DATABASE MANAGEMENT SYSTEMS

COURSE CODE: B21CA05DC

Bachelor of Computer Applications

Discipline Core Course

Self Learning Material

SREENARAYANAGURU OPEN UNIVERSITY

SREENARAYANAGURU OPEN UNIVERSITY

The State University for Education, Training and Research in Blended Format, Kerala

SREENARAYANAGURU OPEN UNIVERSITY

Vision

To increase access of potential learners of all categories to higher education, research and training, and ensure equity through delivery of high quality processes and outcomes fostering inclusive educational empowerment for social advancement.

Mission

To be benchmarked as a model for conservation and dissemination of knowledge and skill on blended and virtual mode in education, training and research for normal, continuing, and adult learners.

Pathway

Access and Quality define Equity.

Database Management Systems

Course Code: B21CA05DC

Semester - III

Discipline Core Course
Undergraduate Programme
Bachelor of Computer Applications
Self Learning Material

(With Model Question Paper Sets)

SREENARAYANAGURU
OPEN UNIVERSITY

SREENARAYANAGURU OPEN UNIVERSITY

The State University for Education, Training and Research in Blended Format, Kerala

SREENARAYANAGURU
OPEN UNIVERSITY

DATABASE MANAGEMENT SYSTEMS

Course Code: B21CA05DC
Semester- III
Discipline Core Course
Bachelor of Computer Applications

Scan this QR Code for reading the SLM on a digital device.

Edition
January 2025

ISBN 978-81-984969-0-4

www.sgou.ac.in

Visit and Subscribe our Social Media Platforms

MESSAGE FROM VICE CHANCELLOR

Dear

With immense joy and excitement, I extend my heartfelt greetings to all of you and warmly welcome you to Sreenarayanaguru Open University.

Established in September 2020 as a state-driven initiative, Sreenarayanaguru Open University is dedicated to advancing higher education through open and distance learning. Our vision is guided by the principle of "access and quality define equity," laying the foundation for a celebration of excellence in education. I am delighted to share that we are steadfast in our commitment to uphold the highest standards and refrain from compromising on the quality of education we offer. The university draws its inspiration from the legacy of Sreenarayana Guru, a revered figure in the Indian renaissance movement. His name serves as a constant reminder for us to prioritize quality in all our academic endeavors.

Sreenarayanaguru Open University operates within the practical framework of the widely recognized "blended format." Acknowledging the constraints faced by distance learners in accessing traditional classroom settings, we have curated a pedagogical approach centered on three main components: Self Learning Material, Classroom Counselling, and Virtual Modes. This comprehensive blend is poised to deliver dynamic learning and teaching experiences, maximizing engagement and effectiveness. Our unwavering commitment to quality ensures excellence across all aspects of our educational initiatives.

The University aims to offer you an engaging and stimulating educational environment that fosters active learning. The SLM is designed to offer a comprehensive and cohesive learning experience, fostering a deep interest in the study of technological advancements in IT. Careful consideration has been given to ensure a logical progression of topics, facilitating a clear understanding of the discipline's evolution. The curriculum is thoughtfully crafted to provide ample opportunities for students to navigate through the current trends in information technology. Furthermore, this course is designed to provide essential insights into computer hardware, software classification, and foundational HTML concepts crucial for web development.

We assure you that the university student support services will closely stay with you for the redressal of your grievances during your studentship. Feel free to write to us about anything that seems relevant regarding the academic programme.

Wish you the best.

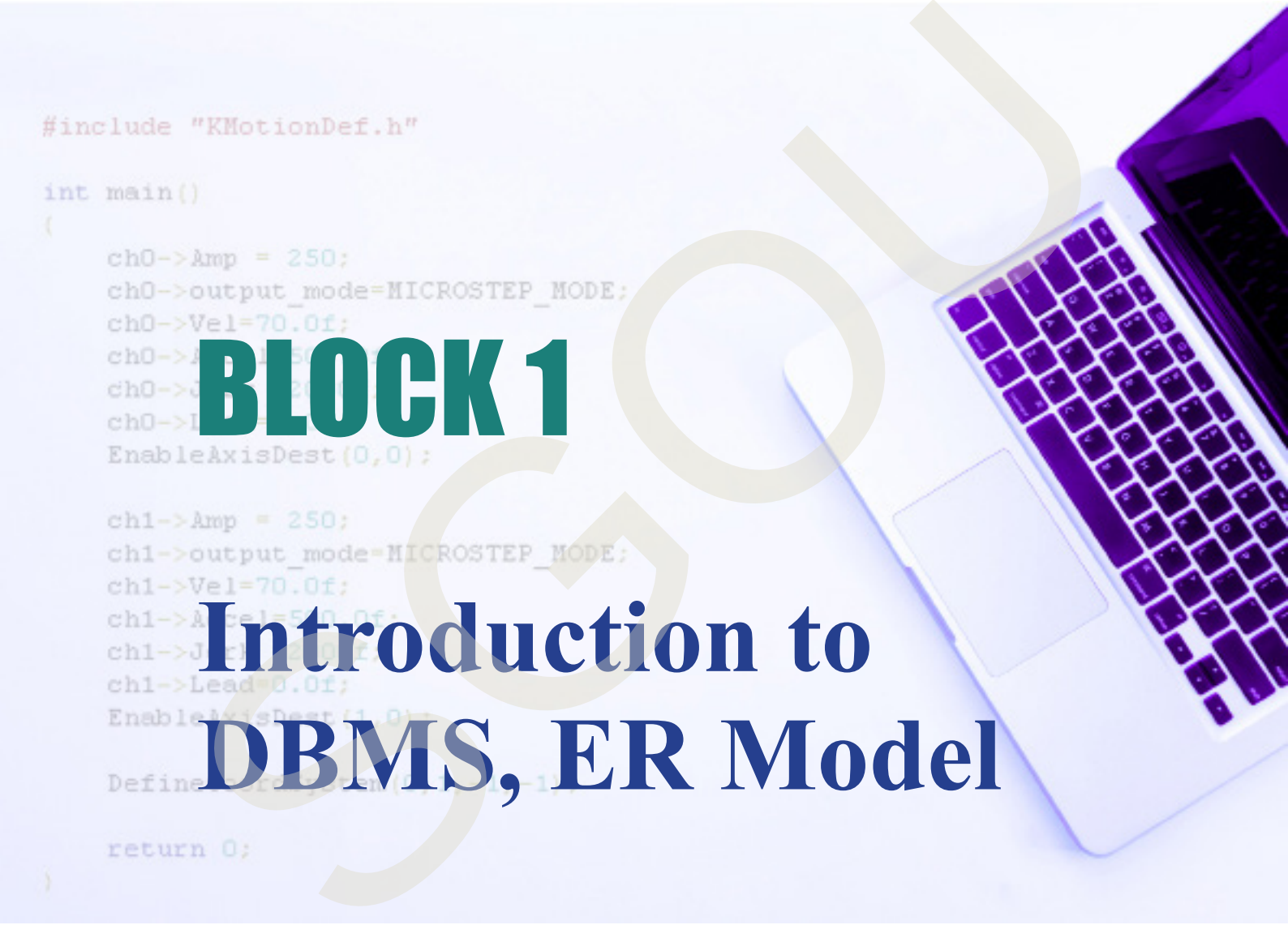
Regards,
Dr. Jagathy Raj V. P.

01-01-2025

Contents

PART - I

PART - II



BLOCK 1

Introduction to DBMS, ER Model

Introduction to DBMS

Learning Outcomes

At the end of this unit, the learner will be able to;

- Familiarize the concept of Database
- Introduce types of data abstraction
- Make aware about the data models
- Narrate about data independence

Prerequisites

Data plays a crucial role in information processing, as processed data serves as valuable information to support various operations within an organization. For example, in a hospital, patient records are processed to generate reports for diagnosis, treatment history, and billing. To ensure efficient operations, data must be properly stored, organized, and managed. Without proper management, data can become inconsistent, inaccurate, or insecure, leading to serious consequences such as poor decision-making, operational delays, or data breaches.

Organizations require accurate and timely data to perform their daily activities. Banks, for instance, need real-time data to process transactions, update account balances, and detect fraudulent activity. Educational institutions rely on accurate data to manage student enrollments, courses, and grades. Similarly, hospitals maintain sensitive patient data, which must remain confidential and secure. Each organization has different privacy and security requirements, making it essential to have a system that can enforce these rules effectively.

A database system is designed to address these needs by providing tools to store, organize, and manage data efficiently.

Keywords

Entities, relationships, main memory, database administrator, Entity-Relationship model, data independence, data models.

Discussion

1.1.1 Introduction to DBMS

Data needs to be stored in a fashion that it can be accessed very quickly and randomly when it is required and this is the primary purpose of a database. Database systems are used to manage large amounts of information. It requires both data definition structures for storage purpose and efficient mechanisms for its manipulation.

Organizations often need to manage multiple categories of data, which must be stored and retrieved in an organized and efficient manner to support various operations. A college requires data on components called as entities such as students, faculty, courses, and classes, as well as the relationships between these components. These relationships may include students enrolling in courses, faculty members teaching those courses, and classes being assigned for course sessions.

A database is an organized collection of related information that is structured in a way easy access, management and updating.In this context, a college database maintains comprehensive details about each component and their connections, ensuring data consistency and accessibility. A Database Management System (DBMS) enhances this process by providing a set of interrelated data and software programs designed to facilitate efficient access, manipulation, and management of the stored data. Through the use of a DBMS, organizations can maintain data integrity, support complex queries, and optimize their data operations for improved performance and decision-making.

1.1.2 File System versus DBMS

Traditional file system has some kinds of disadvantages as compared with database M.S. like more space, redundancy, lack of integrity, backup, etc. Some disadvantages of file system are,

1. Storage issue

Need large space to store data so we need to use disk storage and for processing we need to bring the data to main memory. Required frequent data transfer from disk to main memory.

2. Redundancy

Data duplication occurs when repeated data is stored in multiple locations. Redundancy in file systems arises due to the lack of structured data management and relationships between files, resulting in multiple copies of the same data across different

directories. Users may manually create duplicates, and file systems do not have built-in mechanisms to prevent this.

3. Difficulty in accessing related data.

Difficulty in accessing related data occurs when information is stored across multiple files without a structured system. For example, in a file system, customer information may be stored in separate files for orders, invoices, and shipping records. If a user needs to find all data related to a specific customer, they must manually search through each file to extract relevant details. This process is time-consuming and prone to errors, as there is no direct link between these files.

4. Limited security features.

File systems have limited security features, offering only basic access restrictions such as read, write, or execute permissions. For example, a company storing employee data in a file system can only restrict access to entire folders, making it difficult to enforce role-based access to specific records. Implementing more advanced security, such as restricting access to sensitive fields like salary details, becomes complex and often requires manual workarounds.So only minimal access restrictions can be imposed and it will be complex to implement.

5. Data inconsistency.

Data inconsistency occurs when multiple updates are made to the same data simultaneously without proper control. For example, in a file system, if two employees update a customer's contact information at the same time, one update may overwrite the other, leading to conflicting records. Concurrent access needs to be controlled; otherwise, multiple updates on the same data simultaneously will result in data inconsistency. Similarly, if any one updates in one file it will not effect in other file having some data, it cause data inconsistency.

1.1.3 Advantages of DBMS

1. Data independence

Data independence refers to the ability to change the schema, which is the structure of data, without affecting other parts of the system. For example, if a database administrator adds a new column, such as "Date of Birth," to a table (internal schema), application programs that use the table will continue to function without needing any changes. Similarly, if the way data is stored on disk is modified (e.g., by adding an index to improve search speed), users and programs accessing the data are not impacted because the logical structure remains the same. This separation between internal storage and logical structure simplifies system updates and maintenance.

Data independence means ability to modify schema definition in one level without affecting the schema definition in the next higher level.

2. Efficient data access

Needed data can be retrieved in a convenient and efficient manner. Efficient data access allows users to retrieve the required data quickly and conveniently. In database systems, optimized indexing and query mechanisms help reduce search time. This improves performance and ensures faster access to large amounts of data without unnecessary delays.

3. Security and integrity of data

Security is maintained by allowing only authorized users to access the database, preventing unauthorized access and data breaches. Integrity is ensured by managing concurrent access, meaning that when multiple users access or update the same data simultaneously, the system prevents conflicts and inconsistencies, ensuring that data remains accurate and consistent at all times. Only authorized users can access the database for maintaining security. Concurrent access to the same data will be managed for providing integrity.

4. Data administration

DBMS have central control of both data and programs accessing that data. The person having such Central control over the database system is called the database administrator (DBA).

5. Minimized Application development time

Minimized application development time refers to how a Database Management System (DBMS) reduces the effort needed to develop applications by managing many key functions like data storage, retrieval, and security. In traditional systems, developers had to create separate applications to handle tasks such as searching, updating, and maintaining data, which was time-consuming and complex. With a DBMS, these tasks are automated and standardized, allowing developers to focus on the core functionality of their applications, thus speeding up development and improving efficiency.

1.1.4 Data Models

A Database Management System (DBMS) allows users to define how data is stored using a **data model**, which serves as a set of high-level constructs to describe data and hides the complexities of low-level storage. This data model provides conceptual tools to represent data, relationships between data, the meaning (semantics) of the data, and rules to ensure data consistency. Among the various data models, the **relational model** is the most commonly used in modern DBMSs, where data is organized in tables with rows and columns.

A **semantic data model** offers a more abstract, high-level representation of data, making it easier for users to define their data structures initially. This abstract model can later be translated into a more concrete database design that the DBMS supports. One popular semantic data model is the **Entity-Relationship (ER) model**, which visually represents data through entities (objects) and their relationships, simplifying the design process for complex databases.

Relational Model

In the relational model, a relation is the primary data structure used to describe and organize data. A relation can be thought of as a table containing rows (records), where each record represents an individual entry, and columns (fields) define the attributes of that entry. The schema provides a description of the data in terms of the data model by defining the name of the relation (table), the names of its fields (attributes), and the data types for each field.

For example, in a college database that stores information about students, a relation schema might look like this:

Student *(stud_id:* string, *stud_name*: string, *login*: string, *age:* integer, *gpa:* real).

This schema indicates that each record (row) in the Student relation will have five fields: a student ID, student name, login, age, and GPA. The data types such as string and integer specify the kind of data that each field can hold. This structured approach helps maintain consistency, allows efficient querying, and provides a clear framework for organizing and accessing data.

Table1.1.1 Student Table

Stud_id	Stud_name	login	age	gpa
2100	Smith	Smith@cs	20	4.2
2155	Jack	Jack@hindi	21	3.9
2345	Nimal	Nimal@english	20	3.5
2454	Job	Job@history	22	3.8

From the above table 1.1.1, every row follows the schema of the Student relation. Each row in the Students relation is a record that describes a student. The description about a student is adequate for college purposes, so it can be considered as complete. Integrity constraints can be applied by specifying every student has a unique stud_id.

Relational data model used in various systems which includes IBM's DB2, Informix, Oracle, Sybase, Microsoft's Access, FoxBase, Paradox, Tandem and Teradata.

1.1.5 Other data models

Other important data models include

1. The hierarchical model

The hierarchical data model organizes data in a tree-like structure, where each piece of data, called a record, is connected by links that show relationships. In this model, one record (parent) can have multiple linked records (children), but each child can only have one parent. For example, imagine a university system where the main record is

the University, which has several Colleges as children. Each College can have multiple Departments, and each Department can have many Students. For instance, the College of Applied Science might have a Computer Science Department, which includes students like "John Doe" and "Jane Smith." This structure makes it easy to quickly find related data, such as viewing all students in a department by following the links. However, it has some limitations, such as being difficult to update when new relationships are needed and being unable to easily handle complex relationships where data belongs to more than one parent. This model was commonly used in older systems like IBM's IMS for managing business data.The hierarchical model is used in IBM's IMS DBMS.

2. The network model

The network model represents data as a collection of records, with relationships between them defined by links that act like pointers. Unlike the hierarchical model, which follows a strict tree structure, the network model allows records to be connected in a more flexible structure called a graph. This means a record can have multiple parents and multiple children, making it suitable for complex relationships. For example, in a library database, a book record might be linked to multiple author records, while an author can be linked to multiple book records.

This model is commonly used in systems like IDS (Integrated Data Store) and IDMS (Integrated Database Management System). It provides faster data access for complex queries and is useful for applications requiring many-to-many relationships, such as supply chain networks and banking systems. However, managing and maintaining these links can be complex and requires careful planning.

3. The object oriented model

The object-oriented model organizes data as a collection of objects, similar to how objects are used in object-oriented programming. Each object represents a real-world entity and contains both data and methods. The data is stored in instance variables (attributes), and the methods define the operations that can be performed on the object.

For example, in a library database, a Book object might have attributes like "title," "author," and "ISBN," as well as methods like "borrow" or "return." This model makes it easier to handle complex data by encapsulating both data and behavior within objects. The object-oriented model is commonly used in advanced database systems such as ObjectStore and Versant, which are designed to efficiently manage large sets of interconnected objects. This approach simplifies the development of applications that require complex data structures, such as multimedia databases or simulations.

4. The object-relational model:

The object-relational model is a combination of the relational model and object-oriented concepts. It enhances the relational model by allowing the database to store complex data types, such as objects, which are commonly used in object-oriented programming. In this model, data is still organized in tables like in the relational model, but those tables can contain objects with attributes (properties) and methods (functions). For example, consider a university database where a Student record can not only have basic fields like name and ID but also include an object for Address, which has its own

fields like street, city, and zip code. This allows better data organization and reusability. Object-relational databases are used in many modern database systems, such as those from IBM, Oracle, and Informix. These systems support complex applications that need both structured data and advanced features like inheritance, encapsulation, and polymorphism, making data management more efficient and flexible.

1.1.6 Levels of abstraction

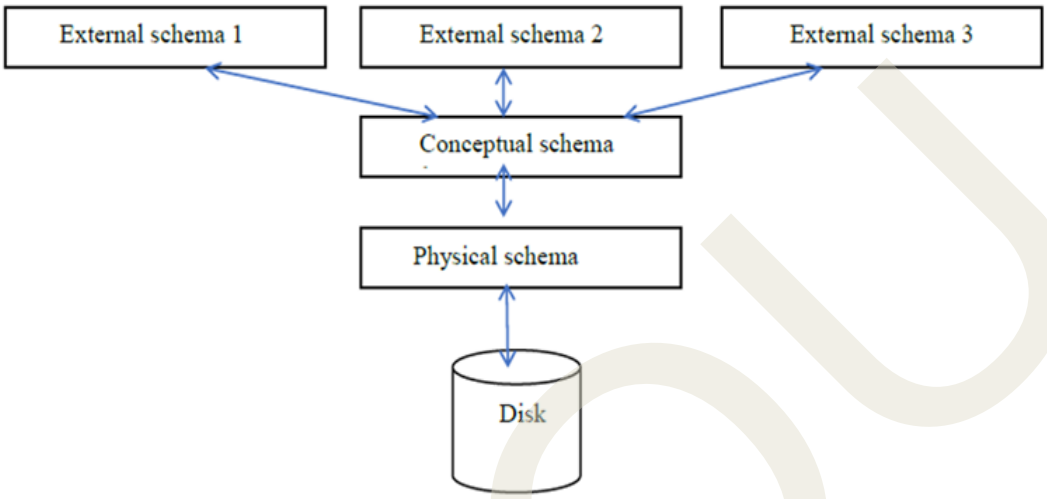


Fig. 1.1.1 Abstraction levels in database

Data in a DBMS describes at three levels of abstraction.

1. Physical Schema.
2. Conceptual Schema.
3. External Schema.

The Conceptual and External schemas in a database are defined using a Data Definition Language (DDL), which specifies the structure of the database. The Conceptual schema, also known as the logical schema, describes how data is organized and stored according to the data model of the Database Management System (DBMS). It defines the overall structure of the database by detailing all the tables (or relations) and their attributes. This schema provides a complete and consistent view of the data stored in the system without focusing on how the data is physically stored.

For example, in a university database, the conceptual schema might define the following tables:

- **Student:** Stores student details with attributes such as *stud_id, stud_name, login, age,* and *gpa.*
- **Faculty:** Contains information about faculty members, including *fac_id, fac_name,* and *sal.*
- **Course:** Represents academic courses with attributes like *c_id, c_name, credits*, and *enrolment.*

- **Classes:** Stores details of classrooms, including *class_no, class_address*, and *capacity.*
- **Enrolled:** Represents the relationship between students and courses, with attributes such as *stud_id, c_id,* and *grade.*

This conceptual schema helps in organizing and linking data logically and acts as a blueprint for how data is represented and maintained across the system. The conceptual schema representation for above table is,

Student (stud_id: string, stud_name: string, login: string, age: integer, gpa : real)

faculty (fac_id: string, fac_name: string, sal: real)

course (c_id: string, c_name: string, credits: integer, enrolment: integer)

classes (class_no: string, class_address: string, capacity: integer)

Enrolled(stud_id: string, c_id: string, grade :string)

The physical schema explains how the data described in the conceptual schema is actually stored in the database. It specifies details like how the data is organized into files, how records are stored on disk, and whether indexes are created to speed up data retrieval. For example, a table containing student records may be stored as a collection of records or pages, and an index on the student ID may be created to allow faster searches.

Every database has one physical schema and one conceptual schema because there is only one physical storage system for the entire database. However, databases can have multiple external schemas, which define views, customized representations of the data for different users or purposes. These views do not store actual data but instead display data extracted from the database.

For example, if a student wants to see course details, a view can be created to show only the course name, faculty name, and number of enrolled students. The student does not need access to all other data in the system, such as administrative or financial records, which are hidden by the external schema. This structure improves both security and usability by tailoring data access to the needs of specific users.

Examples:

Courseinfo (c_id: string, fac_name: string, enrolment: integer)

As explained above Courseinfo is not included in the conceptual schema. It can be computed from the relations in the conceptual schema.

1.1.7 Data Independence

Data independence means that changes made to the database's internal structure do not affect how users interact with the data through applications. It allows application programs to be insulated from changes in the physical and conceptual schemas. This independence is achieved through three levels of data abstraction: the physical schema, conceptual schema, and external schema.

1. **Physical Schema:**

   The physical schema describes how data is actually stored on disk. It includes details like file structures, storage formats, and indexes. For example, a database administrator may decide to change the way student records are stored by creating an index on student IDs to speed up search operations. This change improves performance but does not impact how users or applications access the student records

2. **Conceptual Schema:**

   The conceptual schema provides a high-level view of the data, focusing on the organization of tables and relationships. For instance, a Student table may be linked to a Courses table through an enrollment relationship. Users working with this structure only need to understand how the tables are related, not how the data is physically stored.

3. **External Schema (or View):**

   The external schema defines how specific users or applications view the data. It creates views that present only the necessary data while hiding other details. For example, a student may have access to a view that shows course information (course name, instructor, and enrollment count) without seeing sensitive data like faculty salaries.

Types of Data Independence

1. Physical Data Independence:

Physical data independence insulates users and application programs from changes to the physical storage of data. If changes are made to organisation of data on disk (e.g., switching from a sequential file format to a partitioned structure), users will not be affected. The database management system (DBMS) handles these changes internally. This allows administrators to optimize performance without disrupting application functionality.

2. Logical Data Independence:

Logical data independence protects users from changes to the conceptual schema, such as modifications to the organization or structure of the database tables. For example, in a college database, the Faculty table might originally have fields like faculty ID, name, department, and salary. To improve security, the database administrator may split this table into two separate relations:

- Faculty_Public (fac_id, fac_name, dept)
- Faculty_Private (fac_id, sal)

Now, sensitive information like salaries is stored in a separate table, but a CourseInfo view can be redefined to pull data from both relations. Users querying this view still receive the same results as before, without knowing that the underlying structure has changed. This ability to modify the logical structure while maintaining data access

consistency is an example of logical data independence.

Consider a college database with multiple users, including students, faculty, and administrators. Each user needs access to different types of data:

- Students: They may view course information, including course names, faculty names, and enrollment numbers.
- Faculty: They can access information about their assigned courses and students enrolled in those courses.
- Administrators: They have access to detailed records, including faculty salaries and department budgets.

Now, let's say the database is restructured to improve data privacy. The Faculty table is split into Faculty_Public and Faculty_Private, where sensitive data like salaries is moved to the Faculty_Private table. However, the CourseInfo view is updated to combine both tables and still provide complete information to authorized users. This change in the schema does not require updates to application programs or user queries because the external schema (view) remains the same.

If the database administrator decides to optimize performance by reorganizing how course records are stored on disk, users remain unaffected. The conceptual and external schemas abstract away these low-level details, maintaining a consistent user experience.

Advantages of Data Independence

Data independence is a critical feature of modern databases, achieved through the separation of physical, conceptual, and external schemas. By insulating users from changes at lower levels, it enhances flexibility, security, consistency and maintainability, making databases more efficient and reliable for both administrators and end-users.

Recap

- Databases store and manage large amounts of information for quick and random access.
- A DBMS supports the organization, retrieval, and manipulation of data through structured storage.
- Organizations use databases to manage multiple data entities and their relationships, such as students, faculty, and courses in a college.
- In a file system, frequent data transfers between disk and main memory slow down data processing.
- File systems experience redundancy when the same data is duplicated across multiple locations without control mechanisms.

- Accessing related data is difficult in file systems due to scattered information across separate files.
- File systems have limited security features, offering only basic access control mechanisms.
- Data inconsistency occurs in file systems when simultaneous updates result in conflicting data records.
- Data independence allows changes in one schema level (physical, conceptual, or external) without affecting others.
- Efficient data access in a DBMS is achieved through indexing and query optimization for faster retrieval.
- A DBMS ensures security by restricting access to authorized users and managing concurrent access to maintain data integrity.
- A database administrator (DBA) centrally manages data and applications accessing the database.
- The relational model organizes data in tables (relations), with rows representing records and columns defining attributes.
- Other data models include hierarchical, network, object-oriented, and object-relational models.
- DBMS data abstraction levels include physical, conceptual, and external schemas, each serving different functions to simplify data handling and accessibility.

Objective Type Questions

1. What is the primary purpose of a database system?
2. Which system is used to manage large amounts of data efficiently?
3. Which system provides efficient data access and manipulation in organizations?
4. What is required for processing data stored in disk storage?
5. What term refers to storing repeated data in multiple locations in a file system?
6. In file systems, what challenge occurs when related data is scattered across multiple files?

7. What type of access control does a file system primarily support?
8. What problem arises from multiple users updating the same data simultaneously in a file system?
9. What feature allows database schema changes without affecting application programs?
10. What mechanism helps retrieve data efficiently in a DBMS?
11. Who has central control of a database system?
12. Which data model organizes data into tables, rows, and columns?
13. What is the main structure used in the relational data model?
14. Which data model is commonly used in systems like IBM's IMS?

Answers to Objective Type Questions

1. Storage
2. DBMS
3. DBMS
4. Memory
5. Redundancy
6. Search difficulty
7. Permissions
8. Inconsistency
9. Logical Data Independence
10. Indexing
11. DBA
12. Relational
13. Relation
14. Hierarchical

Assignments

1. Explain the concept of data independence in a database management system (DBMS) with examples of different schema levels.
2. Compare and contrast file systems and database management systems in terms of redundancy, data access, security, and consistency.
3. Describe the relational data model, including its components such as tables, rows, and columns, with a detailed example.
4. Discuss the key advantages of using a DBMS over traditional file systems, focusing on security, integrity, data access, and administration.
5. Explain the hierarchical and network data models, including their structure, real-world applications, and advantages and limitations.

Suggested Reading

1. Ramakrishnan, R., & Gehrke, J. (2002).*Database management systems (3rd ed.)*. McGraw-Hill.
2. Silberschatz, A., Korth, H. F., & Sudarshan, S. (2011)., *Database system concepts (6th ed.). McGraw-Hill.*

Reference

1. https://www.geeksforgeeks.org/dbms
2. https://www.tutorialspoint.com/dbms/dbms_architecture.htm

Database System Architecture

Learning Outcomes

At the end of this unit, the learner will be able to;

- To familiarize the concept of database Architecture
- To introduce types of database Architecture
- Make aware about the Single tier and Multi-tier Architectures
- To narrate about the benefits of transaction managers, file managers and buffer managers

Prerequisites

Imagine an online retail platform like Amazon that handles millions of records, including products, customers, and orders. When a customer searches for a product, the system needs to quickly retrieve data such as product names, prices, and availability. At the same time, other users may be placing orders or updating their profiles. Without a proper system to manage data, these simultaneous operations could lead to slow processing, incorrect data updates, and even system failures. To prevent such issues, businesses use a structured approach to efficiently store, retrieve, and manage large volumes of data.

In such a system, data is organized based on a data model, which defines the structure and relationships between different data elements. The relational model is the most commonly used, where data is stored in tables consisting of rows (records) and columns (attributes). In a retail system might have tables for products, customers, and orders, each with specific fields like product IDs, customer details, and order amounts. These tables are interconnected to allow smooth data retrieval and updates.

Keywords

External schema, Conceptual Schema, Physical schema, multi tier architecture, single tier architecture, File server, Client computer, transaction manager, Concurrency Control.

Discussion

A DBMS is software designed to manage large volumes of data efficiently. It ensures quick data access, concurrent transaction handling, data integrity, security, and system recovery from failures. It allows multiple users and applications to interact with the database simultaneously while maintaining data accuracy and security.

Data in a DBMS is structured based on a data model, which defines how data is organized and accessed. The most popular model today is the relational model, where data is stored in tables with rows (records) and columns (attributes). For example, in the retail system, there might be separate tables for products, customers, and orders, all related through keys and relationships.

The different levels of data management offer data independence, meaning that changes in data storage or organization do not disrupt user views or application logic. This separation simplifies system maintenance and improves flexibility, allowing the system to scale as data grows.

A DBMS is therefore crucial for organizations that rely on complex data operations. By offering efficient data access, security, and scalability, it helps businesses like Amazon deliver a smooth and reliable user experience, even with millions of transactions occurring every day

1.2.1 Database Architecture

The architecture of a DBMS can be either single-tier or multi-tier. In a single-tier architecture, the database and applications run on the same system, which is suitable for smaller setups. In contrast, multi-tier architecture divides the system into multiple layers or modules, such as presentation, application, and database layers. This n-tier architecture allows for better scalability, security, and maintainability. For example, in a three-tier architecture, the user interface (external level) is handled by a web application, the application logic (conceptual level) processes user requests, and the database server (physical level) manages data storage and retrieval.

These three levels of abstraction in a DBMS provide a flexible and efficient way to manage large amounts of data, ensuring that users can access data securely and reliably without being affected by changes to the underlying storage or data structures. This design makes DBMSs suitable for a wide range of applications, from small businesses to large enterprises.

1.2.1.1 One tier architecture

The **one-tier architecture** is a simple database architecture where all operations, including data storage, application logic, and presentation services, are handled on a single machine, often referred to as the file server. In this setup, the communication occurs between the **file server** and the client computers. There is no dedicated separation between application services and the database, meaning both the data management and user interface are tightly coupled on the same server.

The file server is responsible for performing all major tasks, including:

1. **User Interface:** Providing a way for users to interact with the data, such as through forms or graphical interfaces.
2. **Presentation Services:** Handling how data is displayed to the user.
3. **Application Logic:** Processing user requests, managing business logic, and performing data manipulation operations.
4. **Data Storage:** Storing and retrieving files or data from the server's storage system.

For example, in a small office setup, a **local database application** installed on a file server may store all employee records. When a client computer requests data, the file server performs all the processing, retrieves the data, and returns the result to the client. The client computer mainly displays the output without performing any complex operations.

This architecture is simple to implement and suitable for small-scale environments with limited users. However, it has several limitations, including scalability and performance issues. As all tasks are processed on the file server, it can become a bottleneck under heavy workloads or with multiple concurrent users. Additionally, since the client computers depend entirely on the file server for data access and processing, any failure on the server can disrupt the entire system.

One-tier architecture is efficient for standalone applications but may not be suitable for larger organizations requiring high availability, performance, and scalability. Shown as Fig. 1.2.1 One tier architecture.

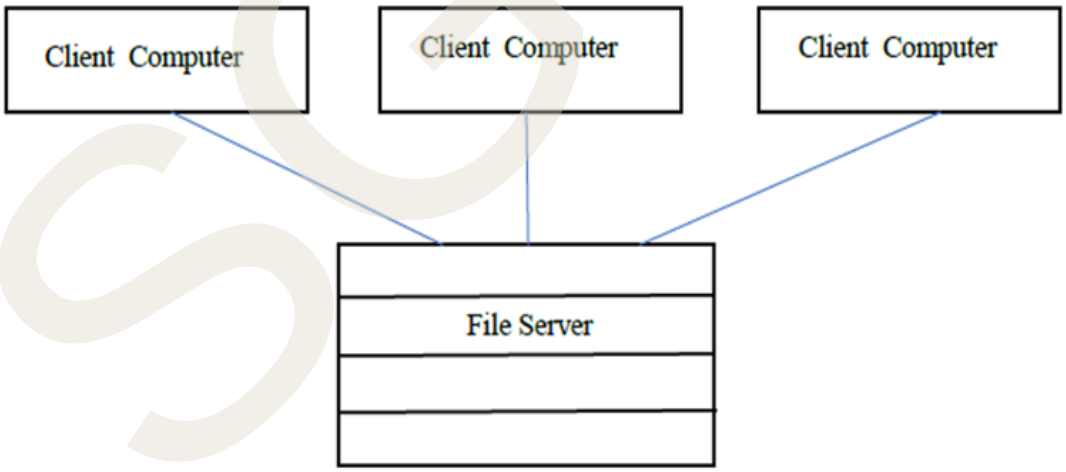


Fig. 1.2.1 One tier architecture

1.2.1.2 Two tier architecture

A **two-tier architecture** is a type of **client-server architecture** where communication occurs directly between client computers and the database server. In this setup, the client application (or user interface) and the database server are the two main components. The client sends requests to the server, such as queries to retrieve or update data, and

the server processes these requests and returns the results to the client.

Structure of Two-Tier Architecture

Client Tier:

This is the front-end application that users interact with. The client handles user inputs, displays data, and sends queries to the server. Examples include desktop applications like accounting software or student management systems.

Database Server Tier:

The server is responsible for managing data storage, handling queries, and enforcing security and data integrity. It processes client requests, executes database operations, and sends responses back to the client. The server is typically located within the organization's network infrastructure.

How Communication Works

The client establishes a direct connection to the database server over the network. When a user performs an action on the client application (e.g., searching for a product in inventory), the application sends a query to the server. The server retrieves the requested data, processes it, and returns the result to the client, where it is displayed to the user. This real-time interaction enables quick and efficient data access.

1.2.1.2 Benefits of Two-Tier Architecture

1. Direct Communication:

Clients communicate directly with the database server, reducing intermediate layers and improving response times.

2. Faster Performance:

Since there are fewer components between the client and server, data retrieval and processing operations are faster compared to multi-tier architectures.

3. Simplified Setup:

Two-tier systems are easier to develop and maintain due to their straightforward structure, making them suitable for smaller organizations or applications with limited users.

Consider a small college's student information system. In a two-tier setup, client computers used by faculty and staff run an application that connects directly to a database server within the college. When a faculty member queries student records, the client application sends a request to the server, retrieves the data, and displays it instantly. This setup allows for quick access to information without the need for additional layers of application processing. While two-tier architecture offers simplicity and speed, it may have limitations in scalability and security, especially when handling a large number of clients. In such cases, organizations may adopt a more complex multi-tier architecture to distribute workloads across different layers. Fig. 1.2.2 shows two tier architecture.

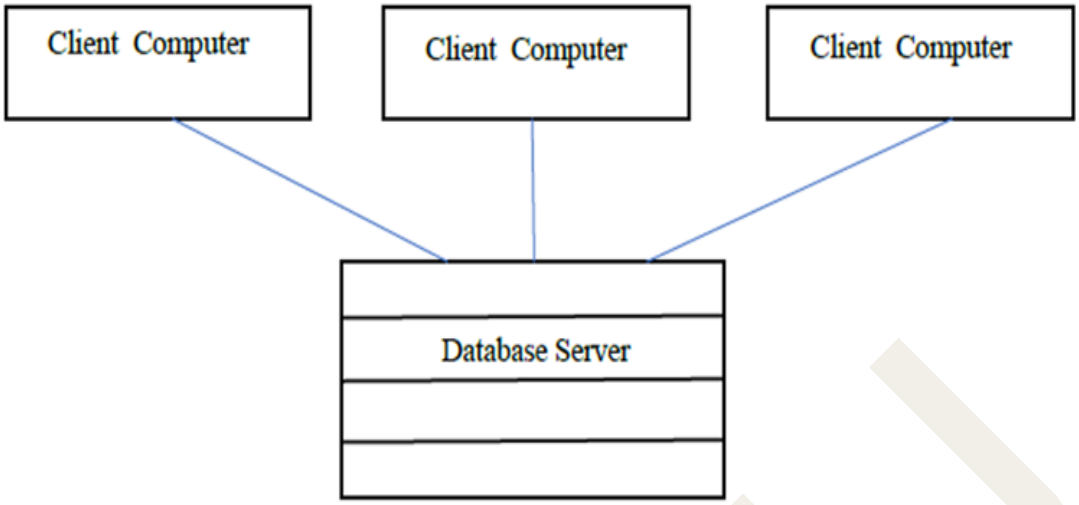


Fig. 1.2.2 Two tier architecture

1.2.1.3 Three tier architecture:

The three-tier architecture is one of the most commonly used architectures in database systems, especially for web-based applications. It consists of three tiers: the client tier, business logic tier, and database tier. Each tier is independent of the others, with clear separation based on the complexity of user interactions and the way data is used and managed. This separation enhances security, scalability, and maintainability of the system.

1. Database Tier (Base Layer)

The database tier is the lowest layer, responsible for managing the actual storage and retrieval of data. This tier contains the database server and is accessed through a query processing language such as SQL. It stores and organizes all data, enforces security, and manages transactions and data integrity.

In an e-commerce application, the database tier would store data related to products, customers, orders, and payments. The database server is unaware of the users directly accessing the system because all communication passes through the application tier.

2. Business Logic Tier (Middle Layer)

The business logic tier, also known as the application tier or middle tier, serves as a mediator between the client tier and the database tier. It processes business rules, logic, and operations required by the application. This tier handles complex tasks such as data validation, security checks, business rules enforcement, and interaction with external services. It presents an abstract view of the database to users by hiding the internal structure of data storage.

When a user places an order in an e-commerce application, the business logic tier verifies product availability, calculates totals, processes payment, and updates the database accordingly. This tier often runs on application servers and provides APIs or services that the client tier can interact with.

3. Client Tier (Presentation Layer)

The client tier is the topmost layer where users interact with the system. This tier is responsible for presenting data to users and collecting their input. It could be a web browser, mobile app, or desktop application. Users operate at this level without needing any knowledge of how the data is stored or processed internally. The client tier sends requests to the business logic tier, which then communicates with the database to fetch or manipulate data.

A user browsing products on an e-commerce website interacts with the client tier, which displays product details and allows the user to search, filter, and place orders.

How the Tiers Communicate

1. The client tier sends a request (e.g., search for a product) to the business logic tier.
2. The business logic tier processes the request, applies business rules, and interacts with the database tier to retrieve or update data.
3. The database tier performs the requested operations and sends the results back to the business logic tier, which formats the response and returns it to the client tier for display.

Advantages of Three-Tier Architecture

1. **Scalability:** Each tier can be scaled independently based on system load. Additional application servers can be added to handle more business logic requests without modifying the client or database tiers.
2. **Security:** The business logic tier acts as a gatekeeper, preventing unauthorized access to the database. Users only interact with the data through controlled interfaces.
3. **Maintainability:** Changes to one tier do not require changes to other tiers. The database structure can be modified without affecting the client interface.
4. **Reusability:** The business logic tier can serve multiple client applications, such as mobile apps, web browsers, and desktop applications, without duplicating code.

Example of Three-Tier Architecture in Action

In a university management system:

Client Tier: Students and faculty access the system through a web portal to view schedules, grades, and course details.

Business Logic Tier: The application server validates user credentials, applies access control, and processes data requests.

Database Tier: The database server stores all academic data, including student records, course information, and grades.

By dividing responsibilities across these tiers, the system becomes more organized, secure, and efficient. Fig. 1.2.3 shows three tier architecture.

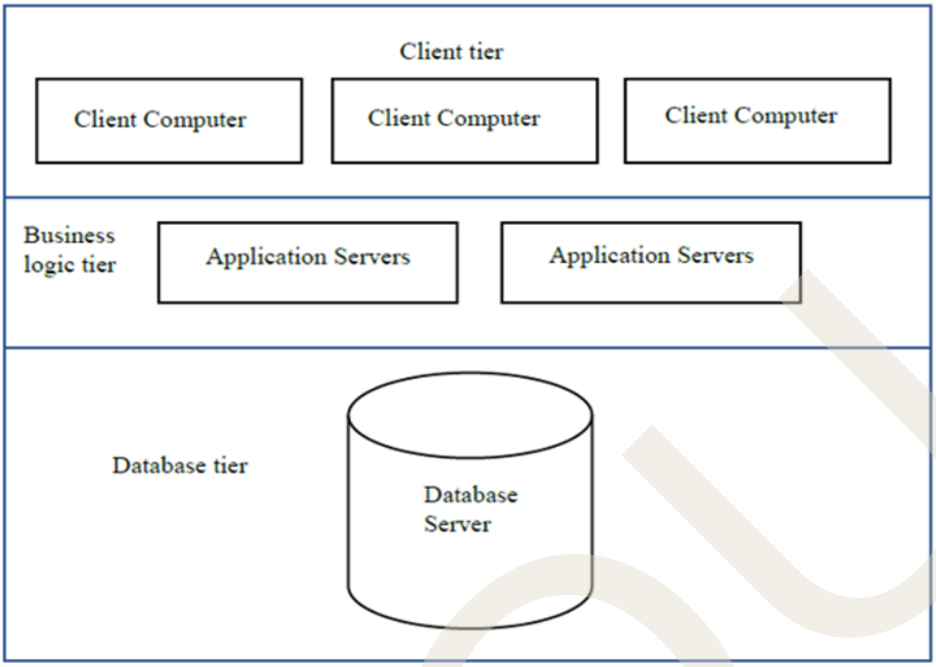


Fig. 1.2.3 Three tier architecture

1.2.4 Generalized Architecture of DBMS

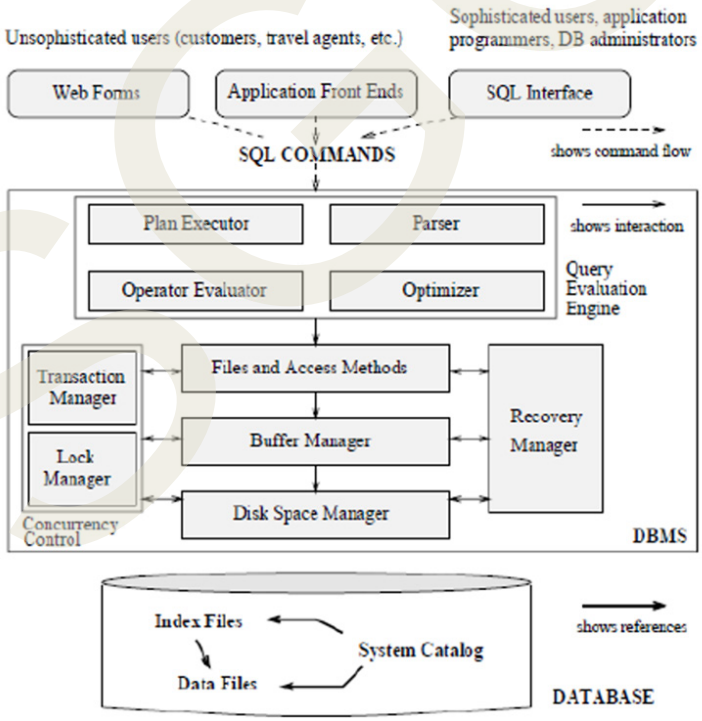


Fig. 1.2.4 General Architecture of Database Management System

(Figure Source: Database Management Systems - Raghu Ramakrishnan and Johannes Gehrke, Third Edition, McGraw Hill, 2003)

1.2.2.1 Types of DBMS users

A database management system caters to different categories of users, each with varying levels of technical expertise and roles in data access and management. These user types include unsophisticated users, sophisticated users, application programmers, and database administrators (DBAs).

1. Unsophisticated Users (End-Users)

Unsophisticated users are those who interact with the database through simple, user-friendly interfaces such as web forms, mobile apps, or application front ends. They do not require knowledge of the underlying database structure or how queries are executed. Their tasks often involve basic operations like searching, entering, or updating data.

In an e-commerce platform, customers browsing products or placing orders are unsophisticated users. They use the interface to view products without needing to know how the database stores or retrieves product information.

Key Characteristics:

- Use pre-built applications or forms.
- Have no direct access to SQL or database queries.
- Focus on high-level tasks like data entry, search, and reporting.

2. Sophisticated Users

Sophisticated users have a deeper understanding of the database system and may interact with it directly using SQL or other query languages. These users typically perform complex data analysis, generate custom reports, or run ad-hoc queries to extract insights from the database.

A data analyst in a university may run SQL queries to generate reports on student performance or enrollment trends.

Key Characteristics:

- Use direct query interfaces like SQL.
- Perform complex data retrieval and analysis.
- Require an understanding of database schemas and query optimization.

3. Application Programmers

Application programmers develop and maintain software applications that interact with the database. They write application code that includes SQL queries to retrieve, insert, update, or delete data. Programmers are responsible for implementing business logic and ensuring that applications communicate with the database effectively.

A programmer working on a banking application may write code to handle customer transactions, account updates, and loan processing.

Key Characteristics:

- Develop applications that connect to the database.
- Write embedded SQL queries within application code.
- Focus on integrating business logic with database operations.

4. Database Administrators (DBAs)

Database administrators are responsible for the overall management, maintenance, and optimization of the database system. They ensure that the database operates efficiently, remains secure, and can handle concurrent user access without data conflicts or performance degradation. DBAs also perform tasks like backup and recovery, database tuning, user access management, and schema updates.

In a healthcare organization, a DBA might ensure that patient records are stored securely, that backups are performed regularly, and that users have appropriate access permissions.

Key Characteristics:

- Manage database security, backups, and performance tuning.
- Control user access and maintain data integrity.
- Monitor system performance and optimize query execution..

1.2.4.1 Query Evaluation Engine

The Query Evaluation Engine is responsible for processing and executing SQL queries in a relational database system. It consists of several components that work together to generate query results efficiently.

1. Plan Executor

A query plan (or query execution plan) is a series of steps designed to retrieve data from the database. The query execution engine takes this plan as input, processes it, and returns the output for the user's query.

2. Parser

The parser breaks down a SQL statement into its components and organizes them into a data structure that other parts of the system can understand and process. This parsing stage is a critical step in query execution to ensure that the query is correctly interpreted.

3. Operator Evaluator

When a query uses operators like =, <, <=, >, >=, or ≠, the DBMS employs various techniques to evaluate these conditions efficiently.

The most commonly used methods for operator evaluation are:

1.Iteration

This method compares records by sequentially checking each entry in one table against another, similar to a nested loop join. It is simple but can be time-consuming for large datasets.

2. Indexing

If the columns used in the query condition have an index, the DBMS leverages the index to quickly locate matching records, avoiding full table scans. This significantly improves query performance by reducing the number of records that need to be checked.

3. Partitioning

In this method, the database divides a table into partitions or groups, often through techniques like hashing. When a query is executed, the DBMS searches within the relevant partition first, then scans only the records in that block, speeding up the search process.

Optimizer

The query optimizer is a crucial part of a Database Management System (DBMS) that evaluates Structured Query Language (SQL) queries to find the most efficient way to execute them. It creates one or more query plans, which outline possible methods for executing the query, and selects the plan that optimizes performance and resource usage.

Concurrency Control

Concurrency control in a DBMS is the process of managing multiple transactions that occur at the same time to maintain atomicity, isolation, consistency, and serializability. When multiple transactions are executed simultaneously in a random order, it can lead to issues like data conflicts and inconsistency. Effective concurrency control is crucial to prevent these problems and requires timely management.

To implement concurrency control, a transaction manager and lock manager are used to coordinate and control access to shared data resources. These mechanisms ensure that transactions do not interfere with each other, preserving the accuracy and integrity of the database.

Transaction Manager

The transaction manager is a crucial component in a Database Management System (DBMS) responsible for coordinating transactions across one or more resources. It ensures that transactions are executed with atomicity (either all operations succeed or none are applied) and durability (completed transactions remain permanent even after a failure). The transaction manager creates and maintains transaction objects, monitors their progress, and ensures that all transaction properties are met.

In a banking system, a transaction to transfer money between accounts involves updating both the sender's and receiver's balances. The transaction manager ensures that either both updates occur successfully or neither of them is applied, preserving data consistency.

Lock Manager

The lock manager is responsible for implementing concurrency control by managing locking protocols. When multiple transactions attempt to access the same data simultaneously, the lock manager ensures that data access conflicts are handled correctly to prevent issues like dirty reads, lost updates, or deadlocks.

The lock manager maintains a lock table, which is a data structure used to track locks on data items. The lock table stores information about which transactions hold or are waiting for locks on particular data items. Fig. 1.2.4 shows the general architecture of the database management system.

Structure of the Lock Table

1. The lock table is implemented as a hash table, where the names of data items act as hash keys.
2. Each locked data item has a linked list associated with it.
3. Each node in the linked list represents a transaction that has requested a lock on that data item.
4. When a new lock request is made, a node is added to the end of the linked list.

If multiple transactions request a lock on a data item "Account_123," the lock manager creates a linked list for that item. The first transaction receives the lock, while subsequent requests are queued in the list. If no data item is locked, the lock table remains empty.

Consider a scenario where two transactions try to update the same customer's account balance at the same time. The lock manager ensures that one transaction acquires the lock first, completes its operation, and releases the lock before the other transaction can proceed. This prevents conflicts and ensures data consistency.

A DBMS includes several other components to manage different aspects of data handling: File manager, Buffer Manager, Disc Space manager and Recovery Manager are included.

File Manager, Buffer Manager, and Recovery Manager in DBMS

In a Database Management System (DBMS), several components work together to ensure efficient data storage, access, and protection. Among these components are the file manager, buffer manager, and recovery manager, each with distinct roles and responsibilities.

File Manager

The file manager is responsible for managing the hierarchical structure of files and folders within the database system. It provides users and applications with essential functions to organize and manipulate files and directories. These functions include creating, copying, moving, renaming, and deleting folders and files. However, while the file manager can create and manage folders, applications are responsible for creating individual files.

For example, in a university database system, the file manager may organize student data files into folders for different departments (e.g., "Engineering" or "Science"). Within these folders, applications can create files such as transcripts or enrollment records.

Buffer Manager

The buffer manager handles the allocation of memory buffers to optimize data access between main memory and disk storage. When a user or application requests data, the buffer manager checks if the data block is already in the buffer. If the block is unavailable, the buffer manager allocates space to load the block from disk.

To maintain efficient buffer usage, the buffer manager may need to free up space by removing older or less frequently used blocks. This process is known as buffer replacement and typically follows algorithms such as LRU (Least Recently Used) or FIFO (First In, First Out).

In an e-commerce system, the buffer manager may temporarily store product details and user search results in memory, allowing faster access for frequently requested data. If the buffer is full, older search results might be removed to make space for new data.

Recovery Manager

The recovery manager plays a crucial role in ensuring that the database can recover from failures such as hardware crashes, power outages, or data corruption. It performs both backup and recovery operations to protect the database's integrity. The recovery manager stores metadata about these operations in the control file of the target database, which helps track backup history and recovery points.

The database administrator (DBA) has key responsibilities in managing the recovery process, including:

1. **Managing Complexity:** The DBA must plan and implement efficient backup and recovery strategies.
2. **Scalability and Reliability:** Backup processes should handle large amounts of data without failures.
3. **Hardware Utilization:** The DBA ensures optimal use of hardware resources (e.g., storage devices) during backups.
4. **Efficient Recovery:** Recovery time should be proportional to the amount of data being restored, minimizing downtime.

In a banking system, the recovery manager can restore transaction logs to ensure that customer account balances remain accurate after a system crash. Regular backups and transaction logs help reduce data loss and allow for quick recovery.

These managers work together to maintain a stable and efficient database environment. The file manager organizes data storage, the buffer manager optimizes data access, and the recovery manager safeguards data integrity by handling backup and recovery operations.

Recap

- A DBMS has three abstraction levels: external, conceptual, and physical.
- TThe separation of abstraction levels provides data independence and flexibility in data management.
- DBMS architecture can be either single-tier or multi-tier based on system requirements.
- In one-tier architecture, all operations run on a single server, making it simple but less scalable.
- Two-tier architecture connects client applications directly to the database server for faster and simpler communication.
- Three-tier architecture separates the client, business logic, and database tiers for better scalability and security.
- The client tier handles user interaction by collecting input and displaying data.
- The business logic tier processes user requests, validates data, and enforces business rules.
- The database tier manages data storage, retrieval, security, and transaction handling.
- A buffer manager optimizes memory allocation by temporarily storing frequently used data blocks.
- A recovery manager ensures the database can recover from failures by managing backups and transaction logs.
- A lock manager controls access to data by managing locks to prevent data conflicts during simultaneous transactions.

Objective Type Questions

1. Which level deals with how data is physically stored?
2. What type of architecture separates database operations into presentation, application, and database layers?
3. Which architecture combines all data storage, logic, and presentation on a single machine?
4. In a two-tier architecture, which two components communicate directly?
5. What does the client tier handle in a three-tier architecture?
6. Which layer in the three-tier architecture is responsible for business rules and logic?
7. Who is responsible for overall management & maintenance of database?
8. What is the main responsibility of the buffer manager?
9. Which component is responsible for managing backup and recovery operations in a DBMS?
10. What type of users interact with databases through web forms and application front ends?
11. What component processes and executes SQL queries in a relational DBMS?
12. What mechanism is used to control simultaneous access to data by multiple transactions?

Answers to Objective Type Questions

1. Physical
2. Multi-tier
3. One-tier
4. Client-server
5. Presentation
6. Business Logic Tier (Middle Layer)
7. DBA
8. Allocation of memory buffers
9. Recovery Manager
10. Unsophisticated

11. Query Evaluation Engine
12. Concurrency control

Assignments

1. Compare and contrast one-tier, two-tier, and three-tier database architectures in terms of structure, communication flow, advantages, and limitations. Provide relevant examples for each architecture.
2. Discuss the roles of the file manager, buffer manager, and recovery manager in a DBMS. Explain how these components work together to maintain data integrity and optimize system performance.
3. Describe the concept of concurrency control in a DBMS. Explain the importance of ensuring atomicity, isolation, consistency, and serializability in concurrent transactions, along with the roles of the transaction manager and lock manager.
4. Explain the query processing steps in a relational database system. Describe the roles of the parser, optimizer, plan executor, and operator evaluator, and how they contribute to efficient query execution.

Suggested Reading

1. Ramakrishnan, R., & Gehrke, J. (2002). *Database management systems* (3rd ed.). McGraw-Hill.
2. Silberschatz, A., Korth, H. F., & Sudarshan, S. (2011). *Database system concepts* (6th ed.). McGraw-Hill.

