**Schema Conversion**

### **Goals of Logical Database Design**

Logical database design serves several key purposes:

1. **Refinement of Conceptual Data Model**: Enhances the conceptual schema by providing a clearer structure for data relationships.
2. **Implementation Support**: Prepares the data model for implementation on relational DBMSs, focusing on creating a design that supports business rules.
3. **Data Quality Management**: Ensures that data adheres to defined business rules, improving the overall quality of the database.

## **Managing Redundancy**

### **Importance of Redundancy Management**

Managing redundancy is a central theme in logical database design. Unwanted redundancy can lead to several issues:

* **Data Anomalies**: Redundant data can create inconsistencies when updating records.
* **Complex Transaction Processing**: Redundant data complicates operations such as insert, update, and delete, making them error-prone.

### **Business Intelligence vs. Transaction Processing**

* **Transaction Processing**: Sensitive to redundancy as it involves modifying data.
* **Business Intelligence**: Less sensitive to redundancy because it focuses primarily on reporting and analyzing data rather than modifying it.

## **The Logical Database Design Process**

### **Transformation from ERD to Table Design**

The logical database design process begins with the ERD created during the conceptual data modeling phase. The goal is to convert this ERD into a format that a relational DBMS can understand.

### **Focus of the Logical Database Design Phase**

* **Refinement Activities**: The logical database design consists of two primary activities: conversion and normalization.

#### **Conversion Activity**

* This involves transforming an ERD into a table design. A well-defined table design will include:
  + **Tables**: Represent entities.
  + **Columns**: Represent attributes of entities.
  + **Primary Keys**: Uniquely identify each record in a table.
  + **Foreign Keys**: Establish relationships between tables.
  + **Constraints**: Enforce rules for data integrity.

#### **Normalization Activity**

* Normalization is the process of removing redundancies in a table design. It focuses on:
  + **Constraints and Dependencies**: Establishing rules that dictate how data relates to one another, helping to identify and eliminate redundancy.
  + **Uniqueness Constraints**: Adding constraints that further ensure data integrity and reduce duplication.

### **Steps of the Logical Database Design Process**

* 1. **Conversion from ERD to Table Design**: Focus on applying conversion rules to create a table structure.
  2. **Normalization**: Ensuring the table design is free of redundancy and meets business rules.
  3. **Specification**: This step becomes crucial as it defines constraints that guide the normalization process. Design tools can often automate normalization if specifications are complete.

## **Introduction to ERD Notation and Relational Data Model**

### **Difference Between ERD Notation and Relational Data Model**

The **Entity-Relationship Diagram (ERD)** is a visual representation of the data entities and their relationships within a system. In contrast, the **Relational Data Model** structures this data into tables, which can be more easily queried and manipulated in a relational database management system (DBMS). The most prominent difference lies in how relationships are represented: ERDs use named relationships, while the relational model employs **foreign keys** to establish links between tables.

## 

## **Key Conversion Rules**

### **1. Entity Type Rule**

The **Entity Type Rule** is fundamental to the conversion process:

* **Conversion to Tables**: Each entity type in the ERD is converted into a table within the relational model.
* **Primary Key Definition**: The primary key of the entity type (if it is not a weak entity) becomes the primary key of the corresponding table.
* **Attribute Mapping**: The attributes of the entity type are mapped to the columns of the table.

**Application Order**: This rule is always applied first before considering any relationships to ensure that all entities are represented as tables before establishing connections between them.

### **2. One-to-Many Relationship Rule**

The **One-to-Many Relationship Rule** governs how to handle relationships between two entity types:

* **Foreign Key Placement**: A foreign key is added to the child table. The child table is determined by the entity type located near the crow’s foot symbol, which signifies the ‘many’ side of the relationship.
* **Constraint Management**: If the relationship is mandatory (meaning every child entity must have a parent), the foreign key should be set to **NOT NULL** to enforce this constraint.

This rule ensures that each instance of the child entity is linked back to a parent entity, maintaining referential integrity.

### **3. Many-to-Many Relationship Rule**

In cases where relationships are many-to-many, the **Many-to-Many Relationship Rule** applies:

* **Creation of a Linking Table**: This rule dictates the creation of a new table (often referred to as a junction table) that contains a **combined primary key** made up of the primary keys of the two entity types involved in the relationship.
* **Attribute Inclusion**: Any attributes specific to the many-to-many relationship are included as columns in this new table.

