CCINFOM Concepts of Database Design

Relations

The concept of relations is important to relational databases, defining how data is stored and connected.

Keys in Database Design

- **Superkey and Candidate Key**
  - **Superkey**: Any set of one or more columns that uniquely identify a row in a table.
  - **Candidate Key**: A minimal superkey, meaning it uniquely identifies a row without any unnecessary columns.
  - Example: In a Students table, both `{StudentID}` and `{StudentID, Email}` could be superkeys, but only `{StudentID}` would be a candidate key.
- **Primary Key vs. Alternate Key**
  - **Primary Key**: The chosen candidate key to uniquely identify rows in a table.
  - **Alternate Key**: Other candidate keys not chosen as the primary key.
  - Example: In a `Users` table, both `Username` and `Email` could serve as candidate keys, but only one is chosen as the primary key, making the other an alternate key.
- **Natural Key vs. Surrogate Key**
  - **Natural Key**: A key derived from real-world attributes (e.g., `SSN`, `Email`).
  - **Surrogate Key**: An artificially generated key (e.g., `UserID`) with no business meaning, often used for simplicity and efficiency.
  - Example: `CustomerID` (surrogate) vs. `SSN` (natural).
- **Simple, Composite, vs. Compound Keys**
  - **Simple Key**: A single column serving as a primary key (e.g., `StudentID`).
  - **Composite Key**: A primary key made of multiple columns (e.g., `{OrderID, ProductID}`).
  - **Compound Key**: Often used interchangeably with composite key, but technically refers to combining columns that have meaning together.
  - Example: A table with `{EmployeeID, DepartmentID}` as a composite key.

CCINFOM Concepts of Database Design

Cardinality / Multiplicity

Defines the number of instances of one entity that can or must be associated with instances of another entity

- **One-to-One**: Each record in one table is associated with exactly one record in another table.
    - Used when data logically belongs to a single entity but is split into two tables for security or performance reasons.
    - Example: `Person` table linked to `Passport` table, where each person has only one passport.
- **One-to-Many**: A single record in one table can be related to multiple records in another table, but each record in the second table relates to only one record in the first.
    - Example: A `Department` table linked to an `Employees` table where each department has many employees, but each employee belongs to only one department.
- **Many-to-Many**: Records in one table can relate to multiple records in another and vice versa.
    - Typically implemented using a junction table.
    - Example: `Students` and `Courses` tables linked via an `Enrollments` table, where a student can take multiple courses, and each course can have multiple students.

Modality / Participation

Defines whether an entity's participation in a relationship is mandatory or optional.

- **Optional Participation**: Entities in one table may or may not have related records in another table.
    - Example: A `Customer` may or may not place an `Order`.
- **Mandatory Participation**: Every record in one table must have a related record in another table.
    - Example: Every `OrderItem` must be linked to an existing `Order`.

Relation Schema vs. Relation Instance

- **Relation Schema**: The structure of a table, defined by its name, attributes, and data types. It is static and does not change frequently.

- **Relation Instance**: A snapshot of the data in the table at a specific point in time. It consists of all rows (tuples) currently stored in the table.

Types of Constraints

- **Implicit Constraints**: Inherent to the data model itself, such as atomic values in the relational model.
- **Explicit Constraints**: Defined in the schema, such as domain, primary key, foreign key, and unique constraints.
- **Semantic (Application-based) Constraints**: Rules specific to the business logic, enforced at the application level.
    - Example: A bank may enforce a rule where a customer cannot withdraw more money than their current balance.

Relational Integrity Constraints

Relational integrity constraints ensure the correctness and consistency of data in a relational database:

- **Domain Constraints**: Ensure that values stored in an attribute match its defined data type and range.
    - Example: The `Age` attribute must only accept positive integers.
- **Key Constraints**: Ensure uniqueness for rows in a table.
    - **Primary Key**: Uniquely identifies each row (e.g., `StudentID`).
    - **Unique Key**: Ensures all values in a column are unique (e.g., `Email`).
- **NULL Constraints**: Restrict whether an attribute can be left empty.
    - Example: The `Email` field cannot be `NULL` for registered users.
- **Entity Integrity**: Ensures that no primary key value is `NULL`, as every record must be uniquely identifiable.
    - Example: Every `OrderID` must have a valid value.
- **Referential Integrity**: Ensures that a foreign key value in one table matches a primary key value in another table.
    - Example: An `Order` table with a `CustomerID` foreign key must reference a valid `CustomerID` in the `Customer` table.