Introduction

A Database Management System (DBMS) is a software system that enables users to create, maintain, and manipulate data stored in a database. A database is a collection of data that is organized in a specific way, typically stored on a computer system.

The primary function of a DBMS is to provide a way for users to store and retrieve data efficiently and securely. It does this by managing the storage of the data, controlling access to the data, and ensuring data integrity and consistency. A DBMS typically includes several components, such as a data dictionary, which stores information about the data in the database, a query language, which allows users to interact with the database, and tools for managing and administering the database.

Advantages of DBMS

1. Data Consistency: DBMS ensures that data is consistent across the database, thus reducing the possibility of data inconsistencies and errors.
2. Data Security: DBMS provides various security features, such as access controls, encryption, and backup and recovery mechanisms, to ensure data security.
3. Improved Data Sharing: DBMS allows multiple users to access and manipulate data simultaneously, improving data sharing and collaboration.
4. Improved Data Accessibility: DBMS provides powerful tools for searching and retrieving data, making it easier for users to access the needed data.
5. Improved Data Integration: DBMS allows data from different sources to be integrated and stored in a single database, making it easier to manage and analyze.

Disadvantage of DBMS

1. Complexity: DBMS can be complex to set up and maintain, requiring specialized skills and expertise.
2. Cost: DBMS can be expensive, especially for larger systems or those that require specialized features or capabilities.
3. Performance: DBMS can sometimes be slower than alternative approaches, such as flat files or spreadsheets, especially when dealing with large amounts of data.
4. Dependence: DBMS can create a dependence on a single vendor or technology, making it difficult to switch to a different system or technology in the future.
5. Potential for Data Loss: DBMS is vulnerable to data loss due to hardware failures, software bugs, or other technical issues, requiring robust backup and recovery mechanisms to be in place.

Features of Database Management System

1. Data Definition Language (DDL): This feature allows users to define and modify the structure of the database, including creating, modifying, and deleting tables, views, and other database objects.
2. Data Manipulation Language (DML): This feature allows users to manipulate the data stored in the database, including inserting, updating, and deleting data.
3. Query Language: A query language is a tool that allows users to retrieve specific data from the database by specifying certain conditions or criteria.
4. Data Integrity: DBMS enforces data integrity rules, such as referential integrity, to ensure the consistency and accuracy of the data stored in the database.
5. Transaction Management: DBMS provides transaction management features, which enable users to perform a series of operations as a single unit of work, ensuring that all operations are completed or none at all.

Introduction to the Entity-Relationship (ER) Model

The Entity-Relationship (ER) Model is a conceptual framework used to design and represent the data structure of a database in terms of entities, relationships, and attributes. It is an essential component of database management systems because it provides a visual representation of how data is organized, stored, and related to one another. This model was introduced by Peter Chen in 1976 and has since become one of the most widely used techniques for database design.

In the ER model, data is seen as a collection of entities, which are objects that exist independently and have a distinct identity. Relationships define how these entities interact with each other, and attributes describe the properties of the entities and relationships. An ER diagram (ERD) is used to graphically depict the ER model, making it easier to understand and communicate the database structure to both technical and non-technical stakeholders.

Key Benefits of the ER Model:

- Provides a clear and organized structure for complex databases.
- Facilitates the visualization of the relationships between different entities.
- Supports logical and conceptual database design.
- Enables easy conversion into a relational database model.

Symbols Used in ER Model

ER Model is used to model the logical view of the system from a data perspective which

consists of these symbols:

• **Rectangles:** Rectangles represent Entities in the ER Model.

• **Ellipses:** Ellipses represent Attributes in the ER Model.

• **Diamond:** Diamonds represent Relationships among Entities.

• **Lines:** Lines represent attributes to entities and entity sets with other relationship types.

• **Double Ellipse:** Double Ellipses represent Multi-Valued Attributes.

• **Double Rectangle:** Double Rectangle represents a Weak Entity.