Reference

1. https://www.geeksforgeeks.org/dbms
2. https://www.tutorialspoint.com/dbms/dbms_architecture.htm

Digital Transmission Entity-Relationship Model

Learning Outcomes

Upon the completion of the unit, the learner will be able to;

- Define the term Entity-Relationship (ER) Model
- State the purpose of an ER diagram in database design
- Recall the symbols used to represent entities, attributes, and relationships in an ER diagram
- Make learners to draw an ER diagram of different cases

Prerequisites

A model must provide a clear and comprehensive representation of any real-time scenarios to facilitate understanding. In the case of database design, the Entity-Relationship (ER) model plays a crucial role in this process. By utilizing predefined symbols, such as rectangles for entities, ovals for attributes, and diamonds for relationships, the ER model ensures that even complex data structures are easy to comprehend. These visual elements help to establish clarity, making it simpler to identify the relationships between different data components and the constraints governing them. As a result, the ER model becomes an indispensable tool for database designers in mapping real-world problems to database solutions effectively.

Keywords

Semantic data model, Entity-Relationship model, attributes Candidate key, Composite key, super key, primary key, weak entity

Discussion

Designing a database for an application typically involves several key steps. First, we gather requirements—both functional and data-related—by consulting with database users to understand their needs. Next, we create a logical or conceptual design, where the Entity-Relationship (ER) model plays a crucial role as the most widely used graphical representation of a database's conceptual structure. Finally, we move to the physical database design, which includes aspects like indexing for performance optimization, and external design, such as defining views to control data access and presentation.

A high-level model in DBMS is a conceptual framework that represents data in an abstract and user-friendly way, independent of physical storage details. It focuses on how data is organized and related, making it easier for users to understand and design databases. The Entity-Relationship (ER) model is a common high-level model, using entities, attributes, and relationships to depict real-world data. This model helps in the early stages of database design, ensuring a clear structure before moving to the logical and physical levels of implementation.

1.3.1 Entity-Relationship (ER) model

The Entity-Relationship (ER) model is a high-level conceptual framework used in database design to visually represent the structure of data and its relationships. Proposed by Peter Chen in 1976, the ER model helps designers and developers understand and organize data before implementing it in a database. It consists of entities (real-world objects like students, employees, or products), attributes (characteristics of entities like name, age, or ID), and relationships (connections between entities like "enrolled in" or "works for"). The model is typically represented using ER diagrams, which use symbols such as rectangles for entities, ovals for attributes, and diamonds for relationships. The ER model plays a crucial role in designing relational databases by ensuring data is properly structured, reducing redundancy, and enhancing efficiency.

1.3.1.1 Why Use ER Diagrams In DBMS?

- ER diagrams represent the E-R model in a database, making them easy to convert into relations (tables).
- ER diagrams provide the purpose of real-world modeling of objects which makes them intently useful.
- ER diagrams require no technical knowledge of the underlying DBMS used.
- It gives a standard solution for visualizing the data logically.

1.3.1.2 Symbols Used in ER Model

ER Model is used to model the logical view of the system from a data perspective which consists of these symbols it is shown in fig 1.3.1

- Rectangles: Rectangles represent Entities in the ER Model.
- Ellipses: Ellipses represent Attributes in the ER Model.
- Diamond: Diamonds represent Relationships among Entities.
- Lines: Lines represent attributes to entities and entity sets with other relationship types.
- Double Ellipse: Double Ellipses represent Multi-Valued Attributes.
- Double Rectangle: Double Rectangle represents a Weak Entity.
- Double Diamonds → Represent Weak Relationships.

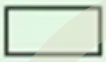
Figures	Symbols	Represents
Rectangle		Entities in ER Model
Ellipse		Attributes in ER Model
Diamond		Relationships among Entities
Line		Attributes to Entities and Entity Sets with Other Relationship Types
Double Ellipse		Multi-Valued Attributes
Double Rectangle		Weak Entity

Fig 1.3.1 Symbols in ER model

1.3.2 Components of ER Diagram

ER diagram used to show the relationship among entity sets. ER Diagram is able to represent the complete logical structure of a database. ER Diagram can be easily converted to schema.

ER model includes shaw in fig 1.3.2:

1. Entity and entity set.
2. Attributes and type of attributes.
3. Keys
4. Relationships among entities.

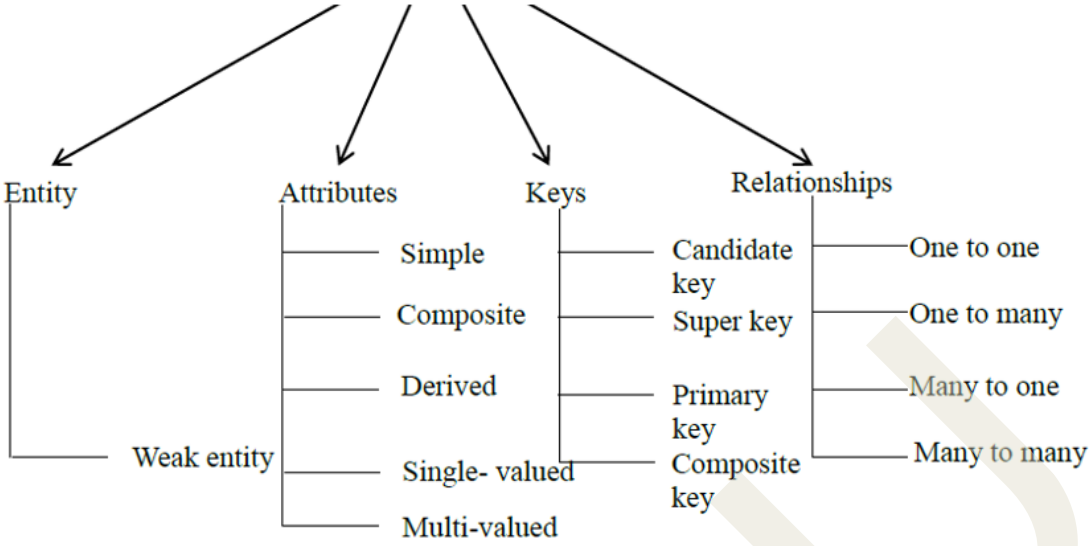


Fig 1.3.2 Components of ER Model

1.3.2.1 Entity and Entity Set

An entity represent real-world objects that have a distinct identity. They can be **physical** (e.g., a student, an employee, a car) or **conceptual** (e.g., a job position, an order).

An entity is an object of Entity Type and a set of all entities is called an entity set. For Example, E1 is an entity having Entity Type Student and the set of all students is called Entity Set. In ER diagram, Entity Type is represented as in fig 1.3.3 :

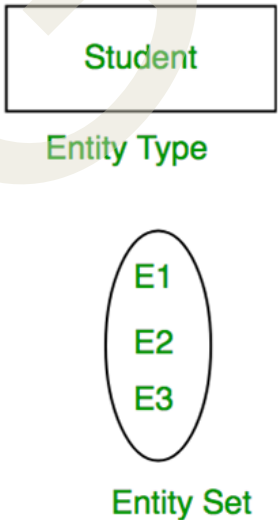


Fig 1.3.3 Entity Set

We can represent the entity set in ER Diagram but can't represent entity in ER Diagram because entity is row and column in the relation and ER Diagram is graphical representation of data.

Types of Entities

- **Strong Entity:** An entity that has a **primary key** and can exist independently. Student is a strong entity because it can exist independently.
- **Weak Entity:** An entity that **depends** on a strong entity and does not have a primary key of its own. It relies on a **foreign key** (from the related strong entity) and has a **partial key** (discriminator) to differentiate its instances. Dependent (child of an employee) is a weak entity because it exists only if the Employee entity exists.

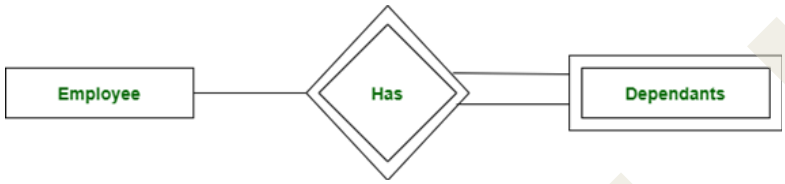


Fig 1.3.4 Strong and Weak Entity

1.3.2.2 Attributes in the ER Model

In the Entity-Relationship (ER) Model, attributes define the characteristics or properties of an entity or relationship. Attributes provide meaningful information about entities and help in uniquely identifying them. Fig 1.3.5 shows attribute symbol,

Fig 1.3.5 Attribute symbol

Types of Attributes in the ER Model

a. Simple attribute:

Simple attributes are that have atomic values, which cannot be split up further. An example for it is stud_id in student entity. Fig 1.3.6 shows example of Simple attribute

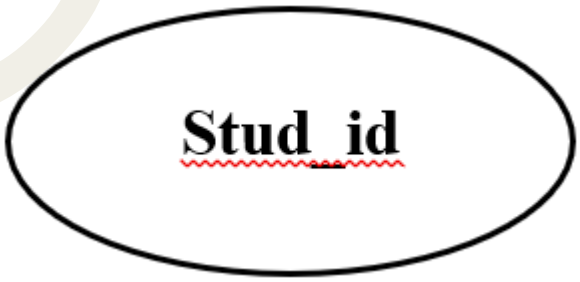


Fig 1.3.6 Simple attribute

b. Composite attribute:

Composite attributes have non-atomic values made up of more than one simple attribute. Consider address as an example, it contains house_name, city_name, pincode etc.

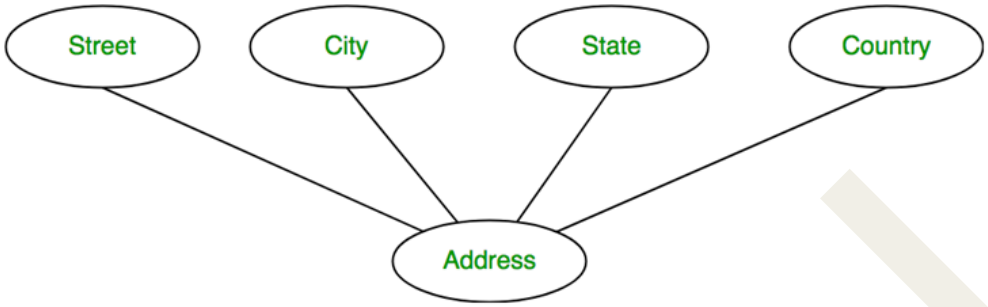


Fig 1.3.7 composite attribute

c. Derived attribute:

Derived attributes are not directly present in the database management system; it can be derived using other attributes. For example age is a derived attribute which can be computed from date_of_birth attribute it is shown in fig 1.3.8.

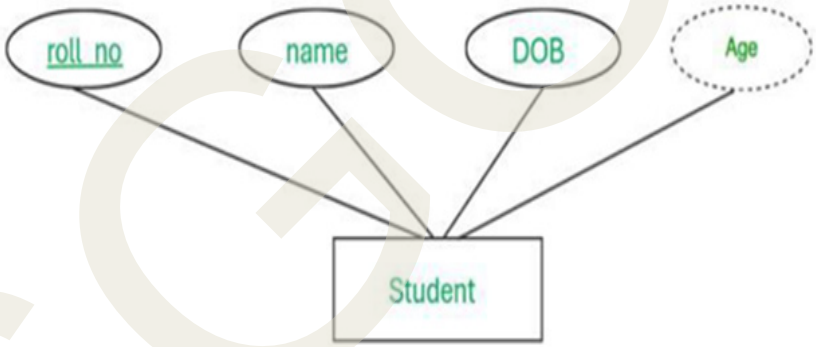


Fig 1.3.8 Derived attribute

d. Single-valued attribute:

Single-valued attributes have single value, example stud_id in student entity.

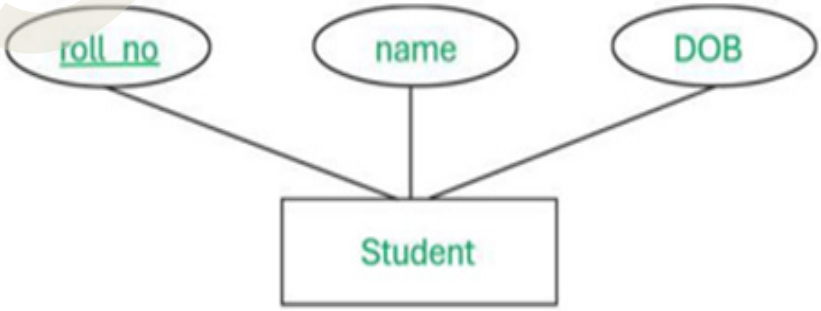


Fig 1.3.9 Single valued attribute

e. Multi-valued attribute:

Multi-valued attributes that can hold multiple values. For example phone_number is a multi-valued attribute because a student can have multiple phone numbers. It is represented in the ER Diagram using double ovals.

Fig 1.3.10 Multivalued attribute

The Complete Entity Type Student with its Attributes can be represented as:

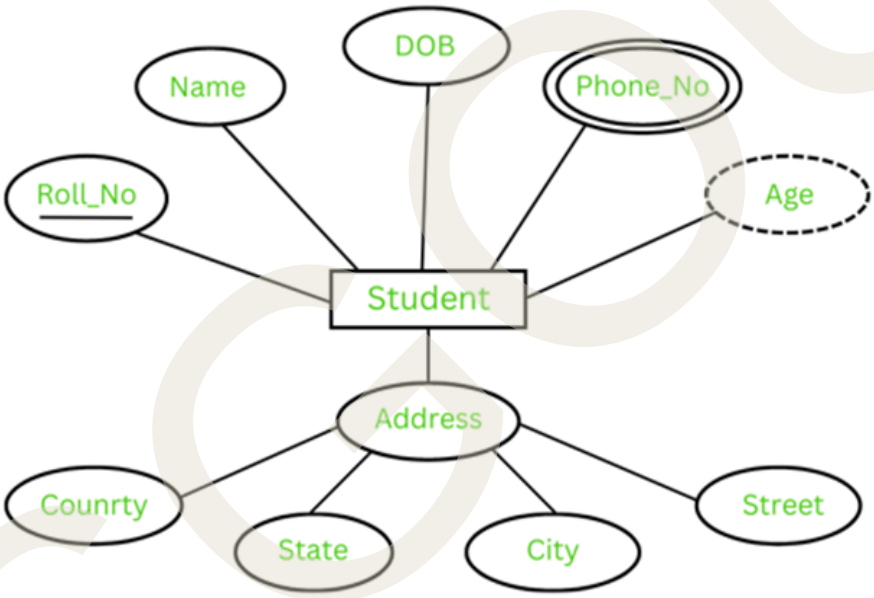


Fig 1.3.11 Entity type Student with its attributes

1.3.2.3 Keys

In a database, keys are attributes or sets of attributes that help uniquely identify records in a table and establish relationships between tables. We require keys in a DBMS to ensure that data is organized, accurate, and easily accessible. Keys help to uniquely identify records in a table, which prevents duplication and ensures data integrity. Keys also establish relationships between different tables, allowing for efficient querying and management of data. Without keys, it would be difficult to retrieve or update specific records, and the database could become inconsistent or unreliable.

Different keys are:

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

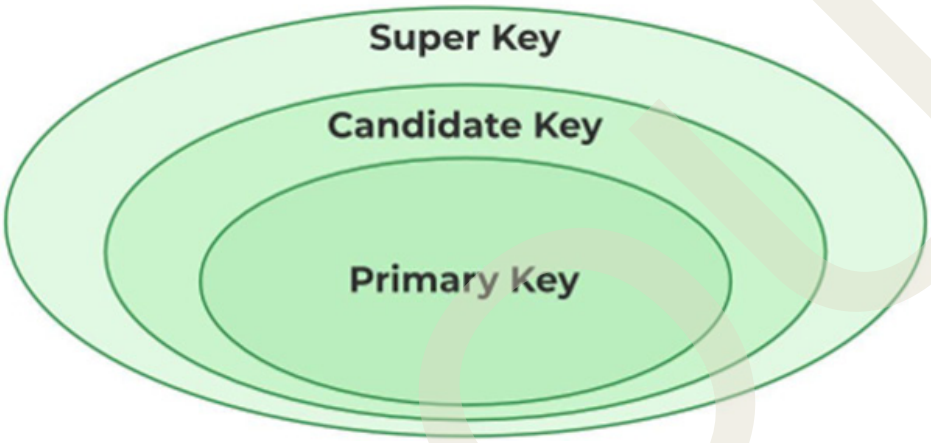


Fig 1.3.12 Keys hierarchy in ER model

Consider an example of a university entity with attributes univ_id and univ_name, both univ_id and univ_name can act as a Candidate key for the table as they contain unique and non-null values.

a. Candidate key

The minimal set of attributes that can uniquely identify a tuple is known as a candidate key. A table can have multiple candidate keys.

Stud_id	Sname	Address	Phno

The student has three attributes: Stud_id, Sname, Address and Phno. Here stud_id can be considered as a candidate key. stud_id attributes can uniquely identify the tuples.

b. Super key

A **superset of a candidate key** that uniquely identifies records. It is a single key or a group of multiple keys that can uniquely identify tuples in a table. It supports NULL values in rows. A **candidate key** is a minimal super key, meaning it has no unnecessary attributes.

Stud_id	Sname	Address	Phno

For example (Stud_id, Sname), (Stud_id, Address), (Stud_id, Phno), (Stud_id, Sname, Address), (Stud_id, Sname, Phno), (Stud_id, Address, Phno), (Stud_id, Sname,

Address, Phno) can be considered as super keys as they all uniquely identify the tuples of the table. Because of the presence of the stud_id attributes which are able to uniquely identify the tuples.The other attributes in the keys are unnecessary, but as a combination it can uniquely identify tuples.

c. Primary key

There can be more than one candidate key in relation out of which one can be chosen as the primary key that uniquely identifies records. This candidate key is called the primary key. A unique identifier for each record in a table. Cannot have duplicate or NULL values.

Stud_id	Sname	Address	Phno

Here we can choose Studid as primary key, it can identify each tuple uniquely.

d. Composite key

In some cases, there is no single attribute that can uniquely identify each tuple. In such cases, more than one attribute is taken as the primary key; this is called a composite key.

Sname	Address	Phno

In this example, Sname alone cannot identify each tuple. A combination of any two or all three attributes can uniquely identify each row.

e. Alternate Key

An alternate key is any candidate key in a table that is not chosen as the primary key. In other words, all the keys that are not selected as the primary key are considered alternate keys. An alternate key is also referred to as a secondary key because it can uniquely identify records in a table, just like the primary key.

Stud_id	Sname	Address	Phno

Here Sname, Address and Phno is an alternate key.

f. Foreign Key

A foreign key is an attribute in one table that refers to the primary key in another table. The table that contains the foreign key is called the referencing table, and the table that is referenced is called the referenced table.

Student

Stud_id	Sname	Address	Phno

Course

Stud_id	Teacher's Name	Course Name

Stud _id is a primary key of the table Student, and foreign key of the table Course.

Here Course is a referencing table and Student is a referenced table.

1.3.2.4 Relationship

It is used for showing the relation between entities. Diamond is used to represent relationships in an ER Diagram. A relationship set can be thought of as a set of n-tuples: Each n-tuple denotes a relationship involving n entities e1 through en, where entity ei is in the entity set Ei. For example in fig 1.3.13, 'Enrolled in' is a relationship type that exists between entity type Student and Course. In the ER diagram, the relationship type is represented by a diamond and connects the entities with lines.

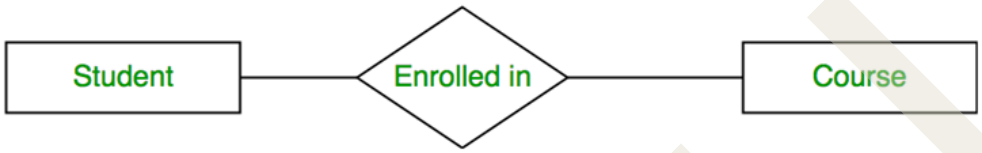


Fig 1.3.13 Relationship in ER Model

A set of relationships of the same type is known as a relationship set. The following relationship set depicts S1 as enrolled in C2, S2 as enrolled in C1, and S3 as registered in C3.

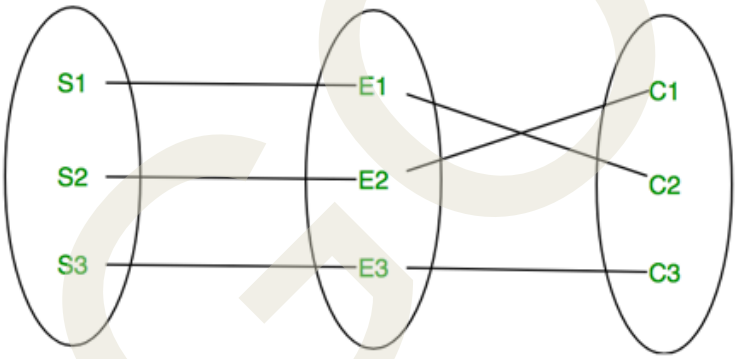


Fig 1.3.14 Relationship set

Degree of a Relationship Set

The number of different entity sets participating in a relationship set is called the degree of a relationship set.

1. Unary Relationship: When there is only ONE entity set participating in a relation, the relationship is called a unary relationship. For example in fig 1.3.15, one person is married to only one person.

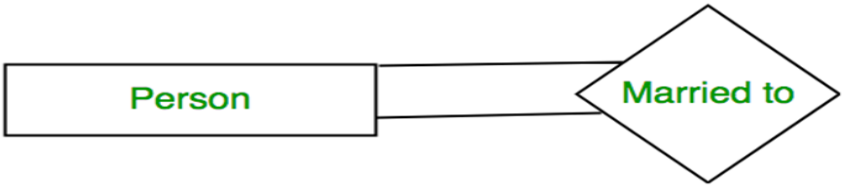


Fig 1.3.15 Unary Relationship

2. Binary Relationship: When there are TWO entities participating in a relationship,

the relationship is called a binary relationship. For example in fig 1.3.16, a Student is enrolled in a Course.

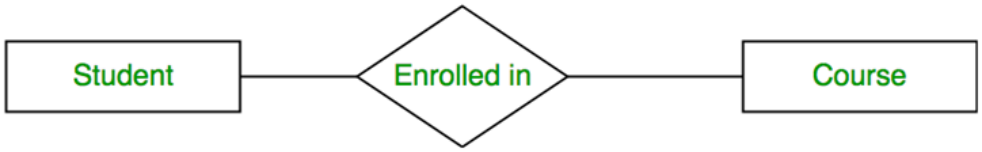


Fig 1.3.16 Binary Relationship

3. Ternary Relationship: When there are three entity sets participating in a relationship, the relationship is called a ternary relationship. In a data model for an educational institution as shown in fig 1.3.17, a single relationship called "teaches" can link the entities Subject, Teacher, and Student. This relationship helps determine which teachers teach each subject and which students are enrolled in it.

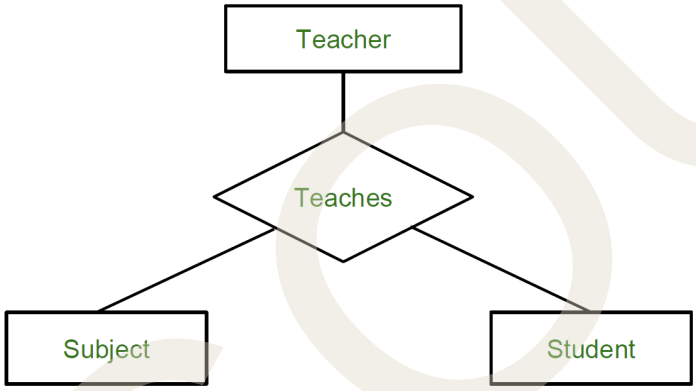


Fig 1.3.17 Ternary Relationship

4. N-ary Relationship: When there are n entities set participating in a relationship, the relationship is called an n-ary relationship.

Cardinality

The number of times an entity of an entity set participates in a relationship set is known as cardinality . Based on Cardinality relationships can be of different types:

- One to one
- One to many
- Many to one
- Many to many

a. One to one:

A student has only one birth_certificate and a birth_certificate is given to one student. the student to bith_certificate relationship is one to one.

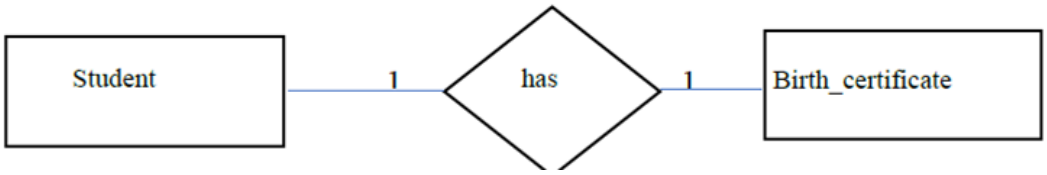


Fig 1.3.18 One to One relationship

b. One to many:

A student can borrow many books but a book can be borrowed by only one student. This student to book relationship is one to many.

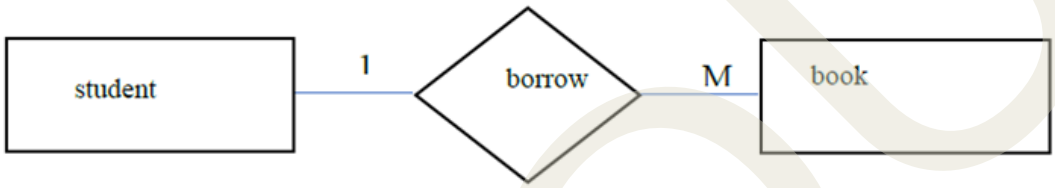


Fig 1.3.19 One to many relationship

c. Many to one:

Many students can enrol in a single department. But a single student cannot enroll in multiple departments at the same time

Fig 1.3.20 Many to one relationship

Many to many:

Many students can teach by many faculties and many faculties can teach many students.

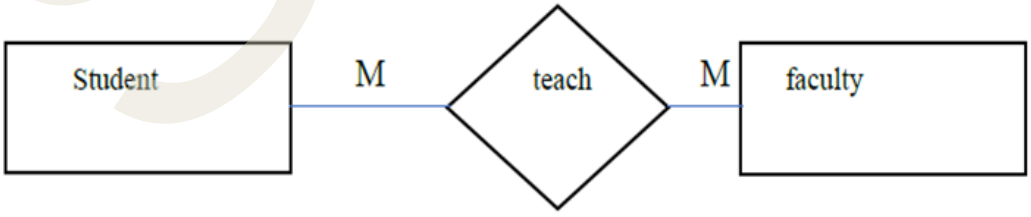


Fig 1.3.21 Many to many relationship

1.3.2.5 Generalization and Specialization

Generalization: Combining two or more entities into a higher-level entity. Example Car and trucks can be generalized into vehicles.

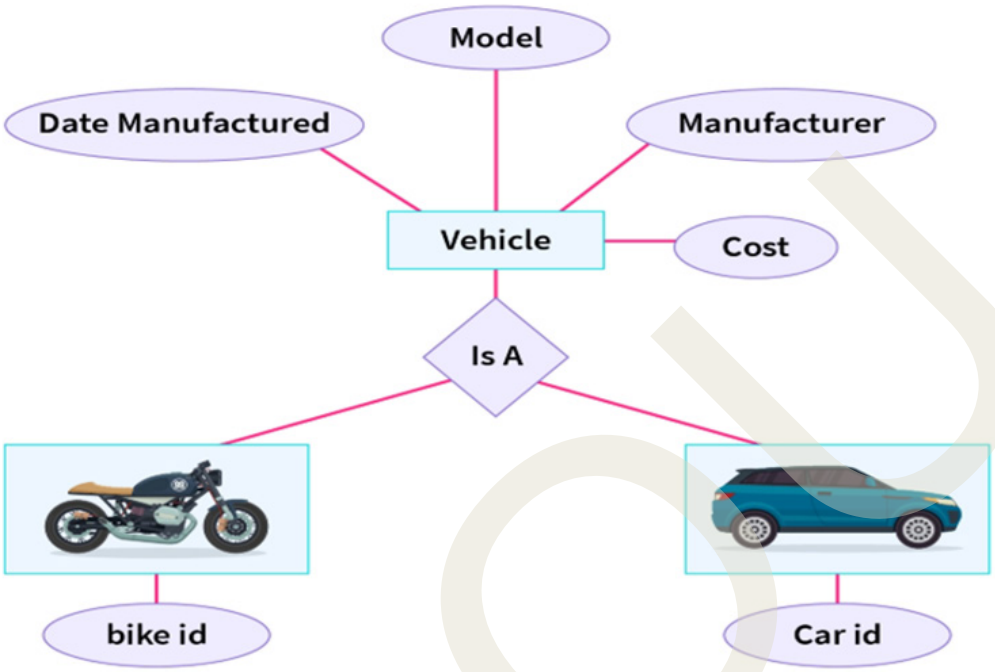


Fig 1.3.22 Generalization of Vehicle entity

Specialization: Creating sub-entities from a higher-level entity. Example Employee can be specialized into Faculty and admin_staff).

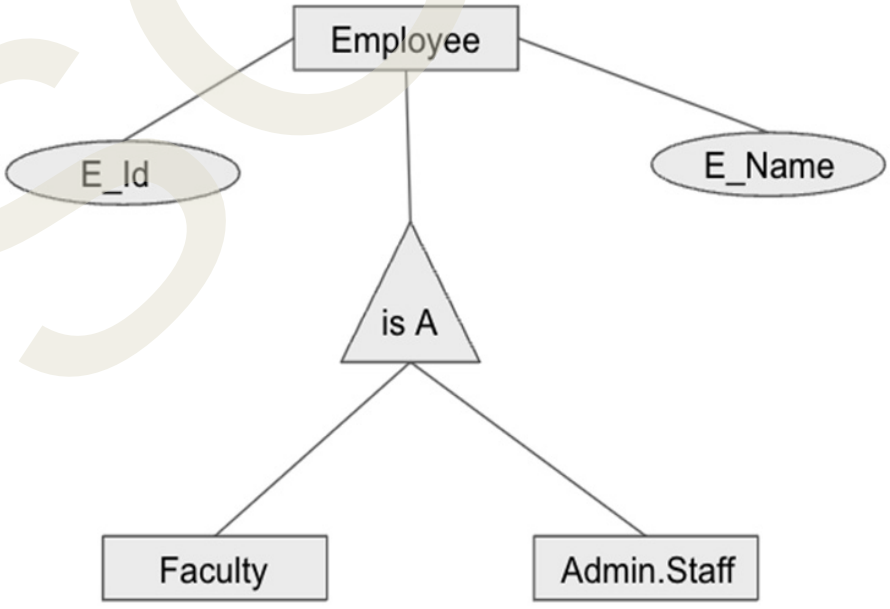


Fig 1.3.23 Specialization of Employee entity

1.3.2.6 Aggregation

Aggregation is an abstract relationship where a relationship itself is treated as an entity. It is useful when one relationship depends on another relationship. A Loan entity is related to a Customer and a Bank, but the relationship between Bank and Loan can be further aggregated to represent a Loan Approval process.

1.3.2.7 Participation Constraint

Participation Constraint is applied to the entity participating in the relationship set.

1. Total Participation : Each entity in the entity set must participate in the relationship. If each student must enroll in a course, the participation of students will be total. Total participation is shown by a double line in the ER diagram.

Fig 1.3.24 Total Participation and Partial Participation

2. Partial Participation: The entity in the entity set may or may NOT participate in the relationship. If some courses are not enrolled by any of the students, the participation in the course will be partial.

The diagram depicts the 'Enrolled in' relationship set with Student Entity set having total participation and Course Entity set having partial participation.

Using Set, it can be represented as,

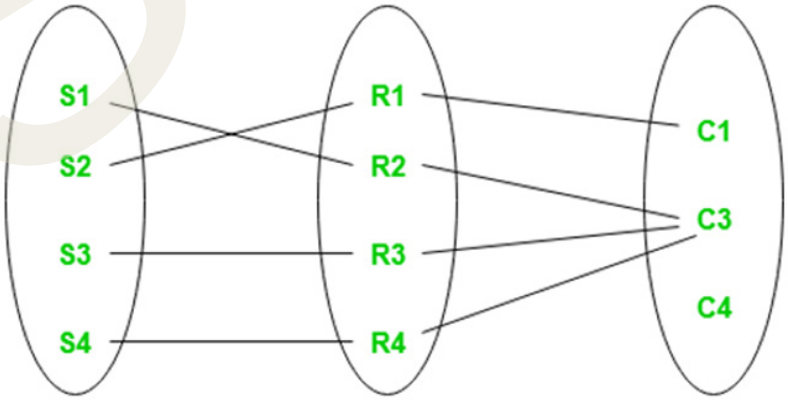


Fig 1.3.26 Set representation of Total Participation and Partial Participation

Every student in the Student Entity set participates in a relationship but there exists a course C4 that is not taking part in the relationship.

1.3.3 Example of ER Model

Let us discuss the ER Model with the example of the University Academic Management System.

The system includes:

- Student information
- Faculty information
- University
- Classes
- Subject

In this student, faculty, classes and subjects are entities. A real world object is termed as an entity in a DBMS. An entity set is a group of similar entities and these entities can have attributes.

A student is an entity or a real world object with properties stud_id, stud_name and age. These properties are called attributes of a student entity. Similarly universities have attributes univ_id, univ_name etc.

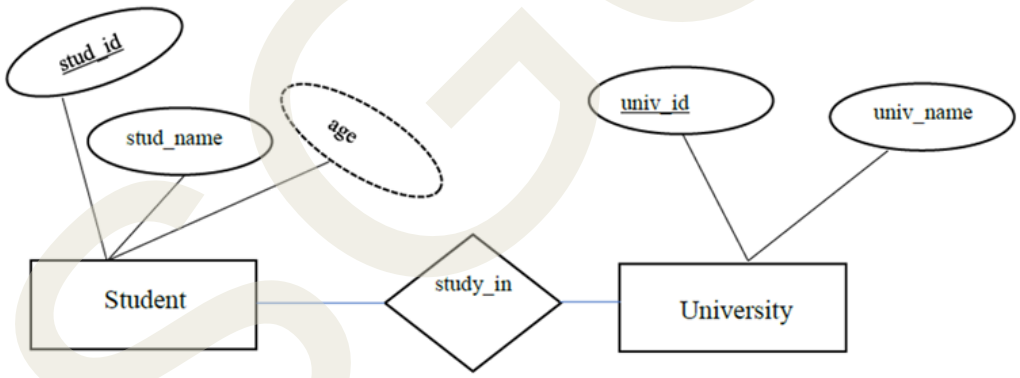


Fig 1.3.27 Sample ER diagram with two entities: student and university.

Example : ER model for Hospital Management System

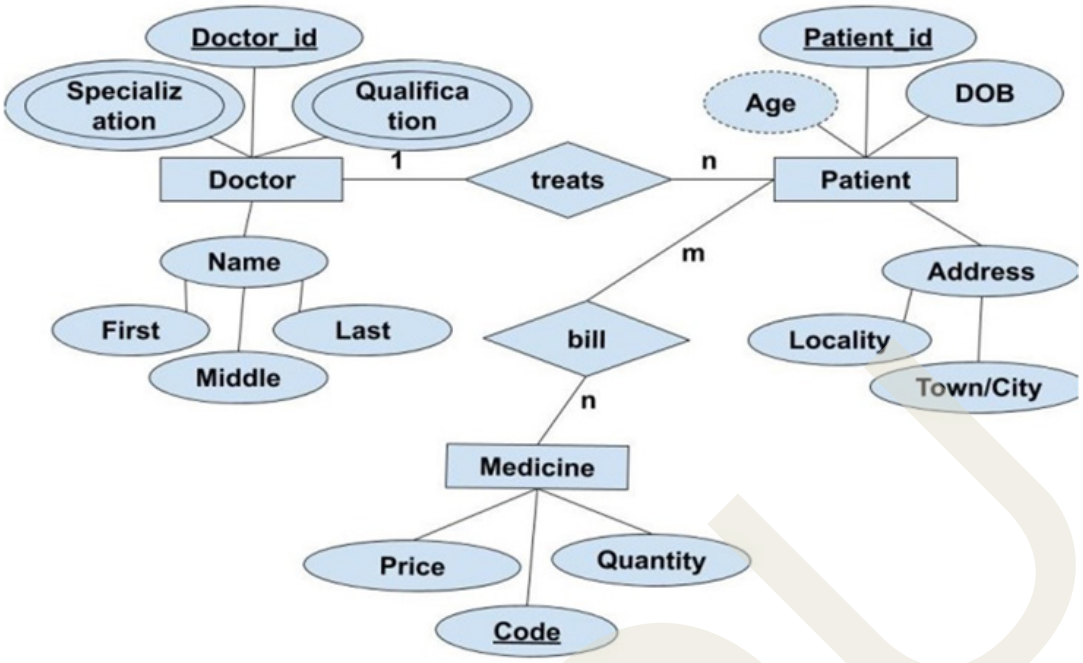


Fig 1.3.28 ER Model for Hospital Management System

Entities and Attributes

1. Doctor
    - Doctor_ID (Primary Key)
    - Name (First, Middle, Last)
    - Specialization
    - Qualification
2. Patient
    - Patient_ID (Primary Key)
    - Name
    - Age
    - DOB
    - Address (Locality, Town/City)
3. Medicine
    - Code (Primary Key)
    - Price
    - Quantity
4. Bill

- Bill_ID (Primary Key)

Relationships

1. Treats (1:N relationship between Doctor and Patient)
    - A doctor can treat multiple patients, but each patient is treated by only one doctor.
2. Bill (M:N relationship between Patient and Medicine)
    - A patient can have multiple medicines in a bill, and a medicine can be billed for multiple patients.

1.3.4 Advantages of the ER Model

1. **Simple and Clear Representation:** ER diagrams provide an intuitive and visual way to represent data and relationships, making it easier to understand.
2. **Efficient Database Design:** Helps in structuring and organizing data efficiently before implementing the database.
3. **Flexibility:** Can be easily modified and extended to accommodate changes in requirements.
4. **Better Communication:** Acts as a bridge between database designers, developers, and stakeholders by providing a clear visual representation.
5. **Standardized Notation:** Uses well-defined symbols (entities, attributes, relationships) for clarity.
6. **Logical and Conceptual Design:** Helps in defining business rules and constraints before moving to physical database design.

1.3.5 Disadvantages of the ER Model

1. Limited Representation of Complex Data – ER models struggle with representing complex data types, inheritance, and advanced constraints.
2. Not Suitable for Small Databases – Can be overkill for simple databases with fewer relationships.
3. Lack of Implementation Details – Does not specify how data will be stored physically, requiring additional steps for database implementation.
4. Challenging for Large Systems – As complexity increases, ER diagrams can become too large and difficult to manage.
5. Normalization Issues – Sometimes, relationships may need further refinement using normalization techniques to avoid redundancy.
6. Ambiguity in Many-to-Many Relationships – Requires additional resolution (e.g., junction tables) to implement many-to-many relationships properly.

Recap

- Introduction to ER Model
- Symbols used in ER Diagram.
- Components of ER model
- Entity and Entity set
- The Concept of –strong entity ,weak entity.
- Different types of attributes.
- Candidate key, Super key, Primary Key and Composite Key.
- Generalisation, Specialisation and aggregation
- Relationship types one to one, one to many, many to one and many to many.
    - Degree of a relation
    - Cardinality of a relation

Objective Type Questions

1. The relationship between entities in a data model can be represented using which Model?
2. What is the term used for real world objects in the ER model?
3. Which is the term used for an entity that can only exist when owned by another entity?
4. Which shape is used for denoting entities?
5. Which shape is used for denoting attributes?
6. Which of the following is a key attribute?
7. Which shape is used for denoting relationships?
8. Which shape is used for denoting links?
9. Which type of attribute can have multiple values?
10. Which shape is used for denoting multi-valued attributes?
11. Which shape is used for denoting derived attributes?

12. Generalization in an ER model represents:

13. Which shape is used for denoting total participation?

14. Which shape is used for denoting a weak entity set?

15. How can we represent a Primary key attribute?

Answers to Objective Type Questions

1. ER Model.
2. Entity
3. Weak entity
4. Rectangle
5. Ellipse
6. An attribute that uniquely identifies an entity
7. Diamond
8. Lines
9. Multi-valued attribute
10. Double ellipse
11. Dashed ellipse
12. The process of combining multiple entities into a higher-level entity
13. Double lines
14. Double rectangle
15. Under line

Assignments

1. Explain the components of an ER model with examples. Include entities, attributes, relationships, and constraints.
2. Describe different types of attributes in an ER model. Provide examples of simple, composite, derived, and multi-valued attributes.
3. Differentiate between strong and weak entities. Explain how weak entities are represented in an ER diagram with an example.
4. Discuss the types of relationships (one-to-one, one-to-many, many-to-many) in the ER model. Give a real-world example for each.
5. Explain the concepts of generalization, specialization, and aggregation in ER modeling with examples.
6. Draw an ER diagram for a University database that includes entities such as Students, Courses, Professors, and Departments. Indicate primary keys and relationships.
7. Convert the following scenario into an ER diagram: A hospital has doctors, patients, and nurses. Each patient is treated by one doctor, and a doctor can treat multiple patients. Nurses assist doctors, and each nurse is assigned to one doctor.
8. Identify the entities, attributes, and relationships for an online shopping system that includes Customers, Orders, Products, and Payments. Create an ER diagram for the system.
9. Discuss the advantages and limitations of the ER model. Suggest ways to overcome its limitations in real-world database design.
10. Convert an ER model into relational tables. Given an ER diagram for a Library Management System, define the relational schema with primary and foreign keys.

Suggested Reading

1. Ramakrishnan, R., & Gehrke, J. (2014). *Database management systems* (2nd ed.). McGraw-Hill Education.
2. Silberschatz, A., Korth, H. F., & Sudarshan, S. (2011). *Database system concepts* (6th ed.). McGraw-Hill Education.

Reference

1. https://www.geeksforgeeks.org/dbms
2. https://www.tutorialspoint.com/dbms/dbms_architecture.htm

UNIT 4 ER Model to Database Schema

Learning Outcomes

At the conclusion of this unit, the learner will be able to;

- Identify ER diagrams with different types of entity
- To convert ER Model with different types of relationships to schema
- Identify the concept of total participation in ER diagram
- study the methods for conversion of multivalued attributes to schema

Prerequisites

The relational model organizes data into structured tables with clearly defined relationships, ensuring efficiency, accuracy, and scalability. Converting relationships like one-to-one (1-1), one-to-many (1-M), and many-to-many (M-N) into relational tables improves data integrity and accessibility. It enforces Primary Keys (PK) for uniqueness and Foreign Keys (FK) for referential integrity, preventing orphaned records. Normalization reduces redundancy and optimizes storage. The model is flexible and scalable, allowing easy modifications. SQL queries efficiently retrieve data using JOIN operations, and ACID properties ensure reliable transactions. It also offers security through user access control, making database management efficient and secure.

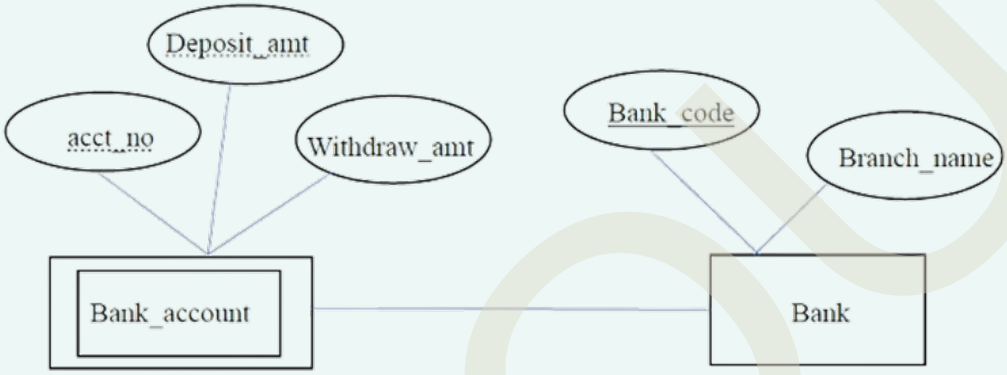
Entity types and relationships play a crucial role in schema design. There are two main types of entities: strong entities and weak entities.

A strong entity is independent and does not rely on any other entity for its existence. It has a primary key attribute that uniquely identifies each record. Strong entities are represented by a single rectangle in an Entity-Relationship (ER) diagram. Examples of strong entities include students, faculty, and courses, as they can exist without depending on any other entity. For instance, a Student entity with attributes like stud_id, stud_name, login, age, and gpa is a strong entity because stud_id serves as the primary key, ensuring uniqueness.

In contrast, a weak entity cannot exist independently and relies on a strong entity for identification. A weak entity lacks a sufficiently unique attribute to serve as a primary key and instead uses a partial key (discriminator) to differentiate records within a group.

Weak entities are represented by a double rectangle in an ER diagram. For example, a bank account entity is a weak entity because it cannot exist without a corresponding bank entity. In this case, the bank's IFSC code is required for the account number to have meaning, making acc_no the discriminator or partial key.

When a strong entity and a weak entity are related, their relationship is called an identifying relationship, which is represented by a double diamond in ER diagrams. This relationship ensures that the weak entity is uniquely identified in the database with the help of the strong entity it depends on.



The given ER diagram represents a banking system with two entities: Bank (Strong Entity) and Bank_Account (Weak Entity). The Bank entity has attributes Bank_code (Primary Key) and Branch_name, while the Bank_Account entity includes acct_no (Partial Key), Deposit_amt, and Withdraw_amt. Since Bank_Account depends on Bank for identification, their relationship is an Identifying Relationship, where Bank_code serves as a Foreign Key (FK) in Bank_Account. The relational schema consists of two tables: Bank (Bank_code PRIMARY KEY, Branch_name) and Bank_Account (acct_no, Bank_code (FK), Deposit_amt, Withdraw_amt, PRIMARY KEY (acct_no, Bank_code)). This ensures normalization (avoiding redundancy), referential integrity (ensuring accounts are linked to a valid bank), and data consistency, making the database efficient, scalable, and well-structured.

Keywords

Foreign key, arity, cardinality, one to one, one to many, many to many, total participation.

Discussion

1.4.1 Introduction to relational model

The Relational Model represents data and their relationships through a collection of tables. Each table also known as a relation consists of rows and columns. Every column has a unique name and corresponds to a specific attribute, while each row contains a set

of related data values representing a real-world entity or relationship. This model is part of the record-based models which structure data in fixed-format records each belonging to a particular type with a defined set of attributes.

E.F. Codd introduced the Relational Model to organize data as relations or tables. After creating the conceptual design of a database using an ER diagram, this design must be transformed into a relational model which can then be implemented using relational database systems like Oracle SQL or MySQL.

The relational model represents DB in the form of a collection of various relations. This relation refers to a table of various values. And every row present in the table happens to denote some real-world entities or relationships. The names of tables and columns help us interpret the meaning of the values present in every row of the table. This data gets represented in the form of a set of various relations. Thus, in the relational model, basically, this data is stored in the form of tables. However, this data's physical storage is independent of its logical organisation.

Popular Relational Database Management Systems:

- IBM – DB2 and Informix Dynamic Server
- Oracle – Oracle and RDB
- Microsoft – SQL Server and Access
- Properties of a Relational Model

The relational databases consist of the following properties

- Every row is unique
- All of the values present in a column hold the same data type
- Values are atomic
- The columns sequence is not significant
- The rows sequence is not significant
- The name of every column is unique

1.4.2 Key Concepts of the Relational Model

1. 1.4.2.1 Relation (Table)

A **relation** refers to a **table** that organizes data into rows and columns. Each relation represents an entity or a real-world object, and has a name with **tuples (rows)** storing individual records and **attributes (columns)** defining the properties of those records.

Example : Student Relation

Stud_Id	Name	Age	Course
S101	Anu S	16	Computer Science
S102	Daya M	17	Biology
S103	Rahul G	17	Commerce
S104	Surya S	16	Biology
S105	Hari A	18	Computer Science

2. Field

A field is the smallest unit of data in a database. It represents a single data item within a table. In a "Student" table, a field can be “S101”, “Surya S”, “Commerce” are examples of fields.

3. Tuple (Row)

It is a single row of a table that consists of a single record. The relation above consists of five tuples, one of which is like:

S102	Daya M	17	Biology

4. Attribute (Column)

It refers to every column present in a table. The attributes refer to the properties that help us define a relation. Each attribute has a unique name and a specified data type. Above relation contain four attributes, one of which is like:

Stud_Id
S101
S102
S103
S104
S105

5. Domain

A domain refers to the set of valid values that an attribute (column) can take. It defines the permissible data types and constraints for a given attribute.

Student_Id→ Domain: Varchar (Positive Numbers Only)

Name → Domain: Text (String with a max length, e.g., 50 characters)

Age →Domain: Integer (Between 16 and 22 only)

6. Relation Schema

A relation schema defines the structure of the relation and represents the name of the relation with its attributes. e.g. STUDENT (Stud_Id, NAME, Age, Course) is the relation schema for STUDENT. The schema specifies the

- Relation name
- The name of each column or attribute.
- The domain of each field.

7. Relation Instance

The set of tuples of a relation at a particular instance of time is called a relation instance. It can change whenever there is an insertion, deletion or update in the database.

8. Degree

The number of attributes in the relation is known as the degree of the relation. The STUDENT relation defined above has degree 4.

9. Cardinality

The number of tuples in a relation is known as cardinality. The STUDENT relation defined above has cardinality 5.

10. NULL Values

The value which is not known or unavailable is called a NULL value. It is represented by NULL. e.g. PHONE of STUDENT having ROLL_NO 4 is NULL

11. Key

A key is an attribute or a set of attributes that uniquely identifies each tuple in a relation. In a Student table, "Student_ID" can be a primary key.

12. Constraints

Rules enforced to maintain the integrity of data in the database. Primary Key Constraint, Foreign Key Constraint, NOT NULL, UNIQUE, etc.

13. Relational Algebra & Relational Calculus

- Relational Algebra: A procedural query language using operations like SELECT, PROJECT, JOIN, UNION, INTERSECTION, etc.
- Relational Calculus: A declarative query language describing what to retrieve rather than how to retrieve it.

1.4.3 Constraints in Relational Model

While we design a Relational Model, we have to define some conditions that must

hold for the data present in a database. These are known as constraints. One has to check these constraints before performing any operation (like insertion, updating and deletion) in the database. If there occurs any kind of a violation in any of the constraints, the operation will ultimately fail.

1.4.3.1 Domain Constraints

The domain constraints are like attribute level constraints. Now an attribute is only capable of taking values that lie inside the domain range. For example, if a constraint ID_NO>0 is applied on the EMPLOYEE relation, inserting some negative value of ID_NO will result in failure.

1.4.3.2 Key Integrity

Each and every relation present in the database should have at least one set of attributes that uniquely defines a tuple. Those sets of attributes are known as keys. For example, ID_NO in EMPLOYEE is a key. Now, remember that no two students would be capable of having the very same ID number. Thus, a key primarily consists of these two properties:

- It has to be unique for all the available tuples.
- It can not consist of any NULL values.
- Referential Integrity

Whenever one of the attributes of a relation is capable of only taking values from another attribute of the same relation or other relations, it is termed referential integrity.

1.4.4 Relation Schema

A Relational Schema is like a blueprint for a relational database. It defines how data is organized into tables (relations) and how these tables are connected through keys and constraints.

Think of it like an architectural plan for a building—before constructing a house, an architect designs a layout showing rooms, doors, and connections. Similarly, a relational schema defines the structure of the database before storing actual data. e.g. STUDENT (Stud_Id, NAME, Age, Course) is the relation schema for STUDENT. The schema specifies the

1. Relation name
2. The name of each column or attribute.
3. The domain of each field.

1. Relational Name

Name of the table that is stored in the database. It should be unique and related to the data that is stored in the table. For example- The name of the table can be Employee store the data of the employee.

2. The name of each column or attribute.

Attributes specify the name of each column within the table. Each attribute has a specific data type.

3. The domain of each field

The set of possible values for each attribute. It specifies the type of data that can be stored in each column or attribute, such as integer, string, or date.

4. Primary Key

The primary key is the key that uniquely identifies each tuple. It should be unique and not be null.

5. Foreign Key

The foreign key is the key that is used to connect two tables. It refers to the primary key of another table.

6. Constraints

Rules that ensure the integrity and validity of the data. Common constraints include NOT NULL, UNIQUE, CHECK, and DEFAULT.

Example:

Student (stud_id: string, stud_name: string, login: string, age: integer, gpa : real)

Table 1 : Student relation

Stud_id	Stud_name	login	age	gpa
2100	Smith	Smith@cs	20	4.2
2155	Jack	Jack@hindi	21	3.9
2345	Nimal	Nimal@english	20	3.5
2454	Job	Job@history	22	3.8

In the above schema, stud_id, stud_name, login have a domain named string, age has integer and gpa has real.

A relation instance in a database management system (DBMS) is a set of tuples in a relation at a specific moment in time. A relation instance can change if data is added, deleted, or updated in the database.

When accessing records in a Student relation, stud_id =2100, stud_name= smith, login = Smith@cs, age=20, gpa=4.2 which will return a single tuple or row. That is "Smitha, Smith@cs, 20, 4.2" is an example for a single tuple or row in a relational database.

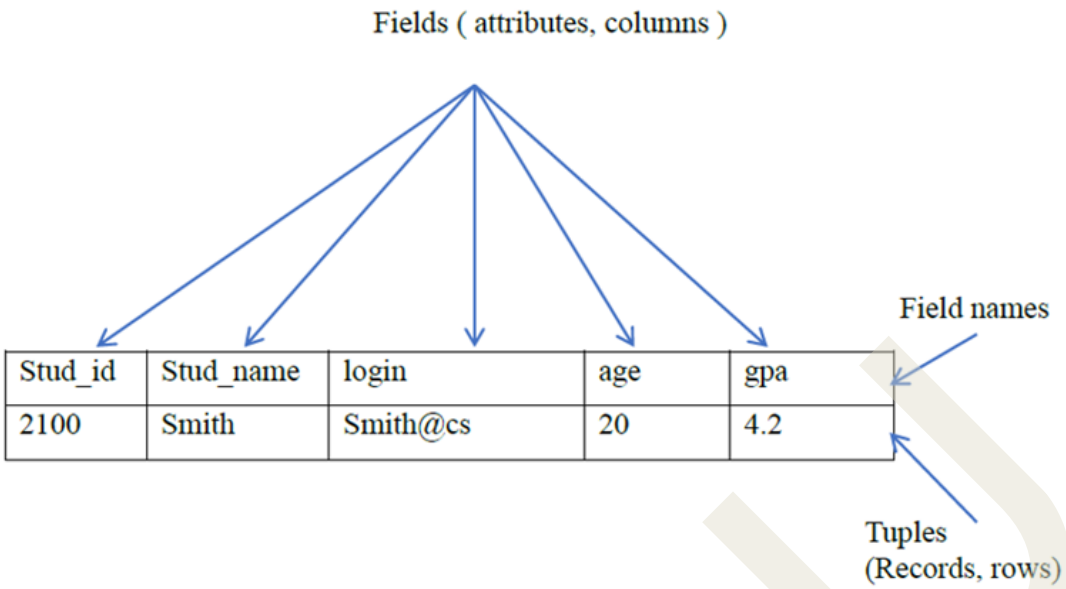


Fig.1.4.1 Fields and Attributes

Degree of a relation: The number of attributes in a relation represents the degree or arity of the relation. Table "Student" has degree five.

Cardinality : The number of tuples or rows represents the cardinality of a relation instance.

Table "Student" has cardinality four.

A relational Database has a collection of relations with distinct relation names. For example university databases have relations named student, faculty, courses, classes etc.

