MAGLAWAY, MARK S. ACTIVITY 8
BSIT 2101

Questions (Explain the Code)

1. What does the Student class represent in the database?

⇒The Student class represents the data model for a single record within the application's database, specifically mapping to a single table. This model-to-table mapping is a core function of the SQLAlchemy ORM (Object-Relational Mapper), allowing developers to interact with database structures using Python classes and objects instead of raw SQL code. Every object created from this Student class will correspond to one row in the database table.

The class defines the structure and schema of the student table by declaring three columns using db.Column. The id column is defined as an integer and serves as the primary_key=True, which uniquely identifies each student record and is automatically managed by the database. The other columns, name and grade, are defined as strings with maximum lengths of 100 and 10 characters, respectively, enforcing data constraints.

Therefore, the Student class is an abstraction layer over the database table. When the database schema is created, a table named something like student will be generated, containing id (primary key), name, and grade as its fields. Any actions performed on Student objects in Python, such as creating, reading, updating, or deleting, will be translated by SQLAlchemy into the appropriate SQL commands for execution against the database.

2. Explain what happens when /add is accessed with a POST request.

⇒When the /add route is accessed with an HTTP POST request (as defined on line 20), the browser is submitting data from an HTML form to the server. The add_student() function is executed, which first uses request.form['name'] and request.form['grade'] to read the data the user submitted. This process extracts the values for the student's name and grade from the form data sent in the request body.

Next, the extracted Python strings are used to create a new Student object on line 24: new_student = Student(name=name, grade=grade). This action creates an instance of the ORM model, essentially creating a new row of data in memory, but the student record has not yet been saved to the database. At this stage, the new_student object is a temporary Python representation of the student data.

Finally, the function executes the database session commands (db.session.add and db.session.commit) to send the data to the database, followed by return "Student added!". This means the application has successfully received the data, validated it (implicitly, as no explicit validation is shown), constructed the ORM object, and initiated the process to permanently save the new student record before sending a confirmation message back to the client.

3. What is the purpose of db.session.add() and db.session.commit()

⇒The db.session.add(new_student) command's purpose is to stage the new_student object for saving. When an object is added to the session, it tells SQLAlchemy to track that object and include it in the next batch of changes to be written to the database. The session acts

as a temporary holding area for all pending database operations—such as insertions, deletions, or updates—before they are actually executed.

The db.session.commit() command is the critical step that makes the changes permanent. When a commit is called, SQLAlchemy takes all the staged operations (including the new_student object) and translates them into corresponding SQL INSERT commands. It then sends these commands to the database for execution. This process is known as a transaction: if any part of the batch fails, the entire set of changes is rolled back to maintain database integrity.

In essence, the two commands work together as a two-step process for persistence. The add() method queues the specific change (the new student record), and the commit() method executes all pending changes in the session as a single atomic operation. Without commit(), the new_student object would remain only in memory and would not be permanently saved to the database table.

4. How can you verify that the student was successfully saved using SQLAlchemy?

⇒The primary way to verify a successful save is to use SQLAlchemy's query interface to retrieve the data from the database. The most straightforward method is to use Student.query.all(). This command instructs SQLAlchemy to execute a SELECT * FROM student query and return all existing student records from the database as a list of Student objects.

A developer would typically execute this query after the db.session.commit() line. They could then check the results of the query to ensure that the newly added student's details (name and grade) are present in the returned list of objects. This confirms that the transaction was completed, and the data was written to the persistent storage.

Alternatively, for a very large database, one could use a more specific query like Student.query.filter_by(name='[submitted_name]').first(). This method searches for the specific student based on a known attribute (like their name) and only returns that one record. If this query returns a valid Student object instead of None, it serves as a more direct and efficient verification that the particular student record was successfully saved and is accessible in the database.