Types of Relationships in the ER Model

Relationships define how entities are connected to each other. There are three basic types of relationships based on the number of instances that can be associated with each other:

1. One-to-One Relationship

In a one-to-one relationship, an entity in one set is related to at most one entity in another set, and vice versa.

- Example: In a university, each Student has one Student_ID Card, and each Student_ID Card is assigned to one Student.

2. One-to-Many Relationship

In a one-to-many relationship, an entity in one set can be associated with multiple entities in another set, but each entity in the second set is related to only one entity in the first set.

- Example: A Professor can teach multiple Courses, but each Course is taught by only one Professor.

3. Many-to-Many Relationship

In a many-to-many relationship, multiple entities in one set can be associated with multiple entities in another set.

- Example: A Student can enroll in many Courses, and a Course can have many Students enrolled in it.

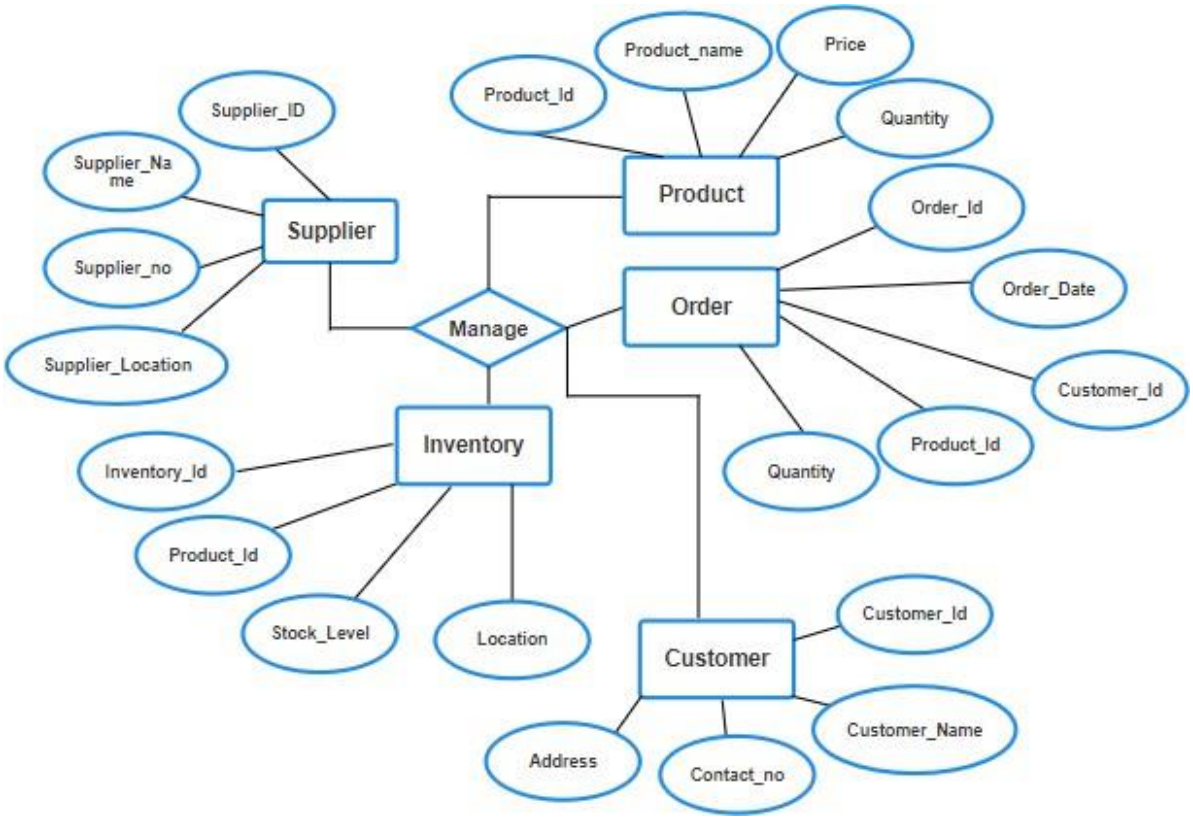
4. Recursive (Unary) Relationship

A recursive relationship occurs when an entity is related to itself. This relationship describes associations among instances of the same entity set.