Foreign key constraints

If information stored in a relation is linked to another relation, the integrity constraint used is foreign key.

The Enrolled table

Enrolled (stud_id: string, course_id: string, grade: string)

For ensuring that only registered students are enrolled in courses. The values in the stud_id field of enrolled should match the stud_id field of student relation. The stud_id field of enrolled is called a foreign key and refers to student relation. The foreign key in the Enrolled relation (stud_id in enrolled relation) must match the primary key of the referenced relation (student relation). Name of the attribute can differ and datatypes need to be compatible.

1.4.5 Conversion of strong entity into a relational relation

In the Entity-Relationship (ER) model, a strong entity is an entity that can exist independently in the database and has a primary key that uniquely identifies each of its instances. When converting a strong entity into a relational database table, follow these steps:

Steps for Conversion:

- ♦ Create a Table:
  - Each strong entity is converted into a separate relation (table).
- ♦ Include Attributes as Columns:
  - All attributes of the entity become columns in the table.
  - Multivalued attributes can be ignored
  - Composite attribute in the ER diagram should be divided to simple attributes and included in the table
- ♦ Select a Primary Key:
  - The primary key of the strong entity becomes the primary key of the corresponding table.

Consider this ER diagram shown in figure 1.4.2

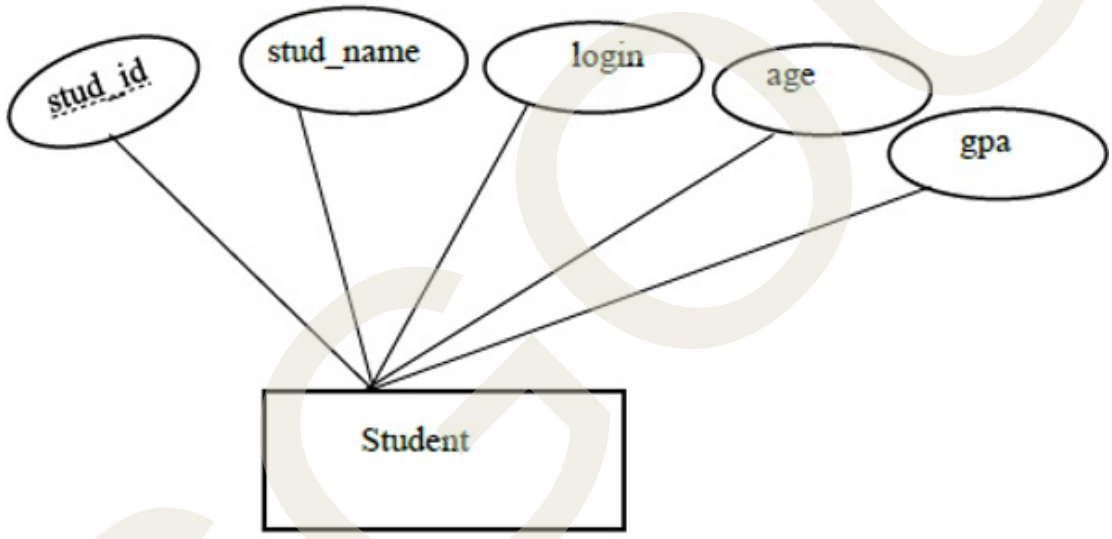


Fig 1.4.2 ER diagram of Student

In the below example faculty is a strong entity with attributes fac_id as primary key, fac_name, email_id.

In this example a student is a strong entity with attributes stud_id as primary key, stud_name, login, age, gpa.

1.4.5.1 Conversion of ER Model to Relational Schema

Stud_id	Stud_fname	Stud_lname	login	age	gpa
2100	Neo	Smith	Smith@cs	20	4.2
2155	Harry	Jack	Jack@economics	21	3.9
2345	John	Nimal	Nimal@english	20	3.5
2454	Dennis	Job	Job@history	22	3.8

Schema of the above relation is:

Student (stud_id: string, stud_fname: string, stud_lname: string, login: string, age: integer, gpa : real)

In the student table all the strong entity conversion rules are followed ie. ,

- No change in strong entity name
- All entities in ER diagram included in student table
- Composite attribute name is divided into simple attributes Stud_fname and stud _lname.
- No multi valued attributes present in the table.
- Stud_id attribute selected as the Primary Key.

1.4.6 Conversion of weak entity into a relational relation

A **weak entity** is an entity that **cannot exist independently** and relies on a **strong entity** for its existence. It does not have a **sufficient primary key** on its own and instead uses a **foreign key** referencing the related strong entity.

Steps for Conversion:

- Create a Separate Table for the Weak Entity
  - Convert the weak entity into a relation (table).
- Include Attributes as Columns
  - All attributes of the weak entity become columns in the table.
- Use a Composite Primary Key
  - A weak entity does not have a unique identifier, so its primary key is a combination of:
  - Primary key of the strong entity must be included as the foreign key of the weak entity
  - The discriminator (partial key) of the weak entity.
- Establish a Foreign Key Relationship
  - Choose the primary key as a combination of foreign key and decimator attribute of the weak entity

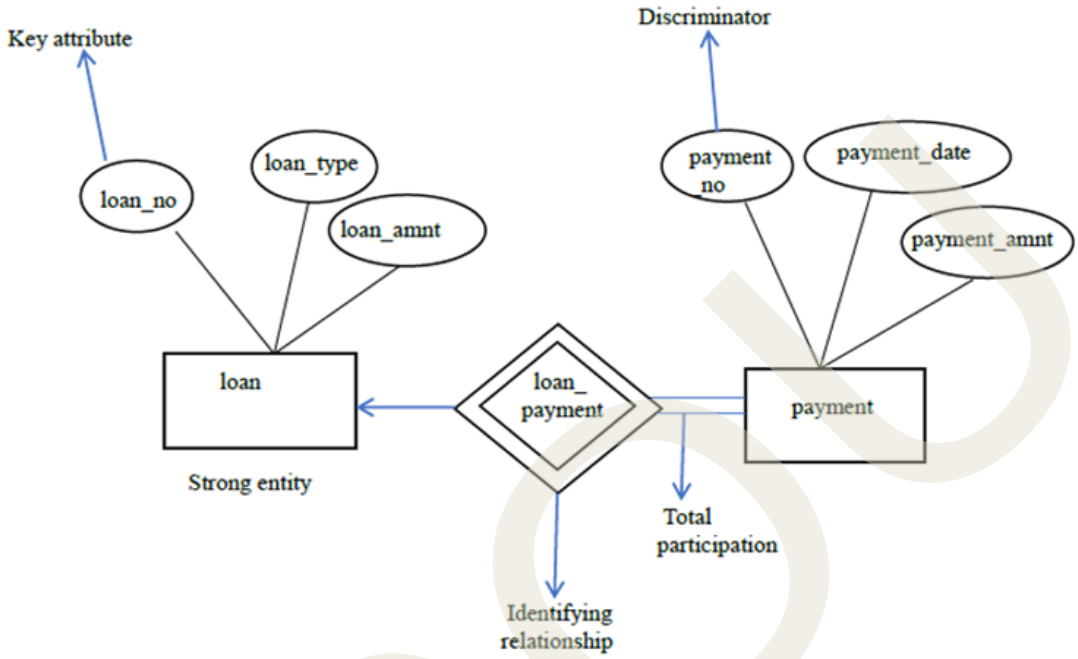
Notes

Weak entities need a strong entity to exist. A composite primary key is formed using

the foreign key + a unique attribute of the weak entity. A foreign key constraint ensures referential integrity.

Example

Consider this ER diagram with loan entity and payment entity



Loan is a strong entity and payment is a weak entity. The Discriminator or the partial key for a payment entity is payment_no attribute. The partial key along with the key attribute of the loan helps in identifying each payment record uniquely. Strong and weak entity sets always have parent-child relationships.

1.4.7 Convert Relationships to Tables

1.4.7.1 Conversion of a 1:1 Relationship into a Relational Table

A 1:1 (one-to-one) relationship in an ER model means that for each instance of Entity A, there is at most one corresponding instance of Entity B, and vice versa.

Steps for Conversion:

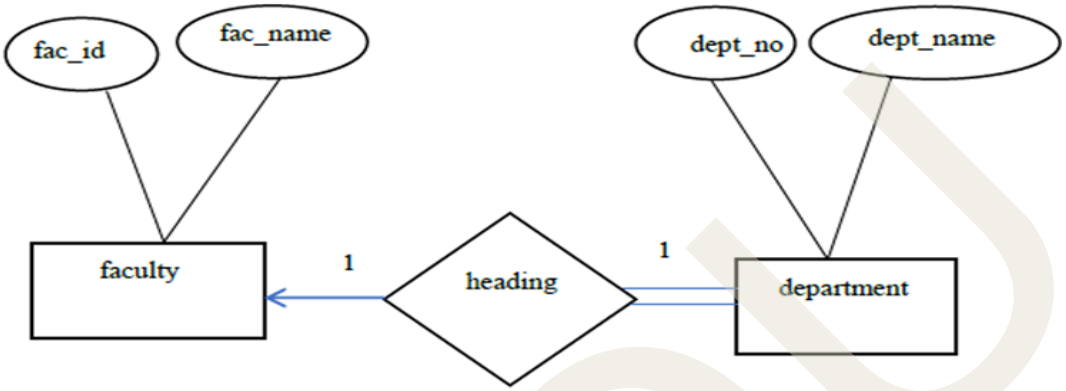
1. Merge into a Single Table (if Possible)
    - If both entities have the same primary key, they can be combined into a single table.
    - This avoids redundancy.
2. Use Separate Tables with a Foreign Key
    - If merging is not ideal, one entity's primary key is added as a

foreign key in the other.

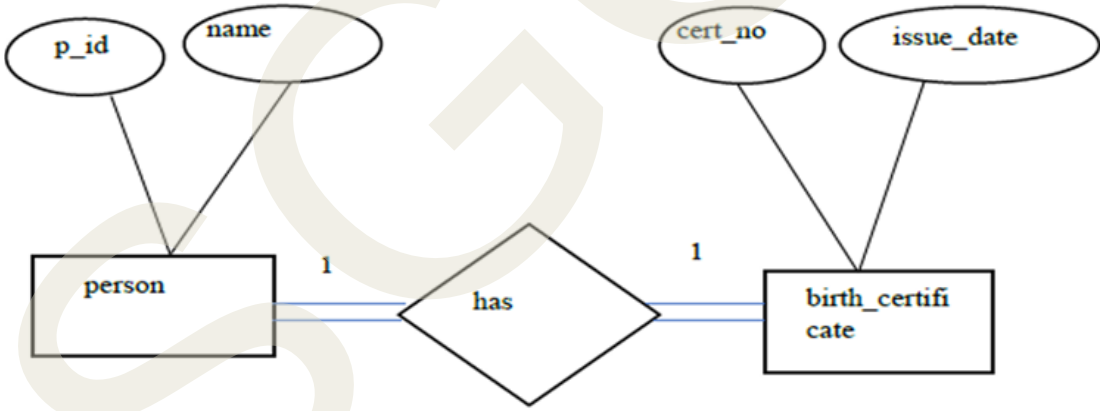
- The foreign key should have a unique constraint to maintain the 1:1 relationship.

3. Foreign Key Placement

- Choose the entity with fewer instances to hold the foreign key.



In this example if we are making dept_no as foreign key of the faculty table, then we can‟t say that all faculties are heading departments. So it is convenient to make fac_id as foreign key to the department table, i.e., for every department there exists a head of the department , which represents the total participation.



In this one to one relationship both entities have total participation so that both tables can be merged to a single table.

1.4.7.2 Conversion of one-many(1-M) relationship

In a one-to-many (1-M) relationship in a database, one record in the first table can be related to multiple records in the second table. The conversion of a 1-M relationship typically involves the following steps:

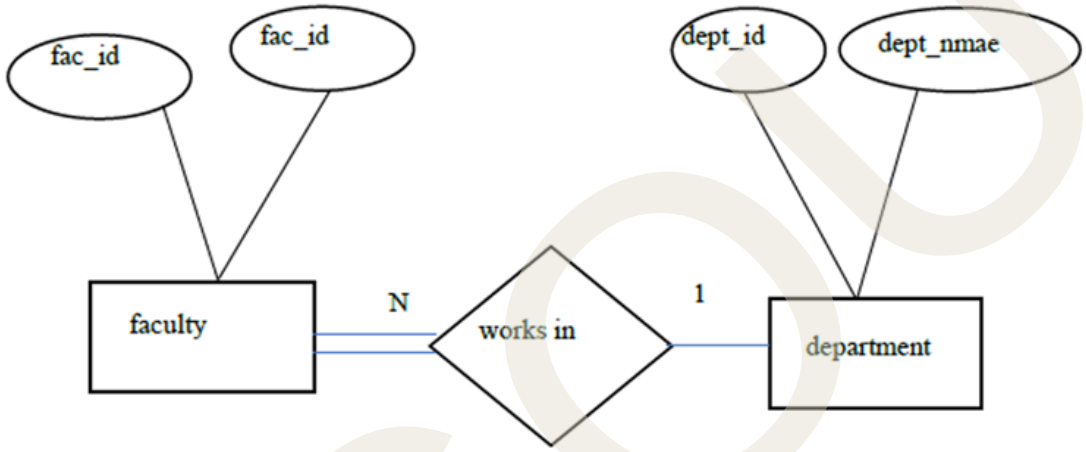
a. Identifying Entities and Relationship

- A 1-M relationship occurs when one entity (say A) is related to

multiple occurrences of another entity (say B).

b. Conversion to Tables in a Relational Database
   - The "one" side becomes a table with a Primary Key (PK).
   - The "many" side becomes another table, containing:
   - Its own Primary Key (PK)
   - A Foreign Key (FK) referencing the PK of the "one" side.

c. c. Referential Integrity
   - The Foreign Key (FK) in the Faculty table ensures that each dep_id is linked to a valid fac_id.



In this one too many examples, dept_id can be made as foreign key for the faculty table. If we are making fac_id as foreign key to the department table, the fac_id will be multi-valued so it needs to be avoided.

1.4.7.3 Conversion of Many-to-Many (M-N) Relationship

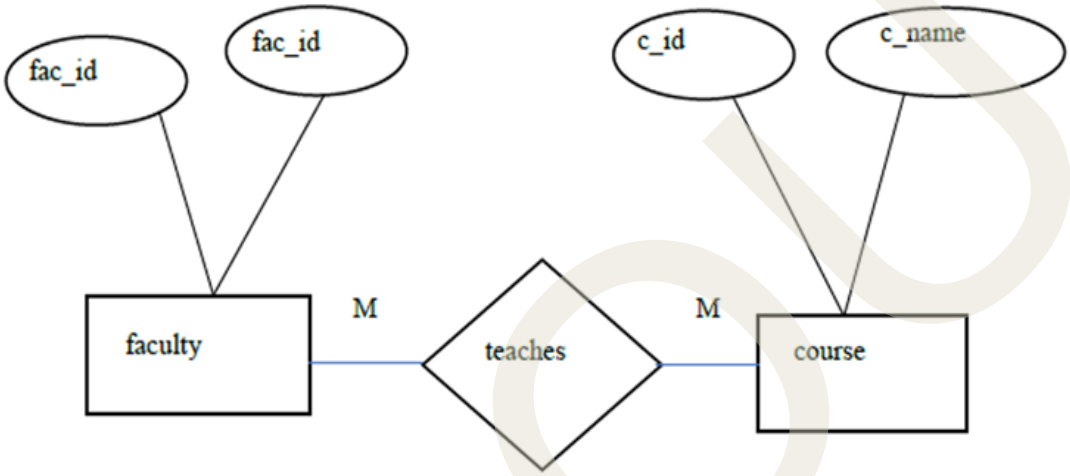
A many-to-many (M-N) relationship exists when multiple records in one entity are related to multiple records in another entity. This type of relationship cannot be directly implemented in a relational database. Instead, it is converted into two one-to-many (1-M) relationships using a junction (bridge) table.

Identifying Entities and Relationship

- Suppose we have two entities:
  - Students (who can enroll in multiple courses)
  - Courses (which can have multiple students enrolled)
- Since a student can enroll in multiple courses, and a course can have multiple students, this forms an M-N relationship.

Conversion Strategy: Create a Junction Table

- Create two one-to-many (1-M) relationships using a new table (junction table).
- This new table contains Foreign Keys (FKs) referencing the primary keys of both entities.
- Primary Key: EnrollmentID uniquely identifies each enrollment.
- Foreign Keys: StudentID and CourseID link to the respective tables.
- ON DELETE CASCADE: Ensures that if a student or course is deleted, associated enrollments are also removed.



In this example if we need to add hours_handled then we need to create a new table with selected attributes from both tables, such as fac_id, c_id, hours_handled .

Conversion of multivalued attributes

Create a separate table for each multivalued attribute and include the primary key of the current table as foreign key of new table

The combination of foreign keys and multivalued attributes must be considered as the primary key of the new table

Recap

- Concepts of Relational Database Model
- Components of Relational Database model
  - Relation
  - Attribute
  - Tuple
  - Domain
  - Relation Instance
  - Degree
  - Cardinality
  - Key
  - Constraints
- Concepts of foreign key reference.
- Strong entity to a table: separate table for strong entity-ignore multivalued attributes - composite attribute to simple attributes
- Weak entity to table: separate table for strong entity - primary key of strong entity as foreign key
- 1-1 relationship to table: primary key of one of the sides as foreign key
- 1-M relationship to table: the table of M side should have the primary key of the other side as a foreign key
- M-N relationship to table: create a separate table - include the primary key of M side and N sides
- Multivalued attributes: separate table for each multivalued attribute

Objective Type Questions

1. Which is the name of the integrity constraints used to refer to another table?
2. What is the name of the key contained in a weak entity?
3. Which is the other name for the partial key?
4. Which is the term used for representing data in relational models?
5. What is the other name for record in a relation?
6. Which is the type of participation that every entity of the corresponding entity type participates in the relationship?
7. Which is the type of participation that some of the entities of the corresponding entity type participates in the relationship?
8. What is the term used to represent the number of fields in a relation?
9. What is the term used to represent the number of rows in a relation?
10. Double line is used to represent which type of participation in a relationship?
11. Double diamond is used to represent which relationship?

Answers to Objective Type Questions

1. Foreign key
2. Partial key
3. Discriminator
4. Relation
5. Tuple
6. Total participation
7. Partial participation
8. Degree/arity
9. Cardinality
10. Total participation
11. Identifying relationship

Assignments

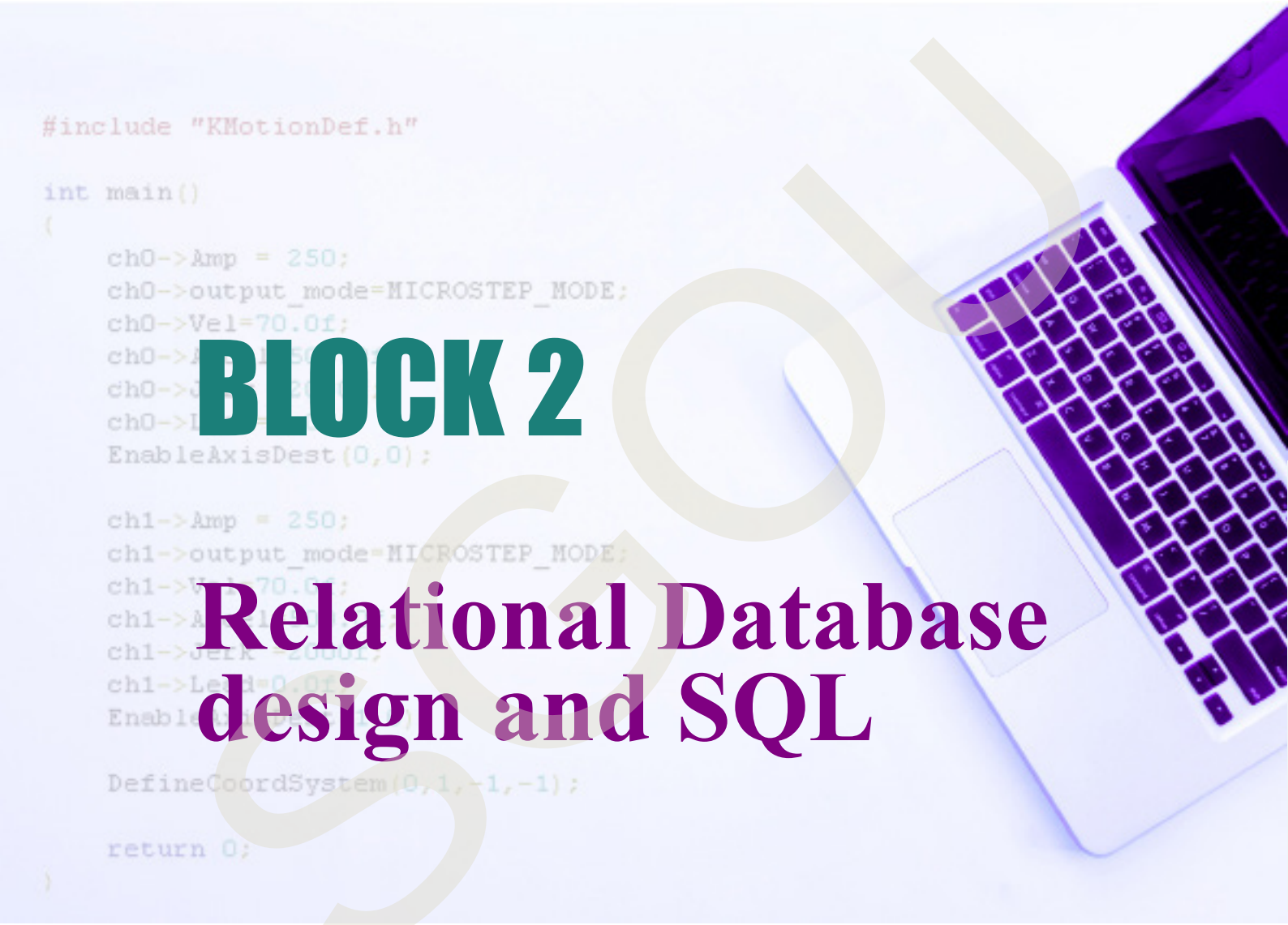
1. Explain the key components of the Relational Database Model. How do they contribute to data organization and integrity?
2. Define and differentiate between Relation, Attribute, Tuple, and Domain in the context of a relational database. Provide examples.
3. What is a Relation Instance? How does it differ from Relation Schema? Illustrate with an example.
4. Explain the concepts of Degree and Cardinality in a relational database. How do they impact database design?
5. Discuss the different types of Keys (Primary Key, Candidate Key, Super Key) in a relational model. Why is a Primary Key important?
6. What are Constraints in a relational database? Explain different types of constraints with suitable examples.
7. What is a Foreign Key? How does foreign key reference help maintain referential integrity? Provide an example using two related tables.
8. Describe the process of converting a Strong Entity into a relational table. How are composite and multivalued attributes handled?
9. Explain how Weak Entities are converted into relational tables. Why is the Primary Key of the Strong Entity used as a Foreign Key?
10. Discuss the relational mapping rules for different types of relationships (1-1, 1-M, M-N). How are multivalued attributes handled in relational tables?

Suggested Reading

1. Ramakrishnan, R., & Gehrke, J. (2014). *Database management systems* (2nd ed.). McGraw-Hill Education.
2. Silberschatz, A., Korth, H. F., & Sudarshan, S. (2010). *Database system concepts* (6th ed.). McGraw-Hill Education

Reference

1. https://www.geeksforgeeks.org/dbms
2. https://www.tutorialspoint.com/dbms/dbms_architecture.htm



BLOCK 2

Relational Database design and SQL

Functional Dependency

Learning Outcomes

At the end of this unit, the learner will be able to;

- To define functional dependency
- To explain the need for functional dependency
- To introduce different types of functional dependencies
- To make the student aware of the need for normalization
- To familiarize the various normal forms

Prerequisites

A relation schema is a list of attributes that describe the structure of a relation. A relation schema is formed by grouping attributes. The attributes of a relation schema can be grouped in different ways. How do we determine one grouping of attributes is better than that of another? How do we analyse the goodness of the design of a relations schema?

One of the serious difficulties in using relations is information redundancy there by increasing file storage. Another difficulty is the problem of update anomalies. These anomalies are insertion, deletion, and modification anomalies. Inserting, updating or deleting data in one relation, while omitting to make corresponding changes in other relations may cause inconsistencies in the relations. While de- signing relation schema we should avoid such anomalies and redundancies. Functional dependency and normalization are formal tools that help us to design a good relation schema.

Keywords

Functional dependency, full functional dependency, partial functional dependency, transitive functional dependency, normal forms

Discussion

2.1.1 Functional dependency

Consider the relation STUDENT(student id, student name, date of birth). Con- sider the attributes date of birth and student name. There might be multiple stu- dents with the same date of birth on the table. Therefore, we cannot uniquely identify a student's name from his/her date of birth. So we can say that the attribute date of birth does not determine the attribute student name or we can say student name is not functionally dependent on date of birth.

Now consider the attributes student id and student name. Every student will have a unique student id. Therefore, we can uniquely identify a student name from his/her student id. Then we can say that student id determines student name or student name is functionally dependent on student id.

Functional dependency: A functional dependency, denoted by X → Y, be- tween two sets of attributes X and Y that are subsets of relation R specifies a constraint on the possible tuples that can form a relation state r of R. The con- straint is that, for any two tuples t1 and t2 in r that have t1[X] = t2[X], they must also have t1[Y] = t2[Y]. This means that the values of the X component of a tuple uniquely (or functionally) determine the values of the Y component.

Consider the relation R(A,B,C) given in Table 2.1.1.

Table 2.1.1 A relation R(Order ID, Customer Name, Item)

Order ID	Customer Name	Product
101	John	Laptop
101	John	Mouse
102	John	Laptop
102	John	Keyboard

Consider the following functional dependencies for the above relation R:

Order ID Customer Name

Order ID,Product—> Customer Name

Consider attributes Order ID and Customer Name. 101 and 102 are the distinct values of attribute Order ID in the table. For each VALUE OF 101, there is a unique value in B that is value as John. Therefore, Order ID→ Customer Name. i.e. Order ID determines Customer Name or we can say Customer Name is functionally dependent on Order ID.

Consider the values of the attributes Customer Name and Product. For each value 'john' in Customer Name there are multiple corresponding values in the Product column. Therefore, the functional dependency Customer Name → Product cannot hold true.

Consider the attributes Order ID, Product and Customer Name. The values of Order ID, Product are (101, Laptop), (101,Mouse), (102, Laptop) and (102, Keyboard). For each combination of Order ID and Item, there is a unique value in Customer Name. Therefore, Order ID and Item functionally determines Customer Name. i.e. Order ID, Product—-> Customer Name.

2.1.2 Types of functional dependencies

There are three types of functional dependencies.

1. Partial functional dependency
2. Transitive functional dependency
3. Full functional dependency

2.1.2.1 Partial functional dependency

Definition:

A partial functional dependency occurs when a non-prime attribute (an attribute that is not part of any candidate key) is functionally dependent on part of a composite candidate key, but not on the entire key.

Consider the relation *R(A, B, C, D)* where AB is the primary key. Following functional dependencies are defined on the relation:

$AB \rightarrow C$

$B \rightarrow D$

D is a non key attribute as it is not part of the primary key *AB*. *B* is a part of the primary key *AB*. *D* is dependent on B. Therefore, $B \rightarrow D$ is a partial functional dependency as a non key attribute is dependent on part of the primary key.

All non-key attributes in a table should be totally dependent on the entire primary key. If they depend on only a part of the primary key, it is called a partial functional dependency.

Under the following circumstances a table cannot have partial functional dependencies.

- if primary key consists of only one attribute
- if table consists of only two attributes
- if all the attributes in a table are part of primary key

2.1.2.2 Transitive functional dependency

Consider the relation *R(A, B, C, D)* where *AB* is the primary key. Following functional dependencies are defined on the relation:

$AB \rightarrow C$ (Primary key AB determines C)

$C \rightarrow D$ (C determines D, where C and D are non key attributes)

Thus $AB \rightarrow D$ *exists transitively. ie* $AB \rightarrow D$ via C

C and D are non key attributes and C determines D. We have a transitive dependency. ie, D is transitively dependent on AB because it is not directly determined by AB, but rather through C.

A transitive functional dependency is a type of functional dependency where a non- key attribute depends on another non key attributes which in turn depends on the primary key.

2.1.2.3 Full functional dependency

A full functional dependency occurs in a relational database when a non-prime attribute is functionally dependent on the entire candidate key and not on any proper subset of it. This means the dependency holds only when all attributes of the candidate key are considered together, and removing any attribute from the candidate key would break the dependency.

Consider the functional dependency $ABC \rightarrow D$ where D is dependent on A,B and C. If any attribute is removed from ABC the resulting functional dependency becomes invalid. For example, If A is removed, the resulting dependency would be $BC \rightarrow D$. However, since D depends on all three attributes A,B and C, the dependency $BC \rightarrow D$ is invalid. Therefore, $ABC \rightarrow D$ represents a full functional dependency.

In general, a dependency of the form $X \rightarrow Y$, is considered a full functional dependency if the removal of any attribute from X makes the dependency $X \rightarrow Y$ invalid.

2.1.3 Database Anomalies and the Need for Normalization

In a database, improper organization of data can lead to problems such as redundancy and anomalies. Let's understand these issues using the example of a STUDENT table.

Example Table: STUDENT

Attributes:

1. student_id: Unique ID for each student
2. student_name: Name of the student
3. department: The department in which the student is enrolled
4. dept_head: The head of the department

2.1.3.1 Redundancy

Definition:

When the same data is repeated unnecessarily in a database, it leads to redundancy.

Example:

Suppose there are 100 students in the BCA department. For all these students, the values for the attributes department and dept_head will be repeated 100 times. This repetition is an example of redundancy, which increases storage requirements and makes the database inefficient.

2.1.3.2 Anomalies

Anomalies occur when there are issues in inserting, updating, or deleting data due to poor database design. The major types of anomalies are:

1. Update Anomaly

Example:

If the head of the BCA department changes, the change must be made for all 100 students in the STUDENT table. If we forget to update the dept_head for any of these students, the database will become inconsistent, as different records may show different department heads.

2. Insertion Anomaly

Example:

Suppose a new department is created, but there are no students yet in that department. To record the new department in the STUDENT table, we would have to insert NULL values for student_id and student_name. However, primary keys like student_id cannot be NULL. This situation is an example of an insertion anomaly.

3. Deletion Anomaly

Example:

If a department has only one student and that student discontinues, deleting the student's record would also remove the information about the department. This results in a loss of important data about the department.

2.1.3.3 Solving Anomalies with Normalization

To solve updation, insertion, deletion anomalies and minimize redundancies, we need a formal way of analysing the given relation. Normalization provides a for- mal framework for analysing relation schemas based on functional dependencies among attributes of a relation. Normalization checks whether a relation schema satisfies a normal form. A relation schema that does not satisfy the given normal form can be decomposed into smaller relation schemas that satisfies the normal form. In our case, we are considering the following normal forms.

- First normal form (1NF)
- Second normal form (2NF)

- Third normal form (3NF)
- Boyce-Codd normal form (BCNF)

2.1.4 First Normal Form

Definition:

A table is said to be in First Normal Form (1NF) if:

1. All the values in the table are atomic (indivisible).
2. Each column contains only one value for each row (no repeated or grouped values in a single cell).
3. Each row is unique, meaning it can be identified by a unique primary key.

Consider a relation STUDENT(Student_ID, Name, Subjects) as shown in the table 2.1.2

Table 2.1.2 STUDENT relation

Student_ID	Name	Subjects
1	John	Math, Physics
2	Alex	Chemistry, Biology
3	Helen	Math, Chemistry, Biology

In this table:

The Subjects column contains multiple values in every row. (such as "Math, Physics" for John), which is not atomic.

This violates 1NF, as each column must contain indivisible values.

Solution (Convert to 1NF):

To convert the relation into 1NF, we separate the multiple values in the Subjects column into individual rows, making sure that each cell holds only one value.

Table 2.1.3 Relation after converted to 1NF

Student_ID	Name	Subjects
1	John	Math
1	John	Physics
2	Alex	Chemistry
2	Alex	Biology
3	Helen	Math
3	Helen	Chemistry
3	Helen	Biology

Now, every column contains atomic values, and the relation is in 1NF.

In simple terms, First Normal Form (1NF) does not allow a relation to have other relations or attribute values within a tuple. It is used to remove multivalued attributes in a table, ensuring each attribute holds only a single value.

2.1.5 Second Normal Form

Second Normal Form (2NF) is a level of database normalization that ensures a table is organized to reduce redundancy and dependency. It builds upon the rules of First Normal Form (1NF). A table is in 2NF if it meets the following conditions:

1. It is already in First Normal Form (1NF)
2. It has no partial dependencies

Example 1:

Suppose we have the following relation as shown in table 2.1.4:

Table 2.1.4 Customer data

Customer_ ID	Order_ ID	Product_ ID	Product_ Name	Product_ Price
1	101	P1	Laptop	40000
1	102	P2	Phone	20000
2	103	P1	Laptop	40000
3	104	P3	Tablet	10000

Consider the following functional dependencies:

Customer_ID, Order_ID → Product_ID, Product_Name, Product_Price

Product_ID → Product_Name, Product_Price

The primary key of the relation is Customer_ID, Order_ID.

Product_Name and Product_Price depend only on Product_ID, not on the full primary key

Customer_ID, Order_ID. This is a partial dependency, which violates 2NF. So the above given relation is not in 2NF.

Solution: Convert to 2NF:

To convert the relation into 2NF, we need to remove the partial dependency by splitting the relation:

Relation 1: Store customer-order-product data that depends on the full primary key Customer_ID, Order_ID.

- R1(Customer_ID, Order_ID)

- Functional Dependency:

  Coustomer_ID, Order_ID → Product_ID

Table 2.1.5 Relation R1

Customer_ID	Order_ID	Product_ID
1	101	P1
1	102	P2
2	103	P1
3	104	P3

Relation 2: Store product details that depend only on Product_ID.

- R2(Product_ID, Product_Name, Product_Price)
- Functional dependency:

  Product_ID → Product_Name, Product_Price

Table 2.1.6 Relation R2

Product_ID	Product_Name	Product_Price
P1	Laptop	40000
P2	Phone	20000
P3	Tablet	10000

Now the relations, R1 and R2 are in 2NF.

2.1.6 Third Normal Form

Third Normal Form (3NF) is a way of organizing a database to remove unnecessary data duplication and ensure the data is logically stored.

A table is in 3NF if:

1. It is already in Second Normal Form (2NF).
2. It has no transitive dependencies.

Example:

Consider the following relation **Employee**

Table 2.1.7 Employee Data

Emp_ID	Emp_Name	Dept_ID	Dept_Name
1	Alice	D1	HR
2	Helen	D2	IT
3	John	D1	HR

- Emp_ID is the primary key.
- Emp_Name, Dept_ID, and Dept_Name are non-prime attributes.
- Functional Dependencies:

1. Emp_ID → Emp_Name, Dept_ID
2. Dept_ID → Dept_Name

The given relation Employee is not in 3NF. Because the non-prime attribute Dept_Name depends on another non-prime attribute Dept_ID, which creates a transitive dependency:

ie, nonprime attribute Dept_Name indirectly depends on Emp_ID

- Emp_ID → Dept_ID → Dept_Name

According to 3NF, all non-prime attributes should depend only on the primary key and not on other non-prime attributes.

Solution to Convert into 3NF:

Break the table into two smaller tables:

1. Employee Table

Table 2.1.8 Employee table

Emp_ID	Emp_Name	Dept_ID
1	Alice	D1
2	Helen	D2
3	John	D1

2. Department Table

Table 2.1.9 Department Table

Dept_ID	Dept_Name
D1	HR
D2	IT

- In the Employee Table, all non-prime attributes (Emp_Name, Dept_ID) depend directly on the primary key (Emp_ID).
- In the Department Table, the non-prime attribute (Dept_Name) depends directly on the primary key (Dept_ID).
- There are no transitive dependencies. So the relations are in 3NF.

2.1.7 Boyce-Codd Normal Form

BCNF is a higher version of the Third Normal Form (3NF). It resolves anomalies caused by certain types of functional dependencies. A relation is in BCNF if, for every

functional dependency X → Y, X is a superkey.

Key Characteristics:

- A superkey is any attribute or combination of attributes that can uniquely identify a row in a table.
- BCNF eliminates redundancy and dependency issues more rigorously than 3NF.
- Every BCNF relation is in 3NF, but the reverse is not always true.

Example:

Consider the following relation Student_Course.

Table 2.1.10 Student_Course Data

Student_ID	Course_ID	Instructor_Name
1	C1	Dr.Smith
2	C2	Dr. Johnson
3	C1	Dr.Smith

Functional Dependencies:

1. Student_ID → Course_ID
2. Course_ID → Instructor_Name

The above given relation Student_Course is not in BCNF because:

- Course_ID → Instructor_Name violates BCNF because Course_ID is not a superkey, yet it determines Instructor_Name.
- The key for the table is Student_ID + Course_ID, but Course_ID alone determines Instructor_Name.

Solution to Convert into BCNF:

Decompose the table into two relations:

Table 2.1.11 Student_Course table

Student_ID	Course_ID
1	C1
2	C2
3	C1

Table 2.1.12 Course_Instructor

Course_ID	Instructor_Name
C1	Dr. Smith
C2	Dr. Johnson

Now, both tables are in BCNF.

Recap

Functional Dependency

- Functional Dependency (FD) is a concept in database design that describes a relationship between two sets of attributes in a relation (table).
- It is denoted as X $\rightarrow$ Y, where:
    - X is called the determinant (a set of one or more attributes).
    - Y is called the dependent (another set of attributes).

Types of Functional Dependencies

- Partial functional dependency: a non key attribute depends on part of primary key
- Transitive functional dependency: relationship between non key attributes
- Full Functional Dependency: Non-prime attributes depend on the entire primary key, not on any subset.

Anomalies:

- Update Anomaly: Changes must be made in multiple places, risking inconsistency.
- Insertion Anomaly: Inability to insert data without violating integrity constraints.
- Deletion Anomaly: Loss of data due to deletion of related data.

Normalization: Solves these issues by decomposing tables into smaller, structured relations.

- 1NF: no multi valued dependencies
- 2NF: In 1NF, no partial dependencies
- 3NF: In 2NF, no transitive dependencies
- BCNF: In 3NF, determinants superkey

Objective Type Questions

1. Consider two sets of attributes X and Y in a relation. Let t1 and t2 are two tuples of the relation. If t1[X] = t2[X], then what is the constraint on Y so that Y is functionally dependent on X?
2. Which functional dependency occurs when a non key attribute depends on part of the primary key attribute in a relation?
3. Which functional dependency occurs when there is a relationship between non key attributes in a table?
4. Consider the table below.

A	B	C
1	b1	3
2	b2	1
3	b3	3
4	b1	2

Is the functional dependeny AB→C a capital, full or transitive dependency?.

5. Consider the table below.

A	B	C
1	b1	3
2	b2	1
3	b3	3
4	b1	2

Will the functional dependency C → B hold true?

6. Consider X → Y . Which function dependency occurs if removal of any attribute in X makes the given functional dependency invalid?
7. Consider the relation R(A, B, C, D) where AB is the primary key. Which functional dependency is satisfied by B → D?
8. Consider the relation R(A, B, C, D) where AB is the primary key. Which functional dependency is satisfied by C → D?

9. Which anomaly makes the database inconsistent on updating a tuple?
10. Which anomaly makes the database inconsistent on deleting a tuple?
11. Which anomaly makes the database inconsistent on inserting a tuple?
12. Which process checks whether a relation is good based on functional dependencies?
13. Which normal form satisfies the condition that the value of any attribute in a tuple must be a single value from the domain of that attribute?
14. Which normal form satisfies the condition that the relation is free from partial dependencies?
15. Which normal form satisfies the condition that the relation is free from transitive dependencies?
16. Which normal form satisfies the condition that determinants must be superkey?

Answers to Objective Type Questions

1. t1[Y] = t2[Y]
2. Partial functional dependency
3. Transitive functional dependency
4. Full functional dependency
5. No
6. Full functional dependency
7. Partial functional dependency
8. Transitive functional dependency
9. Updation anomaly
10. Deletion Anomaly
11. Insertion Anomaly
12. normalization

13. INF

14. 2NF

15. 3NF

16. BCNF

Assignments

1. Define functional dependency in a relational database. Explain its significance with relevant examples.

2. Explain the importance of normalization in database design. Compare and contrast 1NF, 2NF, and 3NF with appropriate examples.

3. Define BCNF and discuss how it is different from 3NF. Provide an example of converting a relation in 3NF to BCNF.

4. Discuss the concept of data redundancy and anomalies in relational databases. How does normalization help reduce redundancy and anomalies? Illustrate with examples.

5. Consider the relation R(A, B, C, D) with the following functional dependencies:

   $A \rightarrow B$

   $B \rightarrow C$

   $A \rightarrow D$

   a. Determine the candidate keys for the relation.

   b. Explain if the given relation is in 1NF, 2NF, or 3NF. Provide detailed reasoning.

   c. Suggest modifications to the relation to achieve 3NF if it is not already in that form.

Suggested Reading

1. Ramakrishnan, R., & Gehrke, J. (2002). *Database management systems*. McGraw-Hill.
2. Coronel, C., & Morris, S. (2019). *Database systems: Design, implementation, and management* (13th ed.). Cengage Learning.
3. Foster, E., & Godbole, S. (2022). *Database systems: A pragmatic approach*. Auerbach Publications.
4. Silberschatz, A., Korth, H. F., & Sudarshan, S. (2006). *Distributed databases. In Database system concepts* (5th ed., pp. 705-749). McGraw-Hill.
5. Groff, J. R., Weinberg, P. N., & Oppel, A. J. (2002). *SQL: The complete reference* (Vol. 2). McGraw-Hill/Osborne.

Reference

1. geeksforgeeks.org/pl-sql-tutorial/
2. archive.nptel.ac.in/courses/106/105/106105175/
3. tutoxialspoint.com/plsql/index.htm

SQL Concepts

Learning Outcomes

At the end of this unit, the learner will be able to;

- familiarize with the concept of network switching and its significance in To introduce SQL concepts and queries
- To describe the usage of SQL commands to create databases and tables
- To identify SQL queries for retrieving data from databases
- To recognize the SQL queries for modifying the data in a database

Prerequisites

A database stores data in the form of tables. How do we create a database? How can we access the data in a database? How do we modify data in a database? How do we retrieve data from a database in a desired format? This is where a query language comes in. A query language provides commands for creating databases, modifying them, and accessing data in a database. They provide a standard for inter- acting with the databases. Structured Query Language (SQL) is the most widely used database query language. It provides statements to define the structure of the data, modify data in the database, and specify security constraints.

In addition to the basic functionality provided by SQL for creating, modifying, and accessing data in a database, it also allows for advanced operations such as data manipulation, data retrieval, and the establishment of relationships between tables. SQL enables users to define constraints, such as primary keys, foreign keys, and unique keys, to ensure data integrity. It also supports complex queries involving joins, subqueries, and aggregations, which help in retrieving data in the desired format. SQL's versatility and efficiency make it an essential tool for managing relational databases and conducting data analysis. Understanding SQL is fundamental to working with modern database management systems (DBMS) and is a critical skill for database administrators, developers, and data analysts.

Keywords

Structured Query Language,SQL, Data Definition Language, DDL, Data Modification Language

Discussion

2.2.1 SQL concepts

Structured Query Language(SQL) is the most commonly used query language for databases. It consists of various statements to define and manipulate relatons.

The SQL statements are categorized as follows:

Data-definition language (DDL): The SQL DDL provides commands for defining relation schemas, deleting relations, and modifying relation schemas.

Data-manipulation language (DML):The SQL DML provides the ability to query information from the database and to insert tuples into, delete tuples from, and modify tuples in the database.

2.2.2 SQL Data Types

SQL data type specifies the types of the attributes of a table. The basic SQL data types are:

- **Numeric:** includes integers (INTEGER or INT , and SMALLINT), floating point numbers (FLOAT or REAL , and DOUBLE PRECISION), formatted numbers. Formatted numbers can be declared by using DECIMAL (i, j) or DEC (i, j) or NUMERIC (i, j) where i, the precision, is the total number of decimal digits and j, the scale, is the number of digits after the decimal point.
- **Character-string:**includes fixed length characters or variable length characters. CHAR(n) or CHARACTER(n) indicates fixed length charaters where n is the length of characters. VARCHAR (n) indicates variable length characters where n is the maximum number of characters.
- **Bit-string:** includes fixed length or varying length bits. BIT (n) indicates fixed length n number of bits. BIT VARYING (n) indicates varying length bits where n is the maximum number of bits.
- **Boolean:** data type has values of TRUE or FALSE
- **DATE:** the components of the datatype are YEAR , MONTH , and DAY in the form YYYY-MM-DD.
- **TIME:** the components of the datatype are HOUR , MINUTE , and SECOND in the form HH:MM:SS.

2.2.3 DDL Commands

2.2.3.1 CREATE DATABASE command

The CREATE DATABASE command in SQL is used to create a new database. A database is a collection of data that is stored in a structured way, and it can contain tables, views, indexes, and other database objects. This command allows you to define a new database that will hold the data for your applications.

Syntax:

```
CREATE  DATABASE  database_name;
```

- CREATE DATABASE is the command used to create a new database.
- database_name is the name you want to give to the database you're creating.

For example the following command creates a database with name UNIVERSITY.

```
CREATE   DATABASE   UNIVERSITY;
```

2.2.3.2 DROP DATABASE command

The DROP DATABASE command is used in SQL to permanently delete an entire database, including all of its tables, data, schemas, and related objects. Once executed, this action cannot be undone, and all the data stored in the database will be lost. This command is typically used when a database is no longer needed.

Syntax:

```
DROP DATABASE database_name;
```

For example, the following command deletes the database UNIVERSITY.

```
DROP DATABASE UNIVERSITY;
```

2.2.3.3 CREATE TABLE

CREATE TABLE command creates a table in a database. We should specify the attributes and initial constraints of the table in the CREATE TABLE command.

Syntax:

```
CREATE  TABLE table_name
(
column1 datatype constraints,
column2 datatype constraints,
.....
ColumnN datatype constraints
);
```

- table_name: The name of the table you want to create.
- column1, column2, ... columnN: The names of the columns within the table
- data type: The data type for each column, specifying the type of data that can be stored in it.
- constraints: optional constraints, such as primary keys, unique constraints, and foreign keys, that define rules and relationships within the table.

Constraints in DBMS Table Creation

Constraints in Database Management Systems (DBMS) ensure the accuracy and reliability of data by enforcing rules on the values that can be inserted, updated, or deleted. They help maintain data integrity and prevent invalid data entry. Common constraints include Primary Key, Foreign Key, Unique, Not Null, Check, and Default.

Types of Constraints

When creating a table in SQL, constraints can be applied at the column level (to a specific column) or at the table level (affecting multiple columns). Below are the main types of constraints:

1. **NOT NULL Constraint**

Ensures that a column cannot contain NULL values. It is used when a field must always have data, such as a primary key or an email. For example, an employee ID and name cannot be NULL because every employee must have a unique ID and a name.

2. **UNIQUE Constraint**

Ensures that all values in a column are distinct, preventing duplicate entries while allowing NULLs (unless the NOT NULL constraint is also applied). For example, each employee's ID must be unique to distinguish them from others.

3. **PRIMARY KEY Constraint**

Uniquely identifies each record in a table. Combines NOT NULL + UNIQUE constraints. Only one primary key per table, but it can have multiple columns (composite primary key). EmpID uniquely identifies each employee.

4. **FOREIGN KEY Constraint**

Enforces referential integrity by linking a column to another table's **Primary Key**. Prevents actions that would destroy links between tables. `CustomerID` in `Orders` must match an existing `CustomerID` in `Customers`.

5. **CHECK Constraint**

Ensures that values in a column meet a specific condition. Used for data validation. `Age` must be at least 18, and `Salary` must be greater than 0.

6. DEFAULT Constraint

Assigns a default value if no value is provided during insertion. If no value is inserted for `Status`, it defaults to **'Active'**.

7. AUTO_INCREMENT (MySQL only)

Automatically generates unique values for a column. Commonly used for primary keys. `EmpID` automatically increments for new records.

Summary Table

Constraint	Description	Example
NOT NULL	Ensures a column cannot be NULL	`Age INT NOT NULL`
UNIQUE	Ensures all values are distinct	`Email VARCHAR(100) UNIQUE`
PRIMARY KEY	Uniquely identifies a row (NOT NULL + UNIQUE)	`EmpID INT PRIMARY KEY`
FOREIGN KEY	Ensures referential integrity	`FOREIGN KEY (DeptID) REFERENCES Department(DeptID)`
CHECK	Restricts values based on a condition	`CHECK (Age >= 18)`
DEFAULT	Assigns a default value if none is provided	`Status VARCHAR(20) DEFAULT 'Active'`
AUTO_INCREMENT	Auto-generates unique values (MySQL)	`EmpID INT AUTO_INCREMENT PRIMARY KEY`

For example, the following command creates a table STUDENT with ID as the primary key. NOT NULL is a constraint that specifies that the value of the field cannot be null.

Example:

```
CREATE TABLE STUDENT(
ID INT NOT NULL,
NAME VARCHAR(20) NOT NULL,
SEMESTER INT NOT NULL,
DEPARTMENT CHAR(25),
PRIMARY KEY (ID)
);
```

2.2.3.4 DROP TABLE

The DROP TABLE command in SQL is used to permanently delete a table from a database. When executed, it removes the specified table, including all its data, structure,

and associated objects such as indexes or constraints.

Syntax:

```
DROP TABLE table_name;
```

Example:

DROP Table STUDENT;

The above command will completely remove the STUDENT table from the database. Once a table is dropped, all its data is lost, and this action cannot be undone unless a backup exists.

2.2.4 Selecting and checking databases

Checking Databases: Involves listing all available databases in the system using commands like SHOW DATABASES; to view and verify which databases exist.

Syntax:

```
SHOW DATABASES;
```

Example Output:

After executing SHOW DATABASES;, you might see output like the table 2.2.1

Table 2.2.1 Result of SHOW DATABASE command

Databases
UNIVERSITY_Database
EMPLOYEE_Database
STUDENT_Database

Selecting a Database:

Once you identify the database you want to use, you can make it active with the USE command.

Syntax:

```
USE   database_name;
```

Example:

USE UNIVERSITY;

This sets UNIVERSITY as the active database.

2.2.5 DML Commands (DATA MANIPULATION LANGUAGE)

2.2.5.1 INSERT INTO

Add rows into a table

The INSERT INTO command in SQL is used to add rows to a table. To insert a row, you must specify the table name and provide a list of values. The values should be listed in the same order as the attributes defined in the CREATE TABLE command. For example, the following command adds a row to the STUDENT table:

```
INSERT INTO STUDENT

VALUES (1, 'Aman', 2, 'BCA');
```

Another form of the INSERT command allows you to explicitly specify the attribute names. This is useful when you want to insert values for only a subset of attributes in the table. However, you must ensure that all attributes with NOT NULL constraints and no default values are included. For example:

```
INSERT INTO STUDENT(ID, NAME, SEMESTER)

VALUES (1, 'Aman', 2);
```

2.2.5.2 SELECT command

The SELECT statement in SQL is used to retrieve information from databases. It is one of the most commonly used commands for querying data. The basic structure of a SELECT statement is as follows:

```
SELECT column1, column2, ...

FROM table_name

WHERE condition;
```

where

- attribute list is a list of attribute names whose values are to be retrieved by the query.
- table list is a list of the relation names required to process the query.
- condition is a conditional (Boolean) expression that identifies the tuples to be retrieved by the query.

In SQL, the basic comparison operators are =, <, <=, >, >= and <> which cor-

responds to the relational algebra operations =, ≤, ≤, ≥, ≥, and ! =.

Table 2.2.2 STUDENT table with attributes ID, NAME, SEMESTER, and DEPARTMENT

ID	NAME	SEMESTER	DEPARTMENT
1	RAM	1	BCA
2	GEORGE	2	HISTORY
3	KEERTI	3	ENGLISH
4	IRFAN	1	ECONOMICS
5	KOMAL	2	MALAYALAM
6	KIRAN	4	MATHEMATICS
7	NIRMAL	1	ENGLISH

Consider the Table 2.2.2, to retrieve the ID and NAME of the students in semester 1 we have to give the query:

```
SELECT ID, NAME
FROM STUDENT
WHERE SEMESTER=1;
```

The result of the above query will be displayed as shown in Table 2.2.3. .

Table 2.2.3 Result of SELECT command

ID	NAME
1	RAM
4	IRFAN
7	NIRMAL

If you want to retrieve all the rows of STUDENT table use the following query:

```
SELECT * FROM STUDENT;
```

AND and OR operators can be used to combine multiple conditions in the WHERE clause. For example, the following query will retrieve the ID and NAME of students in first semester BCA.

```
SELECT ID, NAME
FROM STUDENT
WHERE SEMESTER=1 AND DEPARTMENT='BCA';
```

Result of above query is shown in table 2.2.4

Table 2.2.4 SELECT command with AND and OR operators

ID	NAME
1	RAM

2.2.5.3 DELETE command

The DELETE command in SQL is used to remove one or more rows from a table based on a specified condition. It allows you to delete data without affecting the structure of the table. For example, the following query on table 2.2.2 will produce the result in table 2.2.5

```
DELETE FROM STUDENT

WHERE ID=3;
```

Result of the above query is:

Table 2.2.5 Result of DELETE query after deleting the record with ID=3

ID	NAME	SEMESTER	DEPARTMENT
1	RAM	1	BCA
2	GEORGE	2	HISTORY
4	IRFAN	1	ECONOMICS
5	KOMAL	2	MALAYALAM
6	KIRAN	4	MATHEMATICS
7	NIRMAL	1	ENGLISH

2.2.5.4 UPDATE command

The UPDATE command is used to modify rows in a table.

Syntax:

```
UPDATE table_name

SET column1 = value1, column2 = value2, ...

WHERE condition;
```

The SET clause in the UPDATE command specifies the attributes to be modified and their new values. For example, the following query will change the semester of the student with ID 2 to 3 in the table 2.2.2. The result is shown in table 2.2.6.

```
UPDATE STUDENT

SET SEMESTER=3

WHERE ID=2;
```

Result of the above query is:

Table 2.2.6 Result of UPDATE query after updating the semester of student with

ID=3

ID	NAME	SEMESTER	DEPARTMENT
1	RAM	1	BCA
2	GEORGE	3	HISTORY
3	KEERTI	3	ENGLISH
4	IRFAN	1	ECONOMICS
5	KOMAL	2	MALAYALAM
6	KIRAN	4	MATHEMATICS
7	NIRMAL	1	ENGLISH

2.2.6 Substring pattern matching

In SQL, substring pattern matching can be performed using several string functions. The most common ones are:

2.2.6.1 LIKE Operator

The LIKE operator is used to search for a specified pattern in a column. It is typically used with wildcard characters (% and _) for pattern matching.

%: Represents zero or more characters.

_: Represents exactly one character.

