TASK DAY -1

1. Installation of MySQL Workbench and setup a local instance connection for hands on.
2. Create a word documentation of Introduction to DBMS, RDMS - Include all points that we have discussed so far.

    - Include additional important points from your research related to DBMS

    - Include Advantages and disadvantages of DBMS and RDMS with examples - Include different other types of normalization techniques.

Database Management System (DBMS)

- Database Management System (DBMS) is a software that is used to manage the data. Some of the popular DBMS software are MySQL, IBM Db2, Oracle, PostgreSQL etc.

- DBMS provides many operations e.g. creating a database, storing in the database, updating an existing database, delete from the database. DBMS is a system that enables you to store, modify and retrieve data in an organized way. It also provides security to the database.

Why DBMS?

DBMS is used for small organization and deal with small data. The system offers many benefits over the traditional file system, including the following:

- It helps maintain data uniformity.
- Handles large sets of data efficiently.
- Versatile
- Faster way of managing data

Types of DBMS

1. **Relational DBMS (RDBMS):** An RDBMS stores data in tables with rows and columns and uses SQL (Structured Query Language) to manipulate the data.

2. **Object-Oriented DBMS (OODBMS):** An OODBMS stores data as objects, which can be manipulated using object-oriented programming languages.

3. **NoSQL DBMS:** A NoSQL DBMS stores data in non-relational data structures, such as key-value pairs, document-based models, or graph models.

Applications of DBMS

- Telecom: DBMS is crucial for managing information related to calls, network usage, and customer details in the telecom industry. Without DBMS, handling the continuous flow of data would be impractical.

- Industry: Whether it's a manufacturing unit, warehouse, or distribution centre, all require a database to track inventory, shipments, and product movements.

- Banking System: DBMS is essential for storing customer information, tracking transactions, and ensuring the security of sensitive data in banking systems.

- Sales: DBMS helps in storing customer information, managing inventory, and generating reports for sales analysis.

- Airlines: DBMS is used to manage flight reservations, schedules, and real-time updates to ensure accurate and efficient allocation of seats.

  Education Sector: Schools and colleges use DBMS to store and retrieve data related to student records, staff details, course information, and attendance.

- Online Shopping: E-commerce platforms like Amazon and Flipkart rely on DBMS to manage product catalogues, user preferences, and secure transactions, ensuring a smooth shopping experience for users.

Pros of DBMS:

- **Multiple User Interface:** DBMS provides users with different types of UI like graphical UI, application program interfaces (API), etc.
  Example: An employee uses a graphical UI to access and update customer records in a retail DBMS.

- **Controls Database Redundancy:** DBMS can control data redundancy by storing all information in a single database file.
  Example: In a university database, student information like name, address, and course enrolment is stored in a single table, reducing redundancy.

- **Reduce Time:** DBMS reduces the time of development and the need for maintenance.

Example: A software company uses a DBMS to quickly develop and deploy a new customer relationship management (CRM) system, reducing development time.

- **Data Sharing:** Authorized users in an organization can easily share data and info among multiple other users in DBMS.
  Example: A marketing team accesses customer data stored in a DBMS to personalize marketing campaigns.

- **Easily Maintenance:** DBMS can be easily maintained because of the database system's centralized nature.
  Example: An IT administrator performs routine maintenance tasks like backups and updates on a centralized DBMS server.

- **Backup:** DBMS provides users with subsystems of backup and recovery that create an automatic backup of data from software and hardware failures.
  Example: A financial institution uses DBMS backup features to recover customer transaction data in case of a hardware failure.

Cons of DBMS:

- **Cost of Hardware and Software:** A high-speed data processor and large memory size are required to run any DBMS software.
  Example: A company invests in high-end server hardware and DBMS software licenses to support its enterprise-level database.

- **Size:** DBMS occupies large disk space and memory to run efficiently.
  Example: A database for a multinational corporation occupies several terabytes of disk space to store vast amounts of transaction data.

- **Complexity:** Additional complexity and requirements are created by the database systems.
  Example: DBMS configuration and administration require specialized knowledge and training, adding complexity to IT operations.

- **Higher Impact of Failure:** Failure impacts the database significantly because all data is usually stored in a single database.
  Example: A retail business faces severe disruptions in operations if its centralized DBMS server experiences a critical failure, leading to loss of sales data and customer information.

Relational Database Management System (RDBMS)

- RDBMS or Relational Database Management System is a database system software that manages and maintains data in a tabular format.

- It is the software that operates on a relational schema (database arranged in tables with rows and columns).

- In a relational database, each row in the table is a record with a unique ID called the key. The columns of the table hold attributes of the data, and each record usually has a value for each attribute, making it easy to establish the relationships among data points.

