# 0. Topics

Tuesday, May 13, 2025 11:16 PM

1. **Database Basics**
2. **SQLAlchemy Basics**
3. **CRUD app using FastAPI & SQLAlchemy**

# 1. Database Basics

Tuesday, May 13, 2025 11:22 PM

## What is a Database?

- A database is a structured collection of data that can be easily accessed, managed, and updated
- In web apps, databases store persistent data (which doesn't get lost)

## What are Relational Databases?

These are databases that store data in tables with rows and columns, where each row is a record, and each column is a field

### Popular Relational DBs:

| DATABASE | FEATURES |
|---|---|
| SQLite | Lightweight, file-based, no server required. Great for prototyping. |
| PostgreSQL | Open-source, full-featured, robust. Excellent for production use. |
| MySQL | Widely used, fast and reliable. Used in many production apps. |

## Why integrate it with FastAPI?

1. **Persistent Data Storage:**
   - APIs often need to store and persist data across sessions
   - Without a database, all data would live temporarily in memory (RAM) and be lost once the app restarts

2. **Real-World Use Cases:**
   - Most web apps require reliable, structured, and persistent data storage — exactly what databases are built for

3. **Seamless Backend Operations:**
   - Using SQLAlchemy with FastAPI enables users to map these HTTP operations directly to database operations in a clean and Pythonic way
   - Users can implement clean, readable, Python-based models that map to your database
   - It supports asynchronous interactions, aligning perfectly with FastAPI's async-first design

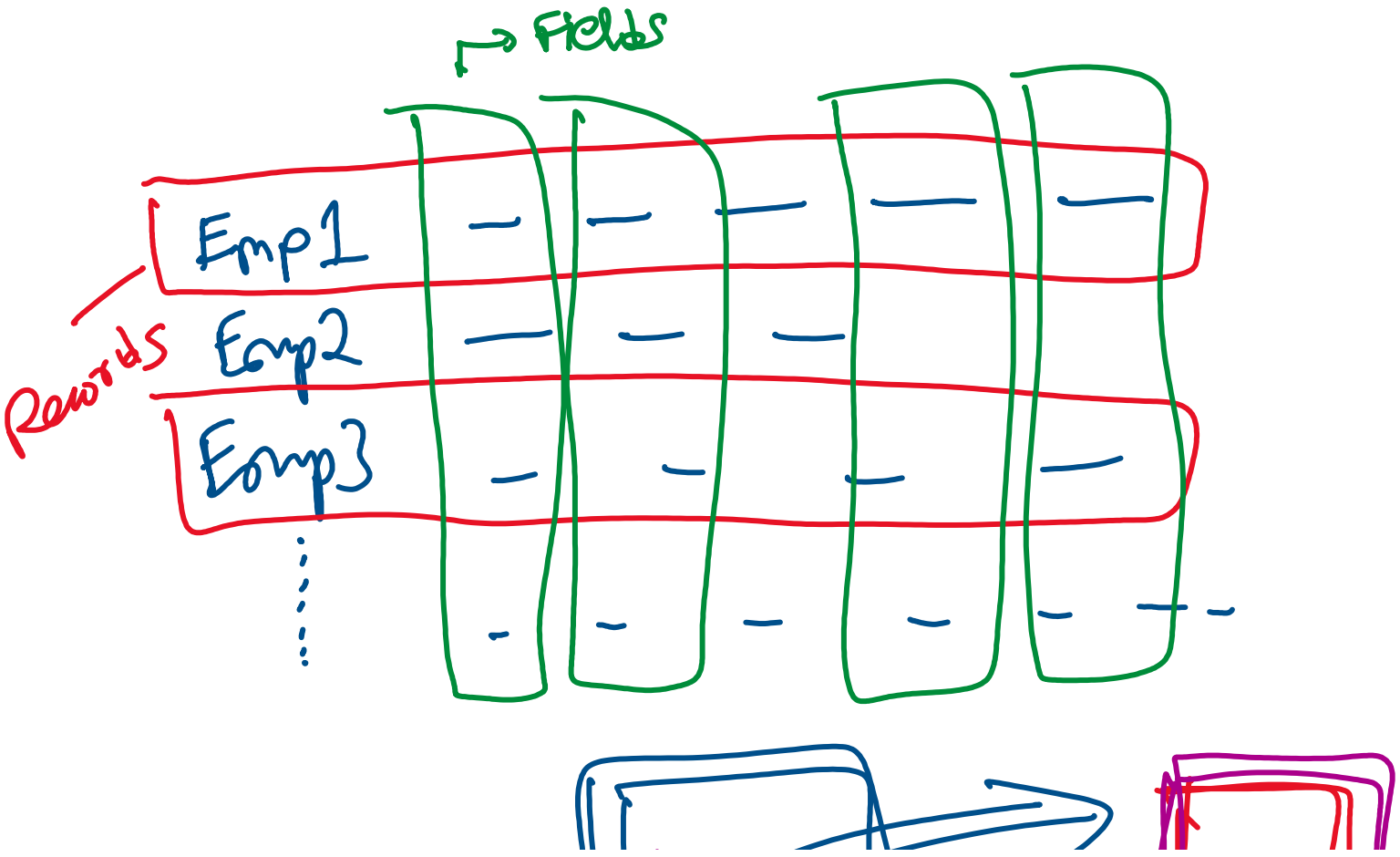
- Users can utilize FastAPI's dependency injection to cleanly manage DB sessions

