

# MODULE 5 (DATABASE)

Q.1. What do you understand By Database.

ANS.

A database is a structured collection of data, typically stored electronically in a computer system. A database management system (DBMS) is used to create, read, update, and delete data in a database. Databases are used to store and manage all sorts of data, including customer information, product information, financial data, and medical records.

Here are some of the key characteristics of databases:

- **Organized:** Databases store data in a structured way, so that it can be easily accessed and managed. This is typically done by storing data in tables, which are made up of rows and columns.
- **Persistent:** Databases are designed to store data permanently, even after the power is turned off or the computer crashes. This is achieved by storing data on disk.
- **Scalable:** Databases can be scaled to store large amounts of data. This is done by adding more storage space or by using multiple servers.
- **Secure:** Databases can be secured to protect data from unauthorized access, modification, or deletion. This is done by using passwords, encryption, and other security measures.

Q.2. What is Normalization?

ANS.

Normalization is a process in database design that aims to eliminate data redundancy and improve data integrity by organizing data into separate, related tables with well-defined relationships. The primary goal of normalization is to reduce data anomalies and ensure that the data is stored in a structured and efficient manner. This process is typically applied to relational databases, where data is organized into tables.

The normalization process is divided into several normal forms, each with specific rules and guidelines. The most commonly used normal forms are the first normal form (1NF), second normal form (2NF), and third normal form (3NF). Higher normal forms like Boyce-Codd Normal Form (BCNF) and Fourth Normal Form (4NF) exist, but they are used in more specific cases.

Here's a brief overview of the first three normal forms:

1. First Normal Form (1NF):

- In 1NF, each table should have a primary key, which uniquely identifies each row.
- Each column in the table should contain atomic values, meaning no multi-valued or composite attributes should be present.
- The order of rows and columns in the table should not affect the data's representation.

## 2. Second Normal Form (2NF):

- A table is in 2NF if it is in 1NF and all non-key attributes are fully functionally dependent on the entire primary key.
- This means that if a table has a composite primary key, each non-key attribute should depend on the entire key, not just a part of it.

## 3. Third Normal Form (3NF):



- A table is in 3NF if it is in 2NF and all transitive dependencies are removed. Transitive dependencies occur when a non-key attribute depends on another non-key attribute.
- To eliminate transitive dependencies, you may need to create additional tables and establish relationships between them.

Normalization is an iterative process, and achieving higher normal forms often results in more complex table structures with additional relationships. The level of normalization to be achieved depends on the specific requirements of the application and the trade-offs between data redundancy and query complexity. Over-normalization can lead to complex queries, so it's essential to strike a balance based on the specific use case.

Normalization helps ensure data consistency and reduces the likelihood of update anomalies (insertion, deletion, and modification) in the database. It also makes it easier to maintain and modify the database structure as the application evolves. However, in some cases, denormalization (introducing controlled redundancy) may be necessary to optimize query performance, but this should be done carefully.

## Q.3. What is Difference between DBMS and RDBMS?

ANS.

DBMS	RDBMS
 A DBMS can support various data models, including hierarchical, network, or object-oriented data models, in addition to the relational model. It is more general and may not enforce	 An RDBMS strictly follows the relational data model, where data is organized into tables with rows and columns. Tables have predefined schemas with well-defined relationships between

strict structure and relationships between data elements.	them, and data integrity is enforced through constraints.
✚ Data integrity constraints, such as referential integrity, may not be as rigorously enforced in a generic DBMS, and it may allow more flexibility in terms of data organization.	✚ RDBMS systems enforce strong data integrity rules, ensuring that data adheres to predefined constraints, keys, and relationships, which helps maintain data consistency.
✚ A DBMS can have its query language, which may not be as standardized as SQL. It may support various ways of querying and retrieving data based on the chosen data model.	✚ An RDBMS uses SQL as its query language, which is a standardized and powerful language for querying and manipulating data stored in relational databases.
✚ In a generic DBMS, relationships between data elements can be implemented in various ways, depending on the chosen data model.	✚ Relationships in an RDBMS are explicitly defined through foreign keys, and they follow the principles of referential integrity, ensuring that data relationships are maintained.
✚ A DBMS may allow for more data redundancy, as it does not enforce strict normalization rules, and data may be stored in various ways based on the chosen data model.	✚ RDBMS systems emphasize data normalization, which aims to minimize data redundancy and reduce the likelihood of anomalies in the database, promoting efficient storage and data consistency.
✚ Examples of generic DBMS include MongoDB (NoSQL database), Redis (key-value store), and CouchDB (document database).	✚ Examples of RDBMS include MySQL, PostgreSQL, Oracle Database, and Microsoft SQL Server.