Syntax:

```
SELECT column_name
FROM table_name
WHERE column_name LIKE 'pattern';
```

Example:

The following query retrieves the details of all the students whose name starts with 'K'.

```
SELECT * FROM STUDENT
WHERE NAME LIKE 'K%';
```

Result is shown in table 2.2.7

Table 2.2.7 Result of LIKE operator

ID	NAME	SEMESTER	DEPARTMENT
3	KEERTI	3	ENGLISH
5	KOMAL	2	MALAYALAM
6	KIRAN	4	MATHEMATICS

2.2.7 Ordering of query result

ORDER BY clause can be used to order the tuples in the result of a query by the values of one or more of the attributes that appear in the query result. For example, the following query retrieves the ID and NAME of all students in the STUDENT table with names ordered alphabetically in ascending order. ASC or DESC keyword is used with order by clause to retrive result in ascending on descending order.

If not specify anything it will generate ascending order of selected column.

For example,

```
SELECT ID, NAME
FROM STUDENT
ORDER BY NAME;
```

Result:

Table 2.2.8 Result of ORDER BY command

ID	NAME
2	GEORGE
4	IRFAN
3	KEERTI
6	KIRAN
5	KOMAL
7	NIRMAL
1	RAM

If we have to retrieve the results in the descending order of names, use the DESC keyword.

```
SELECT ID, NAME
FROM STUDENT
ORDER BY NAME DESC;
```

Result:

Table 2.2.9 Result of ORDER BY command with DESC Keyword

ID	NAME
1	RAM
7	NIRMAL
5	KOMAL

6	KIRAN
3	KEERTI
4	IRFAN
2	GEORGE

2.2.8 Aggregate functions

Aggregate functions are used to perform calculations on a set of values and return a single result. The five built-in aggregate functions:

8. Average: AVG
9. Minimum: MIN
10. Maximum: MAX
11. Total: SUM
12. Count: COUNT

2.2.8.1 Average: AVG

The AVG function calculates the average (mean) of a set of numeric values. It is commonly used to find the average salary, grade, or any other numeric metric in a table.

Syntax:

```
SELECT AVG(column_name)
FROM table_name;
```

Example:

Consider the following table 2.2.10

Table 2.2.10 STUDENT table with attributes ID, NAME, SEMESTER, MARKS and DE- PARTMENT

ID	NAME	SEMESTER	MARKS	DEPARTMENT
1	RAM	1	90	BCA
2	GEORGE	3	95	HISTORY
3	KEERTI	3	90	ENGLISH
4	IRFAN	1	85	ECONOMICS
5	KOMAL	2	75	MALAYALAM
6	KIRAN	4	60	MATHEMATICS
7	NIRMAL	1	72	ENGLISH

The AVG function calculates the average marks obtained by all students in the given table. Consider the following query:

SELECT AVG(MARKS)

FROM STUDENT

WHERE DEPARTMENT=’ENGLISH’;

The query calculates the average marks of students who belong to the 'ENGLISH' department. It uses the AVG() function to compute the mean value of the MARKS column, filtering only those rows where the DEPARTMENT is 'ENGLISH'.

Result:

AVG(MARKS): 81

2.2.8.2 Minimum: MIN

The MIN function returns the smallest value in a column. It is used to find the minimum value in datasets like the lowest price, minimum marks, or shortest time.

Syntax:

```
SELECT MIN(column_name)
FROM table_name;
```

Example:

Consider the following query

SELECT MIN(MARKS)

FROM STUDENT

WHERE DEPARTMENT = 'ENGLISH';

The above query retrieves the minimum marks of students from the ENGLISH department by filtering rows with DEPARTMENT = 'ENGLISH' and applying the MIN() function on the MARKS column.

Result:

MIN(MARKS): 72

2.2.8.3. Maximum (MAX)

The MAX function retrieves the largest value in a column. For example It helps identify the highest salary, maximum marks, or largest order value in a dataset.

Syntax:

```
SELECT MAX(column_name)
FROM table_name;
```

Example:

Consider the following query

```
SELECT MAX(MARKS)
FROM STUDENT
WHERE DEPARTMENT = 'ENGLISH';
```

The above SQL query retrieves the highest marks (maximum value) from the MARKS column in the STUDENT table, but only for rows where the DEPARTMENT is 'ENGLISH'.

Result:

MAX(MARKS): 90

2.2.8.4 Total (SUM)

The SUM function adds all the numeric values in a column and returns the total.

Syntax:

```
SELECT SUM(column_name)
FROM table_name;
```

Example:

Consider the following query:

```
SELECT SUM(MARKS)
FROM STUDENT
WHERE DEPARTMENT = 'ENGLISH';
```

The above query calculates the total marks of all students in the ENGLISH department by summing up the values in the MARKS column.

Result:
SUM(MARKS): 162

2.2.9 Aggregation with grouping

The GROUP BY clause is used to group rows (tuples) in a table based on the values of one or more specified columns. Tuples with the same value in all the columns mentioned in the GROUP BY clause are combined into one group, and aggregate functions (like AVG, SUM, COUNT, etc.) are applied to each group. For example, to find the average marks of students in each department, the following query can be used:

Consider the following STUDENT table 2.2.11

Table 2.2.11 STUDENT table with attributes ID, NAME, SEMESTER, MARKS,

DEPARTMENT

ID	NAME	SEMESTER	MARKS	DEPARTMENT
1	RAM	1	90	BCA
2	GEORGE	3	95	HISTORY
3	KEERTI	3	90	ENGLISH
4	IRFAN	1	85	ECONOMICS
5	KOMAL	2	75	MALAYALAM
6	KIRAN	4	60	MATHEMATICS
7	NIRMAL	1	72	ENGLISH

Consider the following query:

```
SELECT DEPARTMENT, AVG(MARKS)
FROM STUDENT
GROUP BY DEPARTMENT;
```

This query calculates the average marks of students in each department using the AVG() function. The GROUP BY DEPARTMENT clause groups the rows by department, and the average of the MARKS column is calculated for each group.

Result:

Table 2.2.12 Result of aggregation with grouping

DEPARTMENT	AVG(MARKS)
BCA	90
HISTORY	95
ENGLISH	81
ECONOMICS	85
MALAYALM	75
MATHEMATICS	60

2.2.10 HAVING clause

The HAVING clause is used to specify conditions on groups created by the GROUP BY clause, typically in combination with aggregate functions like AVG, SUM, or COUNT. Unlike the WHERE clause, which filters rows before grouping, the HAVING clause filters groups after the aggregation. For example, to find the departments with an average mark greater than 90, use the following query:

```
SELECT DEPARTMENT, AVG(MARKS)
FROM STUDENT
GROUP BY DEPARTMENT
HAVING AVG(SALARY)>90;
```

Result:

Table 2.2.13 Result of Having clause

DEPARTMENT	AVG(MARKS)
HISTORY	95

Recap

- SQL: Most widely used query language
- SQL DDL: Commands for defining relation schemas, deleting relations, and modifying relation schemas.
- SQL DML: Query information, insert tuples , delete tuples from, and modify tuples in the database.
- SQL data type: Types of the attributes
- Basic SQL datatypes: Numeric, Character string, Bit string, Boolean, DATE, and TIME
- CREATE DATABASE command: Creates a new database.
- DROP DATABASE command: Deletes an existing database.
- SHOW DATABASES: To check available databases
- CREATE TABLE command: Creates a table in a database.
- DROP TABLE command: Delete a table in a database.
- INSERT: Add rows in a table
- SELECT: Retrieve information from databases.
- DELETE: Remove rows in a table.
- UPDATE: Modify rows in a table.
- The LIKE comparison operator: String pattern matching.
- ORDER BY clause: To order the tuples in the result of a query
- Built-in aggregate functions: AVG, MIN, MAX, SUM, COUNT
- GROUP BY clause: Group a set of tuples.
- HAVING clause: Specify conditions to groups.

Objective Type Questions

1. Which SQL command creates a new database?
2. Which SQL command deletes an existing database?
3. Which SQL command checks available databases use the command?
4. Which SQL command selects a database from available databases?
5. Which SQL command creates a table in a database?
6. Which SQL command is used for deleting a table in a database?
7. Which SQL command is used to add rows into a table?
8. Which SQL command is used for retrieving information from databases?
9. Which SQL command is used for removing rows in a table?
10. Which SQL command is used for modifying the rows in a table?
11. Which SQL comparison operator is used for string pattern matching?
12. Which SQL clause is used to order the tuples in the result of a query?
13. Which SQL clause is used to group a set of tuples?
14. Which SQL clause is used to specify conditions to groups?

Answers to Objective Type Questions

1. CREATE DATABASE
2. DROP DATABASE
3. SHOW DATABASES
4. USE
5. CREATE TABLE
6. DROP TABLE
7. INSERT
8. SELECT
9. DELETE
10. UPDATE
11. LIKE
12. ORDER BY
13. GROUP BY
14. HAVING

Assignments

1. Explain the difference between DDL (Data Definition Language) and DML (Data Manipulation Language) commands in SQL. Use the EMPLOYEE table below to perform the following:

    - Create the table using CREATE TABLE.
    - Insert 3 rows into the table using INSERT.
    - Delete one record using DELETE.

ID	NAME	DEPARTMENT	SALARY
1	HELEN	HR	50000
2	JAMES	IT	70000
3	ALICE	FINANCE	60000

2. Describe various SQL data types with examples. Create a table called COURSES using appropriate data types to store the following data, and explain the structure:
3. Explain DML & DDL commands.
4. Explain different constaints database creation.

Suggested Reading

1. Teate, Renee MP. *SQL for Data Scientists: A Beginner's Guide for Building Datasets for Analysis*. John Wiley & Sons, 2021.
2. Groff, James R., Paul N. Weinberg, and Andrew J. Oppel. *SQL: the complete reference. Vol. 2*. McGraw-Hill/Osborne, 2002.
3. Beaulieu, Alan. *Learning SQL: master SQL fundamentals.* O'Reilly Media, 2009.
4. CELKO, J., and S. Q. L. Joe Celko's. *"for Smarties: Advanced SQL Programming.* 3. vyd. Massachusetts." (2005).

Reference

1. geeksforgeeks.org/pl-sql-tutorial/
2. archive.nptel.ac.in/courses/106/105/106105175/
3. tutoxialspoint.com/plsql/index.htm

Built-in Functions

Learning Outcomes

The learner will be able to:

- introduce the concepts of built in functions in SQL
- make aware of the need of built in functions in SQL
- describe how to use built in functions to manipulate the query results
- familiarize with converting data into desired formats using built in functions

Prerequisites

Built-in functions in a database offer several key benefits that make data management efficient and convenient. They simplify complex operations by providing predefined solutions for common tasks such as calculations, data formatting, and string manipulation, reducing the need for lengthy code. These functions are optimized by the DBMS, ensuring faster execution and better performance. Additionally, they help maintain code consistency and accuracy since they are well-tested and reliable. Using built-in functions enhances the readability and maintainability of queries, making them easier to understand and modify. Overall, they save time, reduce errors, and improve productivity in database operations.

Keywords

Aggregate function, CONCAT() , YEAR(), LENGTH(), Timestamp

Discussion

2.3.1 Introduction to Built-in Functions in SQL

A **Built-in Function** in a database is a **predefined function provided by the Database Management System (DBMS)** to perform common tasks easily and efficiently. These functions simplify operations like calculations, data manipulation, formatting, and summarizing data without the need to write custom code.

Features of Built-in Function are

- Provided by the DBMS.
- Easy to use, reliable, and efficient.
- Used in SQL queries.
- Categories include: Aggregate functions, String functions, Numeric functions, Date functions, etc.

SQL has a number of built-in functions. MONTH() and YEAR() take a DATE or TIMESTAMP value as their input and return an integer that is the month or year portion of the value. The following query list the name and month of admission date of students in STUDENT database.

```
SELECT NAME, MONTH(ADMISSION_DATE)

FROM STUDENTS;
```

Built in functions are used for data reformatting. TO CHAR function takes a DATE datatype and a format specification and return the formatted version of the date. Consider the following query.

```
SELECT NAME, TO_CHAR(ADMISSION_DATE, 'DAY MONTH DD, YYYY) FROM STUDENTS;
```

The results produced by the query will have admission dates in the format 'Mon- day January 01, 2021'.

2.3.1.1 Importance of built in functions for data Manipulation and Querying

Built-in functions are incredibly valuable tools for working with data. They provide ready-made solutions for handling common tasks like cleaning, transforming, and analyzing data, which can save a lot of time and effort. For example, in Python, functions like len(), sum(), and min() allow you to quickly calculate the length of a list, the total of numbers, or the smallest value without having to write extra code. Similarly, in databases like SQL, functions such as COUNT(), AVG(), and GROUP BY make it simple to summarize and organize data during queries.

Using built-in functions also makes your code easier to understand and maintain. Since these functions are well-documented and widely used, others who read your code

can easily follow what you've done. For instance, Python's pandas library includes tools like merge() for combining data, groupby() for aggregating it, and pivot() for reshaping it—all of which make data manipulation more straightforward. These functions allow you to perform complex operations in a clear and concise way, which is especially useful when analyzing data or creating reports.

Another major benefit of built-in functions is that they are optimized for performance. Instead of writing your own algorithms, you can rely on these pre-designed tools, which are often faster and more efficient. For example, SQL functions like JOIN and WHERE help you filter or combine data in ways that are both powerful and quick. Similarly, libraries like NumPy in Python include vectorized functions such as mean() and std() that can process large datasets much faster than traditional loops. This efficiency is particularly important when working with big data, where speed can make a huge difference.

2.3.2 Scalar Functions

Scalar functions in DBMS are predefined functions that operate on a single value (one row at a time) and return one single value as a result. They are used for string manipulation, mathematical operations, date processing, and type conversion.

Example:

STUDENT Table2.3.1

Student ID	Name	Marks	City	DOB
1	John	85	New York	2000-05-14
2	Alice	92	Chicago	1999-11-22
3	Bob	76	Boston	2001-03-10
4	Carol	89	Dallas	2000-09-05

Types of Scalar Functions:

2.3.2.1 String Functions

String functions in DBMS are built-in functions used to manipulate and perform operations on string (text) data. They help in modifying, analyzing, and formatting string values in a database.

a. UPPER() → Converts text to uppercase

```
SELECT UPPER(Name) FROM STUDENT;
```

Output:

UPPER(Name)
JOHN
ALICE
BOB
CAROL

b. LENGTH() → Returns length of string

```
SELECT Name, LENGTH(Name) FROM STUDENT;
```

Output:

Name	LENGTH(Name)
John	4
Alice	5
Bob	3
Carol	5

c.CONCAT() → Combines two strings

```
SELECT CONCAT(Name, ' from ', City) AS Student_Info FROM
STUDENT;
```

Output:

Student_Info
John from New York
Alice from Chicago
Bob from Boston
Carol from Dallas

2.3.2.2 Numeric Functions

Numeric functions are built-in functions in DBMS used to perform mathematical operations on numeric data types like integers, decimals, and floats. These functions help in calculations, rounding, and other number-related tasks within SQL queries.

a. ROUND() → Rounds numeric value

```
SELECT Marks, ROUND(Marks/7, 2) AS GPA FROM STUDENT;
```

Output:

Marks	GPA
85	12.14
92	13.14
76	10.86
89	12.71

b. MOD() → Finds remainder

```
SELECT Marks, MOD(Marks, 2) AS Even_Odd FROM STUDENT;
```

Output:

Marks	Even_Odd
85	1
92	0
76	0
89	1

2.3.2.3 Date Functions

a. YEAR() → Extracts year from date

```
SELECT Name, YEAR(DOB) AS Birth_Year FROM STUDENT;
```

Output:

Name	Birth_Year
John	2000
Alice	1999
Bob	2001
Carol	2000

b. DATEDIFF() → Difference between two dates

```
SELECT Name, DATEDIFF(CURDATE(), DOB) AS Age_Days FROM
STUDENT;
```

This shows how many days old each student is.

2.3.2.4 Conversion Functions

a. CAST() → Converts data type

```
SELECT CAST(Marks AS CHAR) FROM STUDENT;
```

Converts numeric `Marks` to string.

2.3.3 Aggregate Functions

Aggregate functions in DBMS are **predefined functions** that perform **calculation or summarization on a group of rows (multiple values)** and return a **single result**. They are mostly used with **GROUP BY** clause in SQL to group data and perform operations on each group.

Example:

EMPLOYEE Table 2.3.2

Emp ID	Name	Department	Salary	Age
1	John	HR	50000	30
2	Alice	IT	70000	28
3	Bob	IT	60000	35
4	Carol	HR	55000	40
5	David	Finance	75000	45

Common Aggregate Functions:

1. COUNT() → Counts number of rows

```
SELECT COUNT(*) FROM EMPLOYEE;
```

Output: 5 (Total number of employees)

2. SUM() → Adds up values in a column

```
SELECT SUM(Salary) FROM EMPLOYEE;
```

Output: 310000 (Total salary: 50000+70000+60000+55000+75000)

3. AVG() → Calculates average value

```
SELECT AVG(Salary) FROM EMPLOYEE;
```

Output: 62000 (Total salary / Number of employees = 310000 / 5)

4. MIN() → Finds minimum value

```
SELECT MIN(Age) FROM EMPLOYEE;
```

Output: 28 (Youngest employee's age)

5. MAX() → Finds maximum value

```
SELECT MAX(Salary) FROM EMPLOYEE;
```

Output: 75000 (Highest salary)

2.3.3.1 Using Aggregate Functions with GROUP BY

We can also group data and apply aggregate functions per group.

Example: Calculate total salary per department:

```
SELECT Department, SUM(Salary)
FROM EMPLOYEE
GROUP BY Department;
```

Output:

Department	SUM(Salary)
HR	105000
IT	130000
Finance	75000

Recap

- Built-in functions in SQL help simplify common data tasks and save time.
- These functions are already available in SQL, so no need to write complex code.
- The COUNT() function is used to determine how many rows are in a table.
- COUNT() can also be used to count specific records that meet certain conditions.
- The SUM() function adds the values in a numeric column.
- SUM() is useful for calculating totals, like total sales or expenses.
- The AVG() function calculates the average value of a numeric column.
- AVG() helps determine things like the average salary or score.
- String functions like CONCAT() join two or more text strings together.
- CONCAT() makes it easy to combine first and last names into a full name.
- Date functions such as CURRENT_DATE() retrieve the current date.
- Numeric functions perform calculations on numbers, like ROUND for rounding numbers and MOD for finding remainder.
- Date functions work with date and time values, such as YEAR for extracting the year from date.
- Conversion functions allow data type conversions, like CAST

Objective Type Questions

1. Give an example of a string function in SQL
2. What is the purpose of numeric functions in SQL?
3. Name a function used to find the difference between two dates in SQL.
4. Give an example of an aggregate function.
5. What does the COUNT() function do in SQL?
6. Can aggregate functions be used without GROUP BY?
7. What function is used to find the highest value in a column?
8. How does the AVG() function help in data analysis?
9. What function is used to convert data from one type to another?
10. Which function is used to find the lowest value in a column?

Answers to Objective Type Questions

1. CONCAT()
2. They perform mathematical calculations on numeric data.
3. DATEDIFF()
4. SUM()
5. It counts the number of rows in a table
6. Yes, but it is typically used for overall dataset calculations.
7. MAX()
8. It calculates the average of numeric values in a column.
9. CAST()
10. MIN()

Assignments

1. Explain the role of built-in SQL functions in data manipulation. Discuss how functions like COUNT(), SUM(), and AVG() simplify the process of data aggregation and analysis. Provide examples of how these functions can be applied in real-world scenarios, such as business analytics or financial reporting, to demonstrate their importance in extracting valuable insights from data.
2. Analyze the practical applications of string and date functions in SQL. How can functions like CONCAT() and CURRENT_DATE() be used to manipulate text and date values in a database? Explain the importance of these functions in organizing and presenting data effectively, especially when handling customer records or transaction data.
3. Built-in functions in SQL provide powerful tools for data transformation and analysis. Describe the different categories of built-in functions available in SQL, such as String Functions, Numeric Functions, Date & Time Functions, Aggregate Functions, and Conversion Functions. For each category, explain at least two commonly used functions with example queries and their expected outputs. Additionally, explain a real-life scenario where each category of functions can be applied to solve a data-related problem in a database system
4. Discuss the importance of grouping data in SQL and its relationship to aggregation functions. How does grouping data help in summarizing large datasets? Analyze the relationship between grouping data and aggregation functions such as SUM(), AVG(), and COUNT(). Provide a detailed example of how grouping can be used to analyze sales data by different regions or product categories.
5. Critically assess the role of built-in SQL functions in improving data efficiency and query performance. Explain how SQL built-in functions, like aggregation and conversion functions, enhance the efficiency of querying and managing large datasets. What are the advantages of using these functions over manually processing data, especially in terms of performance and time savings in large-scale databases?

Suggested Reading

1. Teate, Renee MP. *SQL for Data Scientists: A Beginner's Guide for Building Datasets for Analysis*. John Wiley & Sons, 2021.
2. Groff, James R., Paul N. Weinberg, and Andrew J. Oppel. *SQL: the complete reference. Vol. 2*. McGraw-Hill/Osborne, 2002.
3. Beaulieu, Alan. *Learning SQL: master SQL fundamentals.* O'Reilly Media, 2009.
4. CELKO, J., and S. Q. L. Joe Celko's. *"for Smarties: Advanced SQL Programming.* 3. vyd. Massachusetts." (2005).

Reference

1. geeksforgeeks.org/pl-sql-tutorial/
2. archive.nptel.ac.in/courses/106/105/106105175/
3. tutoxialspoint.com/plsql/index.htm

Views and Transaction Control Statement

Learning Outcomes

The learner will be able to;

- define a view in database
- describe the usage of views in SQL
- make the student aware of transactions in DBMS
- familiarize with various transaction control commands.

Prerequisites

Till now we have assumed that we are operating on the actual relation of the database. But it is not desirable for all users to see the real database. For example, in a UNIVERSITY database, it is not desirable for the student to see the salary details of the instructor. Due to these concerns, SQL provides a way of operating with a virtual relation defined by a query called a view.

Suppose a user wants to store a personalized collection of relations that is different from the actual relations in a table. For example, in a UNIVERSITY database an INSTRUCTOR would like to store the pass percentage of the subjects that he has taught for the previous 10 years. To get the result, the INSTRUCTOR should create a new relation by executing a query. Rather than creating a new relation in the database for the above query, a view can provide a virtual relation that contains the result of the query.

Keywords

WITH CHECK OPTION, ACID Properties, Consistency, Durability, Constriants

Discussion

2.4.1 Introduction to Views

A view in a database is like a window that shows a specific part of the data from one or more tables. It doesn't actually store data itself but instead holds a saved query that gets the data when needed. Think of it as a custom way to look at the data without changing the original tables. Views make it easier to work with complex data by simplifying how we access it, and they can also protect sensitive information by allowing access to only certain parts of the data.

To create a view, we use a special SQL command that tells the database what kind of data to show. Views are useful because they help organize and present data in a way that fits a specific need, without altering how the data is stored. There are different types of views: simple views that pull data from a single table, and complex views that combine information from multiple tables. Some views allow us to make changes to the data they show, while others only allow viewing without editing.

2.4.1.1 Purpose of Views in SQL

A view in SQL is like a virtual table that combines data from one or more real tables. It's created using a simple query, and it serves several important purposes.

1. Simplifies Complex Queries: Views make it easier to work with complicated queries. Instead of writing long and complex SQL commands every time, you can just create a view once and use it whenever needed.
2. Improves Security: Views help protect sensitive data. For example, if you don't want certain users to see all the information in a table, you can create a view that only shows the necessary data, keeping other data hidden.
3. Provides Abstraction: Views allow users to access data without knowing the details of the database structure. This makes it easier to work with the data because users don't need to understand the complexity behind it.
4. Makes Data Reusable: Once a view is created, it can be used in multiple places. This saves time and effort since you don't have to write the same query repeatedly.

2.4.1.2 How Views are Different from Tables

A table is like a storage box in a database where data is kept in rows and columns. It holds actual information, such as names, numbers, or dates. You can change or add new data directly in a table.

A view, however, is like a window that shows data from one or more tables. It doesn't store the data itself but simply displays it based on a query (or question) you ask the database. For example, a view might show only certain columns from a table or only specific rows of data. Unlike tables, you can't directly change data in a view, but you can see it as if it were a table. Views are useful for simplifying complex information or protecting sensitive data by showing only what's necessary.

In short:

- Table: Stores real data and allows changes.
- View: Shows data from tables, but doesn't store it and is mostly for viewing.

2.4.2 Creating Views

A view is a SQL query that is permanently stored in a database and assigned a name. To the database user, a view looks like a real table. But the view does not exists in a database. The rows and columns in a view are the results of the query that defines the view.

2.4.2.1 Syntax for Creating a View

To define a view, we must give the view a name and must state the query that computes the view. The form of the create view command is:

create view v as <query expression>;

where v is the view name and query expression is any logical query.

2.4.2.2 Example of Creating a Simple View

Consider a student who want to know the courses taken by a faculty but not authorized to access the salary details of the faculty in an INSTRUCTOR table. A view named FACULTY can be given to the student.

CREATE VIEW FACULTY AS

SELECT ID, NAME,DEPARTMENT, COURSES FROM INSTRUCTOR;

2.4.2.3 Using Views in SQL

After defining the view, the view name can be used to refer to the relation generated by the view. Suppose, we need to find the DEPARTMENT of the faculties in the FACULTY view, use the following query.

SELECT NAME,

DEPARTMENT FROM

FACULTY;

The relation generated by a view is the result of the evaluation of the query that defines the view. If the relation defining a view are modified, then the view be comes out of date. To prevent this views are implemented as follows. When we define a view, the database stores the definition of the view. When a view relation is used in a query, its is replaced by the query expression. Whenever the query is executed, the view is recomputed.

2.4.3 Update of a View

If the database is expressed in terms of a view, then the modifications to it might cause some problems. All the insertions, deletions, and updates in a query using view should be translated to a modification in the actual table.

An SQL view is said to be updatable (that is, inserts, updates or deletes can be applied on the view) if the following conditions are all satisfied by the query defining the view:

- The from clause has only one database relation.
- The select clause contains only attribute names of the relation, and does not have any expressions, aggregates, or distinct specification.
- Any attribute not listed in the select clause can be set to null; that is, it does not have a not null constraint and is not part of a primary key.
- The query does not have a group by or having clause.

Suppose we want to update the department of the faculty named TOM to ENGLISH in the FACULTY view use the following query. The query satisfies all the above conditions, therefore the the view will be updated and the update will be reflected in the actual table i.e. INSTRUCTOR.

```
UPDATE FACULTY
SET DEPARTMENT="ENGLISH" WHERE NAME = "TOM";
```

2.4.3.1 Inserting and Deleting rows in a view

The rules for updating the view applies to INSERT and DELETE operations on a view. For example to delete a row in FACULTY view with ID=3 use the following query.

```
DELETE FROM FACULTY WHERE ID=3;
```

This query will delete a row from the actual INSTRUCTOR table and it will be reflected in the view FACULTY.

WITH CHECK OPTION

WITH CHECK OPTION at the end of the view definition ensures that the insertions, deletions, and updates satisfies the WHERE clause in the view definition. If a tuple inserted doesn't satisfies the WHERE condition of the view with WITH CHECK option, then the insertion is rejected. Updates and deleted are similarly rejected if they don't satisfies the WHERE clause conditions in view. The follow- ing query is an example of a view with WITH CHECK option.

```
CREATE VIEW FACULTY AS SELECT ID, NAME, DEPARTMENT FROM
INSTRUCTORS

WHERE DEPARTMENT IS NOT NULL WITH CHECK OPTION;
```

The WITH CHECK option in the above query denies the entry of NULL values in DEPARTMENT column of the view.

2.4.4 Types of Views

In SQL, a view is a virtual table created by querying one or more base tables. It can simplify data retrieval by presenting data in a specific format or by hiding complex queries. There are different types of views, each serving a unique purpose depending on the complexity of the data being queried.

- **Simple view**

A simple view is a view based on a single table. It typically selects specific columns or rows from the table, providing a simplified version of the data. For example, if a database contains an employee table, a simple view might display only the employee names and salaries, excluding other details like employee ID or department. Simple views are easy to create and use, offering an efficient way to extract relevant data from one table.

- **Complex view**

A complex view, on the other hand, involves more than one table and usually includes joins. These views allow users to combine data from different tables, providing a more comprehensive view of related information. For instance, a complex view might join an employee table with a department table to display employee names along with their department names. Complex views can also include filtering, grouping, and aggregation to provide more detailed insights.

- **Materialized view**

Materialized views are a special type of view that stores the result of a query physically, rather than computing it every time the view is accessed. These views are particularly useful when dealing with large datasets, as they can improve performance by reducing the need to recompute data. However, materialized views need to be refreshed periodically to ensure the data remains up to date. They are commonly used in data warehousing environments, where quick access to large amounts of pre-aggregated data is essential.

2.4.5 Advantages and Disadvantages of Views

2.4.5.1 Advantages of Views

- **Simplifies Complex Queries:** Views make it easier to work with complex queries. You can write a complicated SQL query once, and then reuse it in different situations. This means you don't need to repeat the same logic each time you want to retrieve data.
- **Customizes Data Presentation:** Views allow you to show only the relevant data. For example, if you only need to see certain columns or rows, you can set up a view to display just that, making the data easier to understand and work with.
- **Enhances Security:** Views can help protect sensitive information by restricting access to only the necessary data. For instance, you can create a

view that shows employee names but hides their salaries, so users only see the information they are allowed to view.

2.4.5.2 Disadvantages of Views:

- **Read-Only in Some Cases:** Some views are read-only, meaning you can't make changes to the data directly through them. If you need to update information, you'll have to go back to the original tables, which can be a bit inconvenient.
- **Performance Issues:** If the view involves joining many tables or large datasets, it may slow down the process. Every time you access a view, the database has to process the query, which can take longer, especially with more complex data.
- **Data Freshness:** Regular views always show the latest data, but materialized views store data that needs to be manually refreshed. If you rely on materialized views, there might be a delay in seeing the most current information, which could be an issue in time-sensitive environments.

2.4.6 Introduction to Transaction Control Statements

Consider a bank transaction of transferring money from one account to another (say account X to account Y). This consists of following database updates.

- Debit the amount from account X
- Update the balance of account X
- Credit the amount to account Y
- Update the balance of account Y

The above updates must occur as a logical unit. If all the updates are not performed due to some error or system failure, the integrity of the database will be compromised. To address such problems SQL provides transaction processing features.

2.4.6.1 Definition of Transactions

A transaction is one or more SQL statements that together form a logical unit of work. The statement in transactions are related and perform independent actions. The statements in a transaction are part of a task and all of them are required to complete the task. Transactions have some standard characteristics referred to as ACID properties. The ACID properties of a transaction are:

- **Atomicity:** Either all operations in a transactions are performed or none of them are performed.
- **Consistency:** A transaction must transform from one consistent state to another.
- **Isolation:** Each transaction must execute on its own without interference from other transactions.

- **Durability:** Once transaction is completed, all changes made by it should be preserved.

2.4.6.2 Importance of Transaction Control in SQL

Transaction control in SQL plays a key role in making sure data stays accurate and consistent when multiple actions are happening at once. It allows you to bundle a series of SQL commands together into one single task. This way, all changes are either completed successfully or not at all. For example, in a bank transfer, if money is being taken from one account and added to another, transaction control makes sure both actions happen together. If something goes wrong, you can use the ROLLBACK command to cancel everything, keeping the data correct. Using transaction control helps to avoid mistakes, especially in busy systems where many users are working with the database at the same time.

2.4.6.3 Basic Transaction States (Started, Committed, Rolled Back)

In SQL, transactions go through three basic states: Started, Committed, and Rolled Back. The transaction begins in the Started state, where changes are made to the database, but they aren't saved yet. This is the stage where you can add, modify, or delete data, but nothing is final until you decide what to do next. If everything goes as planned and the changes are correct, the transaction moves to the Committed state. This means all the changes are permanently saved to the database and can be seen by other users. However, if something goes wrong or you need to undo the changes, the transaction can be Rolled Back. In this state, all the changes made during the transaction are canceled, and the database goes back to how it was before the transaction started. These three states help ensure that the database stays accurate and that errors can be fixed easily without affecting the overall system.

2.4.7 Transaction Control Statements

The commands used for transaction control are:

START TRANSACTION: Sets the properties of new transaction and starts the transaction

SET TRANSACTION: Sets the properties of the next transaction to be executed.

SET CONSTRAINTS: sets the constraint mode within a current transaction. The constraint mode controls whether a constraint is applied immediately to data as it is modified or enforcement of the constraint is to be deferred until later in the transaction

SAVEPOINT: Creates a savepoint within a transaction. A savepoint is a place within a transaction's sequence of events that can act as an immediate recovery point. A current transaction can be rolled back to the savepoint instead of the beginning of the transaction

RELEASE SAVEPOINT: Releases a savepoint, freeing up any resources it may be holding

COMMIT: Terminates a successful transaction and commits all changes to a

database

ROLLBACK: When used without a savepoint, terminates an unsuccessful transaction and roll back any changes to the beginning of the transaction, restoring the database to its consistent state before the transaction. When used with a savepoint, rollback the transaction to the named savepoint, but allows it to continue

2.4.8 Advantages and Disadvantages of Transaction Control

2.4.8.1 Advantages of Transaction Control

- **Maintains Data Integrity:** Transaction control helps keep your data accurate and consistent. If something goes wrong during a transaction, you can roll back the changes to make sure no incorrect data is saved in the database.
- **Ensures Atomicity:** With transaction control, all actions in a transaction are treated as one unit. This means that either all the changes will happen, or none of them will, ensuring your database remains in a valid state even if an error occurs.
- **Guarantees Consistency:** It ensures that the database always moves from one valid state to another. If an error happens, the transaction can be rolled back, preventing partial updates from causing problems.
- **Provides Isolation:** When one transaction is running, others can't interfere with it. This helps avoid data conflicts or inconsistencies when multiple users are working with the database at the same time.
- **Ensures Durability:** Once a transaction is committed, its changes are permanent. Even if the system crashes or there is a power failure, the changes made in the transaction are safe and won't be lost.
- **Handles Errors Easily:** If an error happens during a transaction, transaction control lets you undo the changes and revert to the original state. This makes managing errors easier and helps protect the database from issues.
- **Supports Multiple Users:** Transaction control makes it easier for many people to use the database at the same time without causing problems. Each transaction runs independently, so multiple users can work without affecting each other's data.
- **Improves Data Management:** It helps organize large sets of changes by grouping them into a single transaction. If an error happens, only that transaction is affected, not others, making it simpler to manage changes to the data.

2.4.8.2 Disadvantages of Transaction Control

- **Complexity in Management:** Using transaction control can make managing the database more complicated. In systems with many users and transactions, it can be tricky to keep track of when transactions start, when they are committed, or when they should be rolled back. This extra complexity can lead to mistakes.

- **Performance Overhead:** Transaction control requires extra processing to make sure changes are either saved or undone. This extra work can slow down the system, especially when handling large or complicated transactions, affecting the overall performance of the database.

- **Potential for Data Inconsistency:** If transactions are rolled back at the wrong time or in the wrong way, it can cause inconsistencies in the data. This is particularly risky in systems where multiple transactions happen at once, as it can confuse the system and cause errors in the data.

- **Resource Consumption:** To manage transactions, the system often needs to keep logs and other records to ensure it can undo changes if needed. This can use up a lot of system resources, like memory and storage, which can become an issue when there are a lot of transactions happening frequently.

- **Difficulty in Handling Deadlocks:** In busy systems with many transactions happening at the same time, deadlocks can occur, where two or more transactions block each other. Resolving these deadlocks and making sure the system keeps running smoothly can be difficult and requires extra resources.

2.4.10 Practical Applications of Views and Transaction Control

- **Simplifying Complex Queries:** Views are great for making complicated queries easier to use. Instead of writing the same complex query every time, you can save it as a view and reuse it whenever needed. This saves time and effort.

- **Data Security:** Views help keep sensitive information safe by showing only the data that's needed. For example, if you have employee data, you could create a view that shows only their names and roles, but hides their salary or personal information from unauthorized users.

- **Data Presentation:** Views allow you to organize data in a way that makes sense for the situation. For instance, you could create a view that shows sales figures by region or department, without overwhelming users with unnecessary details like individual transactions.

- **Enhancing Data Consistency:** Views help ensure that everyone is looking at the same data in the same way. They simplify how data is accessed, reducing errors that could happen if people wrote their own complex queries.

- **Transaction Control for Data Integrity:** Transaction control ensures that changes to the database are safe and reliable. With commands like COMMIT, ROLLBACK, and SAVEPOINT, you can make sure that if something goes wrong, changes can be undone and the database remains accurate.

- **Error Recovery:** If there's an error during a transaction, you can use ROLLBACK to undo the changes and restore the database to its previous state. This helps prevent mistakes from affecting the system.

- **Ensuring Atomicity in Operations:** Transactions make sure that a group

of changes happens together or not at all. If one part of the operation fails, the whole transaction is canceled, ensuring that the database stays in a good state.

- **Multiple Users and Concurrency:** When many people are working with the database at the same time, views and transaction control help ensure that their actions don't interfere with each other. This keeps everything running smoothly and prevents data conflicts.
- **Data Reporting:** Views are helpful for creating reports. They can gather and display data from different parts of the database in a way that's easy to read and understand, without changing the original data.
- **Transactional Workflow Management:** In systems like banking or inventory management, transaction control makes sure that important actions—like transferring money or updating stock levels—are done correctly. If something goes wrong, it ensures no partial changes are made, keeping everything consistent.

Recap

1. Views simplify complex queries by saving commonly used query logic.
2. Views help protect sensitive data by allowing controlled access.
3. They provide data in a user-friendly format, making it easier to understand.
4. Views ensure consistency by presenting the same data format to everyone.
5. Transaction control commands like COMMIT, ROLLBACK, and SAVEPOINT ensure reliable data operations.
6. COMMIT permanently saves changes made during a transaction.
7. ROLLBACK undoes changes if something goes wrong, returning the database to its original state.
8. SAVEPOINT allows partial rollbacks, letting you undo specific parts of a transaction.
9. Views make it easier to generate reports by combining data from multiple tables.
10. Views are reusable, saving time and reducing the need to rewrite complex queries.
11. They help in presenting specific subsets of data, reducing clutter for users.
12. Views can act as a security measure by exposing only necessary data to users.
13. Transaction control maintains the integrity of data when multiple users are accessing the database simultaneously.

14. Atomicity ensures that all operations in a transaction are completed successfully or none at all.
15. Views allow for efficient data presentation without altering underlying tables.
16. Using views reduces errors by standardizing how data is retrieved and displayed.
17. Transaction control helps manage multiple user actions, ensuring no conflicts arise.
18. Views allow users to work with summarized or filtered data without accessing raw tables.
19. Concurrency control in transaction management prevents one user's work from interfering with another's.
20. Views provide a quick and easy way to view specific datasets without direct interaction with the database.
21. Transaction control ensures smooth workflow in applications that require precise data management, like banking or inventory systems.

Objective Type Questions

1. What command is used to save changes in a transaction?
2. Which command is used to undo changes in a transaction?
3. What is the primary purpose of a view in SQL?
4. Which SQL command allows partial rollback of a transaction?
5. What feature of views helps limit user access to sensitive data?
6. Which type of data operation ensures that all changes in a transaction are completed or none at all?
7. What type of view stores the result physically?
8. In transaction control, what ensures data consistency when multiple users access the database?
9. What SQL clause is used to define the columns retrieved in a row?
10. What term describes combining multiple data sources into one view for reporting purposes?

Answers to Objective Type Questions

1. COMMIT
2. ROLLBACK
3. Simplification
4. SAVEPOINT
5. Security
6. Atomicity
7. Materialized
8. Concurrency
9. SELECT
10. Aggregation

Assignments

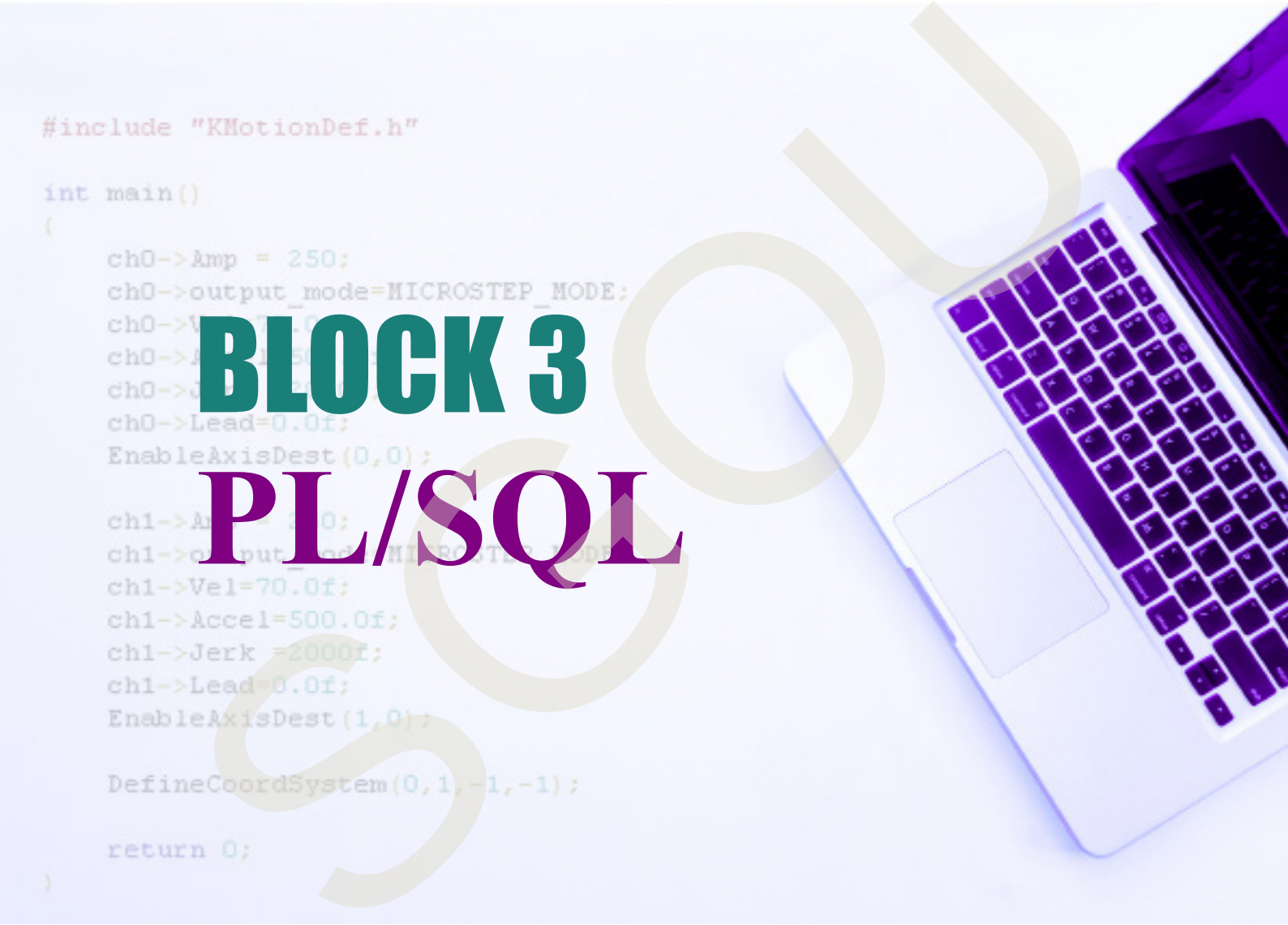
1. Explain how views can be used to simplify complex database queries. Discuss their role in improving data retrieval and user experience, and provide examples where views can enhance the efficiency of large-scale data systems.
2. Analyze the advantages and disadvantages of using views in SQL. How can views help with data security and access control? In what scenarios could the limitations of views affect their usability in a real-world application?
3. Discuss the role of transaction control statements like COMMIT, ROLLBACK, and SAVEPOINT in maintaining data integrity. How do these statements ensure that transactions are executed properly, especially in high-concurrency environments? Provide examples from database-driven applications where these controls are crucial.
4. Evaluate the impact of using materialized views in SQL for improving performance. Compare materialized views with regular views in terms of data refresh cycles, storage requirements, and application performance. When would materialized views be preferable?
5. Consider a scenario where multiple users are interacting with a database simultaneously. Explain how transaction control mechanisms like atomicity, consistency, isolation, and durability (ACID properties) ensure that the database remains in a consistent state. How do these properties apply to SQL views in multi-user environments?

Suggested Reading

1. Ramakrishnan, R., & Gehrke, J. (2002). *Database management systems*. McGraw-Hill.
2. Coronel, C., & Morris, S. (2019). *Database systems: design, implementation, and management*. Cengage Learning.
3. Silberschatz, A., Korth, H. F., & Sudarshan, S. (2006). *Distributed databases. In Database system concepts* (5th ed., pp. 705–749). McGraw-Hill.
4. Groff, J. R., Weinberg, P. N., & Oppel, A. J. (2002). *SQL: The complete reference* (2nd ed.). McGraw-Hill/Osborne.

Reference

1. geeksforgeeks.org/pl-sql-tutorial/
2. archive.nptel.ac.in/courses/106/105/106105175/
3. tutoxialspoint.com/plsql/index.htm



BLOCK 3

PL/SQL

Introduction to PL/SQL

Learning Outcomes

Studying this unit will enable the student to;

- Define PL/SQL and describe its purpose
- Make them Understand the basic structure of a PL/SQL block
- Define the concept of PL/SQL Programming
- Describe the features of PL/SQL
- Detail the PL/SQL code block
- List the control structures used in PL/SQL

Prerequisites

In today's world data is one among the most expensive resources. So it is important to study about how the data can be stored and retrieved effectively. Much of our data is now stored digitally. Almost all information on the Internet is stored on servers or databases operated by various companies. Using and manipulating this data requires an accessible and convenient medium. Structured Query Language (SQL) is one of the known query languages used for creating and manipulating databases. SQL streamlines the process of collecting, storing, and querying data. What if your code needs to do more complex loops and conditionals than the SQL programming language allows? This may exceed SQL's limit. PL/SQL allows you to do this.

Keywords

PL/SQL, Oracle, SQL, Datatype, Variable, Conditional Statement, LOOP

Discussion

Structured Query Language, popularly known as SQL (pronounced as SEQUEL), is a query language used to create and manipulate relational databases. In practical applications database access is implemented through software programs. These software programs are created using general purpose programming languages such as, C, C++, JAVA etc. SQL commands for database access are embedded in these programs. SQL is the natural language of relational database management systems.

3.1.1 Disadvantages of SQL

Although SQL is widely used, it does not support programming features like condition checking, looping, and branching. These features are important for verifying data integrity before storing it in a database.

In multi-user environments, SQL statements can slow down data processing. Each time an SQL query runs, the database engine is called, which can increase network traffic. SQL also lacks built-in error-handling mechanisms. If an error occurs, the database engine displays its own predefined error messages, which cannot be customized by the user.

All these disadvantages prevent SQL from being a fully structured programming language. Oracle, most popular relational DBMS, provides a fully structured programming language called PL/SQL.

3.1.2 Advantages of PL/SQL

- Provides the procedural capabilities like condition checking, branching and looping.
- The entire block of code is passed to the Oracle engine in one go. Hence the network traffic is reduced considerably.
- PL/SQL provides facilities for displaying user-friendly error messages.
- PL/SQL allows the use of variables to store results of queries.
- Applications written in PL/SQL can be executed in any platform, where Oracle is operational.

3.1.3 PL/SQL Block

A PL/SQL block consists of SQL statements organized in a structured manner. A **PL/SQL block** is the fundamental structure of a PL/SQL program, consisting of four main sections:

- The Declare section

- The Begin Section
- The Exception Section
- The End Section

The following Fig.3.1.1shows PL/SQL block structure

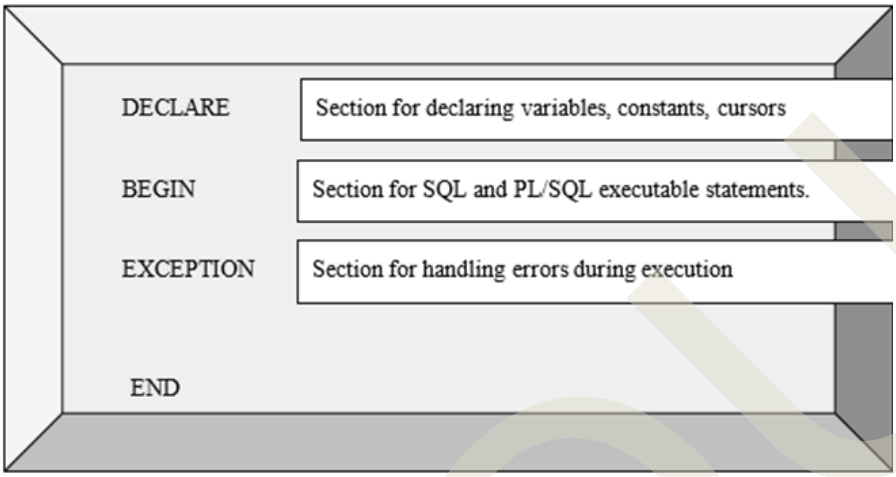


Fig 3.1.1 : PL/SQL block structure

PL/SQL engine resides in Oracle engine. Oracle engine can process the entire PL/SQL blocks. The Fig.3.1.2 shows the PL/SQL execution environment.

A PL/SQL block starts with the keyword **DECLARE** (if declarations are included) and ends with **END;.** This structure ensures modular, readable, and maintainable code.

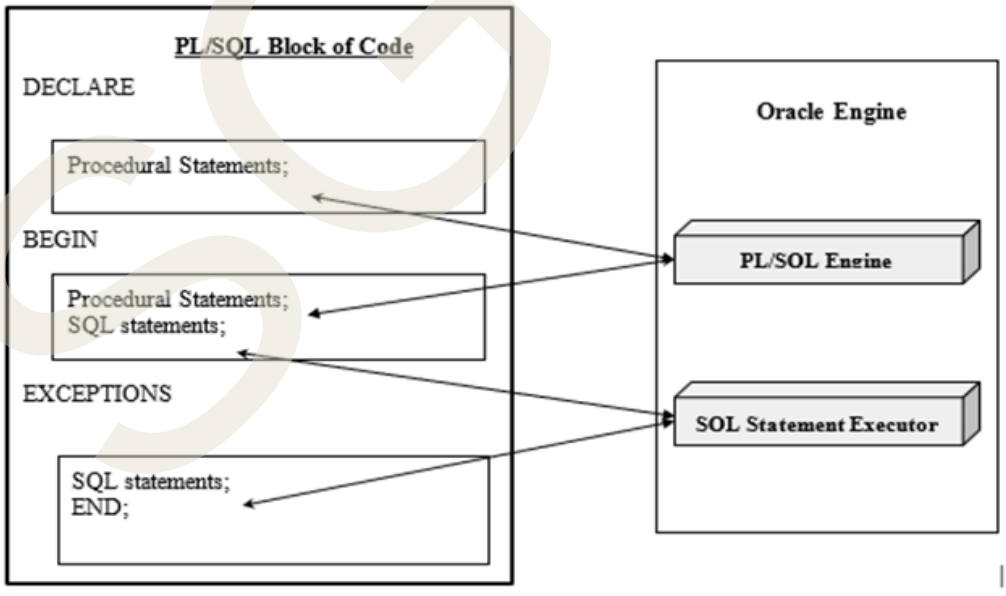


Fig 3.1.2 PL/SQL Block of Code

3.1.4 PL/SQL Character Set

The **PL/SQL character set** includes a collection of symbols, letters, and special characters that can be used in PL/SQL programs. It consists of:

1. **Letters:** Uppercase (A-Z) and lowercase (a-z) alphabets.
2. **Digits:** Numbers from 0 to 9.
3. **Symbols:** Special symbols like +, -, *, /, =, >, <, and more.
4. **Whitespace Characters:** Spaces, tabs, and newline characters for readability.

This character set ensures compatibility with SQL and facilitates smooth interaction with database objects.

3.1.5 PL/SQL Literals and Constants

In PL/SQL, **literals** are fixed values directly written in the code, such as numbers, strings, or dates, which do not change during execution. For example, **42**, **'Hello'**, and **SYSDATE** are literals. **Constants**, on the other hand, are named variables whose value is assigned once and cannot be modified later in the program. They are declared using the keyword **CONSTANT** and help ensure data integrity by preventing unintended changes. Both literals and constants are fundamental for defining fixed values in PL/SQL programs.

- Literal is a character string or numeric used to represent itself.
- The following table shows literals supported by PL/SQL

Table 3.1.1 Literals supported by PL/SQL

Literal Type	Example
Numeric - Integers or floats	37,1.2, 1.e5, 8g3
String-one or more characters enclosed in single quotes	'Hello', 'C', '*',
Boolean-Predetermined Constants	TRUE, FALSE, NULL
Date and Time literals	DATE'2022-01-01'; TIMESTAMP'2022-01-01 10:00:01;

3.1.6 PL/SQL Data Types

PL/SQL provides various **data types** to define variables, constants, and parameters, ensuring compatibility with database operations. These data types are categorized as follows:

1. **Scalar Data Types:** Represent single values such as numbers (NUMBER, INTEGER), characters (CHAR, VARCHAR2), and dates (DATE).
2. **Composite Data Types:** Store multiple values in collections or records, like

TABLE, VARRAY, and RECORD.

3. **LOB Data Types:** Handle large objects such as CLOB (Character Large Object) and BLOB (Binary Large Object).

4. **Reference Data Types:** Include pointers like REF CURSOR.

These data types provide flexibility for handling diverse data in PL/SQL programs. See the table

Table 3.1.2 PL/SQL Qata Types

Default Data Types	Number, Char, Date, Boolean
%TYPE	Declare variables based on definitions of columns in the database tables.
%ROWTYPE	Composite Data Type. Similar to structure in C language. Declare a record variable based on the database table.
BLOB-Binary Large Objects	Stores unstructured binary data upto 4GB.
PLS_INTEGER	Used for whole numbers in arithmetic operations. Not stored in the database.
Number(p,s)	Used for storing floatingpoint values 'p' represents the total number of digits allowed for the value 's' number of digits right to the decimal place.

3.1.7 DBMS_OUTPUT Package

The DBMS_OUTPUT package is used in PL/SQL to display messages from a program. It helps in debugging and understanding the flow of execution by showing output during or after program execution.

Messages written using DBMS_OUTPUT are temporarily stored in a buffer. Once the program finishes running, these messages are retrieved and displayed on the screen.

The package includes several procedures to manage and display output, enabling developers to effectively track and debug their PL/SQL code.

Procedures in DBMS_OUTPUT package are given below

Table 3.1.3 procedures in DBMS_OUTPUT package

Procedure	Purpose
DBMS_OUTPUT.ENABLE	Allow messages to display
DBMS_OUTPUT.DISABLE	Does not allow messages to display
DBMS_OUTPUT.PUT	Places information in the buffer
DBMS_OUTPUT.PUT_LINE	Places information in the buffer with an end-of-line marker
DBMS_OUTPUT.NEW_LINE	Places an end-of -line marker in the buffer

Examples of DBMS_OUTPUT Package

PL/SQL Block	OUTPUT

BEGIN
DBMS_OUTPUT.PUT('This is');
DBMS_OUTPUT.PUT_LINE (This is line 1.');
DBMS_OUTPUT.PUT('This is line 2.');
END;
	This is line one. This is line 1.
	This is line 2.

