Welcome to the CoGrammar ORM

The session will start shortly...

Questions? Drop them in the chat. We'll have dedicated moderators answering questions.

CoGrammar

Software Engineering Session Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly. **(Fundamental British Values: Mutual Respect and Tolerance)**
- No question is daft or silly - **ask them!**
- There are **Q&A sessions** midway and at the end of the session, should you wish to ask any follow-up questions. Moderators are going to be answering questions as the session progresses as well.
- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Academic Sessions. You can submit these questions here: **Questions**

CoGrammar

Software Engineering Session Housekeeping cont.

- For all **non-academic questions**, please submit a query: www.hyperiondev.com/support

- Report a **safeguarding** incident: www.hyperiondev.com/safeguardreporting

- We would love your **feedback** on lectures: Feedback on Lectures

Skills Bootcamp
8-Week Progression Overview

Fulfil 4 Criteria to Graduation

✅ Criterion 1: Initial Requirements

- ***Timeframe:*** First 2 Weeks
- ***Guided Learning Hours (GLH):*** Minimum of 15 hours
- ***Task Completion:*** First four tasks

✅ Criterion 2: Mid-Course Progress

- ***Guided Learning Hours (GLH):*** **60**
- ***Task Completion:*** 13 tasks

CoGrammar

Skills Bootcamp Progression Overview

✅ Criterion 3: Course Progress

- ***Completion:*** All mandatory tasks, including Build Your Brand and resubmissions by study period end
- ***Interview Invitation:*** Within 4 weeks post-course
- ***Guided Learning Hours:*** Minimum of 112 hours by support end date (10.5 hours average, each week)

✅ Criterion 4: Demonstrating Employability

- ***Final Job or Apprenticeship Outcome:*** Document within 12 weeks post-graduation
- ***Relevance:*** Progression to employment or related opportunity

CoGrammar

SKILLS FOR LIFE
SKILLS BOOTCAMPS
Department for Education

CoGrammar

ORM

June 2024

Learning Objectives

- Define and Explain Object-Relational Mapping (**ORM**)
- Set up a Simple **Database Connection**
- Create **Python Classes** for Database **Models**
- Demonstrate Basic **CRUD Operations** with an ORM
- Implement **Relationships** between Database Models
- Explain the **Concept** of **Database Migrations**

CoGrammar

Poll

1. Which SQL clause is used to filter rows based on a condition?

   a. SELECT
   b. FROM
   c. WHERE
   d. FILTER

Poll

2. How do you calculate the total salary for each department using SQL?

a. SELECT SUM(salary) FROM employees GROUP BY department;
b. SELECT TOTAL(salary) FROM employees GROUP BY department;
c. SELECT MAX(salary) FROM employees GROUP BY department;
d. SELECT MIN(salary) FROM employees GROUP BY department;

CoGrammar

Poll

3. How do you find the second highest salary in a table using SQL?

a. SELECT MAX(salary) FROM employees WHERE salary < (SELECT MAX(salary) FROM employees);
b. SELECT MIN(salary) FROM employees WHERE salary > (SELECT MAX(salary) FROM employees);
c. SELECT DISTINCT salary FROM employees ORDER BY salary DESC LIMIT 1,1;
d. SELECT AVG(salary) FROM employees WHERE salary > (SELECT MIN(salary) FROM employees);

CoGrammar

Introduction

Intuition

Object-Relational Mappings (**ORM**s) are designed to be **database-agnostic**, allowing developers to interact with different databases without requiring **specific SQL dialects**. This means that **the same ORM code** can be used to interact with multiple databases, such as **MySQL**, **PostgreSQL**, and **SQL Server**, **without** needing to write **database-specific SQL**.

By generating SQL queries automatically, ORMs simplify database interactions and **reduce the need for manual SQL writing**. This approach improves productivity and makes it easier to switch between different databases if needed. With ORMs, developers can **focus on application logic** rather than low-level database operations, leading to more efficient and maintainable code.

