# Database Design

The design of a database is crucial to ensuring data is stored, accessed, and maintained efficiently. A well-designed database supports data integrity, performance, and scalability.

### Parts of a Database Design

Database design consists of both structural components and rules that govern the organization and behavior of data.

- **Structures**: These include tables, columns (fields), data types, keys (primary and foreign), and relationships. These structures define how data is stored, related, and accessed.
    - **Tables**: The core unit where data is stored in rows and columns. Each table represents an entity (e.g., customers, products).

| CustomerID | CustomerName | Email | Phone |
|---|---|---|---|
| 1 | Miyuki Shirogane | shirogane@shujin.edu.jp | 123-456-7890 |
| 2 | Kaguya Shinomiya | shinomiya@shujin.edu.jp | 987-654-3210 |

    - **Columns/Fields**: Define the attributes of entities (e.g., `CustomerName`, `ProductPrice`).
        - `CustomerID`: INTEGER (primary key)
        - `CustomerName`: VARCHAR(50)
        - `Email`: VARCHAR(100)
        - `Phone`: VARCHAR(15)
    - **Primary Key**: Uniquely identifies each record in a table.
        - Example: In the `Customers` table, `CustomerID` is the primary key, ensuring that no two customers have the same ID.
    - **Foreign Key**: Establishes a relationship between two tables.
        - Example: Suppose we have a table for orders:

| OrderID | OrderDate | CustomerID |
|---|---|---|
| 1001 | 2024-11-05 | 1 |
| 1002 | 2024-11-01 | 2 |

        - Here, the `CustomerID` column in the `Orders` table is a foreign key referencing the `CustomerID` in the `Customers` table.

- **Business Rules**: These are the constraints that enforce data integrity and business logic.

**Business Rules**

Business rules define how data can be created, stored, and manipulated, ensuring that operations align with organizational policies. These can be classified into two categories:

- **Database-oriented Business Rules**:
    - Enforced directly within the database through constraints, triggers, and stored procedures.
    - Example: A university database enforces that a student cannot register for more than 21 credits per semester. This can be implemented using a trigger that checks the total credits during registration.
- **Application-oriented Business Rules**:
    - Implemented within the application logic rather than directly in the database.
    - Example: An e-commerce platform may restrict product returns to within 30 days of purchase. This rule is enforced by the application's order management module rather than the database itself.

Suppose you have a library database with tables for books and loans. A database-oriented rule might enforce that a book can only be loaned out if it is not already checked out, while an application-oriented rule might limit users to borrowing a maximum of 5 books at a time.

**Database Requirements**

Database requirements are the specifications that the database must fulfill to align with business needs and user expectations. This phase involves gathering and analyzing data requirements from stakeholders to define:

- **Functional Requirements**: These describe what the database should do for users to accomplish their tasks (e.g., storing customer orders, managing employee records).
- **Non-functional Requirements**: These are set of specifications that describe the system's operation capabilities and constraints
    - **Performance**: Use indexing to optimize query performance.
    - **Scalability**: Design the database to accommodate increasing data volume.
    - **Security**: Implement user access control and data encryption.

This process usually involves creating a requirements specification document that outlines what the database must achieve.

# CCINFOM Concepts of Database Design

## Criteria of a Good Database Design

A good database design is important for ensuring data integrity, efficiency, and scalability. Some key criteria include:

- **Data Integrity and Consistency**: Ensures that data remains accurate and consistent across tables and transactions.
    - Example: If a product is deleted, its related orders should not reference a non-existent product (cascading and restricting deletion rules).

```sql
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    ProductID INT,
    FOREIGN KEY (ProductID)
        REFERENCES Products(ProductID)
        ON DELETE CASCADE
);
```

- **Normalization**: Eliminates redundancy and improves data integrity by organizing tables so that each table contains data related to a single entity or concept.
    - Example: Separating customer information into a `Customer` table and orders into an `Order` table, instead of mixing them.
- **Scalability**: The database should be designed to handle growth in data volume and user load.

## Data Integrity

Data integrity refers to the correctness, consistency, and accuracy of data in a database. It ensures that data remains reliable and valid over its lifecycle. There are three main types of integrity:

- **Entity Integrity**
    - Ensures that each row in a table is uniquely identifiable.
    - Achieved by having a primary key for each table, where the key must be unique and not null.
    - Example: In a `Students` table, `StudentID` serves as a primary key, ensuring no two students have the same ID and that no student entry has a null ID.
- **Referential Integrity**
    - Ensures that relationships between tables remain consistent.
    - Enforced using foreign keys, which link a record in one table to a primary key in another table.

  - If a foreign key value is used, it must correspond to an existing primary key value in the related table.
  - Example: In an `Orders` table, `CustomerID` should reference an existing `CustomerID` in the `Customers` table.

- **Domain Integrity**
  - Ensures that the values in a column fall within a specified range, type, or set.
  - Enforced using data types, constraints, and validation rules (like `CHECK` constraints).
  - Example: A column `Age` may have a constraint to accept only positive integers.