- A relational database management system (RDBMS) is a program used to create, update, and manage relational databases. Some of the most well-known RDBMSs include MySQL, PostgreSQL, MariaDB, Microsoft SQL Server, and Oracle Database.

Why RDBMS?

An RDBMS offers businesses a systematic view of data, which can be used to enhance different aspects of decision-making. Relational databases offer several other benefits as well, including:

- Allow multiple user access.
- Store large packs of data.
- Minimum Data Redundancy
- Maintains Data Integration
- Better Tools for Structuring and Organizing Data

Primary key:

The primary key finds out the similarity in the relationship. For the entire table, there is only one primary key. Every table has got a particular primary key that cannot be shared by other tables.

Foreign key:

The foreign key is a key used for a different table of data which is referred by the primary key. There are many foreign keys for a single table. It depends on the primary key and its decision to refer those foreign keys to the table. Every foreign key can be shared, and it speaks about the coordination among the data of different tables.

Properties of Relational Database

There are following four commonly known properties of a relational model known as ACID properties, where:

1. A means Atomicity: This ensures the data operation will complete either with success or with failure. It follows the 'all or nothing' strategy. For example, a transaction will either be committed or will abort.

2. C means Consistency: If we perform any operation over the data, its value before and after the operation should be preserved. For example, the account balance before and after the transaction should be correct, i.e., it should remain conserved.

3. I mean Isolation: There can be concurrent users for accessing data at the same time from the database. Thus, isolation between the data should remain isolated. For example, when multiple transactions occur at the same time, one transaction effects should not be visible to the other transactions in the database.

4. D means Durability: It ensures that once it completes the operation and commits the data, data changes should remain permanent.

Applications of RDBMS:

- Business Management: They help companies organize customer data, manage supply chains, and track inventory.

- Web Services: RDBMS form the backbone of websites and apps, storing user profiles, posts, and comments.

- Data Analysis: They're crucial for analysing large datasets, providing insights for businesses and scientific research.

- Scientific Research: Scientists use RDBMS to manage experimental data and simulations, aiding research in fields like genomics and climate modelling.

- Government Services: Governments use RDBMS to manage citizen records, tax data, and public service information.

Pros of RDBMS:

- **Data Integrity**: RDBMS enforces relationships between data, ensuring referential integrity.
  Example: In a university database, the relationship between student records and course enrolment ensures that only valid student IDs are associated with enrolled courses.

- **Query Optimization:** RDBMS optimizes queries for efficient data retrieval and manipulation.

Example: A retail business executes complex SQL queries to analyse sales data and identify trends, helping in strategic decision-making.

- **Scalability:** RDBMS systems can scale to accommodate growing volumes of data and users.
  Example: An e-commerce platform expands its database infrastructure to handle increased website traffic during holiday seasons without sacrificing performance.

- **Standardization**: SQL provides a standardized language for interacting with RDBMS systems.
  Example: An IT professional proficient in SQL can seamlessly transition between different RDBMS platforms like MySQL, PostgreSQL, and Oracle Database.

- **ACID Compliance:** RDBMS systems ensure ACID (Atomicity, Consistency, Isolation, Durability) properties for transactions.
  Example: A banking system guarantees that a funds transfer transaction is atomic, consistent, isolated from other transactions, and durable even in the event of a system failure.

Cons of RDBMS:

- **Normalization Overhead:** Normalization can introduce complexity and overhead for database design and maintenance.
  Example: In a healthcare database, normalization may require splitting patient information across multiple tables, increasing the complexity of queries to retrieve patient records.

- **Performance Limitations:** Complex queries or large datasets can impact performance in RDBMS systems.
  Example: A data analytics platform experiences slowdowns when processing large volumes of transactional data stored in an RDBMS, requiring optimization of SQL queries.

- **Vendor Lock-in:** Switching between RDBMS vendors can be challenging due to differences in SQL dialects and features.
  Example: A company faces migration challenges when transitioning from Oracle Database to PostgreSQL due to differences in stored procedures and data types.

- **Scalability Challenges:** Scaling RDBMS systems horizontally can be difficult compared to NoSQL databases.
  Example: A social media platform encounters difficulties in horizontally scaling its RDBMS to handle the exponential growth of user-generated content, prompting consideration of NoSQL alternatives.