CoGrammar

ORM

CoGrammar

Traditional Database Interaction

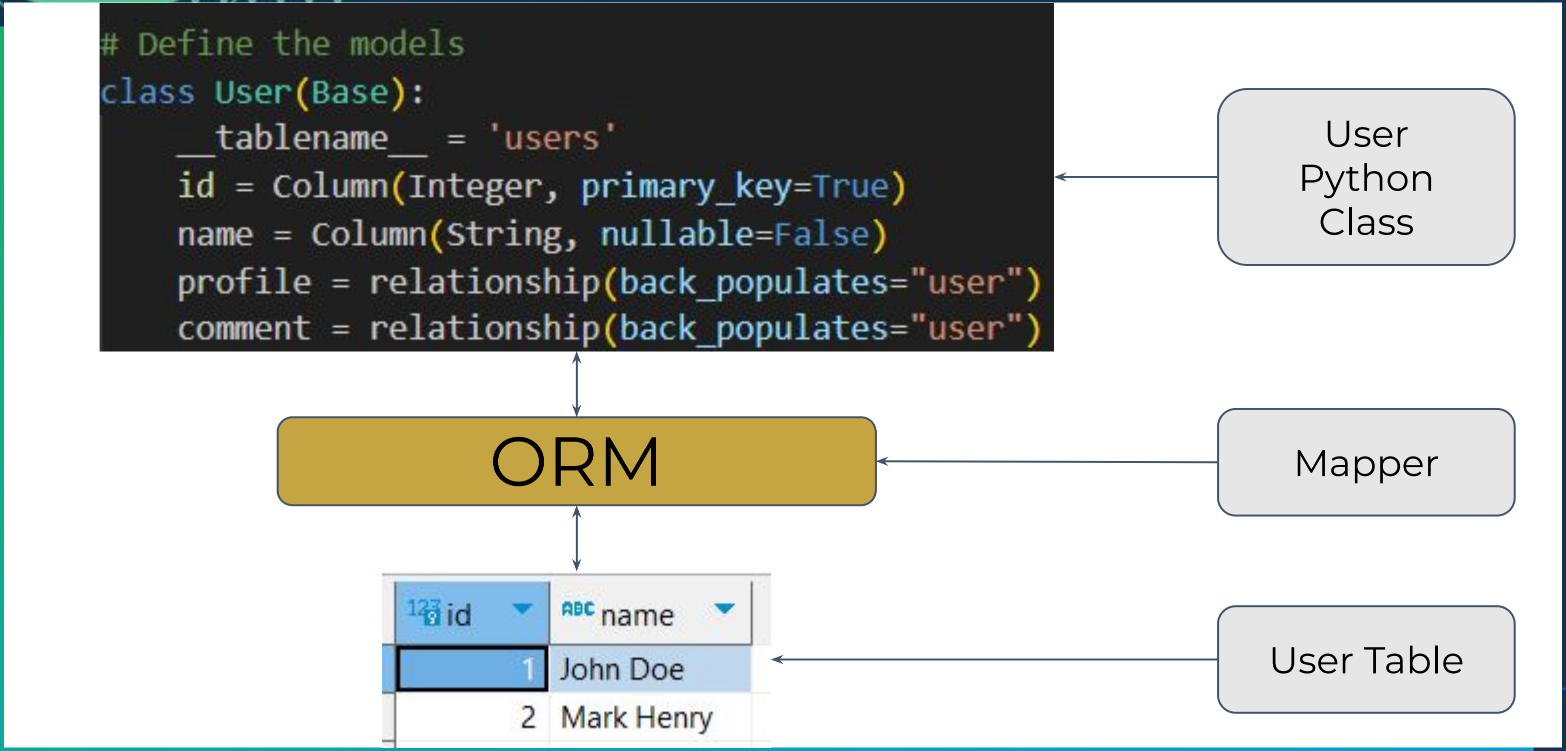
- **Manual SQL:** Writing SQL queries directly to interact with databases.
  - Example:
    - **SELECT** * **FROM** users **WHERE** age > 30;
  - Drawbacks: Verbose, error-prone, and tightly coupled to database specifics.

ORM

- **ORM (Object-Relational Mapping):**
  - A technique to interact with databases using an object-oriented programming (OOP) language.
  - Simplifies database operations by mapping database tables to class structures in code.
  - We'll use **SQLAlchemy** ORM tool

CoGrammar

ORM



```python
# Define the models
class User(Base):
    __tablename__ = 'users'
    id = Column(Integer, primary_key=True)
    name = Column(String, nullable=False)
    profile = relationship(back_populates="user")
    comment = relationship(back_populates="user")
```

id	name
1	John Doe
2	Mark Henry

CoGrammar

ORM as an Abstraction Layer

- **Abstraction:**
  - Provides a layer between application code and the database.
  - Enables developers to work with high-level objects instead of raw SQL.