- Example: In an employee hierarchy, an Employee can supervise other Employees. The Supervises relationship relates an employee to another employee within the same Employee entity.

Product Supply Management System ER Model

This ER (Entity Relationship) Diagram represents the model of the Product Supply Management System Entity. The entity-relationship diagram of the Product Supply Management System shows all the visual instruments of database tables and the relations between Supply, Stocks, Products, Accounts etc. It used structure data and to define the relationships between structured data groups of Product Supply Management System functionalities.



'Order' Table

Order_ID	Order_Date	Quantity	Product_ID	Customer_ID

'Supplier' Table

Supplier_ID	Supplier_Name	Supplier_no	Supplier_Location

'Product1' Table

Product_ID	Product_name	Price	Quantity

'Inventory' Table

Inventory_ID	Product_ID	Stock_Level	Location

'Customer' Table

Customer_ID	Customer_Name	Address	Contact_no

'Manage' Table

Supplier_ID	Product_ID

SQL Queries to Create Table

```
1  CREATE TABLE Supplier (
2      Supplier_ID INT PRIMARY KEY,
3      Supplier_Name VARCHAR(100),
4      Supplier_no VARCHAR(10),
5      Supplier_Location VARCHAR(100)
6  );
```

```
13 CREATE TABLE Inventory (
14     Inventory_ID INT PRIMARY KEY,
15     Product_ID INT,
16     Stock_Level INT,
17     Location VARCHAR(100),
18     FOREIGN KEY (Product_ID) REFERENCES Product(Product_ID)
19 );
```

```sql
CREATE TABLE Customer (
    Customer_ID INT PRIMARY KEY,
    Customer_Name VARCHAR(100),
    Address VARCHAR(150),
    Contact_no VARCHAR(10)
);
CREATE TABLE `Order` (
    Order_ID INT PRIMARY KEY,
    Order_Date DATE,
    Quantity INT,
    Product_ID INT,
    Customer_ID INT,
    FOREIGN KEY (Product_ID) REFERENCES Product(Product_ID),
    FOREIGN KEY (Customer_ID) REFERENCES Customer(Customer_ID)
);
CREATE TABLE Product1 (
    Product_ID INT PRIMARY KEY,
    Product_name VARCHAR(100),
    Price INT,
    Quantity INT
);
```

```sql
#Manage Relationship (between Supplier and Product)
CREATE TABLE Manage (
    Supplier_ID INT,
    Product_ID INT,
    PRIMARY KEY (Supplier_ID, Product_ID),
    FOREIGN KEY (Supplier_ID) REFERENCES Supplier(Supplier_ID),
    FOREIGN KEY (Product_ID) REFERENCES Product1(Product_ID));
```

