Eurasian National University named after L.N. Gumilyov

Faculty of «Information Technologies»

Department of «Information Systems»

**A blue and white logo

Description automatically generated**

**Independent work of the student: 1**

**by discipline** «Development of client server IS»

**Student:** Iskakov Y.K.

**Group:** IS-35

**Checked by:** PhD. Akhmetova. A.Z.

**Independent work of the student: 1.**

**Option №.10 DB – ER-models of the subject area**

**Table of contents**

|  |  |  |
| --- | --- | --- |
| 1 | Introduction | 3 |
| 2 | Main part |  |
| 2.1 | The Entity Relationship Data Model | 4 |
| 2.2 | Entity, Entity Set and Entity Type, Kinds of Entities | 4 |
| 2.3 | Attributes | 6 |
| 2.4 | Keys | 9 |
| 2.5 | Relationships | 10 |
| 3 | Conclusion | 13 |
| 4 | References | 14 |

**Introduction**

In the ever-evolving data management landscape, organizations increasingly rely on structured data models to represent and manage the complexities of real-world entities and their interactions. One of the most fundamental and widely used tools in this area is the Entity Relationship (ER) data model. Introduced in 1976 by Peter Chen, the ER model provides a structure for describing data in terms of entities, attributes, and relationships, and has since become a cornerstone of database design. By offering a clear and structured way to model data, the ER model simplifies the process of organizing large amounts of information and supports the creation of efficient databases that meet business requirements.

At its core, the Entity Relationship data model provides a conceptual blueprint that defines data entities (such as customers, products, or orders), their attributes (such as a customer’s name or a product’s price), and the relationships between those entities (such as a customer placing an order). This framework helps database designers and developers visualize and define how data elements interact in a system before translating that conceptual design into the physical structure of the database. The ER model uses a diagrammatic approach commonly known as an entity-relationship diagram (ERD), which graphically represents entities as boxes, attributes as ovals, and relationships as diamonds connecting them. These visual tools make it easier to communicate complex data structures to non-technical stakeholders, ensuring that the database design aligns with the organization’s real-world processes.

The ER model is especially valuable because of its ability to capture complex relationships, including one-to-one, one-to-many, and many-to-many associations, which are critical to understanding how different pieces of data interact in a system. For example, in an e-commerce application, there might be a one-to-many relationship between customers and orders (one customer can place multiple orders) or a many-to-many relationship between products and categories (a product can belong to multiple categories, and a category can contain multiple products). By accurately representing these relationships, an ER model helps create a more robust and flexible database that can handle a wide range of queries and operations.

**Main part**

**2.1 The Entity Relationship Data Model**

The Entity Relationship (ER) data model has been in use for more than three decades and remains a key tool for data modeling in databases. Its strength lies in its abstract nature, which makes it easy to conceptualize, discuss, and explain. Additionally, ER models can be easily converted into relational models for databases. Often referred to as an ER schema, these models are typically visualized through ER diagrams.

***The ER model is founded on two fundamental concepts:***

*Entities: Represented as tables, these store specific pieces of data.*

*Relationships: These describe the interactions or associations between different entities.*

To illustrate these concepts, consider the following example: Prof. Ba (entity) teaches (relationship) the Database Systems course (entity). Here, "Prof. Ba" and "Database Systems course" are entities, while "teaches" is the relationship that connects them.

Throughout this chapter, we will refer to a sample database called the COMPANY database to demonstrate the key elements of the ER model. This database contains data related to employees, departments, and projects. Key features of this database include the following:

The company consists of multiple departments. Each department is uniquely identified by an ID, and has a name, office location, and a manager who oversees it.

Each department is responsible for several projects. Every project has a unique name, a distinct number, and a specified budget.

Each employee in the company has a unique identification number, along with details such as their name, address, salary, and birthdate. Employees are assigned to a single department but can participate in multiple projects. For each project an employee joins, the start date needs to be recorded. Additionally, we must track the employee’s direct supervisor.

The company also tracks each employee's dependents, recording details such as their name, birthdate, and their relationship with the employee.

**2.2 Entity, Entity Set and Entity Type, Kinds of Entities**

An entity is a real-world object with a distinct and independent existence that can be distinguished from other objects. Entities can represent either:

*Physical objects (e.g., a lecturer, a student, a car)*

*Conceptual objects (e.g., a course, a job, a position)*

Entities are categorized based on their strength. An entity is considered weak if it is existence-dependent, meaning it cannot exist without being associated with another entity. Additionally, a weak entity’s primary key is derived from the primary key of its related parent entity. For instance, in the COMPANY database, the Spouse table is a weak entity because its primary key depends on the Employee table. A spouse record cannot exist without a corresponding employee record.

