**Trigger Procedures in DBMS:**

A trigger in a Database Management System (DBMS) is a set of instructions or a program that automatically executes in response to a specific event occurring in the database. These events can include INSERT, UPDATE, DELETE operations on a table. Trigger procedures are widely used to enforce data integrity, implement business rules, and automate certain actions within the database. Let's break down key aspects of trigger procedures:

**1. Types of Triggers:**

* **Before Triggers:** Executed before the triggering event (e.g., BEFORE INSERT). Used for validation and modification of data before it is added to the database.
* **After Triggers:** Executed after the triggering event (e.g., AFTER UPDATE). Used for actions that depend on the successful completion of the triggering event.

**2. Trigger Actions:**

* **Validation:** Triggers can be used to check data integrity constraints, ensuring that only valid data is added or modified in the database.
* **Automated Actions:** Triggers can automate actions such as updating related tables, maintaining audit logs, or sending notifications.
* **Enforcing Business Rules:** Trigger procedures are often employed to enforce specific business rules and ensure that the data adheres to predefined standards.

**3. Example:**

Consider a scenario where you have a database table called 'Orders,' and you want to ensure that no order with a total cost below a certain threshold is allowed. You can create a trigger procedure that fires before an INSERT or UPDATE operation on the 'Orders' table. This trigger would check the total cost and raise an error if it falls below the specified threshold.

sqlCopy code

CREATE TRIGGER check\_order\_total BEFORE INSERT OR UPDATE ON Orders FOR EACH ROW BEGIN IF NEW.total\_cost < 1000 THEN SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Order total must be at least $1000.'; END IF; END;

This trigger ensures that any attempt to insert or update an order with a total cost below $1000 is rejected.

**Normalization in DBMS:**

Normalization is the process of organizing data in a database to reduce redundancy and dependency issues. It involves breaking down large tables into smaller, related tables and establishing relationships between them. The goal is to eliminate data anomalies such as insertion, update, and deletion anomalies, and to achieve a higher level of data integrity. There are several normal forms, with each successive normal form addressing specific types of dependencies. Let's explore key aspects of normalization:

**1. Normal Forms:**

* **First Normal Form (1NF):** Ensures that each attribute contains only atomic (indivisible) values, and there are no repeating groups or arrays.
* **Second Normal Form (2NF):** Builds on 1NF by ensuring that non-prime attributes are fully functionally dependent on the primary key, addressing partial dependencies.
* **Third Normal Form (3NF):** Extends the normalization process by eliminating transitive dependencies, ensuring that non-prime attributes are not dependent on other non-prime attributes.
* **Boyce-Codd Normal Form (BCNF):** A stricter form of 3NF, ensuring that every determinant is a superkey.

**2. Advantages of Normalization:**

* **Data Integrity:** Normalization reduces redundancy and eliminates data anomalies, leading to improved data integrity.
* **Simplified Updates:** The database becomes more modular and easier to maintain because updates only need to be made in one place.
* **Consistency:** Relationships between tables are explicitly defined, ensuring consistent data across the database.

**3. Example:**

Consider an unnormalized table called 'EmployeeInfo' with attributes **EmployeeID, EmployeeName, Department, and ManagerName**. This table might have redundant information if multiple employees are in the same department.

| **EmployeeID** | **EmployeeName** | **Department** | **ManagerName** |
| --- | --- | --- | --- |
| 1 | Alice | HR | Bob |
| 2 | Bob | IT | Charlie |
| 3 | Charlie | HR | Alice |

By normalizing this table into separate tables for 'Employees' and 'Departments,' we eliminate redundancy:

**Employees Table:**

| **EmployeeID** | **EmployeeName** |
| --- | --- |
| 1 | Alice |
| 2 | Bob |
| 3 | Charlie |

**Departments Table:**

| **Department** | **ManagerName** |
| --- | --- |
| HR | Alice |
| IT | Charlie |

**4. Considerations:**

* **Trade-offs:** Achieving higher normal forms may result in more complex queries and joins, potentially impacting performance.
* **Application Design:** The normalization process should align with the specific requirements and use cases of the application.

Normalization is a crucial aspect of database design, aiming to create a database structure that is efficient, maintainable, and free from data anomalies. It helps ensure the consistency and reliability of data in a relational database.