ORM as an Abstraction Layer

```
session.execute(select(User).where(User.age == 'John Doe')).all()
```

- Example:
  - **select(User).where(User.age > 30).all()**

SELECT * FROM USERS WHERE AGE > 30

CoGrammar

Benefits of Using ORM

- **Easier Data Handling:**
  - Write less code and avoid SQL syntax errors.
  - Focus on business logic rather than database details.
- **Security:**
  - Reduces risk of SQL injection attacks.
  - Automatically escapes query inputs.
- **Code Readability and Maintenance:**
  - Improved readability with clear class definitions and relationships.
  - Easier to maintain and update as database schema evolves.
- **Productivity:**
  - Faster development with less boilerplate code.
  - Auto-generates SQL queries from code.

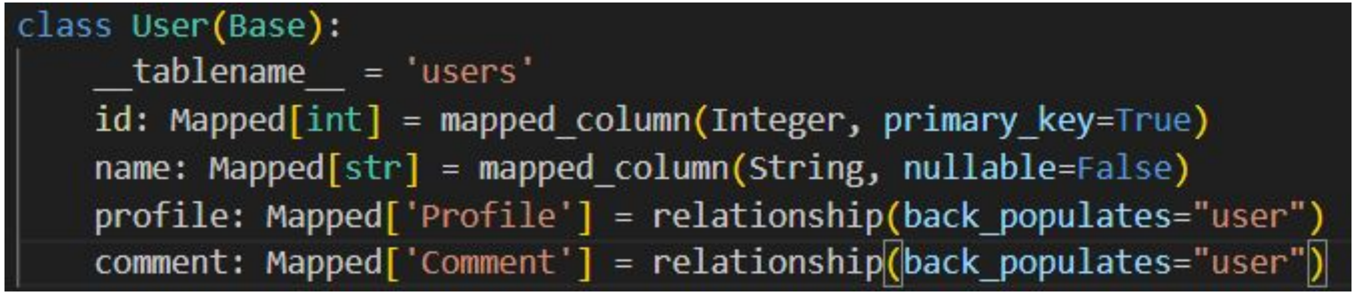
CoGrammar

Migrations

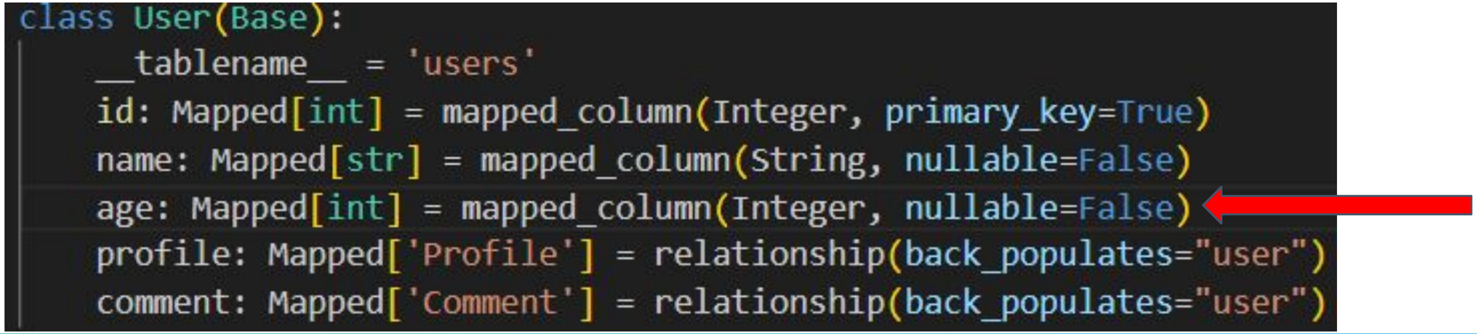
Migrations are a way of propagating changes you make to your models (adding a field, deleting a model, etc.) into your database schema. They’re designed to be mostly automatic, but you’ll need to know when to make migrations, when to run them, and the common problems you might run into. Django's Migrations

Migrations

Let's assume that you have this Python Class, we need to add **age**:



```
class User(Base):
    __tablename__ = 'users'
    id: Mapped[int] = mapped_column(Integer, primary_key=True)
    name: Mapped[str] = mapped_column(String, nullable=False)
    profile: Mapped['Profile'] = relationship(back_populates="user")
    comment: Mapped['Comment'] = relationship(back_populates="user")
```

