changes. New attributes, tables, or relationships might be required to accommodate additional data elements. This can be done through well-planned database design modifications, including adding new tables, altering existing schemas, or introducing new relationships. Ensuring that the expanded database maintains data integrity and relational coherence is vital.

In addition to the relational model, there are other data models that offer different approaches to organizing and managing data within Database Management Systems (DBMS). In addition to relational model there are other models, such as, Hierarchical Data Model, Network Data Model, Object–Oriented Data Model, Entity-Relationship (ER) Data Model.

3.2 Levels of Abstraction in DBMS

In a DBMS, levels of abstraction play an important role in simplifying data management while optimizing efficiency. These levels help manage the complexity of data storage, organization, and interaction. The three primary levels of abstraction are

1. External Schema
2. Logical Schema
3. Physical Schema

External Schema: – Customized Views for Users

The External Schema, also known as the User or View level, represents data as perceived by end users and applications. It provides tailored views of the database, showing only the data relevant to each user or application.

Purpose and Importance: External schemas enable users to interact with the database without needing to understand its underlying structure. By customizing views, users can focus on the specific data they require, enhancing usability and reducing complexity.

Functions: External schemas define subsets of data and control access permissions, ensuring that users only see the data they're authorized to access. They provide an intuitive interface for data retrieval and manipulation.

Example:

In an online bookstore, different users have distinct needs. A customer may see their order history and wish list, an author might access their published books, and an administrator could manage inventory. Each user is presented with a customized view that caters to their specific requirements. By tailoring views to user roles, the external schema enhances user experience, ensuring that individuals only interact with the data relevant to their tasks. It simplifies the interface and prevents information overload.

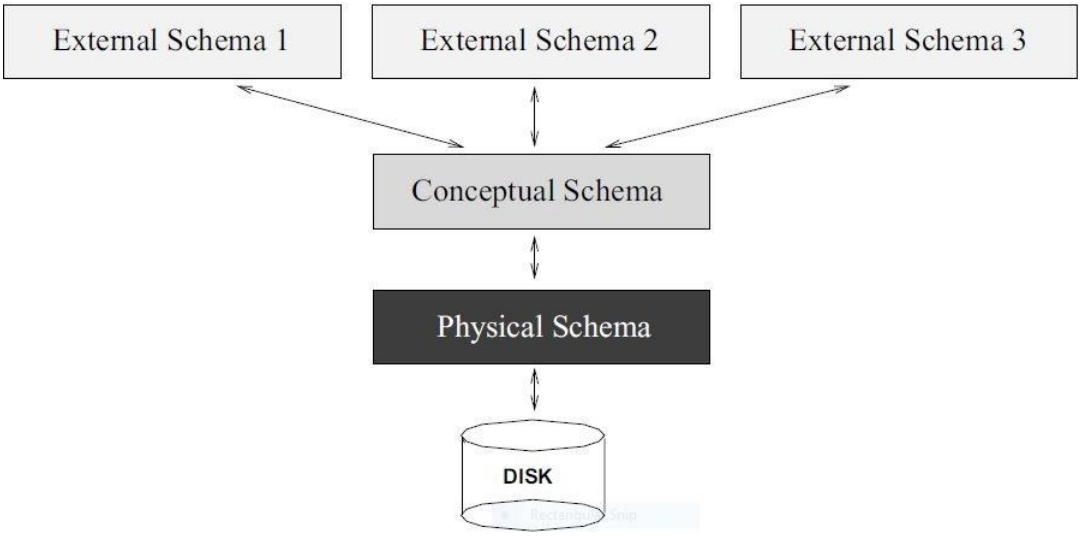

Figure 1: Levels of Abstraction in a DBMS

Logical Schema: – Structure and Relationships

The Logical Schema describes the entire structure of the database, including tables, attributes, relationships, and integrity constraints. It abstracts the way data is organized and stored physically.

Purpose and Importance: The Logical Schema separates data organization from physical storage details, allowing for a coherent and consistent data representation across various applications and user views.

Functions: Logical schemas define the overall data structure and relationships. They ensure data integrity by enforcing constraints and serve as a blueprint for efficient data manipulation.

Example:

In the online bookstore, the logical schema defines entities like ”Books,” ”Authors,” and ”Orders,” along with their attributes and relationships. It outlines how books are linked to authors and how orders are associated with customers. The schema for online book store is shown in Listing.2 The logical schema provides a standardized structure for the entire bookstore’s data. It ensures consistency in data representation across different views and applications, making data management coherent and manageable.

Physical Schema: – Storage and Performance Optimization

The Physical Schema deals with the actual storage and access mechanisms of

```
-- Books Table Schema
book_id INTEGER PRIMARY KEY,
title VARCHAR,
author_id INTEGER REFERENCES Authors(author_id),
genre VARCHAR,
price REAL,
publication_date DATE;

-- Authors Table Schema
author_id INTEGER PRIMARY KEY,
author_name VARCHAR,
nationality VARCHAR;

-- Customers Table Schema
customer_id INTEGER PRIMARY KEY,
first_name VARCHAR,
last_name VARCHAR,
email VARCHAR,
address VARCHAR;

-- Orders Table Schema
order_id INTEGER PRIMARY KEY,
customer_id INTEGER REFERENCES Customers(customer_id),
order_date DATE,
total_amount REAL;
```

Listing 2: Logical Schema for Online Bookstore

data on physical storage devices like disks. It focuses on data storage formats, indexing, and performance optimization.