This approach resolves the complexities of many-to-many relationships by breaking them down into manageable one-to-many relationships through a linking table.

### **4. Identification Dependency Rule**

The **Identification Dependency Rule** addresses situations involving weak entities:

* **Primary Key Composition**: This rule involves adding components to the primary key of a table when the entity is dependent on another entity for its identification. For instance, if an entity cannot be uniquely identified without reference to another entity, its primary key will include the primary key of the related entity.
* **Foreign Key Usage**: In this rule, the dependent entity’s primary key will consist of its own identifying attribute(s) along with the foreign key from the parent entity.

This rule is crucial for maintaining relationships among dependent entities and ensuring that all necessary information for identification is retained.

## **Practical Application of Conversion Rules**

* The **Entity Type Rule** is first applied to convert entities like "Course" and "Offering" into tables, establishing primary keys and corresponding attributes.
* The **One-to-Many Relationship Rule** is then utilized to introduce foreign keys into the child tables based on their relationships.
* In the case of a **Many-to-Many Relationship**, such as between "Student" and "Offering," a separate "Enrollment" table is created, incorporating foreign keys from both parent tables as a combined primary key.

## **University Database ERD**

The university database ERD showcases various features, excluding many-to-many relationships. A significant aspect of this ERD is the self-referencing relationship, particularly represented by the supervisor's relationship, which outlines the organizational chart for faculty.

### **Self-Referencing Relationship**

* **Definition**: A self-referencing relationship occurs when an entity has a relationship with itself, commonly used to represent hierarchies or organizational structures.
* **Example in ERD**: The supervisor's relationship where faculty can supervise other faculty.

## **Conversion Rules Overview**

Specific conversion rules are applied to convert the university database ERD into a relational design. The rules discussed include:

1. **Entity Type Rule**
2. **One-to-Many Relationship Rule**
3. **Identifying Relationship Rule**

### **Entity Type Rule**

* **Purpose**: This rule is applied to convert entity types into tables in the relational model.
* **Application**: In the university ERD, the following entities are converted into tables: Course, Student, Offering, Faculty, and Enrollment. Each table is assigned a primary key, with additional columns for attributes.
* **Enrollment Table Note**: The enrollment table only retains the 'Enroll Grade' column after applying this rule.

### **One-to-Many Relationship Rule**

* **Purpose**: This rule converts relationships into foreign keys.
* **Application**:
  + In the Enrollment table, StdNo, OfferNo, and CourseNo are added as foreign keys, each with a NOT NULL constraint.
  + In the Offering table, FacNo and FacSupNo are designated as foreign keys.
* **Omission of Many-to-Many Rule**: Since the university ERD does not utilize many-to-many relationships, this rule is not applied in the conversion process.

### **Identifying Relationship Rule**

* **Purpose**: This rule is used to define primary keys in dependent tables.
* **Application**:
  + For the Enrollment table, StudentNumber and OfferNumber are added as components of the primary key.
  + Notably, because these fields are primary keys, NOT NULL constraints are redundant.

## **Conversion Result**

Partial CREATE TABLE statements illustrate the conversion outcome, highlighting the following key aspects:

* **Table Names**: Each table is named according to its entity type (e.g., Course, Student, Offering, Faculty, Enrollment).
* **Constraints**: Primary keys, foreign keys, and required foreign keys are defined explicitly.

### **Complete Table Design**

The relational database diagram represents the complete table design after conversion. Additional columns may be included in the design for attributes that were abbreviated in the ERD for simplicity.

### **Data Type Considerations**

Be aware that discrepancies in data types may exist between ERD drawing tools and relational Database Management Systems (DBMS).

## **Order Entry Database Practice Problem**

1. Identify the number of applications of each conversion rule.
2. Specify primary keys and other attributes for each table.
3. Indicate foreign keys and NOT NULL constraints where applicable.

### **Rules to Apply**

* **Entity Type Rule**: Determine the number of tables and specify primary keys and other columns.
* **One-to-Many Relationship Rule**: Identify foreign keys, their corresponding tables, and specify NOT NULL constraints for required foreign keys.
* **Many-to-Many Relationship Rule**: Define the number of tables created and specify the keys.
* **Identifying Relationship Rule**: Specify components added to the primary key and identify foreign keys included as primary key components.