Q.4. What is MF Cod Rule of RDBMS Systems?

ANS.

There appears to be a typo in your question. The correct term is "Codd's Rules for RDBMS Systems".

Codd's 12 rules, also known as Codd's Twelve Commandments, are a set of 13 rules (numbered from 0 to 12) proposed by Edgar F. Codd, a pioneer of the relational model for databases, to define the requirements for a database management system (DBMS) to be considered a relational database management system (RDBMS).

These rules aim to ensure that RDBMSs are consistent and reliable in their data management practices. They cover aspects like data representation, access, manipulation, and integrity.

Here's a brief summary of the 13 rules:

Rule 1: The Foundation Rule: A system must manage databases entirely through its relational capabilities to be considered an RDBMS.

Rule 2: The Information Rule: All data in an RDBMS must be represented explicitly and logically in tables.

Rule 3: The Guaranteed Access Rule: Every atomic value in a table must be logically accessible using a combination of table name, primary key value, and column name.

Rule 4: Systematic Treatment of Null Values: Null values represent missing or inapplicable information and must be handled systematically.

Rule 5: Active/Dynamic Online Catalog Based on the Relational Model: The database schema description must be stored online and accessible using the same query language as the data.

Rule 6: The Comprehensive Data Sublanguage Rule: The system must support at least one language to manipulate data with well-defined syntax.

Rule 7: View Updating Rule: All views that are theoretically updatable must also be practically updatable.

Rule 8: High-Level Insert, Update, and Delete: The system must support high-level operations for inserting, updating, and deleting data.

Rule 9: Physical Data Independence: Applications should not be affected by changes in the physical storage of data.

Rule 10: Logical Data Independence: Applications should not be affected by changes in the logical structure of data (e.g., tables, columns).

Rule 11: Integrity Independence: Integrity constraints should be specified separately from the application logic and stored in the catalog.

Rule 12: Distribution Independence: The distribution of the database across multiple locations should be transparent to applications.

Rule 13: Non-Subversion Rule: If a system has a low-level language that bypasses the relational interface, it cannot be considered an RDBMS.

While most modern RDBMSs comply with most of Codd's rules, few follow all of them strictly. However, Codd's rules remain a valuable benchmark for assessing the maturity and rigor of a database management system.


Q.5. What do you understand By Data Redundancy?

ANS.

Data redundancy, in simple terms, is the presence of the same information stored in multiple locations or formats. It's like having multiple copies of the same book on your bookshelf; while it might seem excessive, it serves different purposes.

Here's a breakdown of the concept:


Types of Data Redundancy:

 **Intentional:** This redundancy is deliberately implemented for various reasons, such as:

**Backup and disaster recovery:** Storing data in multiple locations ensures it remains accessible even if one location suffers a failure.

**Performance enhancement:** Replicated data on different servers can speed up access by reducing latency for users in different locations.

**Data integrity:** Certain techniques like checksums or parity bits involve storing redundant information to detect and correct errors in the data itself.

 **Unintentional:** This redundancy occurs due to inefficient data management practices and can lead to:

**Wasted storage space:** Duplicate data takes up unnecessary storage resources.

**Data inconsistency:** Different copies of the same data might be updated at different times, leading to discrepancies and confusion.

**Increased maintenance effort:** Managing and synchronizing multiple copies of data adds to the workload.

Finding the Balance:

Data redundancy is a double-edged sword. While it offers advantages like resilience and performance, it's crucial to find the right balance. Overdoing it can lead to the downsides mentioned above. Striking the right balance involves:

Minimizing unnecessary redundancy: Eliminate redundant data wherever possible to reduce storage and maintenance costs.