Purpose and Importance: The Physical Schema abstracts the complexities of data storage and retrieval from users and applications, allowing optimization strategies without affecting external and logical views.

Functions: Physical schemas define data storage formats, indexing methods, and strategies for efficient retrieval. They help optimize storage space, enhance performance, and manage data distribution across storage devices.

The interplay of these three levels of abstraction allows the DBMS to maintain a delicate balance between providing users with a user-friendly, application-specific interface (External Schema), ensuring a coherent and structured data representation (Logical Schema), and optimizing storage and retrieval performance (Physical Schema). This separation of concerns simplifies database usage, enhances data integrity, and allows for efficient management of increasingly complex data systems.

3.3 Data Independence

One of the significant advantages offered by a DBMS is data independence. Data independence ensures that application programs remain unaffected by changes in the underlying data structure and storage methods. This crucial feature is made possible through the implementation of the three levels of data abstraction, primarily utilizing the conceptual schema and the external schema.

In the context of an online bookstore, data independence becomes evident. Imagine altering the data structure of the bookstore's database. Suppose we add a new attribute, "availability," to the "Books" entity to track the number of available copies of each book. This modification could impact how data is organized at the logical level. However, due to logical data independence, application programs that interact with the bookstore's data won't need to be extensively modified. This insulation from changes in logical structure is achieved through the external schema's view relations, which can be adjusted to maintain consistency with the conceptual schema.

Moreover, the conceptual schema in the online bookstore example shields users from alterations in physical storage details. For instance, if the database's physical storage approach evolves from using traditional hard drives to solid-state drives

(SSDs) for performance enhancement, the conceptual schema remains unaltered. This demonstrates the concept of physical data independence, allowing for modifications in storage details without disrupting applications' functionality.

Data independence is a key feature of a DBMS that ensures applications can continue to operate smoothly even when changes occur in the logical or physical aspects of the database. This separation between data management and application logic enables greater flexibility, maintainability, and longevity in the ever-evolving world of data systems.

4 Queries in DBMS

The value of a database to a user often hinges on the ease with which information can be extracted. In the realm of Database Management Systems (DBMS), the ability to pose questions efficiently is a defining factor. Relational database systems, in particular, have gained popularity due to their capacity to easily handle a diverse range of inquiries. For instance, envision an online bookstore as our sample database, and consider the following queries a user might make:

What is the title of the book with ISBN 978-1234567890?

What is the average price of books written by author John Smith?

How many copies of the book "Introduction to Databases" are in stock?

What percentage of books in the "Fiction" genre have received ratings above 4 stars?

Are there any books with a price lower than $10 and an average rating above 3.5?

These inquiries, aimed at extracting valuable insights from the stored data, are what we term "queries." A Database Management System provides a specialized language known as the query language, within which these questions can be posed. A remarkable strength of the relational model lies in its support for powerful query languages.

Efficiency in query evaluation is a critical consideration for a DBMS. The way data is physically stored plays a substantial role in determining efficiency. Indexes, for example, can expedite numerous queries. Optimal index choices can significantly accelerate query processing, benefiting each query on the list. The DBMS ensures queries are evaluated as swiftly and efficiently as possible, often considering the physical storage layout as a crucial factor.

Beyond queries, a DBMS empowers users to interact with data through a data manipulation language (DML). The query language constitutes a part of the DML, which encompasses constructs for data insertion, deletion, and modification. Thus, a DBMS not only enables users to pose insightful questions but also offers a comprehensive suite of tools for managing data effectively.

The ability to formulate and execute queries is a cornerstone of a DBMS's utility. Through query languages, users can unearth valuable insights from the data, enabling informed decision-making and fostering a deeper understanding of the stored information.

5 Transaction Management

In a DBMS, the seamless coexistence of multiple users concurrently accessing and modifying data is a crucial requirement. For instance, consider an airline reservation system where travel agents simultaneously check seat availability and make new bookings. Another example arises in banking, where multiple users perform actions like calculating deposits and transferring funds. In these scenarios, proper management of concurrent access is essential to prevent conflicts and anomalies.

For the airline reservation system, imagine two travel agents querying seat availability for the same flight simultaneously. Without careful management, one agent might see an available seat while the other agent is in the process of reserving it, leading to inconsistencies. A similar issue can arise in banking if one user computes total deposits while another transfers funds between accounts.

The DBMS must orchestrate these concurrent activities to avoid data conflicts. Concurrent access optimization aims to provide users with an illusion that they are working in isolation, despite the underlying shared data access. The DBMS ensures that transactions—individual executions of user programs—are scheduled and executed in an order that maintains data consistency.

A locking protocol is a crucial mechanism for managing concurrent transactions. Locks control access to database objects. A DBMS enforces rules where transactions acquire locks before reading or modifying data and release them after completion. Locks come in two forms: shared locks allow multiple transactions to read simultaneously, while exclusive locks grant exclusive access to a single transaction for modification.

Consider a locking protocol where transactions begin by obtaining shared locks for reading and exclusive locks for modification. Transactions release these locks after finishing their actions. For example, in an online bookstore, if one transaction intends to update book prices while another wants to display book details, the locking protocol ensures that the modification is completed before the display operation starts, maintaining consistency.

The DBMS employs locking protocols and sophisticated scheduling mechanisms to ensure that concurrent access to data remains controlled and orderly. By preventing anomalies and conflicts, the DBMS guarantees that transactions can safely coexist, enhancing data integrity and user experience.