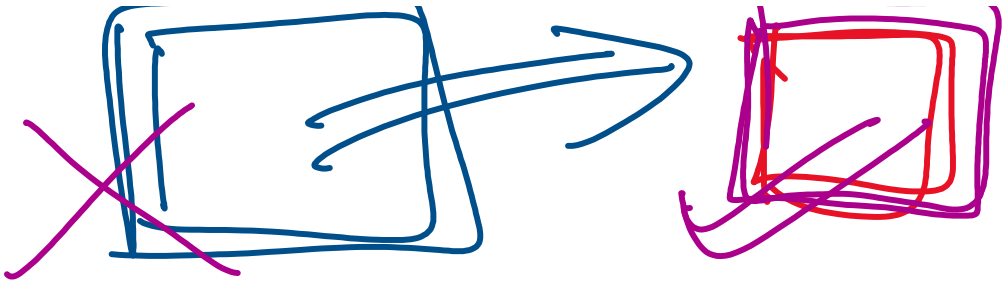
### 4. Enables Feature-Rich Applications:

- With a database in place, your FastAPI application can:-
  - Add authentication and authorization
  - Track historical data
  - Perform analytics
  - Manage file uploads and metadata

### 5. Security and Data Integrity:

- Relational databases provide:
  - Data integrity constraints
  - Transactions to ensure consistency
  - Access control and permissions
- This is essential for building secure and reliable APIs





# 2. SQLAlchemy Basics

Tuesday, May 13, 2025 11:39 PM

## 1. What is SQLAlchemy ?

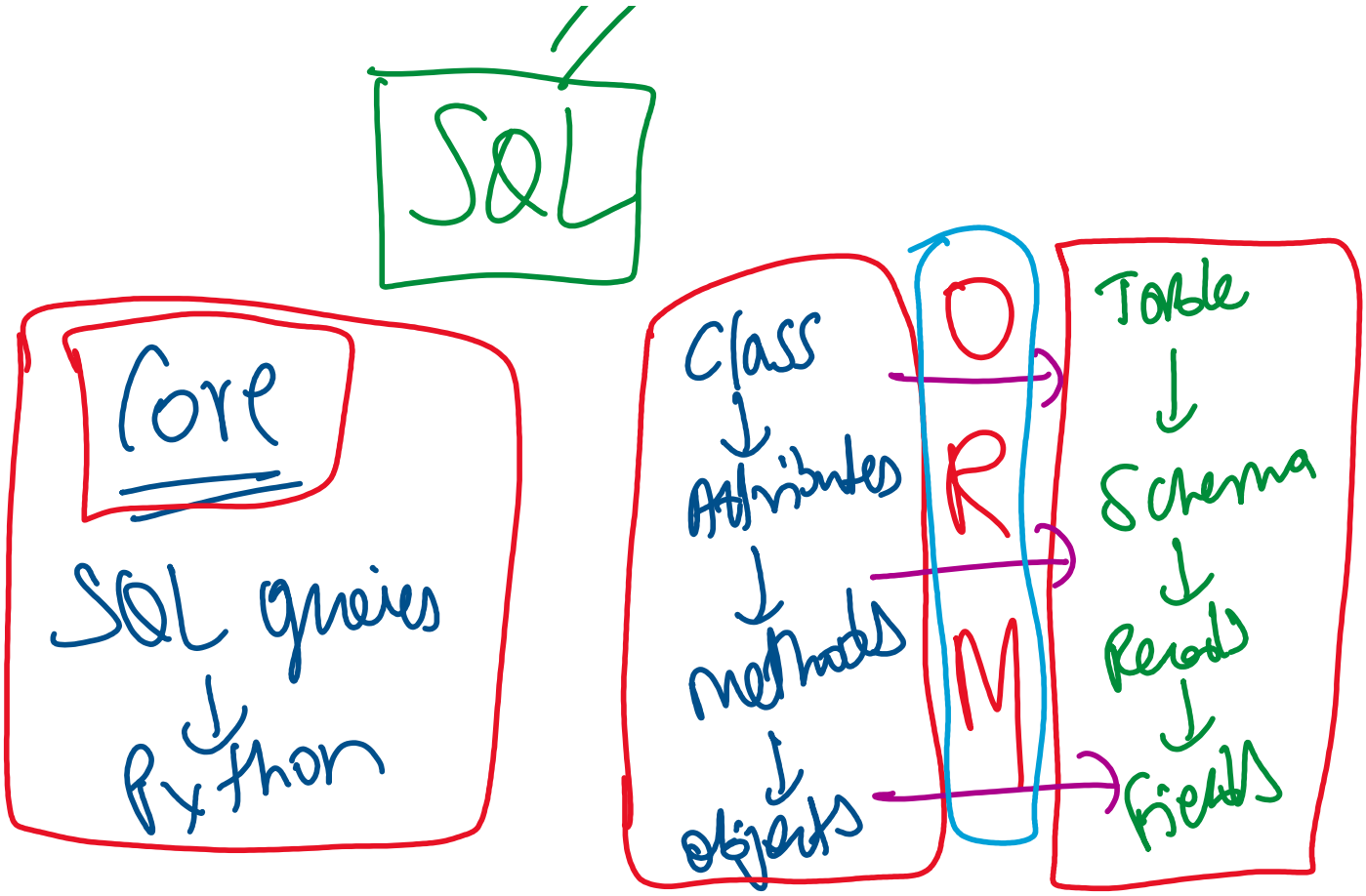
- SQLAlchemy is a powerful Python SQL toolkit and Object Relational Mapper (ORM) that helps Python applications interact with relational databases
- SQLAlchemy Has Two Main Parts:
  - **SQLAlchemy Core:**
    - A lower-level SQL expression language that lets users build SQL queries using Python
    - Involves writing raw SQL, but in Pythonic syntax
  - **SQLAlchemy ORM (Object Relational Mapper):**
    - A higher-level tool that lets users map Python classes to database tables
    - Users write Python code, and SQLAlchemy handles the SQL under the hood