On the other hand, an entity is classified as strong if it can exist independently of any related entities. These strong entities, also known as kernels, do not rely on foreign keys to establish their existence. A table without a foreign key, or one that contains a nullable foreign key, is considered a strong entity.

Another key concept is the *entity type*, which refers to a group of similar entities. An entity set is the collection of entities of a specific entity type at a given point in time. In an entity relationship diagram (ERD), an entity type is represented by a name inside a rectangle. For example, in Figure 8.1, the entity type is EMPLOYEE.

Figure 2.2.1 ERD with entity type EMPLOYEE.

A diagram of a diagram

Description automatically generated

***Existence dependency***

An entity’s existence is dependent on the existence of the related entity. It is existence-dependent if it has a mandatory foreign key (i.e., a foreign key attribute that cannot be null). For example, in the COMPANY database, a Spouse entity is existence -dependent on the Employee entity.

***Kinds of Entities***

You should also be familiar with different kinds of entities including independent entities, dependent entities and characteristic entities. These are described below.

***Independent entities***

Independent entities, also referred to as kernels, are the backbone of the database. They are what other tables are based on. Kernels have the following characteristics:

They are the building blocks of a database.

The primary key may be simple or composite.

The primary key is not a foreign key.

They do not depend on another entity for their existence.

If we refer back to our COMPANY database, examples of an independent entity include the Customer table, Employee table or Product table.

***Dependent entities***

Dependent entities, also referred to as derived entities, depend on other tables for their meaning. These entities have the following characteristics:

Dependent entities are used to connect two kernels together.

They are said to be existence dependent on two or more tables.

Many to many relationships become associative tables with at least two foreign keys.

They may contain other attributes.

The foreign key identifies each associated table.

There are three options for the primary key:

Use a composite of foreign keys of associated tables if unique

Use a composite of foreign keys and a qualifying column

Create a new simple primary key

***Characteristic entities***

Characteristic entities provide more information about another table. These entities have the following characteristics:

They represent multivalued attributes.

They describe other entities.

They typically have a one to many relationship.

The foreign key is used to further identify the characterized table.

Options for primary key are as follows:

Use a composite of foreign key plus a qualifying column

Create a new simple primary key. In the COMPANY database, these might include:

Employee (EID, Name, Address, Age, Salary) – EID is the simple primary key.

EmployeePhone (EID, Phone) – EID is part of a composite primary key. Here, EID is also a foreign key.

**2.3 Attributes**

There are a few types of attributes you need to be familiar with. Some of these are to be left as is, but some need to be adjusted to facilitate representation in the relational model. This first section will discuss the types of attributes. Later on we will discuss fixing the attributes to fit correctly into the relational model.

***Simple attributes***

Simple attributes are those drawn from the atomic value domains; they are also called single-valued attributes. In the COMPANY database, an example of this would be: Name = {John} ; Age = {23}

***Composite attributes***

Composite attributes are those that consist of a hierarchy of attributes. Using our database example, and shown in Figure 8.3, Address may consist of Number, Street and Suburb. So this would be written as → Address = {59 + ‘Meek Street’ + ‘Kingsford’}

Blue rectangle with the word EMPLOYEE. Under this are four yellow ovals with the words Name, Address, Birthdate, Salary. There are lines between the rectangle and yellow ovals. Under the Address oval are three white ovals with the words Number, Street, Suburb.

Figure 2.3.1. An example of composite attributes.

A diagram of a company

Description automatically generated

***Multivalued attributes***

Multivalued attributes are attributes that have a set of values for each entity. An example of a multivalued attribute from the COMPANY database, as seen in Figure 2.3.2, are the degrees of an employee: BSc, MIT, PhD.

Figure 2.3.2. Example of a multivalued attribute.

A diagram of a company

Description automatically generated

Blue rectangle with the word EMPLOYEE. A line connects this to each of five yellow ovals with these words inside the ovals: Degrees, Name, Address, Birthdate, Salary

***Derived attributes***

Derived attributes are attributes that contain values calculated from other attributes. An example of this can be seen in Figure 2.3.3. Age can be derived from the attribute Birthdate. In this situation, Birthdate is called a stored attribute, which is physically saved to the database.

Figure 2.3.3. Example of a derived attribute.

A diagram of a company

Description automatically generated

**2.4 Keys**

An important constraint on an entity is the key. The key is an attribute or a group of attributes whose values can be used to uniquely identify an individual entity in an entity set.

