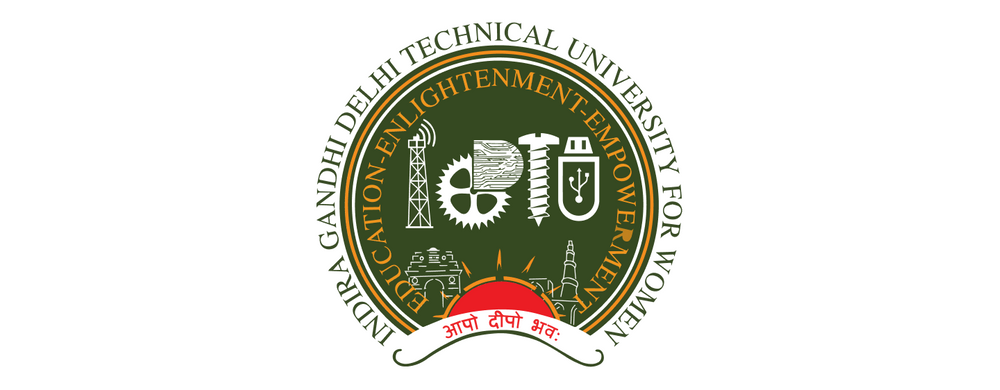
**INDIRA GANDHI DELHI TECHNICAL UNIVERSITY FOR WOMEN**



**DBMS PRACTICAL FILE**

Semester 3

**Submitted By:**

Name: Ananya Sharma

B.Tech CSE-1

Enrolment No.: 02301012023

**Submitted To:**

Prof. D.K. Tayal

**INDEX**

|  |  |  |
| --- | --- | --- |
| **S.no** | **Topic** | **Page.No** |
| 1 | Introduction to DBMS | 3-5 |
| 2 | Introduction to ER Diagram Components | 6-8 |
| 3 | Introduction to DDL, DML, and DCL | 9-10 |
| 4 | Introduction to SQL and MySQL | 11 |
| 5 | SQL Data Types | 12-14 |
| 6 | Attribute Constraints | 15-17 |
| 7 | SQL Queries (Real Life Example) | 18 |
| 8 | Queries to create a table | 19-21 |
| 9 | Queries to edit a table | 22-23 |
| 10 | Queries to insert data | 24-26 |
| 11 | Queries to retrieve data | 27-30 |
| 12 | Aggregate functions | 31-33 |
| 13 | Queries for Dates | 34-38 |

**Introduction to DBMS**

**(Database Management System)**

A **Database Management System (DBMS)** is a software system that provides an interface for creating, managing, and interacting with databases. It is designed to handle the storage, retrieval, and manipulation of data efficiently. The key features of a DBMS include:

1. **Data Storage Management**: DBMS stores data in an organized way using tables, rows, and columns. It ensures efficient data retrieval and storage.
2. **Data Security**: It provides mechanisms to protect data from unauthorized access through user authentication and permissions.
3. **Data Integrity**: Ensures that the data is accurate, consistent, and reliable by enforcing constraints like primary keys, foreign keys, etc.
4. **Concurrency Control**: DBMS manages simultaneous data access by multiple users to ensure that transactions are processed correctly.
5. **Backup and Recovery**: It provides features for data backup and restoration in case of system failure.
6. **Data Independence**: DBMS abstracts the physical storage of data from the user, providing a logical view of the data.
7. **Query Processing**: DBMS enables querying and reporting through languages like SQL, making it easier to retrieve and manipulate data.

**Examples of DBMSs**:

* Oracle DB
* MySQL
* Microsoft SQL Server
* PostgreSQL
* MongoDB (for NoSQL databases)

**File System vs. DBMS**

A **file system** and a **DBMS** both store data, but they differ significantly in their design, structure, and functionality. Below is a comparison between the two:

|  |  |  |
| --- | --- | --- |
| **Aspect** | **File System** | **DBMS** |
| **Data Storage** | Stores data in files (e.g., text, binary, etc.). | Stores data in tables with rows and columns (structured data). |
| **Data Organization** | Data is stored sequentially, often without structure. | Data is organized using schemas, tables, rows, and columns. |
| **Data Access** | Data access is through file manipulation operations (read, write, delete). | Access is done using high-level query languages like SQL. |
| **Concurrency** | No inherent mechanisms for concurrent access. | Supports concurrent access and transactions, ensuring data consistency. |
| **Data Integrity** | Limited support for data integrity. Data may be inconsistent or redundant. | Ensures data integrity through constraints (e.g., primary key, foreign key). |
| **Security** | Basic file system permissions. | Advanced security features like role-based access control, encryption, and auditing. |
| **Backup and Recovery** | Manual process to back up files. | Automatic and efficient backup and recovery mechanisms. |
| **Data Redundancy** | High potential for data duplication. | Reduces redundancy through normalization and referential integrity. |
| **Performance** | Typically faster for simple file access. | Optimized for complex queries and large datasets, though sometimes slower for simple tasks. |
| **Scalability** | May become inefficient with large datasets or complex data relationships. | Designed to efficiently manage large amounts of data with complex relationships. |
| **Usage** | Suitable for simple applications with small datasets. | Suitable for enterprise-level applications with complex data needs and multi-user access. |

**Key Differences:**

1. **Data Structure**:
   * A file system stores data in a flat or hierarchical structure, without any predefined schema.
   * A DBMS stores data in a structured format (tables, rows, columns) with relationships defined between them.
2. **Efficiency in Data Management**:
   * File systems can become inefficient as the size and complexity of data grow, with no built-in features for querying or handling large amounts of data.
   * DBMS is optimized for managing large datasets and allows for fast, complex querying, sorting, and indexing.
3. **Security and Integrity**:
   * File systems provide basic security (file permissions) but have no mechanism for enforcing data consistency.
   * DBMS ensures data integrity through features like transactions, foreign keys, and constraints.
4. **Backup and Recovery**:
   * File systems require manual backup and recovery procedures.
   * DBMS provides automatic backup, logging, and recovery features.
5. **Concurrency Control**:
   * File systems don’t manage concurrent access, which can lead to data corruption if multiple users try to access or modify files simultaneously.
   * DBMS offers concurrency control to ensure that data remains consistent even with multiple simultaneous accesses.