PL/SQL Block	OUTPUT

DECLARE
PI CONSTANT NUMBER:=3.14;
radius NUMBER(5, 2);
area NUMBER(10, 2);
 	
BEGIN
radius:=9.5;
area:=PI*radius*radius;
DBMS_OUTPUT.PUT_LINE('The area of the circle is '||area);
END;
/
	The area is 283.53
	PL/SQL procedure successfully executed

3.1.8 PL/SQL Basic Syntax

Example: Write a PL/SQL block to print HELLO WORLD

PL/SQL Block	OUTPUT

BEGIN
DBMS_OUTPUT.PUT_
LINE('HELLO WORLD')
END;
	HELLO WORLD

PL/SQL procedure successfully

'END;' marks the end of the block.

Activity 1:Write a PL/SQL block to print your name

Comments in PL/SQL

- Comments are used to improve the readability of code.
- In PL/SQL single line comments comments with '--';
- Block comments or multi line comments can be included using '/*...*/.

3.1.9 Variable Declaration in PL/SQL

Syntax:

variable_name [CONSTANT] datatype [NOT NULL]:=value;

where,

- variable_name : The name of the variable
- [CONSTANT]: Optional. If a variable is declared as CONSTANT the value cannot be changed.
- datatype- can be the default data type or %TYPE or %ROWTYPE.
- [NOT NULL]: Optional. Specified when the variable cannot have null values.
- value:value assigned to the variable'

Examples:

```
radius NUMBER(2);
Area NUMBER(5,2);
Pi CONSTANT NUMBER:=3.14;
Grade CHAR(1) DEFAULT 'A';
Marks NUMBER NOT NULL;
```

The above examples show various types of variable declarations. Default values can be assigned as shown in example 4.

- Every PL/SQL statement ends with a semicolon;
- ':=' is used as assignment operator;

Example: Write a PL/SQL block that finds the area and circumference of a circle.

The above example computes the area of a circle given the radius 9.5.

In PL/SQL input can be accepted from the user directly. Consider the following example.

PL/SQL Block	OUTPUT

DECLARE
PI CONSTANT NUMBER:=3.14;
radius NUMBER;
area NUMBER;
BEGIN
radius:=&radius;
area:=PI*radius*radius;
DBMS_OUTPUT.PUT_LINE('The area of the circle is '||area);
END;
/
	10
	The area is of the circle is 314
	PL/SQL procedure successfully executed

In the DECLARE section the variable radius is declared. In the BEGIN section, the value for variable is accepted from the user.

3.1.10 PL/SQL Operators

The following table shows different types of operators in PL/SQL

Table 3.1.4 Arithmatic Operators

Arithmetic Operations	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation a**b computer a raise to b

Table 3.1.5 Relational Operators

Relational Operators	Description
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
!= or<>	Not equal to
=	Equal to

Table 3.1.6 Special Operators

Operators	Description	Example
LIKE	Pattern Matching Compare a character or string to a pattern specified. Return TRUE if the value matches the pattern and FALSE if it does not	'Suma' like 'S%a' returns a Boolean true, because the string starts with S and ends with A. Otherwise it will return FALSE, '%' is used to match any number of characters.
BETWEEN	To specify a range	SALARY BETWEEN 10000 AND 50000
IN	To test whether an item is present in a list of values	If a='x' then, a in ('r', 's', 't') returns Boolean false but a in ('x', 'y', 'z') returns Boolean true.
IS NULL	To test whether the variable has Null value. Returns TRUE, if the value of a variable is NULL. Otherwise returns FALSE.	a is null

Table 3.1.7 Logical Operators

Logical Operator	Description
AND	Logical AND Returns TRUE only if both conditions are TRUE
OR	LOGICAL OR Returns TRUE if any of the conditions are evaluates to TRUE
NOT	Negation. Used to reverse the logical state of the operand

- '||' operator is used as concatenation operator.

3.1.11 PL/SQL Conditional Statement

Conditional statements in PL/SQL are used to make decisions based on specified conditions. Like other programming languages, PL/SQL uses the IF statement to evaluate conditions. These conditions are represented as boolean expressions, which return either TRUE or FALSE.

PL/SQL supports the following conditional constructs for decision-making:

- **IF-THEN**
- **IF-THEN-ELSE**

- **IF-THEN-ELSIF-ELSE**

Syntax and Explanation

IF-THEN: Executes a block of code if the condition evaluates to TRUE.

Syntax and Example:

Syntax	Sample Code
IF condition THEN	IF(a>b) THEN
Statements;	larger;=a;
END IF;	END IF;

5. If the condition evaluates to TRUE, the statements within the corresponding IF block will be executed.
6. If the condition evaluates to FALSE, the statements inside the IF block will be skipped. If there is an ELSE or ELSIF block, the program will evaluate those accordingly.

- **IF-THEN-ELSE**: Provides an alternative block of code to execute when the condition is FALSE. Here ELSE provides an alternative. If the condition specified in IF is FALSE, the ELSE part will be executed.

Syntax

IF condition THEN

-- statements if condition is true

ELSE

-- statements if condition is false

END IF;

IF-THEN-ELSIF-ELSE: Allows multiple conditions to be checked sequentially.

IF condition1 THEN

-- statements for condition1

ELSIF condition2 THEN

-- statements for condition2

ELSE

-- statements if none of the above conditions are true

END IF;

Example

```
DECLARE
grade NUMBER := 85;
BEGIN
IF grade >= 90 THEN
DBMS_OUTPUT.PUT_LINE('Grade: A');
ELSIF grade >= 75 THEN
DBMS_OUTPUT.PUT_LINE('Grade: B');
ELSE
DBMS_OUTPUT.PUT_LINE('Grade: C');
END IF;
END;
```

This PL/SQL block evaluates the grade variable and outputs the appropriate grade message based on the value.

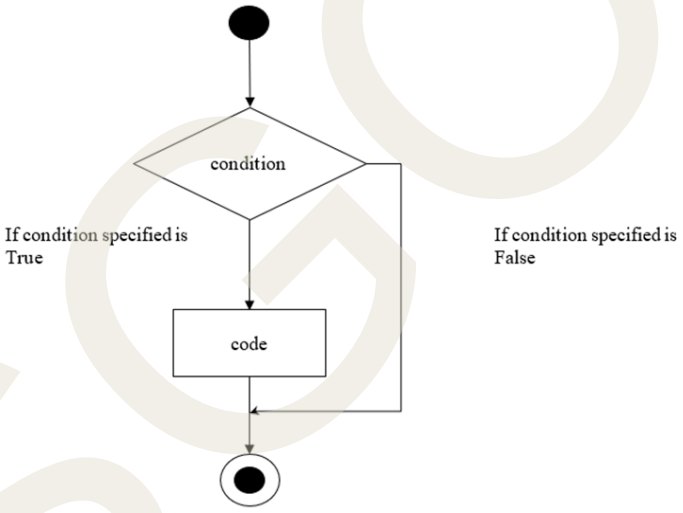


Fig. 3.1.13 Flow chart of Conditional Statement

The above flowchart(Fig. 3.1.13) represents the general form of conditions constructed in all programming languages. The general syntax for IF statement in PL/SQL is :

Syntax

```
IF<condition> THEN
        <PL/SQL statements>
ELSIF<condition> THEN
        <PL/SQL statements>
```

```
        ELSE
                <PL/SQL statements>
        END IF;
```

IF-THEN-ELSIF statement

- Provides the facility to choose between several alternatives.
- IF-THEN statement followed by an optional ELSIF-ELSE statement.

Syntax	Sample Code

IF condition1 THEN
Statements;
ELSIF condition2 THEN
Statements;
ELSIF condition3 THEN
Statements;
ELSE
Statements;
END IF;
	DECLARE
	color varchar2(6):='Red';
	BEGIN
	IF (color:='Green') THEN
	DBMS_OUTPUT.PUT_LINE:('Your car is Green');
	ELSIF:(color:='White') THEN
	DBMS_OUTPUT.PUT_LINE:('Your car is White');
	ELSIF:(color:='Red') THEN
	DBMS_OUTPUT.PUT_LINE:('Your car is Red');
	ELSE
	DBMS_OUTPUT.PUT_LINE:('Your car is not Red or Green or White');
	DBMS_OUTPUT.PUT_LINE:('Your car is in '||color||'color');
	END IF;
	END;
	/

- IF statements can be nested. An IF statement can be written inside another IF statement.

3.1.12 Loop Statements

Loops are used to execute a set of statements multiple times and are an essential control structure in all programming languages. They help automate repetitive tasks, reduce code redundancy, and improve program efficiency. A loop generally involves four key steps: initialization, where the starting point is set; a condition that determines whether the loop continues; execution of the loop body if the condition evaluates to TRUE; and an update that modifies variables to ensure the condition is eventually met. This structure ensures controlled and efficient repetition.

- In PL/SQL there are mainly three types of LOOPS:
  - Simple LOOP
  - FOR LOOP
  - WHILE LOOP

3.1.12.1 Simple LOOP

- A sequence of statements are enclosed between LOOP and END LOOP statements.
- Syntax and Example

Syntax	Sample Code

LOOP
 	
-----Sequence of Statements
 	
EXIT WHEN condition;-----condition to terminate the LOOP
 	
END LOOP
	DECLARE
	i number:=0;
	BEGIN
	LOOP
	i:=i+2;
	
	EXIT WHEN i>10;
	END LOOP
	
	Output
	The value of i is 10

In the above program variable i is initialized as 0. In each iteration the value of i is incremented by 2. The loop will be terminated as soon as the value i is greater than 10. The statement after the loop will display the final value of i ie.,10.

3.1.12.2 FOR LOOP

- Executes the statements inside the loop for a specific number of times.

Syntax	Sample Code

FOR Variable IN
 	
start.... end
LOOP
Statements;
END LOOP;
	DECLARE
	i number:=0;
	BEGIN
	FOR i IN 0..5
	LOOP
	DBMS_OUTPUT.PUT_LINE(i);

```
        END LOOP;
END;
```

Output

0
1
2
3
4
5

PL/SQL procedure completed successfully

The above program displays the value of i in each iteration. Initially the value of i is 0. So 0 will be displayed. In the next iteration the value of i is incremented by 1. So 1 will be displayed. Likewise the values of i will be displayed till i reaches 5. After that the loop will be terminated.

- The variable in the FOR LOOP need not be declared.
- The loop variable is incremented by 1.

3.1.12.3 FOR LOOP REVERSE

Syntax	Sample Code

FOR Variable IN REVERSE
 	
start---end
LOOP
Statements;
END LOOP;
	DECLARE
	i number(2);
	BEGIN
	FOR i IN REVERSE 1.....10
	LOOP
	DBMS_OUTPUT.PUT_LINE(i);
	END LOOP;
	END;
	/

Output

10
9
8
7
6
5
4
3
2
1

PL/SQL procedure completed successfully

In PL/SQL, the **REVERSE** keyword is used to reverse the order of iteration in a loop. When using the FOR loop with the **REVERSE** keyword, the loop variable starts from the highest value and decrements with each iteration until it reaches the lowest value.

The **REVERSE** keyword causes the loop to iterate in descending order, starting from 10 and ending at 1. During each iteration, the loop variable i is decremented automatically. The above program displays the value of i in each iteration in the reverse order.

3.1.12.4 WHILE LOOP

- This type of loop is commonly known as a WHILE loop in PL/SQL and other programming languages. It is used in situations where the number of iterations is not known beforehand. The loop continues to execute as long as the specified condition evaluates to TRUE. Unlike a FOR loop, the loop variable in a WHILE loop needs to be explicitly incremented or decremented within the loop body to ensure proper control of the loop.

Syntax and Example

Syntax	Sample Code

WHILE condition
LOOP
Statements;
ENDLOOP
	DECLARE
	i number:=0;
	BEGIN
	WHILE i<5
	LOOP
	DBMS_OUTPUT.PUT_LINE(i);
	i=i+1;
	END LOOP;
	END;
	/

Output

0
1
2
3
4
5

PL/SQL proceedure completed successfully

In the above example i is initialized to 0. When the iteration starts, the value of i is checked. i<5 is TRUE. The value of i is displayed and i is incremented by 1. The new

value of i is 1. Again the condition is checked and i<5 holds TRUE. The value of i is displayed as 1. i is incremented by 1. The new value of i is 2. The loop continues as long as i<5 holds TRUE. Once the value i is greater than 5, the loop will be terminated.

Anchored Declaration: %TYPE

- Used to declare variables based on definitions of columns in the database tables.
- Usage 1: Declare variables that directly map to column definitions in the database. Hence datatype need not be specified explicitly. Constraints will not be copied.
    - Syntax: variable_name TABLENAME COLUMN NAME % TYPE;
    - Example:ename EMPLOYEE.NAME%TYPE;

Here EMPLOYEE is a database table, NAME is a column in EMPLOYEE. The data type of NAME is applied to ename. Any change in the data type or precision for the NAME column in the EMPLOYEE table will be applied automatically to the variable anchored.

Usage 2: To declare a variable whose data type that directly maps to a datatype of a previously declared variable.

The data type and NOT NULL constraint of name is applied to the variable ename. If the value is assigned to the name variable, it will not be copied.

Composite Data type %ROWTYPE

- Used to store multiple column values in a single name.
- Similar to the concept of structure in C language.
- Allows to declare a record variable based on the column definitions in the database table.
    - Syntax: recordvariablename tablename%ROWTYPE;
    - Example:emprec EMPLOYEE%ROWTYPE;

The above declaration creates a record variable for the EMPLOYEE table in the database. Each column in the table can be accessed using the record variable.

Using SELECT statement in PL/SQL

- Values from database tables can be fetched to PL/SQL variables using the SELECT INTO statement.
- Only one row value can be returned to the variable.
- If no row is returned then the 'No Data Found' exception is thrown.
- If more than one record is returned, then the exception 'Exact fetch returns more than requested number of records' is thrown.

Example: An EMPLOYEE table is given below.

Table 3.1.8 Employee Table

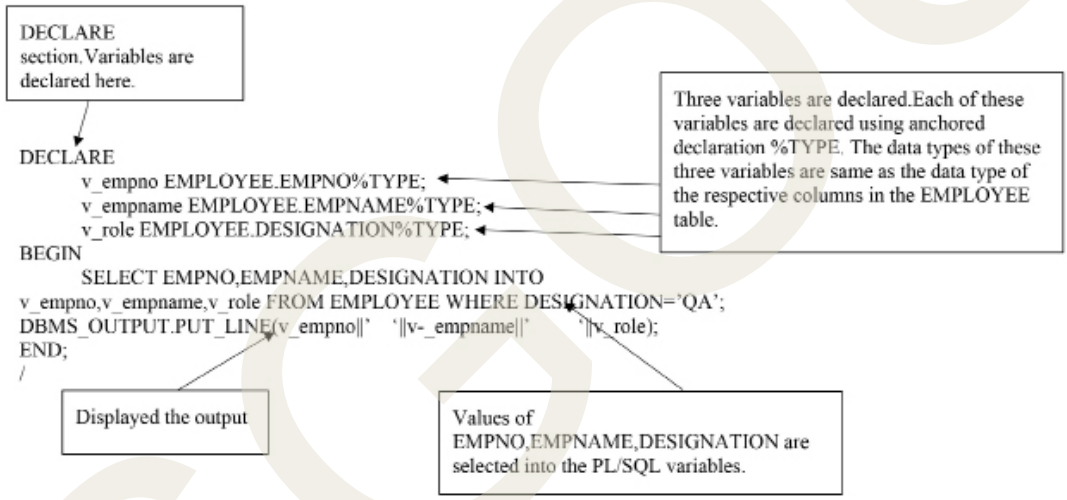
E M P NO	EMP NAME	DESIGNATION	STATE	SALARY	DEPT NO
E0001	JACOB	SYSADMIN	MAHARASHTRA	16500	D0001
E0002	JOHN	MANAGER	TAMILNADU	32000	D0002
E0003	ABRAHAM	DEVELOPER	KARNATAKA	26500	D0002
E0004	JOY	QA	TAMILNADU	19560	D0001
E0005	CHRISTY	SECRETARY	MAHARASHTRA	13200	D0003

Our aim is to select the name and employee number QA

The actual SQL query will be:

SELECT EMPNO, EMPNAME, DESIGNATION FROM EMPLOYEE WHERE DESIGNATION="QA";

Let us display the details of the particular employee using a PL/SQL block.



OUTPUT

E0004 JOY QA

PL/SQL procedure successfully completed.

Recap

- SQL lacks programming capabilities
- PL/SQL is introduced by Oracle as an extension to SQL.
- PL/SQL provides the facilities to include major programming constructs like condition checking, looping, procedures.
- A PL/SQL code consists of the following sections: DECLARE, BEGIN, EXCEPTION, END.
- As in other programming languages PL/SQL also incorporates variables, data types, and different types of operators.
- Conditional execution can be implemented using IF.
- PL/SQL consists of three loops; Simple LOOP, FOR LOOP, WHILE LOOP.

Objective Type Questions

1. Which relational DBMS introduced PL/SQL?
2. What marks the beginning of a PL/SQL block?
3. How will the variables based on column definitions be declared in PL/SQL?
4. Which is the datatype used to store binary data upto 4GB?
5. How many looping statements are there in PL/SQL?
6. What is the function of the exception section in a PL/SQL block?
7. Which loop is used if the number of iterations is known in advance?
8. What is the use of the REVERSE keyword?
9. What are the logical operators used in PL/SQL?
10. Which operator is used for pattern matching?

Answers to Objective Type Questions

1. Oracle.
2. DECLARE section
3. %TYPE

4. BLOB
5. 3. Simple LOOP, FOR, WHILE.
6. Deals with handling of errors.
7. FOR LOOP
8. To reverse the order of iteration from a higher value to a lower value.
9. AND, OR, NOT
10. LIKE

Assignments

1. Write a program PL/SQL block that adds two numbers and displays the result.
2. Write a PL/SQL program that finds the largest among three numbers.
3. Write a PL/SQL program to find the cube of a number.
4. Write a PL/SQL program to print the multiplication table of a number.

Suggested Reading

1. Bayross, I. (n.d.). SQL, *PL/SQL: The programming language of Oracle* (3rd ed.). BPB Publications.
2. Feuerstein, S. (n.d.). *Oracle PL/SQL programming.* O'Reilly Media.
3. McDonald, C., Katz, C., Beck, C., Kallman, J. R., & Knox, D. C. (n.d.). *Mastering Oracle PL/SQL: Practical solutions*. Apress Media.
4. Oracle Tutorial. (n.d.). *PL/SQL tutorial.* Retrieved March 5, 2025, from https://www.oracletutorial.com/plsql-tutorial/

Reference

1. geeksforgeeks.org/pl-sql-tutorial/
2. archive.nptel.ac.in/courses/106/105/106105175/
3. tutoxialspoint.com/plsql/index.htm

Cursors

Learning Outcomes

The learner will be able to;

- Define what a cursor is in PL/SQL
- Identify the difference between implicit and explicit cursors
- Describe how a cursor loop works
- Identify when to use cursors in PL/SQL
- Make them understand a basic PL/SQL code to declare and use a cursor

Prerequisites

Think of a restaurant where orders are served one at a time to customers seated at a table. The waiter collects the list of orders (like a query fetching data) and serves each dish individually to ensure everyone gets what they ordered. A **cursor** in PL/SQL works just like the waiter—it helps handle and process the rows in a query result set one at a time, ensuring that data is delivered and handled sequentially.

As a learner, you already know how SQL queries retrieve data from a table. But sometimes, we need to work with each row individually, such as calculating a total or applying specific logic row by row. Cursors make this possible by acting as a mechanism to fetch and process data one row at a time, just like the waiter carefully serving each dish.

The student must have an understanding about the DDL, DML and DCL commands in SQL. The students must be able to create and manipulate relational databases using the basic commands in SQL. The student must have an idea about the basic programming language concepts like variable, data type, functions, branching statements, and loops. The student must have an understanding of PL/SQL block, variables, conditional statements and loops. The student should know how to execute SQL statements, SELECT, INSERT, DELETE, UPDATE, in a PL/SQL block.

Keywords

Cursor, Active dataset, Implicit Cursor, Explicit Cursor.

Discussion

In PL/SQL, a cursor is a database object used to retrieve and process multiple rows of data from a query, one row at a time. It acts as a pointer that allows the program to navigate through the result set of a query, providing control over how data is fetched and processed. Cursors are especially useful when you need to handle complex queries or perform operations on individual rows. There are two main types of cursors in PL/SQL: implicit cursors, which are automatically created by the system for single SQL queries, and explicit cursors, which are manually declared and controlled by the programmer for more complex operations.

SQL statements commands operate the rows in the result in one go . Consider the following table EMPLOYEE.

Table 3.2.1 Employee Table

E M P NO	EMP NAME	DESIGNATION	STATE	SALARY	DEPT NO
E0001	JACOB	SYSADMIN	MAHARASHTRA	16500	D0001
E0002	JOHN	MANAGER	TAMILNADU	32000	D0002
E0003	ABRAHAM	DEVELOPER	KARNATAKA	26500	D0002
E0004	JOY	QA	TAMILNADU	19560	D0001
E0005	CHRISTY	SECRETARY	MAHARASHTRA	13200	D0003

If we want to update the salary of each employee by 10%, the following SQL query can be executed. .

```
UPDATE EMPLOYEE SET SALARY=SALARY+(SALARY*(10/100));
```

The above query updates all the values in the SALARY column with an increment of 10%. What if the percentage of increment is different for different values of salary? ie, Update SALARY by 10% whose current salary is between 10000 and 20000. Similarly, update SALARY by 15 whose current salary is between 21000 and 30000. In order to update salaries with different values, multiple SQL queries have to be executed. The rows are to be updated one by one by checking the condition. In such situations we can create a work area in Oracle, store the result set and update the rows one by one.

3.2.1 Cursor

For every query executed, Oracle creates a work space, known as the context area, for processing an SQL statement. The context area contains all the information needed for processing the SQL statement. For example, the number of rows processed. In PL/SQL this context area is called Cursor

A Cursor is a memory area used by the Oracle engine for its internal processing in order to execute SQL statements.

- Cursor points to the context area.
- The context area contains the following information:
  - rows returned by the SQL query.
  - number of rows processed by the SQL query.
  - A pointer to the parsed query.
- The data stored in the cursor is called Active Dataset.

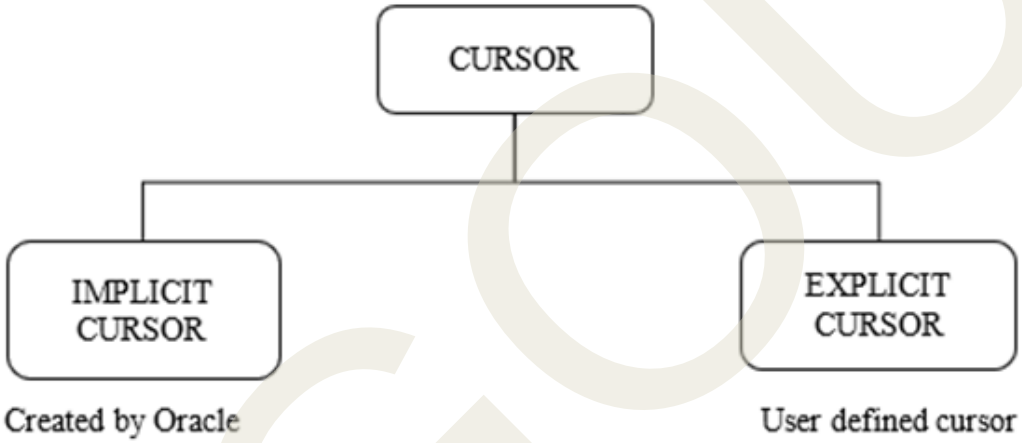


Fig.3.2.1 types of cursors

3.2.2 Implicit Cursor

An **implicit cursor** is automatically created and managed by the Oracle engine when executing SQL statements like SELECT INTO, INSERT, DELETE, or UPDATE. It is used to handle the execution status of these SQL statements, such as whether the operation was successful, how many rows were affected, or if there were any errors.

The entire execution cycle of implicit cursors—opening, fetching, and closing—is managed by the Oracle engine, so the programmer doesn't need to explicitly define or control it. Implicit cursors are ideal for simple SQL operations where complex row-by-row processing isn't required.

The following table shows Implicit Cursor attributes and their meaning

Table 3.2.2 Implicit Cursor Attributes

Implicit Cursor Attributes	Meaning
SQL%ROWCOUNT	Number of records affected by the most recent SQL statement. Returns how many are there in the result set of the SQL query.
SQL%FOUND	Returns TRUE if the most recent SQL query affects at least one row
SQL%NOTFOUND	Returns TRUE if the most recent SQL query does not affect any rows.
SQL%ISOPEN	Automatically opened by Oracle engine and closes immediately after the query is executed. Hence the value of the cursor is always FALSE.

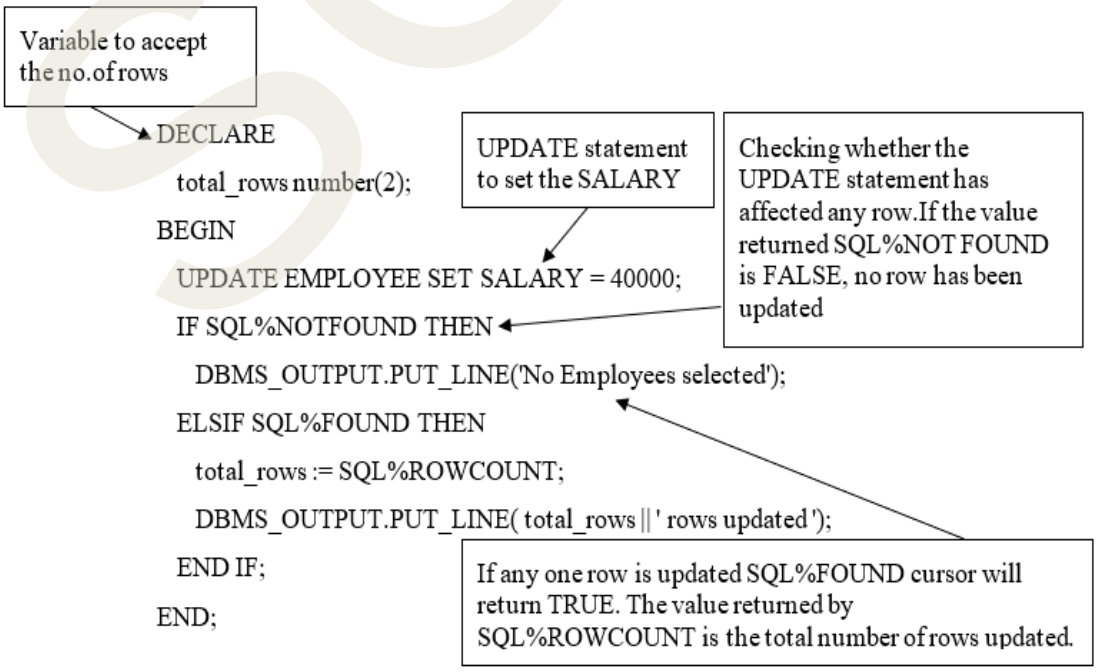
Example: Consider the table

SELECT * FROM EMPLOYEE;

Table 3.2.3 EMPLOYEE Table

E M P NO	EMP NAME	DESIGNATION	STATE	SALARY	DEPT NO
E0001	JACOB	SYSADMIN	MAHARASHTRA	16500	D0001
E0002	JOHN	MANAGER	TAMILNADU	32000	D0002
E0003	ABRAHAM	DEVELOPER	KARNATAKA	26500	D0002
E0004	JOY	QA	TAMILNADU	19560	D0001
E0005	CHRISTY	SECRETARY	MAHARASHTRA	13200	D0003

Query: Update the salary of an employee to 40000



OUTPUT

5 rows updated

PL/SQL procedure successfully completed

3.2.3 Explicit Cursor

An Explicit Cursor is a cursor that is manually defined and controlled by the programmer in PL/SQL to handle query results requiring row-by-row processing. Unlike implicit cursors, explicit cursors provide more control and are suitable for scenarios where specific operations need to be performed on each row of the result set.

To use an explicit cursor, it must be declared, opened, fetched, and closed explicitly. The cursor declaration specifies the SQL query, while the fetch operation retrieves rows one at a time into PL/SQL variables. By closing the cursor, resources are released. Explicit cursors are particularly useful when working with complex queries or when conditional logic needs to be applied to each row of the result set.

- User defined cursor.
- The programmer controls the operations in the cursor.
- Operations in explicit cursor are:
  - Declaring the cursor.
  - Opening the cursor.
  - Fetching the records from the cursor.
  - Closing the cursor.

We will be using the following CUSTOMER table for explaining the explicit cursor.

Table 3.2.4 Customer Table

ID	NAME	AGE	CITY	SALARY
1	Ramesh	32	Ahmedabad	2500.00
2	Sajith	25	Delhi	2000.00
3	George	23	Kota	2500.00
4	John	25	Mumbai	7000.00
5	Mariya	27	Bhopal	9000.00
6	Saju	22	TVM	5000.00

3.2.4 Declaring the cursor

To use an Explicit Cursor, it must first be declared in the DECLARE section of the PL/SQL block. Declaring a cursor informs the Oracle engine about the name of the cursor and the SQL query it will execute. However, at this stage, the cursor is not yet opened or executed; it simply reserves a placeholder for the query.

The syntax for declaring a cursor is as follows:

CURSOR CursorName IS [SELECT statement]; OR

CURSOR cursor_name IS sql_query;

For example:

CURSOR emp_cursor IS SELECT emp_id, salary FROM employees WHERE department_id = 10;

This declaration sets up the cursor emp_cursor to retrieve employee IDs and salaries for employees in department 10.

Example:

CURSOR CUR_CUST IS SELECT ID, NAME, CITY FROM CUSTOMER;

3.2.5 Opening the cursor

In PL/SQL, cursors are opened in the **execution** or **exception** section of the block. Opening a cursor allocates memory for fetching the rows returned by the associated SQL query. At this stage, the SQL query specified during the cursor declaration is executed, and the result set is prepared for row-by-row processing. However, if an attempt is made to open a cursor that is already open, the runtime exception CURSOR_ALREADY_OPEN will be raised. Proper management of cursor states is essential to avoid such exceptions and ensure efficient execution.

Syntax:

OPEN Cursor_Name;

Example:

OPEN CUR_CUST;

3.2.5.1 Fetching the record from the cursor

- Retrieves the rows into the memory variables, one row at a time.
- Each time a row is fetched, the cursor pointer is advanced to the next row in the Active Data Set.
- Fetching an unopened row will throw a runtime exception.
- If there are more than one is to be processed, the fetch statement has to be written in the LOOP…END LOOP.

Syntax:

FETCH CursorName INTO variable1,variable2…;

The variables have to be declared in the declaration section. After fetching the values to PL/SQL variables, we can process it.

Example:

FETCH CUR_CUST INTO v_id, v_name, v_city;

The above statement fetches the values of ID, NAME and CITY columns of the CUSTOMER table into the PL/SQL variables v_id, v_name and v_city.

3.2.6 Closing the cursor

- Disables the cursor.
- Memory area allocated is released.
- Closing an unopened cursor will throw a runtime exception.

Syntax:

CLOSE Cursor Name;

Example:

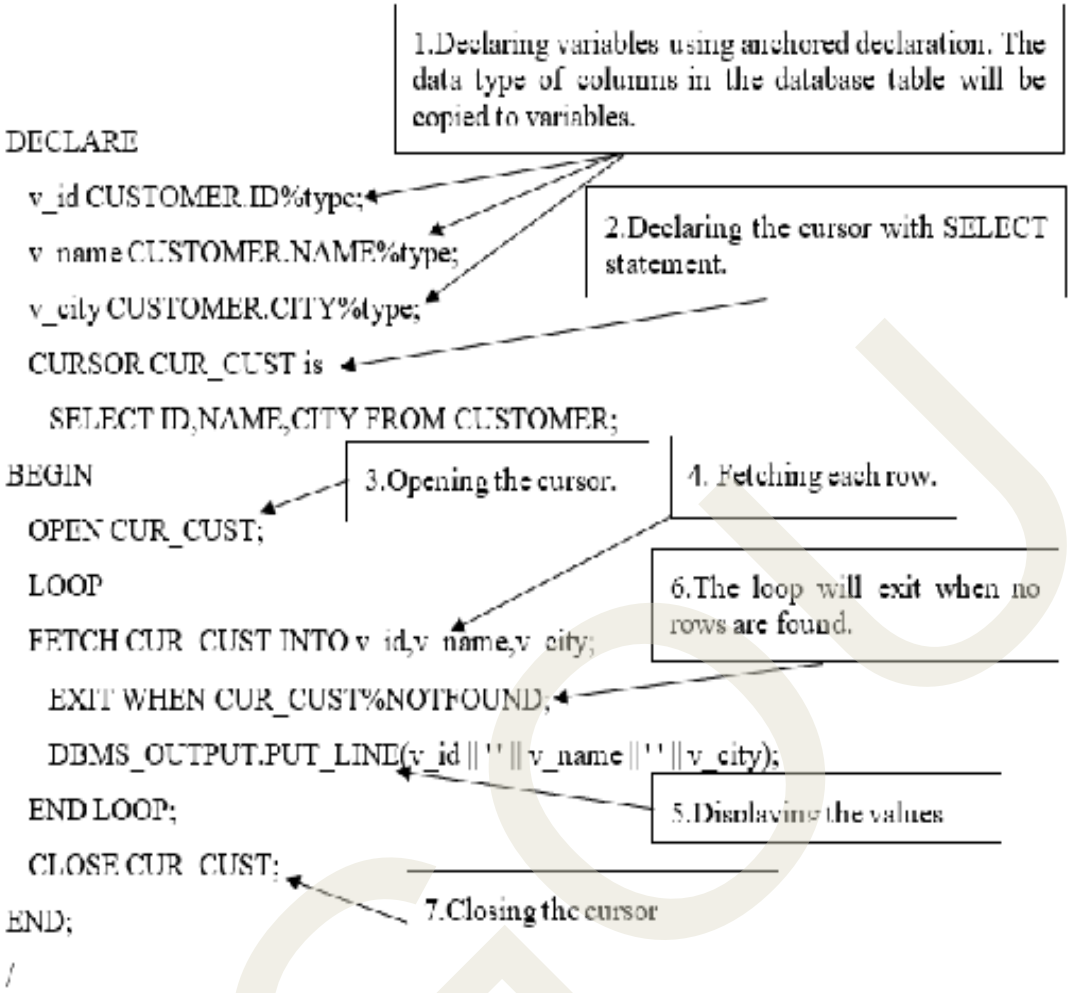
CLOSE CUR_CUST;

3.2.7 PL/SQL Cursor with LOOP

Using a cursor with a loop in PL/SQL allows for row-by-row processing of the result set returned by an explicit cursor. This approach is commonly used when operations need to be performed on each row individually. The process involves declaring a cursor, opening it, and iterating through its rows in a loop until all rows are processed.

First, the cursor is declared in the DECLARE section, specifying the SQL query to fetch data. In the BEGIN section, the cursor is explicitly opened, allocating memory for its result set. Inside the loop, each row is fetched into a record variable matching the structure of the cursor. The loop continues to process rows until no more rows are available, as indicated by the cursor attribute %NOTFOUND.

Finally, the cursor is closed to release resources. Proper cursor management ensures efficient memory usage and avoids runtime errors. Using a cursor with a loop is particularly helpful in scenarios where conditional logic or row-specific operations are required.

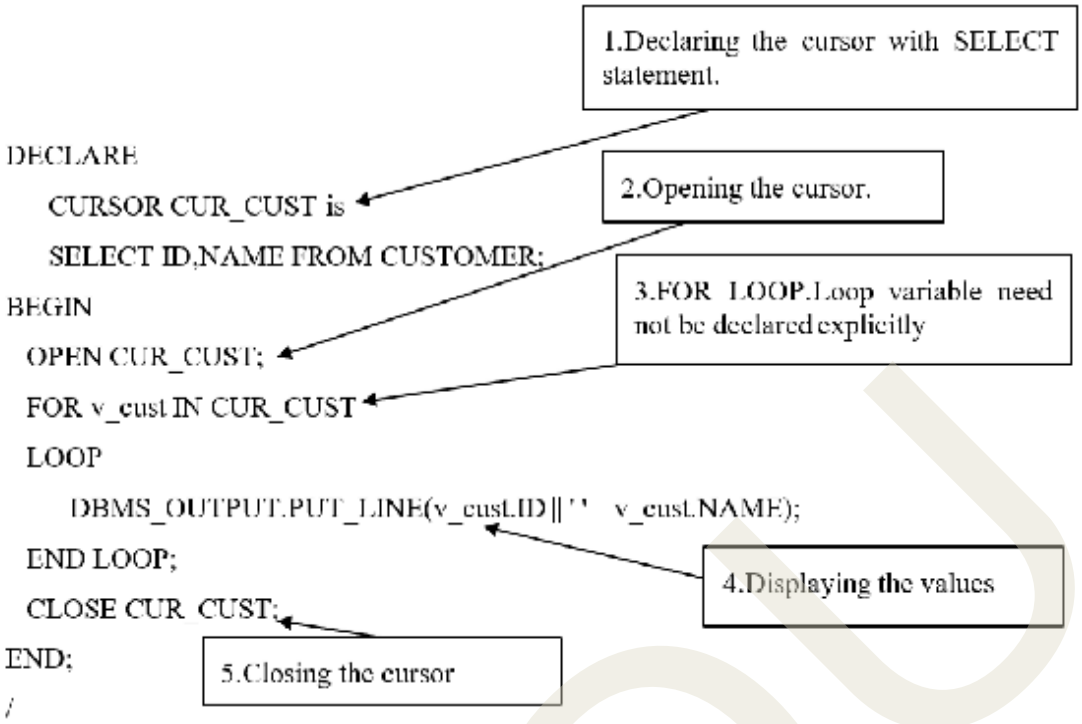


The above PL/SQL program will display the following output.

1. Ramesh Ahmedabad
2. Sajith Delhi
3. George Kota
4. John Mumbai
5. Mariya Bhopal
6. Saju TVM

3.2.8 PL/SQL Cursor FOR LOOP

In the previous section, we discussed the cursor with a simple loop. FOR Loop can also be used to fetch and process each row from a cursor. The previous example can be rewritten using FOR LOOP.



The above program will display the following output

```
1 Ramesh
2 Sajith
3 George
4 John
5 Mariya
6 Saju

PL/SQL procedure successfully completed.
```

In the above example, v_cust is a record variable.

- v_cust is the index of FOR LOOP.
- FOR LOOP statement declares the index implicity as a record variable of %ROWTYPE.
- After cursor FOR LOOP statement execution ends, the record variable will be undefined.

Thus CURSOR in PL/SQL can be used to process each record in the database one by one.

Recap

- A cursor is a memory area used by the Oracle engine for its internal processing in order to execute SQL statements.
- The records stored in the cursor are called Active Dataset.
- 2 types of cursors are: implicit cursor and explicit cursor.
- Implicit cursor is used to access the information about the status recently executed SQL statement.
- SQL%ROWCOUNT, SQL%FOUND, SQL%NOTFOUND, SQL%ISOPEN are the implicit cursors.
- Explicit cursors are declared and managed by the programmer.
- The main operations in explicit cursor are: Declaring the cursor, Opening the cursor, Fetching the cursor and Closing the cursor.
- In explicit cursor fetching of records is done using a loop.

Objective Type Questions

1. What is a cursor?
2. What are the two types of cursors?
3. What is Active Dataset?
4. Which cursor is automatically opened by the Oracle engine?
5. Which implicit cursor gives the value of the number of rows affected by the recent SQL statement?
6. What are the values returned by SQL%FOUND?
7. Which cursor is a user defined cursor?
8. What are the main operations involved in an explicit cursor?
9. Which SQL statement is used in the declaration of an explicit cursor?
10. How do the records in the database table be retrieved using an explicit cursor?
11. An OPEN CURSOR statement must be executed before fetching the rows in to the cursor. TRUE or FALSE.

Answers to Objective Type Questions

1. A cursor is a memory area used by the Oracle engine for its internal processing in order to execute SQL statements.
2. Implicit cursor and Explicit cursor.
3. The records stored in the cursor are called Active Dataset.
4. Implicit Cursor.
5. SQL%ROWCOUNT
6. TRUE: If the recent SQL statement affects any row in the table. FALSE, if recent SQL statement does not affect any row in the table.
7. Explicit Cursor.
8. Declaring the cursor, Opening the cursor, Fetching the cursor and Closing the cursor.
9. SELECT
10. Using the FETCH statement.
11. TRUE.

Assignments

1. Write a PL/SQL block to declare a cursor that retrieves employee names and their salaries.
2. Create a PL/SQL block to fetch and display the first employee's name from a cursor.
3. Use a cursor FOR LOOP to display the names of employees in department 20.
4. Write a PL/SQL block to update the salary of employees whose salary is less than 5000 by 10%.
5. Write a PL/SQL block to fetch and display employee details until no more rows are available using a cursor.

Suggested Reading

1. Bayross, I. (n.d.). SQL, *PL/SQL: The programming language of Oracle* (3rd ed.). BPB Publications.
2. Feuerstein, S. (n.d.). *Oracle PL/SQL programming.* O'Reilly Media.
3. McDonald, C., Katz, C., Beck, C., Kallman, J. R., & Knox, D. C. (n.d.). *Mastering Oracle PL/SQL: Practical solutions*. Apress Media.
4. Oracle Tutorial. (n.d.). PL/SQL tutorial. Retrieved March 5, 2025, from https://www.oracletutorial.com/plsql-tutorial/

Reference

1. tutorialspoint.com/plsql_cursor.htm
2. geeksforgeeks.org/cursors-in-pl-sql/
3. mygreatlearning.com/plsql/tutorials/pl-sql-cursors

Procedures and Functions

Learning Outcomes

Studying this unit will enable the student to;

- Define a procedure
- Familiarize how to create procedures inside a PL/SQL block
- Explore how to compile and execute a PL/SQL procedure

Prerequisites

You've already gained a solid understanding of SQL, mastering how to query, manipulate, and manage data efficiently. SQL has equipped you with the tools to interact with databases, allowing you to retrieve specific information, filter data, and perform updates or deletions. These foundational skills have made you adept at handling data in a structured and straightforward manner.

Now, it's time to take your skills to the next level with PL/SQL, which builds on standard SQL by introducing programming elements like loops, conditions, and variables. This allows you to create more complex and reusable code structures. PL/SQL helps you automate repetitive tasks and streamline complex operations, making your database interactions more efficient and dynamic.

As you explore the world of PL/SQL procedures and functions, you'll learn to encapsulate tasks into reusable blocks of code. Procedures allow you to store and execute predefined routines, while functions perform operations that return specific values. These tools will enable you to streamline processes, automate complex tasks, and handle more intricate data operations, significantly boosting your efficiency and effectiveness in managing databases.

Keywords

Procedure, Procedure header, Procedure body, Parameter mode

Discussion

In the previous units we discussed the basic concepts of PL/SQL and Cursors. Sometimes we need to execute a set statement repeatedly in various parts of a program. For example, we want to calculate the area of a circle in various parts of the program. We may write the code for calculating the area of a circle again and again as required. This may increase the number of lines in the program. Inorder to reduce the number of lines we can write the statements in a block and name it. Then the name of the block is called as and when required. Such a named block is called **Procedure**.

What if you have an SQL query to be executed over and over again? In such cases also the query can be saved in a stored procedure and just call the procedure to execute the query. The main advantage of a stored procedure is code reusability.

3.3.1 PL/SQL Procedure

A PL/SQL procedure is a subprogram designed to perform a specific task. It is a named block stored as a schema object in the Oracle database. Procedures are primarily used to perform actions. A PL/SQL procedure consists of three main parts:

3.3.1.1 Declarative Part

Optional: This part does not begin with the DECLARE keyword.

Content: It includes declarations for types, cursors, constants, variables, exceptions, and nested subprograms.

Scope: These declarations are local to the procedure and cease to exist after the procedure completes execution.

3.3.1.2 Executable Part

Mandatory: This part contains the statements that perform the designated action.

3.3.1.3 Exception-handling Part

Optional: This section includes code to handle runtime errors.

3.3.2 Syntax for Creating a PL/SQL Procedure

```
CREATE [OR REPLACE ] PROCEDURE procedure_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
IS
[declaration statements]
BEGIN
[execution statements]
EXCEPTION
```

[exception handler]

END [procedure_name];

3.3.2.1 Procedure Header

The CREATE statement in Oracle is used to define a new procedure, specifying its name and associated logic. The optional [OR REPLACE] clause allows for modifying an existing procedure by replacing it with a new definition. The procedure can include a parameter list, where parameters are designated as IN, OUT, or INOUT to specify the direction of data flow between the procedure and the calling program.

3.3.2.2 Procedure Body

The body of a procedure contains the core executable part, where the main operations and logic are performed. To define a standalone procedure, you can use either the keyword IS or AS. The procedure's structure typically includes three key sections: the declarative part for declaring variables and constants, the execution part where the procedure's main actions take place, and the exception handling part, which is used to handle any errors that might occur during execution.

3.3.3 Executing a PL/SQL Procedure

3.3.3.1 Syntax

EXECUTE procedure_name(arguments);

or

EXEC procedure_name(arguments);

Example:

EXECUTE proc1();

EXECUTE proc_add(10,5);

3.3.3.2 Examples

Example 1: The following example displays the string “HELLO WORLD”.

```
CREATE OR REPLACE PROCEDURE proc1
AS
BEGIN
DBMS_OUTPUT.PUT_LINE('Hello World!');
END;
```

When the above code is executed using the SQL prompt, it will produce the following result:

Procedure created

To execute the above procedure can be called using EXECUTE keyword.

EXECUTE proc1

The above call will display the following output

Hello World

PL/SQL procedure successfully completed.

A procedure can also be called from a PL/SQL block.

The above call will display the following output

Hello World

PL/SQL procedure successfully completed.

Example 2: Create a PL/SQL procedure that takes customer id as input and delete that customer.

Table 3.3.1 CUSTOMER table

ID	NAME	AGE	CITY	SALARY
1	Ramesh	32	Ahmedabad	2500.00
2	Sajith	25	Delhi	2000.00
3	George	23	Kota	2500.00
4	John	25	Mumbai	7000.00
5	Mariya	27	Bhopal	9000.00
6	Saju	22	TVM	5000.00

Query:

```
CREATE OR REPLACE PROCEDURE CUST_DELETE(v_id CUSTOMER.ID%TYPE)
IS
BEGIN
  DELETE FROM CUSTOMER WHERE ID = v_id;
END;
```

The above procedure can be executed as follows

EXECUTE CUST_DELETE(1)

The output will be

PL/SQL procedure successfully completed.

3.3.4 Parameter Modes in Procedures

In Oracle procedures, the parameter mode defines how a parameter can be used, whether it can be read, written to, or both. There are three distinct parameter modes: IN, OUT, and INOUT. Each mode serves a specific purpose in the procedure's functionality.

3.3.4.1 IN Mode

The IN mode is used for read-only parameters. Parameters in this mode are treated as constants within the procedure, which means that their values cannot be altered during execution. This mode is particularly useful for passing input data to the procedure. By default, all parameters are assumed to be in IN mode if no specific mode is declared. IN mode parameters are passed by reference, allowing the procedure to reference the original data without making a copy.

Example

```
CREATE OR REPLACE PROCEDURE DISPLAY_EMPLOYEE_NAME(EMPLOYEE_ID IN EMPLOYEES.EMPLOYEE_ID%TYPE)
IS
   EMPLOYEE_NAME EMPLOYEES.FIRST_NAME%TYPE;
BEGIN
    SELECT FIRST_NAME
   INTO EMPLOYEE_NAME
   FROM EMPLOYEES
   WHERE EMPLOYEE_ID = DISPLAY_EMPLOYEE_NAME.EMPLOYEE_ID;
DBMS_OUTPUT.PUT_LINE('EMPLOYEE NAME: ' || EMPLOYEE_NAME);
END;
```

In this example, the display_employee_name procedure takes an employee_id as an IN parameter. The procedure retrieves the corresponding first_name from the employees table and prints it. Since employee_id is an IN parameter, its value is passed to the procedure but cannot be modified within the procedure.

3.3.4.2 OUT Mode

The OUT mode is designed for parameters that need to return values back to the calling program. Unlike IN mode, OUT mode parameters act like variables inside the procedure, allowing their values to be assigned and modified. It is important to note that the actual parameter provided in the calling program must be a variable, as it will store the returned value. OUT parameters are passed by value, meaning the procedure works with a copy of the value, which is then returned to the caller.

Example

```
CREATE OR REPLACE PROCEDURE GET_EMPLOYEE_NAME(
EMPLOYEE_ID IN EMPLOYEES.EMPLOYEE_ID%TYPE,
EMP_NAME OUT EMPLOYEES.FIRST_NAME%TYPE
)
IS
BEGIN
EMPLOYEE_ID
SELECT FIRST_NAME
INTO EMP_NAME
FROM EMPLOYEES
WHERE EMPLOYEE_ID = EMPLOYEE_ID;
END;
```

The GET_EMPLOYEE_NAME procedure demonstrates the use of IN and OUT parameters in PL/SQL. The EMPLOYEE_ID is an IN parameter that specifies the employee's ID, while EMP_NAME is an OUT parameter used to return the employee's first name. Inside the procedure, a SELECT INTO statement retrieves the FIRST_NAME of the employee with the given EMPLOYEE_ID and assigns it to EMP_NAME. If no matching employee is found, the exception handler assigns 'Not Found' to EMP_NAME. In the calling block, a variable V_EMP_NAME is declared to capture the output. The procedure is called with an employee ID of 100, and upon execution, V_EMP_NAME holds the employee's name, which is then printed using DBMS_OUTPUT.PUT_LINE. This example effectively shows how an OUT parameter is used to return data from a procedure to the caller.

3.3.4.3 INOUT Mode

The INOUT mode combines the functionalities of both IN and OUT modes. It allows a parameter to both receive an initial value from the calling program and return an updated value after the procedure executes. Parameters in INOUT mode can be read and modified within the procedure, making them versatile for scenarios where input data needs to be processed and returned. Like OUT mode, the actual parameter must be a variable to capture the modified value after the procedure completes.

Example

Procedure Definition:

```
CREATE OR REPLACE PROCEDURE UPDATE_SALARY(
EMPLOYEE_ID IN EMPLOYEES.EMPLOYEE_ID%TYPE,
```

```
SALARY INOUT EMPLOYEES.SALARY%TYPE
)
IS
BEGIN
SELECT SALARY * 1.10
INTO SALARY
FROM EMPLOYEES
WHERE EMPLOYEE_ID = EMPLOYEE_ID;
END;
```

Calling the Procedure:

```
DECLARE
V_EMP_ID EMPLOYEES.EMPLOYEE_ID%TYPE := 100;
V_SALARY EMPLOYEES.SALARY%TYPE;
BEGIN
SELECT SALARY INTO V_SALARY FROM EMPLOYEES WHERE EMPLOYEE_ID = V_EMP_ID;
UPDATE_SALARY(V_EMP_ID, V_SALARY);
DBMS_OUTPUT.PUT_LINE('Updated Salary: ' || V_SALARY);
END;
```

The INOUT mode in PL/SQL allows a parameter to both receive an initial value from the calling program (like an IN parameter) and return an updated value after the procedure executes (like an OUT parameter). This mode is useful when input data needs to be processed and modified, such as updating values. The parameter can be read and modified within the procedure, and the actual parameter passed must be a variable to capture the modified value after execution. In the provided example, the procedure UPDATE_SALARY takes an EMPLOYEE_ID as an input to identify the employee and an INOUT parameter SALARY that starts with the employee's current salary and returns the updated salary after applying a 10% increase, which is then displayed in the calling block.

3.3.5 Example Questions

Example 1:

Write a PL/SQL program to find the smallest among two numbers using procedure.

```
DECLARE
x number;
y number;
z number;
PROCEDURE proc_Min(a IN number, b IN number, c OUT number) IS
BEGIN
IF a < b THEN
c:= a;
ELSE
  c:= b;
END IF;
END;
BEGIN
x:=40 ;
y:= 35;
proc_Min(x, y, z);
dbms_output.put_line(' Minimum of (40, 35) : ' || z);
```

In the above example a procedure is created in the PL/SQL DECLARE section. It accepts three parameters. Values of two parameters are supplied in IN mode. The third parameter is in OUT mode and returns the minimum value to the program. The procedure is called and the output is displayed.

```
Minimum of (40, 35) : 35
PL/SQL procedure successfully completed.
```

Example: Write a PL/SQL program to find the square of a number using procedure.

```
DECLARE
a number;
PROCEDURE squareNum(a IN OUT number) IS
BEGIN
  a := a * a;
END;
```

```
BEGIN
x:= 50;
squareNum(x);
dbms_output.put_line(' Square of (50): ' || x);
END;
```

In the above example the parameter is IN OUT parameter. The value is passed to procedure and square is returned to the calling program. Executing the above PL/SQL block will give the following result.

```
Square of (50): 2500
PL/SQL procedure successfully completed.
```

3.3.6 Deleting a Standalone Procedure

A standalone procedure can be deleted using the DROP keyword.

Syntax:

```
DROP   procedure_name;
```

Example:

```
DROP    proc1;
```

In this way you can create named procedures and reuse the code as and when needed.

Recap

- A procedure is a named part of a program.
- A procedure can be executed using the EXECUTE keyword or it can be called directly in a PL/SQL program.
- Purpose of Procedures: Used to reduce code repetition by grouping commonly executed statements into a named block.
- PL/SQL Procedure: A subprogram stored in the database, performing specific tasks, improving code reusability.
- Procedure Components:
- Declarative Part: Optional; contains declarations (variables, cursors, constants).
- Executable Part: Mandatory; includes the logic to perform actions.

- Exception-handling Part: Optional; handles errors during execution.
- Procedure Syntax:
- CREATE PROCEDURE procedure_name [(parameters)] IS [declarations] BEGIN [actions] EXCEPTION [error handling] END;
- IN, OUT, INOUT for defining parameter directions.
- Procedure Header:
- CREATE defines the procedure.
- OR REPLACE allows modifying an existing procedure.
- Executing Procedures:
- Use EXECUTE procedure_name(arguments); or EXEC procedure_name(arguments); to call.
- Three parameter passing modes are: IN, OUT, INOUT.
- A standalone procedure can be deleted using DROP procedure_name.
- Purpose of Parameter Modes: Define how a parameter is used within a procedure (read, write, or both).

Types of Parameter Modes:

- **IN Mode:**
  - Read-only parameters.
  - Passed by reference, values cannot be modified within the procedure.
  - Default mode for parameters.
- **OUT Mode:**
  - Used to return values to the calling program.
  - Passed by value (working with a copy of the value).
  - The actual parameter in the calling program must be a variable.
- **INOUT Mode:**
  - Combines functionalities of both IN and OUT modes.
  - Receives an initial value and can return an updated value.

- Used for parameters that need to be processed and modified.

- **IN Mode:** Useful for passing input data that should not be altered in the procedure.
- **OUT Mode:** Best for situations where data needs to be returned from a procedure.
- **INOUT Mode:** Suitable for scenarios where a parameter is both input and output, like updating values.

Objective Type Questions

1. What is the correct syntax to create a procedure in PL/SQL?
2. Which keyword is used to terminate a PL/SQL block?
3. How do you specify an OUT parameter in a PL/SQL procedure?
4. Can a PL/SQL procedure return a value?
5. What is the difference between a procedure and a function in PL/SQL?
6. What is the purpose of the EXCEPTION section in a PL/SQL block?
7. How can you handle exceptions in a PL/SQL procedure?
8. What command is used to delete a procedure in PL/SQL?
9. What is the role of the RETURN keyword in a PL/SQL function?
10. What is a procedure?
11. How can you create a standalone procedure?
12. How will you execute a standalone procedure?
13. What are the three parameter passing modes in PL/SQL procedure?
14. Which is the default parameter mode?
15. If you want to read and modify the value of a variable in a procedure, which parameter mode is suitable?