Difference between DBMS and RDBMS

No.	DBMS	RDBMS
1)	DBMS applications store data as file.	RDBMS applications store data in a tabular form.
2)	In DBMS, data is generally stored in either a hierarchical form or a navigational form.	In RDBMS, the tables have an identifier called primary key and the data values are stored in the form of tables.
3)	Normalization is not present in DBMS.	Normalization is present in RDBMS.
4)	DBMS does not apply any security with regards to data manipulation.	RDBMS defines the integrity constraint for the purpose of ACID (Atomicity, Consistency, Isolation and Durability) property.
5)	DBMS uses file system to store data, so there will be no relation between the tables.	in RDBMS, data values are stored in the form of tables, so a relationship between these data values will be stored in the form of a table as well.
6)	DBMS must provide some uniform methods to access the stored information.	RDBMS system supports a tabular structure of the data and a relationship between them to access the stored information.
7)	DBMS does not support distributed database.	RDBMS supports distributed database.
8)	DBMS is meant to be for small organization and deal with small data. it supports single user.	RDBMS is designed to handle large amount of data. it supports multiple users.
9)	Examples of DBMS are file systems, xml etc.	Example of RDBMS are MySQL, Postgres, SQL server, oracle etc.

Normalization

Normalization is a process used in database design to organize tables and minimize redundancy. There are several normalization techniques, each with its own rules and objectives.

Types of Normalization Techniques

1. **First Normal Form (1NF):**
    - Eliminates repeating groups within a table.
    - Ensures that each column contains atomic values.
    - Example: Breaking down a table of customer orders into individual rows for each order item.

2. **Second Normal Form (2NF):**
    - Builds on 1NF.
    - Ensures that non-key attributes are fully functionally dependent on the primary key.
    - Removes partial dependencies.
    - Example: Splitting a table of orders into separate tables for orders and order details, where order details depend fully on the order ID.

3. **Third Normal Form (3NF):**
    - Builds on 2NF.
    - Eliminates transitive dependencies between non-key attributes.
    - Ensures that every non-key attribute is dependent only on the primary key.
    - Example: Breaking down a table of employees into separate tables for employee details and department details, removing dependencies on department attributes from the employee table.

4. **Boyce-Codd Normal Form (BCNF):**
    - A stricter form of 3NF.
    - Ensures that every determinant is a candidate key.
    - Example: Decomposing a table of student enrolments into separate tables for student details and course details, ensuring each determinant uniquely determines all other attributes.

5. **Fourth Normal Form (4NF):**
    - Addresses multi-valued dependencies.
    - Ensures that no multi-valued dependencies exist between attributes.
    - Example: Splitting a table of customer preferences into separate tables for individual preferences, removing multi-valued dependencies.

6. **Fifth Normal Form (5NF):**
    - Addresses join dependencies.
    - Ensures that all join dependencies are implied by the candidate keys.
    - Example: Decomposing a table of sales transactions into separate tables for products, customers, and sales details, eliminating join dependencies.

TASK DAY-2

1. What is normalization in the context of database design?
   Normalization is the process of organizing data in a database table to eliminate redundancy and dependency issues. It involves breaking down a table into multiple smaller tables, each containing a specific set of related attributes. By applying normalization techniques, such as first, second, and third normal forms, data redundancy and update anomalies are minimized, leading to more efficient storage and improved data integrity.

2. Why is normalization important for database management?
   Here are some key reasons why normalization is essential in database management:
   1) Reduction of Data Redundancy:
      Normalization helps eliminate duplicate data by breaking down tables into smaller, more manageable pieces. Redundant data is stored only once, reducing storage space, and minimizing the risk of inconsistencies.
   2) Minimization of Data Anomalies:
      Anomalies, such as insertion, update, and deletion anomalies, can occur when data is not properly organized. Normalization helps minimize these anomalies by structuring the data to meet specific requirements, ensuring that changes to the database are logical and consistent.
   3) Improved Data Integrity:
      By eliminating redundancy and dependencies, normalization improves data integrity. Each piece of data is stored in only one place, reducing the likelihood of conflicting information and making it easier to maintain accurate and reliable data.
   4) Simplified Maintenance:
      Normalized databases are generally easier to maintain. Updates, inserts, and deletions are more straightforward, and changes to the structure of the database are less likely to impact multiple parts of the system.
   5) Efficient Query Performance:
      Well-normalized databases often lead to more efficient query performance. Queries can be written with simpler JOIN operations, and indexes can be used more effectively to speed up data retrieval.
   6) Scalability and Flexibility:
      Normalization provides a foundation for a scalable and flexible database structure. As the data model becomes more complex, normalization allows for the addition of new tables and relationships without introducing unnecessary complications.

3. Explain the concept of data redundancy and how normalization helps to mitigate it.
   Data redundancy refers to the duplication of data within a database or data storage system. When the same piece of data is stored in multiple places, it can lead to several issues, including:
   - Wasted storage space: Storing the same data in multiple locations consumes more storage space than necessary.
   - Inconsistencies: If the same data is updated in one place but not in others, it can lead to inconsistencies and errors in the database.
   - Complexity: Managing redundant data increases the complexity of the database, making it harder to maintain and understand.