SQL Queries to Insert Data

```sql
#Inserting values into Supplier table
INSERT INTO Supplier (Supplier_ID, Supplier_Name, Supplier_no, Supplier_Location) VALUES
(001, 'ARUN', '1534524485', 'DELHI'),
(002, 'VARUN', '2571842871', 'UTTAR PRADESH'),
(003, 'RAHUL', '4828215729', 'UTTAR PRADESH'),
(004, 'SAHIL', '4268572921', 'HARYANA');

#Inserting values into Product table
INSERT INTO Product1 (Product_ID, Product_name, Price, Quantity) VALUES
(708, 'SAMSUNG GALAXY PHONE', 35000, 100),
(609, 'APPLE IPHONE', 75000, 200),
(304, 'APPLE IPAD', 68000, 150),
(405, 'APPLE MACBOOK', 120000, 75),
(507, 'ONEPLUS PHONE', 30000, 300);

#Inserting values into Inventory table
INSERT INTO Inventory (Inventory_ID, Product_ID, Stock_Level, Location) VALUES
(01, 708, 80, 'NOIDA'),
(02, 609, 150, 'NOIDA'),
(03, 304, 100, 'GURUGRAM'),
(04, 405, 50, 'GURUGRAM'),
(05, 507, 200, 'NOIDA');

#Inserting values into Customer table
INSERT INTO Customer (Customer_ID, Customer_Name, Address, Contact_no) VALUES
(101, 'AKSHAT', 'DELHI', '1110223333'),
(201, 'VIDHI', 'PUNJAB', '22233300444'),
(301, 'TANYA', 'HARYANA', '33344400555'),
(401, 'CHIRAG', 'UTTAR PRADESH', '4440550666'),
(501, 'RISHABH', 'HIMACHAL PRADESH', '55566000777');

#Inserting values into Order table
INSERT INTO `Order` (Order_ID, Order_Date, Quantity, Product_ID, Customer_ID) VALUES
(9876, '2024-01-01', 20, 708, 101),
(8765, '2024-01-02', 10, 609, 201),
(7654, '2024-01-03', 50, 304, 301),
(6543, '2024-01-04', 30, 405, 401),
(5432, '2024-01-05', 10, 507, 501);
```

```
89      #Inserting values into Manage table (representing supplier-product relationships)
90 •    INSERT INTO Manage (Supplier_ID, Product_ID) VALUES
91      (001, 708),
92      (001, 609),
93      (002, 304),
94      (002, 507),
95      (003, 405),
96      (003, 708),
97      (004, 507),
98      (004, 609);
```

SQL Queries to Analyze Data

Q1. Display all the Customer.

```
Select* from Customer;
```

Result Grid

Customer_ID	Customer_Name	Address	Contact_no
101	AKSHAT	DELHI	1110223333
201	VIDHI	PUNJAB	22233300444
301	TANYA	HARYANA	33344400555
401	CHIRAG	UTTAR PRADESH	4440550666
501	RISHABH	HIMACHAL PRADESH	55566000777
NULL	NULL	NULL	NULL

Q2. List all orders placed in January 2024.

```
SELECT * FROM `Order`
WHERE Order_Date BETWEEN '2024-01-01' AND '2024-01-31';
```

Result Grid

Order_ID	Order_Date	Quantity	Product_ID	Customer_ID
5432	2024-01-05	10	507	501
6543	2024-01-04	30	405	401
7654	2024-01-03	50	304	301
8765	2024-01-02	10	609	201
9876	2024-01-01	20	708	101
NULL	NULL	NULL	NULL	NULL

Q3. Display Customer whose contact no. is ‘1110223333’.

```
Select Customer_ID, Customer_Name FROM customer where Contact_no= '1110223333';
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

Customer_ID	Customer_Name
101	AKSHAT
NULL	NULL

Q4. Display Customer whose name ends with ‘a’.

```
Select* from Customer where Customer_Name Like '%A';
```

Result Grid | Filter Rows: | Edit: | Export/I

Customer_ID	Customer_Name	Address	Contact_no
301	TANYA	HARYANA	33344400555
NULL	NULL	NULL	NULL

Q5. Find the average price of all products.

```
SELECT AVG(Price) AS Average_Price FROM Product1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell C

Average_Price
65600.0000

Q6. Display Customers in ascending order.

```
1 • Select Customer_ID, Customer_name from CUSTOMER order by Customer_name ASC;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell C

Customer_ID	Customer_name
101	AKSHAT
401	CHIRAG
501	RISHABH
301	TANYA
201	VIDHI
NULL	NULL

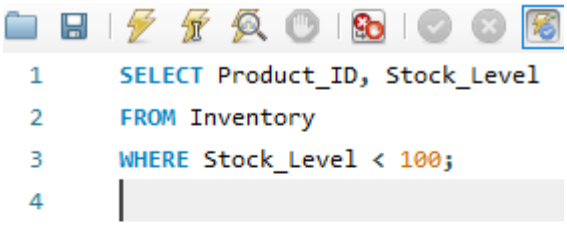
Q7. Update phone no. of Customer whose ID is 201 to 888555222.