Implementing efficient data management practices: Use proper data modeling and synchronization techniques to ensure data consistency across all copies.

Evaluating the cost-benefit of each redundancy: Consider the trade-offs between the benefits and drawbacks of redundancy for specific situations.

By understanding the types of data redundancy and its implications, organizations can optimize their data storage and management strategies for efficiency and reliability.

I hope this explanation provides a clear understanding of data redundancy. If you have any further questions or specific scenarios you'd like me to elaborate on, feel free to ask!

Q.6. What is DDL Interpreter?

ANS.

A DDL Interpreter is a software component in a database management system (DBMS) responsible for interpreting Data Definition Language (DDL) statements. DDL statements are used to define and modify the structure of databases, such as creating, deleting, or altering tables.

Here's how the DDL Interpreter works:

1. **User submits a DDL statement:** The user submits a DDL statement through a database client or application.
2. **DDL Interpreter parses the statement:** The DDL Interpreter parses the statement to ensure it is syntactically correct and conforms to the database syntax rules.
3. **DDL Interpreter validates the statement:** The DDL Interpreter validates the statement to ensure it is semantically correct and does not violate any existing database constraints.
4. **DDL Interpreter executes the statement:** If the statement is valid, the DDL Interpreter executes it. This involves updating the database schema and any relevant metadata tables.
5. **DDL Interpreter returns a result:** The DDL Interpreter returns a result to the user, indicating whether the statement was successful or not.

Q.7. What is DML Compiler in SQL?

ANS.

The term "DML Compiler" in SQL is not widely used and can have different interpretations depending on the context. Here are two possible interpretations:

#### 1. DML Preprocessor:

In some cases, "DML Compiler" might refer to a preprocessor stage in the SQL execution process that specifically handles DML (Data Manipulation Language) statements like INSERT, UPDATE, and DELETE. This preprocessor might perform tasks such as:

Variable substitution: Replacing placeholder variables with actual values.

Expression evaluation: Evaluating expressions within the DML statement.

Authorization checks: Ensuring the user has the necessary permissions to perform the requested operation.

Generating optimized query plans: Creating efficient plans for executing the DML statement against the database.

This interpretation of DML Compiler emphasizes its role in preparing and optimizing DML statements before they are passed to the execution engine.

#### 2. DML Query Rewriter:

Another interpretation of "DML Compiler" focuses on its ability to rewrite DML statements into a more efficient form. This might involve:

Converting complex expressions into simpler ones.

Eliminating redundant operations.

Leveraging database-specific optimizations.

This interpretation focuses on the role of the DML Compiler in improving the performance and efficiency of DML operations.

It's important to note that not all SQL implementations have a distinct DML Compiler component. The tasks mentioned above might be integrated into other stages of the SQL execution process, such as the query parser or the optimizer.

If you have a specific context or source where you encountered the term "DML Compiler," it would be helpful to provide more information to understand its specific meaning in that context.

Q.8. What is SQL Key Constraints writing an Example of SQL Key Constraints.

ANS.

In SQL, key constraints are rules enforced on tables to ensure data integrity and consistency. They help to prevent invalid data from being entered into the database and maintain the relationships between different tables. There are several types of key constraints, each with its own purpose:

#### 1. PRIMARY KEY:

- \* Uniquely identifies each row in a table.
- \* A table can have only one primary key.
- \* Can consist of a single column or multiple columns.
- \* Example: sql

```
CREATE TABLE customers  
(  
    customer_id INT PRIMARY KEY,  
    name VARCHAR(255),  
    email VARCHAR(255)  
);
```

#### 2. UNIQUE KEY:

- \* Ensures that no two rows in a table have the same value for the specified column(s).
- \* A table can have multiple unique keys.
- \* Cannot contain null values.
- \* Example: sql

```
CREATE TABLE orders (  
    order_id INT PRIMARY KEY,  
    customer_id INT,  
    order_date DATE,  
    UNIQUE (customer_id, order_date)
```



);