   Normalization helps mitigate data redundancy by:

   - Eliminating duplicate data: By breaking down tables and storing data in separate tables, normalization ensures that each piece of data is stored only once.
   - Reducing storage space: Storing data in a normalized form typically requires less storage space compared to storing redundant data.
   - Improving data integrity: By reducing redundancy, normalization helps ensure that data remains consistent and accurate across the database.
4. What are the primary goals of normalization?

Normalization is a crucial concept in database design, aimed at organizing data efficiently and avoiding redundancy and anomalies. The primary goals of normalization are:

- Eliminate Data Redundancy: Redundant data can lead to inconsistencies and increase storage space requirements. Normalization helps to reduce redundancy by breaking down data into smaller, more manageable tables.

- Minimize Data Modification Anomalies: Data modification anomalies occur when changes to data in one part of the database cause inconsistencies or errors elsewhere. Normalization helps to minimize these anomalies by organizing data logically.

- Ensure Data Integrity: By structuring data properly, normalization helps maintain data integrity. This means that the data accurately reflects the real-world entities it represents and adheres to defined rules and constraints.

- Facilitate Query Optimization: Normalization simplifies querying by breaking down data into smaller, related tables. This makes it easier to retrieve and manipulate data efficiently, leading to better query performance.

- Reduce Storage Space: Normalization typically leads to more efficient use of storage space by eliminating redundant data and organizing data more compactly.

5. List and explain the different normal forms in normalization theory.

1) First Normal Form (1NF):

- Eliminates repeating groups within a table.
- Ensures that each column contains atomic values.
- Example: Breaking down a table of customer orders into individual rows for each order item.

2) Second Normal Form (2NF):

- Builds on 1NF.
- Ensures that non-key attributes are fully functionally dependent on the primary key.
- Removes partial dependencies.
- Example: Splitting a table of orders into separate tables for orders and order details, where order details depend fully on the order ID.

3) Third Normal Form (3NF):

- Builds on 2NF.
- Eliminates transitive dependencies between non-key attributes.
- Ensures that every non-key attribute is dependent only on the primary key.
- Example: Breaking down a table of employees into separate tables for employee details and department details, removing dependencies on department attributes from the employee table.

4) **Boyce-Codd Normal Form (BCNF):**
   - A stricter form of 3NF.
   - Ensures that every determinant is a candidate key.
   - Example: Decomposing a table of student enrolments into separate tables for student details and course details, ensuring each determinant uniquely determines all other attributes.

5) **Fourth Normal Form (4NF):**
   - Addresses multi-valued dependencies.
   - Ensures that no multi-valued dependencies exist between attributes.
   - Example: Splitting a table of customer preferences into separate tables for individual preferences, removing multi-valued dependencies.

6) **Fifth Normal Form (5NF):**
   - Addresses join dependencies.
   - Ensures that all join dependencies are implied by the candidate keys.
   - Example: Decomposing a table of sales transactions into separate tables for products, customers, and sales details, eliminating join dependencies.

6. What is First Normal Form (1NF) and why is it necessary? Explain with example.

First Normal Form (1NF)

- A relation will be 1NF if it contains an atomic value.
- It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
- In simpler terms, each column in a table must have a single, indivisible value for each row. This eliminates repeating groups and ensures that each piece of data is uniquely identified by the primary key.

For example, consider a table that stores information about students and their courses:

Student ID	Student Name	Course IDs
1	Anu	101, 102
2	Joby	102, 103

In this table, the "Course IDs" column violates the 1NF because it contains multiple values separated by commas. To normalize this table to 1NF, we would split the courses into separate rows:

Student ID	Student Name	Course ID
1	Anu	101
1	Anu	102
2	Joby	102
2	Joby	103

Now, each row has a single value for the "Course ID" column, adhering to the 1NF requirement.

7. How does Second Normal Form (2NF) differ from First Normal Form (1NF)? explain with example.
   - Second Normal Form (2NF) builds upon the foundation of First Normal Form (1NF) by addressing the issue of partial dependencies within a relational database table.
   - A table is in 2NF if it meets the criteria of 1NF and if all non-prime attributes are fully functionally dependent on the primary key.

   Example:

   Suppose we have a table that stores information about employees, their projects, and the hours they worked on each project:

Employee ID	Employee Name	Project ID	Project Name	Hours Worked
1	James	101	Project A	20
1	James	102	Project B	25
2	Aby	101	Project A	30
2	Aby	103	Project C	15

   In this table, the primary key could be (Employee ID, Project ID) since each combination uniquely identifies a row.

   However, the "Project Name" column is dependent only on the "Project ID" column, not on the entire primary key. This violates 2NF because "Project Name" is not fully functionally dependent on the primary key.