## 2. Why Use SQLAlchemy?

| FEATURE | DESCRIPTION |
|---|---|
| **ORM** | Easily map Python classes to database tables |
| **Cross-DB Compatibility** | Works with PostgreSQL, MySQL, SQLite, etc. |
| **Security** | Protects against SQL injection |
| **Asynchronous Support** | Plays well with async in FastAPI and modern Python |
| **Mature & Well-Documented** | Battle-tested and production-ready |

## 3. Installation:

- pip install sqlalchemy sqlalchemy[asyncio]


SQL
Core
SQL queries
Python
Class
Attributes
methods
objects
O
R
M
Table
Schema
Records
Fields

# 3. CRUD App

Wednesday, May 14, 2025 5:24 AM

**This app is a simple REST API to manage employee records, implementing CRUD operations using:**

- **FastAPI** — Web framework for building APIs
- **SQLAlchemy** — ORM (Object Relational Mapper) for database interaction
- **SQLite** — Lightweight database for storage
- **Pydantic** — Data validation and serialization

## App Structure:

- **crud-app**
  - database.py
  - models.py
  - schemas.py
  - crud.py
  - main.py

# 3.1 - database.py

Saturday, May 17, 2025 12:48 PM

- Helps setup database connection and provide foundational components for ORM
- Centralizes DB setup, making it reusable across the app for sessions and models

**create_engine():**

- Establishes the connection to the database
- Here, it connects to a SQLite file named test.db

**connect_args:**

- SQLite-specific to allow connection sharing across threads

**sessionmaker:**

- Helps create new database sessions
- Each session represents a transactional scope to the DB
- **autoflush=False**:
  - SQLAlchemy will not automatically flush changes to the DB unless explicitly committed or refreshed
- **autocommit=False**:
  - Disables automatic commit after each query
  - Commit manually to control transactions

**declarative_base():**

- Creates a base class for models to inherit from, linking the Python classes with DB tables

# 3.2 - models.py

Saturday, May 17, 2025 1:02 PM

Defines the **Employee** model, mapping the Python class to the **employees** table in the DB

- Helps auto-generate fields like **ID**
- Can make different attributes optional or use default values for specific tasks as required
- Helps add validations specific to one operation without affecting others
- Having explicit classes for each action makes the code self-explanatory
- Makes it easier to debug and extend

# 3.3 - schemas.py

Saturday, May 17, 2025 1:04 PM

- Defines Pydantic models (schemas) for request validation and response formatting
- Promotes data consistency and validation against invalid inputs

**orm_mode = True:**

- Allows Pydantic to read data directly from ORM objects (SQLAlchemy models)
- Enables smooth conversion to JSON

| SCHEMA CLASS | PURPOSE |
|---|---|
| **EmployeeBase** | Shared fields for DRY (Don't Repeat Yourself) |
| **EmployeeCreate** | Input for creating a new employee |
| **EmployeeUpdate** | Input for updating existing employee |
| **EmployeeOut** | Output for returning employee data |