Transactions within a Database Management System (DBMS) can sometimes be interrupted prematurely, due to factors like system crashes. In such cases, the DBMS must maintain the integrity of the database by reversing any changes made by incomplete transactions. Imagine a scenario in an online bookstore where a transaction is transferring money from one account to another. If the system crashes after debiting the first account but before crediting the second, the DBMS must ensure that the money debited is returned upon system recovery.

To achieve this, the DBMS employs a vital mechanism called a transaction log. This log meticulously records all writes to the database. Crucially, every write action is logged before its corresponding change is applied to the actual database. This precaution is known as the Write-Ahead Log (WAL) property. Ensuring WAL property is essential; it prevents a situation where a crash occurs after a change is made in the database but before it's recorded in the log. Without the ability to detect and undo such changes, data integrity could be compromised.

Moreover, the transaction log serves another crucial purpose. It safeguards the effects of successfully completed transactions against system crashes. Recovering the database to a consistent state after a crash can be time-consuming, as the DBMS must ensure that effects of all completed transactions are restored, and

incomplete transactions are reversed. Minimizing recovery time is achieved by periodically writing certain information to disk, known as a checkpoint.

6 Structure of a DBMS

The architecture of a typical relational Database Management System (DBMS), as depicted in Figure. 2, embodies a multi-layered structure that orchestrates a wide range of tasks related to query processing, transaction management, data storage, and system reliability.

Users and SQL Commands: The journey begins with SQL commands generated by users through various interfaces. These commands set the stage for interactions with the DBMS.

Parser: Once SQL commands are received, the parser takes over, breaking down the commands into a format that the DBMS can comprehend.

Optimizer: The parsed query then makes its way to the query optimizer. This component leverages information about data storage to formulate efficient query evaluation plans. These plans outline how queries will be executed, leveraging relational operators to navigate the data.

Operator Evaluator: The execution plans, containing relational operators, guide the operator evaluator in performing actual data operations as stipulated by the query. This layer bridges the gap between high-level query structures and the actual data.

Plan Executor: The operator evaluator is responsible for executing the plan created by the optimizer, traversing relational operators to retrieve and process data.

Transaction Manager: Ensuring the integrity and consistency of transactions is the role of the transaction manager. It oversees the initiation, execution, and conclusion of transactions, adhering to proper protocols.

Lock Manager: The lock manager operates in parallel with the transaction manager, managing requests for locks to maintain proper concurrency control. It grants and releases locks according to specific locking protocols.

Recovery Manager: Equipped with a transaction log, the recovery manager is essential for safeguarding data in case of system crashes. It restores the database

to a consistent state after such events, ensuring that committed transactions' effects are preserved while reversing the effects of incomplete ones.