Answers to Objective Type Questions

1. CREATE OR REPLACE PROCEDURE procedure_name IS BEGIN ... END;
2. The END keyword.
3. By specifying the parameter as OUT.
4. No, a procedure cannot return a value; only a function can.
5. A procedure performs an action, while a function returns a value.
6. To handle runtime errors or exceptions.
7. By using BEGIN ... EXCEPTION ... END block with specific or generic exception handlers.
8. DROP PROCEDURE procedure_name;
9. To specify the value that a function will return.
10. A named part of a program.
11. Using CREATE OR REPLACE.
12. Using the EXECUTE keyword.
13. IN, OUT, INOUT.
14. IN
15. INOUT.

Assignments

1. Write a PL/SQL procedure named calculate_bonus that takes an employee's ID as input and calculates a 10% bonus based on their current salary. The procedure should update the employee's bonus in the employees table.
2. Develop a PL/SQL function named get_department_name that accepts a department ID as input and returns the name of the department from the departments table. Include appropriate error handling for invalid department IDs.
3. Create a PL/SQL procedure named update_salary that increases an employee's salary by a percentage specified by a function named calculate_increment. The function should take the current salary and a percentage as parameters and return the new salary.

Suggested Reading

1. Bayross, I. (n.d.). SQL, *PL/SQL: The programming language of Oracle* (3rd ed.). BPB Publications.
2. Feuerstein, S. (n.d.). *Oracle PL/SQL programming*. O'Reilly Media.
3. McDonald, C., Katz, C., Beck, C., Kallman, J. R., & Knox, D. C. (n.d.). *Mastering Oracle PL/SQL: Practical solutions*. Apress.
4. Oracle Tutorial. (n.d.). PL/SQL tutorial. Retrieved March 5, 2025, from https://www.oracletutorial.com/plsql-tutorial/

Reference

1. Feuerstein, S. (2005). *Oracle PL/SQL programming*. O'Reilly Media.
2. Rosenzweig, B., & Silvestrova, E. (2016). *Oracle PL/SQL by example*. Pearson.
3. Pribyl, B., & Feuerstein, S. (2003). *Learning Oracle PL/SQL*. O'Reilly Media.
4. Feuerstein, S. (2014). *Oracle PL/SQL for dummies*. Wiley.

Triggers

Learning Outcomes

Studying this unit will enable the student to;

- Define triggers
- Create triggers
- Identify the key components of a trigger.
- Explain the usage of triggers

Prerequisites

In your journey through PL/SQL, you've mastered procedures and functions—powerful tools that allow you to automate tasks and encapsulate complex logic within reusable blocks of code. Procedures have taught you how to execute predefined sequences of statements, while functions have enabled you to perform operations and return specific values based on inputs. These skills have made your database interactions more efficient and structured, allowing you to manage and manipulate data with greater precision.

Now, it's time to build on that foundation by exploring **triggers**. While procedures and functions are explicitly called by users or applications, triggers operate automatically in response to specific events within the database. Imagine you need certain actions to be performed automatically whenever data in a table is inserted, updated, or deleted. Instead of relying on manual execution, triggers allow the database to react instantly to these changes, ensuring consistency, enforcing rules, and automating repetitive tasks without human intervention.

As you delve into triggers, you'll discover how to extend the automation capabilities of PL/SQL. Triggers act as vigilant guardians of your data, ensuring that specific actions occur at the right time, maintaining the integrity and reliability of your database. With your solid grasp of procedures and functions, you are well-prepared to take on the dynamic world of triggers and unlock even greater efficiency in your database management.

Keywords

Trigger, CREATE TRIGGER, row-level trigger, statement-level trigger

Discussion

3.4.1 Introduction to Triggers

In the previous unit, we discussed PL/SQL procedures, which allow for code reusability. Similar to procedures, triggers enable multiple applications to execute a set of SQL statements, reducing redundant code when multiple applications need to perform the same database operation.

For instance, consider an online shopping scenario where a business rule restricts a customer from ordering more than one bottle of perfume at a time. If an attempt is made to order more than one, a trigger can enforce this rule by displaying a warning message.

3.4.2 Definition and Purpose of Triggers

- Triggers are stored programs that are automatically executed or "fired" when a specific event occurs in the database.
- Triggers are typically written to respond to SQL commands such as INSERT, DELETE, or UPDATE.

3.4.3 Usages of Triggers

Triggers are used for various purposes, including:

- Enforcing **UNIQUE**, **CHECK**, and **NOT NULL** constraints.
- Preventing **invalid transactions**.
- Generating values for **derived attributes**. For example, while inserting the date of birth of an employee, the age column can be automatically calculated and filled.
- **Auditing** sensitive data.

Syntax for creating triggers:

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER}
ON table_name
[FOR EACH ROW]
```

[FOLLOWS | PRECEDES another_trigger]

[ENABLE / DISABLE]

[WHEN condition]

DECLARE

Declaration-statements

BEGIN

Executable-statements

EXCEPTION

Exception-handling-statements

3.4.3.1 Trigger Header

The following part is the trigger header.

CREATE [OR REPLACE] TRIGGER trigger_name

{BEFORE | AFTER}

ON table_name

[FOR EACH ROW]

[FOLLOWS | PRECEDES another_trigger]

[ENABLE / DISABLE]

[WHEN condition]

3.4.3.2 Trigger Body

The following part illustrates trigger body

DECLARE

Declaration-statements

BEGIN

Executable-statements

EXCEPTION

Exception-handling-statements

END;

- **CREATE OR REPLACE**
    - Keyword for creating a trigger.

- OR REPLACE is used to modify an existing trigger. Optional. But it is recommended.

Example:

CREATE OR REPLACE trg1_example

The above statement creates a trigger named trg1_example. If such a trigger exists, it will replace the existing trigger.

trigger_name

- specifies the name of the trigger.

BEFORE | AFTER

- Specifies when the trigger should be fired.
- BEFORE- the trigger is fired before INSERT, DELETE, UPDATE statement.
- AFTER-the trigger is fired after INSERT, DELETE, AND UPDATE statement.

ON table_name

- Name of the table on which the trigger is associated.

FOR EACH ROW

- Specifies a row-level trigger.
- For each row inserted or deleted or updated, the trigger will be fired.
- **Statement-level trigger**-Fired only once regardless of number of rows affected by the triggering event.
- If the clause FOR EACH ROW is not specified, the trigger will be created as a statement-level trigger.

ENABLE | DISABLE

- Specifies whether the trigger is in ENABLE or DISABLE state.
- If a trigger is in DISABLE state, it will not be fired.
- Default state is ENABLE.

FOLLOWS | PRECEDES another_trigger

- Multiple triggers can be fired for each trigger.
- Specify firing sequence using PRECEDES or FOLLOWS keyword

3.4.3.3 Dropping Triggers

A trigger can be dropped using DROP TRIGGER command

Syntax

DROP TRIGGER trigger_name;

Example

DROP TRIGGER trg1_example;

Example:The following program illustrates a trigger

Consider the CUSTOMER table as shown below.

Table 3.4.1 CUSTOMER Table

ID	NAME	AGE	CITY	SALARY
1	Ramesh	32	Ahmedabad	2500.00
2	Sajith	25	Delhi	2000.00
3	George	23	Kota	2500.00
4	John	25	Mumbai	7000.00
5	Mariya	27	Bhopal	9000.00
6	Saju	22	TVM	5000.00

Our aim is to create a trigger that is fired during the insertion of a new record.

```
CREATE OR REPLACE TRIGGER trg_display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON CUSTOMER
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
diff_sal number;
BEGIN
diff_sal := :NEW.salary  - :OLD.salary;
dbms_output.put_line('Old salary: ' || :OLD.salary);
dbms_output.put_line('New salary: ' || :NEW.salary);
dbms_output.put_line('Salary difference: ' || diff_sal);
END;
```

The above code, when executed gives the following output.

Trigger created.

The trigger created will be fired each time a row is inserted into the table.

Now, let us insert a row into the table using an INSERT statement.

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)

VALUES (7, 'Joseph', 22, 'HP', 7500.00);

When a record is inserted, the trigger trg_display_salary_changes is fired and the following message is displayed.

Old salary:

New salary: 7500

Salary difference:

Recap

- Triggers are stored programs that execute automatically when a specific event occurs in the database.
- Triggers respond to SQL commands like INSERT, DELETE, or UPDATE.
- Purpose: Enforce constraints, prevent invalid transactions, generate values for derived attributes, audit sensitive data.
- Syntax to create triggers:
  - CREATE [OR REPLACE] TRIGGER trigger_name
  - {BEFORE | AFTER} on a specified table.
  - Optional: [FOR EACH ROW], [ENABLE/DISABLE], [WHEN condition], [FOLLOWS | PRECEDES another_trigger]
- Trigger Header:
  - Includes trigger name, timing (BEFORE/AFTER), table name, optional conditions, and enable/disable options.
- Trigger Body:
  - Includes declaration, executable, and exception-handling statements.
- CREATE OR REPLACE allows for trigger modification.
- BEFORE or AFTER defines when the trigger is fired.
- FOR EACH ROW specifies row-level triggers (fires for each row affected).

- Statement-level triggers fire once regardless of affected rows.
- ENABLE/DISABLE defines trigger state.
- FOLLOWS/PRECEDES defines firing sequence for multiple triggers.
- Triggers can be dropped using DROP TRIGGER.

Objective Type Questions

1. What is the purpose of a trigger in PL/SQL?
2. Which SQL operations can trigger a trigger?
3. What is the default state of a trigger when created?
4. Which clause specifies when a trigger is fired?
5. What does the FOR EACH ROW clause indicate in a trigger?
6. What keyword is used to modify an existing trigger?
7. How can a trigger be disabled?
8. Which part of the trigger contains the logic to execute?
9. What is the syntax to drop a trigger?
10. What does the WHEN clause do in a trigger?
11. Which subprogram is automatically executed when an event like insertion or deletion or updation occurs?
12. What keyword should be used to modify an existing trigger?
13. What is the difference between row-level triggers and statement-level triggers?
14. Which clause specifies when the trigger should be fired?

Answers to Objective Type Questions

1. Automation
2. INSERT, UPDATE, DELETE
3. ENABLE
4. BEFORE, AFTER
5. Row-level trigger
6. OR REPLACE
7. DISABLE
8. Trigger body
9. DROP TRIGGER
10. Conditional execution
11. Trigger
12. REPLACE
13. Row-level triggers are fired for each row, while statement-level triggers are fired only once.
14. BEFORE | AFTER

Assignments

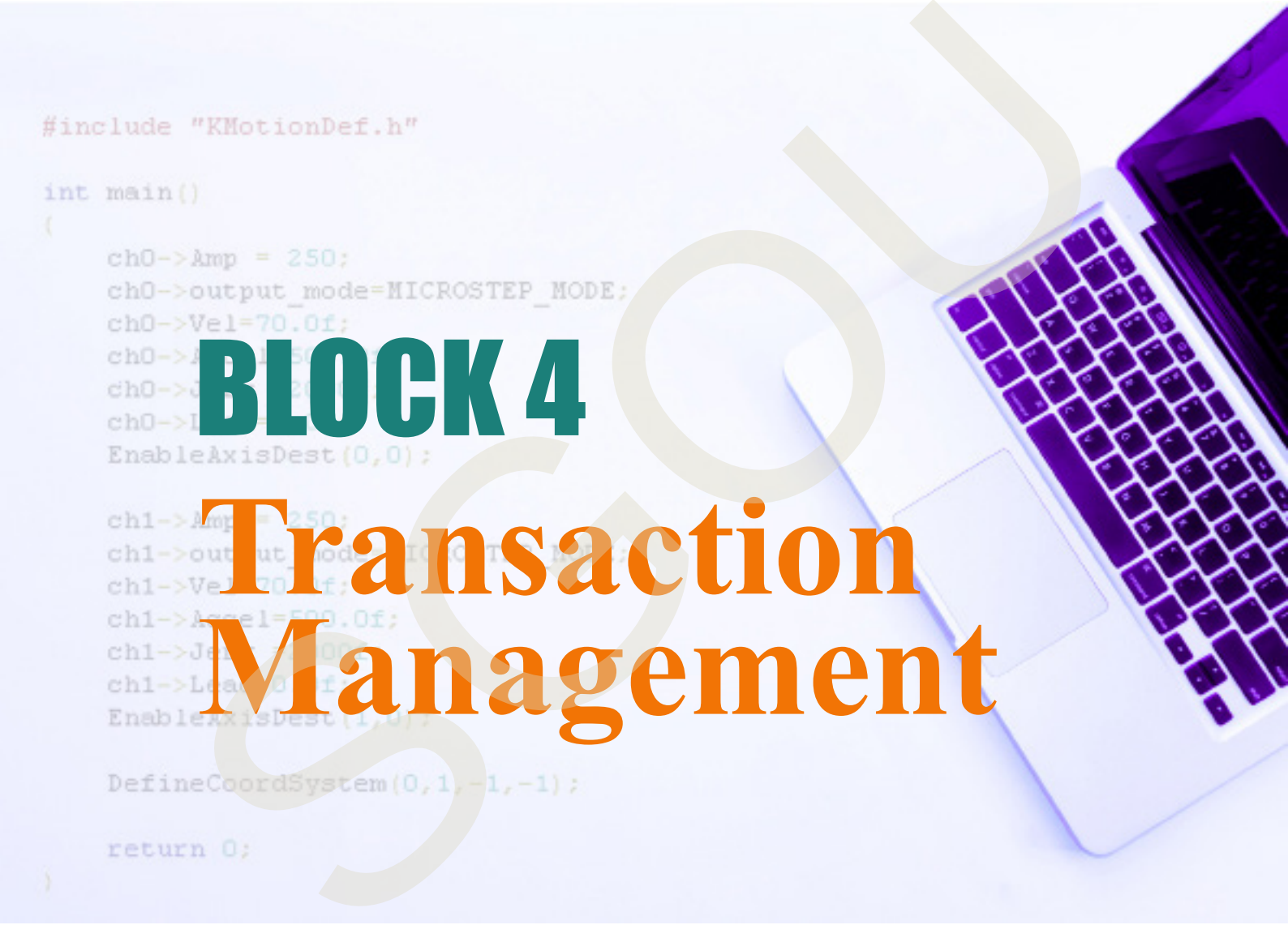
1. Write a PL/SQL trigger named log_employee_changes that activates after any UPDATE operation on the employees table. The trigger should insert a record into an employee_audit table, logging the employee ID, old salary, new salary, and the update timestamp.
2. Create a trigger named prevent_negative_salary on the employees table that fires before any INSERT or UPDATE. The trigger should ensure that the salary value cannot be negative. If a negative salary is detected, raise a custom exception with an appropriate error message.
3. Develop a trigger named delete_department_employees that activates before a DELETE operation on the departments table. The trigger should automatically delete all employees associated with the department being removed, ensuring data consistency.
4. Write a trigger named update_inventory_stock that fires after an INSERT or UPDATE on the order_details table. The trigger should automatically update the stock_quantity in the inventory table based on the quantity of items ordered.

Suggested Reading

1. Bayross, I. (n.d.). SQL, *PL/SQL: The programming language of Oracle* (3rd ed.). BPB Publications.
2. Feuerstein, S. (n.d.). *Oracle PL/SQL programming*. O'Reilly Media.
3. McDonald, C., Katz, C., Beck, C., Kallman, J. R., & Knox, D. C. (n.d.). *Mastering Oracle PL/SQL: Practical solution*s. Apress.
4. Oracle Tutorial. (n.d.). *PL/SQL tutorial*. Retrieved March 5, 2025, from https://www.oracletutorial.com/plsql-tutorial/

Reference

1. geeksforgeeks.org/sql-trigger-student-Database/
2. archieve.nptel.ac.in/course/106/104/106104135/



BLOCK 4

Transaction Management

Introduction to Transaction Management

Learning Outcomes

The learner will be able to:

- familiarise the concept of transaction in the database.
- describe the properties of transactions.
- understand the different states of transaction.
- define system log

Prerequisites

In your journey through PL/SQL, you have explored how to create, manage and manipulate databases using procedural extensions to SQL. You have written stored procedures, triggers and functions ensuring that data is processed efficiently and securely. These skills have given you the power to structure complex queries and automate tasks. But working with data is not just about writing queries and procedures. It is also about ensuring that every transaction is handled correctly, even in unexpected situations. Databases are the backbones of businesses, finance, healthcare and countless other fields. A well managed transaction system keeps this backbone strong. As you move into transaction management, you will learn how to maintain consistency, enforce automacy and recover from failures.

Keywords

Transaction, Commit, Abort, Rollback, System Log, Atomicity, Concurrency, Isolation, Durability

Discussion

Raju wanted to withdraw some amount of money from his bank account. He went to the ATM counter, entered the pin and the amount to be withdrawn. The machine dispensed the amount and Raju collected the cash. The amount withdrawn was debited from his account and the new balance is displayed on the ATM screen. This is an example of everyday activity that we are familiar with. Similarly, Raju can deposit money in his bank account or transfer the amount from his account to somebody else's account.

Today, many people prefer shopping online for their favorite items. Similarly, when booking tickets for flights or trains, we use an online reservation system. All these activities rely on accessing and managing a database. Each of these actions involves processing data in the database, and this collection of actions is known as a **transaction**.

4.1.1 Transaction

- Any logical work or set of works that are done on the data of a database is known as a transaction.
- Logical work can be inserting a new value in the current database, deleting existing values, or updating the current values in the database.
- The primary goal of a transaction is to guarantee that the database stays accurate, consistent, and dependable, even if an error occurs during the process.
- For example, when you buy a book online, the system checks if it is in stock, processes your payment, and updates the inventory. If anything goes wrong, like the payment failing or the book not being available, the transaction is canceled, and nothing changes
- The database operation can be implemented either using a query language called Structured Query Language or embedded in an application program.
- A transaction can be
  - **read-only**
    - Only retrieves data from the database. There is no update operation.
    - For example, checking the balance amount in the bank account.
  - **read-write**
    - Performs any modification in the database like insertion of data, deletion of data or updation of database.
    - For example, deposit money in the bank account or withdraw

4.1.2 Transaction Management

Raju went to the ATM to withdraw money from his bank account. He entered the pin

and amount to be retrieved. A system crash occurred, and Raju did not get the money. But the amount was debited from his account. The transaction has failed because Raju did not receive the money. Here the bank will revert the transaction and the amount will be credited back to his account. The transaction will also fail if Raju does not have sufficient funds in his account. A transaction may fail or may not be completed due to several reasons like system crash, unavailability of resources, network error. When a failure occurs, the transaction will be cancelled, and the database will be restored to the same previous state before the transaction is executed.

There are three essential concepts related to transaction management.

1. Commit

- If all the operations in a transaction are completed successfully, the changes are recorded permanently in the database.

2. Rollback

- If the transaction is not successful, the changes are not recorded in the database and the transaction will be aborted or canceled. Undo all the current operations and the database is restored to the previous state.

3. System Log

- A record maintained by DBMS to keep track of all transactions that affect data in the database.
- Each entry is a record or row, which consists of:
  - Transaction id, a unique identifier assigned to every transaction by DBMS.
  - Operations; read or write.
  - Commit or Abort.

4.1.3 Transaction States

A **transaction state** refers to the status or condition of a transaction at any given

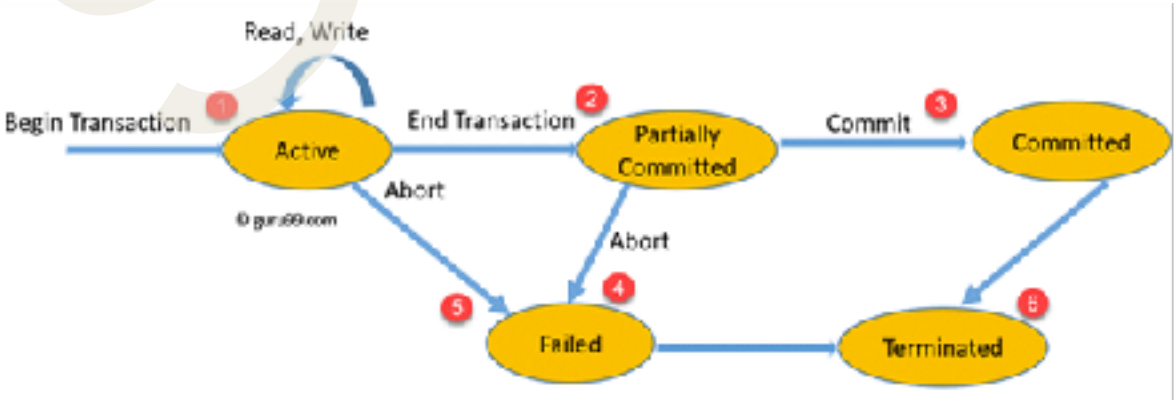


Fig 4.1.1 States of transaction in DBMS

point during its life cycle. Fig 4.1.1 shows how a transaction state works

- **New**
  - Marks the beginning of a transaction
- **Active**
  - The transaction is executing. Operations like insertion, deletion, updation are done in this stage. However, the changes have not been saved to the database permanently
- **Partially Committed**
  - This state happens after the last operation of the transaction is completed, but before the transaction is officially finalized in the database.
  - The transaction is almost complete, but the system is verifying that everything is accurate before making the changes permanent.
- **Committed**
  - The transaction executed successfully, and the modifications are made permanent on the database
- **Failed**
  - The transaction failed to complete successfully because of an error or some issue (such as data inconsistency or a system crash). In this state, the transaction is not finalized, and its changes are not applied to the database.
- **Aborted**
  - The state when a transaction is rolled back due to failure, restoring the database to its previous state.
- **Terminated**
  - The state after a transaction has reached a committed or aborted state

4.1.4 Transaction Properties

- To maintain database consistency both before and after a transaction, specific properties, known as **ACID properties, shown in Fig 4.1.2** must be followed.
- These properties are essential when multiple transactions occur at the same time
- **Atomicity**
  - A transaction is atomic means that it is treated as a single, indivisible unit.
  - Atomicity is also known as the 'All or nothing rule'

- Either the entire transaction is completed successfully, or not executed at all in case of failure
- If something goes wrong, any changes made by the transaction are rolled back.
- In the ATM example, If the system crashes while Raju withdraws money, Atomicity ensures the transaction is either fully completed (money withdrawn and debited) or fully canceled (no money withdrawn and no change in the account).

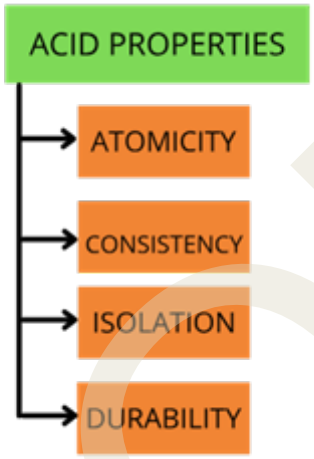


Fig 4.1.2 ACID properties

- **Consistency**
  - Refers to the correctness of data
  - data integrity must be maintained, which means that any modifications to the database should be permanent
  - Example: Raju's account balance is correct before and after he tries to withdraw money, even if something goes wrong.
- **Isolation**
  - Each transaction should operate independently of others.
  - Even if multiple transactions occur simultaneously, they should behave as if they were processed one at a time, without affecting each other.
  - Example: If someone else is using the ATM while Raju is, their transaction will not interfere with his, and he will not see their balance
- **Durability**
  - Once a transaction is committed (successfully completed), its changes are permanent, even if the system crashes.

- The database ensures that the data persists and can be recovered if needed.
- Durability property is the responsibility of the recovery subsystem in DBMS. ample**:** If Raju successfully withdraws money and the system crashes right after, his balance will still show the correct amount when the system is back online.

Recap

- Transaction is a logical unit of work
- A transaction can be either read-only or read-write.
- Successful transactions are committed.
- Failed transactions are rolled back.
- Transaction states are: New, Active, Partially Committed, Committed, Failed, Aborted, Terminated.
- ACID Properties
  - ATOMICITY - The transaction either happens completely or not at all
  - CONSISTENCY- The database stays correct before and after the transaction
  - ISOLATION-Transactions should not interfere with each other.
  - DURABILITY - Data should be saved permanently

Objective Type Questions

1. A logical unit of work done in a database is called
2. A transaction that involves only retrieval of data is a
3. The operation that is used to permanently save data is
4. An undo operation is called
5. The statement "All or Nothing" refers to
6. What is the state of a transaction when it is being processed
7. The record that keeps track of transactions in the database.
8. Ensuring consistency is the responsibility of the recovery subsystem. True or False.
9. The part of DBMS that ensures durability of transactions.
10. Two transactions T1 and T2 are being executed simultaneously. Which property of the transaction should be ensured.

Answers to Objective Type Questions

1. Transaction
2. Read-only transaction
3. Commit
4. Rollback
5. Atomicity
6. Active
7. System Log
8. False
9. Recovery sub system.
10. Isolation

Assignments

1. What are the different states of a transaction in DBMS? Explain each briefly.
2. What is a transaction in a database management system (DBMS)?
3. Explain the ACID properties of a transaction. Why are they important?
4. What is the difference between a committed transaction and a failed transaction in DBMS?
5. Give an example of a database transaction and explain how it ensures data consistency.

Reference

1. Transaction Processing: Concepts and Techniques - Jim Gray and Andreas Reuter
2. Database systems: The complete book - Hector Gracia - Mohna, Jeffery. D. Ullman and Jennifer Widom

Suggested Reading

1. Elmasri and Navathe, Fundamentals of Database Systems, 5th Edition, Pearson.
2. Abraham Silbersehatz, Henry F. Korth and Sudarshan, Database System Concepts, 6th Edition, Tata McGraw-Hill.
3. Raghurama Krishnan, Johannes Gehrke, Database Management Systems, Tata McGrawHill.
4. C.J. Date, A. Kannan, S. Swami Nadhan, An Introduction to Database systems.

UNIT 2 Serializability

Learning Outcomes

The learner will be able to:

- define a schedule.
- distinguish between serial schedule and non-serial schedule.
- describe the serializability
- explain conflict serializability and view serializability

Prerequisites

You have learned how Transaction management ensures that database operations follow the ACID properties. You have seen how commit permanently save changes and rollback undo them when needed. These concepts maintain reliability when transactions are executed. But what happens when multiple transactions occur at the same time? Can their execution order impact the final result? This is where the challenge of concurrency comes in.

Consider a scenario where two customers try to withdraw money from the same bank account simultaneously. If the database does not carefully manage the execution order, one transaction might overwrite the other's changes, leading to incorrect balances. To prevent such issues, we need serializability. Let us explore how serializability plays a crucial role in database concurrency control.

Keywords

Transaction, Schedule, Serial Schedule, Non-serial schedule, Serializable schedule, Serializability, Conflict Serializability, View Serializability

Discussion

Consider two processes, A and B, running simultaneously on a multiprogramming operating system. The operating system alternates between executing commands from process A and process B to make efficient use of the CPU. Similarly, in a DBMS, when multiple transactions run concurrently, their operations are executed in an interleaved manner. The order in which these operations are executed - (whether for processes or database transactions) - is known as a schedule. This ensures that operations are carried out efficiently, while maintaining the integrity and consistency of the database.

4.2.1 Schedule

- The order in which the operations (such as read, write, commit, etc.) of multiple transactions are executed in a database system
- The primary goal of a schedule is to ensure that concurrent transactions are executed in a way that maintains the integrity and consistency of the database
- For example, if two people, John and Sam try to withdraw money from the same ATM at the same time, the ATM handles their requests in a specific order to make sure both receive the correct amount, and the balance is updated correctly. The schedule is simply the plan that ensures these transactions happen step by step without errors.
- **There are 4 operations** in a schedule. They are:
  - read
  - write
  - commit
  - abort

Example, consider two transactions, T1 and T2 given in Figure 4.2.1 executing in an inter-leaved fashion.

T1	T2
$r_1(A)$	
$w_1(A)$	
	$r_2(B)$
	$w_2(B)$

Fig 4.2.1 Schedule S of transactions T1 and T2

The schedule S will be as follows: S: $r_1(A)$; $w_1(A)$; $r_2(B)$; $w_2(B)$;

where $r_1(A)$ and $w_1(A)$ denote read and write operations performed by T1 $r_2(B)$ and $w_2(B)$ denote read and write operations performed by T2.

4.2.2 Types of Schedules

There are different types of schedules based on how the operations are arranged and whether they maintain the required properties of transactions. Fig 4.2.2 shows the different types of schedules.

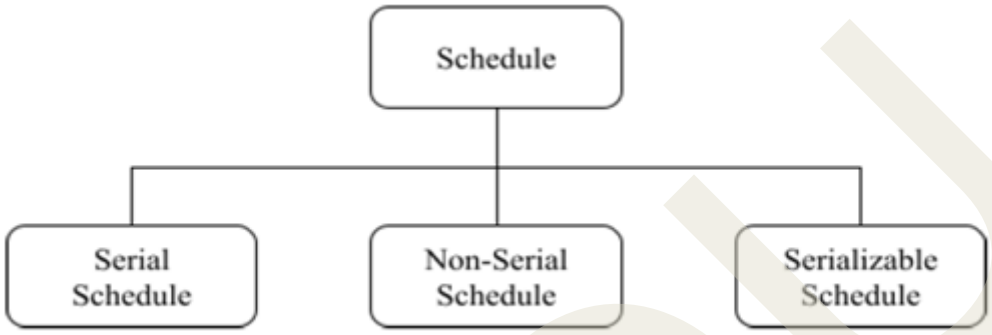


Fig 4.2.2 Schedule Types

4.2.2.1 Serial Schedule

- Transactions are executed one after the other, without any overlap or interleaving of operations
- Operations of one transaction are executed in sequence, and once a transaction finishes, the next one begins
- Only one transaction is active at a time
- The second transaction is initiated only if the first one is committed or aborted
- Database remains in a consistent state throughout the execution since transactions are executed one at a time
- Consider two transactions T1 and T2 in Fig 4.2.3. In a serial schedule either execute all the operations of T1 transaction and start T2, or execute all the

T1	T2
$r_1(A)$	
$w_1(A)$	
	$r_2(B)$
	$w_2(B)$

T1	T2
	$r_2(B)$
	$w_2(B)$
$r_1(A)$	
$w_1(A)$	

Fig 4.2.3 Schedule 1-a serial schedule of T1 and T2

operations of T2 transaction and then start T1

4.2.2.2 Non-Serial Schedule

- Operations of multiple transactions are interleaved as shown in Fig 4.2.4
- Improve system throughput and resource utilization

T1	T2
$r_1(A)$	
$w_1(A)$	
	$r_2(A)$
	$w_2(A)$
$r_1(B)$	
$w_1(B)$	

T1	T2
$r_1(A)$	
	$r_2(A)$
$w_1(A)$	
$r_1(B)$	
	$w_2(A)$
$w_1(B)$	

Fig 4.2.4 Schedule 2 -a non-serial schedule of T1 and T2

4.2.2.3 Serializable Schedule

- If the sequence of operations of a non-serial schedule is equivalent to a serial schedule, then the schedule is serializable
- Consider the following non-serial schedule of two transactions T1 and T2 in Fig 4.2.5

T1	T2
$r_1(A)$	
$w_1(A)$	
	$r_2(A)$
	$w_2(A)$
$r_1(B)$	
$w_1(B)$	

Fig 4.2.5 Schedule 3 - showing only the read and write instructions

- A non-serial schedule will be serializable if its result is equal to the result of its transactions executed serially. The above schedule is converted to a serial

T1	T2
$r_1(A)$	
$w_1(A)$	
$r_1(B)$	
$w_1(B)$	$r_2(A)$
	$w_2(A)$

T1	T2
	$r_2(A)$
$r_1(A)$	$w_2(A)$
$w_1(A)$	
$r_1(B)$	
$w_1(B)$	

Fig 4.2.6 Schedule 4 - a Non serial schedule that is equivalent to Serial schedule

schedule as shown in Fig 4.2.6

- Non-serial schedules may lead to data inconsistency. Hence it is necessary to identify whether non-serial schedules are serializable.
- Serializable schedule improves throughput and resource utilization

4.2.3 Types of Serializability

There are different methods to determine if a schedule is serializable. The most common ones include

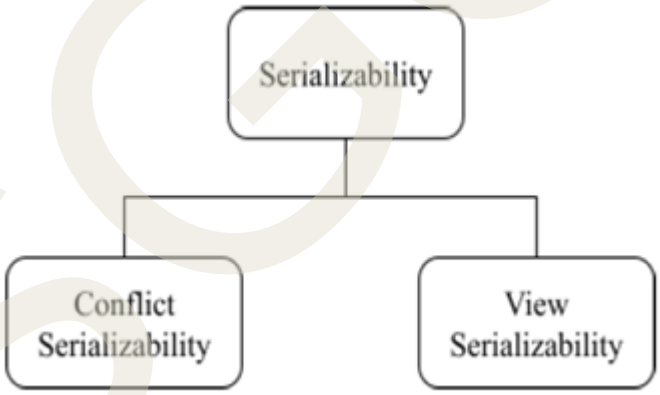


Fig 4.2.7 Types of Serializability

4.2.3.1 Conflict Serializability

- A **conflict serializable schedule** allows transactions to run concurrently, but the outcome will be as if they had been executed sequentially, without overlapping.
- Two operations are said to be in **conflict** if they meet all the following conditions:
  - They belong to different transactions.

- They operate on the same data item.
- At least one of the operations is a write

Consider a schedule in Figure 4.2.8

T1	T2
$r_1(A)$	
$w_1(A)$	
	$r_2(A)$
$r_1(B)$	

Fig 4.2.8 Schedule 5-A conflicting schedule

- In the above schedule $w_1(A)$ and $r_2(A)$ are conflicting operations. They belong to different transactions. They operate on the same data and one of them is a write operation.
- To determine whether a schedule is conflict serializable, we can use a **precedence graph** (also called a **serializability graph)**

Algorithm for testing Conflict Serializability

Step 1: Create nodes for transactions

For each transaction Ti in the schedule S, create a node labeled Ti in the precedence graph.

Step 2: Handle read-write conflicts

For every pair of operations where Tj executes a read(A) after Ti executes a write(A), create a directed edge from Ti to Tj (i.e., Ti → Tj).

Step 3: Handle write-read conflicts

For every pair of operations where Tj executes a write(A) after Ti executes a read(A), create a directed edge from Ti to Tj (i.e., Ti → Tj)

Step 4: Handle write-write conflicts

For every pair of operations where Tj executes a write(A) after Ti executes a write(A), create a directed edge from Ti to Tj (i.e., Ti → Tj).

Step 5: Check for cycles

The schedule S is conflict serializable if and only if the precedence graph has no

cycles. If the graph contains a cycle, the schedule is not conflict serializable

Example: Consider the serial schedule in Figure 4.2.9

T1	T2
$r_1(A)$	
$w_1(A)$	
	$r_2(A)$
	$w_2(A)$

Fig 4.2.9 Schedule 6-serial schedule

The precedence graph for the above schedule is as shown in Figure 4.2.10

Fig 4.2.10 Precedence graph of Schedule 6

Consider a non-serial schedule in Figure 4.2.11

T1	T2
$r_1(A)$	
$w_1(A)$	
	$r_2(A)$
	$w_2(A)$
$r_1(B)$	
$w_1(B)$	

Fig 4.2.11 Schedule 7-non serial schedule

In Schedule 7 $w_1(A)$ and $r_2(A)$ are conflicting operations. The precedence graph for the above schedule is

Fig 4.2.12 Precedence graph of Schedule 7 is equivalent to Schedule 6

The above graph is the same as the schedule explained in the previous example, hence the schedule is serializable.

4.2.3.2 View Serializability

- View serializability check if the outcome of the transactions is the same as it would be in a serial execution
- The schedule is view serializable if the read and write operations are arranged in such a way that the final database state is the same as if the transactions were executed serially, even if the operations are interleaved
- Two schedules T1 and T2 are said to be view equivalent, if they satisfy all the following conditions

1. Initial Read: If transaction T1 in schedule S1 reads a data item A for the first time, then in schedule S2, T1 must also read A as its first read. This ensures that the first read of a data item is the same in both schedules.

   Example: If T1 reads A first in S1, then in S2, T1 should also read A first.

2. Updated Read: If T1 reads A and T2 writes A in both schedules, then T1 must read the value that T2 wrote in both schedules.

   Example: If T2 writes A in S1 and T1 reads A, then in S2, T1 should read the same value of A written by T2.

3. Final Write: If T1 writes A in schedule S1 and T2 writes A in schedule S2, the final write of A in both schedules must be the same.

   Example: If T1 writes A in S1 and T2 writes A in S2, both schedules should have the same final value of A.

Recap

- The order of execution of operations of different transactions is called schedule.
- A schedule in which transactions are executed one after the other is a serial schedule.
- A schedule in which the operations of transactions are executed in an interleaved manner is called non-serial schedule.
- A schedule is said to be serializable if it is equivalent to a serial schedule
- Serializability ensures consistency in the database

Objective Type Questions

1. The order of execution of operations is called
2. How many types of schedules are there?
3. Interleaving is allowed in the serial schedule. True or False.
4. Which graph is used to test conflict serializability
5. What type of transactions are not allowed in a serializable schedule?
6. What must be absent in a precedence graph for a schedule to be serializable?
7. What is the simplest form of a schedule where transactions execute one after another?
8. What is the main goal of serializability?
9. Schedules can be constructed using
10. Which type of schedule allows transactions to be executed in any order, if the final result is consistent?

Answers to Objective Type Questions

1. Schedule
2. 3

3. False
4. Precedence Graph
5. Conflicting
6. Cycle
7. Serial
8. Consistency
9. Precedence Graph
10. Serializable

Assignments

1. Explain the concept of serializability in database transactions. Why is it important?
2. Differentiate between conflict serializability and view serializability. Provide examples for each.
3. What is a precedence graph? How is it used to test conflict serializability?
4. Given a schedule of transactions, explain how you would determine whether the schedule is conflict serializable or not.
5. Write a short note on the conditions for a schedule to be view serializable. How does view serializability differ from conflict serializability?

Reference

1. Transaction Processing: Concepts and Techniques - Jim Gray and Andreas Reuter
2. Database systems: The complete book - Hector Gracia - Mohna, Jeffery. D. Ullman and Jennifer Widom

Suggested Reading

1. Elmasri and Navathe, Fundamentals of Database Systems, 5th Edition, Pearson.
2. Abraham Silbersehatz, Henry F. Korth and Sudarshan, Database System Concepts, 6th Edition, Tata McGraw-Hill.
3. Raghurama Krishnan, Johannes Gehrke, Database Management Systems, Tata McGrawHill.
4. C.J. Date, A.Kannan, S.Swami Nadhan, An Introduction to Database systems.

UNIT 3

Types of Failures and Recovery Outlines

Learning Outcomes

The learner will be able to:

- list the different types of failures that can occur in computing systems.
- define the concept of system failures and their impact on data integrity, performance, and user experience.
- identify the role of recovery outlines in ensuring the restoration of systems to a consistent state after failure.
- list the best practices for creating recovery plans that minimize downtime and ensure data consistency.

Prerequisites

Before studying the topic of Types of Failures and Recovery Outlines, it is essential to have a basic understanding of computer systems and how they operate. This includes knowledge of system architecture, database management systems, and basic programming concepts. Familiarity with how software interacts with hardware and how data is processed and stored in various systems will provide a foundation for understanding how and why failures occur. Additionally, an understanding of basic networking concepts and how communication happens between systems will be helpful, as failures can often be related to network issues or server downtime.

Furthermore, a solid understanding of transaction management in databases is crucial. This includes concepts such as atomicity, consistency, isolation, and durability (ACID properties), which are foundational to understanding how transactions are executed and maintained in a system. Knowledge of error handling and basic debugging techniques in software development will also be beneficial, as it will allow you to grasp the process of identifying, diagnosing, and resolving issues that result in system or transaction failures. Understanding these prerequisites will enable you to effectively learn about recovery strategies and failure management techniques in complex systems.

Keywords

System Failure, Recovery Mechanisms, Transaction Failure, Checkpointing, Roll-back, Data Integrity, Fault Tolerance

Discussion

4.3.1 Introduction to Failures and Recovery

Failures are disruptions that prevent a computing system or process from working as it should. These disruptions can occur due to various reasons, such as hardware issues, software errors, or unexpected external events. Recovery refers to the steps taken to restore the system to its proper state after a failure happens. Understanding failures and planning for recovery is crucial to keep systems reliable and ensure they work consistently, even when problems arise.

4.3.1.1 What is a Failure?

A failure happens when a system, application, or device cannot perform its intended task. For example, a hardware failure might occur if a hard drive crashes, or a software failure might happen due to a bug in a program. Failures can also result from external causes, like a power outage or a network disruption. They are typically categorized into types such as hardware failures, software failures, or transaction failures in databases. Knowing what type of failure has occurred helps in deciding how to fix the problem and prevent it in the future.

4.3.1.2 Importance of Recovery in Computing Systems

Recovery is essential to keep systems running smoothly after a failure. Without a good recovery plan, even small issues can cause significant problems like data loss, long downtimes, or financial setbacks. Recovery techniques, such as saving checkpoints or rolling back incomplete tasks, help bring the system back to a consistent and functional state. For instance, in databases, recovery ensures that all completed transactions are saved, while unfinished ones are safely reversed. Overall, recovery strategies are key to ensuring that systems remain reliable and users can trust them to work correctly.

4.3.2 Types of Failures

Failures in computing systems can happen for many reasons, and understanding the type of failure is essential for solving the problem effectively. Failures can stem from hardware issues, software bugs, transaction problems, communication breakdowns, or human mistakes. Each type of failure affects the system differently, and addressing them ensures the system stays reliable and efficient.

4.3.2.1 Hardware Failures

Hardware failures occur when physical components of a computer system stop working as expected. This could involve hard drives crashing, memory becoming

corrupted, or power supplies failing. These issues can lead to data loss or system downtime. For example, a server might stop functioning if its hard drive fails. To prevent such problems, solutions like creating data backups, using redundant hardware, and monitoring system health are commonly used.

4.3.2.2 Software Failures

Software failures happen when programs or applications don't work as they should. Common causes include bugs in the code, software updates that cause conflicts, or a lack of sufficient resources like memory or processing power. For instance, an application might crash because of a coding error. Fixing software failures often requires debugging, applying updates, or reverting to an earlier, stable version. Regular testing and maintenance can also help prevent software failures.

4.3.2.3 Transaction Failures

Transaction failures are specific to systems that handle multiple operations in a sequence, such as databases. These failures occur when a transaction cannot be completed successfully, often due to insufficient resources, a system crash, or a violation of data rules. For example, in an online purchase, if money is debited but the order is not confirmed, the system is left in an inconsistent state. Recovery processes like rolling back or retrying the transaction ensure that the data remains accurate and consistent.

4.3.2.4 Communication Failures

Communication failures occur when there is a problem in transferring data between devices or systems. This might happen because of network outages, poor connectivity, or transmission errors. For example, if a video call drops due to a weak internet connection, it is a communication failure. Systems address these issues by using techniques like error detection, retries, and backup communication channels to ensure data is successfully delivered.

4.3.2.5 Human Errors and Failures

Human errors are mistakes made by people while using or managing systems. These could include accidental deletion of important files, incorrect system configurations, or misuse of tools. For instance, a user might accidentally delete key data from a database. To reduce the chances of human errors, organizations can provide proper training, implement automated systems, and set up strict access controls. Even though human mistakes cannot always be avoided, user-friendly designs and clear instructions can help minimize their impact.

4.3.3 Impact of Failures

Failures in computing systems can cause a range of challenges, from reduced system performance to data loss and downtime. These disruptions can affect user experience, lead to financial losses, and compromise system reliability. Understanding the impact of failures helps in designing systems that are better equipped to handle such challenges effectively.

4.3.3.1 Effects on System Performance

Failures can significantly affect how efficiently a system operates. For instance, if a server crashes, the system may slow down or stop responding entirely. This not only delays processes but can also affect other systems connected to it. For example, a web application may face reduced performance due to issues with its database server. To address this, strategies like load balancing and regular monitoring can help maintain consistent performance.

4.3.3.2 Data Loss and Corruption

One of the most serious impacts of failures is the loss or corruption of valuable data. A hardware failure, such as a damaged hard drive, can result in the permanent loss of files. Similarly, software issues like incomplete transactions can corrupt databases, making the information unusable. Losing critical data, such as customer records or financial transactions, can harm both users and organizations. Regular data backups and recovery plans are essential to reduce the risks associated with such failures.

4.3.3.3 Downtime and System Availability

Failures often cause downtime, which means systems are unavailable for use. This can disrupt business operations, reduce productivity, and lead to financial losses. For example, if an online shopping platform goes down during a sale, customers may leave and revenue will be lost. Downtime also affects user trust, as people expect systems to be reliable. To minimize downtime, organizations use strategies like fault-tolerant system designs and disaster recovery plans to ensure systems remain available even during unexpected issues.

4.3.4 Recovery Mechanisms

Recovery mechanisms play a vital role in maintaining the stability and reliability of computer systems after a failure. They are designed to restore the system to its normal state, prevent data loss, and minimize downtime. These techniques ensure that disruptions don't compromise the integrity of the system and allow processes to resume smoothly and efficiently. Let's explore some common recovery mechanisms.

4.3.4.1 Rollback Mechanism

The rollback mechanism helps the system return to its last known consistent state after a failure. This is especially useful in cases where a process or transaction is incomplete, such as in a banking transaction interrupted by a power outage. Rollbacks work by reversing any changes made during the incomplete transaction using stored logs or backups. For instance, if a user tries to transfer money and the process fails, a rollback ensures that no partial transfer or incorrect data is recorded. This mechanism safeguards the system's data integrity.

4.3.4.2 Checkpointing

Checkpointing involves saving a snapshot of the system's state at specific intervals. If a failure occurs, the system can restart from the most recent checkpoint instead of

starting over completely. This approach saves time and reduces the chances of data loss. For example, when performing a long-running task, such as a scientific simulation, checkpointing ensures that progress is saved periodically. If an interruption happens, the simulation can pick up from the last checkpoint rather than starting from scratch. This method is widely used in large-scale systems to increase efficiency and reliability.

4.3.4.3 Replication and Redundancy

Replication and redundancy involve creating backup copies of data or system components to ensure smooth operation in case of a failure. For instance, cloud storage services maintain multiple copies of your data across different servers. If one server goes down, the data can still be accessed from another location. Similarly, redundant hardware components like power supplies or storage drives can take over when one fails, ensuring that the system remains operational. This approach is especially important in critical systems where continuous availability is essential, such as e-commerce or healthcare systems.

4.3.4.4 Deferred and Immediate Recovery

Deferred and immediate recovery strategies focus on when the recovery process begins after a failure. Deferred recovery waits until the system stabilizes or additional information is available before taking action. For example, in database systems, incomplete transactions can be resolved after the system has been restored. On the other hand, immediate recovery begins corrective actions as soon as a failure is detected. This is crucial in systems like online trading platforms, where even brief downtime can have significant consequences. Both approaches are designed to bring the system back to normal while considering the urgency and nature of the failure.

4.3.5 Steps in Recovery Process

Recovering from a system failure involves a series of steps that help restore stability and functionality. These steps focus on identifying the problem, fixing any inconsistencies, and bringing the system back to normal operations. Each stage in the recovery process is essential to ensure data integrity and minimize the disruption caused by the failure. Below is a simplified explanation of these steps.

4.3.5.1 Detecting Failures

The first step is recognizing that a failure has occurred. Systems use tools like error logs, monitoring software, or alerts to identify problems as soon as they happen. For example, if a server goes offline, an automated alert can notify system administrators immediately. Detecting failures quickly helps to address issues before they grow into more significant problems, ensuring smoother recovery.

4.3.5.2 Restoring Data to a Consistent State

After identifying the failure, the next task is to make sure the system's data is consistent and reliable. This might involve rolling back incomplete transactions or using a previous checkpoint to recover lost data. For example, in an online shopping platform, if an order

transaction is interrupted, rollback mechanisms ensure the customer's account balance and order details remain accurate. This step prevents data corruption and ensures the system remains trustworthy.

4.3.5.3 System Restart and Recovery Phases

The final step is restarting the system and gradually resuming normal operations. This might involve restarting critical services first, followed by non-essential ones. For instance, in a hospital's database system, restoring patient records and scheduling systems would take priority over less urgent functions. Restarting in phases ensures that the most important tasks are addressed first, reducing downtime for critical services.

These steps form the foundation of an effective recovery process. By detecting issues early, restoring data properly, and carefully restarting operations, organizations can minimize disruptions and ensure their systems run reliably.

4.3.6 Recovery Strategies

When a system experiences a failure, recovery strategies are put in place to bring it back to a working state. These strategies help minimize the impact of failures, ensuring the system gets back online quickly and with as little data loss as possible. Different strategies are used depending on the type of failure and how critical it is to restore the system.

4.3.6.1 Synchronous vs. Asynchronous Recovery

There are two main types of recovery: synchronous and asynchronous. In synchronous recovery, the system works to restore itself immediately when a failure occurs. It's like the system takes action right away, ensuring everything is in sync in real time. This is useful for systems where data consistency is very important, like in financial systems.

Asynchronous recovery, on the other hand, happens at a later time. The system can continue working normally, and the recovery happens in the background. This method doesn't interrupt the system's operations but may take longer to restore everything fully. Depending on the system's needs, one method might be preferred over the other.

4.3.6.2 Crash Recovery

Crash recovery is used to bring the system back to its last known working state after a sudden crash, like a power outage or system shutdown. The system looks at logs or saved points in time to figure out where it left off and restores everything back to a consistent state. For example, if you're in the middle of making a transaction on an online store and the system crashes, crash recovery ensures that the transaction is either completed or canceled, keeping everything in order.

4.4.6.3 Partial Recovery

Partial recovery focuses on fixing only the parts of the system that were affected by the failure, rather than restoring everything. For example, if a part of the system, like the database, crashes but other parts, like the user interface, are still working fine,

partial recovery can restore just the database. This approach helps get the system back up and running quickly without having to fix things that aren't broken.

4.3.6.4 Full Recovery

In full recovery, the entire system is restored to its state before the failure. This is useful when a significant part of the system, like a server or database, crashes. Full recovery uses backups, logs, or saved states to ensure everything is returned to normal, from the data to the services. It might take longer but is crucial when the failure is severe and affects multiple components of the system.

Each of these strategies has its place, and the best one depends on how severe the failure is and how much downtime the system can afford. The goal is always to get the system back online as quickly and smoothly as possible while minimizing any potential data loss.

4.3.7 Importance of Backup and Recovery Plans

Backup and recovery plans are very important because they help ensure that data and systems can be restored when something goes wrong. These plans act like a safety net for organizations, allowing them to recover from problems such as hardware failure, software issues, or even natural disasters. Without proper backup and recovery plans, there's a higher chance of losing important data, which can lead to a lot of problems, including costly downtime and damage to the business.

Having a backup and recovery plan is a way to make sure business operations continue smoothly, even when things don't go as planned. These plans outline what needs to be done to restore data and systems, minimizing the effect of failures. With a good backup plan, organizations can quickly recover and keep their operations running, even in tough situations.

4.3.7.1 Regular Backup Techniques

Regular backups are one of the best ways to protect important data. By backing up data frequently, organizations can make sure that even if something goes wrong, they won't lose too much. There are different ways to back up data, such as full backups, incremental backups, and differential backups.

A full backup copies all of the data, while an incremental backup only saves the changes that have been made since the last backup. A differential backup saves the changes made since the last full backup. These different backup techniques make it easier to restore systems to various points in time, which can be very helpful when recovering from a problem.

4.3.7.2 Disaster Recovery Planning

Disaster recovery planning is about preparing for unexpected events that could disrupt business, like fires, floods, or cyberattacks. This plan helps organizations quickly recover and get back to work after a disaster, reducing the amount of time they are offline and preventing serious problems.

A disaster recovery plan includes steps for identifying which systems are most important, setting priorities for recovery, and defining how long it will take to restore systems. It also includes instructions for communicating during a crisis. Having a clear disaster recovery plan helps organizations react quickly to emergencies, allowing them to return to normal operations faster.

4.3.7.3 Best Practices in Data Protection

Protecting data is key to keeping it safe and secure. To make sure data is well-protected, organizations should follow some best practices. For example, they can encrypt backups to keep them secure, test backups regularly to make sure they'll work when needed, and store backup copies in different locations.

Other important practices include automating backups to avoid mistakes, restricting access to backups to prevent unauthorized users from making changes, and regularly reviewing backup procedures to ensure they are still effective. By following these best practices, organizations can be confident that their data is safe and can be recovered if something goes wrong.

Recap

- Backup and recovery plans are essential for protecting data and systems from unexpected failures.
- These plans help businesses quickly get back on track when things go wrong, minimizing disruptions.
- Having a solid recovery plan ensures that even if there's a problem, operations can continue smoothly.
- Regularly backing up data is a simple but important way to keep everything safe.
- Full backups copy all your data, while incremental and differential backups focus on the changes made.
- Incremental backups are quicker because they only save the latest changes but take longer to restore.
- Differential backups are a middle ground, saving changes since the last full backup, making them easier to restore.
- A disaster recovery plan helps guide you through the steps to take in case of big events like fires or cyberattacks.
- These plans are key to helping an organization bounce back quickly after something goes wrong.
- Good disaster recovery plans include figuring out which systems are most important, setting priorities, and figuring out how long it will take to recover.

- Being able to communicate well during a disaster is really important for a recovery plan to work.
- It's a good practice to make sure backup data is encrypted so it stays safe from unauthorized access.
- Testing backups regularly helps ensure they will work when you need them the most.
- Backups should be stored in different locations to protect them from disasters like floods or fires.
- Automating the backup process makes it less likely for mistakes to happen and ensures it's done on time.
- It's important to restrict backup access to authorized personnel only to keep data safe.
- Make sure backup procedures are reviewed regularly to keep them up-to-date and effective.
- Deciding how often to back up data depends on the needs of the business and how critical the data is.
- Keep the disaster recovery plan updated to reflect any changes in the business environment.
- Backup and recovery planning helps reduce downtime and minimize the risk of losing important data.
- Following good practices for backup and recovery ensures that even during unexpected events, your business can continue operating.

Objective Type Questions

1. What is the process of making copies of data to protect it from loss?
2. What type of backup copies only the changes made since the last backup?
3. What is the plan that helps restore a system after a disaster?
4. What is the full copying of all data, regardless of changes?
5. What type of backup stores all changes since the last full backup?
6. What process ensures that backup data is protected from unauthorized access?

7. What is a key part of a disaster recovery plan that prioritizes which systems need to be restored first?
8. What ensures that the backup plan is effective when it is needed?
9. What type of disaster recovery strategy involves recovering data from an offsite location?
10. What term describes the practice of automating the backup process?
11. What type of backup is faster because it only saves new data?
12. What should backup data be stored in to protect it from disasters?
13. What term refers to unexpected events like cyberattacks or natural disasters that cause system failures?
14. What should be reviewed regularly to ensure it remains effective?
15. What kind of backup can be restored the quickest?

Answers to Objective Type Questions

1. Backup
2. Incremental
3. Recovery
4. Full
5. Differential
6. Encryption
7. Prioritization
8. Testing
9. Remote
10. Automation
11. Incremental
12. Multiple
13. Disasters
14. Plan

15. Full

Assignments

1. Explain the different types of backup techniques (full, incremental, differential) and analyze the advantages and disadvantages of each in the context of data recovery.
2. Discuss the importance of having a disaster recovery plan in place for an organization. How do regular backups contribute to minimizing downtime during a system failure?
3. Compare and contrast synchronous and asynchronous recovery methods. Which method would be more suitable for a large enterprise with critical data, and why?
4. Evaluate the best practices in data protection and discuss how these practices can be integrated into a comprehensive backup and recovery plan to ensure business continuity.
5. Analyze the role of encryption in backup and recovery processes. How does encryption help in securing backup data, and what are the challenges associated with its implementation?

Reference

1. Transaction Processing: Concepts and Techniques - Jim Gray and Andreas Reuter
2. Database systems: The complete book - Hector Gracia - Mohna, Jeffery. D. Ullman and Jennifer Widom

Suggested Reading

1. Teate, Renee MP. SQL for Data Scientists: A Beginner's Guide for Building Datasets for Analysis. John Wiley & Sons, 2021.
2. Groff, James R., Paul N. Weinberg, and Andrew J. Oppel. SQL: the complete reference. Vol. 2. McGraw-Hill/Osborne, 2002.
3. Beaulieu, Alan. Learning SQL: master SQL fundamentals. O'Reilly Media, 2009.
4. CELKO, J., and S. Q. L. Joe Celko's. "for Smarties: Advanced SQL Programming. 3. vyd. Massachusetts." (2005).

UNIT 4 Deadlocks

Learning Outcomes

The learner will be able to:

- describe locking mechanism of concurrency control
- define deadlock in database management systems
- explain deadlock detection
- describe deadlock prevention mechanisms

Prerequisites

In everyday life, we often experience situations where multiple people need access to the same limited resource. For example, imagine two people trying to cross a narrow bridge from opposite sides at the same time. Since the bridge is too narrow for both to pass together, they block each other and cannot move forward. This situation is similar to a **deadlock** in computer systems.

When multiple processes in a database or operating system try to access shared resources, they must follow rules to avoid conflicts. If these rules are not properly managed, processes can get stuck, waiting indefinitely for resources held by each other. This leads to a state called **deadlock**, where none of the processes can proceed.

To understand deadlock and its related concepts, such as locks, timestamps, concurrency control, and starvation, we will explore how systems manage multiple transactions efficiently while preventing conflicts and ensuring smooth execution.

Keywords

Concurrency Control, Lock, Deadlock, Timestamp, Starvation, Mutual exclusion, Circular wait

Discussion

A deadlock occurs when two or more processes in a system are blocked forever, each waiting for a resource held by another. This results in a cycle of dependencies where no process can proceed, and the system is in a state of constant waiting. Deadlock can arise in concurrent systems, such as operating systems or databases, and needs to be managed carefully through detection, prevention, or avoidance techniques to ensure smooth execution of transactions.

4.4.1 Deadlock

An undesirable situation in which two or more transactions wait indefinitely for each other to release locks.

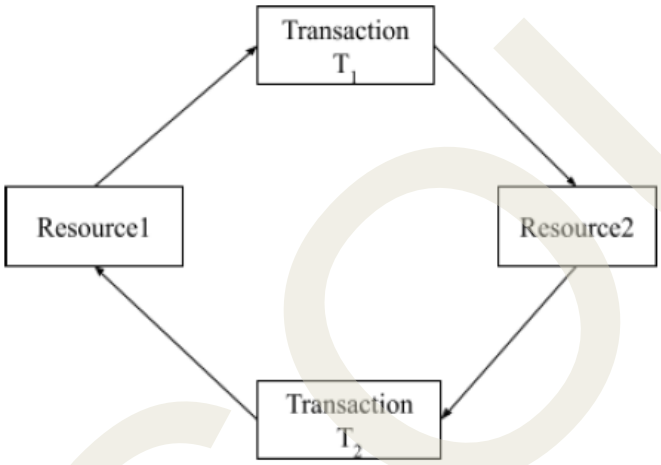


Fig 4.4.1 A system in deadlock

In the above figure 4.4.1, T_1 has Resource1 and needs Resource2. Similarly T_2 has Resource2 and needs Resource1. Each of these processes needs other's resources to complete but neither of them is willing to give away their resources. So, T_1 and T_2 are in deadlock. All operations stop and stop forever unless the DBMS detects the deadlock and aborts one of the transactions.

4.4.2 Conditions for deadlock

A deadlock will occur if the following conditions hold:

a. Mutual Exclusion

There should be a resource that can only be owned by one transaction at a time as shown in Fig.4.4.2.

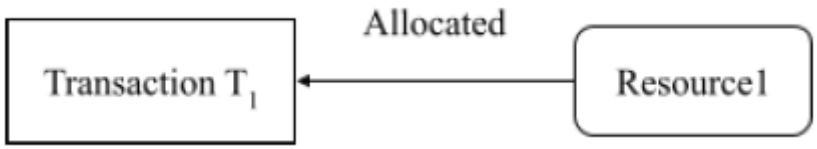


Fig 4.4.2 Mutual exclusion condition

b. Hold and Wait

A transaction holds more than one resource and still requests for more resources as shown in Fig.4.4.3.

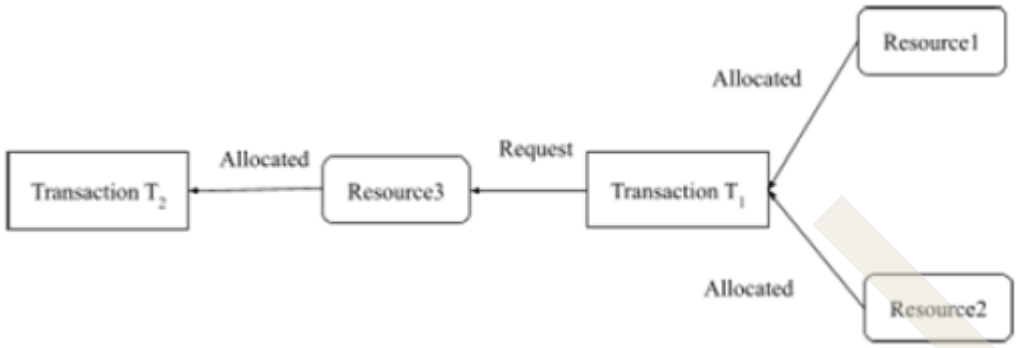


Fig 4.4.3 Hold and Wait condition

c. No Preemption

Once a resource is allocated to a transaction, it cannot be forcibly taken away (preempted) by the system. A transaction can only release the resource voluntarily after completing its operation.

d. Circular Wait

A transaction is waiting for the resource held by the second transaction, which is waiting for the resource held by a third transaction and so on. The last transaction is waiting for a resource held by the first transaction. This forms a circular chain as shown in Fig.4.4.4.

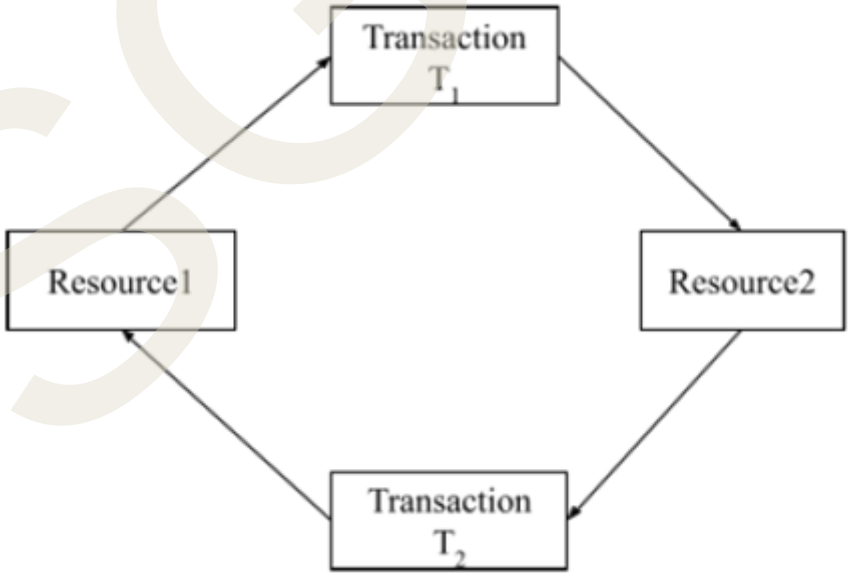


Fig 4.4.4 Circular wait Condition

These conditions are called Coffman Conditions.

4.4.3 Deadlock Prevention

Deadlock prevention ensures that the system does not enter a deadlock state by imposing restrictions on transaction execution. It is particularly suitable for large databases where deadlock detection and resolution may be costly. The system closely examines each transaction before it is executed to ensure that it does not lead to a deadlock. If there is even a chance that a transaction leads to a deadlock, it should never be allowed to run.

> A lock is a control mechanism associated with a database item that manages access to it. Before a transaction can read or modify a data item, it must acquire a lock; if another transaction holds the lock, it must wait.

4.4.3.1 Deadlock Prevention Protocols

1. Preemptive Locking

A transaction must lock all required data items in advance before execution. If any item cannot be locked, the transaction does not acquire any locks and must wait or restart. This method reduces deadlocks but limits concurrent execution.

How does Preemptive Locking Works?

a. Acquire All Locks in Advance

- A transaction must ask for all the locks it needs before it starts doing any work.
- If any lock is unavailable, the transaction cannot start and must wait until it can lock all the resources it needs.

b. Waiting for Lock Availability

- If some locks are already taken by other transactions, the requesting transaction must wait until those locks are released.
- During this waiting time, the transaction cannot do anything. It will only continue once it gets all the locks it needs.

c. No Partial Locking

- A transaction cannot start using the data it has locked partially. It must wait until it locks all the required resources.
- If it still can't lock everything, it may restart and try to acquire the locks again.

d. Transaction Abort or Restart

- If a transaction cannot get all the locks because other transactions are holding them, it may be aborted.

- The transaction can restart later, going through the process of acquiring all locks again before proceeding.

2. Ordering of Resources (Lock Ordering)

All data items in the database are assigned a predefined order, and transactions must request locks in that order. This prevents circular wait conditions. However, this method requires that the system or programmer know the order of items in advance, which may not always be practical.

For example, if resources are ordered as A, B, and C, a transaction must always request locks in this order (i.e., A first, then B, then C).

If a transaction requests a resource out of order, it is forced to wait or restart. This approach avoids situations where transactions form a circular wait.

4.4.3.2 Transaction Timestamp

- A timestamp (TS) is a unique identifier assigned to each transaction, usually based on its start time.
- It helps determine the relative order of transactions in concurrency control mechanisms.
- **Notation:** TS(T) represents the timestamp of transaction T.
- If T_1 starts before T_2, then T1 has an older timestamp:

  $TS(T_1) < TS(T_2)$.

- There are two deadlock prevention schemes based on timestamps.
  - Wait-Die Scheme (Non-preemptive)
  - Wound-Wait Scheme (Preemptive)

a. Wait-Die Scheme (Non-preemptive)

- Older transactions (with smaller timestamps) are allowed to wait for a younger transaction to release the resource.
- Younger transactions (with larger timestamps) die (are aborted) if they request a resource held by an older transaction.
- If $TS(T_1) < TS(T_2)$, then T_1 is older than T_2.
- T_1 is allowed to wait for the data item which will be free when T_2 has completed its execution.
- Otherwise abort T_1 and restart it later with the same timestamp.

Example:

- T_1 (older) requests a resource held by T_2 (younger) → T_1 wait.

- T_2 (younger) requests a resource held by T_1 (older) → T_2 is aborted ("dies") and restarted later.

b. Wound-Wait Scheme (Preemptive)

- Older transactions "wound" (forcefully abort) younger transactions instead of waiting.
- If an older transaction requests a resource held by a younger transaction, it forces the younger transaction to abort (wound).
- If a younger transaction requests a resource held by an older transaction, it is forced to wait.

Example:

- If TS(T_1) < TS(T_2), then T_1 is older than T_2.
- T_1 (older) requests a resource held by T_2 (younger) → T_2 is aborted ("wounded") and restarted later.
- T_2 (younger) requests a resource held by T_1 (older) → T_2 wait.

4.4.4 Deadlock Detection

- Deadlock detection is a more practical approach than prevention in systems where deadlocks are rare and preventing them entirely may reduce performance.
- The DBMS periodically checks whether a deadlock has occurred by examining waiting transactions and their resource dependencies.
- A common method for detecting deadlocks is using a **Wait-for Graph (WFG)**:
  - In the wait-for Graph Each transaction is a node.
  - An edge from T_1 → T_2 means T_1 is waiting for a resource held by T_2.
  - If the graph contains a cycle, a deadlock exists.

4.4.4.1 Wait-for graph

- A Wait-for Graph is drawn based on the currently executing transactions and locks on data items.
- A node is created for each current transaction.
- If a transaction T_1 is waiting for a data item X that is locked by another transaction T_2, a directed edge is drawn from T_1 to T_2.
- When T_2 releases the lock on X, the corresponding edge is removed from the graph.

- If the graph contains a cycle, it indicates a deadlock state.
- The system must check for a cycle every time an edge is added to the graph.

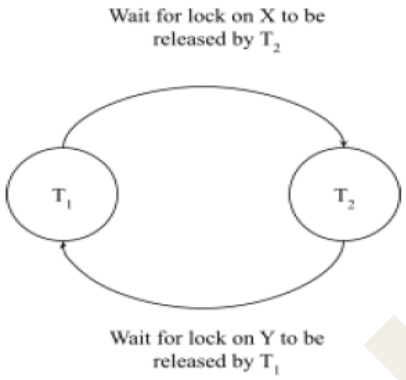


Fig 4.4.5 Wait-for graph

4.4.4.2 Timeouts in Deadlock Handling

- The system assigns a timeout period to each transaction.
- If a transaction waits longer than the timeout period while trying to acquire a lock, it is assumed to be in a deadlock.
- The system may abort the transaction to break the deadlock and free up resources.
- This approach is simple but may lead to unnecessary transaction rollbacks if the timeout is set too short.

4.4.5 Starvation

- Starvation occurs when a transaction is indefinitely delayed due to an unfair lock allocation scheme.
- This happens when certain transactions are continuously given priority over others, preventing some transactions from ever acquiring the required resources.
- As a result, the affected transaction remains waiting indefinitely and cannot proceed.
- Starvation can occur due to:
  - Priority-based scheduling, where high-priority transactions are always executed first.
  - Frequent rollbacks of low-priority transactions.
  - Resource preemption policies that always favor certain transactions.

Recap

Concurrency Control

- Manages simultaneous transactions to maintain data consistency and integrity.
- Prevents issues like lost updates, dirty reads, and uncommitted data.
- Locking Mechanisms help control concurrent execution.

Locks

- A lock is associated with a data item to control access.
- A transaction must acquire a lock before accessing a data item.
- If a data item is already locked, the transaction must wait.

Deadlock

- Occurs when two or more transactions wait indefinitely for each other's resources.
- Transactions form a cycle of dependencies, blocking execution.

Conditions for Deadlock (Coffman Conditions)

1. **Mutual Exclusion** – A resource can be held by only one transaction at a time.
2. Hold and Wait – Transactions hold resources while waiting for others.
3. No Preemption – Resources cannot be forcibly taken away from a transaction.
4. Circular Wait – Transactions form a circular dependency.

Deadlock Prevention

- Ensures the system never enters a deadlock by imposing restrictions.
- Preemptive Locking: Transactions lock all required resources in advance.
- Lock Ordering: Transactions request locks in a predefined order.

Transaction Timestamp-Based Deadlock Prevention

- Timestamps (TS) determine the order of transaction execution.
    1. Wait-Die (Non-preemptive)
- Older transactions wait; younger transactions abort and restart.

2. Wound-Wait (Preemptive)

- Older transactions abort (wound) younger ones; younger ones wait.

Deadlock Detection

- More practical when deadlocks are rare.
- Uses a Wait-for Graph: A cycle in the graph indicates a deadlock.
- The system checks for cycles when transactions request resources.

Timeouts in Deadlock Handling

- A timeout period is assigned to transactions.
- If a transaction waits too long, it is assumed to be in a deadlock and aborted.
- Simple but may cause unnecessary rollbacks if the timeout is too short.

Starvation

- Occurs when some transactions wait indefinitely due to an unfair lock allocation scheme.
- Causes:
  - Priority-based scheduling (low-priority transactions keep waiting).
  - Frequent rollbacks of certain transactions.
  - Resource preemption policies favoring specific transactions.
- Prevention Methods: Aging, fair scheduling, and priority adjustments.

Objective Type Questions

1. The variable that indicates the status of a data item in the database.
2. A protocol that guarantees that the system will never enter a deadlock state is
3. Suppose, Transaction T_1 holds a lock on some rows in the StudentsCourse table and needs to update some rows in the Student table. Simultaneously, Transaction T_2 holds locks on those very rows which T_1 needs to update in the Student table but needs to update the rows in the Course table held by TransactionT_1 .This scenario is an example of.

4. A cycle in the Wait-for graph indicates a deadlock. True or False.
5. Wound-wait is a non preemptive technique. True or False
6. In which technique older transactions have to wait for younger transactions to release the resource.
7. A unique identifier assigned for each transaction based on the order in which they are started.
8. Two wait schemes in deadlock are
9. The principal methods for dealing with deadlock are
10. The scheme in which younger transactions are forced to release the locks is

Answers to Objective Type Questions

1. Lock
2. Deadlock Prevention
3. Deadlock
4. True.
5. False
6. Wait-die
7. Timestamp.
8. Wait-die and Wound-wait
9. Deadlock detection and Deadlock prevention.
10. Wound-wait

Assignments

1. Explain the deadlock with an example. Describe why deadlock is a critical issue in concurrency control
2. Discuss the four Coffman conditions necessary for deadlock. Provide a real-life analogy for each condition.

3. Describe the different deadlock prevention techniques. How do these techniques help in avoiding deadlock?
4. What is a wait-for graph? Explain how it is used for deadlock detection with an example.
5. Compare and contrast the wait-die and wound-wait schemes for deadlock prevention. Which one is more efficient and why?

Reference

1. Transaction Processing: Concepts and Techniques - Jim Gray and Andreas Reuter
2. Database systems: The complete book - Hector Gracia - Mohna, Jeffery. D. Ullman and Jennifer Widom

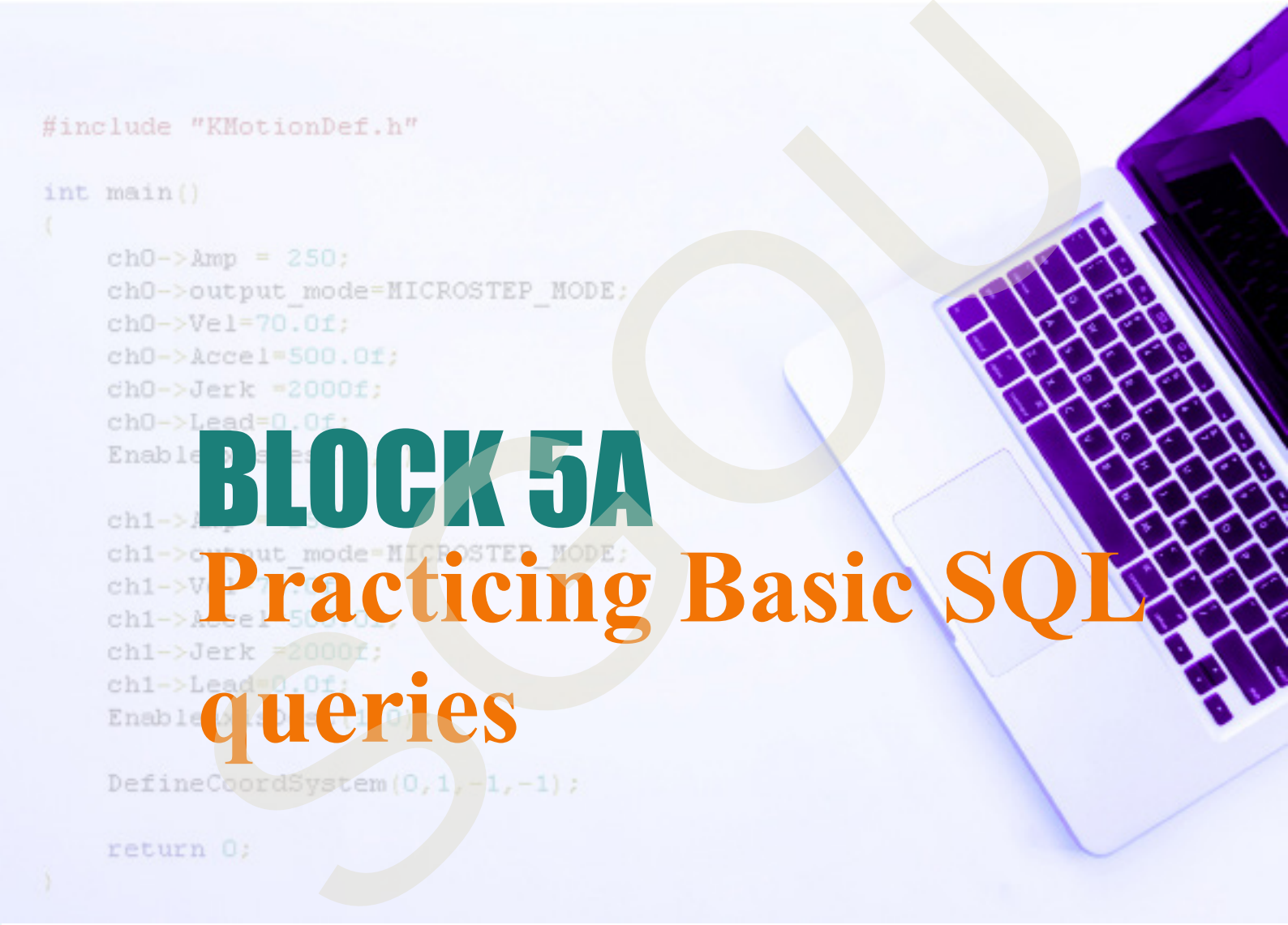
Suggested Reading

1. Elmasri and Navathe, Fundamentals of Database Systems, 5th Edition, Pearson.
2. Abraham Silbersehatz, Henry F. Korth and Sudarshan, Database System Concepts, 6th Edition, Tata McGraw-Hill.
3. Raghurama Krishnan, Johannes Gehrke, Database Management Systems, Tata McGrawHill.
4. C.J. Date, A.Kannan, S.Swami Nadhan, An Introduction to Database systems.

Lab Manual

Database Management System

B21CA05DC



BLOCK 5A

Practicing Basic SQL queries

Introduction

The **Database Management System (DBMS) Lab** is designed to provide students with hands-on experience in working with databases and understanding their critical role in software applications. This lab aims to bridge the gap between theoretical concepts of database systems and their practical applications. Students will learn to create, manipulate, and manage databases using Structured Query Language (SQL) and explore advanced database features.

Objectives of the DBMS Lab

- **Familiarisation with Database Concepts:** Gain an understanding of database design, relational database management systems (RDBMS), and SQL.
- **Database Design:** Learn how to design efficient databases using techniques such as Entity-Relationship (ER) modeling and normalization.
- **SQL Mastery:** Develop proficiency in SQL commands for data definition, manipulation, and querying.
- **Hands-on Experience:** Practice creating, updating, and retrieving data from databases.
- **Problem Solving:** Solve real-world problems by designing and implementing database solutions.
- **Advanced Features:** Explore advanced concepts such as views, stored procedures, triggers, and database security.

Tools and Software

- Students will use a variety of tools to enhance their learning experience, such as:
- **RDBMS Software:** MySQL, PostgreSQL, Oracle, SQLite, or Microsoft SQL Server.
- **Database Design Tools:** Tools like ERDPlus, Lucidchart, or any other ER diagram software.
- **Programming Interface:** Integration of SQL with programming languages like Python or Java (optional).

Scope of the Lab

This lab forms the foundation for understanding databases, an essential skill for careers in data science, software development, and information systems management. By completing this lab, students will:

1. Be equipped to design and manage databases for small to medium-scale applications.
2. Gain the knowledge required to optimize database performance and ensure data integrity.
3. Understand the importance of databases in building robust, real-world applications.

Expectations from Students

- Active Participation: Engage actively in lab sessions to grasp the concepts thoroughly.
- Preparation: Review the lab manual and experiment instructions before attending the lab.
- Consistency: Complete all lab assignments and submit them on time.
- Teamwork: Collaborate with peers for projects and group activities.

This lab serves as a stepping stone to mastering the art and science of managing and utilizing databases efficiently. Students are encouraged to ask questions, explore beyond the experiments, and integrate what they learn into practical applications.

Lab Guidelines

To ensure a productive and disciplined learning environment, the following guidelines must be adhered to during the DBMS lab sessions:

General Instructions

1. Attendance:

- Attendance is mandatory for all lab sessions. Missing a lab session without prior approval may result in reduced grades.

2. Punctuality:

- Arrive at the lab on time. Latecomers may not be allowed to participate in

the session.

3. Preparation:

Read the lab manual and review the objectives of the experiment before the session.

Install any required software and bring all necessary materials, including notebooks and storage devices.

4. Lab Etiquette:

- Maintain a quiet and professional environment in the lab.
- Keep mobile phones on silent or switched off unless required for academic purposes.
- Treat lab equipment and systems with care.

System Usage and Safety

5. Login Credentials:

- Use only the login credentials assigned to you. Do not share your credentials with others.

6. Data Management:

- Save your work frequently to avoid data loss.
- Use external storage or cloud services to back up your work after each session.
- Do not delete or alter files belonging to other users or the system.

7. Software Usage:

- Use only the software authorized by the instructor.
- Do not install or uninstall any software without prior permission.

During Lab Sessions

8. Following Instructions:

- Pay attention to the instructor's directions and follow the experiment steps as outlined in the manual.

9. Teamwork:

- Collaborate with peers when group work is assigned. Ensure that all members contribute equally.

10. Query Resolution:

- Raise your hand and ask questions if you face any difficulties during the experiment.

11. Time Management:

- Complete your tasks within the allotted time. Unfinished work must be submitted as per the instructor's guidelines.

Post-Lab Activities

12. Submission:

- Submit your lab assignments and reports as per the specified deadlines. Late submissions may result in penalties.

13. Plagiarism:

- Ensure that all work submitted is your own. Plagiarism or copying from others will not be tolerated and may lead to disciplinary action.

14. Assessment:

- Be prepared for viva or oral exams to demonstrate your understanding of the experiments.

Disciplinary Actions

15. Non-compliance with lab guidelines, misuse of lab equipment, or engaging in unethical practices will result in disciplinary measures, which may include:

- Warnings.
- Deduction of grades.
- Suspension from lab sessions.

Code of Conduct

- Be respectful to instructors, lab assistants, and peers.
- Always seek help when required, but do not disrupt others' work.

- Strive to develop problem-solving skills and explore advanced features of DBMS tools.

Steps to Install and Configure an RDBMS Tool

1. Installing and Setting Up MySQL

To work with MySQL, follow these steps to install and configure it properly.

Step 1: Download MySQL

1. Visit the official MySQL Community Edition download page.
2. Choose the appropriate version for your operating system (Windows, macOS, or Linux).
3. Download the MySQL Installer and run the setup file.

Step 2: Install MySQL

1. Launch the MySQL Installer and select the **Custom** or **Developer Default** setup type.
2. Ensure that **MySQL Server**, **MySQL Workbench**, and **MySQL Shell** are selected.
3. Proceed with the installation by following the on-screen instructions.

Step 3: Configure MySQL Server

1. Choose the **Standalone MySQL Server** setup.
2. Select the **server type** (typically "Development Computer" for personal use).
3. Set the **authentication method** (MySQL recommends **strong password encryption**).
4. Create a **root user password** and optionally add other users.
5. Configure the **default port (3306)** and network settings.

Step 4: Complete Installation

1. Click **Execute** to apply all configurations and start the MySQL service.
2. Once installation is complete, test the connection using **MySQL Workbench** or the **MySQL Command Line Client**.

Step 5: Launch MySQL and Start Using It

1. Using MySQL Workbench: Open MySQL Workbench, create a new connection, enter your credentials, and start executing SQL queries.
2. Using MySQL CLI: Open the command prompt (or terminal), type mysql -u root -p, enter your password, and run SQL commands directly.

EXPERIMENT 1

Implementation of DDL and basic DML Commands

Objectives

1. Create Tables using SQL with appropriate data types and constraints.
2. Modify Tables by altering structure and dropping columns.
3. Manage Data using INSERT, UPDATE, and DELETE commands.
4. Remove Tables using DROP TABLE and understand its impact.

Theory

1.1 CREATE Command

The CREATE command is used to define and create database objects such as databases, tables, views, and indexes.

1.1.1 CREATE DATABASE

The CREATE DATABASE statement creates a new database.

Syntax:

```
CREATE DATABASE database_name;
```

Example:

Creating a database named University:

```
CREATE DATABASE University;
```

1.1.2 CREATE TABLE

The CREATE TABLE statement defines a new table with specified columns and data types.

Syntax:

```
CREATE TABLE table_name (
```

```
column1 data_type constraints,
column2 data_type constraints,
...
columnN data_type constraints
);
```

Example:

Creating a table **Students** with constraints:

```
CREATE TABLE Students (
RollNo INT PRIMARY KEY,
Name VARCHAR(50) NOT NULL,
Email VARCHAR(100) UNIQUE,
Dept VARCHAR(30) DEFAULT 'CS',
Age INT CHECK (Age <= 18)
);
```

1.2 SQL Constraints

Constraints ensure data integrity and restrict invalid data entry. The given Table 1.1 shows different types of constraints.

Table 1.1 Types of Constraints

Constraint	Description
Primary Key	Ensures each row has a unique identifier
Foreign Key	Enforces relationships between tables
Not Null	Ensures a column cannot store NULL values
Unique	Ensures all values in a column are distinct
Check	Restricts column values to meet a condition
Default	Assigns a default value if no input is provided
Composite Key	Combines multiple columns as a primary key
Auto Increment	Generates unique values automatically (MySQL)

Example:

To create a table Customer in the Bank database with Name, AccountNumber, and Balance, we can write the query as follows.

```
CREATE TABLE Customer (
    Name VARCHAR(100) NOT NULL,
    AccountNumber INT PRIMARY KEY,
    Balance DECIMAL(15,2) NOT NULL
);
```

1.3 Data Types

In a Database Management System(DBMS), data types define the kind of values a column can hold. Understanding and correctly using data types is crucial for maintaining data integrity and optimizing database performance.

1. Numeric Data Types

It is used to store numeric values like integers and floating-point numbers.

Table 1.2 Numeric Data Types

Data Type	Description	Example
INT	Stores integers	10,-100,0
SMALLINT	Stores smaller integers	5,-50
BIGINT	Stores very large integers	9223372036,854775807
DECIMAL(p,s)	Fixed - point numbers with precision p and scale s	123.45,-0.99
NUMERIC(p,s)	Similar to DECIMAL, used for exact precision	12345.67
FLOAT	Stores floating - point numbers(approximation)	123.456,-98.765
DOUBLE	Double-precision floating-point numbers	1.797E+308

2. Character (String) Data Types

It is used to store text data.

Table 1.3 Character Data Types

Data Type	Description	Example
CHAR(n)	Fixed-length string with length n	'Hello'(length=5)
VARCHAR(n)	Variable-length string with maximum length n	'John', 'Database'
TEXT	Large variable-length string	'Lorem ipsum'

3. Date and Time Data Types

It is used to store date, time and timestamp values.

Table 1.4 Date and Time Data Types

Data Type	Description	Example
DATE	Stores date values in YYYY-MM-DD format	'2024-11-28'
TIME	Stores time values	'14:30:00'
DATETIME	Stores date and time values	'2024-11-28 14:30:00'
TIMESTAMP	Stores date and time UTC	'2024-11-28 14:30:00 UTC'
YEAR	Stores a year in YYYY format	1995

4. Boolean Data Type

It represent truth values

Table 1.5 Boolean Data Types

Data Type	Description	Example
BOOLEAN	Stores TRUE, FALSE or NULL	TRUE, FALSE

5. Binary Data Types

It is used to store binary data like images or files

Table 1.6 Binary Data Types

Data Type	Description	Example
BINARY(n)	Fixed length binary data of length n	0x1234ABCD
VARBINARY(n)	Variable length binary data with max n	0x12AB

Example :

Create a Student Database

```
CREATE TABLE Students (
RollNo INT PRIMARY KEY,-------Integer data type
Name VARCHAR(30) NOT NULL,----Variable-length text
Age INT CHECK (Age <= 18)-------Integer with a constraint
Email VARCHAR(100) UNIQUE,-----Unique text data
Enrollment DATE,----------Date for enrollment
Active BOOLEAN DEFAULT TRUE  ----------Boolean for active status
);
```

1.4 DATA MANIPULATION COMMANDS (DML)

1.4.1 INSERT Command

The INSERT command adds new records into a table.

Syntax:

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);    # Insert values into specific fields.
INSERT INTO table_name VALUES (value1, value2, value3, ...);# Insert values into all the fields.
```

Example:

a. Inserting values into specific fields

```
INSERT INTO Students (RollNo, Name, Dept)
VALUES (1, 'Rahul', 'CS'),
(2, 'Hari', 'IT');
```

b. Insert values into all the fields

```
INSERT INTO Students
VALUES (1, 'Rahul', 'CS', 18),
(2, 'Hari', 'IT', 17);
```

1.4.2 SELECT Command

The SELECT command retrieves data from a table.

Syntax:

```
SELECT column1, column2 FROM table_name;
```

Example:

```
SELECT cno, cname FROM Customers;
```

- To retrieve all student details:

Example:

```
SELECT * FROM Students;
```

1.4.3 UPDATE Command

The UPDATE command modifies existing records in a table.

- Update the values of specific field based on a given condition

Syntax:

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

Example:

Updating a student's department:

```
UPDATE Students
SET Dept = 'IT'
WHERE RollNo = 1;
```

- Update all the records

Syntax:

```
UPDATE column name= Values;
```

Example:

Update all students whose department name is 'IT'

```
UPDATE Students SET Dept="IT";
```

1.4.4 DELETE Command

The DELETE command removes specific rows from a table.

- Delete a particular record

Syntax:

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

Example:

```
DELETE FROM Students WHERE RollNo = 1;
```

- Delete all records

Syntax:

```
DELETE FROM Table_name;
```

Example:

```
DELETE FROM Students;
```

1.4.5 TRUNCATE Command

The TRUNCATE command removes all rows from a table but keeps the structure.

Syntax:

```
TRUNCATE TABLE table_name;
```

Example:

```
TRUNCATE TABLE Students;
```

1.5 TABLE MODIFICATION COMMANDS (DDL)

1.5.1 DROP TABLE Command

The DROP TABLE command permanently deletes a table and its data.

Syntax:

```
DROP TABLE table_name;
```

Example:

Dropping the Students table:

```
DROP TABLE Students;
```

♦ To avoid errors if the table does not exist, use:

DROP TABLE IF EXISTS Students;

1.5.2 ALTER TABLE Command–

The ALTER TABLE command modifies an existing table structure

Table 1.6 Alter Command

Operation	Syntax
Add a column	ALTER TABLE table_name ADD column_name data_type;
Drop a column	ALTER TABLE table_name DROP COLUMN column_name;
Modify a column	ALTER TABLE table_name MODIFY column_name new_data_type;
Rename a table	ALTER TABLE table_name RENAME TO new_table_name;

Examples:

1. Add a column to store the student's phone number

```
ALTER TABLE Students ADD Phone VARCHAR(15);
```

2. Drop the Age column

```
ALTER TABLE Students DROP COLUMN Age;
```

3. Modify the Dept column length

```
ALTER TABLE Students MODIFY Dept VARCHAR(50);
```

4. Rename the Students table to Learners

```
ALTER TABLE Students RENAME TO Learners;
```

Sample Experiment

Write the SQL queries for the following questions

1. Write an SQL command to create a database named SchoolDB.
2. Create a table Students with fields: StudentID (Primary Key), Name, Age, Class, and Email (Unique).
3. Insert the following values in the table.

Student ID	Name	Age	Class	Email
101	John	18	B.Sc CS	john@example.com
102	Alice	19	BA Eng.	alice@example.com
103	Robert	20	BCom.	robert@example.com
104	Emily	21	BSc CS	emily@example.com
105	David	22	BTech	david@example.com
106	Sophia	20	BBA	sophia@example.com
107	Michael	23	BSc Math	michael@example.com
108	Olivia	19	BCA	olivia@example.com
109	Daniel	21	BA Hist	daniel@example.com
110	Jessia	22	BSc CS	jessica@example.com

4. Modify the Employees table by adding a new column PhoneNumber (VARCHAR(15)).
5. Rename the table Departments to CompanyDepartments.
6. Drop the column Email from the Employees table.
7. Update the age of the student named David to 21 in the Students table.
8. Remove all Records from the Students table
9. Drop the entire Students table

Answers:

1. CREATE DATABASE SchoolDB;
2. CREATE TABLE Students (
   StudentID INT PRIMARY KEY,
   Name VARCHAR(50) NOT NULL,
   Age INT CHECK (Age >= 18),
   Class VARCHAR(20) NOT NULL,
   Email VARCHAR(100) UNIQUE NOT NULL
   );
3. INSERT INTO Students (StudentID, Name, Age, Class, Email) VALUES
   (101, ‘John’, 18, ‘B.Sc CS’, ‘john@example.com’),
   (102, ‘Alice’, 19, ‘B.A Eng’, ‘alice@example.com’),
   (103, ‘Robert’, 20, ‘B.Com’, ‘robert@example.com’),
   (104, ‘Emily’, 21, ‘B.Sc CS’, ‘emily@example.com’),

(105, ‘David’, 22, ‘B.Tech’, ‘david@example.com’),

(106, ‘Sophia’, 20, ‘BBA’, ‘sophia@example.com’),

(107, ‘Michael’, 23, ‘B.Sc Math’, ‘michael@example.com’),

(108, ‘Olivia’, 19, ‘BCA’, ‘olivia@example.com’),

(109, ‘Daniel’, 21, ‘B.A Hist’, ‘daniel@example.com’),

(110, ‘Jessica’, 22, ‘B.Sc CS’, ‘jessica@example.com’);

4. ALTER TABLE Students ADD PhoneNumber VARCHAR(15);
5. ALTER TABLE Students RENAME TO Learners;
6. ALTER TABLE Students DROP COLUMN Email;
7. UPDATE Students SET Age = 21 WHERE Name = ‘David’;
8. TRUNCATE TABLE Students;
9. DROP TABLE Students;

Lab Practice Experiment

1. Write an SQL command to create a database named UniversityDB.
2. Create a table Courses with fields: CourseID (Primary Key), CourseName (Unique), Credits, and Instructor.
3. Insert values into the Courses table.
4. Modify the Courses table by adding a new column Duration (INT).
5. Update the Instructor name for the course “Database Systems” to “Dr. Smith” in the Courses table.
6. Drop the column Credits from the Courses table.
7. Add a NOT NULL constraint to the Instructor column in the Courses table.
8. Retrieve all courses that have a Duration of more than 4 weeks.
9. Change the CourseName of “Computer Networks” to “Advanced Networking” in the Courses table.
10. Count the total number of courses available in the Courses table.
11. Remove all records from the Courses table
12. Drop the entire Courses Table

EXPERIMENT 2

Implementation of Basic Queries in SQL

Objectives

1. Apply WHERE clause, LIKE operator, and logical operators (AND, OR, NOT) to filter data.
2. Perform ORDER BY to sort query results
3. Use aliases, *, and DISTINCT to format and refine query results.

Theory

2.1 The SELECT Statement

The SELECT statement is the most commonly used SQL query. It allows you to specify which columns of data you want to retrieve from a database table. The result of a SELECT query is a set of rows that meet the criteria specified.

Syntax:

SELECT column1, column2, ...FROM table_name;

Example:

SELECT first_name, last_name FROM employees;

This query retrieves the first_name and last_name columns from the employees table. You can specify multiple columns separated by commas

Selecting All Columns: If you want to retrieve all columns from a table without specifying each column individually, use the * symbol:

SELECT * FROM employees;

This query fetches all columns in the employees table. While this is useful, it's often better to specify only the columns you need for better performance, especially in large tables.

2.2 The WHERE Clause

The WHERE clause is used to filter records based on specified conditions. This clause allows you to specify criteria that rows must meet to be included in the query results.

Syntax:

SELECT column1, column2, ...

FROM table_name

WHERE condition;

Example:

SELECT first_name, last_name FROM employees WHERE department_id = 10;

1. This query retrieves the first and last names of employees who work in department 10.
2. Using Comparison Operators: The WHERE clause allows the use of comparison operators to filter results based on certain conditions. These operators include =, <, >, <=, >=, !=, BETWEEN, and IN.

Equality (=):

SELECT * FROM employees WHERE department_id = 10;

Retrieves employees who belong to department 10.

Inequality (!=):

SELECT * FROM employees WHERE salary != 60000;

Retrieves employees whose salary is not 60,000.

Range (BETWEEN):

SELECT * FROM employees WHERE salary BETWEEN 50000 AND 70000;

This query retrieves employees whose salary is between 50,000 and 70,000.

Set membership (IN):

SELECT * FROM employees WHERE department_id IN (10, 20, 30);

This retrieves employees working in departments 10, 20, or 30.

2.3 Using DISTINCT

The DISTINCT keyword is used to eliminate duplicate rows from the result set. It ensures that the query returns only unique values for the selected columns.

Syntax:

SELECT DISTINCT column1, column2, ... FROM table_name;

Example:

SELECT DISTINCT department_id FROM employees;

This query retrieves all unique department IDs from the employees table, removing any duplicates.

When to Use DISTINCT: If a table contains duplicate values in the selected columns, DISTINCT ensures that only unique entries are returned. It is commonly used when dealing with large datasets where duplicates might skew results.

2.4 The ORDER BY Clause

The ORDER BY clause is used to sort the result set. It can be applied to one or more columns, and the order can be ascending (ASC) or descending (DESC). By default, the sorting order is ascending.

Syntax:

SELECT column1, column2, ...

FROM table_name

ORDER BY column1 [ASC|DESC];

Example:

SELECT first_name, last_name FROM employees ORDER BY last_name ASC;

This query retrieves the first_name and last_name of employees and sorts the result by last_name in ascending alphabetical order.

Descending Order: If you want to sort in descending order, you can use DESC:

SELECT first_name, last_name

FROM employees

ORDER BY salary DESC;

This query retrieves the first_name and last_name of employees, sorted by their salary in descending order.

Sorting by Multiple Columns: You can also sort by multiple columns. For instance:

SELECT first_name, last_name, salary FROM employees

ORDER BY department_id ASC, salary DESC;

This query first sorts the employees by department_id in ascending order, and then by salary in descending order within each department.

2.5 The LIKE Operator

The LIKE operator is used to search for a specified pattern within a column's values. It is often used with wildcard characters to perform flexible searches.

Wildcard Characters:

%: Represents zero or more characters.

_: Represents a single character.

Syntax:

SELECT column1, column2, ...

FROM table_name

WHERE column1 LIKE pattern;

Example:

SELECT first_name, last_name FROM employees

WHERE first_name LIKE 'J%';

This query retrieves the first_name and last_name of employees whose first name starts with "J".

Using _ for Single Character Matching:

SELECT first_name, last_name FROM employees

WHERE first_name LIKE '_a%';

This query retrieves employees whose first name has 'a' as the second letter (e.g., "Sam", "Janet").

Combining Wildcards: You can combine % and _ for more complex patterns. For example:

SELECT first_name, last_name FROM employees

WHERE first_name LIKE '_a%_';

This retrieves employees whose first name contains "a" as the second letter and has more than two characters.

2.6 AliWases in SQL

SQL allows the use of aliases, which are temporary names given to tables or columns for the duration of a query. Aliases are especially useful when working with complex queries or when you need to give a column a more readable name in the result set.

Column Aliases: You can use the AS keyword to assign a new name to a column. SELECT first_name AS “First Name”, last_name AS “Last Name” FROM employees;

This query assigns the column first_name the alias “First Name” and last_name the alias “Last Name”.

Table Aliases: You can also alias tables, especially useful in queries with joins.

```
SELECT e.first_name, e.last_name
FROM employees AS e
WHERE e.department_id = 10;
```

In this example, the employees table is aliased as e, making the query easier to read.

2.7 Logical Operators: AND, OR, and NOT

You can combine multiple conditions in the WHERE clause using logical operators like AND, OR, and NOT.

AND: Returns rows where all conditions are true.

```
SELECT * FROM employees
WHERE department_id = 10 AND salary > 50000;
```

This retrieves employees who work in department 10 and have a salary greater than 50,000.

OR: Returns rows where at least one condition is true.

```
SELECT * FROM employees
WHERE department_id = 10 OR department_id = 20;
```

This retrieves employees who belong to either department 10 or department 20.

NOT: Reverses the result of a condition.

```
SELECT * FROM employees
```

WHERE NOT department_id = 10;

This query retrieves all employees who do not belong to department 10.

Sample Experiment

Consider the following **employees** table:

Table 2.1 employees data

Employee_ id	first_ name	last_name	department_ id	salary
101	John	Doe	10	60000
102	Alice	Smith	20	75000
103	Bob	Johnson	10	55000
104	Charlie	Brown	30	70000
105	David	white	20	80000
106	Emma	Taylor	30	65000
107	Frank	Harris	10	50000
108	Grace	Wilson	20	72000
109	Henry	Moore	30	68000
110	Ivy	Thomas	10	62000

Write SQL queries for the following:

1. Retrieve all columns of data from the **employees** table.
2. Display the **first_name** and **last_name** of employees who belong to department 10.
3. Retrieve the **first_name** and **last_name** of employees whose salary is greater than 60,000.
4. Find employees whose **salary** is between 55,000 and 75,000.
5. Retrieve the unique **department_id** values from the table.
6. List all employees sorted by **last_name** in ascending order.
7. Display the **first_name** and **last_name** of employees, sorting them by **salary** in descending order.
8. Find employees who work in departments 10, 20, or 30 using the **IN** operator.
9. Retrieve employees whose **first_name** starts with 'J'.
10. Find employees whose **first_name** has 'a' as the second letter.
11. Assign an alias **"Employee Name"** to the concatenation of **first_name** and **last_name**.

12. Use a table alias ‘e’ for the **employees** table and retrieve all employee details.
13. Retrieve employees who work in department 10 **and** have a salary greater than 50,000.
14. Retrieve employees who work in department 10 **or** have a salary greater than 75,000.
15. Retrieve all employees who are **not** in department 10.

Answers

1. SELECT * FROM employees;
2. SELECT first_name, last_name FROM employees WHERE department_id = 10;
3. SELECT first_name, last_name FROM employees WHERE salary > 60000;
4. SELECT * FROM employees WHERE salary BETWEEN 55000 AND 75000;
5. SELECT DISTINCT department_id FROM employees;
6. SELECT * FROM employees ORDER BY last_name ASC;
7. SELECT first_name, last_name FROM employees ORDER BY salary DESC;
8. SELECT * FROM employees WHERE department_id IN (10, 20, 30);
9. SELECT first_name, last_name FROM employees WHERE first_name LIKE ‘J%’;
10. SELECT first_name, last_name FROM employees WHERE first_name LIKE ‘_a%’;
11. SELECT CONCAT(first_name, ‘ ‘, last_name) AS “Employee Name” FROM employees;
12. SELECT e.* FROM employees AS e;
13. SELECT * FROM employees WHERE department_id = 10 AND salary > 50000;
14. SELECT * FROM employees WHERE department_id = 10 OR salary > 75000;
15. SELECT * FROM employees WHERE NOT department_id = 10;

Lab Practice Experiment

Consider the following **Products** table 2.2

Table 2.2 Products

product_ id	product_ name	category	price	stock_ quantity	supplier_ id
201	Laptop	Electronics	800	50	1
202	Smartphone	Electronics	500	100	2
203	Headphones	Accessories	50	200	3
204	Monitor	Electronics	300	30	1
205	Keyboard	Accessories	40	150	3
206	Mouse	Accessories	25	180	3
207	Desk Chair	Furniture	120	20	4
208	Table Lamp	Furniture	60	25	4
209	USB Cable	Accessories	10	300	3
210	Printer	Electronics	250	40	2

Write SQL queries for the following:

1. Retrieve all columns from the **products** table.
2. Retrieve **product_name** and price of all products.
3. Find all products in the Electronics category.
4. Retrieve products whose price is not equal to 50.
5. Find products with a price between 50 and 300.
6. Retrieve products that belong to categories Electronics, Accessories, or Furniture.
7. Get unique categories from the **Products** table.
8. List products sorted by **price** in ascending order.
9. List products sorted by **category** (ascending) and price (descending).
10. Find products whose **product_name** starts with 'M'.
11. Find products whose **product_name** has 'o' as the second letter.
12. Retrieve **product_name** and **price**, renaming them as **"Product"** and "Cost".
13. Find products in Electronics category with a price greater than 300.
14. Find products supplied by supplier 2 or supplier 3.
15. Find products not in the Accessories category.

EXPERIMENT 3

Using Set Operations, Joins, Nesting of Queries

Objectives

1. To implement different types of set operations and joins
2. To execute nested queries (subqueries) for filtering and calculations
3. To compare the outputs of different Set Operations and JOINS

Theory

The SQL Joins clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.The join is actually performed by the 'where' clause which combines specified rows of tables.

3.1 Set Operations

Set operations in SQL enable combining the outputs of multiple SELECT queries. They provide functionalities like UNION, UNION ALL, INTERSECT, INTERSECT ALL and EXCEPT, which help in merging, intersecting, or subtracting data from different result sets.

3.1.1 UNION

The UNION operator is used to merge the results of two or more SELECT queries. It eliminates duplicate entries from the final output. As a result, if a record appears in both tables, it will be included only once in the combined result.

Syntax:

```
SELECT column1, column2, ...
FROM table1
UNION
SELECT column1, column2, ...
FROM table2;
```

Example:

Consider two tables Employee and Contractors

Input:

Table 3.1 Employee

ID	NAME	ROLE
1	James	Manager
2	Alice	Developer

Table 3.2 Contractors

ID	NAME	ROLE
3	Helen	Developer
4	Bob	Analyst

Output:

Table 3.3 UNION

ID	NAME	ROLE
1	James	Manager
2	Alice	Developer
3	Helen	Developer
4	Bob	Analyst

To find the union, you can write the query as:

```
SELECT ID, NAME
FROM Employee
UNION
SELECT ID, NAME, ROLE
FROM Contractors;
```

3.1.2 UNION ALL

The UNION ALL operator merges the results of two or more SELECT queries

without filtering out duplicates. Every record from both queries is included in the result set, even if there are repeated entries.

Syntax:

```
SELECT column1, column2, ...
FROM table1
UNION ALL
SELECT column1, column2, ...
FROM table2;
```

Example:

Consider two tables employee_2023 and employee_2024

Input:

Table 3.4 employee_2023

emp_id	emp_name
1	Alice
2	Bob

Table 3.5 employee_2024

emp_id	emp_name
2	Bob
3	James

Output:

Table 3.6 UNION ALL

emp_id	emp_name
1	Alice
2	Bob
2	Bob
3	James

When apply UNION all:

```
SELECT emp_id, emp_name
FROM employee_2023
UNION ALL
SELECT emp_id, emp_name
FROM employee_2024;
```

3.1.3 INTERSECT

The INTERSECT operator in SQL combines the results of two SELECT queries and returns only the rows that are present in both result sets. It extracts the shared records between the two queries, essentially identifying the intersection of the datasets. Similar to the UNION operator, INTERSECT eliminates duplicate rows from the final output.

Syntax:

```
SELECT column1, column2, ...
FROM table1
INTERSECT
SELECT column1, column2, ...
FROM table2;
```

Example: Consider two tables employee_2023 and employee_2024

Input:

Table 3.7 employee_2023

emp_id	emp_name
1	Alice
2	Bob
3	Charlie

Table 3.8 employee_2023

emp_id	emp_name
2	Bob
3	Charlie
4	David

Output:

Table 3.9 INTERSECT

emp_id	emp_name
2	Bob
3	Charlie

When applying intersect operation:

```
SELECT emp_id, emp_name
FROM employee_2023
INTERSECT
SELECT emp_id, emp_name
FROM employee_2024;
```

3.1.4 INTERSECT ALL

Returns all the rows that are common to the result of two or more queries including duplicates.

Syntax:

```
SELECT column1, column2, ...
FROM table1
INTERSECT ALL
SELECT column1, column2, ...
FROM table2;
```

Example:

Consider two tables employee_2023 and employee_2024

Table 3.10 employee_2023

emp_id	emp_name
1	Alice
2	Bob
3	Charlie

Table 3.11 employee_202e

emp_id	emp_name
2	Bob
3	Charlie
4	David

Output:

Table 3.12 INTERSECT ALL

emp_id	emp_name
2	Bob
3	Charlie

When applying intersect operation:

```
SELECT emp_id, emp_name
FROM employee_2023
INTERSECT  ALL
SELECT emp_id, emp_name
FROM employee_2024;
```

3.1.5 EXCEPT

The EXCEPT operator in SQL is used to retrieve the rows from the first SELECT query that do not exist in the second SELECT query. It essentially removes the rows from the first query that are found in the second query, leaving only the unique rows

from the first set.

Syntax:

```
SELECT column1, column2, ...
FROM table1
EXCEPT
SELECT column1, column2, ...
FROM table2;
```

Example: consider two tables, Products_A and Products_B, where some products are common in both tables.

Input:

Table 3.13 Product_1

Product ID	Product Name	Price
1	Laptop	40000
2	Mouse	1000
3	Keyboard	2000
4	Monitor	15000

Table 3.14 Product_2

Product ID	Product Name	Price
3	Keyboard	2000
4	Monitor	15000
5	Tablet	20000

Output 1: Table 3.15 product_1 Not in product_2

Product ID	Product Name	Price
1	Laptop	40000
2	Mouse	1000

When applying except:

```
SELECT Product Name, Price
FROM Product_1
EXCEPT
SELECT Product Name, Price
FROM Product_2;
```

3.2 JOINS

Joins allows retrieval of multiple data from multiple tables in a single query.

3.2.1 INNER JOIN:

The INNER JOIN operator in SQL combines rows from two or more tables that share a common column. It retrieves only the rows where there is a match in both tables. If no matching rows exist in either table, those rows are excluded from the result.

Syntax:

```
SELECT column1, column2, ...
FROM table1
INNER JOIN table2
ON table1.common_column = table2.common_column;
```

Example:

Input:

Table 3.17 Employees

EmpID	EmpName	DeptID
1	Roy	101
2	Helen	102
3	John	103

Table 3.18 Department

DeptID	DeptName
101	Finance
102	HR
104	IT

Output : Table 3.19 INNERJOIN

EmpName	DeptName
Roy	Finance
Helen	HR

```
SELECT Employees.EmpName, Department.DeptName
FROM Employees
INNER JOIN Department
ON Employees.DeptID = Departments.DeptID;
```

3.2.2 LEFT JOIN

A LEFT JOIN (also known as a LEFT OUTER JOIN) in SQL merges rows from two tables by returning all records from the first table (left table) and the corresponding matches from the second table (right table). If no matching records are found in the right table, the result will display NULL for the columns from the right table.