**Types of Keys**

There are several types of keys. These are described below.

***Candidate key***

A candidate key is a simple or composite key that is unique and minimal. It is unique because no two rows in a table may have the same value at any time. It is minimal because every column is necessary in order to attain uniqueness.

From our COMPANY database example, if the entity is Employee(EID, First Name, Last Name, SIN, Address, Phone, BirthDate, Salary, DepartmentID), possible candidate keys are:

EID, SIN

First Name and Last Name – assuming there is no one else in the company with the same name

Last Name and DepartmentID – assuming two people with the same last name don’t work in the same department

***Composite key***

A composite key is composed of two or more attributes, but it must be minimal.

Using the example from the candidate key section, possible composite keys are:

First Name and Last Name – assuming there is no one else in the company with the same name

Last Name and Department ID – assuming two people with the same last name don’t work in the same department

***Primary key***

The primary key is a candidate key that is selected by the database designer to be used as an identifying mechanism for the whole entity set. It must uniquely identify tuples in a table and not be null. The primary key is indicated in the ER model by underlining the attribute.

A candidate key is selected by the designer to uniquely identify tuples in a table. It must not be null.

A key is chosen by the database designer to be used as an identifying mechanism for the whole entity set. This is referred to as the primary key. This key is indicated by underlining the attribute in the ER model.

In the following example, EID is the primary key:

Employee(EID, First Name, Last Name, SIN, Address, Phone, BirthDate, Salary, DepartmentID)

***Secondary key***

A secondary key is an attribute used strictly for retrieval purposes (can be composite), for example: Phone and Last Name.

***Alternate key***

Alternate keys are all candidate keys not chosen as the primary key.

***Foreign key***

A foreign key (FK) is an attribute in a table that references the primary key in another table OR it can be null. Both foreign and primary keys must be of the same data type.

In the COMPANY database example below, DepartmentID is the foreign key:

Employee(EID, First Name, Last Name, SIN, Address, Phone, BirthDate, Salary, DepartmentID)

**Nulls**

A null is a special symbol, independent of data type, which means either unknown or inapplicable. It does not mean zero or blank. Features of null include:

No data entry

Not permitted in the primary key

Should be avoided in other attributes

Can represent

An unknown attribute value

A known, but missing, attribute value

A “not applicable” condition

Can create problems when functions such as COUNT, AVERAGE and SUM are used

Can create logical problems when relational tables are linked

NOTE: The result of a comparison operation is null when either argument is null. The result of an arithmetic operation is null when either argument is null (except functions that ignore nulls).

**2.5 Relationships**

Relationships are the glue that holds the tables together. They are used to connect related information between tables.

Relationship strength is based on how the primary key of a related entity is defined. A weak, or non-identifying, relationship exists if the primary key of the related entity does not contain a primary key component of the parent entity. Company database examples include:

Customer(CustID, CustName)

Order(OrderID, CustID, Date)

A strong, or identifying, relationship exists when the primary key of the related entity contains the primary key component of the parent entity. Examples include:

Course(CrsCode, DeptCode, Description)

Class(CrsCode, Section, ClassTime…)

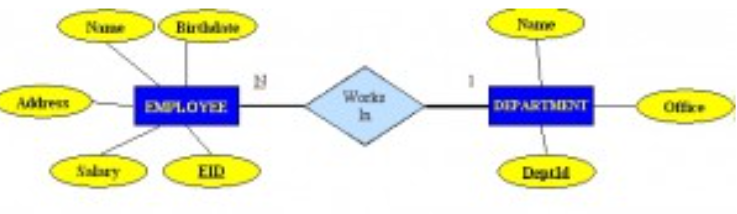
**Types of Relationships**

***One to many (1:M) relationship***

A one to many (1:M) relationship should be the norm in any relational database design and is found in all relational database environments. For example, one department has many employees. Figure 8.7 shows the relationship of one of these employees to the department.

A light blue diamond in the middle connected on either side to a blue rectangle. The rectangle on the left says EMPLOYEE and is connected with a line to five yellow ovals with the words Birthdate, Name, Address, Salary, EID. The diamond is also connected to a blue rectangle on its right with the word DEPARTMENT and that is connected with lines to three yellow ovals with the words Name, Office, DeptID.

Figure 2.5.1. Example of a one to many relationship.



***One to one (1:1) relationship***

A one to one (1:1) relationship is the relationship of one entity to only one other entity, and vice versa. It should be rare in any relational database design. In fact, it could indicate that two entities actually belong in the same table.

