**ASSIGNMENT**

1**.** SQL code to create the messages and likes tables, and the trigger

To create the "messages" table:

CREATE TABLE messages (

id INT PRIMARY KEY,

sender\_id INT NOT NULL,

message TEXT NOT NULL,

timestamp DATETIME NOT NULL

);

To create the "likes" table:

CREATE TABLE likes (

id INT PRIMARY KEY,

message\_id INT NOT NULL,

user\_id INT NOT NULL,

timestamp DATETIME NOT NULL

);

To create a trigger that runs after an insert statement on the "likes" table and updates the "messages" table with the new like count:

CREATE TRIGGER update\_like\_count

AFTER INSERT ON likes

FOR EACH ROW

BEGIN

UPDATE messages

SET like\_count = like\_count + 1

WHERE id = NEW.message\_id;

END;

2. Flask routes and functions to handle creating and querying messages and likes

from flask import Flask, request, jsonify

from flask\_sqlalchemy import SQLAlchemy

app = Flask(\_\_name\_\_)

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:////tmp/test.db'

db = SQLAlchemy(app)

class Message(db.Model):

id = db.Column(db.Integer, primary\_key=True)

sender\_id = db.Column(db.Integer, nullable=False)

message = db.Column(db.String(140), nullable=False)

timestamp = db.Column(db.DateTime, nullable=False)

def \_\_init\_\_(self, sender\_id, message, timestamp):

self.sender\_id = sender\_id

self.message = message

self.timestamp = timestamp

class Like(db.Model):

id = db.Column(db.Integer, primary\_key=True)

message\_id = db.Column(db.Integer, db.ForeignKey('message.id'), nullable=False)

user\_id = db.Column(db.Integer, nullable=False)

timestamp = db.Column(db.DateTime, nullable=False)

def \_\_init\_\_(self, message\_id, user\_id, timestamp):

self.message\_id = message\_id

self.user\_id = user\_id

self.timestamp = timestamp

db.create\_all()

@app.route('/messages', methods=['POST'])

def create\_message():

data = request.get\_json()

new\_message = Message(data['sender\_id'], data['message'], data['timestamp'])

db.session.add(new\_message)

db.session.commit()

return jsonify({'message': 'success'})

@app.route('/messages', methods=['GET'])

def get\_messages():

messages = Message.query.all()

return jsonify({'messages': [message.to\_dict() for message in messages]})

@app.route('/likes', methods=['POST'])

def create\_like():

data = request.get\_json()

new\_like = Like(data['message\_id'], data['user\_id'], data['timestamp'])

db.session.add(new\_like)

db.session.commit()

return jsonify({'like': 'success'})

@app.route('/likes', methods=['GET'])

def get\_likes():

likes = Like.query.all()

return jsonify({'likes': [like.to\_dict() for like in likes]})

3.Instructions for how to set up and run the API

Here are the general instructions to set up and run the API using the code I provided earlier:

Install the necessary packages: Flask and Flask-SQLAlchemy. You can do this by running the command pip install Flask Flask-SQLAlchemy in your terminal.

Create a new Python file and copy the code I provided into it.

In the code, update the SQLALCHEMY\_DATABASE\_URI in the app.config dictionary to point to the location of your SQLite database. If you want to use a different database, you will need to update the database URI accordingly.

Run the db.create\_all() function to create the necessary tables in the database.

Start the Flask development server by running the command flask run in the terminal while in the same directory as the python file.

You should see a message that the server is running at a specific address (usually http://127.0.0.1:5000/). You can test the API by making requests to that address using a tool like curl or postman.

To create a message, you can make a POST request to http://127.0.0.1:5000/messages with a JSON payload containing sender\_id, message, timestamp in the request body.

To retrieve all messages, you can make a GET request to http://127.0.0.1:5000/messages and you will get a JSON response containing all the messages.

To create a like, you can make a POST request to http://127.0.0.1:5000/likes with a JSON payload containing message\_id, user\_id, timestamp in the request body.

To retrieve all likes, you can make a GET request to http://127.0.0.1:5000/likes and you will get a JSON response containing all the likes.

**BONUS**

1. Use hasura to build the tables and triggers

Hasura is an open-source engine that connects to your databases and allows you to manage your data using a simple and secure GraphQL API. With Hasura, you can create tables and triggers without writing any SQL code. Here's how you can use Hasura to create the tables and triggers for the messages and likes example:

1. Install and set up Hasura by following the instructions on the Hasura website (<https://hasura.io/>).
2. Once Hasura is set up, navigate to the Hasura console (usually at [**http://localhost:8080**](http://localhost:8080)) and log in.
3. In the console, click on the "Data" tab in the top navigation menu.
4. Click on the "Create Table" button to create a new table. In the modal that appears, enter the name of the table as "messages" and add the following columns:
   * **id** (integer, primary key)
   * **sender\_id** (integer, not null)
   * **message** (text, not null)
   * **timestamp** (timestamp with time zone, not null)
5. Repeat the step 4 to create a table named "likes" with the following columns:
   * **id** (integer, primary key)
   * **message\_id** (integer, not null)
   * **user\_id** (integer, not null)
   * **timestamp** (timestamp with time zone, not null)
6. Now go to the "Triggers" tab in the top navigation menu and click on the "Create Trigger" button.
7. In the modal that appears, enter the name of the trigger as "update\_like\_count" and select the "likes" table as the table that the trigger should run on.
8. In the "Operations" section, select "INSERT" as the operation to trigger on.
9. In the "SQL" section, write the SQL statement that updates the like count in the messages table:
10. UPDATE messages SET like\_count = like\_count + 1 WHERE id = NEW.message\_id;
11. Click on the "Create Trigger" button to create the trigger.

Now, whenever a new like is added to the "likes" table, the "update\_like\_count" trigger will run and update the like count in the "messages" table accordingly.

You can now use the GraphQL API provided by Hasura to create, read, update and delete messages and likes. You can also make use of the built-in authorization and access control features provided by Hasura.

**2. Use docker to setup db and other containers required**

Docker is a containerization platform that allows you to package your application and its dependencies into a portable container. Using Docker, you can set up and run the database and other required containers for your application. Here's an example of how you can use Docker to set up the database and Hasura for the messages and likes example:

1. Install Docker on your machine by following the instructions on the Docker website (<https://www.docker.com/>).
2. Create a new directory on your machine and navigate to it in the terminal.
3. Create a **docker-compose.yml** file in the directory. This file will be used to define and configure the containers that will be created. Here's an example of what the file should look like for setting up a PostgreSQL database and Hasura:

version: '3'

services:

db:

image: postgres:12

environment:

POSTGRES\_USER: postgres

POSTGRES\_PASSWORD: postgres

POSTGRES\_DB: postgres

ports:

- "5432:5432"

hasura:

image: hasura/graphql-engine:v2.4.0

ports:

- "8080:8080"

environment:

HASURA\_GRAPHQL\_DATABASE\_URL: postgres://postgres:postgres@db:5432/postgres

HASURA\_GRAPHQL\_ENABLE\_CONSOLE: "true"

depends\_on:

- db