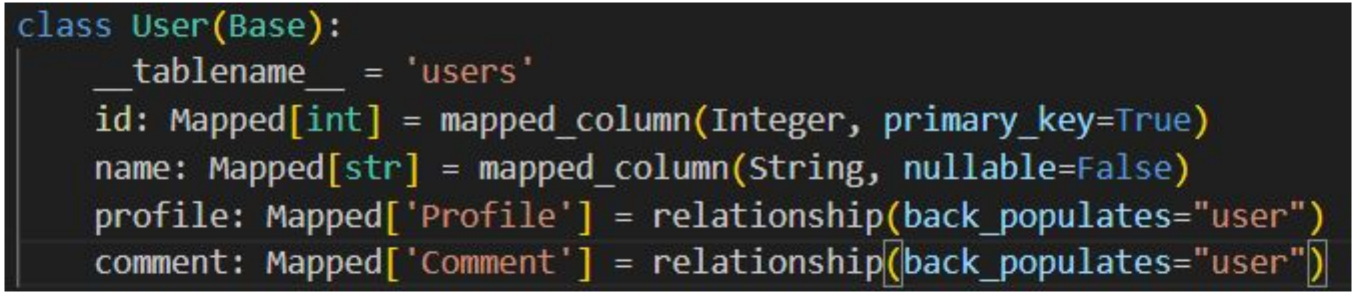


```
class User(Base):
    __tablename__ = 'users'
    id: Mapped[int] = mapped_column(Integer, primary_key=True)
    name: Mapped[str] = mapped_column(String, nullable=False)
    age: Mapped[int] = mapped_column(Integer, nullable=False)
    profile: Mapped['Profile'] = relationship(back_populates="user")
    comment: Mapped['Comment'] = relationship(back_populates="user")
```

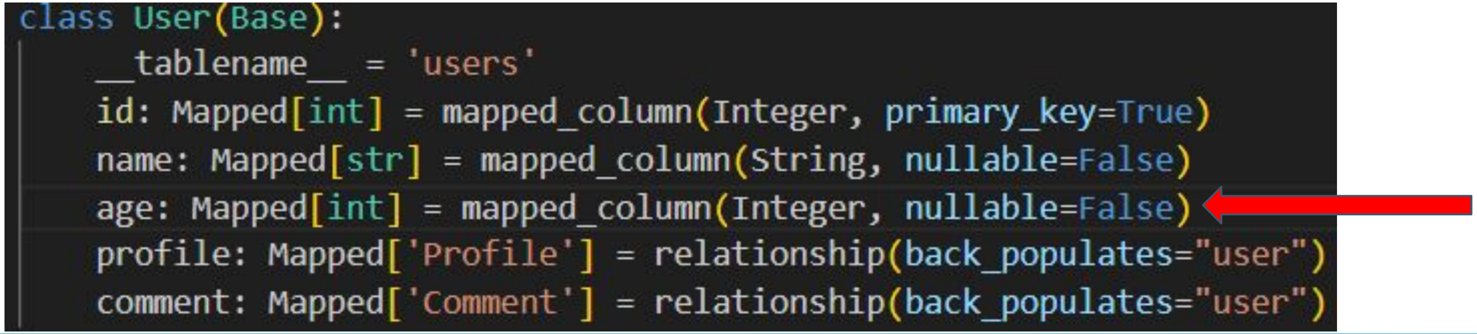
CoGrammar

Migrations

Let’s assume that you have this Python Class, we need to add **age**:



```python
class User(Base):
    __tablename__ = 'users'
    id: Mapped[int] = mapped_column(Integer, primary_key=True)
    name: Mapped[str] = mapped_column(String, nullable=False)
    profile: Mapped['Profile'] = relationship(back_populates="user")
    comment: Mapped['Comment'] = relationship(back_populates="user")
```

