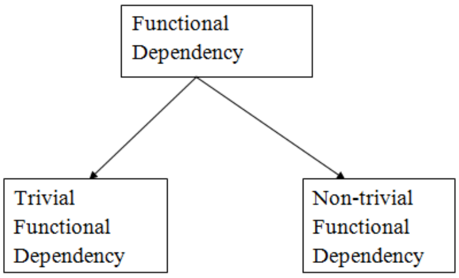
- **Functional Dependency:**

  - **Definition:** Functional dependency is a relation that exists between two attributes. It typically exists between the primary key and non key attribute within a table.

- **X→Y:** The left side of FD is known as determinant; the right side of the production is known as a dependent.


Functional Dependency
Trivial Functional Dependency
Non-trivial Functional Dependency

- **Types of functional Dependency:**
  1. Trivial FD
  2. Non-Trivial FD
  3. Multivalued FD
  4. Transitive FD
  5. Fully FD

  6. Partial FD
     - **Trivial functional dependency:** A→ B has trivial functional dependency if B is a subset of A. A dependent is always a subset of the determinant.

     - **Non-trivial Functional Dependency:** A→B is non-trivial dependency if B is not a subset of A. The dependent is strictly not a subset of the determinant.

     - **Multivalued functional dependency:** If A→ {B, C} and there exists no functional dependency between B and C. Entities of the dependent set are not dependent on each other.

     - **Transitive Functional Dependency:** If A → B & B → C then A → C. Dependent is indirectly dependent on the determinant.

     - **Fully Functional Dependency:** An attribute or a set of attributes uniquely determines another attribute or set of attributes. A relation R has attributes X, Y, Z with dependencies X → Y and X → Z.

     - **Partial Functional Dependency:** a non-key attribute depends on a part of the composite keys, rather than the whole key. If a relation R has attributes X, Y, Z where X and Y are the composite key and Z is non key attribute. Then X->Z is a partial functional dependency in RBDMS.

- **Armstrong's Axioms:** Armstrong's axioms are used to test the logical implication of functional dependencies.

  - **Axiom of Reflexivity**: If Y is a subset of X, then X → Y. Any subset of the attribute is functionally dependent on the whole set of attributes.

  - **Axiom of Augmentation:** If X → Y, then XZ → YZ for any attribute set Z. Adding attributes to both sides of a functional dependency maintains its validity.

  - **Axiom of Transitivity:** If X → Y and Y → Z, then X → Z. If a functional dependency can be derived indirectly, it can be inferred directly.

  - **Interference Rules:** These rules are derived from the above axioms:
    - **Union:** If X→Y and X→Z then X→YZ.

    - **Composition:** If A→B and X→Y hold, then AX→BY holds.
    - **Decomposition:** If X→YZ then X→Y and X→Z.
    - **Pseudo Transitivity:** If X→Y and YZ→W then XZ→W.

- **Normalization:** A database design technique used to eliminate data redundancy and improve data integrity by organizing data in a structured and efficient manner.

  Involves breaking down a relational database into multiple related tables while adhering to specific rules.

- **Benefits of normalization:**
  - **Data integrity:** normalisation minimizes data duplication and inconsistencies, ensuring accurate and reliable information.

  - **Efficiency:** Smaller, more focused tables improve query performance and reduce storage requirements.

  - **Flexibility:** Normalised structures allow for easier data manipulation, updates, and maintenance.
- **Normal Forms:**
  1. **First normal form (1NF):**
     a. Each table should hold a single atomic value.
     b. Consider a “Customers” table with column “Phone Numbers” containing multiple phone numbers. To achieve 1NF, create a separate “Phone Numbers” table with a customer Id and Phone Number.

  2. **Second normal form (2NF):**
     a. Meets 1NF requirements.
     b. No partial dependencies: Non-key attributes depend on the entire primary key, not just part of it.
     c. In a “Sales” tables with “Order ID“, “Product ID” and “Quantity” where “Product ID” depends only on part of the primary key, create a separate “Products” table.

  3. **Third Normal Form(3NF):**
     a. Meets 2NF requirements.
     b. No Transitive dependencies: Non-key attributes depend only on the primary key, not on other non-key attributes.
     c. In a "Students" table with "Student ID," "Course ID," and "Instructor," where "Instructor" depends on "Course ID," move "Instructor" to a separate "Courses" table.

