ER-Model

Bachelor of Technology
Computer Science and Engineering

Submitted By

ARKAPRATIM GHOSH (13000121058)

FEBRUARY 2024

Techno Main Salt Lake
EM-4/1, Sector-V,
Kolkata- 700091
West Bengal
India

TABLE OF CONTENTS

1. Introduction

The Entity-Relationship (ER) model is a conceptual data model used in database design to represent the relationships between entities within a domain. It provides a graphical representation of the entities, attributes, and relationships in a database schema, helping to visualize and organize the structure of the data. In the ER model, entities are represented as rectangles, attributes as ovals, and relationships as diamonds. Entities represent real-world objects or concepts, such as customers, products, or orders, while attributes describe the properties or characteristics of these entities. Relationships define the associations between entities and specify how they are related to each other, such as "one-to-one," "one-to-many," or "many-to-many." By using the ER model, database designers can create clear and concise representations of the data structure, facilitating communication between stakeholders and providing a foundation for the implementation of the database schema in relational database management systems (RDBMS).

2. Body

2.1. Entity Sets

In database design, an entity set refers to a collection of similar entities. Entities within a set share common characteristics and are described by the same attributes. They represent real-world objects, concepts, or events that are relevant to the application domain being modeled. Entity sets are fundamental components of the Entity-Relationship (ER) model and are used to organize and represent data within a database.

Key Characteristics of Entity Sets:

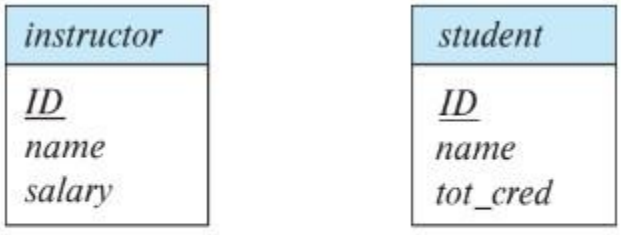
- **Homogeneity**: Entities within an entity set are homogeneous, meaning they have the same attributes and properties. For example, in a database modeling a university, the "Student" entity set would consist of entities representing individual students, each with attributes such as student ID, name, and date of birth.
- **Distinctiveness**: Each entity within an entity set is distinct and uniquely identifiable. This uniqueness is typically enforced by a primary key attribute, which ensures that no two entities within the set have the same identifier.
- **Abstraction**: Entity sets abstract real-world objects or concepts into manageable units for representation in the database. They capture the essential properties and relationships of the entities they represent without unnecessary detail.
- **Relationships**: Entity sets can be related to other entity sets through relationships in the database schema. These relationships define the associations between entities and specify how they interact with each other.

Types of Entity Sets:

- **Strong Entity Sets:** Strong entity sets have a primary key attribute that uniquely identifies each entity within the set. These entities can exist independently of other entities in the database schema. For example, the "Employee" entity set in a human resources database may be considered a strong entity set.
- **Weak Entity Sets:** Weak entity sets do not have a primary key attribute that uniquely identifies each entity on its own. Instead, they rely on a relationship with another entity set, known as the identifying or owner entity set, to provide context for their identification. Weak entity sets typically have a partial key attribute that, combined with the primary key of the identifying entity set, forms a unique identifier. For example, in a database modeling bank accounts, the "Transaction" entity set may be considered weak, relying on the "Account" entity set for identification.

Representation of Entity Sets:

In the Entity-Relationship (ER) model, entity sets are represented graphically as rectangles. The name of the entity set is typically written inside the rectangle, and its attributes are listed alongside. Relationships between entity sets are represented by lines connecting the related entity sets, with optional labels indicating the cardinality and participation constraints of the relationship.



E-R diagram showing entity sets *instructor* and *student*.

2.2. Relationship Sets

In database design, a relationship set represents the associations between entities in an Entity-Relationship (ER) model. It defines how entities from different entity sets are related to each other and specifies the nature of these relationships. Relationship sets capture the connections and dependencies between entities, allowing for a more comprehensive representation of the data model.