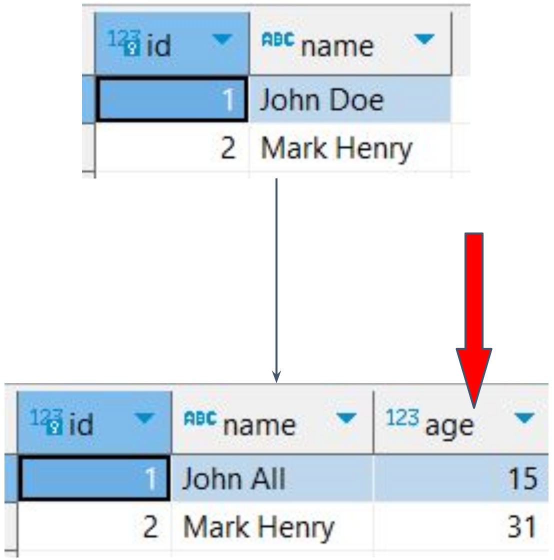


```python
class User(Base):
    __tablename__ = 'users'
    id: Mapped[int] = mapped_column(Integer, primary_key=True)
    name: Mapped[str] = mapped_column(String, nullable=False)
    age: Mapped[int] = mapped_column(Integer, nullable=False)
    profile: Mapped['Profile'] = relationship(back_populates="user")
    comment: Mapped['Comment'] = relationship(back_populates="user")
```

CoGrammar

Migrations

Now we have the new column age **added**:



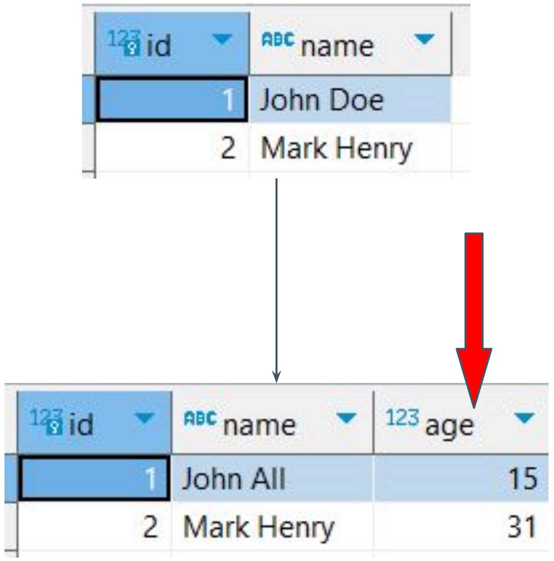
id	name
1	John Doe
2	Mark Henry

id	name	age
1	John All	15
2	Mark Henry	31

CoGrammar

Migrations

Now we have the new column age **added**:



id	name
1	John Doe
2	Mark Henry

id	name	age
1	John All	15
2	Mark Henry	31

CoGrammar

SQLAlchemy

- **SQLAlchemy** is a library used to interact with a wide variety of databases. It enables you to create data models and queries in a manner that feels like normal Python classes statements.

- SQLAlchemy needs to be installed first. To do so, we have to use following code at Terminal or CMD.

pip install sqlalchemy

CoGrammar

Let’s take a break

CoGrammar

Let’s get coding

CoGrammar

Poll

1. What is the primary benefit of using an ORM in a Python application?

   a. Encrypts database communication for security.

   b. Provides a simpler way to interact with relational databases.

   c. Automatically optimises database queries.

   d. Converts Python objects directly into database rows.

CoGrammar

Poll

2. Which of the following is NOT a core function of an ORM?

a. Defining relationships between database tables
b. Manually writing complex SQL queries (Correct)
c. Performing CRUD operations (Create, Read, Update, Delete)
d. Mapping Python objects to database tables

CoGrammar

Poll

3. What is the main purpose of defining a Python class when using an ORM?

a. To create a user interface element for interacting with the database.

b. To represent the structure of a database table with its columns.

c. To store and manage database connection details.

d. To perform complex mathematical calculations on database data.

CoGrammar

Summary

- **ORM Basics**: Simplify database interaction by mapping objects to tables (and vice versa).
- **CRUD Operations with ORMs**: Easily Create, Read, Update, and Delete data using ORM functionalities.
- **Model Relationships**: Manage connections between tables using one-to-one, one-to-many, and many-to-many relationships.
- **ORM Models**: Define database structure with Python classes, representing tables and their columns.
- **Database Migrations (Concept)**: Keep your database schema in sync with your Python models using migrations (explained further in resources).

References

- Full Stack Python’s ORM overview
- Real Python’s SQLAlchemy tutorial
- Özgür Özkök’s beginner’s guide to ORM in Python

Thank you for attending

SKILLS FOR LIFE

SKILLS BOOTCAMPS

Department for Education

CoGrammar