   To bring this table to 2NF, we need to split it into two separate tables:

   Employee Projects:

Employee ID	Project ID	Hours Worked
1	101	20
1	102	25
2	101	30
2	103	15

   Projects:

Project ID	Project Name
101	Project A
102	Project B
103	Project C

   Now, the "Project Name" column is fully functionally dependent on the "Project ID" column in the "Projects" table, meeting the requirement of 2NF.

8. Describe Third Normal Form (3NF) and its significance in database design. Explain with example.
   - Third Normal Form (3NF) is a level of database normalization that ensures the removal of transitive dependencies within a relational database table. A table is in 3NF if it satisfies the following conditions:
   - It is in Second Normal Form (2NF).
   - There are no transitive dependencies, meaning that non-key attributes are not dependent on other non-key attributes.
   - In simpler terms, 3NF ensures that every non-key attribute is dependent only on the primary key and not on any other non-key attributes.

Example:
Consider a table that stores information about employees, their departments, and the locations of those departments:

Employee ID	Employee Name	Department ID	Department Name	Location
1	Leya	101	Sales	Building A
2	Parvathy	102	Technical	Building B
3	John	101	Technical	Building A

In this table, "Location" is functionally dependent on "Department Name," which is not the primary key. This creates a transitive dependency: "Location" depends on "Department Name," and "Department Name" depends on "Department ID," which is the primary key.

To bring this table to 3NF, we need to split it into two separate tables:

Departments:

Department ID	Department Name
101	Sales
102	Technical

Department Locations:

Department ID	Location
101	Building A
102	Building B

Now, each table is in 3NF. There are no transitive dependencies, and all non-key attributes are dependent only on the primary key. This normalization ensures data integrity, reduces redundancy, and simplifies the database structure, making it more efficient and easier to maintain.

9. What is Boyce-Codd Normal Form (BCNF) and how does it differ from Third Normal Form (3NF)? explain with example.

Boyce-Codd Normal Form (BCNF) is a higher level of database normalization than Third Normal Form (3NF). It addresses certain types of anomalies that can still exist in tables normalized to 3NF, specifically those related to functional dependencies involving candidate keys.

A table is in BCNF if it satisfies the following conditions:

It is in 3NF.

For every non-trivial functional dependency

$X \rightarrow Y$, where

X is a super key,

Y is a candidate key.

In simpler terms, BCNF ensures that every non-trivial functional dependency in the table is a dependency on a candidate key, not just any super key.

The difference between 3NF and BCNF with an example:

Consider a table that stores information about employees, their projects, and the departments to which those projects belong:

Employee ID	Employee Name	Project ID	Project Name	Department ID	Department Name
1	John Doe	101	Project A	201	Engineering
2	Jane Smith	102	Project B	202	Marketing
3	Bob Johnson	103	Project C	201	Engineering

In this table, the combination of "Employee ID" and "Project ID" forms a candidate key, as it uniquely identifies each row. The functional dependencies are: Employee ID → Employee Name, Department ID, Department Name

Project ID → Project Name, Department ID, Department Name

Department ID → Department Name

This table is in 3NF because there are no transitive dependencies. However, it's not in BCNF because there is a non-trivial functional dependency where the determinant (Employee ID, Project ID) is not a candidate key.

To normalize this table into BCNF, we need to split it into two separate tables:

Employees Projects:

Employee ID	Project ID
1	101
2	102
3	103

Projects Departments:

Project ID	Project Name	Department ID
101	Project A	201
102	Project B	202
103	Project C	201

Departments:

Department ID	Department Name
201	Engineering
202	Marketing

Now, each table is in BCNF, as every non-trivial functional dependency is a dependency on a candidate key. This normalization ensures data integrity and eliminates certain types of anomalies that can occur in less normalized tables.

10. Explain the concept of transitive dependency and its role in normalization.

Transitive dependency is a concept in database normalization that occurs when a nonprime attribute (an attribute that is not part of any candidate key) depends on another nonprime attribute, rather than directly on the primary key.

In simpler terms, transitive dependency exists when a column's value is determined by another column, which in turn is determined by the primary key.

Transitive dependencies can lead to several issues, including:

- Data Redundancy: Storing the same information in multiple places, which can lead to inconsistencies and wasted storage space.
- Update Anomalies: Difficulty in updating data without introducing inconsistencies.
- Deletion Anomalies: Deleting data might unintentionally remove other related data.

Normalization helps eliminate these problems by breaking down tables with transitive dependencies into smaller, more manageable tables that adhere to higher normal forms (such as 3NF or BCNF).