In summary, while a **file system** is suitable for simple data storage, a **DBMS** offers a much more powerful and structured way to manage, access, and secure data, especially in environments that involve large datasets, multiple users, and complex queries.

**Introduction to ER Diagram Components**

An **Entity-Relationship (ER) Diagram** is a visual representation of the data model that shows the relationships between different entities in a database. It is a key tool used in database design, providing an abstract view of how data entities interact with each other.

**Key Components of an ER Diagram:**

1. **Entities**:
   * **Definition**: An entity represents a real-world object or concept that has data stored about it. Entities are typically nouns, such as *Customer*, *Employee*, or *Product*.
   * **Representation**: In an ER diagram, entities are represented by **rectangles**.
2. **Attributes**:
   * **Definition**: Attributes provide more detail about an entity. They describe the properties or characteristics of an entity.
   * **Representation**: Attributes are depicted as **ovals** connected to their respective entities.
     + **Simple Attribute**: A single-valued attribute (e.g., Name).
     + **Composite Attribute**: An attribute that can be divided into smaller subparts (e.g., Address can have Street, City, State).
     + **Derived Attribute**: An attribute that is derived from another attribute (e.g., Age derived from Date of Birth).
3. **Relationships**:
   * **Definition**: Relationships represent associations between two or more entities.
   * **Representation**: Relationships are depicted as **diamonds** or rhombuses. Each relationship connects two or more entities.
4. **Primary Key**:
   * **Definition**: The primary key is a unique identifier for each record within an entity.
   * **Representation**: Usually underlined in the entity box.
5. **Weak Entities**:
   * **Definition**: A weak entity cannot exist without being associated with a strong entity. It depends on a strong entity for identification.
   * **Representation**: Weak entities are shown using a **double rectangle**.
6. **Identifying Relationship**:
   * **Definition**: A relationship that identifies a weak entity.
   * **Representation**: Depicted as a **double diamond**.

**Cardinality and Participation in ER Diagrams**

**Cardinality** and **participation** are important concepts used to define the relationship between entities in an ER diagram. They describe how entities relate to one another and the constraints of their interactions.

**1. Cardinality:**

Cardinality defines the number of instances of one entity that can be associated with an instance of another entity. Cardinality can be one-to-one (1:1), one-to-many (1:N), or many-to-many (M:N).

**One-to-One (1:1)**: One entity is related to only one instance of another entity. For example, a *Person* has only one *Passport*, and a *Passport* is only linked to one *Person*.

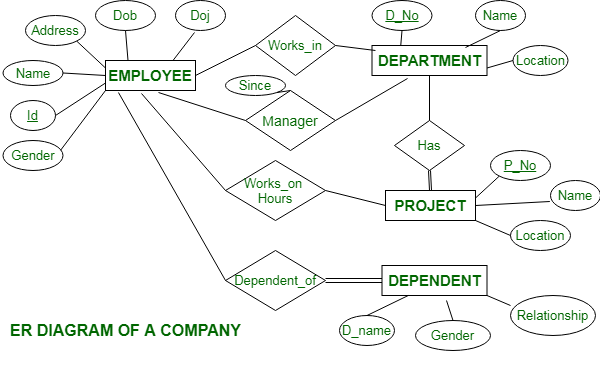
* **One-to-Many (1:N)**: One entity is related to many instances of another entity, but each instance of the second entity is related to only one instance of the first entity. For example, a *Department* can have many *Employees*, but each *Employee* works in only one *Department*.
* **Many-to-Many (M:N)**: Many instances of one entity can be related to many instances of another entity. For example, a *Student* can enroll in many *Courses*, and a *Course* can have many *Students*.

**2. Participation:**

Participation specifies whether all or only some instances of an entity participate in a relationship.

* **Total Participation**: All instances of an entity participate in the relationship. Represented by a **double line** between the entity and the relationship.
  + Example: Every *Employee* must be assigned to a *Department*.
* **Partial Participation**: Some instances of an entity may not participate in the relationship. Represented by a **single line** between the entity and the relationship.
  + Example: A *Student* may or may not be enrolled in a *Course*.

**ER Diagram Example**



ER diagram of the Company has the following description :

* Company has several departments.
* Each department may have several Locations.
* Departments are identified by a name, d\_no, Location.
* A Manager control a particular department.
* Each department is associated with number of projects.
* Employees are identified by name, id, address, job, date\_of\_joining.
* An employee works in only one department but can work on several project.
* We also keep track of number of hours worked by an employee on a single project.
* Each employee has a dependent
* Dependent has D\_name, Gender, and relationship.

**Entities and their Attributes are:**

* **Employee Entity :** Attributes of Employee Entity are Name, Id, Address, Gender, Dob and Doj. Id is Primary Key for Employee Entity.
* **Department Entity :** Attributes of Department Entity are D\_no, Name and Location. D\_no is Primary Key for Department Entity.
* **Project Entity :** Attributes of Project Entity are P\_No, Name and Location. P\_No is Primary Key for Project Entity.
* **Dependent Entity :** Attributes of Dependent Entity are D\_no, Gender and relationship.

**Relationships are :**

* **Employees works in Departments –** Many employee works in one Department but one employee can not work in many departments.
* **Manager controls a Department –** employee works under the manager of the Department and the manager records the date of joining of employee in the department.
* **Department has many Projects –** One department has many projects but one project can not come under many departments.
* **Employee works on project –** One employee works on several projects and the number of hours worked by the employee on a single project is recorded.
* **Employee has dependents –** Each Employee has dependents. Each dependent is dependent of only one employee.

**Introduction to DDL, DML, and DCL**

In the context of **Database Management Systems (DBMS)**, SQL (Structured Query Language) is the standard language used for managing and manipulating relational databases. SQL is divided into several categories, each serving a specific purpose. The three main categories are:

1. **DDL (Data Definition Language)**
2. **DML (Data Manipulation Language)**
3. **DCL (Data Control Language)**

Each of these plays a key role in managing databases, with distinct functions related to database structure, data manipulation, and user permissions.

**1. DDL (Data Definition Language)**

**Definition**: DDL refers to the set of SQL commands used to define the database structure. These commands deal with the creation, alteration, and deletion of database objects like tables, views, schemas, and indexes.