Key Characteristics of Relationship Sets:

- **Association**: Relationship sets describe the associations or connections between entities from different entity sets. These associations can be based on real-world interactions, dependencies, or constraints.
- **Attributes**: Relationship sets may have attributes that describe additional properties or characteristics of the relationships themselves. These attributes provide more context and information about the connections between entities.
- **Cardinality**: Cardinality constraints specify the number of entities that can be associated with each other in a relationship set. Common cardinality constraints include "one-to-one," "one-to-many," and "many-to-many."
- **Participation**: Participation constraints determine whether entities in an entity set must participate in a relationship set or if their participation is optional. Participation constraints are often classified as "total" (mandatory) or "partial" (optional).

Types of Relationship Sets:

- **Binary Relationship Sets:** Binary relationship sets involve associations between two entity sets. Examples of binary relationships include "works-for" between employees and departments, "enrolls-in" between students and courses, and "buys" between customers and products.
- **Ternary Relationship Sets:** Ternary relationship sets involve associations between three entity sets. These relationships represent more complex interactions between entities and are less common than binary relationships. An example of a ternary relationship could be "teaches" between professors, courses, and departments, where a professor teaches a course within a specific department.

Representation of Relationship Sets:

In the Entity-Relationship (ER) model, relationship sets are represented graphically as diamonds connecting the related entity sets. The name of the relationship set is typically written inside the diamond, and any attributes associated with the relationship are listed alongside. Cardinality constraints are often indicated by annotations near the lines connecting the entity sets, specifying the minimum and maximum number of entities allowed in the relationship.

2.3. Complex Attributes

In database management, complex attributes are data elements that are structured and contain multiple components or sub-attributes. Unlike simple attributes, which represent atomic values like integers or strings, complex attributes consist of nested structures, arrays, or other composite data types. Complex attributes are useful for representing data with hierarchical or multi-level structures, such as addresses, organizational charts, or multimedia files.

Types of Complex Attributes:

- **Composite Attributes:** Composite attributes are made up of multiple sub-attributes, each representing a distinct aspect of the complex attribute. For example, an address attribute may consist of sub-attributes such as street, city, state, and zip code.
- **Multivalued Attributes:** Multivalued attributes contain multiple values for a single entity, represented as a set or list of values. For instance, an employee entity may have a multivalued attribute for skills, listing all the skills possessed by the employee.
- **Derived Attributes:** Derived attributes are computed based on other attributes within the entity. These attributes are not stored explicitly in the database but are calculated when needed. An example of a derived attribute is age, which can be calculated based on the birthdate attribute.

Representation of Complex Attributes:

Complex attributes can be represented in various ways depending on the data model being used. In the Entity-Relationship (ER) model, composite attributes are depicted by nesting sub-attributes within the parent attribute. For example, an address attribute may be represented as {street, city, state, zip}. Multivalued attributes are represented using sets or lists of values, while derived attributes are typically shown as derived from other attributes within the entity.

In relational databases, complex attributes can be represented using structured data types such as arrays, structures, or user-defined types. These data types allow for the storage of complex data structures within a single attribute or column of a table. For example, a structured data type for an address may include fields for street, city, state, and zip code.

Benefits of Complex Attributes:

- **Data Organization:** Complex attributes help in organizing and structuring data in a meaningful way, allowing for more efficient storage and retrieval of information.
- Data Integrity: By grouping related data together, complex attributes help maintain data integrity and consistency within the database.

