Learn – Understanding Database Relationships

In relational databases, relationships define how data in different tables is connected. There are three main types of relationships: One-to-One, One-to-Many, and Many-to-Many. These relationships are crucial to structuring data efficiently and reducing redundancy.

A. One-to-One Relationship

In a one-to-one relationship:

- Each record in Table A is associated with one and only one record in Table B, and vice versa.
- Use cases:
  - When certain attributes logically belong to a separate entity, but should not exist without their counterpart.
  - Examples: One employee may have one company car (each car is assigned to only one employee).

Key Points:

- Foreign key: You place the foreign key on either table, but usually on the "dependent" table, to establish the relationship.

```
CREATE TABLE Employees (
    employee_id INT PRIMARY KEY,
    employee_name VARCHAR(100),
    car_id INT UNIQUE,  -- Each employee gets one car
    FOREIGN KEY (car_id) REFERENCES Cars(car_id)
);
CREATE TABLE Cars (
    car_id INT PRIMARY KEY,
    car_model VARCHAR(100),
    car_registration_number VARCHAR(50)
);
```

B.One-to-Many Relationship

In a one-to-many relationship:

- One record in Table A can relate to many records in Table B, but each record in Table B can only relate to one record in Table A.
- Use cases:
    - Common in systems where a single entity (e.g., an author) can have multiple related entries (e.g., books).

Key Points:

- The foreign key resides in Table B, referencing Table A.

SQL Example:

```
CREATE TABLE Authors (
    author_id INT PRIMARY KEY,
    author_name VARCHAR(100)
);

CREATE TABLE Books (
    book_id INT PRIMARY KEY,
    book_title VARCHAR(200),
    author_id INT,
    FOREIGN KEY (author_id) REFERENCES Authors(author_id)
);
```

C. Many-to-Many Relationship

In a many-to-many relationship:

- Many records in Table A can relate to many records in Table B, and vice versa.
- Use cases:
  - When both entities in the relationship can have multiple occurrences of the other.
  - Example: Students and courses – each student can enroll in many courses, and each course can have many students.

Key Points:

- To manage many-to-many relationships, you need a junction table that holds foreign keys referencing both tables.
- The junction table effectively breaks the many-to-many relationship into two one-to-many relationships.

```
CREATE TABLE Students (
    student_id INT PRIMARY KEY,
    student_name VARCHAR(100)
);

CREATE TABLE Courses (
    course_id INT PRIMARY KEY,
    course_name VARCHAR(100)
);
```

```sql
-- Junction table for many-to-many relationship
CREATE TABLE Enrollments (
    student_id INT,
    course_id INT,
    PRIMARY KEY (student_id, course_id),
    FOREIGN KEY (student_id) REFERENCES Students(student_id),
    FOREIGN KEY (course_id) REFERENCES Courses(course_id)
);
```

3.. ER Diagrams (Entity-Relationship Diagrams)

ER Diagrams visually represent the relationships between tables in a database. Key components of ER Diagrams:

- Entities: Represented by rectangles (tables).
- Attributes: Represented by ovals (columns).
- Relationships: Represented by diamonds (connects two or more entities).
- Cardinality: Specifies how many instances of one entity can be associated with instances in another entity.

Cardinality Types in ER Diagrams:

- One-to-One: A line connects two entities with a single line at both ends.
- One-to-Many: A line connects two entities with a "crow's foot" on the "many" side.
- Many-to-Many: Two entities are connected by a crow’s foot at both ends, with a junction table placed in between.

PRACTISE

Task 1: Relationship Identification

Identify relationship type as One-to-One, One-to-Many, or Many-to-Many:

Scenario	Relationship Type
Each library card belongs to exactly one person.	__________
One product category contains many products.	__________
Doctors and patients (each patient has multiple doctors, and each doctor has multiple patients).	__________

Task 2: Establishing a One-to-Many Relationship

Create SQL tables for authors (one author, many books):

```
-- Authors table
CREATE TABLE Authors (
    author_id INT PRIMARY KEY,
    author_name VARCHAR(50)
);

-- Books table
CREATE TABLE Books (
    book_id INT PRIMARY KEY,
    book_title VARCHAR(50),
    author_id INT,
    FOREIGN KEY (author_id) REFERENCES Authors(author_id)
);
```

Task 3: Establishing a Many-to-Many Relationship

Create tables for students and sports (each student can play multiple sports):

```
-- Students table
CREATE TABLE Students (
    student_id INT PRIMARY KEY,
    student_name VARCHAR(50)
);

-- Sports table
```

```
CREATE TABLE Sports (
    sport_id INT PRIMARY KEY,
    sport_name VARCHAR(50)
);

-- Junction table (StudentSports)
CREATE TABLE StudentSports (
    student_id INT,
    sport_id INT,
    PRIMARY KEY (student_id, sport_id),
    FOREIGN KEY (student_id) REFERENCES Students(student_id),
    FOREIGN KEY (sport_id) REFERENCES Sports(sport_id)
);
```

FAQ

Q1: Can a relationship have multiple foreign keys?

A: Yes. For example, in a one-to-many relationship, Table B can have multiple foreign keys referring to different tables.

Q2: What's the difference between a one-to-one and one-to-many relationship?

A: In a one-to-one relationship, each record in both tables maps directly to one record in the other table. In a one-to-many relationship, one record in Table A can relate to many records in Table B.

Q3: Why do we need junction tables for many-to-many relationships?

A: Junction tables allow you to represent the relationship between two tables in cases where each side of the relationship can have multiple entries on the other side. Without this table, you'd be unable to establish these complex relationships directly.

Q4: How do ER Diagrams help in database design?

A: ER Diagrams provide a visual overview of how tables (entities) are related to each other, making it easier to understand the data model and identify potential issues before creating the actual database.

Q5: What's the purpose of foreign keys in relationships?

A: Foreign keys enforce referential integrity between tables, ensuring that a record in one table corresponds to an existing record in the related table.