**Key DDL Commands:**

* **CREATE**: Used to create a new database object (like a table, view, or index).
  + Example:
  + CREATE TABLE Employees (
  + EmployeeID INT PRIMARY KEY,
  + Name VARCHAR(50),
  + DepartmentID INT
  + );
* **ALTER**: Used to modify the structure of an existing database object (e.g., adding or dropping columns from a table).
  + Example:
  + ALTER TABLE Employees ADD Email VARCHAR(100);
* **DROP**: Used to delete an existing database object (like a table or a view).
  + Example:
  + DROP TABLE Employees;
* **TRUNCATE**: Used to delete all rows in a table but keep the structure intact (doesn't log individual row deletions).
  + Example:
  + TRUNCATE TABLE Employees;
* **RENAME**: Used to rename a database object.
  + Example:
  + RENAME TABLE Employees TO Staff;

**2. DML (Data Manipulation Language)**

**Definition**: DML refers to SQL commands that are used for managing and manipulating the actual data within the database. It allows you to perform operations such as inserting, updating, deleting, and retrieving data.

**Key DML Commands:**

* **SELECT**: Retrieves data from one or more tables.
  + Example:
  + SELECT Name, DepartmentID FROM Employees WHERE DepartmentID = 1;
* **INSERT**: Adds new rows of data into a table.
  + Example:
  + INSERT INTO Employees (EmployeeID, Name, DepartmentID) VALUES (1, 'John Doe', 2);
* **UPDATE**: Modifies existing data in a table.
  + Example:
  + UPDATE Employees SET DepartmentID = 3 WHERE EmployeeID = 1;
* **DELETE**: Removes data from a table.
  + Example:
  + DELETE FROM Employees WHERE EmployeeID = 1;

**Note**: DML operations typically affect the data in the database, but the structure of the database (tables, schemas) remains unchanged.

**3. DCL (Data Control Language)**

**Definition**: DCL refers to the SQL commands that control access to the data and database objects. It deals with the permissions and rights of users in relation to the database, ensuring that only authorized users can access, modify, or delete data.

**Key DCL Commands:**

* **GRANT**: Gives specific privileges to users or roles, such as SELECT, INSERT, UPDATE, DELETE, etc.
  + Example:
  + GRANT SELECT, INSERT ON Employees TO User1;
* **REVOKE**: Removes specific privileges from users or roles.
  + Example:
  + REVOKE SELECT ON Employees FROM User1;

**Note**: DCL commands help ensure security and integrity within the database by managing who can perform which operations.

**Summary of DDL, DML, and DCL**

|  |  |  |
| --- | --- | --- |
| **Category** | **Purpose** | **Examples of Commands** |
| **DDL** | Defines the structure of the database and its objects. | CREATE, ALTER, DROP,   TRUNCATE, RENAME |
| **DML** | Manipulates the actual data in the database. | SELECT, INSERT,   UPDATE, DELETE |
| **DCL** | Controls user access and privileges to the database. | GRANT, REVOKE |

These three categories form the foundation of SQL and are essential for designing, populating, managing, and securing databases effectively.

**Introduction to SQL and MySQL**

**SQL (Structured Query Language):**

SQL is a standardized programming language used to interact with relational databases. It allows users to perform various operations like creating, modifying, and querying databases. SQL is essential for managing data efficiently and is widely adopted across industries.

**MySQL:**

MySQL is an open-source relational database management system (RDBMS) that uses SQL as its primary language. Developed by Oracle Corporation, it is known for its speed, reliability, and ease of use. MySQL is widely used for web applications, data warehousing, and e-commerce platforms.

|  |  |  |
| --- | --- | --- |
| **Aspect** | **SQL** | **MySQL** |
| **Definition** | A language for managing relational databases. | A database management system that uses SQL. |
| **Functionality** | Provides commands to manage and manipulate data. | Manages and stores data in relational form. |
| **Purpose** | Acts as a querying tool for databases. | Implements SQL to perform database operations. |
| **Type** | A programming language, not a software. | A software or application. |
| **Usage** | Universal for all relational database management systems (RDBMS). | Specific to the MySQL RDBMS. |
| **Cost** | No cost for SQL as a language. | Open-source with enterprise versions available for purchase. |
| **Developer** | Standardized by ANSI/ISO. | Developed by Oracle Corporation. |
| **Support for Programming** | Works as part of any RDBMS. | Integrates SQL with programming interfaces. |

**SQL Data Types**

SQL data types define the type of data a column can hold in a database. These data types help ensure data consistency, optimize storage, and improve query performance.:

**1. Numeric Data Types**

These data types store numerical values, including whole numbers, decimals, and floating-point numbers.

**Types:**

* **INT (Integer)**: Stores whole numbers.
  + Example: Age INT → 25
* **SMALLINT**: A smaller range of integers.
  + Example: Age SMALLINT → 22
* **BIGINT**: Used for very large integers.
  + Example: Salary BIGINT → 1000000000
* **DECIMAL(p, s)**: Stores fixed-point numbers where p is the total number of digits and s is the number of digits to the right of the decimal point.
  + Example: Price DECIMAL(10, 2) → 99.99
* **FLOAT**: Used to store approximate floating-point numbers.
  + Example: Weight FLOAT → 72.56
* **DOUBLE**: Stores larger floating-point numbers than FLOAT.
  + Example: Temperature DOUBLE → 36.987654

**Difference:**

* **INT vs SMALLINT**: INT stores larger numbers than SMALLINT.
* **FLOAT vs DECIMAL**: FLOAT is for approximate numbers, while DECIMAL is for precise fixed-point numbers.

**2. Character/String Data Types**

These data types store text or alphanumeric data.

**Types:**

* **CHAR(n)**: Fixed-length string, where n is the number of characters. If the string is shorter than n, it’s padded with spaces.
  + Example: Code CHAR(5) → 'AB123'
* **VARCHAR(n)**: Variable-length string, where n is the maximum number of characters.
  + Example: Name VARCHAR(50) → 'John Doe'
* **TEXT**: Stores large amounts of text data, with no specific length limit.
  + Example: Description TEXT → 'This is a description of a product.'

**Difference:**

* **CHAR vs VARCHAR**: CHAR is for fixed-length data, while VARCHAR is more flexible, storing variable-length strings.
* **TEXT**: Can store much larger text than VARCHAR.

**3. Date and Time Data Types**

These data types are used for storing date, time, and timestamp values.