  4. **Boyce-Codd Normal Form (BCNF):**
     a. Meets 3NF requirements.
     b. Every determinant (attributes that uniquely determine other attributes) must be a candidate key.
     c. Example: In an "Employees" table with "Employee ID," "Project ID," and "Project Manager," where "Project Manager" depends on "Project ID," separate "Project Managers" from "Projects."

- **Database security:**
  1. Database security refers to the protection of data stored in database from unauthorised access, tampering, and other malicious activities.

  2. **Threats to database security:**
     - **Unauthorised Access:** Unauthorised users gaining access to the database, either by exploiting vulnerabilities or using stolen credentials.

- **Data leakage:** Sensitive information being leaked to unauthorized parties, often due to poor access controls or misconfigurations.

- **SQL injection:** Malicious SQL statements are injected into user input to manipulate or access the database.

- **Malware and Ransomware:** Malicious software can inject databases, steal data, or hold it ransom.

- **Insider threats:** Authorised individuals with malicious intent accessing, manipulating, or leaking data.

- **Data Tampering:** Unauthorised modification of data to manipulate records or disrupt business operations.

- **Denial of service:** Attackers overwhelm the database with excessive requests, leading to a slowdown or complete outage.

- **Weak authentication and authorization:** Poorly managed user access privileges that can lead to unauthorized actions within the database.

- **Insecure Configurations:** Poorly configured databases with default settings or unnecessary services enabled.

- **Lack of encryption:** Data transmission and storage without encryption can lead to data interception and theft.

3. **Counter measures:**

1. **Access Control:**
   a. Implement strong authentication mechanisms like multi-factor authentication (MFA).
   b. Use role-based access control (RBAC) to assign specific privileges based on user roles.
   c. Regularly review and update access permissions.

2. **Encryption:**
   a. Employ encryption for the data at rest and data in transit using protocols like TLS/SSL.
   b. Implement encryption mechanisms for sensitive fields within the database.

3. **Patch Management:**
   a. Keep database management systems and software up to date with the latest security patches.
   b. Regularly review and apply security updates to the operating system and related software.

4. **Intrusion Detection and Prevention:**
   a. Implement intrusion detection and prevention systems to monitor databases activities and detect suspicious behaviour.
   b. Set up alerts for potential security breaches or anomalies.

5. **SQL injection Prevention:**
   a. Input validation and parameterized queries to prevent SQL injections attacks.
   b. Use web application firewalls (WAFs) to detect and block malicious SQL queries.

6. **Backup and recovery:**
    a. Regularly back up the database and test data restoration procedures.
    b. Store backups in secure locations to mitigate data loss due to attacks.

7. **Auditing and monitoring:**
    a. Implement auditing to track user activities and changes to the database.
    b. Monitor logs and set up alerts for unusual or suspicious activities.

8. **Training and awareness:**
    a. Educate employees about best practices in database security and potential risks of data breaches.
    b. Promote a security-conscious culture within the organization.

9. **Vendor security assessment:**
    a. Access the security practices of third-party vendors providing database-related services.

10. **Data masking and reduction:**
    a. Mask sensitive data so that it remains confidential even to authorised users who don't need to see the full information.
    b. Implement data redaction to selectively show parts of sensitive data.

- **Control Structure:** A set of statements that control the flow of execution of a program.

  There are three types of control structures in MySQL:

1. **Sequences:** A sequence is a set of statements that are executed one after the other.

2. **Conditional Statements:**
   A statement that executes one set of statements if a certain condition is meet.

3. **Loop statements:**
   A statement that repeats a set of statements until a certain condition is met.

- **Trigger:**
  - A set of SQL statements that are executed automatically when a specified change operation (SQL, INSERT, UPDATE, or DELETE statement) is performed on a specified table.
  - Useful for tasks such as enforcing business rules, validating input data, and keeping an audit trail.

- **View:**
  - A virtual table that is created from a select statement. Views are not stored in the database, but they are dynamically created when they are referenced.
  - Used to simplify complex queries, to hide sensitive data, and to provide a consistent view of data to different users.