```
1   Update Customer set Contact_no=888555222 where Customer_ID='201';
2 • Select* from Customer where Customer_ID='201';
```

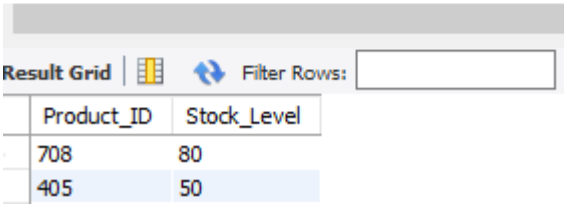
Result Grid | Filter Rows: | Edit: | Export/Import:

Customer_ID	Customer_Name	Address	Contact_no
201	VIDHI	PUNJAB	888555222
NULL	NULL	NULL	NULL

Q8. List products that have a stock level below 100 in the inventory.



```
1   SELECT Product_ID, Stock_Level
2   FROM Inventory
3   WHERE Stock_Level < 100;
4
```



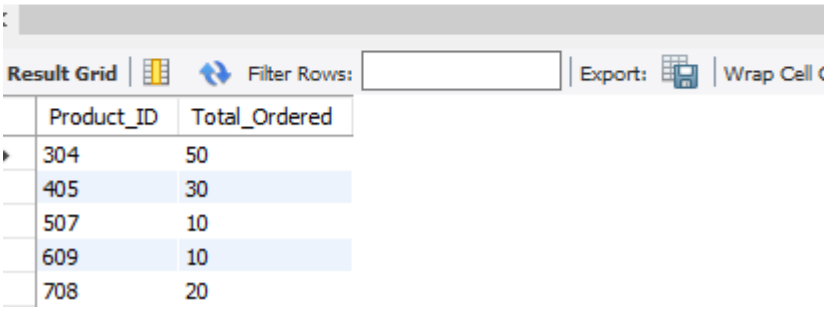
Result Grid | Filter Rows:

Product_ID	Stock_Level
708	80
405	50

Q9. Find the total quantity of each product ordered.

Limit to 1000 rows

```
1   SELECT Product_ID, SUM(Quantity) AS Total_Ordered
2   FROM `Order`
3   GROUP BY Product_ID;
4
```



Result Grid | Filter Rows: | Export: | Wrap Cell

Product_ID	Total_Ordered
304	50
405	30
507	10
609	10
708	20

Q10. Find the total revenue generated by each supplier (by aggregating the revenue from each product they supply)

```
SELECT Supplier.Supplier_Name, SUM(`Order`.Quantity * Product1.Price) AS Total_Revenue
FROM `Order`
JOIN Product1 ON `Order`.Product_ID = Product1.Product_ID
JOIN Manage ON Product1.Product_ID = Manage.Product_ID
JOIN Supplier ON Manage.Supplier_ID = Supplier.Supplier_ID
GROUP BY Supplier.Supplier_Name;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Supplier_Name	Total_Revenue
ARUN	1450000
VARUN	3700000
RAHUL	4300000
SAHIL	1050000

PL/SQL QUERIES

```
DELIMITER //

CREATE PROCEDURE update_ph (IN sid VARCHAR(50), IN pho INT)
BEGIN
    UPDATE customer
    SET customer_no = pho
    WHERE customer_id = sid;
END;
//

DELIMITER ;
```

```
114     DELIMITER //
115     CREATE PROCEDURE inventory_list1(IN cid VARCHAR(50))
116     BEGIN
117         DECLARE done INT DEFAULT FALSE;
118         DECLARE inventory_id VARCHAR(50);
119         DECLARE inventory_cursor CURSOR FOR
120             SELECT inventory_id from inventory WHERE Product_ID = cid;
121         DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
122         OPEN inventory_cursor;
123         read_loop: LOOP
124             FETCH inventory_cursor INTO inventory_id;
125             IF done THEN
126                 LEAVE read_loop;
127             END IF;
128             SELECT CONCAT('inventory: ', inventory_id) AS inventory;
129         END LOOP;
130         CLOSE inventory_cursor;
131     END;
132     //
133
134     DELIMITER ;
```