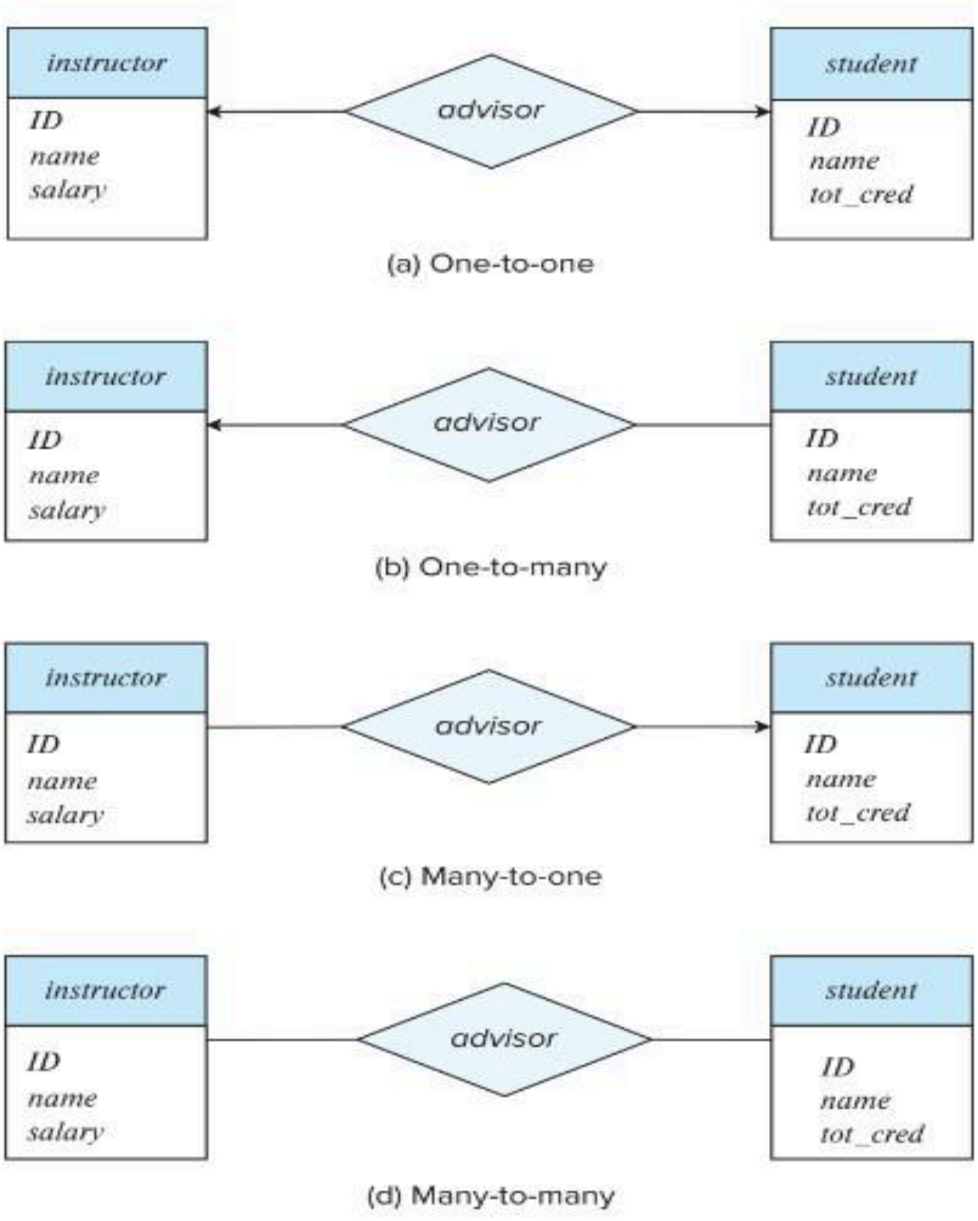
- **Flexibility**: Complex attributes provide flexibility in representing diverse data structures, accommodating a wide range of data types and formats.
- **Reduced Redundancy:** Using complex attributes can help reduce redundancy in the database by eliminating the need to store repetitive data elements separately.

2.4. Mapping Cardinalities

In database design, matching cardinalities describe the relationship between two entity sets by specifying how many entities from one entity set are related to how many entities from another entity set. Cardinality constraints are fundamental in defining the nature and behavior of relationships in an Entity-Relationship (ER) model.

Types of Cardinalities:

- **One-to-One (1:1):** In a one-to-one relationship, each entity in one entity set is associated with exactly one entity in the other entity set, and vice versa. For example, in a database modeling employees and their office locations, each employee is assigned to one office, and each office is occupied by one employee.
- **One-to-Many (1:N):** In a one-to-many relationship, each entity in one entity set can be associated with multiple entities in the other entity set, but each entity in the other entity set is associated with only one entity in the first entity set. For example, in a database modeling a customer and their orders, each customer can place multiple orders, but each order is placed by only one customer.
- **Many-to-One (N:1):** A many-to-one relationship is the inverse of a one-to-many relationship. In this type of relationship, multiple entities in one entity set can be associated with a single entity in the other entity set. For example, in a database modeling students and their classes, many students can enroll in the same class, but each class is taught by only one instructor.
- **Many-to-Many (N:M)**: In a many-to-many relationship, each entity in one entity set can be associated with multiple entities in the other entity set, and vice versa. For example, in a database modeling students and courses, each student can enroll in multiple courses, and each course can have multiple students enrolled.