Syntax:

```
SELECT column1, column2, ...
FROM table1
LEFT JOIN table2
ON table1.common_column = table2.common_column;
```

Example:

Input:

Table 3.20 Employees

EmpID	EmpName	DeptID
1	Roy	101
2	Helen	102
3	John	103

Table 3.21 Department

DeptID	DeptName
101	Finance
102	HR
104	IT

Output : Table 3.22 LEFTJOIN

EmpName	DeptName
Roy	Finance
Helen	HR
John	NULL

Consider two tables, Employees and Department.

```
SELECT Employees.EmpName, Department.DeptName
FROM Employees
LEFT JOIN Department
ON Employees.DeptID = Department.DeptID;
```

3.2.3 RIGHT JOIN

A RIGHT JOIN (also called a RIGHT OUTER JOIN) in SQL merges rows from two tables by returning all records from the second table (right table) along with any

matching rows from the first table (left table). When no corresponding record exists in the left table, NULL values are shown for its columns in the result.

Syntax:

```
SELECT column1, column2, ...
FROM table1
RIGHT JOIN table2
ON table1.common_column = table2.common_column;
```

Example:

Input:

Table 3.23 Employees

EmpID	EmpName	DeptID
1	Roy	101
2	Helen	102
3	John	103

Table 3.24 Department

DeptID	DeptName
101	Finance
102	HR
104	IT

Output : Table 3.25 RIGHT JOIN

EmpName	DeptName
Roy	Finance
Helen	HR
NULL	IT

Consider two tables, Employees and Department. Right join can be written as :

```
SELECT Employees.EmpName, Department.DeptName
```

```
FROM Employees
RIGHT JOIN Department
ON Employees.DeptID = Department.DeptID;
```

3.2.4 FULL JOIN

A FULL JOIN (also known as a FULL OUTER JOIN) in SQL returns all rows from both tables, including all rows from the left table and the right table.. If a row from one table has no corresponding match in the other, NULL values are assigned to the columns of the unmatched table.

Syntax:

```
SELECT column1, column2, ...
FROM table1
FULL JOIN table2
ON table1.common_column = table2.common_column;
```

Example:

Input:

Table 3.26 Employees

EmpID	EmpName	DeptID
1	Roy	101
2	Helen	102
3	John	103

Table 3.27 Department

DeptID	DeptName
101	Finance
102	HR
104	IT

Output : Table 3.28 FULL JOIN

EmpName	DeptName

Roy	Finance
Helen	HR
John	NULL
NULL	Finance

```
SELECT Employees.EmpName, Department.DeptName
FROM Employees
FULL JOIN Department
ON Employees.DeptID = Department.DeptID;
```

3.2.5 CROSS JOIN

Returns the cartesian product of both tables, with each row of one table combined with each row of the other table.

Syntax:

```
SELECT column1, column2, ...
FROM table1
CROSS JOIN table2;
```

Example:

```
SELECT Employees.EmpName, Department.DeptName
FROM Employees
CROSS  JOIN Department;
```

Input:

Table 3.29 Employees

EmpID	EmpName	DeptID
1	Roy	101
2	Helen	102
3	John	103

Table 3.30 Department

DeptID	DeptName
101	Finance
102	HR
104	IT

Output : Table 3.31 CROSS JOIN

EmpName	DeptName
Roy	Finance
Roy	HR
Roy	IT
Helen	Finance
Helen	HR
Helen	IT
John	Finance
John	HR
John	IT

3.3 NESTED QUERIES

A nested query is an SQL query embedded within another query, with the inner query providing results that the outer query can utilize for filtering or calculations. It can yield one or multiple values and is commonly used in clauses such as WHERE, SELECT, or HAVING to handle complex data queries.

A single-row nested query is a subquery that produces a single result, which the main query utilizes for filtering or comparison. It commonly works with operators such as =, <, or > in the conditions.

A multi-row subquery is a subquery that retrieves several values, which the outer query is used for comparison or filtering. It is often paired with operators such as IN, ANY, EXISTS or ALL to handle multiple results from the inner query.

Syntax:

```
SELECT column1, column2, ...
FROM table1
WHERE column_name operator
```

(SELECT column1

FROM table2

WHERE condition);

Example:

Consider two tables:

Table 3.32 Student

StudentID	StudentName	Age	ClassID
1	Alice	15	101
2	Bob	14	102
3	Charlie	16	101
4	David	15	103

Table 3.33 Class

ClassID	ClassName
101	Math
102	Science
103	English

Output : Table 3.34 NESTED QUERY

StudentName
Alice
Charlie

Write a query to find the names of students who are enrolled in the "Math" class

```
SELECT StudentName
FROM Student
WHERE ClassID IN
(SELECT ClassID
    FROM Class
    WHERE ClassName = 'Math');
```

BLOCK 5B

Advanced SQL queries and PL/SQL

EXPERIMENT 4

Implementing advanced SQL Concepts

Objectives

1. To implement the EXISTS and NOT EXISTS operators to filter query results based on subqueries.
2. To apply aggregate functions to perform mathematical calculations.

Theory

This query is used to group by the records in a relation together for each and every value of a specific key(s) and then display them for a selected set of fields in the relation. The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

4.1 The EXISTS

The EXISTS operator is used to test for the existence of any record in a subquery. If a subquery finds any matching records, then the EXISTS operator returns TRUE. Otherwise, it returns FALSE.

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE EXISTS (subquery);
```

Example:

```
SELECT customer_id, customer_name
FROM Customers
WHERE EXISTS (
SELECT *  FROM Orders
WHERE Customers.customer_id = Orders.customer_id );
```

4.1.1 How to use EXISTS Operator

The use of exist operator can explained by the following examples ;

1. Retrieve Customers with Orders

Given two tables: **customers** (Attributes: **customer_id**, **customer_name**) and **orders** (Attributes: **order_id**, **customer_id**, **order_date**), write a SQL query to retrieve the names of customers who have placed at least one order.

```
SELECT customer_name
FROM customers c
WHERE EXISTS (
SELECT 1 FROM orders o
WHERE o.customer_id = c.customer_id
);
```

2. Find Employees with Assigned Projects

Given two tables: **employees** (Attributes: **employee_id**, **employee_name**, **department**) and **projects** (Attributes: **project_id**, **employee_id**, **project_name**), write a query to list employees who are assigned to any project.

```
SELECT employee_name, department
FROM employees e
WHERE EXISTS (
SELECT 1 FROM projects p
WHERE p.employee_id = e.employee_id
);
```

3. Identify Products with Sales Records

Given two tables: **products** (Attributes: **product_id**, **product_name**, **price**) and **sales** (Attributes: **sale_id**, **product_id**, **sale_date**), write a SQL query to find the names of products that have been sold at least once.

```
SELECT product_name
FROM products p
WHERE EXISTS (
SELECT 1 FROM sales s
WHERE s.product_id = p.product_id
```

);

4.1.2 NOT EXISTS Operator

The NOT EXISTS operator is the opposite of EXISTS. It returns TRUE only if the subquery returns no rows. It is used to filter out records that have related entries in another table.

Syntax:

```
SELECT column1, column2, ...
FROM table_name
WHERE NOT EXISTS (subquery);
```

Example:

```
SELECT first_name, last_name
FROM employees e
WHERE NOT EXISTS (
SELECT 1 FROM departments d
WHERE d.department_id = e.department_id
);
```

4.1.2.1 How to Use NOT EXISTS

The use of NOT EXISTS can be explained with the following examples:

1. Find Customers Without Orders

Given two tables: **customers** with attributes **customer_id** and **customer_name**, and **orders** with attributes **order_id** and **customer_id**, write a SQL query to find customers who have never placed an order.

```
SELECT customer_id, customer_name
FROM customers c
WHERE NOT EXISTS (
SELECT 1 FROM orders o WHERE o.customer_id = c.customer_id
);
```

2. List Departments Without Employees

Given two tables: **departments** with columns **department_id** and **department_name**, and **employees** with columns **employee_id** and **department_id**, write a query to list departments that have no employees.

```
SELECT department_id, department_name
FROM departments d
WHERE NOT EXISTS (
SELECT 1 FROM employees e WHERE e.department_id = d.department_id
);
```

3. Identify Products Not Ordered

Given two tables: **products** with columns **product_id** and **product_name**, and **order_details** with columns **order_id** and **product_id**, write a SQL query to find products that have never been ordered.

```
SELECT product_id, product_name
FROM products p
WHERE NOT EXISTS (
SELECT 1  FROM order_details od WHERE od.product_id = p.product_id
);
```

4.2 Aggregate Functions

Aggregate functions in SQL perform calculations on a set of values and return a single summarized result. These aggregate functions simplify summarizing and analyzing data, particularly when working with large datasets. Some aggregate functions are:

4.2.1 COUNT( )

- **COUNT(COLUMN NAME)**

Count the number of rows that match the specified column. Ignore rows where the specified column is NULL.

Syntax:

```
SELECT COUNT(column_name) FROM table_name WHERE condition;
```

Example:

```
SELECT COUNT(ENAME) FROM employees;
```

- **COUNT(*)**

Count all the rows in the table or result set, regardless of whether the rows contain NULL values or not.

Syntax:

```
SELECT COUNT(*) FROM table_name WHERE condition;
```

Example:

SELECT COUNT(*) FROM employees;

4.2.2 SUM( )

Calculates the total sum of a numeric column. It ignores NULL values.

Syntax:

SELECT SUM(column_name) FROM table_name WHERE condition;

Example:

SELECT SUM(salary) FROM employees WHERE department = 'Sales';

This query returns the total salary paid to employees in the Sales department.

4.2.3 AVG( )

Computes the average value of a numeric column, ignoring NULL values.

Syntax:

SELECT AVG(column_name) FROM table_name WHERE condition;

Example:

SELECT AVG(age) FROM customers;

This query returns the average age of all customers.

4.2.4 MIN( )

Returns the smallest value in a specified column.

Syntax:

SELECT MIN(column_name) FROM table_name WHERE condition;

Example:

SELECT MIN(salary) FROM employees;

This query returns the lowest salary in the employees table.

4.2.5 MAX( )

Returns the largest value in a specified column.

Syntax:

SELECT MAX(column_name) FROM table_name WHERE condition;

Example:

SELECT MAX(order_amount) FROM orders;

This query returns the highest order amount in the orders table.

4.2.6 COUNT DISTINCT( )

Returns the number of distinct (unique) values in a column.

Syntax:

SELECT COUNT(DISTINCT column_name) FROM table_name WHERE condition;

Example:

SELECT COUNT(DISTINCT department) FROM employees;

This query returns the number of unique departments in the employees table.

Sample Experiment question

Consider the two tables **Employees** and **Projects** and write the SQL queries for the following questions.

Table 4.1 Employee table

employee_ id	employee_ name	department	salary	age
101	Alice	HR	50000	30
102	Bob	Sales	60000	35
103	Charlie	IT	70000	40
104	David	HR	55000	28
105	Eva	Sales	62000	32
106	Frank	IT	73000	38
107	Grace	Marketing	48000	29
108	Henry	NULL	NULL	NULL

Table 4.2 Projects table

project_id	employee_id	project_name
201	102	Market Expansion
202	103	Software Upgrade
203	106	IT Security
204	107	Ad Campaign

Questions

1. Retrieve the names of employees who are assigned to at least one project.
2. Find the employees working in IT or Sales departments who have at least one project assigned to them.

3. Retrieve the names of employees who are not assigned to any project.
4. Find the total number of employees in the company.
5. Calculate the total salary paid to employees in the HR department.
6. Determine the average age of employees in the company.
7. Identify the lowest salary among all employees.
8. Find the highest salary among employees in the IT department.
9. Count the number of distinct departments in the company.

Answers

1. SELECT employee_name
   FROM employees e
   WHERE EXISTS (
   SELECT 1 FROM projects p
   WHERE p.employee_id = e.employee_id
   );
2. SELECT employee_name
   FROM employees e
   WHERE (e.department = 'IT' OR e.department = 'Sales')
   AND EXISTS (
   SELECT 1 FROM projects p
   WHERE p.employee_id = e.employee_id
3. SELECT employee_name
   FROM employees e
   WHERE NOT EXISTS (
   SELECT 1 FROM projects p
   WHERE p.employee_id = e.employee_id
   );
4. SELECT COUNT(*) FROM employees;
6. SELECT SUM(salary) FROM employees WHERE department = 'HR';
7. SELECT AVG(age) FROM employees;

8. SELECT MIN(salary) FROM employees;

9. SELECT MAX(salary) FROM employees WHERE department = 'IT';

10. SELECT COUNT(DISTINCT department) FROM employees;

Lab Practice Experiment

Consider the two tables **Books** and **Sales** and write the SQL queries for the following questions.

Table 4.3 Books table

Book_ id	title	author	genre	price	copies_ sold
301	The Silent Patient	Alex Michaelides	Thriller	15	500
302	Atomic Habits	James Clear	Self- Help	20	1200
303	Sapiens	Yuval Harai	History	18	800
304	The Alchemist	Paulo Coelho	Fiction	12	1000
305	Thinking, Fast and Slow	Daniel Kahneman	Psychology	22	600
306	Deep work	Cal Newport	Self- Help	17	400
307	Clean Code	Robert Martin	Technology	30	350
308	The Pragmatic Programmer	Andrew Hunt	Technology	28	200

Table 4.4 Sales table

sales_id	book_id	sale_date	quantity
401	302	2024-02-10	15
402	303	2024-02-12	10
403	305	2024-02-15	8
404	307	2024-02-18	6

1. Retrieve the titles of books that have been sold at least once.
2. Find the books from the Self-Help genre that have been sold.
3. Retrieve the titles of books that have never been sold.
4. Find books that do not belong to any genre (i.e., genre is NULL).
5. Find the total number of books available in the database.
6. Calculate the total copies sold for all books in the Fiction genre.
7. Determine the average price of all books.
8. Identify the lowest price among all books.
9. Find the highest number of copies sold among all books.
10. Count the number of distinct genres in the books table.

EXPERIMENT 5

Using GROUP BY, HAVING, and SUBSTRING Functions

Objectives

1. To implement the GROUP BY clause to organize query results into meaningful groups.
2. To use the HAVING clause to filter grouped data based on aggregate conditions.
3. To extract and manipulate string data using the SUBSTRING function in SQL.

Theory

5.1 GROUP BY Clause

The **GROUP BY** clause in SQL is used to **group rows** that have the same values in a specified column into summary rows. This is particularly useful when working with **large datasets** where data needs to be categorized and summarized using aggregate functions.

For example, in a sales database, we may want to calculate **total sales per product category**, or in an HR database, we may need to determine **the number of employees in each department**.

Syntax:

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
GROUP BY column_name;
```

Key Points

- The GROUP BY clause must be placed after the FROM clause and before the ORDER BY clause (if present).
- Every column in the SELECT statement that is not inside an aggregate

function must also appear in the GROUP BY clause.

- Common aggregate functions used with GROUP BY:
  - SUM(column_name): Returns the total sum of values for each group.
  - COUNT(column_name): Counts the number of records in each group.
  - AVG(column_name): Computes the average value for each group.
  - MIN(column_name): Finds the smallest value in each group.
  - MAX(column_name): Finds the largest value in each group.

Example:

```
SELECT category, SUM(price) AS total_sales
FROM products
GROUP BY category;
```

Explanation:

- The query groups records by the category column in the products table.
- The SUM(price) function calculates the total sales amount for each category.
- The result shows the total sales per product category.

5.2 HAVING Clause

The HAVING clause in SQL is used to filter grouped records based on a condition applied to an aggregate function. It is similar to the WHERE clause, but the key difference is:

- WHERE filters records before grouping, while
- HAVING filters groups after aggregation has been applied.

This is particularly useful when we need to find groups that meet specific conditions, such as:

- Finding departments where the average salary exceeds $50,000
- Identifying customers who have made more than three purchases

Syntax:

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
GROUP BY column_name
HAVING condition;
```

Key Points

- The HAVING clause is always used after GROUP BY.
- It is used only with aggregate functions such as SUM(), COUNT(), AVG(), etc.
- It allows filtering of grouped results based on conditions applied to aggregated values.

Example:

```
SELECT department, AVG(salary) AS average_salary
FROM employees
GROUP BY department
HAVING AVG(salary) > 50000;
```

Explanation:

- The query groups employees by department and calculates the average salary for each department.
- The HAVING clause ensures that only departments where the average salary exceeds $50,000 are included in the result.

5.3 SUBSTRING Function

The SUBSTRING function in SQL is used to extract a specific part of a string from a given column. It is useful in scenarios where text-based data needs to be processed or categorized, such as:

- Extracting area codes from phone numbers
- Extracting email domains from email addresses
- Extracting month or year from a formatted date

Syntax:

```
SELECT SUBSTRING(column_name, start_position, length)
FROM table_name;
```

Key Points

- start_position is 1-based, meaning the first character is at position 1.
- length specifies the number of characters to extract from the starting position.
- It is commonly used for text processing and formatting output.

Example:

SELECT full_name, SUBSTRING(full_name, 1, 3) AS short_name

FROM employees;

Explanation:

- The query extracts the first three characters from the full_name column.
- The result creates an abbreviation for each employee's name.

Sample Experiment Question

Consider the following tables

Table 5.1 Sales table

sale_id	product	category	amount	region
101	Laptop	Electronics	1200	East
102	Phone	Electronics	800	West
103	Shirt	Clothing	50	East
104	Laptop	Electronics	1300	North
105	Jeans	Clothing	60	South
106	Phone	Electronics	850	East
107	Shirt	Clothing	40	West
108	Washing Machine	Appliances	500	North

Table 5.2 Employee table

employee_id	full_name	department	salary
201	Alice Johnson	HR	50000
202	Bob Smith	Sales	60000
203	Charlie Brown	IT	70000
204	David Williams	HR	55000
205	Eva Adams	Sales	62000
206	Frank Miller	IT	73000
207	Grace Turner	Marketing	48000
208	Henry Walker	IT	75000

Write queries for the following questions

1. Retrieve the total sales amount for each product category.
2. Find the number of sales transactions made in each region.
3. Determine the average salary of employees in each department.
4. Find product categories where the total sales amount is greater than 2000.

5. Retrieve regions where more than two sales transactions have occurred.
6. Identify departments where the average salary is greater than 60000.
7. Find customers who have placed more than three orders and whose total order value exceeds 5000.
8. Extract the first four characters from each employee's full name.

Answers

1. SELECT category, SUM(amount) AS total_sales

   FROM sales

   GROUP BY category;

2. SELECT region, COUNT(sale_id) AS transaction_count

   FROM sales

   GROUP BY region;

3. SELECT department, AVG(salary) AS average_salary

   FROM employees

   GROUP BY department;

4. SELECT category, SUM(amount) AS total_sales

   FROM sales

   GROUP BY category

   HAVING SUM(amount) > 2000;

5. SELECT region, COUNT(sale_id) AS transaction_count

   FROM sales

   GROUP BY region

   HAVING COUNT(sale_id) > 2;

6. SELECT department, AVG(salary) AS average_salary

   FROM employees

   GROUP BY department

   HAVING AVG(salary) > 60000;

7. SELECT customer_id, COUNT(order_id) AS order_count, SUM(order_total) AS total_value

   FROM orders

GROUP BY customer_id

HAVING COUNT(order_id) > 3 AND SUM(order_total) > 5000;

8. SELECT full_name, SUBSTRING(full_name, 1, 4) AS short_name

FROM employees;

Lab Practice Experiment

Consider the following Table

Table 5.3 Employee table

employee_ id	full_name	department	salary	hire_date
201	Alice Johnson	HR	50000	2019-05-12
202	Bob Smith	Sales	60000	2018-07-22
203	Charlie Brown	IT	70000	2020-11-03
204	David Williams	HR	55000	2017-09-15
205	Eva Adams	Sales	62000	2021-01-08
206	Frank Miller	IT	73000	2016-12-05
207	Grace Turner	Marketing	48000	2022-06-18
208	Henry Walker	IT	75000	2015-04-30

Write queries for the following questions

1. Retrieve the total salary paid for each department.
2. Find the number of employees in each department.
3. Determine the highest salary in each department.
4. Identify departments where the average salary is greater than 60000.
5. Find departments that have more than two employees.
6. Extract the first three characters from each employee's full name.
7. Retrieve the last four characters of each employee's full name.
8. Find the employees who were hired before the year 2018.
9. Extract the year from the hire_date column for each employee.
10. Retrieve the employees whose department name starts with the letter 'S'.

EXPERIMENT 6

SQL Concepts: Views, Triggers, Procedures, Functions, and Cursors

Objectives

1. To create and manage virtual tables to simplify complex queries using views
2. To automate and implement database operations by using triggers
3. To write reusable SQL routines using procedures
4. To create and implement SQL functions
5. To handle row-by-row data processing using cursors

Theory

6.1 Views

A view is a virtual table that is created based on the result of an SQL query. It does not store data itself but provides a way to simplify complex queries by encapsulating them in a single entity. Views can be used to improve security by restricting direct access to tables while allowing controlled data retrieval.

Syntax:

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

Example :

```
CREATE VIEW EmployeeView AS
SELECT EmpID, EmpName, Department
FROM Employees
```

```
WHERE Department = 'HR';
```

- This creates a virtual table (view) called EmployeeView that contains a filtered selection of data from the Employees table.
- It includes only EmpID, EmpName, and Department columns.
- The WHERE clause ensures that only employees from the HR department are included.
- The view does not store data separately; it dynamically reflects changes made to the Employees table.

Now, we can query this view just like a table:

```
SELECT * FROM EmployeeView;
```

6.2 SQL Triggers

A trigger is a stored program that executes automatically in response to certain events, such as INSERT, UPDATE, or DELETE operations on a table. Triggers are useful for enforcing business rules, maintaining audit logs, and automating repetitive tasks.

Syntax:

```
CREATE TRIGGER trigger_name
AFTER INSERT INSTEAD OF{EVENT} ON table_name
FOR EACH ROW
BEGIN
-- Trigger actions
END;
```

Example:

```
CREATE TRIGGER EmployeeAudit
AFTER UPDATE ON Employees
FOR EACH ROW
BEGIN
INSERT INTO AuditLog (EmpID, ChangeDate)
VALUES (NEW.EmpID, NOW());
END;
```

- This creates a trigger named EmployeeAudit that executes after an UPDATE

is performed on the Employees table.

- FOR EACH ROW ensures that the trigger runs for every updated row.
- Whenever an employee record is updated, the trigger automatically inserts the updated employee's ID and the current date (NOW()) into the AuditLog table.
- This is useful for maintaining an audit trail of changes made to employee data.

6.3 SQL Procedures

A procedure is a stored SQL program that executes a set of statements when called. Procedures allow for code reuse and optimization, reducing redundancy in SQL queries. Unlike functions, procedures do not return values but can perform operations like inserting, updating, or deleting records.

Syntax:

```
CREATE PROCEDURE procedure_name (parameters)
AS
BEGIN
 -- SQL statements
END;
```

Example:

```
CREATE PROCEDURE GetEmployeeDetails (IN emp_id INT)
BEGIN
SELECT * FROM Employees WHERE EmpID = emp_id;
END;
```

- This creates a stored procedure called GetEmployeeDetails, which retrieves details of a specific employee when called.
- The IN parameter emp_id allows us to pass an employee ID when executing the procedure.
- When the procedure is called, it retrieves all columns for the employee with the given ID.

To execute this procedure for employee ID **101**, use:

```
CALL GetEmployeeDetails(101);
```

6.4 SQL Functions

A function in SQL is a reusable routine that returns a single value. It is a reusable block of code that performs a specific task and returns a value. Unlike procedures, functions can be used within SELECT statements. Functions help in calculations, data manipulation, and transforming query results.

Syntax:

```
CREATE FUNCTION function_name (parameters)
RETURNS datatype
BEGIN
-- Function logic
END;
```

Example:

```
CREATE FUNCTION CalculateBonus(salary DECIMAL(10, 2))
RETURNS DECIMAL(10, 2)
BEGIN
RETURN salary * 0.10;
END;
```

- This creates a function named CalculateBonus that calculates a 10% bonus on a given salary.
- It accepts a salary amount as input and returns the calculated bonus.
- Unlike a procedure, a function must return a value and can be used in SELECT statements.

To calculate a bonus for an employee with a salary of $50,000, use:

```
SELECT CalculateBonus(50000);
```

This will return 5000.00, which is 10% of 50,000.

6.5 SQL Cursors

A cursor is a database object used to process rows one at a time. Cursors are helpful in iterative operations when data needs to be manipulated row by row within a stored procedure.

Syntax:

```
DECLARE cursor_name CURSOR FOR query;
```

```
OPEN cursor_name;

FETCH cursor_name INTO variable;

CLOSE cursor_name;
```

Example:

```
CREATE PROCEDURE ProcessEmployees()

BEGIN

DECLARE emp_name VARCHAR(100);

DECLARE emp_cursor CURSOR FOR SELECT EmpName FROM Employees;

OPEN emp_cursor;

FETCH emp_cursor INTO emp_name;

WHILE emp_name IS NOT NULL DO

        -- Perform action

        FETCH emp_cursor INTO emp_name;

END WHILE;

CLOSE emp_cursor;

END;
```

- This creates a stored procedure named ProcessEmployees that processes employee names one by one.
- DECLARE emp_cursor CURSOR FOR SELECT EmpName FROM Employees;
  - Declares a cursor that retrieves employee names from the Employees table.
- OPEN emp_cursor;
  - Open the cursor to start retrieving rows.
- FETCH emp_cursor INTO emp_name;
  - Retrieves the first employee name into the variable emp_name.
- WHILE emp_name IS NOT NULL DO ...
  - Iterates through each row in the cursor and performs an action (not specified in the query).
- CLOSE emp_cursor;

- Closes the cursor after processing all employees.

Sample Experiment Questions

Consider the following table and write the queries for the following questions

Table 6.1 Products table

Product ID	Product Name	Category	Price	Stock Quantity	Added Date
101	Laptop	Electronics	75000	10	2022-05-10
102	Smart Phone	Electronics	50000	25	2021-11-22
103	Chair	Furniture	3000	50	2023-01-15
104	Table	Furniture	7000	20	2022-07-18
105	WashingMachine	Appliances	45000	8	2021-09-30
106	Refrigerator	Appliances	60000	5	2020-12-05
107	Head Phones	Electronics	2500	40	2023-04-20
108	Microwave Appliances	Appliances	20000	12	2022-06-25

Questions

1. Views

1. Create a view ExpensiveProducts that displays product names and prices for products costing more than 40,000.
2. Create a view CategoryStockSummary that shows each category and the total stock quantity available.
3. Query the ExpensiveProducts view to display products with a price greater than 60,000
4. Update the price of a product and check the view.
5. Drop the ExpensiveProducts view.

Answers

1. CREATE VIEW ExpensiveProducts AS

   SELECT ProductName, Price

   FROM Products

   WHERE Price > 40000;

2. CREATE VIEW CategoryStockSummary AS

   SELECT Category, SUM(StockQuantity) AS TotalStock

   FROM Products

   GROUP BY Category;

.

3. SELECT * FROM ExpensiveProducts WHERE Price > 60000;
4. UPDATE Products SET Price = 80000 WHERE ProductID = 106;

   SELECT * FROM ExpensiveProducts;
5. DROP VIEW ExpensiveProducts;

2. Triggers

1. Create a trigger LogPriceUpdate to record price changes.
2. Create a trigger PreventLowStockDeletion.
3. Insert a new product and verify the trigger.
4. Update the price of a product and check PriceAudit.
5. Attempt to delete a low-stock product.

1. CREATE TABLE PriceAudit (

   AuditID INT PRIMARY KEY AUTO_INCREMENT,

   ProductID INT,

   OldPrice DECIMAL(10, 2),

   NewPrice DECIMAL(10, 2),

   ChangeDate TIMESTAMP DEFAULT CURRENT_TIMESTAMP

   );

   CREATE TRIGGER LogPriceUpdate

   BEFORE UPDATE ON Products

   FOR EACH ROW

   BEGIN

   INSERT INTO PriceAudit (ProductID, OldPrice, NewPrice)

   VALUES (OLD.ProductID, OLD.Price, NEW.Price);

   END;
2. CREATE TRIGGER PreventLowStockDeletion

   BEFORE DELETE ON Products

   FOR EACH ROW

   BEGIN

IF OLD.StockQuantity < 10 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE_TEXT = 'Cannot delete products with stock less than 10.';

END IF;

END;

3. INSERT INTO Products (ProductID, ProductName, Category, Price, StockQuantity, AddedDate)

   VALUES (109, 'Smartwatch', 'Electronics', 30000, 15, '2024-03-20');

4. UPDATE Products SET Price = 55000 WHERE ProductID = 102;

   SELECT * FROM PriceAudit;

5. DELETE FROM Products WHERE ProductID = 106;

3. Procedures

Questions

1. Procedure to get all products in a category.
2. Procedure to increase product prices by 5% in a category.
3. Procedure to insert a new product
4. Execute Increase Price for Electronics.
5. Call Get Category Products for Furniture

Answers

1. CREATE PROCEDURE GetCategoryProducts(IN category_name VARCHAR(50))

   BEGIN

   SELECT * FROM Products WHERE Category = category_name;

   END;

2. CREATE PROCEDURE IncreasePrice(IN category_name VARCHAR(50))

   BEGIN

   UPDATE Products SET Price = Price * 1.05 WHERE Category = category_name;

   END;

3. CREATE PROCEDURE InsertProduct(IN p_name VARCHAR(50), IN p_

category VARCHAR(50), IN p_price DECIMAL(10,2), IN p_stock INT, IN p_date DATE)

BEGIN

INSERT INTO Products (ProductName, Category, Price, StockQuantity, AddedDate)

VALUES (p_name, p_category, p_price, p_stock, p_date);

END;

4. CALL IncreasePrice('Electronics');
5. CALL GetCategoryProducts('Furniture');

4. Functions

Questions

1. Function to calculate a 10% discount on a product price.
2. Function to return total stock count.
3. Function to get the highest product price.
4. Use the CalculateDiscount function.
5. Call GetTotalStock function.

Answers

1. CREATE FUNCTION CalculateDiscount(price DECIMAL(10,2))

   RETURNS DECIMAL(10,2)

   BEGIN

   RETURN price * 0.90;

   END;
2. CREATE FUNCTION GetTotalStock()

   RETURNS INT

   BEGIN

   DECLARE total_stock INT;

   SELECT SUM(StockQuantity) INTO total_stock FROM Products;

   RETURN total_stock;

   END;
3. CREATE FUNCTION MaxPrice()

RETURNS DECIMAL(10,2)

BEGIN

DECLARE max_price DECIMAL(10,2);

SELECT MAX(Price) INTO max_price FROM Products;

RETURN max_price;

END;

4. SELECT ProductName, CalculateDiscount(Price) AS DiscountedPrice FROM Products WHERE Price > 30000;
5. SELECT GetTotalStock() AS TotalStock;

5. Cursors

Questions

1. Cursor to retrieve product names.
2. Modify cursor to display only Electronics.
3. Cursor tp update stock for low-stock products
4. Use a cursor to find products above average price
5. Close and deallocate cursor.

Answers

1. CREATE PROCEDURE ProductCursor()

BEGIN

DECLARE p_name VARCHAR(50);

DECLARE prod_cursor CURSOR FOR SELECT ProductName FROM Products;

OPEN prod_cursor;

FETCH prod_cursor INTO p_name;

WHILE p_name IS NOT NULL DO

SELECT p_name;

FETCH prod_cursor INTO p_name;

END WHILE;

CLOSE prod_cursor;

END;

2. DECLARE prod_cursor CURSOR FOR SELECT ProductName FROM Products WHERE Category = ‘Electronics’;

3. CREATE PROCEDURE UpdateStockCursor()

   BEGIN

   DECLARE prod_id INT;

   DECLARE stock_cursor CURSOR FOR SELECT ProductID FROM Products WHERE StockQuantity < 10;

   OPEN stock_cursor;

   FETCH stock_cursor INTO prod_id;

   WHILE prod_id IS NOT NULL DO

   UPDATE Products SET StockQuantity = StockQuantity + 5 WHERE ProductID = prod_id;

   FETCH stock_cursor INTO prod_id;

   END WHILE;

   CLOSE stock_cursor;

   END;

4. DECLARE avg_price DECIMAL(10,2);

   SELECT AVG(Price) INTO avg_price FROM Products;

   DECLARE high_price_cursor CURSOR FOR SELECT ProductName FROM Products WHERE Price > avg_price;

5. CLOSE prod_cursor;

   DEALLOCATE prod_cursor;

Lab Practice Experiment

Consider the following table and write queries for the following questions
Table 6.2 Employee table

Employee_ id	full_name	department	salary	hire_date
101	Alice Johnson	HR	60000	2018-05-15
102	Bob Smith	Sales	50000	2017-08-22
103	Charlie Brown	IT	75000	2019-11-12
104	David Williams	Finance	40000	2020-06-10
105	Eva Adams	HR	72000	2016-03-05
106	Frank Miller	Sales	55000	2019-07-20
107	Grace Turner	IT	80000	2015-01-30
108	Henry Walker	Marketing	48000	2021-09-25

Questions

1. Views

1. Create a view HighSalaryEmployees that displays employee names and salaries for employees earning more than 60,000.
2. Create a view DepartmentSummary that shows each department's name and the number of employees in that department.
3. Query the HighSalaryEmployees view to display employees who earn more than 70,000.
4. Update an employee's salary in the Employees table and check whether the change is reflected in the HighSalaryEmployees view.
5. Drop the HighSalaryEmployees view from the database.

2. Triggers

6. Create a trigger LogSalaryUpdate that records salary changes in a table called SalaryAudit whenever an employee's salary is updated.
7. Create a trigger PreventEmployeeDeletion that prevents deleting employees whose salary is greater than 50,000 and displays an error message.
8. Insert a new employee record into the Employees table and check whether the trigger LogSalaryUpdate works correctly.
9. Update an employee's salary in the Employees table and verify if the changes are recorded in the SalaryAudit table.
10. Attempt to delete an employee earning more than 50,000 and verify whether the PreventEmployeeDeletion trigger prevents it.

3. Procedures

11. Write a stored procedure GetDepartmentEmployees that retrieves all employees belonging to a specified department.
12. Create a stored procedure IncreaseSalary that increases the salary of employees in a specific department by 10%.

13. Write a procedure InsertEmployee to insert a new employee record into the Employees table.
14. Execute the IncreaseSalary procedure for the HR department and verify the changes in the Employees table.
15. Call the GetDepartmentEmployees procedure to retrieve all employees in the IT department.

4. Functions

16. Write a function CalculateTax that takes an employee's salary and returns the tax amount (10% of salary).
17. Create a function GetEmployeeCount that returns the total number of employees in the company.
18. Write a function MaxSalary that returns the highest salary among all employees.
19. Use the CalculateTax function to find the tax amount for employees earning more than 50,000.
20. Call the GetEmployeeCount function to display the total number of employees.

5. Cursors

21. Create a cursor EmployeeCursor that retrieves all employees from the Employees table and prints their names one by one.
22. Modify the EmployeeCursor to display employees in the "Sales" department only.
23. Create a cursor UpdateSalariesCursor that increases the salary of employees earning less than 40,000 by 5%.
24. Use a cursor to find and display employees whose salaries are above the company's average salary.
25. Close and deallocate the cursor EmployeeCursor after processing all employees.

SREENARAYANAGURU OPEN UNIVERSITY

QP CODE: …… Reg. No : .....

Name : …….....

Model Question Paper- set-I

End Semester Examination

BACHELOR OF COMPUTER APPLICATIONS

BCA 2024

B21CA01SE: PROGRAMMING IN JAVA

(CBCS - UG)

2023-24 - Admission Onwards

Time: 3 Hours Max Marks: 70

Section A

Answer any 10 questions. Each carries one mark **(10×1=10 marks)**

1. Write any 4 features of Java.
2. What keyword is used to include a package in a Java program?
3. Which class reads bytes from a file?
4. Which access modifier restricts a method's visibility to the defining class only?
5. What is the index of the last element in an array of size 5?
6. Which package in Java is automatically imported by the compiler?
7. What is an abstract class in Java?
8. Which interface must be implemented to define a task for a thread?
9. Which class in Java is used for handling runtime exceptions?
10. Which concept in Object-Oriented Programming allows a subclass to provide a specific implementation of a method already defined in its superclass?
11. What keyword is used to declare a method that cannot be overridden?

12. What type of method is called when a new object is created in Java?

13. What is platform independence in JDBC?

14. Which method is used to begin a thread's execution?

15. Name one SQL operation supported by JDBC.

Section B

Answer any 5 questions. Each carries two marks **(5×2=10)**

16. Write about any two non-primitive data types in java.

17. Write the features of JVM

18. What is a package?

19. What is an array ?

20. What is the use of 'final' keyword in Java?

21. What is the difference between String and StringBuffer in Java?

22. What is an exception in Java, and how is it handled using a try-catch block?

23. List the steps involved in establishing a JDBC database connection.

24. Identify key features of JDBC that make it a versatile choice for database interactions in Java applications.

25. Consider the following code snippet and write the output obtained.

```
class MyThread extends Thread {
    public void run() {
        for (int i = 0; i < 5; i++) {
            System.out.println(i);
        }
    }
}
public class Test {
```

```
    public static void main(String[] args) {
        MyThread t1 = new MyThread();
        t1.start();
    }
}
```

Section C

Answer any 5 questions. Each carries four marks **(4x5=20)**

26. Explain any 5 operators in java.
27. Describe the subclasses of InputStream class in Java
28. How an array is declared in Java?
29. How do you import and use a package in Java?
30. What are the advantages of using methods in java
31. What is method overriding and method overloading in Java? Provide an example of each.
32. Explain the concept of inheritance in Java. How does inheritance promote code reusability?
33. What is exception handling in Java? Explain the use of try, catch, and finally blocks with an example.
34. List the benefits of using JDBC for database connectivity in Java applications.
35. Discuss the life cycle of an applet and how event handling is implemented in Java applets.

Section D

Answer any 2 questions. Each carries fifteen mark **(2x15=30)**

36. Explain the concept of packages in Java. Provide a sample code to demonstrate the creation and usage of a user-defined package.
37. Define the four main access modifiers in Java and discuss their role in controlling visibility and accessibility with proper example programs.

38. Explain in detail the complete process of using stored procedures in JDBC. Include the necessary steps to create a connection, prepare the CallableStatement, pass parameters, execute the procedure, and retrieve output values.

39. Write a Java program that creates two threads using the Thread class, where:

    One thread calculates the sum of numbers from 1 to 10.

    The other thread calculates the factorial of 5.

SREENARAYANAGURU OPEN UNIVERSITY

QP CODE: …… Reg. No : .....

Name : …….....

Model Question Paper- set-II

End Semester Examination

BACHELOR OF COMPUTER APPLICATIONS

BCA 2024

B21CA01SE: PROGRAMMING IN JAVA

(CBCS - UG)

2023-24 - Admission Onwards

Time: 3 Hours Max Marks: 70

Section A

Answer any 10 questions. Each carries one mark **(10×1=10 marks)**

1. Write any 4 common features of Java IDE.
2. Write the syntax for single-line and multi-line comments.
3. What is the full form of JVM?
4. How do you execute an SQL query using JDBC?
5. What is the role of Statement in JDBC for processing SQL queries?
6. Name one SQL operation supported by JDBC.
7. Which class in Java is commonly used to read input from the user?
8. What keyword is used to define an abstract class in Java?
9. Which keyword is used for inheritance in Java?
10. What type of package is created by developers to improve code organization and reuse?
11. Which class in Java is used to create a string that can be modified?

12. What is the purpose of the catch block in exception handling?

13. What is an Applet in Java?

14. What is the role of an event listener in Java's event handling?

15. Which package contains core classes fundamental to Java?

Section B

Answer any 5 questions. Each carries two marks **(5×2=10)**

16. Write about primitive data types in Java.

17. How is abstraction achieved in Java? Name the two ways.

18. List two key rules that must be followed when overriding a method in Java.

19. How do you execute an SQL query using JDBC?

20. How do you call a stored procedure in JDBC?

21. What is abstraction in object-oriented programming? Explain with an example.

22. What is the difference between String and StringBuffer in Java?

23. What is the significance of the finally block in exception handling?

24. Explain how to create a thread using the Runnable interface in Java.

25. What are the advantages of using packages?

Section C

Answer any 5 questions. Each carries four marks **(4x5=20)**

26. Explain about the importance of JVM.

27. Explain different types of inheritance with examples in Java.

28. Discuss method overloading with an example. Why is it useful?

29. List the benefits of using JDBC for database connectivity in Java applications.

30. Explain the different types of statements available in JDBC for executing SQL queries.

31. What is the difference between method overriding and method overloading?

32. Explain the Delegation Event Model in Java with an example.

33. What is exception handling in Java, and why is it important?

34. What is the purpose of the try-catch block in Java? Explain with an example.

35. Consider the following Java code that creates a thread using the Thread class.

Section D

Answer any 2 questions. Each carries fifteen mark **(2x15=30)**

36. Write a Java program that does the following:

    1. Accepts two integers from the user using the Scanner class.

    2. Handles the following exceptions:

       ArithmeticException: If the user enters zero as the second number, display an appropriate error message.

       InputMismatchException: If the user enters a non-integer value, display an appropriate error message.

37. Write a simple Java program that creates two threads using the Thread class. Each thread should print a message to the console 5 times with a 1-second delay between each print. The output should alternate between the two threads.

38. Write a Java program that initializes an array of 10 integers with user-defined values. Calculate and display the average of the numbers in the array.

39. Explain the various OutputStream classes available in Java for writing data.

SREENARAYANAGURU OPEN UNIVERSITY

QP CODE: …… Reg. No : .....

Name : …….....

Model Question Paper- set-I

End Semester Examination

BACHELOR OF COMPUTER APPLICATIONS

BCA 2024

B21CA05DC: DATABASE MANAGEMENT SYSTEMS

(CBCS - UG)

2023-24 - Admission Onwards

Time: 3 Hours Max Marks: 70

Section A

Answer any 10 questions. Each carries one mark **(10×1=10 marks)**

1. What does the COUNT() function do in SQL?
2. Which SQL function is used to find the highest value in a column?
3. What is a view in SQL?
4. What is the key function of a DBMS?
5. Which abstraction level in DBMS hides the details of the physical storage of data?
6. In which technique a transaction does not modify the database until it has committed?
7. Write about cursor in PL/SQL.
8. What is an active dataset in PL/SQL?
9. What is the undesirable situation in which two or more transactions wait indefinitely for each other to release locks called?
10. In which technique older transactions have to wait for younger transactions to release the resource?

11. What is cardinality in an ER diagram?

12. What is a trigger in a database?

13. What is a transaction in a database?

14. Define the term 'commit' in transaction management.

15. What is a serial schedule?

Section B

Answer any 5 questions. Each carries two marks **(5×2=10)**

16. What does the LIKE operator do in SQL, and how is it used in a query? Provide an example.

17. Explain how the LEN() and LENGTH() functions are used in SQL. What is the difference between them?

18. Explain the difference between the external and internal levels of data abstraction in DBMS.

19. Explain the different types of attributes in the ER model with examples.

20. Explain about PL/SQL Cursor FOR LOOP with an example.

21. What is starvation?

22. What is a checkpoint?

23. Define schema and instance in a database.

24. What are the different types of database models apart from the relational model?

25. Differentiate between row-level and statement-level triggers.

Section C

Answer any 5 questions. Each carries four marks **(4x5=20)**

26. Describe the different types of relationships in the ER model.

27. Describe the types of views in SQL: simple views, complex views, and materialized views, with examples of their uses.

28. What is the purpose of rollback in transaction management?

29. Explain the conditions of the deadlock.

30. Explain the structure of a PL/SQLProcedure.

31. Discuss the various types of failures in databases.

32. How do generalization and specialization work in an ER model?

33. Discuss three common usages of triggers in databases.

34. Describe the failed and aborted states of a transaction with examples.

35. Explain how a precedence graph is used to check conflict serializability.

Section D

Answer any 2 questions. Each carries fifteen mark **(2x15=30)**

36. Draw an ER diagram with entities *Student* (StudentID, Name, Age) and *Course* (CourseID, CourseName), and a relationship *Enrolls* between them, where each student can enroll in multiple courses and each course can have multiple students. What type of relationship is represented between the *Student* and *Course* entities, and what is its cardinality?

37. Explain the various built-in functions in SQL.

38. Discuss the different timing events (BEFORE and AFTER) for triggers and how they influence the behaviour of the trigger during INSERT, UPDATE, or DELETE operations.

39. Explain the importance of serializability in maintaining database consistency, comparing conflict and view serializability with examples.

SREENARAYANAGURU OPEN UNIVERSITY

QP CODE: …… Reg. No : .....

Name : …….....

Model Question Paper- set-II

End Semester Examination

BACHELOR OF COMPUTER APPLICATIONS

BCA 2024

B21CA05DC: DATABASE MANAGEMENT SYSTEMS

(CBCS - UG)

2023-24 - Admission Onwards

Time: 3 Hours Max Marks: 70

Section A

Answer any 10 questions. Each carries one mark **(10×1=10 marks)**

1. What is the term used to represent the number of rows in a relation?
2. Which normal form satisfies the condition that the relation is free from transitive

   dependencies?

   Which SQL command is used for deleting a table in a database?
3. How do you calculate the average value of a column in SQL?
4. What does the ROLLBACK command do in SQL?
5. How do you create a view in SQL?
6. What is the primary difference between a file system and a DBMS?
7. What does DBMS stand for ?
8. Which model represents data as a collection of entities and relationships?
9. Write about PL/SQL Data Types.

10. Immediate Update is also called.

11. Which graph is used for deadlock detection?

12. What is the process of restoring a database from a failure called?

13. What is the difference between BEFORE and AFTER triggers?

14. What does the 'A' in ACID properties stand for?

15. Define a non-serial schedule.

Section B

Answer any 5 questions. Each carries two marks **(5×2=10)**

16. What is the difference between DDL (Data Definition Language) and DML (Data Manipulation Language) in SQL?

17. What is a view in SQL, and how does it differ from a table? Provide an example of creating a view.

18. How does the MIN() function work in SQL? Provide an example of finding the minimum value in a column.

19. What are the advantages of using a DBMS over a traditional file system?

20. What is the difference between a primary key and a candidate key in the ER model?

21. Explain about different operators in PL/SQL.

22. What is write-ahead logging?

23. What is a lock?

24. Explain the significance of the WHEN condition in triggers.

25. Explain the difference between read-only and read-write transactions.

26. Why is serializability important in DBMS?

Section C

Answer any 5 questions. Each carries four marks **(4x5=20)**

27. Explain the structure of a DBMS with its key components.

28. Explain how to insert and delete rows in a view. Provide an example of

deleting a row from a view and explain how it affects the base table.

29. Explain the steps involved in converting an ER model to a relational schema with an example.

30. Define functional dependency. Explain the difference between full functional dependency and partial functional dependency with an example.

31. Explain the difference between INNER JOIN, LEFT JOIN, and RIGHT JOIN in SQL with an example.

32. What is a subquery in SQL? Explain its types with an example.

33. Write about create or replace trigger and dropping a trigger with an example.

34. Differentiate immediate update and deferred update.

35. Explain how deadlock can be prevented in a database.

36. How can you drop a trigger? Write the syntax and an example.

37. Discuss the difference between conflict serializability and view serializability.

Section D

Answer any 2 questions. Each carries fifteen mark **(2x15=30)**

38. Compare and contrast the different types of database models.

39. Describe the complete structure of a trigger, including the header and body. Discuss the importance of each section with relevant examples.

40. What are the 3 different types of loops in PL/SQL? Explain each with an example.

41. Describe the ACID properties of transactions in detail, using examples to illustrate each property.

42. Explain the process of transaction rollback with an example of a failed bank transaction.

SREENARAYANAGURU OPEN UNIVERSITY

QP CODE: …… Reg. No : .....

Name : …….....

Model Question Paper- set-I

End Semester Examination

BACHELOR OF COMPUTER APPLICATIONS

BCA 2024

B21CA06DC: COMMUNICATION AND NETWORKING

(CBCS - UG)

2023-24 - Admission Onwards

Time: 3 Hours Max Marks: 70

Section A

Answer any 10 questions. Each carries one mark **(10×1=10 marks)**

1. What are the three types of data flow in a network?
2. What are the key criteria for an effective and efficient network?
3. Which type of transmission media uses physical cables for data transmission?
4. What is the unique identifier for every device on a network?
5. How many layers are there in the OSI model?
6. Name one error detection technique used in the OSI model.
7. Which layer of the OSI model handles error detection and correction?
8. Write any two advantages of subnetting.
9. What are the details included in a Packet header?
10. What type of attack involves forging a source IP address?
11. What does PKI stand for in the context of digital signatures?

12. Which security service ensures neither the sender nor the receiver can deny having participated in a communication?

13. What is the main purpose of the Internet Control Message Protocol (ICMP)?

14. Which protocol layer does HTTP belong to?

15. List the advantages of connectionless protocols?

Section B

Answer any 5 questions. Each carries two marks **(5×2=10)**

16. What is the difference between guided and unguided transmission media?

17. What are the key components of a data communication system?

18. What are the two main types of digital-to-digital conversion?

19. What is VPN?

20. What is a burst error?

21. What is message switching?

22. Why is mesh topology considered highly reliable?

23. Explain about any 3 networking devices.

24. What is Notarization?

25. Explain the difference between IPv4 and IPv6 addressing.

Section C

Answer any 5 questions. Each carries four marks **(4x5=20)**

26. Explain the different types of data flow in a network with examples.

27. Explain the physical structure attributes of a network.

28. Explain unguided transmission media.

29. What are the common security threats to networks?

30. Compare and contrast Star, Bus, and Mesh topologies based on reliability, cost, scalability, and ease of troubleshooting.

31. Why are protocols important?

32. Write about RIP and IGRP.

33. Explain the different types of routing.

34. Discuss the key differences between TCP and UDP protocols.

35. Explain how the Three-Way Handshake process works in connection-oriented protocols.

Section D

Answer any 2 questions. Each carries fifteen mark **(2x15=30)**

36. Discuss in detail the components, data representation, data flow, and network criteria in a data communication system. Provide examples for each component and explain their role in ensuring effective communication.

37. Explain the OSI model in detail, describing the functions of each of its seven layers. Also discuss the role of error detection and correction in network communication.

38. Explain 4-way-Handshake. Why Does TCP Connect Termination Need a 4-way-Handshake?

39. Explain about congestion control methods in detail.

SREENARAYANAGURU OPEN UNIVERSITY

QP CODE: …… Reg. No : .....

Name : …….....

Model Question Paper- set-II

End Semester Examination

BACHELOR OF COMPUTER APPLICATIONS

BCA 2024

B21CA06DC: COMMUNICATION AND NETWORKING

(CBCS - UG)

2023-24 - Admission Onwards

Time: 3 Hours Max Marks: 70

Section A

Answer any 10 questions. Each carries one mark **(10×1=10 marks)**

1. What is the basic unit of digital data?
2. Which transmission mode allows data to be sent in both directions simultaneously?
3. What defines the set of rules for communication in a network?
4. What does TCP/IP stand for?
5. Name one application protocol used in the application layer of the TCP/IP model.
6. Which protocol is used for email transmission?
7. What is the purpose of the Four-Way Handshake in TCP?
8. Write about different classification of networks.
9. What is the duty of a router in networking?
10. Which switching technique is used in traditional telephone networks?
11. Which OSI model layer is responsible for routing data between devices?

12. What type of network connects devices in a small area?

13. Which connector is most commonly used for twisted pair cables?

14. What is a special area in a network for extra security called?

15. Which topology uses a single backbone cable to connect all devices?

Section B

Answer any 5 questions. Each carries two marks **(5×2=10)**

16. What is the role of a transmission medium in data communication?

17. What is the difference between simplex and half-duplex transmission?

18. Explain the two types of network connections based on physical structure?

19. Explain the main difference between connection-oriented and connectionless protocols.

20. What is the primary role of the Internet Protocol (IP)?

21. Why is Congestion Control necessary? Write any four points.

22. Write about different distance vector protocols.

23. Describe the concept of Subnetting and its importance in IPv4 addressing.

24. What are the main roles of the Data Link layer in the OSI model?

25. What is IDS/IPS?

Section C

Answer any 5 questions. Each carries four marks **(4x5=20)**

26. Explain the five key components of a data communication system with an example.

27. Describe the three techniques of digital-to-digital conversion: line coding, block coding, and scrambling, with examples.

28. Explain the different types of guided transmission media and their advantages and disadvantages.

29. Compare and contrast Classful and Classless IP addressing, including the advantages and disadvantages of each.

30. Explain how data is transferred using FTP and the key features of FTP.

31. Analyze the roles of routers, gateways, switches and bridges in a networking structure.

32. Discuss about interior gateway protocols and exterior gateway protocols.

33. Describe the three types of error detection techniques used in the OSI model.

34. Define NAT and explain its types.

35. Write about Fibre optic cable.

Section D

Answer any 2 questions. Each carries fifteen mark **(2x15=30)**

36. Explain the complete working of the TCP/IP suite. Start with the role of each layer (Application, Transport, Network, and Network Access) and the protocols used at each layer. Discuss in detail how TCP handles reliable communication through processes like Three-Way Handshake, Data Transfer, Error Detection, Congestion Control, and Four-Way Handshake.

37. Explain different types of line coding techniques used in digital transmission. Discuss Non-Return to Zero (NRZ), Return to Zero (RZ), Manchester, and Differential Manchester encoding in detail. Describe their working principles, advantages, and disadvantages with appropriate diagrams.

38. Explain the concept of error detection and correction in the OSI model. Discuss types of errors, error control mechanisms, and techniques used to detect and correct errors during data transmission.

39. Explain about Wireless Network configuration.

സർവ്വകലാശാലാഗീതം

വിദ്യയാൽ സ്വതന്ത്രരാകണം
വിശ്വപൗരരായി മാറണം
ഗ്രഹപ്രസാദമായ് വിളങ്ങണം
ഗുരുപ്രകാശമേ നയിക്കണേ

കൂരിരുട്ടിൽ നിന്നു ഞങ്ങളെ
സൂര്യവീഥിയിൽ തെളിക്കണം
സ്നേഹദീപ്തിയായ് വിളങ്ങണം
നീതിവൈജയന്തി പാറണം

ശാസ്ത്രവ്യാപ്തിയെന്നുമേകണം
ജാതിഭേദമാകെ മാറണം
ബോധരശ്മിയിൽ തിളങ്ങുവാൻ
ജ്ഞാനകേന്ദ്രമേ ജ്വലിക്കണേ

കുരീപ്പുഴ ശ്രീകുമാർ

SREENARAYANAGURU OPEN UNIVERSITY

Regional Centres

Kozhikode
Govt. Arts and Science College
Meenchantha, Kozhikode,
Kerala, Pin: 673002
Ph: 04952920228
email: rckdirector@sgou.ac.in

Thalassery
Govt. Brennen College
Dharmadam, Thalassery,
Kannur, Pin: 670106
Ph: 04902990494
email: rctdirector@sgou.ac.in

Tripunithura
Govt. College
Tripunithura, Ernakulam,
Kerala, Pin: 682301
Ph: 04842927436
email: rcedirector@sgou.ac.in

Pattambi
Sree Neelakanta Govt. Sanskrit College
Pattambi, Palakkad,
Kerala, Pin: 679303
Ph: 04662912009
email: rcpdirector@sgou.ac.in

NO TO DRUGS തിരിച്ചിറങ്ങാൻ പ്രയാസമാണ്

ആരോഗ്യ കുടുംബക്ഷേമ വകുപ്പ്, കേരള സർക്കാർ

Database Management Systems

COURSE CODE: B21CA05DC

Sreenarayanaguru Open University
Kollam, Kerala Pin- 691601, email: info@sgou.ac.in, www.sgou.ac.in Ph: +91 474 2966841

ISBN 978-81-984969-0-4