UNIVERSITY OF CALGARY

ENSF 608

Databases

Database Interaction in the Application Documentation

This documentation explains how the Stock Recommendation System application interacts with its database, focusing on the database connection setup and SQL query execution. It will outline the Django settings for database connection, and provide code snippets demonstrating how SQL queries are executed within the application.

1. Database Connection

In Django applications, database connections are handled through the DATABASES setting in the settings.py file. The following code specifies the connection configuration used by the Stock Recommendation System:

File: settings.py

```
DATABASES = {
    'default': {
        "ENGINE": "django.db.backends.mysql",
        "NAME": "stock_recommendation_system_db",
        "USER": "admin",
        "PASSWORD": "admin@123",
        "HOST": "127.0.0.1",
        "PORT": "3306",
    }
}
```

- ENGINE: Specifies the backend to use; here, we are using MySQL.
- NAME: The name of the database.
- USER: Database username.
- PASSWORD: Password associated with the database user.
- HOST: The IP address or hostname of the database server. 127.0.0.1 refers to the local machine.
- PORT: The port number through which the database connection is established. The default for MySQL is 3306.

This setup enables Django to manage the database connection and ensures that queries executed in the application communicate with stock_recommendation_system_db.

2. Query Execution

This section explains how SQL queries are executed within the application, specifically using sql_queries.py.

---

1. **Function:** get_bookmarked_stocks_for_user(user_id)

- **Objective:** Retrieves a list of stocks bookmarked by a specific user.
- **SQL Query:** Executes a SELECT statement that joins the Stocks and user_bookmarked_stocks tables to fetch bookmarked stocks for the given user, ordered by the creation date in descending order.
- **Code:**

```
def get_bookmarked_stocks_for_user(user_id):
    query = """
        SELECT s.stock_id, s.stock_symbol, s.company_name, s.sector,
   s.market_cap, s.created_at
        FROM Stocks s
        INNER JOIN user_bookmarked_stocks ubs ON s.stock_id =
   ubs.stock_id
        WHERE ubs.user_id = %s
        ORDER BY s.created_at DESC
    """


    # Execute the query
    with connection.cursor() as cursor:
        cursor.execute(query, [user_id])
        results = cursor.fetchall()


    # Map the query results to a list of dictionaries
    stocks_data = [
        {
            "stock_id": row[0],
            "symbol": row[1],
            "name": row[2],
            "sector": row[3],
            "market_cap": row[4],
            "created_at": row[5],
        }
        for row in results
```

```
    ]
    return stocks_data
```

Description:

- The function starts by defining a SQL query to select stock details (stock_id, stock_symbol, company_name, sector, market_cap, created_at) from the Stocks table and joining it with the user_bookmarked_stocks table using the stock_id.
- The WHERE clause filters records to include only the bookmarked stocks for the specified user_id.
- The ORDER BY clause ensures the results are sorted by the created_at timestamp in descending order.
- The cursor.execute() method executes the query, passing the user_id as a parameter to prevent SQL injection.
- The fetchall() method retrieves all matching rows from the database.
- Each result row is mapped to a dictionary containing stock attributes (stock_id, symbol, name, sector, market_cap, created_at).
- The function finally returns a list of these dictionaries, making it easy to work with the data in the application.

2. Function: insert_user(name, email, password)

- Objective: Inserts a new user into the Users table.
- SQL Query: Executes an INSERT statement to add user data.
- Code:

```
def insert_user(name, email, password):
    created_at = timezone.now()
    with connection.cursor() as cursor:
        cursor.execute(
            "INSERT INTO stock_recommendation_system_db.users (name,
email, password, created_at) VALUES (%s, %s, %s, %s)",
            [name, email, password, created_at]
        )
```

Description:

- The cursor.execute() method is used to send the INSERT query directly to the database.
- The query uses parameterized values for name, email, password, and created_at, which are provided as a list to prevent SQL injection.
- created_at is set to the current timestamp using timezone.now(), and this value is inserted along with the other fields into the Users table