Representation of Cardinalities:

Cardinalities are typically represented graphically in an ER diagram using Crow's Foot notation or Chen's notation. In Crow's Foot notation, cardinalities are indicated using symbols such as "1" for one-to-one, "M" for many, and a crow's foot symbol for one-to-many relationships. In Chen's notation, cardinalities are represented using lines connecting the related entity sets, with annotations indicating the cardinality constraints.

Importance of Matching Cardinalities:

Matching cardinalities are essential for maintaining data integrity and consistency in the database schema. By specifying the relationships between entity sets and the corresponding cardinality constraints, database designers ensure that each entity in the database has a well-defined and meaningful association with other entities. This helps prevent data anomalies such as data duplication, inconsistency, and referential integrity violations.

2.5. Primary Key

A primary key is a fundamental concept in database management systems (DBMS) that serves as a unique identifier for each record in a table within a relational database. It plays a crucial role in ensuring data integrity, enabling efficient data retrieval, and facilitating relationships between tables. Below are the key aspects of primary keys:

Uniqueness: A primary key must be unique for each record within a table. It ensures that no two records have the same value for the primary key attribute. This uniqueness constraint allows for the unambiguous identification of individual records in the table.

Uniqueness Enforcement: The DBMS enforces the uniqueness of the primary key automatically. When a new record is inserted or an existing record is updated, the DBMS checks if the value of the primary key attribute conflicts with existing values. If a conflict is detected, the operation is rejected, ensuring that the primary key remains unique.

Indexing: Primary keys are typically indexed by the DBMS for efficient data retrieval. Indexing organizes the values of the primary key attribute in a data structure (such as a B-tree or hash table), enabling fast lookup operations. Indexing enhances the performance of queries that involve filtering or sorting records based on the primary key.

Data Integrity: The primary key constraint ensures data integrity within the database. It prevents duplicate records and inconsistencies, maintaining the accuracy and reliability of the data. By enforcing the uniqueness of the primary key, the DBMS guarantees the integrity of the data stored in the table.

Relationships: Primary keys play a crucial role in establishing relationships between tables in a relational database. In a relational database schema, foreign keys are used to reference primary keys in other tables, creating links between related data. These relationships enable data normalization, maintain referential integrity, and support complex queries across multiple tables.

Attributes Selection: The selection of attributes to serve as a primary key is critical. Commonly, primary keys are chosen from attributes that uniquely identify each record, such as employee ID, customer ID, or order number. It is essential to select attributes that are stable, immutable, and unlikely to change over time to ensure the consistency and reliability of the primary key.

Composite Primary Keys: A composite primary key consists of multiple attributes that together uniquely identify each record in the table. It is used when no single attribute can uniquely identify each record, but the combination of multiple attributes does. Composite primary keys are useful in modeling complex relationships and maintaining data integrity in certain scenarios.

2.6. Removing Redundant attributes from an Entity Set

Removing redundant attributes from an entity set is a critical step in database normalization, aiming to streamline the database schema and improve data integrity and efficiency. Redundant attributes are those that can be derived or inferred from other attributes within the same entity set. By eliminating redundant attributes, database designers reduce data duplication, minimize storage requirements, and simplify data maintenance processes. Additionally, removing redundant attributes helps ensure data consistency and accuracy by reducing the risk of update anomalies and inconsistencies. Through careful analysis of the data model and normalization techniques, redundant attributes can be identified and removed without compromising the integrity or functionality of the database. This optimization process results in a more efficient and streamlined database schema, facilitating smoother data management and enhancing overall system performance.

Let's consider an example of a database modeling employees within an organization. Suppose we have an "Employees" entity set with attributes such as EmployeeID, Name, Department, and DepartmentLocation. In this scenario, the DepartmentLocation attribute might be considered redundant if we assume that the department's location can be derived from the Department attribute.

Here's how we can illustrate this:

Employees:

EmployeeID	Name	Department	DepartmentLocation
1	John	Sales	New York
2	Alice	HR	Los Angeles
3	Bob	Marketing	Chicago
4	Mary	Sales	New York

In this example, the DepartmentLocation attribute duplicates information already present in the "Department" attribute. Each employee's department implicitly identifies its location. Therefore, the "DepartmentLocation" attribute is redundant because it can be derived from the

"Department" attribute. Removing the "DepartmentLocation" attribute would streamline the database schema, reduce data duplication, and ensure consistency.