For example, consider a table of employees and their departments, where the department's location is also included:

Employee ID	Employee Name	Department ID	Department Name	Location
1	John Doe	101	Engineering	New York
2	Jane Smith	102	Marketing	Los Angeles
3	Bob Johnson	101	Engineering	New York

In this table, the "Location" attribute is transitively dependent on "Department ID" through "Department Name." To resolve this, we can normalize the table by creating separate tables for departments and locations, eliminating the transitive dependency:

Employees:

Employee ID	Employee Name	Department ID
1	John Doe	101
2	Jane Smith	102
3	Bob Johnson	101

Departments:

Department ID	Department Name
101	Engineering
102	Marketing

Locations:

Department Name	Location
Engineering	New York
Marketing	Los Angeles

This normalization ensures that each table has a single responsibility, eliminates data redundancy, and prevents update and deletion anomalies.

11. Can you provide examples illustrating the process of normalization and its application in real-world database scenarios?
Scenario: A hospital maintains a database of patients and their appointments. **Unnormalized Table:**

Patient ID	Patient Name	Date of Birth	Appointment Date	Appointment Time	Doctor
1234	John Smith	1980-01-01	2024-03-07	10:00 AM	Dr. Jones
5678	Jane Doe	1990-02-02	2024-03-08	2:00 PM	Dr. Lee

Issues:

- Redundancy: Patient details (name, date of birth) are repeated for each appointment.
- Update Anomalies: If a patient's name changes, it needs to be updated in every row associated with that patient.

Normalized Tables:

1. **Patients Table:**

Patient ID	Patient Name	Date of Birth
1234	John Smith	1980-01-01
5678	Jane Doe	1990-02-02

2. **Appointments Table:**

Appointment ID	Patient ID	Appointment Date	Appointment Time	Doctor
1	1234	2024-03-07	10:00 AM	Dr. Jones
2	5678	2024-03-08	2:00 PM	Dr. Lee

Benefits:

- Reduced redundancy: Patient information is stored only once, minimizing storage space and inconsistencies.
- Efficient updates: Updating a patient's name involves modifying only the Patients table, not every appointment entry.
- Improved data integrity: Data remains consistent and accurate as updates are centralized.
- Simpler queries: Retrieving specific information (e.g., all appointments for Dr. Jones) becomes easier with well-defined relationships between tables.
This is a simplified example, but it demonstrates how normalization helps create a more efficient and reliable database structure for managing real-world data in various scenarios.

12. Define SQL constraints and explain their significance in database management. Provide examples of different types of SQL constraints.

SQL constraints act as rules that govern the data stored within a table. They ensure the accuracy, consistency, and validity of information in your database. Here's why they're essential:

- Data Integrity: Constraints prevent invalid or inconsistent data from entering the database, safeguarding its reliability.
- Improved Data Quality: By enforcing data rules, constraints maintain the overall quality of information within the database.
- Reduced Errors: Constraints catch errors during data entry or modification, minimizing the risk of inaccurate or incomplete data.

Types of SQL Constraints:

1. NOT NULL: This constraint mandates that a column cannot contain null values. It ensures every row has a valid entry for that specific column.
   Example: In a `Customers` table, the `Customer ID` column should be declared `NOT NULL` to guarantee each customer has a unique identifier.

2. UNIQUE: This constraint enforces that all values within a column (or a set of columns) must be distinct. It prevents duplicate entries.
   Example: The `Email` column in a `Users` table can be declared `UNIQUE` to ensure only one user has a specific email address.

3. PRIMARY KEY: A table can have only one primary key constraint. It enforces uniqueness and also cannot contain null values. It essentially acts as the unique identifier for each row in the table.
   Example: The `Books` table can have an `ISBN` column declared as the primary key, guaranteeing each book has a unique identifier.

4. FOREIGN KEY: This constraint establishes a link between two tables. It references a primary key in another table, ensuring data consistency across related tables. Example: An `Orders` table can have a `Customer ID` foreign key referencing the primary key in the `Customers` table. This ensures an order is always linked to a valid customer.

5. CHECK: This constraint allows you to define a specific condition that every value in a column must satisfy. It offers more granular control over data validity.
   Example: An `Age` column can have a `CHECK` constraint defined as `Age >= 18` to ensure only users above 18 can be registered.

13. Discuss the purpose of the NOT NULL constraint in SQL. How does it differ from the UNIQUE constraint?
    - In SQL, the NOT NULL constraint is used to ensure that a column cannot have a NULL value. It enforces that every row in a table must have a value for that column, preventing the insertion of NULL values.
    - The purpose of the NOT NULL constraint is to ensure data integrity by requiring the presence of meaningful values in critical columns. It helps avoid unexpected errors or inconsistencies that could arise from missing data. For example, if a column represents a person's age, it should not allow NULL values because age is typically considered a required attribute.

- On the other hand, the UNIQUE constraint is used to ensure that all values in a column (or a combination of columns) are unique and distinct across the entire table. Unlike the NOT NULL constraint, the UNIQUE constraint does not mandate the presence of a value; it simply requires that if a value exists, it must be unique among all rows.

