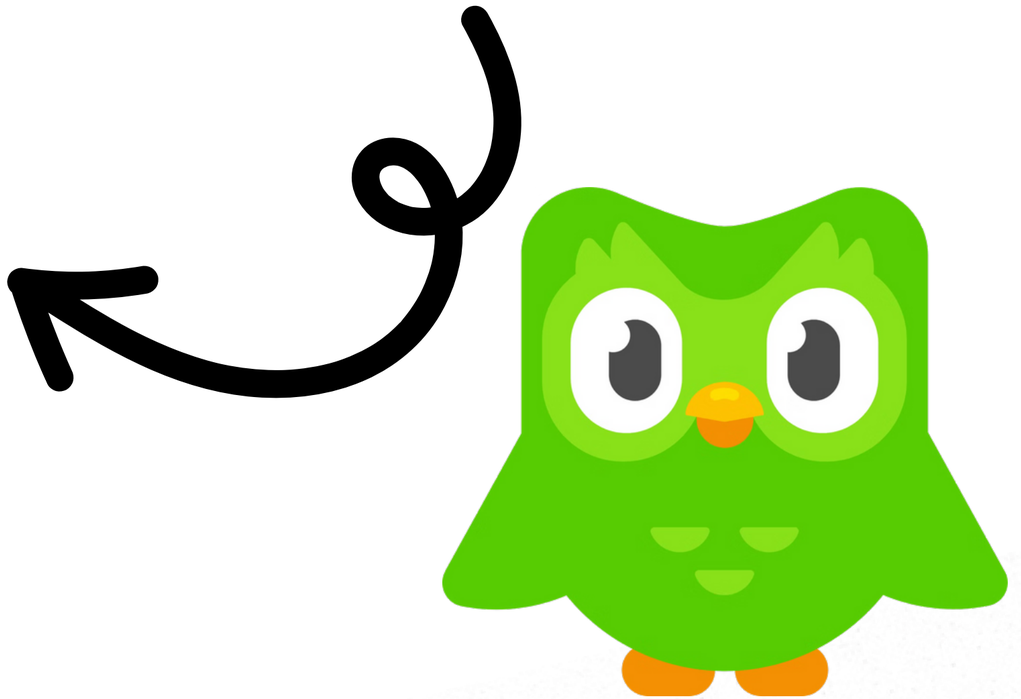
DUOLINGO

APP

BY: MARLON YECID RIVEROS
BRAYAN ESTIVEN AGUIRRE

BUSINESS MODEL

Duolingo is a language-learning platform that employs a freemium business model. Users can access basic features for free, while advanced features, such as personalized learning plans and progress tracking, are available through subscriptions.



DATA REQUIRED: USER PROFILES, LEARNING PROGRESS, AND PREFERENCES

It needs user registration and authentication to create and manage accounts, as well as the storage of user information to track progress. A catalog of available languages allows users to choose their target language