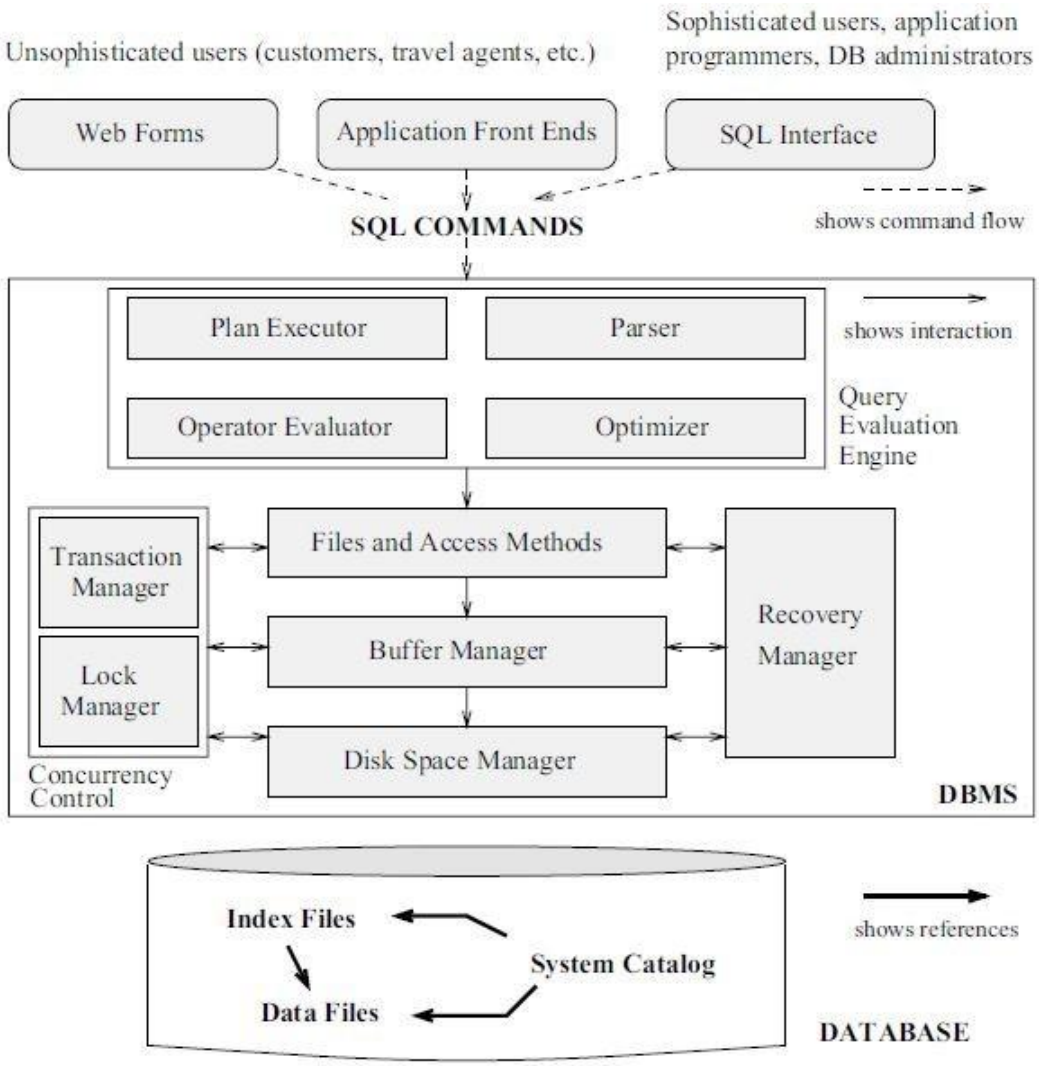

Figure 2: Structure of DBMS

Files and Access Methods: At the foundation of the structure lies the files and access methods layer. It manages the physical storage of data and supports various file types, including heap files and indexes. This layer also organizes data within pages and interfaces with the higher levels.

Buffer Manager: The buffer manager, sitting atop the files and access methods layer, is tasked with managing memory buffers, bringing necessary data from

disk to memory for efficient data retrieval.

Disk Space Manager: Responsible for space management on the disk, the disk space manager allocates, deallocates, and organizes disk space to accommodate data storage needs.

System Catalog: Although not explicitly mentioned in the provided text, the system catalog, often part of the structure, maintains metadata about the database's structure, schema, and relationships.

To provide a chronological perspective, queries are first parsed, optimized, and transformed into execution plans. Then, the operator evaluator implements these plans, fetching and processing data. During this process, transactions are managed by the transaction manager and lock manager to ensure proper concurrency control. The recovery manager maintains a safety net in case of crashes, and the files and access methods layer, buffer manager, and disk space manager work in concert to facilitate data storage and retrieval.

7 Role Based Access

People who work with DBMS are typically involved in various roles related to the design, development, maintenance, and management of databases. These roles collectively contribute to the efficient and effective use of data within an organization. Here are some key roles and responsibilities of individuals who work with DBMS:

Database Administrators (DBAs): DBAs are responsible for the overall management and maintenance of databases. They handle tasks such as database design, installation, configuration, performance optimization, security management, backup and recovery, and user management. DBAs ensure that the database system runs smoothly and securely to meet the organization's data needs.

Database Developers: Database developers design, implement, and maintain database systems. They create the database schema, define tables, relationships, and constraints. They also write SQL queries, stored procedures, triggers, and other database-related code to support application development and data retrieval.

Data Architects: Data architects focus on designing the overall structure and organization of databases. They collaborate with stakeholders to determine data requirements, design data models, define data integration strategies, and ensure data integrity and consistency across the organization.

Data Analysts: Data analysts extract insights and valuable information from databases to support decision-making. They design and execute complex queries to retrieve relevant data, analyze trends, generate reports, and visualize data using tools like business intelligence (BI) software.

Data Scientists: Data scientists leverage data stored in databases to perform advanced analytics, predictive modeling, and machine learning tasks. They extract valuable patterns, trends, and predictions from data to drive business strategies and innovations.

Application Developers: Application developers integrate databases with software applications. They create APIs, develop front-end and back-end systems, and ensure seamless interaction between applications and databases. Developers work with database APIs and libraries to implement data-related functionalities.

Database Testers: Database testers focus on validating the functionality and performance of databases. They design and execute tests to identify potential issues, such as data integrity problems, performance bottlenecks, and security vulnerabilities.

Database Security Experts: Security experts specialize in ensuring the security and compliance of databases. They implement access controls, encryption, and authentication mechanisms to protect sensitive data from unauthorized access, breaches, and data leaks.

Role-based access control in a DBMS is a security model that grants permissions to users based on their assigned roles within an organization. In RBAC, access to data and resources is determined by the user's role, rather than individually assigning permissions to each user.

Roles: Different roles are defined within the organization, representing specific job functions, responsibilities, or levels of authority. For instance, roles

could be ”Manager,” ”Employee,” ”Administrator,” etc.

Permissions: Each role is associated with a set of predefined permissions or privileges that outline what actions can be performed within the system. These permissions could include read, write, update, delete, and other specific actions.

User-Role Mapping: Users are then assigned to one or more roles based on their job requirements or responsibilities. This mapping connects users to the permissions associated with their roles.

Access Control: When a user accesses the DBMS, their permissions are determined by the roles they are assigned to. The DBMS checks the user’s role(s) and grants or denies access to specific data and functionalities accordingly.

8 Data Models

A data model is a conceptual representation of how data is organized, stored, and accessed within a database. It defines the structure, relationships, constraints, and semantics of the data that will be stored in the database. Data models provide a high-level abstraction that helps users and developers understand and communicate about the data in a standardized and systematic way.

Data models help in achieving data organization, consistency, and integrity. They provide a blueprint for database designers to create efficient structures that meet the needs of applications and users. Additionally, they aid in database management by providing a clear understanding of how data should be stored and retrieved. Different data models are chosen based on the nature of data, the requirements of the application, and the intended usage of the database system.