An example from the COMPANY database is one employee is associated with one spouse, and one spouse is associated with one employee.

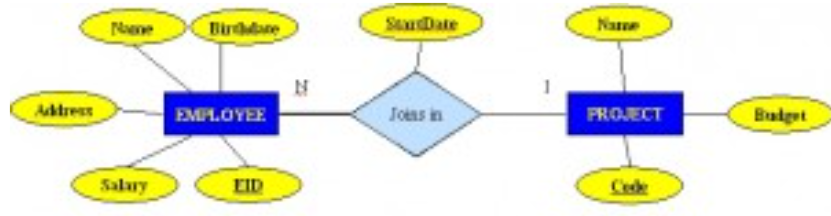
***Many to many (M:N) relationships***

For a many to many relationship, consider the following points:

* It cannot be implemented as such in the relational model.
* It can be changed into two 1:M relationships.
* It can be implemented by breaking up to produce a set of 1:M relationships.
* It involves the implementation of a composite entity.
* Creates two or more 1:M relationships.
* The composite entity table must contain at least the primary keys of the original tables.
* The linking table contains multiple occurrences of the foreign key values.
* Additional attributes may be assigned as needed.
* It can avoid problems inherent in an M:N relationship by creating a composite entity or bridge entity. For example, an employee can work on many projects OR a project can have many employees working on it, depending on the business rules. Or, a student can have many classes and a class can hold many students.

Figure 2.5.2 shows another another aspect of the M:N relationship where an employee has different start dates for different projects. Therefore, we need a JOIN table that contains the EID, Code and StartDate.

Figure 2.5.2 Example where employee has different start dates for different projects.



***Unary relationship (recursive)***

A unary relationship, also called recursive, is one in which a relationship exists between occurrences of the same entity set. In this relationship, the primary and foreign keys are the same, but they represent two entities with different roles. See Figure 2.5.3 for an example.

For some entities in a unary relationship, a separate column can be created that refers to the primary key of the same entity set.

Figure 2.5.3. Example of a unary relationship.

A diagram of a company

Description automatically generated

***Ternary Relationships***

A ternary relationship is a relationship type that involves many to many relationships between three tables.

Refer to figure for an example of mapping a ternary relationship type. Note n-ary means multiple tables in a relationship. (Remember, N = many.)

Figure 2.5.4 Example of a ternary relationship.

A diagram of a supply chain

Description automatically generated

For each n-ary (> 2) relationship, create a new relation to represent the relationship.

The primary key of the new relation is a combination of the primary keys of the participating entities that hold the N (many) side.

In most cases of an n-ary relationship, all the participating entities hold a many side.

**Conclusion**

The Entity Relationship (ER) data model has remained an important tool for designing and structuring databases for over three decades. Its success lies in its ability to abstract real-world entities, their attributes, and the relationships between them in a format that is easy to understand and implement efficiently. The ER model provides a clear representation of complex data relationships, making it easier for database designers to plan systems that accurately reflect the real-world processes they must support. From basic one-to-one and one-to-many relationships to more complex many-to-many and ternary relationships, the ER model provides a flexible framework for capturing a wide range of data interactions.

By distinguishing between strong (independent) and weak (dependent) entities, and using tools such as primary keys, foreign keys, and composite keys, the ER model ensures strong data integrity and clear data relationships. This is further enhanced by entity sets, entity types, and attributes, which provide structure to the data and ensure that the relationships between different entities are clearly defined and consistent.

Ultimately, the power of the ER model lies not only in its ability to conceptually model data, but also in its practicality. ER diagrams, with their intuitive graphical representation, make it easy to communicate complex database structures to both technical and non-technical stakeholders. As a fundamental element of database design, the ER model continues to play a significant role in developing efficient, scalable, and flexible databases that can meet the needs of modern organizations.

**References**

1. Watt, A. and N. Eng. (2014). Database Design – 2nd Edition. Victoria, B.C.: BCcampus. Retrieved from <https://opentextbc>
2. Hardy, N., & Fuell, H. Databases, data modeling and schemas. *In Springer eBooks*, 2003, pp. 277–291. doi: 10.1007/978-1-4615-0333-0\_15
3. Bai, Y., & Bhalla, S. C. Introduction to databases*. In Auerbach Publications eBooks*, 2022, pp. 9–66. doi: /10.1201/9781003304029-2
4. Elmasri, R., & Navathe, S. B. Fundamentals of Database Systems 7th ed. Textbook / Pearson, 2016. -1280 p. ISBN-13:‎ 978-0133970777