STEP 1: DEFINE COMPONENTS

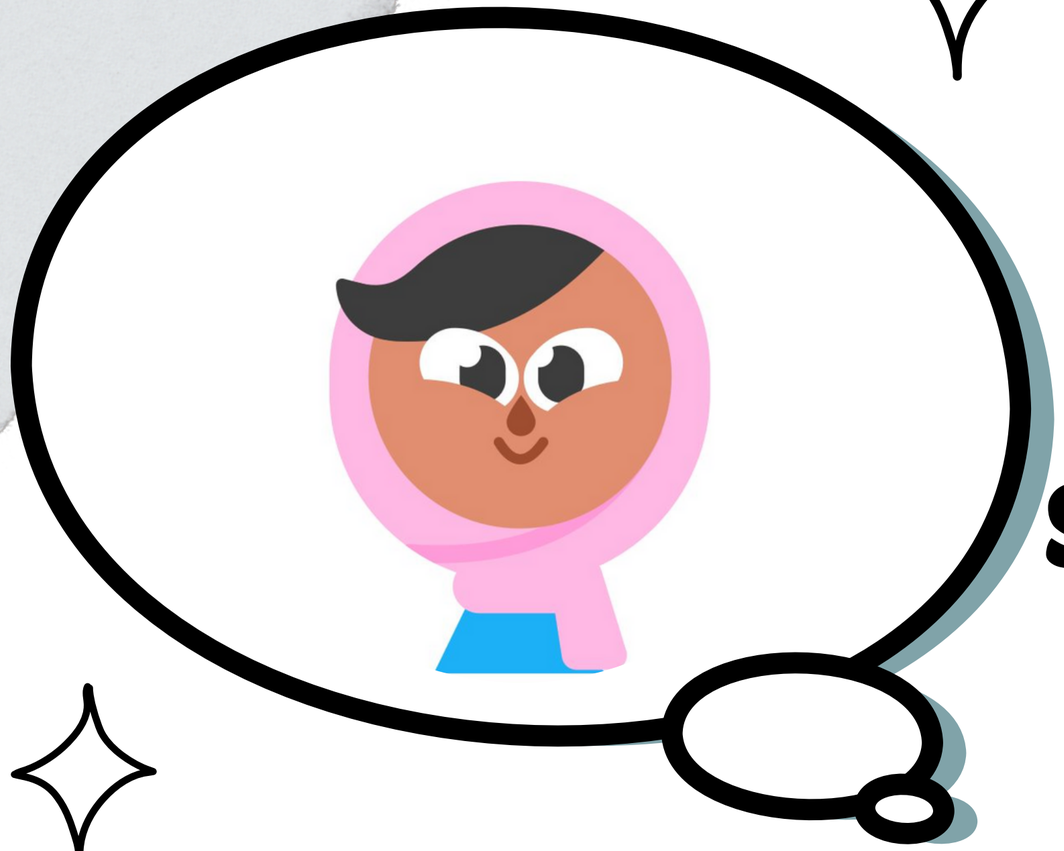
The application requires several key components to function effectively. Some of those components can be: User registration and authentication.Catalog of available languages to learn.Structuring lessons by language.

- User
- Course
- Stage
- Section
- Lesson
- Question

- Division
- challenges
- Boosters
- Achievement
- Progress

STEP 2: DEFINE ENTITIES

STEP 3: DEFINE ATTRIBUTES



In this step we explain each of the entities that we define, and we see what attributes each of these needs.