**Types:**

* **DATE**: Stores the date in YYYY-MM-DD format.
  + Example: BirthDate DATE → 1995-10-15
* **TIME**: Stores the time in HH:MM:SS format.
  + Example: StartTime TIME → 14:30:00
* **DATETIME**: Stores both date and time in YYYY-MM-DD HH:MM:SS format.
  + Example: Event DATETIME → 2024-11-22 18:45:00
* **TIMESTAMP**: Similar to DATETIME but stores data as Unix time (epoch). Automatically updates when a record is modified.
  + Example: LastModified TIMESTAMP → 2024-11-22 14:30:00

**Difference:**

* **DATE vs DATETIME**: DATE only stores the date, while DATETIME stores both date and time.
* **TIMESTAMP** automatically updates when a record is modified, unlike DATETIME.

**4. Binary Data Types**

These data types store binary data like images, files, or multimedia content.

**Types:**

* **BINARY(n)**: Fixed-length binary data. It stores data as raw bytes.
  + Example: Key BINARY(16) → 0xA1B2C3D4E5F6G7H8
* **VARBINARY(n)**: Variable-length binary data.
  + Example: File VARBINARY(255) → 0x9F8E7D6C5B4A
* **BLOB (Binary Large Object)**: Used for large binary data like images or video files.
  + Example: Image BLOB → Stores image data in binary format.

**Difference:**

* **BINARY vs VARBINARY**: BINARY stores fixed-length binary data, whereas VARBINARY stores variable-length binary data.
* **BLOB** is for large binary data, unlike VARBINARY.

**5. Boolean Data Types**

These data types store logical values, either TRUE or FALSE.

**Types:**

* **BOOLEAN or BOOL**: Stores TRUE or FALSE values.
  + Example: IsActive BOOLEAN → TRUE

**6. Spatial Data Types**

These data types store geographical or spatial data such as points, lines, and polygons.

**Types:**

* **GEOMETRY**: A general spatial data type that stores geometric data.
  + Example: Location GEOMETRY → Represents a geometric point or shape.
* **POINT**: Stores a single point in space (x, y coordinates).
  + Example: Coordinates POINT → POINT(10 20)
* **POLYGON**: Stores a polygon defined by a set of points.
  + Example: Area POLYGON → POLYGON((10 20, 30 40, 50 60, 10 20))

**Difference:**

* **GEOMETRY vs POINT**: GEOMETRY can store any geometric shape, while POINT is specifically for a single location.

**7. JSON Data Type**

Used to store data in the JSON (JavaScript Object Notation) format, widely used for semi-structured data.

**Types:**

* **JSON**: Stores data in key-value pairs, providing a flexible format.
  + Example: Details JSON → '{"name": "John", "age": 30}'

**Difference:**

* **JSON** allows semi-structured data, making it more flexible than traditional relational types like VARCHAR.

**Choosing the Right Data Type**

1. **For numbers**: Use INT, DECIMAL, or FLOAT based on the precision and range required.
2. **For text**: Use VARCHAR for variable-length text or TEXT for larger content.
3. **For dates and times**: Use DATE, TIME, or DATETIME depending on your requirements.
4. **For binary data**: Use BLOB or VARBINARY.
5. **For JSON data**: Use JSON for structured, semi-flexible storage.

**Attribute Constraints**

In MySQL, constraints are rules enforced on data in tables to ensure data integrity, consistency, and accuracy.

**1. NOT NULL**

Ensures that a column cannot have a NULL value.

Use case: To ensure a field always has a value.

Example:

CREATE TABLE students (

id INT NOT NULL,

name VARCHAR(100) NOT NULL

);

**2. UNIQUE**

Ensures that all values in a column are unique, i.e., no duplicate values are allowed.

Use case: To enforce uniqueness in fields like email or username.

Example:

CREATE TABLE users (

id INT NOT NULL,

email VARCHAR(255) UNIQUE

);

**3. PRIMARY KEY**

Combines NOT NULL and UNIQUE. It uniquely identifies each record in a table.

A table can have only one primary key, but it can consist of one or more columns (composite key).

Example:

CREATE TABLE orders (

order\_id INT PRIMARY KEY,

order\_date DATE

);

**4. FOREIGN KEY**

Establishes a relationship between two tables by linking the foreign key in one table to the primary key in another table.

Use case: To maintain referential integrity between tables.

Example:

CREATE TABLE orders (

order\_id INT PRIMARY KEY,

customer\_id INT,

FOREIGN KEY (customer\_id) REFERENCES customers(id)

);

**5. CHECK**

Ensures that all values in a column satisfy a specific condition.

Example:

CREATE TABLE employees (

id INT,

salary DECIMAL(10, 2),

CHECK (salary > 0)

);

**6. DEFAULT**

Provides a default value for a column if no value is specified.

Use case: To set a fallback or standard value for columns.

Example:

CREATE TABLE products (

id INT,

name VARCHAR(100),

status VARCHAR(20) DEFAULT 'available'

);

**7. AUTO\_INCREMENT**

Automatically generates a unique number for each new row in a table, typically used with a primary key.

Use case: To auto-generate IDs.

Example:

CREATE TABLE employees (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100)

);

**8. INDEX**

Speeds up query execution by creating an index on one or more columns.

Use case: To optimize searches.

Example:

CREATE INDEX idx\_name ON students(name);

**9. UNIQUE KEY with NULL**

Unlike PRIMARY KEY, a UNIQUE constraint allows one NULL value unless explicitly disallowed.