High-level or conceptual data models focus on presenting the data in a way that is intuitive and understandable to users and stakeholders. These models abstract away technical details and provide a high-level view of the relationships, entities, and attributes within the database. Examples of high-level data models include the Entity-Relationship (ER) model and the Unified Modeling Language (UML) class diagram.

On the other hand, low-level or physical data models focus on the technical implementation aspects of data storage and access. They provide details about how data is physically organized, stored, and retrieved on the underlying storage media, such as hard drives or solid-state drives. These models are essential for database administrators and developers who need to optimize storage and performance. Examples of low-level data models include storage structures, indexing methods, and data distribution strategies.

8.1 Entity–Relation Model (E–R) Model

The Entity-Relationship (E-R) model is a widely used conceptual data modeling technique in the field of Database Management Systems (DBMS). It provides a graphical representation of the relationships between various entities within a system, offering a clear and intuitive way to understand the structure of data and the interactions between different elements.

The E-R model is depicted using E-R diagrams, which consist of entities represented by rectangles, relationships depicted by diamonds, and attributes displayed within ovals. Lines connecting these elements indicate the associations and interactions between them.

The E-R model serves as a foundation for designing and planning the logical structure of a database. It helps database designers communicate with stakeholders, identify key entities and relationships, and ensure data integrity and consistency. The model aids in creating an organized and efficient database schema, facilitating the construction of accurate and reliable database systems.

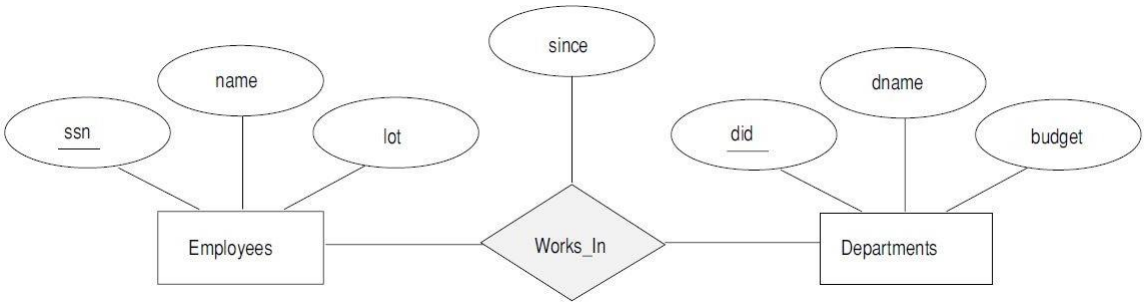

Figure 3: Example of an E-R diagram

8.2 Hierarchical model

The Hierarchical Data Model in database management is a structure that organizes data in a tree-like format, where each record is a parent or child of another record. It was one of the earliest data models used in computing and is rooted in the parent-child relationship.

What is the Hierarchical Data Model? The Hierarchical Data Model represents data in a tree structure, with a single root that connects to various levels of child nodes. Each parent node can have multiple child nodes, but each child node can have only one parent. This model reflects real-world scenarios where data is naturally organized hierarchically.

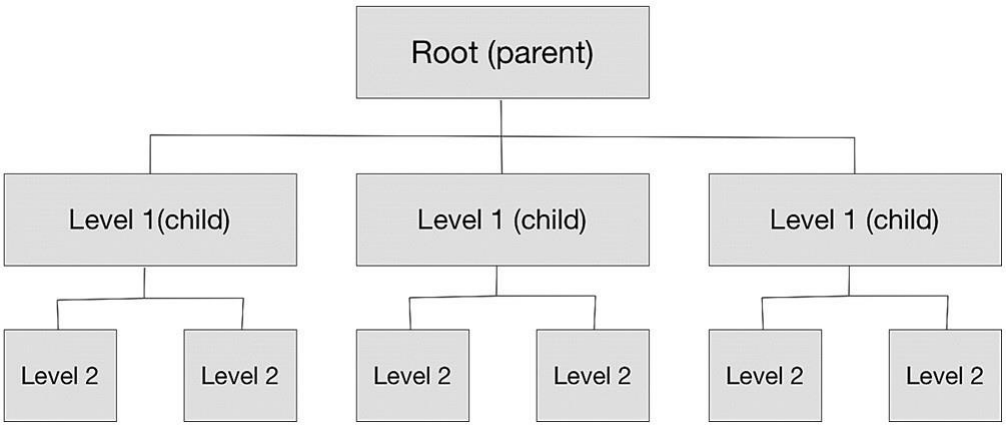

Figure 4: Hierarchical Data Model

How is Data Stored? Data in the Hierarchical Model is organized into records or segments, where each record can have multiple fields. The parent-child relationships are established through pointers or links, where a parent record points to its child records. A record can have only one parent but can have multiple children.

Usage and Implementation: The Hierarchical Data Model found its prominence in early database systems, particularly in the industry sectors like manufacturing, engineering, and network management. It is closely associated with hierarchical file systems and is used to represent parent-child relationships in various scenarios.

Data Structure: This model uses a tree-like data structure, where each parent

node can have multiple child nodes. The child nodes are linked to their parent nodes, forming a clear hierarchy. This structure allows for efficient representation of parent-child relationships.

Advantages:

1. Efficiency: Hierarchical structures are efficient for representing certain types of data with clear parent-child relationships.
2. Simplicity: The model is straightforward and easy to understand, making it suitable for certain applications.
3. Speed: Retrieving data from hierarchical structures can be efficient due to the direct parent-child relationships.