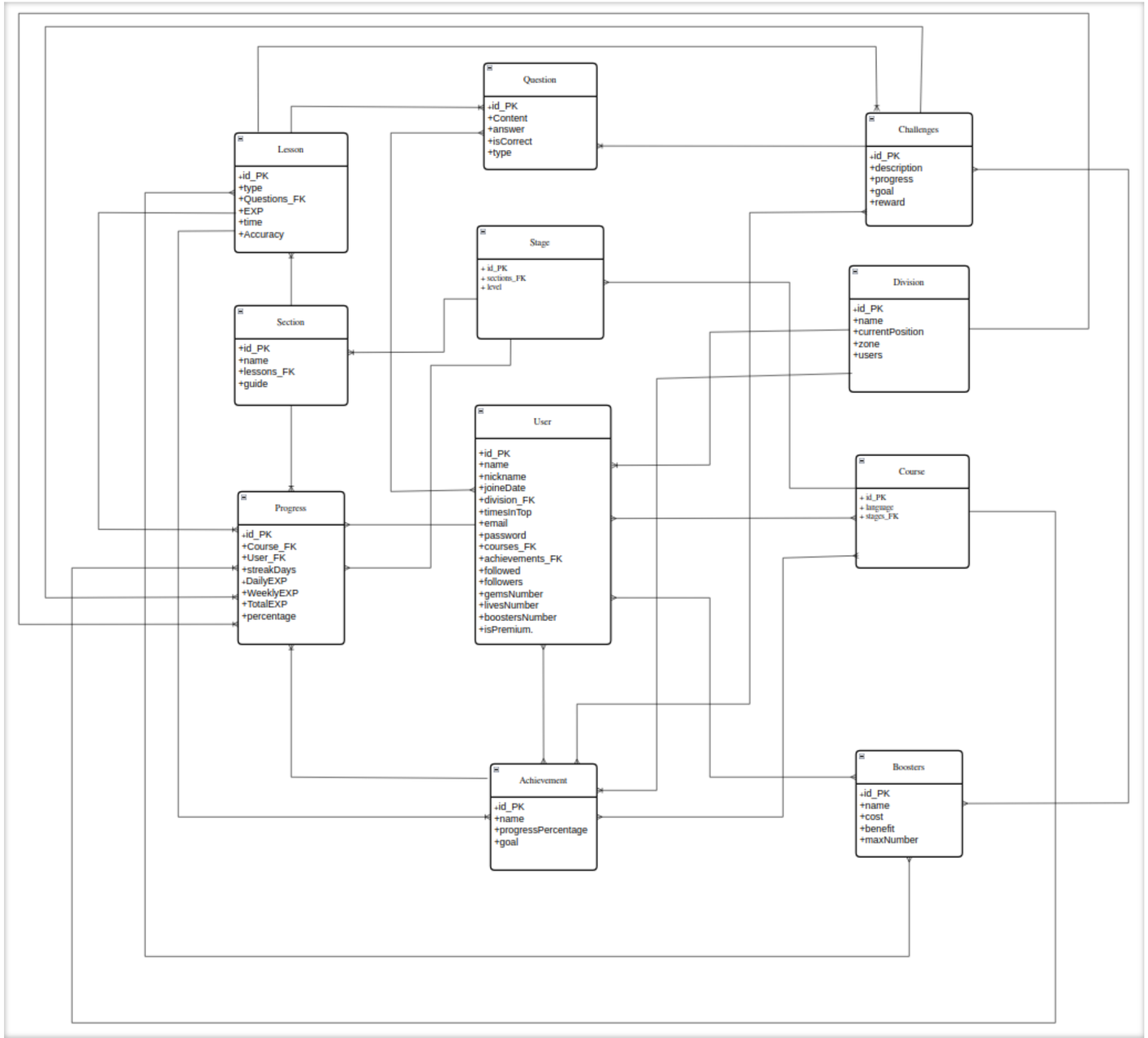
Step 4:Define Relationships

We used a table with the entities, marking with a color which relationships they had with each other.