### 3. FOREIGN KEY:

- \* Creates a relationship between two tables.
- \* References the primary key of another table (parent table).
- \* Ensures that the value in the foreign key column exists in the parent table's primary key.
- \* Helps to maintain data integrity and consistency between related tables.
- \* Example: sql

```
CREATE TABLE orders
```

```
(
```

```
    order_id INT PRIMARY KEY,
```

```
    customer_id INT,
```

```
    FOREIGN KEY (customer_id) REFERENCES customers(customer_id)
```

```
);
```

### 4. CHECK CONSTRAINT:

- \* Restricts the values that can be inserted into a column.
- \* Can be used to enforce specific data formats, ranges, or other conditions.
- \* Example: sql

```
CREATE TABLE products
```

```
(
```

```
    product_id INT PRIMARY KEY,
```

```
    name VARCHAR(255),
```

```
    price DECIMAL(10,2) CHECK (price >= 0)
```

```
);
```

These are just a few examples of SQL key constraints. Each constraint plays a crucial role in ensuring the accuracy and reliability of your data. The specific type of constraint you use will depend on the specific needs of your database and the relationships between your tables.

Q.9. What is save Point? How to create a save Point write a Query?

ANS.

A savepoint is a logical marker within a transaction that allows you to roll back to a specific point without needing to roll back the entire transaction. This can be useful when performing a series of operations within a transaction and you want to have a safety net in case of errors.

Here's how to create a savepoint in SQL:

1. Begin a transaction using the ``BEGIN`` statement.
2. Execute your desired operations.
3. At the point within the transaction where you want to create a savepoint, use the ``SAVEPOINT`` statement followed by a chosen name for the savepoint. For example:

```
sql
```

```
BEGIN;
```

```
INSERT INTO orders (customer_id, order_date) VALUES (1, '2023-12-09');
```

```
SAVEPOINT my_savepoint;
```

```
UPDATE orders SET order_date = '2023-12-10' WHERE order_id = 1;
```

This creates a savepoint named ``my_savepoint`` after the first ``INSERT`` statement.

4. Continue with your desired operations.
5. If something goes wrong, you can roll back to the savepoint using the ``ROLLBACK TO SAVEPOINT`` statement followed by the savepoint name:

sql

```
ROLLBACK TO SAVEPOINT my_savepoint;
```

This will undo all operations made after the `my\_savepoint` was created.

6. You can release the savepoint using the `RELEASE SAVEPOINT` statement, followed by the savepoint name. This is not required, but it can improve performance and free up resources.

Sql

```
RELEASE SAVEPOINT my_savepoint;
```

7. Finally, you can commit the entire transaction using the `COMMIT` statement.

Advantages of using savepoints:

Improved error handling: You can easily roll back to a specific point if an error occurs during a transaction, minimizing data loss.

Increased flexibility: You can structure your transactions more efficiently by creating savepoints at key points.

Better performance: Using savepoints can improve performance compared to rolling back the entire transaction.

Q.10. What is trigger and how to create a Trigger in SQL?

ANS.

A trigger is a database object that automatically executes a set of SQL statements when a specific event occurs on a table. These events can be:

- **DML events:** INSERT, UPDATE, or DELETE statements on the table.
- **DDL events:** CREATE, ALTER, or DROP statements on the table or its schema.

Triggers are powerful tools that can automate various tasks and enforce data integrity in your database. They can be used for:

- **Enforcing data validation:** You can use triggers to automatically validate data before it is inserted or updated.
- **Auditing data changes:** You can use triggers to log changes made to the table, which can be helpful for auditing purposes.
- **Maintaining data consistency:** You can use triggers to automatically update other tables based on changes made to a specific table.
- **Implementing security measures:** You can use triggers to restrict access to certain data or operations.