**Combined Example:**

CREATE TABLE accounts (

account\_id INT AUTO\_INCREMENT PRIMARY KEY,

username VARCHAR(50) NOT NULL UNIQUE,

email VARCHAR(100) UNIQUE,

balance DECIMAL(10, 2) DEFAULT 0 CHECK (balance >= 0),

customer\_id INT,

FOREIGN KEY (customer\_id) REFERENCES customers(id)

);

**SQL Queries**

(Library Management System Database)

**Books Table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **BookID** | **Title** | **Author** | **Genre** | **PublishedYear** |
| 1 | The Great Gatsby | F. Scott Fitzgerald | Fiction | 1925 |
| 2 | 1984 | George Orwell | Dystopian | 1949 |
| 3 | To Kill a Mockingbird | Harper Lee | Fiction | 1960 |
| 4 | The Catcher in the Rye | J.D. Salinger | Fiction | 1951 |
| 5 | The Hobbit | J.R.R. Tolkien | Fantasy | 1937 |
| 6 | Pride and Prejudice | Jane Austen | Romance | 1813 |

**Members Table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **MemberID** | **Name** | **Email** | **Phone** | **JoinDate** |
| 1 | Alice | alice@example.com | 123-456-7890 | 2022-05-15 |
| 2 | Bob | bob@example.com | 987-654-3210 | 2023-06-01 |
| 3 | Charlie | charlie@example.com | 555-123-4567 | 2024-03-20 |
| 4 | Diana | diana@example.com | 555-234-5678 | 2024-07-12 |
| 5 | Edward | edward@example.com | 555-345-6789 | 2023-08-05 |

**Borrowing Table (Sample Data)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **BorrowID** | **BookID** | **MemberID** | **BorrowDate** | **ReturnDate** |
| 1 | 1 | 1 | 2024-01-10 | 2024-01-17 |
| 2 | 2 | 2 | 2024-02-05 | 2024-02-20 |
| 3 | 3 | 1 | 2024-03-01 | NULL |
| 4 | 4 | 3 | 2024-04-15 | 2024-04-25 |
| 5 | 5 | 2 | 2024-05-01 | NULL |
| 6 | 6 | 4 | 2024-07-01 | NULL |
| 7 | 1 | 5 | 2024-07-10 | 2024-07-15 |
| 8 | 2 | 5 | 2024-07-15 | 2024-07-20 |

**SQL Queries to Create Table**

**I. CREATE COMMAND:**

* **Definition :**The CREATE TABLE statement is used to define a new table in the database. It specifies the table name, columns, their data types, and constraints.

**1. General Syntax for CREATE TABLE**

CREATE TABLE TableName (

ColumnName DataType [Constraints],

ColumnName DataType [Constraints],

...

);

**Key Components**:

* **TableName**: The name of the table (e.g., Books, Members).
* **ColumnName**: The name of the column (e.g., BookID, Title).
* **DataType**: The type of data the column will store (e.g., INT, VARCHAR, DATE).
* **Constraints**: Rules applied to the columns (e.g., PRIMARY KEY, NOT NULL, FOREIGN KEY).

**2. Creating Tables for the Library Management System**

**(A) Books Table**

**Query**:

CREATE TABLE Books (

BookID INT AUTO\_INCREMENT PRIMARY KEY,

Title VARCHAR(255) NOT NULL,

Author VARCHAR(255) NOT NULL,

Genre VARCHAR(100),

PublishedYear YEAR

);

**Explanation**:

* BookID: Auto-incremented unique identifier, set as the primary key.
* Title and Author: Mandatory fields (NOT NULL).
* Genre: Optional field.
* PublishedYear: Stores the year the book was published.

**OUTPUT:**

+---------------+--------------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+---------------+--------------+------+-----+---------+----------------+

| BookID | int | NO | PRI | NULL | auto\_increment |

| Title | varchar(255) | NO | | NULL | |

| Author | varchar(255) | NO | | NULL | |

| Genre | varchar(100) | YES | | NULL | |

| PublishedYear | year | YES | | NULL | |

+---------------+--------------+------+-----+---------+----------------+

**(B) Members Table**

**Query**:

CREATE TABLE Members (

MemberID INT AUTO\_INCREMENT PRIMARY KEY,

Name VARCHAR(255) NOT NULL,

Email VARCHAR(255) UNIQUE NOT NULL,

Phone VARCHAR(15),

JoinDate DATE NOT NULL

);

**Explanation**:

* MemberID: Auto-incremented primary key.
* Email: Must be unique (UNIQUE constraint).
* JoinDate: Mandatory field.

**Output (Schema)**:

+-----------+--------------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+-----------+--------------+------+-----+---------+----------------+

| MemberID | int | NO | PRI | NULL | auto\_increment |

| Name | varchar(255) | NO | | NULL | |

| Email | varchar(255) | NO | UNI | NULL | |

| Phone | varchar(15) | YES | | NULL | |

| JoinDate | date | NO | | NULL | |

+-----------+--------------+------+-----+---------+----------------+

**(C) Borrowing Table**

**Query**:

CREATE TABLE Borrowing (

BorrowID INT AUTO\_INCREMENT PRIMARY KEY,

BookID INT NOT NULL,

MemberID INT NOT NULL,

BorrowDate DATE NOT NULL,

ReturnDate DATE,

FOREIGN KEY (BookID) REFERENCES Books(BookID),

FOREIGN KEY (MemberID) REFERENCES Members(MemberID)

);

**Explanation**:

* BorrowID: Auto-incremented primary key.
* BookID and MemberID: Foreign keys referencing Books and Members respectively.
* BorrowDate: Mandatory field.

**Output (Schema)**:

+-------------+-------------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+-------------+-------------+------+-----+---------+----------------+

| BorrowID | int | NO | PRI | NULL | auto\_increment |

| BookID | int | NO | MUL | NULL | |

| MemberID | int | NO | MUL | NULL | |

| BorrowDate | date | NO | | NULL | |

| ReturnDate | date | YES | | NULL | |

+-------------+-------------+------+-----+---------+----------------+

**SQL Queries to Edit a Table**

**ALTER COMMAND:**

**1. Definition of ALTER Command**

The **ALTER** command is used to modify the structure of an existing table. This can include:

* Adding new columns.
* Dropping (removing) existing columns.
* Modifying column definitions (e.g., data type, constraints).
* Renaming tables or columns.

**2. Adding a Column**

**(A) Definition:**

The **ADD** operation in the ALTER command is used to add a new column to an existing table.

**(B) Syntax:**

ALTER TABLE TableName

ADD ColumnName DataType [Constraints];

**(C) Example:**

Suppose we want to add a ISBN column to the Books table.

ALTER TABLE Books

ADD ISBN VARCHAR(13) UNIQUE;

**(D) Explanation:**

* Adds a new column ISBN of type VARCHAR(13) with a UNIQUE constraint.
* ISBN is a unique identifier for books.