Disadvantages:

1. Lack of Flexibility: The hierarchical model is rigid and may not handle all types of relationships well, especially those that require more complex structures.
2. Data Integrity: Maintaining data integrity can be challenging when dealing with complex relationships.
3. Complex Queries: Performing complex queries that involve traversing multiple levels of hierarchy can be cumbersome.

Applications:

File Systems: Hierarchical data structures are used in organizing files and directories in hierarchical file systems.

Network Management: The Hierarchical Model is used to represent hierarchical structures in network management, such as organization hierarchies.

Manufacturing and Engineering: In scenarios where parts, components, and assemblies are organized in a hierarchical manner, this model is useful.

8.3 Network Data Model

The Network Data Model is a database management model that extends the Hierarchical Data Model by allowing multiple parent-child relationships among records. It was designed to address some of the limitations of the Hierarchical Model and offer more flexibility in representing complex data relationships.

What is the Network Data Model? The Network Data Model is a data organization structure that allows multiple parent-child relationships between records. Unlike the Hierarchical Model, where each child has only one parent, records in the Network Model can have multiple parents. This capability makes it suitable for representing complex data relationships.

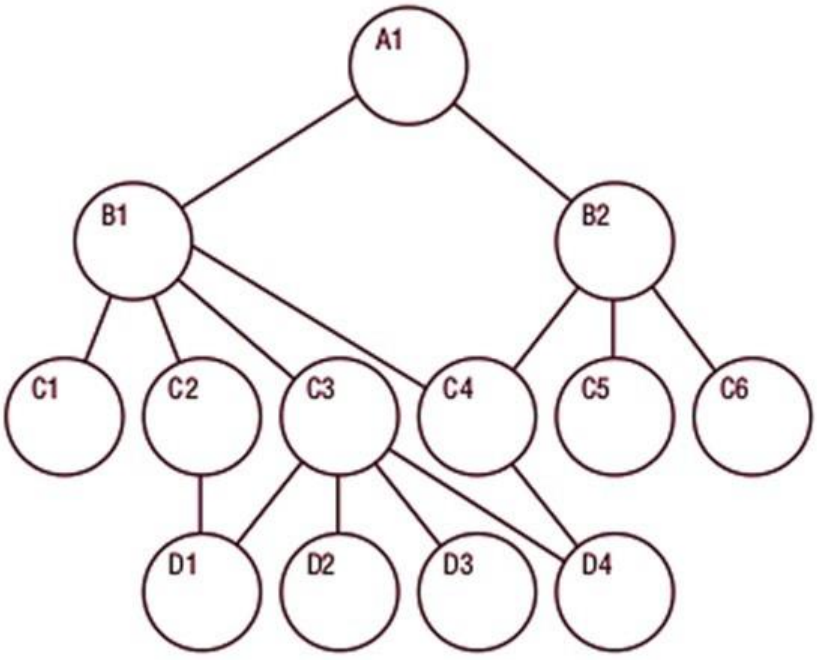

Figure 5: Network Data Model

How is Data Stored? In the Network Model, records are organized into sets (equivalent to tables in the Relational Model). Each set defines a record type, and individual records are members of these sets. Relationships are established through pointers or links that connect records from one set to another. Each record can have multiple pointers to other records, creating a network of interconnected records.

Usage and Implementation: The Network Model gained popularity in the 1960s and 1970s and found application in various fields, including scientific research, manufacturing, and business data management. It was used to represent complex data structures where multiple relationships between records were necessary.

Data Structure: The Network Model uses a graph-like data structure, where records are nodes, and the relationships are represented by arcs or pointers between nodes. This structure allows for more complex relationships and the representation of one-to-many and many-to-many relationships.

Advantages:

1. Complex Relationships: The Network Model can accurately represent complex data relationships that are not easily managed by the Hierarchical Model.
2. Efficiency: Traversing direct relationships is efficient due to the use of pointers.
3. Record Flexibility: Records can participate in multiple relationships, providing more flexibility in data representation.

Disadvantages:

1. Complexity: The Network Model's complexity can make it challenging to design and understand.
2. Data Integrity: Ensuring data integrity with multiple interrelated records can be complicated.
3. Maintenance: Modifying relationships or adding new record types can be labor-intensive.

Applications:

Scientific Research: Used to model complex relationships in scientific experiments or studies involving various variables.

Business Data: In scenarios where multiple relationships between entities are necessary, such as representing employee-job-project relationships in organizations.

Manufacturing and Engineering: Applied when managing complex relationships between parts, components, and assemblies in manufacturing processes.

8.4 Object-Oriented Data Model

The Object-Oriented Data Model is a database management model that extends the concepts of object-oriented programming to the storage and manipulation of data. It treats data as objects, which encapsulate both data and the methods that operate on the data. This model was developed to bridge the gap between traditional data models and object-oriented programming paradigms.

What is the Object-Oriented Data Model? The Object-Oriented Data Model treats data as objects, similar to the way objects are treated in object-oriented programming languages. Each object represents an entity, encapsulating its attributes (data) and behaviors (methods). This approach allows for more intuitive representation of real-world entities and their relationships.