Here are the key differences between the NOT NULL and UNIQUE constraints:

1. Purpose:
    - NOT NULL: Ensures that a column cannot have NULL values.
    - UNIQUE: Ensures that values in a column (or combination of columns) are unique.
2. Enforcement:
    - NOT NULL: Enforces the presence of a value in a column.
    - UNIQUE: Enforces uniqueness of values in a column or set of columns.
3. Behaviour with NULL values:
    - NOT NULL: Does not allow NULL values in the column.
    - UNIQUE: Allows NULL values, but if a value is present, it must be unique.
4. Impact on data integrity:

- NOT NULL: Helps maintain data integrity by ensuring the presence of meaningful values.
- UNIQUE: Helps maintain data integrity by preventing duplicate values. 14. Explain the concept of a PRIMARY KEY constraint in SQL. What role does it play in database design and data integrity?

In SQL, a PRIMARY KEY constraint is used to uniquely identify each record in a table. It ensures that the values in the specified column (or columns) are unique for each row and that these values are not NULL. A table can have only one PRIMARY KEY constraint.

The PRIMARY KEY constraint plays a crucial role in database design and data integrity by:

- Uniquely identifying rows: The PRIMARY KEY constraint ensures that each row in a table is uniquely identified by its key values. This helps distinguish one record from another and enables efficient retrieval of specific rows.

- Enforcing data integrity: By requiring that key values be unique and not NULL, the PRIMARY KEY constraint helps maintain data integrity. It prevents duplicate records and ensures that each row can be uniquely identified, which is essential for relational databases.

- Relationships between tables: In relational database design, the PRIMARY KEY of one table is often used as a FOREIGN KEY in another table to establish relationships between them. This enables the creation of meaningful associations between data in different tables.

- Indexing: The PRIMARY KEY column(s) are automatically indexed in most database systems. This means that queries involving the PRIMARY KEY column(s) can be executed more efficiently, resulting in faster data retrieval.

15. Explain the difference between Data Definition Language (DDL), Data Manipulation Language (DML), and Data Control Language (DCL) in SQL. Provide examples of scenarios where DDL commands would be used.

1. Data Definition Language (DDL):
    - DDL commands focus on the structure of the database. They are used to create, modify, and remove the blueprint of your database, including tables, views, indexes, and users.
    - DDL statements typically have a permanent impact on the database schema.

    Examples of DDL commands and their use cases:

    1. CREATE TABLE: This command establishes a new table, defining its structure by specifying column names, data types (e.g., text, integer, date), and constraints (like primary key). You might use this to create a `Customers` table with columns for `Customer ID` (primary key), `Name`, `Email`, and `Phone number`.
    2. ALTER TABLE: This command allows you to modify the structure of an existing table. You could use it to add a new column for `Loyalty Points` to the `Customers` table.
    3. DROP TABLE: This command permanently removes a table from the database. You might use it to remove a table named `Old Orders` that is no longer needed.

2. Data Manipulation Language (DML):

    - DML commands deal with the actual data stored within the database tables. They are used to insert, update, and delete data records.
    - DML statements typically have a temporary impact on the database; changes can be rolled back if necessary.

    Examples of DML commands and their use cases:

    1. NSERT: This command adds new rows of data to a table. You might use this to insert information for a new customer into the `Customers` table.
    2. UPDATE: This command modifies existing data within a table. You could use it to update a customer's email address in the `Customers` table.
    3. DELETE: This command removes rows of data from a table. You might use it to delete a customer record if they request account deletion.

3. Data Control Language (DCL):

    DCL commands manage user permissions and access control within the database. They are used to grant, revoke, and manage privileges for users to interact with the database objects (tables, views, etc.).

    Examples of DCL commands and their use cases:

    1. GRANT: This command assigns specific permissions (e.g., select, insert, update, delete) to users on database objects. You might use this to grant a customer service representative permission to view and update customer information in the `Customers` table.
    2. REVOKE: This command takes away previously granted permissions from users. You could use this to revoke update permissions from a marketing team on the `Customers` table if they no longer need to modify customer data directly.

In essence, DDL defines the database schema, DML manipulates the data within the schema, and DCL controls user access to that data. These categories work together to create a well-structured, secure, and manageable database environment.