**(E) Output (Schema):**

+---------------+--------------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+---------------+--------------+------+-----+---------+----------------+

| BookID | int | NO | PRI | NULL | auto\_increment |

| Title | varchar(255) | NO | | NULL | |

| Author | varchar(255) | NO | | NULL | |

| Genre | varchar(100) | YES | | NULL | |

| PublishedYear | year | YES | | NULL | |

| ISBN | varchar(13) | YES | UNI | NULL | |

+---------------+--------------+------+-----+---------+----------------+

**3. Dropping a Column**

**(A) Definition:**

The **DROP** operation in the ALTER command is used to remove an existing column from a table.

**(B) Syntax:**

ALTER TABLE TableName

DROP COLUMN ColumnName;

**(C) Example:**

Suppose we want to remove the Genre column from the Books table.

ALTER TABLE Books

DROP COLUMN Genre;

**(D) Explanation:**

* Removes the Genre column from the Books table.

**(E) Output (Schema):**

DESCRIBE Books;

+---------------+--------------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+---------------+--------------+------+-----+---------+----------------+

| BookID | int | NO | PRI | NULL | auto\_increment |

| Title | varchar(255) | NO | | NULL | |

| Author | varchar(255) | NO | | NULL | |

| PublishedYear | year | YES | | NULL | |

| ISBN | varchar(13) | YES | UNI | NULL | |

+---------------+--------------+------+-----+---------+----------------+

**SQL Queries to Insert Data**

**INSERT, UPDATE, DELETE:**

**INSERT Command**

**(A) Definition:**

The INSERT command is used to add new rows (records) into a table.

**(B) Syntax:**

INSERT INTO TableName (Column1, Column2, ...)

VALUES (Value1, Value2, ...);

**(C) Example:**

Add a new book to the Books table:

INSERT INTO Books (Title, Author, Genre, PublishedYear, ISBN)

VALUES ('The Great Gatsby', 'F. Scott Fitzgerald', 'Fiction', 1925, '9780743273565');

**(D) Explanation:**

* Adds a record with the specified details for Title, Author, Genre, PublishedYear, and ISBN.
* BookID is auto-incremented and does not need to be included.

**(E) Output (Data):**

SELECT \* FROM Books;

+--------+-----------+---------+----------+---------------+---------------+

| BookID | Title | Author | Genre | PublishedYear | ISBN |

+--------+------------+--------+----------+---------------+---------------+

| 1 | The Great Gatsby | F. Scott Fitzgerald | Fiction | 1925 | 9780743273565 |

+--------+-------------------+---------------------+----------+---------------+---------------+

**2. UPDATE Command**

**(A) Definition:**

The UPDATE command modifies existing records in a table.

**(B) Syntax:**

UPDATE TableName

SET Column1 = Value1, Column2 = Value2, ...

WHERE Condition;

**(C) Example:**

Update the Genre of a book with BookID = 1:

UPDATE Books

SET Genre = 'Classic'

WHERE BookID = 1;

**(D) Explanation:**

* Updates the Genre column to Classic for the book with BookID = 1.
* The WHERE clause specifies which row(s) to update. Without it, all rows will be updated.

**(E) Output (Data):**

SELECT \* FROM Books WHERE BookID = 1;

+--------+-------------------+---------------------+----------+---------------+---------------+

| BookID | Title | Author | Genre | PublishedYear | ISBN |

+--------+-------------------+---------------------+----------+---------------+---------------+

| 1 | The Great Gatsby | F. Scott Fitzgerald | Classic | 1925 | 9780743273565 |

+--------+-------------------+---------------------+----------+---------------+---------------+

**3. DELETE Command**

**(A) Definition:**

The DELETE command removes rows (records) from a table.

**(B) Syntax:**

DELETE FROM TableName

WHERE Condition;

**(C) Example:**

Delete the book with BookID = 1 from the Books table:

DELETE FROM Books

WHERE BookID = 1;

**(D) Explanation:**

* Deletes the record where BookID = 1.
* The WHERE clause specifies the condition for deletion. Without it, all rows in the table will be deleted.

**(E) Output (Data):**

SELECT \* FROM Books;

+--------+-------+--------+-------+---------------+------+

| BookID | Title | Author | Genre | PublishedYear | ISBN |

+--------+-------+--------+-------+---------------+------+

| (No rows) |

+--------+-------+--------+-------+---------------+------+

**SQL Queries to Retrieve Data**

**1. SELECT**

* **Purpose**: Used to retrieve data from a database.
* **Syntax**:

SELECT column1, column2

FROM table\_name;

* **Notes**:
  + Can be used to retrieve all columns.
  + Supports aliases with AS for renaming columns in the result.

**2. FROM**

* **Purpose**: Specifies the table(s) from which to retrieve data.
* **Syntax**:

SELECT column1

FROM table\_name;

* **Notes**: Often used in combination with JOIN for queries involving multiple tables.

**3. WHERE**

* **Purpose**: Filters rows based on specified conditions.
* **Syntax**:

SELECT column1

FROM table\_name

WHERE condition;

* **Operators**:
  + Comparison: =, !=, <, >, <=, >=
  + Logical: AND, OR, NOT
  + Pattern Matching: LIKE, % (wildcard for zero or more characters), \_ (wildcard for one character)
  + Range: BETWEEN x AND y
  + Null Check: IS NULL, IS NOT NULL

**4. ORDER BY**

* **Purpose**: Sorts the result set in ascending (ASC) or descending (DESC) order.
* **Syntax**:

SELECT column1

FROM table\_name

ORDER BY column1 ASC;

**5. GROUP BY**

* Purpose: Groups rows with the same values in specified columns and allows aggregation functions to be applied to each group.
* Syntax:

SELECT column1, COUNT(\*)

FROM table\_name

GROUP BY column1;

* Notes: Often used with aggregate functions like COUNT(), SUM(), AVG(), MIN(), and MAX().

**6. HAVING**

* Purpose: Filters grouped records (used after GROUP BY).
* Syntax:

SELECT column1, SUM(column2)

FROM table\_name

GROUP BY column1

HAVING SUM(column2) > 100;

**7. JOIN**

* Purpose: Combines rows from two or more tables based on a related column.
* Types:
  + **INNER JOIN**: Returns records with matching values in both tables.
  + **LEFT JOIN (OUTER JOIN)**: Returns all records from the left table and matching records from the right table (NULL if no match).
  + **RIGHT JOIN (OUTER JOIN)**: Returns all records from the right table and matching records from the left table.
  + **FULL OUTER JOIN**: Returns all records when there is a match in either table.