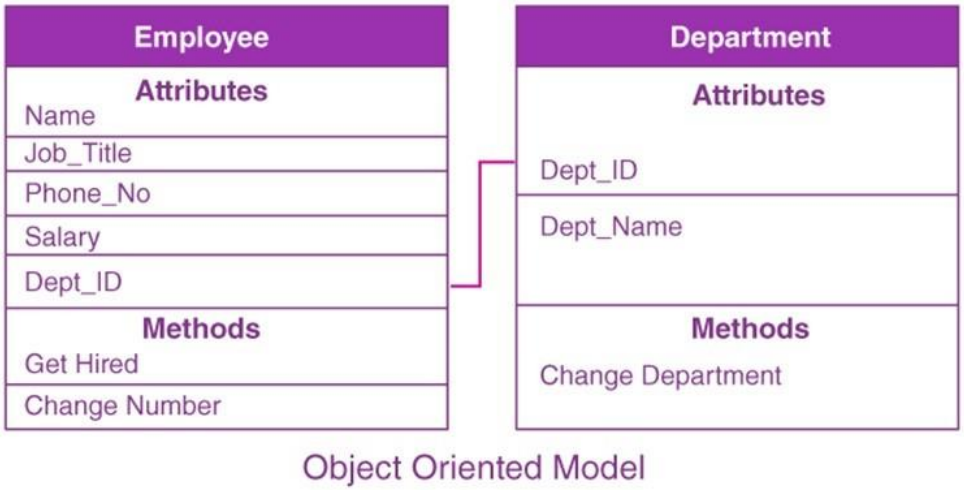

Figure 6: Object – oriented Data Model

How is Data Stored? In the Object-Oriented Model, data is stored as objects, which are instances of classes. A class defines the structure (attributes) and

behavior (methods) of objects. Objects can inherit attributes and methods from parent classes, enabling data hierarchy and abstraction. Relationships between objects are established using references.

Usage and Implementation: The Object-Oriented Data Model is used in applications where the data can be naturally represented as objects with attributes and methods. It's widely used in software engineering, multimedia databases, geographic information systems, and complex applications requiring rich data modeling.

Data Structure: The data structure in the Object-Oriented Model consists of classes, objects, attributes, methods, and relationships between objects. Objects can contain attributes that store data values and methods that define their behavior.

Advantages:

1. Real-World Mapping: The model closely resembles real-world entities, making it more intuitive for developers and users.
2. Code Reusability: Methods defined in classes can be reused across objects, promoting code efficiency.
3. Complex Relationships: Complex relationships between objects can be represented effectively through inheritance and composition.

Disadvantages:

1. Complexity: Developing and managing object-oriented databases can be complex due to the combination of data and behavior.
2. Lack of Standardization: Unlike the Relational Model, which has well-defined standards, the Object-Oriented Model lacks such standardization.
3. Performance: Object-oriented databases might face performance issues when dealing with large datasets or complex inheritance hierarchies.

Applications:

Software Engineering: Used to represent software components, classes, and their relationships in software design and development.

Multimedia Databases: Suitable for managing multimedia content like images, videos, and audio, where data and behavior are closely linked.

Geographic Information Systems: Applied to model and manage geographical features and their attributes.

8.5 Object–Relational Data Model

The Object-Relational Data Model is a database management model that combines the features of both the Relational Data Model and the Object-Oriented Data Model. It seeks to bridge the gap between the structured nature of relational databases and the flexibility of representing complex data structures and behaviors using object-oriented concepts.

What is the Object-Relational Data Model? The Object-Relational Data Model extends the Relational Model by incorporating object-oriented concepts, allowing for the storage of both structured tabular data and more complex data structures with object-like attributes and behaviors. It aims to provide the benefits of both models while addressing their limitations.

How is Data Stored? In the Object-Relational Model, data is stored in tables similar to the Relational Model, but with the addition of user-defined data types and object-like attributes. These attributes can include complex data types such as arrays, lists, and even user-defined objects. The model supports both structured tabular data and more flexible, object-oriented representations.

Usage and Implementation: The Object-Relational Model is used in scenarios where traditional relational databases fall short in representing complex data structures and behaviors. It's commonly applied in scientific and engineering applications, multimedia databases, and applications that require rich data modeling.

Data Structure: The data structure in the Object-Relational Model comprises tables, rows, columns, and relationships similar to the Relational Model. However, it also allows for the definition of user-defined data types, methods, and object-oriented features like inheritance and encapsulation.

Advantages:

1. Flexibility: Combines the structured nature of the Relational Model with

the flexibility of representing complex data structures using object-oriented concepts.

2. Rich Data Modeling: Allows developers to represent real-world entities more accurately by incorporating object-like attributes and behaviors.
3. Reuse and Extension: Supports object-oriented programming principles, enabling code reusability and extension of database functionality.

Disadvantages:

1. Complexity: Managing the integration of relational and object-oriented features can increase complexity in design, development, and maintenance.
2. Performance: Storing and querying complex object-oriented data can result in performance challenges, especially for large datasets.
3. Lack of Standardization: Like the Object-Oriented Model, the Object-Relational Model lacks standardized implementation and features.

Applications:

Scientific and Engineering Databases: Used to store and manage complex scientific data and engineering models that require both structured and object-like representations.

Multimedia Databases: Applied to represent multimedia content along with associated metadata, allowing for more comprehensive data modeling.

Complex Applications: Suitable for applications where the data structure requires a mix of tabular and object-oriented representations, such as content management systems.

9 E–R Diagrams

Entity-Relationship (E-R) diagrams are graphical representations used in database design to visually depict the structure of a database system. They provide a clear and concise way to illustrate the relationships between different entities and the attributes associated with them.