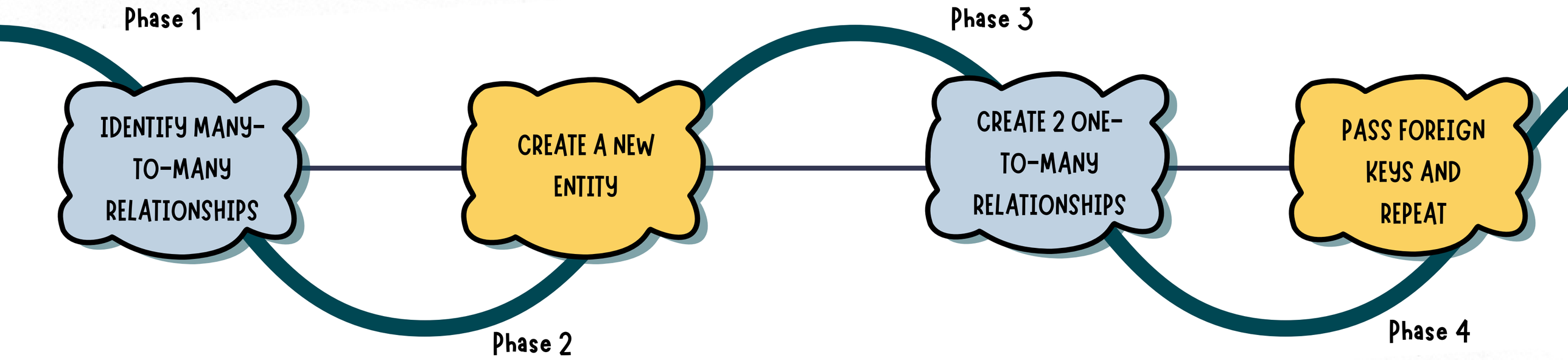
STEP 5: DEFINE RELATIONSHIPS TYPES

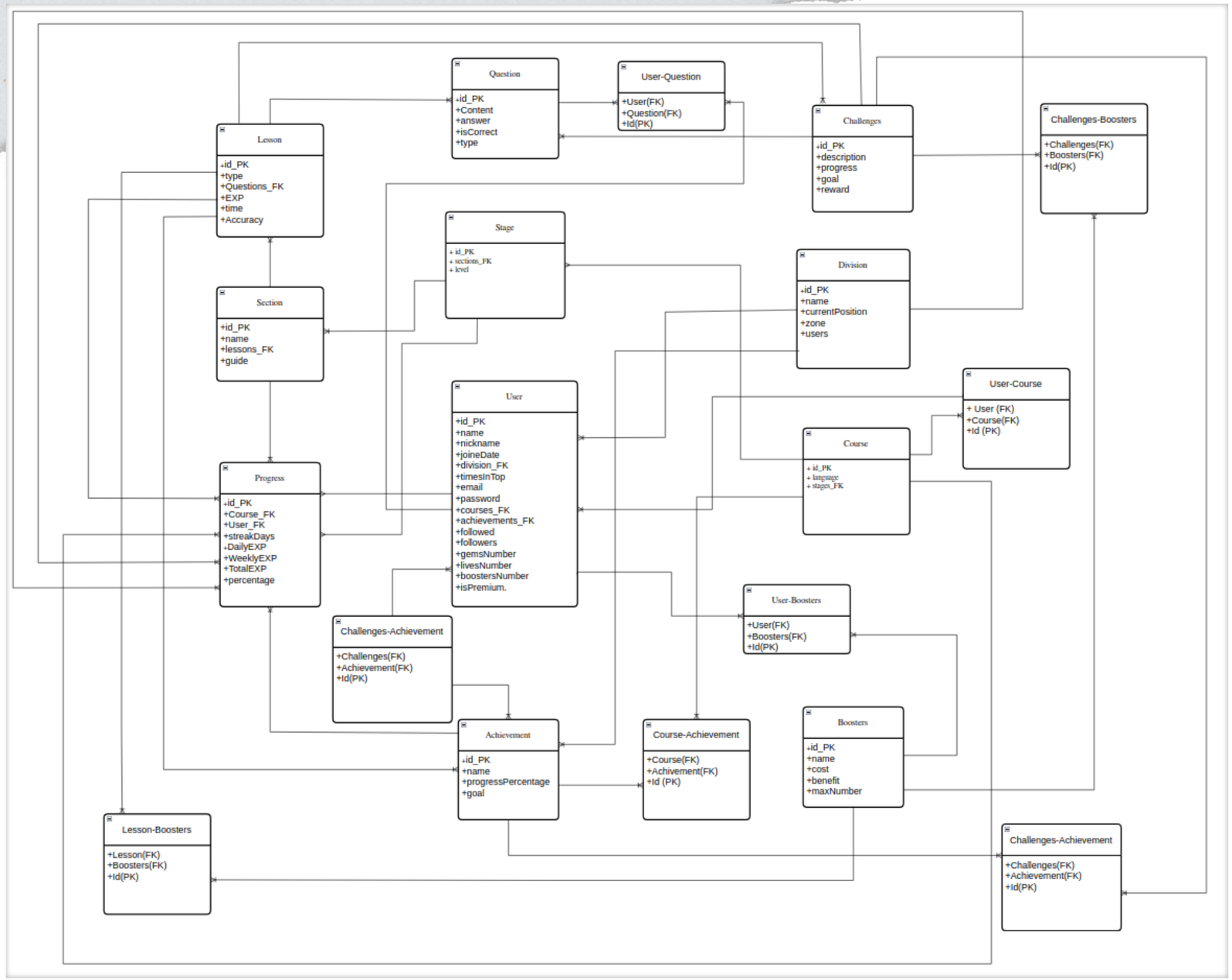
Reviewing the previous step, we focus on carefully reviewing each of the relationships to determine its type, that is, if it is one to one, one to many, or many to many.