## To Create a Trigger in SQL

The syntax for creating a trigger varies slightly depending on the specific database you are using. However, the general structure is as follows:

### SQL

```
CREATE TRIGGER trigger_name
{ BEFORE | AFTER } { INSERT | UPDATE | DELETE }
ON table_name
FOR EACH ROW
BEGIN
    -- Trigger body
END;
```

Let's break down the syntax:

- `trigger_name`: The name given to the trigger.
- `{ BEFORE | AFTER }`: This specifies whether the trigger will fire before or after the triggering event occurs.
- `{ INSERT | UPDATE | DELETE }`: This specifies the event that will trigger the trigger.
- `table_name`: The name of the table on which the trigger will be applied.
- `FOR EACH ROW`: This specifies that the trigger will be fired for each row affected by the triggering event.
- `BEGIN`: This marks the beginning of the trigger body, where you can write your trigger logic using SQL statements.
- `END`: This marks the end of the trigger body.

Here is an example of how to create a trigger that automatically sets the `created_at` and `updated_at` columns for each new row inserted into the `users` table:

### SQL

```
CREATE TRIGGER set_timestamps
BEFORE INSERT

ON users
FOR

EACH
```

```

ROW

BEGIN

SET NEW.created_at =

CURRENT_TIMESTAMP;
    SET NEW.updated_at =

CURRENT_TIMESTAMP;
END;

```

This trigger will automatically set the `created_at` and `updated_at` columns to the current timestamp whenever a new row is inserted into the `users` table.

## TASK

1. Create Table Name : Student and Exam

Query:-

```

CREATE TABLE student( Rollno INT PRIMARY KEY, NAME
VARCHAR(10), Branch VARCHAR(25) );
INSERT INTO `student`(`Rollno`, `NAME`, `Branch`)
VALUES(1,"Jay","Computer Science"),(2,"Suhani","Electronic and Com"),
(3,"Kriti","Electronic and Com");
CREATE TABLE Exam( Rollno INT, S_code TEXT, Marks INT, P_code
CHARACTER, FOREIGN KEY (Rollno) REFERENCES student(Rollno) );
INSERT INTO `exam`(`Rollno`, `S_code`, `Marks`, `P_code`)
VALUES(1, "CS11", 50, "CS"),(1,"CS12",60,"CS"),(2,"EC101",66,"EC"),
(2,"EC102",70,"EC"),(3,"EC101",45,"EC"),(3,"EC102",50,"EC");

```

2. Create table given below.

Query:-

```

CREATE TABLE Emp( FirstName varchar(10), LastName
varchar(10), Address text, City varchar(10), Age int );
INSERT INTO `emp`(`FirstName`, `LastName`, `Address`, `City`, `Age`)
VALUES("Mickey","Mouse","123 Fantasy Way","Anaheim",73),
("Bat","Man","321 Cavern Ave","Gotham",54),
("Wonder","Woman","987 Truth Way","Paradise",39),
("Donald","Duck","555 Quack Street","Mallard",65),

```

("Bugs","Bunny","567 Carrot Street","Rascal",58),  
("Wiley","Coyote","999 Acme Way","Canyon",61),  
("Cat","Woman","234 Purrfect Street","Hairball",32),  
("Tweety","Bird","543","Itotltaw",28);

3. Create table given below: Employee and Incentive.

Query:-

(a)SELECT First\_name FROM employee WHERE First\_name =  
'Tom';  
(b)SELECT First\_name,Salary,Joining\_date FROM employee;  
(c)SELECT \* FROM employee ORDER BY First\_name ASC; AND SELECT \*  
FROM employee ORDER BY Salary DESC;  
(d)SELECT \* FROM employee WHERE First\_name LIKE 'j%';  
(e)SELECT Department, MAX(Salary) AS MaxSalary FROM Employee  
GROUP BY Department ORDER BY MaxSalary ASC;  
(f)SELECT First\_name, incentive\_amount FROM incentive,employee  
HAVING COUNT(incentive\_amount) > 3000;

4. Create table given below: Salesperson and Customer

Query:-

(a)SELECT RATING FROM customer WHERE RATING > 100;  
(b)SELECT SNAME,CITY from salesperson WHERE city="london" AND  
COMM > 0.12;  
(c)SELECT \* FROM salesperson WHERE CITY="barcelona" OR city  
="london";  
(d)SELECT \* FROM salesperson WHERE COMM BETWEEN .10 AND .12;  
(e)SELECT \* FROM `customer` WHERE CITY = "roe" and RATING<=100;