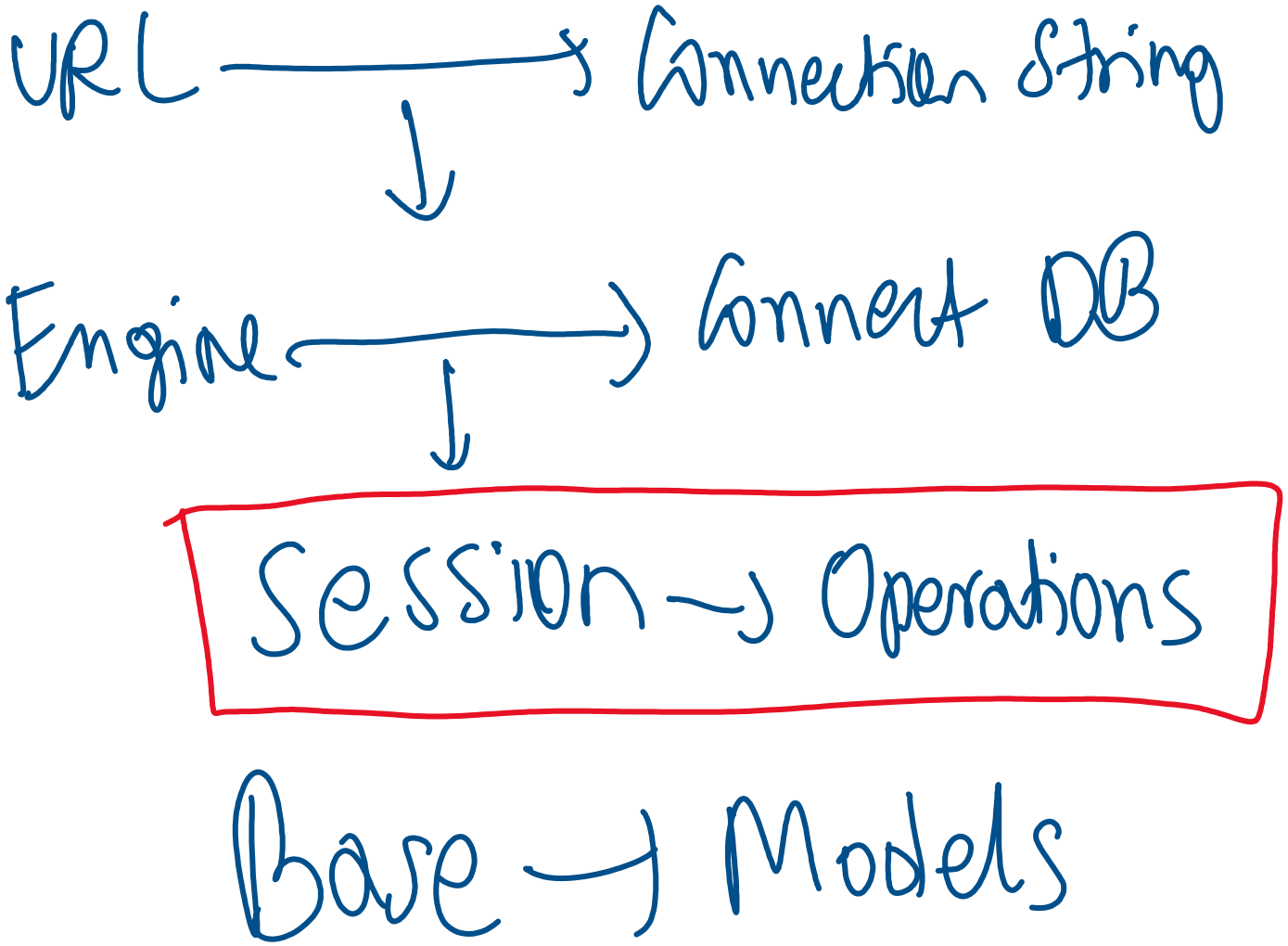
# 3.4 - crud.py

Saturday, May 17, 2025 1:07 PM

- This module encapsulates all database operations for **Employee**
- Abstracts away raw DB queries from the API routes, keeping the code modular
- Keeps DB logic separate and reusable, making the app cleaner and easier to maintain

**refresh():**

- reloads the object from DB to get autogenerated fields (**id**)



# 3.5 - main.py

Saturday, May 17, 2025 1:07 PM

- This is where the API is defined, exposing endpoints for the end users
- Denotes the entry point of the API server
- Defines the FastAPI app and all associated routes

**create_all(bind=engine):**

- creates DB tables based on your models if they don't exist

**get_db():**

- Dependency to provide a DB session to routes
- Uses the **yield** keyword to make it a generator
- Helps FastAPI manage the opening and closing of DB connections cleanly