STEP 6: FIRST ENTITY-RELATIONSHIP DRAW



STEP 7: FIRST SPLIT MANY-TO-MANY RELATIONSHIPS





STEP 8: SECOND ENTITY-RELATIONSHIP DRAW

STEP 9: GET DATA-STRUCTURE

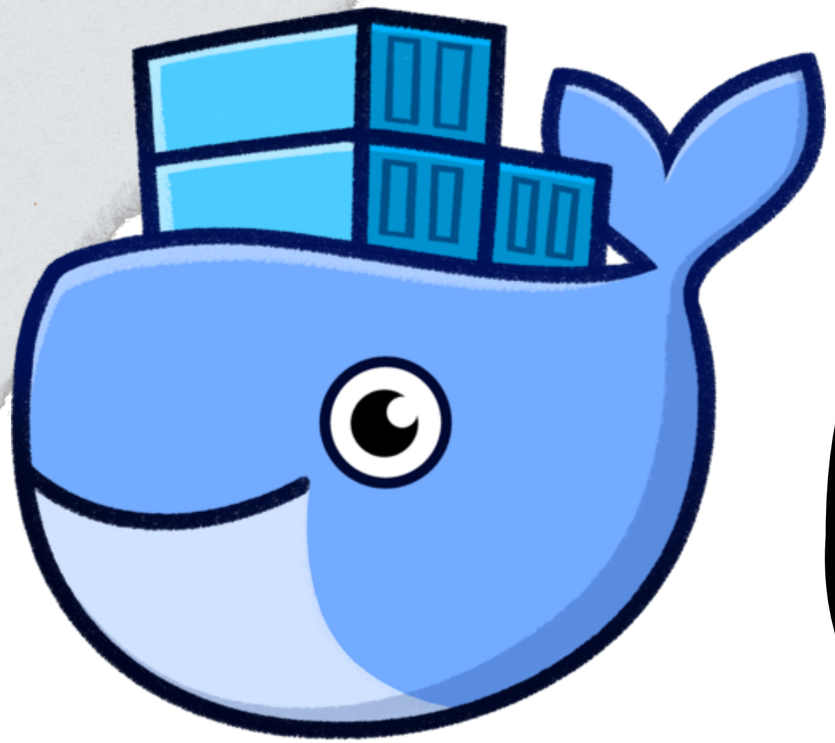
In this step what was done was to assign the data type corresponding to each of the attributes for our entities

Step 10:Define Constraints and Properties of Data

We used a table with the entities, marking with a color which relationships they had with each other.

ONCE THE ERM IS FINISHED

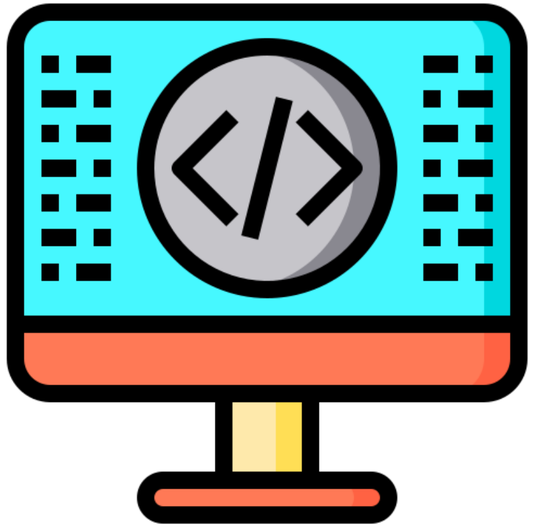
the next step is to set up a structured environment for database deployment using Docker. This ensures a consistent and reproducible setup across different systems.



STEP 1: CREATE FILE

A Docker Compose file is created to define the database services, specifying images, ports, volumes, and environment variables.

```yaml
mysql-db:
  image: mysql:latest
  restart: always
  environment:
    MYSQL_ROOT_PASSWORD: duolingo_root
    MYSQL_DATABASE: DuolingoDB
    MYSQL_USER: duolingo_user
    MYSQL_PASSWORD: duolingo_password
  ports:
    - "3306:3306"
  volumes:
    - mysql_data:/var/lib/mysql
  networks:
    - duolingo_network
```



STEP 2: SET UP SERVICES

```yaml
postgres-db:
  image: postgres:latest
  restart: always
  environment:
    POSTGRES_USER: duolingo_user
    POSTGRES_PASSWORD: duolingo_password
    POSTGRES_DB: DuolingoDB
  ports:
    - "5432:5432"
  volumes:
    - postgres_data:/var/lib/postgresql/data
  networks:
    - duolingo_network
```

RELATIONAL ALGEBRA

In the project, relational algebra was used to validate the database model, ensuring it could efficiently handle queries like identifying premium users or tracking progress.

