CCINFOM Concepts of Database Design

Database Design Techniques

Database design techniques ensure efficient data organization and retrieval while maintaining data integrity.

Identification of Relations

The first step in database design is to identify the relations (tables) required for representing the entities and their attributes.

- **Entities**: Real-world objects or concepts that need to be stored in the database (e.g., `Student`, `Course`).
- **Relations**: Tables that store information about entities.
    - Example: If you have entities like `Customer` and `Order`, you would create corresponding tables:
        - `Customers(CustomerID, Name, Email)`
        - `Orders(OrderID, CustomerID, OrderDate)`

This step involves deciding how to break down the data into tables to avoid redundancy and ensure data integrity.

Identification of Relationships

Once the relations are identified, the next step is to define how these tables are connected to one another using relationships.

- **Types of Relationships**:
    - **One-to-One**: Each record in one table corresponds to exactly one record in another table.
        - Example: A `Person` table with a one-to-one relationship with a `Passport` table.
    - **One-to-Many**: A record in one table can be related to multiple records in another table.
        - Example: A `Customer` can place multiple `Orders`.
    - **Many-to-Many**: Multiple records in one table can relate to multiple records in another table. This is usually implemented using a junction table.
        - Example: `Students` enrolling in multiple `Courses` and vice versa, managed through an `Enrollments` table.

Required and Unique Values

In database design, attributes can be marked as required or unique to enforce data integrity.

- **Required Values**: Ensures that an attribute cannot be left empty (i.e., `NOT NULL`).
    - Example: `CustomerID` in the `Orders` table must always have a value.
- **Unique Values**: Ensures that no two records have the same value for a specific attribute.
    - Example: An `Email` field in a `Users` table should be unique to prevent duplicate entries.

Composite and Multi-valued Attributes

- **Composite Attributes**: Attributes that can be divided into smaller parts. Since this violates the Atomic Design Pattern (which also means it violates 1NF), composite attributes are never used in practical application.
    - Example: A `FullName` attribute can be broken down into `FirstName`, `MiddleName`, and `LastName`.
- **Multi-valued Attributes**: Attributes that can have multiple values for a single entity.
    - Example: A `PhoneNumbers` attribute for a `Customer` might store more than one phone number.
    - Solution: Multi-valued attributes are typically normalized into a separate table.

Controlled or Free-form Values

- **Controlled Values**: Also known as enumerated values, these are predefined sets of values that an attribute can take.
    - Example: A `Status` field in an `Orders` table may only accept values like 'Pending', 'Shipped', or 'Delivered'.
- **Free-form Values**: Allow users to enter any data without restrictions.
    - Example: A `Comments` section where users can input feedback.

Generalization and Specialization

These techniques are used in Entity-Relationship Modeling to represent hierarchical relationships between entities.

- **Generalization**: The process of extracting common characteristics from multiple entities to create a generalized entity.
    - Example: `Car` and `Motorcycle` entities can be generalized into a `Vehicle` entity.

- **Specialization**: The opposite of generalization, where a broad entity is divided into more specialized entities.
    - Example: A `Person` entity can be specialized into `Student` and `Teacher`.