9.1 Entities:

Entities are the fundamental building blocks of an E-R diagram. They represent real-world objects, concepts, or things that have distinct properties. Entities are usually nouns and can have attributes that describe their characteristics.

An object with a physical existence (a person, car, house, or employee) or it may be an object with a logical existence (a company, a job, or a university course) forms an entity in E–R diagram.

Types of Entities:

1. **Weak Entity:** An entity that depends on another entity for its existence and cannot be identified by its attributes alone. They are represented with double rectangles as shown in Figure.7. Weak entity doesn't contain any key attribute of its own. For example, "Installment" in a "Loan–Installment" relationship.

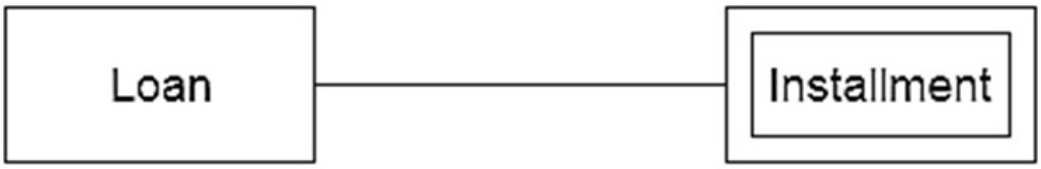

Figure 7: Weak Entity representation in E–R Model

2. **Strong Entity:** An entity that can exist independently and has attributes that uniquely identify it. For example, "Employee" in a company database. Entities are represented with rectangles as shown in Figure.8. All the employees in the database form an *Entity Set*.

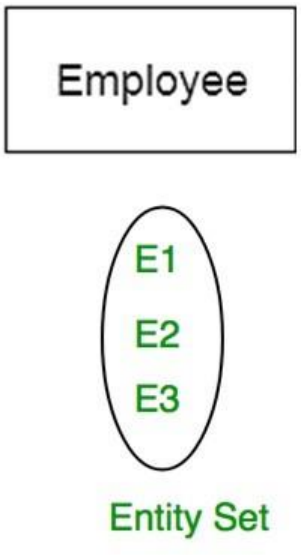

Figure 8: Entity representation in E–R Model

9.2 Attributes:

Attributes are properties or characteristics that describe entities. They provide details about entities and help define their attributes. Attributes can be of different types.

Types of Attributes:

1. **Simple Attribute:** An attribute that cannot be divided into smaller parts and holds a single value. For example, ”Name” of an employee.In ER diagram, attribute is represented by an oval or ellipse as shown in Figure.9.

Figure 9: Attribute representation in E–R Model

2. **Key Attribute** A key attribute is an attribute within an entity that uniquely

identifies instances of that entity. It is represented by an oval with underlying lines as shown in Figure.10.

Figure 10: Key Attribute representation in E–R Model

3. **Composite Attribute:** An attribute composed of multiple subparts that represent more basic attributes with independent meanings. For example, ”Address” with components like street, city, and State, and Country. A Composite attribute is shown as in Figure.11.

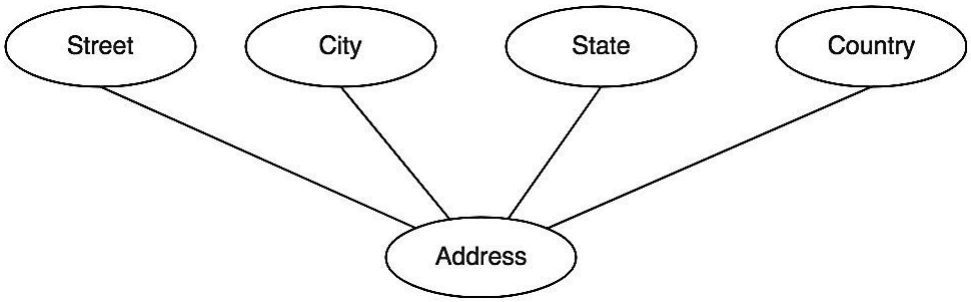

Figure 11: Composite Attribute representation in E–R Model

4. **Derived Attribute:** An attribute whose value is derived from the values of other attributes. For example, ”Age” can be derived from ”Date of Birth.” Derived attribute is represented by dashed oval as shown in Figure.

Figure 12: Derived Attribute representation in E–R Model

5. **Multi-valued Attribute:** An attribute that can hold multiple values. For example, ”Phone Numbers” of a contact. Multivalued attribute is represented by double oval as shown in Figure. 13

Figure 13: Multi–valued Attribute representation in E–R Model

9.2.1 Entity Sets, Relationships, and Relationship Sets:

Entity Sets: An entity set is a collection of entities of the same type. For instance, if ”Student” is an entity, the set of all students in the database is the ”Student” entity set.

Relationships: Relationships define associations between different entities. They represent the connections or interactions between entities. Relationships can be binary (between two entities), ternary (between three entities), etc.

Relationship Sets: A relationship set is a set of similar relationships. It defines the type of interaction between entity sets. For example, a ”Teaches” relationship set between ”Professor” and ”Course.”

References

[1] *Database Management Systems* by Raghurama Krishnan, Johannes Gehrke, TATA McGrawHill 3rd Edition

[2] *Fundamentals of Database Systems* by Elmasri Navathe Pearson Education

[3] *Database System Concepts* by Silberschatz, Korth, McGraw hill, Sixth Edition

[4] *An Introduction to Database systems* by C.J. Date, A.Kannan, S.Swami Nadhan, Pearson, Eight Edition