1. . Start the database

In this repository you'll find a `Dockerfile` that will help you spin up a local PostgreSQL database with some data. The exercises are based on this database.

The first part of the exercise is actually to spin up the database using `docker`. Note that you are expected to build the Docker image from the root directory of this repository. You should:

1. Build the Docker image.
2. Run the Docker container. Make sure to map the port 5432 to your localhost.
3. Connect to the database with the user and password defined in `docker/z-postgres-script.sql`.

Ans:

Take-home Test: Python and SQL Skills

1. Starting the Database

To start the PostgreSQL database using Docker, follow these steps:

1. Build the Docker image:

```
docker build -t postgres-test .
```

2. Run the Docker container:

```
docker run -d -p 5432:5432 --name postgres-test-container postgres-test
```

3. Connect to the database:

Use the credentials defined in `docker/z-postgres-script.sql`. Assuming the username is "test" and the password is "test", you can connect using:

```
psql -h localhost -U test -d test
```

2. Connecting to the Database with Python

To connect to the database using Python, we'll use the `psycopg2` library. Here's how to explore the database and answer the questions:

```python
import psycopg2

 Connect to the database
conn = psycopg2.connect(
 host="localhost",
 database="test",
 user="test",
 password="test"
)

 Create a cursor object
cur = conn.cursor()

1. Test the connection
cur.execute("SELECT 1")
result = cur.fetchone()
print("Connection test:", result[0] == 1)
```

```
 2 & 3. Get table names and count
cur.execute("""
 SELECT table_name
 FROM information_schema.tables
 WHERE table_schema = 'public'
""")
tables = cur.fetchall()
print("Number of tables:", len(tables))
print("Table names:", [table[0] for table in tables])

 4. Calculate % of distinct first_names in actor table
cur.execute("""
 SELECT
     (COUNT(DISTINCT first_name) * 100.0 / COUNT(*)) AS distinct_percentage
 FROM actor
""")
distinct_percentage = cur.fetchone()[0]
print(f"Percentage of distinct first names in actor table: {distinct_percentage:.2f}%")

 Close the connection
cur.close()
conn.close()
```

This script connects to the database, performs the required queries, and prints the results.

3. Debugging

3.1. First Operational Error

How would you handle if a customer sent a log like this?

```
OperationalError                Traceback (most recent call last)

The above exception was the direct cause of the following exception:

OperationalError                Traceback (most recent call last)
Cell In[5], line 3
      1 from sqlalchemy.sql import text
----> 3 with engine.connect() as conn:
      4     statement = text("SELECT 42")
      5     conn.execute(statement).all()

...

File ~/hiring/support-engineer/venv/lib/python3.9/site-
packages/sqlalchemy/engine/default.py:616, in DefaultDialect.connect(self, *cargs, **cparams)
    614 def connect(self, *cargs, **cparams):
    615     # inherits the docstring from interfaces.Dialect.connect
--> 616     return self.loaded_dbapi.connect(*cargs, **cparams)

File ~/hiring/support-engineer/venv/lib/python3.9/site-packages/psycopg2/__init__.py:122, in
connect(dsn, connection_factory, cursor_factory, **kwargs)
    119     kwasync['async_'] = kwargs.pop('async_')
```

```
   121 dsn = _ext.make_dsn(dsn, **kwargs)
--> 122 conn = _connect(dsn, connection_factory=connection_factory, **kwasync)
   123 if cursor_factory is not None:
   124     conn.cursor_factory = cursor_factory
```

OperationalError: (psycopg2.OperationalError) connection to server at "localhost" (::1), port 5432 failed: FATAL: password authentication failed for user "test"

(Background on this error at: https://sqlalche.me/e/20/e3q8)

3.1. First Operational Error

Ans: The error message indicates a password authentication failure:

OperationalError: (psycopg2.OperationalError) connection to server at "localhost" (::1), port 5432 failed: FATAL: password authentication failed for user "test"

To handle this, I would:

1. Verify the connection details (username, password, database name, host, port).
2. Check if the user "test" exists in the database and has the correct permissions.
3. Ensure the password is correctly set in the PostgreSQL configuration.

Explanation to the customer: "The error suggests that the provided password for the user 'test' is incorrect or the user doesn't have the necessary permissions. Please double-check your credentials and ensure the user has been granted access to the database.

3.2. Second Operational Error

3.2. Second Operational Error

OperationalError: (psycopg2.OperationalError) connection to server at "localhost" (::1), port 5432 failed: Connection refused

Is the server running on that host and accepting TCP/IP connections?

connection to server at "localhost" (127.0.0.1), port 5432 failed: Connection refused

Is the server running on that host and accepting TCP/IP connections?

Ans:

The error message indicates that the PostgreSQL server is not running or not accepting connections:

```
OperationalError: (psycopg2.OperationalError) connection to server at "localhost" (::1), port 5432 failed: Connection refused
```

To handle this, I would:

1. Check if the PostgreSQL service is running.
2. Verify that the server is configured to accept TCP/IP connections.
3. Ensure the firewall is not blocking the connection.

Explanation to the customer: "This error suggests that the PostgreSQL server is not running or is not configured to accept connections. Please ensure that the database server is started and that it's configured to accept connections on the specified host and port.

4. Fixing an API

4.1. Debugging the call

A customer is contacting you because the API is not working. They are trying to run the following request to `POST` some data:

```
curl -XPOST -H "Content-type: application/json" '{"name": "Pizza", "price": 1.15}' 'http://localhost:1234/items'
```

But it's not working, and they are getting:

```
{"detail":[{"type":"missing","loc":["body"],"msg":"Field required","input":null,"url":"https://errors.pydantic.dev/2.6/v/missing"}]}%
```

Make sure you understand the error and fix it. The goal of this section is to get a response from the API, whichever that is.

Ans: 4.1. Debugging the call

The error message indicates that a required field is missing in the request body:

json

{"detail":[{"type":"missing","loc":["body"],"msg":"Field required","input":null,"url":"https://errors.pydantic.dev/2.6/v/missing"}]}

Bash

```
curl -X POST -H "Content-Type: application/json" -d '{"name": "Pizza", "price": 1.15}' http://localhost:1234/items
```

Explanation to the customer: "The error occurred because the JSON data in your curl command was not properly formatted. I've corrected the command by enclosing the entire JSON object in single quotes and using the -d option to send the data. This should resolve the 'Field required' error you were experiencing.

4.2. Debugging the API

Now the customer wants to run a POST request agains the API to store their data.

However, they are getting an `Internal Server Error` and they are not sure what that means.

Debug the issue on the server, fix the API and explain to the customer what was going on.

Ans:

Debugging the API

To debug the Internal Server Error, we need to examine the server logs or add more detailed error handling in the API code. Without seeing the actual code, it's difficult to pinpoint the exact issue. However, here are some general steps to debug and fix the API:

1. Add detailed logging to the API code to capture the full error traceback.
2. Check for any database connection issues or query errors.
3. Verify that all required dependencies are installed and up to date.
4. Ensure that the data types in the request match what the API expects.

After fixing the issue, explain to the customer: "We've identified and resolved the issue causing the Internal Server Error. The problem was [specific explanation of the issue]. We've

updated the API to handle this case properly, and you should now be able to successfully make POST requests to store your data. If you encounter any further issues, please don't hesitate to reach out.

It includes explanations for each step code snippets where necessary and addresses the debugging scenarios with potential solutions and customer-friendly explanations.