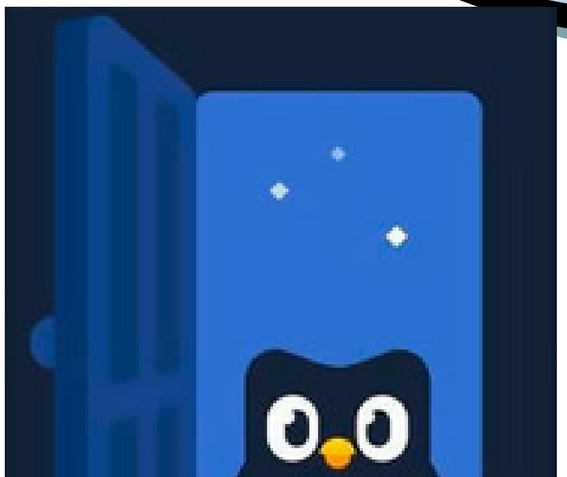
$$\pi_{\text{nickname, email}}\left(\sigma_{\text{isPremium}=\text{True}}(\text{User})\right)$$

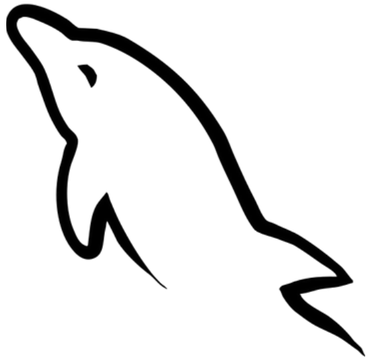
$$\pi_{id\_PK,\, name}\left(\sigma_{timesInTop \geq 3}(\text{User})\right)$$

SQL CREATION

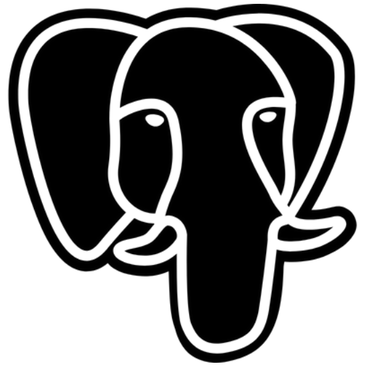
the SQL script was created based on the finalized Entity-Relationship Model (ERM) to ensure a structured and consistent database implementation

- MySQL and PostgreSQL



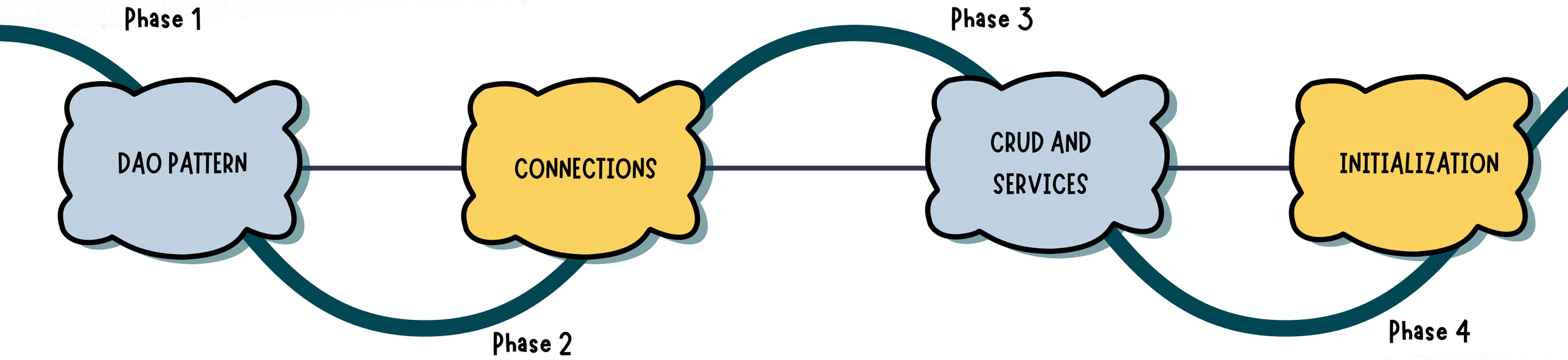


```sql
CREATE TABLE IF NOT EXISTS User (
    id_PK INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(50) NOT NULL,
    nickname VARCHAR(50) NOT NULL UNIQUE,
    joinDate DATE DEFAULT (CURRENT_DATE()),
    division_FK INT,
    timesInTop INT DEFAULT 0,
    email VARCHAR(75) NOT NULL UNIQUE,
    password VARCHAR(32) NOT NULL,
    courses_FK INT NOT NULL,
    achievements_FK INT,
    followed INT DEFAULT 0,
    followers INT DEFAULT 0,
    gemsNumber INT DEFAULT 0,
    livesNumber INT DEFAULT 5,
    boostersNumber INT DEFAULT 0,
    isPremium BOOLEAN DEFAULT FALSE,
    FOREIGN KEY (division_FK) REFERENCES Division(id_PK),
    FOREIGN KEY (courses_FK) REFERENCES Course(id_PK),
    FOREIGN KEY (achievements_FK) REFERENCES Achievement(id_PK)
);
```