SELECT column1

FROM table1

INNER JOIN table2 ON table1.common\_column = table2.common\_column;

**8. LIMIT**

* Purpose: Restricts the number of rows returned.
* Syntax:

SELECT column1

FROM table\_name

LIMIT 10;

**9. DISTINCT**

* Purpose: Eliminates duplicate rows in the result set.
* Syntax:

SELECT DISTINCT column1

FROM table\_name;

**10. UNION**

* Purpose: Combines the result sets of two or more SELECT queries and removes duplicates.
* Syntax:

SELECT column1 FROM table1

UNION

SELECT column1 FROM table2;

**11. NESTED QUERY**

* Purpose: A query nested within another query to provide intermediate results.
* Syntax:

SELECT column1

FROM table\_name

WHERE column2 = (SELECT MAX(column2) FROM table\_name);

**Example:**

We want to find the top 3 books that have been borrowed the most, along with the member details of the person who borrowed them. We also want to filter the results to only show books borrowed after January 1, 2023. The result should include the book's title, the member's name, and the total number of times each book was borrowed, sorted in descending order of the number of times borrowed.

SELECT

Books.Title,

Members.Name AS MemberName,

COUNT(Borrowing.BorrowID) AS BorrowCount

FROM

Borrowing

INNER JOIN

Books ON Borrowing.BookID = Books.BookID

INNER JOIN

Members ON Borrowing.MemberID = Members.MemberID

WHERE

Borrowing.BorrowDate >= '2023-01-01'

GROUP BY

Books.BookID, Members.MemberID

HAVING

COUNT(Borrowing.BorrowID) > 1

ORDER BY

BorrowCount DESC

LIMIT 3;

**Explanation of SQL Query:**

1. SELECT: We are selecting the Title of the book from the Books table, the Name of the member (renamed as MemberName), and the COUNT() of the BorrowID to count how many times a specific book was borrowed.
2. FROM: The query starts from the Borrowing table, which holds the relationship between the books and the members.
3. INNER JOIN:
   * Books: The Books table is joined on the BookID to get the title of each borrowed book.
   * Members: The Members table is joined on the MemberID to get the name of the member who borrowed the book.
4. WHERE: We filter the results to only include books that were borrowed after January 1, 2023 (BorrowDate >= '2023-01-01').
5. GROUP BY: We group the results by both BookID and MemberID to ensure we count how many times each book was borrowed by each member.
6. HAVING: This ensures that only books borrowed more than once by a member are included in the result.
7. ORDER BY: We order the results by BorrowCount in descending order to get the most borrowed books first.
8. LIMIT: The LIMIT 3 ensures that only the top 3 books are returned.

This query demonstrates a combination of essential SQL keywords, filtering, aggregation, and sorting to retrieve meaningful insights from the data.

**Sample Output:**

+---------------------------+------------+-------------+

| Title | MemberName | BorrowCount |

+---------------------------+------------+-------------+

| The Great Gatsby | Alice | 2 |

| To Kill a Mockingbird | Bob | 2 |

| 1984 | Alice | 1 |

+---------------------------+------------+-------------+

**Output Explanation:**

* **"The Great Gatsby"** is the most borrowed book, with Alice borrowing it twice.
* **"To Kill a Mockingbird"** is the second most borrowed, with Bob borrowing it twice.
* **"1984"** is borrowed once by Alice.

This SQL query demonstrates the use of multiple SQL keywords to extract useful insights from the library's database, such as identifying the most borrowed books by each member after a certain date.

**Aggregate Functions**

**MIN()**:

* The MIN() function returns the smallest value from a set of values in a column. It is commonly used for finding the minimum value (e.g., earliest date or lowest number).

**MAX()**:

* The MAX() function returns the largest value from a set of values in a column. It helps find the maximum value (e.g., most recent date or highest number).

**COUNT()**:

* The COUNT() function returns the number of rows in a result set or the number of non-null values in a specified column. It's often used to count entries (e.g., how many members or books exist).

**SUM()**:

* The SUM() function calculates the total sum of numeric values in a column. It’s useful for adding up values (e.g., total number of books borrowed).

**AVG()**:

* The AVG() function calculates the average value of numeric data in a column. It is commonly used to find the average (e.g., average publication year or average number of books borrowed).

**Example Queries Using the Library Management Schema**

Here’s the explanation, query, and expected output in the requested format for each of the aggregate functions:

**1. MIN() - Find the earliest published book**

This query finds the earliest published book by selecting the minimum PublishedYear from the Books table.

**Query:**

SELECT MIN(PublishedYear) AS EarliestPublishedYear

FROM Books;

**Output:**

+---------------------+

| EarliestPublishedYear|

+---------------------+

| 1990 |

+---------------------+

**2. MAX() - Find the most recently published book**

This query finds the most recently published book by selecting the maximum PublishedYear from the Books table.

**Query:**

SELECT MAX(PublishedYear) AS LatestPublishedYear

FROM Books;

**Output:**

+---------------------+

| LatestPublishedYear |

+---------------------+

| 2024 |

+---------------------+

**3. COUNT() - Count total number of books in the library**

This query counts the total number of books in the library by counting the BookID in the Books table

**Query:**

SELECT COUNT(BookID) AS TotalBooks

FROM Books;

**Output:**

+-------------+

| TotalBooks |

+-------------+

| 500 |

+-------------+

**4. SUM() - Find the total number of books borrowed by all members**

This query calculates the total number of books borrowed by all members by summing up the number of rows in the Borrowing table. Since each row represents one borrowed book, SUM(1) counts the total number of borrowed books.

**Query:**

SELECT SUM(1) AS TotalBooksBorrowed

FROM Borrowing;

**Output:**

+---------------------+

| TotalBooksBorrowed |

+---------------------+

| 1200 |

+---------------------+

**5. AVG() - Find the average number of books borrowed per member**

This query calculates the average number of books borrowed per member. First, it counts the number of books borrowed by each member and then calculates the average from those counts.