TASK DAY -3 SQL
Keywords

1. SELECT: This keyword is used to retrieve data from one or more database tables. It specifies the columns that you want to retrieve in the result set.
   Example:
   SELECT column1, column2 FROM table_name;

2. INSERT: This keyword is used to add new records (rows) into a database table.
   Example:
   INSERT INTO table_name (column1, column2) VALUES (value1, value2);

3. DISTINCT: This keyword is used to return unique values from a query result, eliminating duplicate rows.
   Example:
   SELECT DISTINCT column_name FROM table_name;

4. WHERE: This clause is used to filter records based on specified conditions. It is used in conjunction with the SELECT, UPDATE, DELETE, and similar statements.
   Example:
   SELECT column1, column2 FROM table_name WHERE condition;

5. AND: This operator is used to combine multiple conditions in a WHERE clause. All conditions must be true for the record to be included in the result.
   Example:
   SELECT column1, column2 FROM table_name WHERE condition1 AND condition2;

6. OR: This operator is used to combine multiple conditions in a WHERE clause. At least one of the conditions must be true for the record to be included in the result.
   Example:
   SELECT column1, column2 FROM table_name WHERE condition1 OR condition2;

7. IN: This keyword is used to specify multiple values for a condition in a WHERE clause. It is often used as an alternative to multiple OR conditions.
   Example:
   SELECT column1, column2 FROM table_name WHERE column1 IN (value1, value2, value3);

8. BETWEEN: This keyword is used to specify a range for a condition in a WHERE clause. It selects values within a specified range, including the endpoints.
   Example:

SELECT column1, column2 FROM table_name WHERE column1 BETWEEN value1 AND value2;

SQL Task Day -4 Joins:

Joins are used to combine rows from two or more tables based on a related column between them. There are several types of joins:

1. INNER JOIN: Returns rows when there is at least one match in both tables. SELECT * FROM table1 INNER JOIN table2 ON table1.column = table2.column;

2. LEFT JOIN (or LEFT OUTER JOIN): Returns all rows from the left table and the matched rows from the right table. If there's no match, NULL values are returned for the right table.
   SELECT * FROM table1 LEFT JOIN table2 ON table1.column = table2.column;

3. RIGHT JOIN (or RIGHT OUTER JOIN): Returns all rows from the right table and the matched rows from the left table. If there's no match, NULL values are returned for the left table.
   SELECT * FROM table1 RIGHT JOIN table2 ON table1.column = table2.column;

4. CROSS JOIN: Returns the Cartesian product of the two tables, meaning all possible combinations of rows.
   SELECT * FROM table1 CROSS JOIN table2;

SQL offers various functions for performing operations on data. Some important functions include:

1. COUNT (): Counts the number of rows that match a specified condition.
   SELECT COUNT (*) FROM table_name WHERE condition;

2. SUM (): Calculates the sum of values in a column.
   SELECT SUM (column_name) FROM table_name;

3. AVG (): Calculates the average value of a numeric column.
   SELECT AVG (column_name) FROM table_name;

4. MIN (): Returns the minimum value of a column.
   SELECT MIN (column_name) FROM table_name;

5. MAX (): Returns the maximum value of a column.
   SELECT MAX (column_name) FROM table_name;

6. GROUP BY: Groups rows that have the same values into summary rows.
   SELECT column1, COUNT (*) FROM table_name GROUP BY column1;

7. ORDER BY: Sorts the result set in ascending or descending order based on specified columns.
   SELECT * FROM table_name ORDER BY column1 ASC;

8. WHERE: Filters rows based on specified conditions.
   SELECT * FROM table_name WHERE condition;

9. HAVING: Having clause is similar to the `WHERE` clause, but it is used specifically with aggregate functions when you want to filter grouped rows returned by a `GROUP BY` clause.
   Example:
   SELECT column1, COUNT (*)
   FROM table_name
   GROUP BY column1
   HAVING COUNT (*) > 2;

10. DATE (): Returns the date part of a datetime expression.
    SELECT DATE (datetime_column) FROM table_name;

11. NOW (): Returns the current date and time.
    SELECT NOW ();

12. YEAR (): Extracts the year from a date.
    SELECT YEAR (date_column) FROM table_name;

13. CONCAT (): Concatenates two or more strings together.
    SELECT CONCAT (first_name, ' ', last_name) AS full_name FROM table_name;

14. UPPER (): Converts a string to uppercase.
    SELECT UPPER (column_name) FROM table_name;

15. LOWER (): Converts a string to lowercase.
    SELECT LOWER (column_name) FROM table_name;

16. SUBSTRING (): Extracts a substring from a string.
    SELECT SUBSTRING (column_name, start_index, length) FROM table_name;

17. ROUND (): Rounds a numeric value to a specified number of decimal places.
    SELECT ROUND (column_name, decimals) FROM table_name;

18. ABS (): Returns the absolute value of a numeric expression.
    SELECT ABS (column_name) FROM table_name;

19. CEILING (): Returns the smallest integer greater than or equal to a numeric expression.
    SELECT CEILING (column_name) FROM table_name;

20. FLOOR (): Returns the largest integer less than or equal to a numeric expression.
    SELECT FLOOR (column_name) FROM table_name;