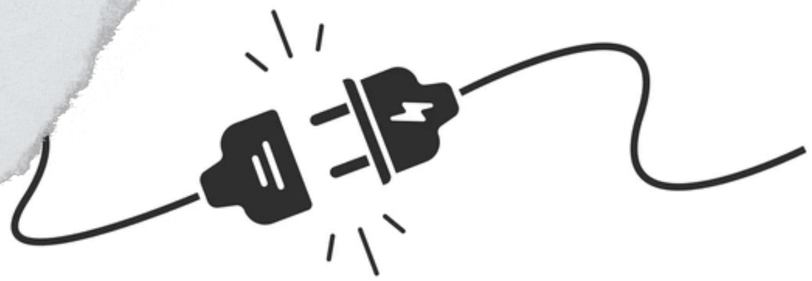
```sql
CREATE TABLE IF NOT EXISTS "User" (
    id_PK SERIAL PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    nickname VARCHAR(50) NOT NULL UNIQUE,
    joinDate DATE DEFAULT CURRENT_DATE,
    division_FK INT,
    timesInTop INT DEFAULT 0,
    email VARCHAR(75) NOT NULL UNIQUE,
    password VARCHAR(32) NOT NULL,
    courses_FK INT NOT NULL,
    achievements_FK INT,
    followed INT DEFAULT 0,
    followers INT DEFAULT 0,
    gemsNumber INT DEFAULT 0,
    livesNumber INT DEFAULT 5,
    boostersNumber INT DEFAULT 0,
    isPremium BOOLEAN DEFAULT FALSE,
    FOREIGN KEY (division_FK) REFERENCES Division(id_PK),
    FOREIGN KEY (courses_FK) REFERENCES Course(id_PK),
    FOREIGN KEY (achievements_FK) REFERENCES Achievement(id_PK)
);
```

PYTHON APPLICATION



DAO PATTERN

the DAO pattern was used to separate the database access logic from the business logic. By example, in the UserDAO class we define the same attributes as in the database



CONNECTIONS

We created a package 'Connections' where we setted up all the connections between the application and the Database, one file for MySql and one file for PostgreSql

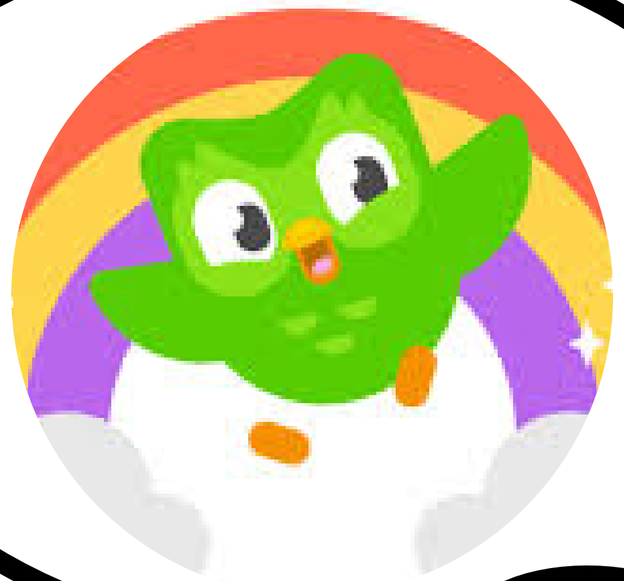
CRUD

Here we created the CRUD operations (create, read, update, delete) per each table in our Database

```
def get_all(self) -> List[UserDAO]:

    query = """
        SELECT id_PK, name, nickname, email, isPremium
        FROM User;
    """
    return self.db_connection.get_many(query)
```