**Query:**

SELECT AVG(BooksBorrowed) AS AvgBooksBorrowed

FROM (

SELECT MemberID, COUNT(BookID) AS BooksBorrowed

FROM Borrowing

GROUP BY MemberID

) AS MemberBookCount;

**Output:**

+---------------------+

| AvgBooksBorrowed |

+---------------------+

| 5.0 |

+---------------------+

**Queries for Dates**

1. **CURDATE()**:
   * This function returns the current date. It is useful for comparing against other date fields, like filtering for records that are today, in the past, or within a date range.
2. **DATE()**:
   * This function is used to extract the date part (year, month, day) from a DATETIME or TIMESTAMP column. It's useful when you want to compare only the date (ignoring the time) from a DATETIME column.
3. **YEAR() and MONTH()**:
   * The YEAR() function extracts the year from a DATE, DATETIME, or TIMESTAMP field, while the MONTH() function extracts the month part. Both are helpful for grouping or filtering records by specific years or months.
4. **INTERVAL**:
   * The INTERVAL keyword is used to add or subtract a time unit (like days, months, or years) from a date. It is commonly used in conjunction with CURDATE() to filter records within a relative date range (e.g., the past 30 days).
5. **BETWEEN**:
   * The BETWEEN operator is used to filter records that fall within a specified range. It can be used with dates to get records between two specific dates.

**Example Queries Using Date Functions**

**1. Get all books borrowed on a specific date**

This query is designed to retrieve all the books that were borrowed on a specific date from the library database.

**Query:**

SELECT \* FROM Borrowing

WHERE BorrowDate = '2024-11-01';

**Output:**

+-------------+--------+----------+-------------+------------+

| BorrowID | BookID | MemberID | BorrowDate | ReturnDate |

+-------------+--------+----------+-------------+------------+

| 1 | 101 | 500 | 2024-11-01 | 2024-11-10 |

| 2 | 102 | 503 | 2024-11-01 | NULL |

+-------------+--------+----------+-------------+------------+

**2. Get all books borrowed in a specific month or year**

This query retrieves all books that were borrowed in a specific month or year. You can adjust the date to match the required month or year.

**Query:**

SELECT \* FROM Borrowing

WHERE YEAR(BorrowDate) = 2024 AND MONTH(BorrowDate) = 11;

**Output:**

+-------------+--------+----------+-------------+------------+

| BorrowID | BookID | MemberID | BorrowDate | ReturnDate |

+-------------+--------+----------+-------------+------------+

| 1 | 101 | 500 | 2024-11-01 | 2024-11-10 |

| 2 | 102 | 503 | 2024-11-01 | NULL |

+-------------+--------+----------+-------------+------------+

**3. Get books that are overdue**

This query helps find books that were borrowed but not yet returned, making them overdue

if the return date has passed.

**Query:**

SELECT \* FROM Borrowing

WHERE ReturnDate IS NULL AND BorrowDate < CURDATE();

**Output:**

+-------------+--------+----------+-------------+------------+

| BorrowID | BookID | MemberID | BorrowDate | ReturnDate |

+-------------+--------+----------+-------------+------------+

| 2 | 102 | 503 | 2024-10-15 | NULL |

| 5 | 105 | 507 | 2024-10-20 | NULL |

+-------------+--------+----------+-------------+------------+

**4. Count the number of books borrowed in a specific date range**

This query counts how many books were borrowed between two dates.

**Query:**

SELECT COUNT(\*) AS BooksBorrowed

FROM Borrowing

WHERE BorrowDate BETWEEN '2024-10-01' AND '2024-10-31';

**Output:**

+------------------+

| BooksBorrowed |

+------------------+

| 320 |

+------------------+

**5. Find members who borrowed books within the last 30 days**

This query identifies which members borrowed books within the past 30 days.

**Query:**

SELECT MemberID, COUNT(\*) AS BooksBorrowed

FROM Borrowing

WHERE BorrowDate >= CURDATE() - INTERVAL 30 DAY

GROUP BY MemberID;

**Output:**

+----------+---------------+

| MemberID | BooksBorrowed |

+----------+---------------+

| 500 | 3 |

| 503 | 5 |

+----------+---------------+

**6. Get the total number of books borrowed each month**

This query calculates the total number of books borrowed each month, grouped by year and month.

**Query:**

SELECT YEAR(BorrowDate) AS Year, MONTH(BorrowDate) AS Month, COUNT(\*) AS TotalBooksBorrowed

FROM Borrowing

GROUP BY YEAR(BorrowDate), MONTH(BorrowDate);

**Output:**

+------+-------+---------------------+

| Year | Month | TotalBooksBorrowed |

+------+-------+---------------------+

| 2024 | 1 | 120 |

| 2024 | 2 | 150 |

| 2024 | 3 | 110 |

+------+-------+---------------------+

**7. Find the most recent borrowing date**

This query retrieves the most recent date when a book was borrowed.

**Query:**

SELECT MAX(BorrowDate) AS MostRecentBorrowing

FROM Borrowing;

**Output:**

+---------------------+

| MostRecentBorrowing |

+---------------------+

| 2024-11-01 |

+---------------------+

**8. Get all books borrowed by a specific member in the past week**

This query finds all books borrowed by a specific member within the last week.

**Query:**

SELECT \* FROM Borrowing

WHERE MemberID = 1 AND BorrowDate >= CURDATE() - INTERVAL 7 DAY;

**Output:**

+-------------+--------+----------+-------------+------------+

| BorrowID | BookID | MemberID | BorrowDate | ReturnDate |

+-------------+--------+----------+-------------+------------+

| 3 | 103 | 1 | 2024-10-25 | 2024-11-02 |

+-------------+--------+----------+-------------+------------+

**9. Find books that were borrowed and returned on the same day**

This query identifies books that were borrowed and returned on the same day.

**Query:**

SELECT \* FROM Borrowing

WHERE DATEDIFF(ReturnDate, BorrowDate) = 0;

**Output:**

+-------------+--------+----------+-------------+------------+

| BorrowID | BookID | MemberID | BorrowDate | ReturnDate |

+-------------+--------+----------+-------------+------------+

| 10 | 110 | 520 | 2024-11-01 | 2024-11-01 |

+-------------+--------+----------+-------------+------------+

**10. Get the number of books returned in a specific month**

This query counts how many books were returned in a specific month.

**Query:**

SELECT COUNT(\*) AS BooksReturned

FROM Borrowing

WHERE MONTH(ReturnDate) = 11 AND YEAR(ReturnDate) = 2024;

**Output:**

+-----------------+

| BooksReturned |

+-----------------+

| 150 |

+-----------------+