After removing the redundant attribute, the updated schema would look like this:

Employees:

| EmployeeID | Name | Department |

|---|---|---|

| 1 | John | Sales |

| 2 | Alice | HR |

| 3 | Bob | Marketing |

| 4 | Mary | Sales |

In this revised schema, the DepartmentLocation information is no longer explicitly stored, as it can be derived from the Department attribute whenever needed. This optimization results in a more efficient database schema that is easier to maintain and less prone to data inconsistencies.

2.7. Reducing ER diagrams to Relational Schemas

Reducing Entity-Relationship (ER) diagrams to relational schemas involves transforming the conceptual model represented in the ER diagram into a set of relational tables that adhere to the principles of relational database design. This process typically involves mapping entities, attributes, and relationships from the ER diagram to tables, columns, and foreign keys in the relational schema. Below, I'll explain this process step-by-step with an example.

Step 1: Identify Entities and Attributes:

Review the ER diagram to identify entities and their attributes. Each entity becomes a table in the relational schema, and each attribute becomes a column in the corresponding table.

Example:

Consider an ER diagram representing a library database with entities for books, authors, and publishers. The "Book" entity has attributes such as BookID, Title, and ISBN, while the "Author" entity has attributes such as AuthorID and Name.

Step 2: Define Relationships:

Identify relationships between entities in the ER diagram. These relationships become foreign key constraints in the relational schema, linking related tables.

Example:

In our library database example, there might be a many-to-many relationship between books and authors, as a book can have multiple authors, and an author can write multiple books. This relationship would be represented by a junction table in the relational schema.

Step 3: Normalize the Schema:

Normalize the relational schema to eliminate redundancy and ensure data integrity. This involves applying normalization rules such as ensuring each table has a primary key, avoiding repeating groups, and removing transitive dependencies.

Example:

Continuing with our library database example, we might normalize the schema by creating separate tables for authors and books, with a junction table to represent the many-to-many relationship between them. This ensures data integrity and reduces redundancy in the database.

Step 4: Map Attributes and Relationships:

Map attributes and relationships from the ER diagram to tables, columns, and foreign keys in the relational schema, ensuring that each entity, attribute, and relationship is accurately represented.

Example:

In our library database example, we might create the following relational schema:

- Authors (AuthorID, Name)
- Books (BookID, Title, ISBN, PublisherID)
- Publishers (PublisherID, Name)
- BookAuthors (BookID, AuthorID)

In this schema, the "Books" table represents the Book entity, the "Authors" table represents the Author entity, and the "Publishers" table represents the Publisher entity. The "BookAuthors" table serves as the junction table to represent the many-to-many relationship between books and authors.

Step 5: Implement Constraints:

Implement constraints such as primary keys, foreign keys, and unique constraints to enforce data integrity and maintain consistency in the relational schema.

Example:

In our library database schema, we would define primary keys for each table (e.g., AuthorID for Authors, BookID for Books) and foreign keys to establish relationships between tables (e.g., BookID and AuthorID in the BookAuthors table).

By following these steps, we can effectively reduce an ER diagram to a relational schema, ensuring that the resulting database design accurately represents the conceptual model and adheres to relational database principles.

3. Conclusion

In conclusion, the Entity-Relationship (ER) model serves as a powerful tool for conceptualizing and designing relational databases. By providing a clear graphical representation of entities, attributes, and relationships, the ER model enables database designers to capture the structure and semantics of the application domain in a systematic manner. Throughout this process, various constructs such as entity sets, attributes, relationships, and cardinalities are employed to organize and define the data model effectively. The ER model facilitates communication between stakeholders, aids in the identification of requirements, and guides the development of relational database schemas. Moreover, it lays the foundation for normalization, data integrity, and query optimization, thus contributing to the creation of efficient and well-structured databases. Overall, the ER model is an indispensable tool in database design, playing a crucial role in the development of robust and scalable database systems that meet the needs of modern applications.

4. References

Database System Concepts

Book by Abraham Silberschatz, Henry F. Korth, and S. Sudarshan