SERVICES

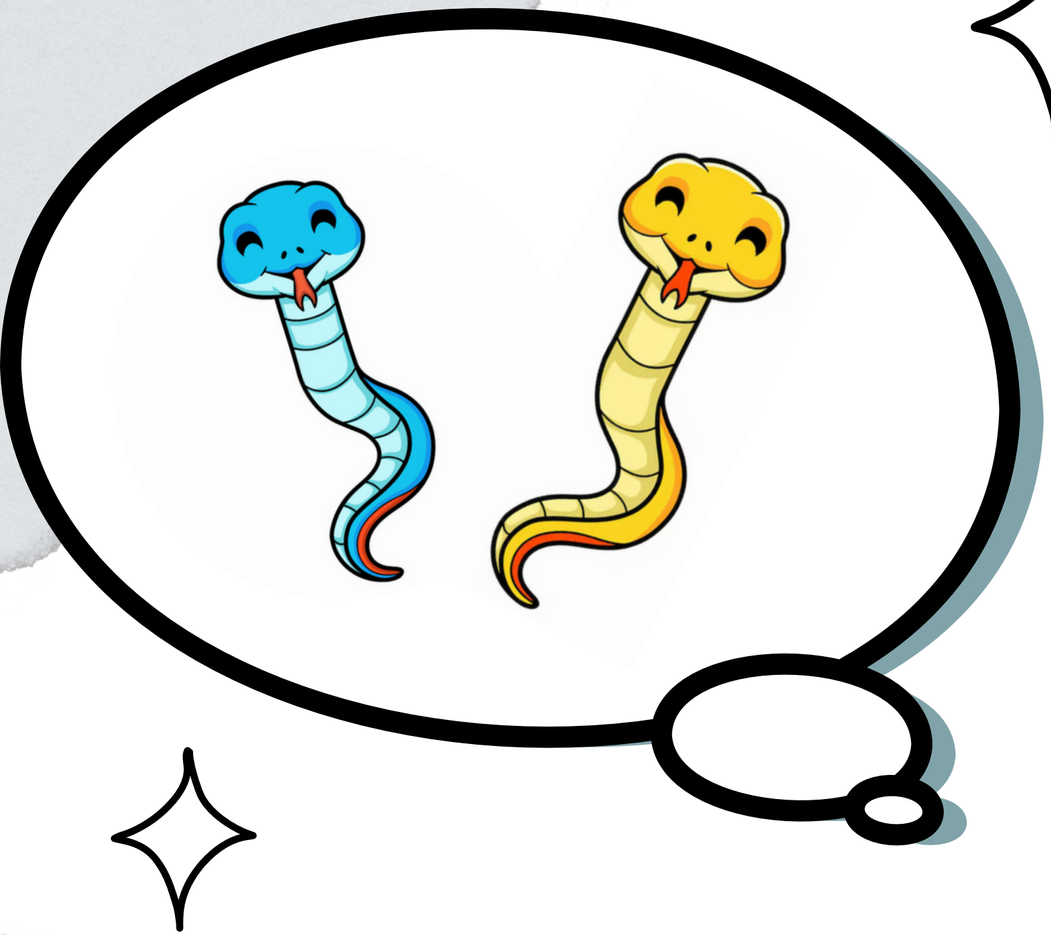


In this package we handle all the web services in the classes by giving the methods an the route that should be used

```python
@router.get("/Achievements/get_all", response_model=List[AchievementDAO])
def get_all_Achievements():

    Achievements = Achievement.get_all()
    return Achievements
```

INITIALIZATION



Here we created the CRUD operations (create, read, update, delete) per each table in our Database

```
def get_all(self) -> List[UserDAO]:

    query = """
        SELECT id_PK, name, nickname, email, isPremium
        FROM User;
    """
    return self.db_connection.get_many(query)
```

Don't forget to take your lessons or your pet will be kidnapped

THANK YOU

WE HOPE YOU LIKED IT