Тема 8. Работа с базой данных в приложении.

Цель занятия:

Получить навыки взаимодействия приложения с базой данных, используя прямые SQL-запросы и ORM

Учебные вопросы:

1. Основные принципы взаимодействия приложения и базы данных.
2. Подключение к базе данных MySQL в Python.
3. Выполнение прямых SQL-запросов из приложений.
4. Использование ORM для работы с базой данных
6. Заключение.

1. Основные принципы взаимодействия приложения и базы данных.

1. Подключение к базе данных

Каждое приложение, взаимодействующее с базой данных, должно установить соединение с сервером базы данных. Это осуществляется через драйверы или библиотеки, которые обеспечивают возможность подключения к различным системам управления базами данных (СУБД), включая MySQL.

Драйверы — это программные компоненты, которые позволяют приложениям разных языков программирования взаимодействовать с конкретной СУБД.

Подключение обычно требует указания следующих параметров:

- Хост — адрес сервера базы данных.
- Порт — порт для подключения (по умолчанию для MySQL — 3306).
- Имя пользователя и пароль для авторизации.
- Имя базы данных — указание конкретной базы данных, с которой планируется работа.

2. Выполнение SQL-запросов.

После подключения приложение может отправлять SQL-запросы к базе данных. Запросы могут быть разного типа:

DML (Data Manipulation Language) — для работы с данными: SELECT, INSERT, UPDATE, DELETE.

DDL (Data Definition Language) — для создания и изменения структуры базы данных: CREATE TABLE, ALTER TABLE.

Работа с запросами обычно проходит в несколько этапов:

- Подготовка SQL-запроса.
- Передача запроса к базе данных (с помощью методов драйвера).
- Получение результатов (если это запрос SELECT).
- Закрытие соединения после завершения работы.

3. Параметризация запросов.

Для защиты от атак типа SQL-инъекций (внедрения вредоносного SQL-кода через пользовательский ввод) применяются параметризованные запросы.

Это позволяет передавать данные безопасно, избегая прямого включения их в строку SQL-запроса.

Пример параметризированного запроса в Python:

```
with conn.cursor() as cursor:
    sql = "SELECT * FROM products WHERE product_name = %s"
    cursor.execute(sql, ('Product 1',))
    result = cursor.fetchall()
```

4. Управление транзакциями

Транзакции обеспечивают целостность данных, гарантируя, что серия операций выполнится как единое целое.

Приложения могут явно управлять транзакциями через команды BEGIN, COMMIT, ROLLBACK.

5. Закрытие соединения.

Необходимо закрывать соединение с базой данных после завершения работы. Это предотвращает утечку ресурсов и повышает производительность.

6. Обработка ошибок.

Работа с базой данных должна учитывать возможность возникновения ошибок (ошибки подключения, нарушения целостности данных, синтаксические ошибки в SQL и т. д.). Поэтому важно всегда реализовывать обработку исключений.

7. Кэширование запросов.

Для повышения производительности можно использовать кэширование часто используемых SQL-запросов.

Это снижает нагрузку на базу данных, сокращая количество обращений к серверу.

8. Обработка результатов.

Результаты выполнения запросов могут быть представлены в виде:

- Наборов данных (таблиц), которые можно обрабатывать по строкам и столбцам (например, в виде объектов ResultSet в Java или Cursor в Python).
- Одиночных значений — для получения единичного результата, например, количества записей.

2. Подключение к базе данных MySQL в разных языках программирования.

Python.

- **Подключение с помощью библиотеки pymysql.**

pymysql — популярная библиотека для работы с MySQL через стандартные SQL-запросы.

Установка:

```
pip install pymysql
```

Пример подключения и выполнения простого SQL-запроса:

```
import pymysql


# Устанавливаем подключение к базе данных
conn = pymysql.connect(
    host='localhost',
    user='root',
    password='password',
    database='mydatabase'
)
```

```python
try:
    # Создаем курсор для выполнения SQL-запросов
    with conn.cursor() as cursor:
        # Выполняем простой SQL-запрос
        sql = "SELECT * FROM users"
        cursor.execute(sql)

        # Получаем и выводим результат запроса
        results = cursor.fetchall()
        for row in results:
            print(row)
finally:
    # Закрываем соединение с базой данных
    conn.close()
```

- **Подключение с помощью библиотеки mysql-connector-python**

mysql-connector-python — это официальная библиотека для работы с MySQL от разработчиков MySQL.

Установка:

```
pip install mysql-connector-python
```

```python
import mysql.connector


# Устанавливаем подключение к базе данных
conn = mysql.connector.connect(
    host='localhost',
    user='root',
    password='password',
    database='mydatabase'
)
```

```python
try:
    # Создаем курсор
    cursor = conn.cursor()

    # Выполняем простой SQL-запрос
    cursor.execute("SELECT * FROM users")

    # Получаем и выводим результат
    results = cursor.fetchall()
    for row in results:
        print(row)
finally:
    # Закрываем соединение с базой данных
    conn.close()
```

- **Подключение с помощью библиотеки SQLAlchemy**

SQLAlchemy — это мощная ORM (Object-Relational Mapping), которая позволяет взаимодействовать с базами данных на более высоком уровне, чем просто выполнение SQL-запросов.

Установка:

```
pip install SQLAlchemy pymysql
```

```python
from sqlalchemy import create_engine, text

# Создаем объект для подключения к базе данных
engine = create_engine('mysql+pymysql://root:password@localhost/mydatabase')

# Устанавливаем подключение
with engine.connect() as connection:
    # Выполняем простой SQL-запрос
    result = connection.execute(text("SELECT * FROM users"))

    # Получаем и выводим результат
    for row in result:
        print(row)
```

Описание процесса подключения:

Установка подключения: Во всех примерах подключение к базе данных осуществляется через вызов функции connect() или создание объекта соединения (engine.connect() в SQLAlchemy).

Параметры подключения обычно включают:

host: сервер, на котором работает MySQL.

user: имя пользователя MySQL.

password: пароль пользователя.

database: название базы данных, с которой работает приложение.

Создание курсора: Курсор позволяет выполнять SQL-запросы. В библиотеках pymysql и mysql-connector-python используется объект курсора (cursor), который необходим для выполнения запросов и обработки результатов. В SQLAlchemy запросы выполняются через объект соединения (connection).

Выполнение запроса: SQL-запросы передаются через метод execute() в каждой библиотеке.

Получение результата: Результаты запроса можно получить с помощью метода fetchall() (для выборки всех строк) или итерирования по объекту результата (как в SQLAlchemy).

Закрытие соединения: После завершения работы с базой данных соединение должно быть закрыто, чтобы освободить ресурсы. Это делается вручную или автоматически с использованием контекстного менеджера (with).

Сравнение библиотек:

- **pymysql** — легковесная библиотека для простого выполнения SQL-запросов. Подходит для небольших проектов, где не требуется ORM.
- **mysql-connector-python** — официальное решение от MySQL, с немного более сложной настройкой, но хорошей поддержкой от MySQL.
- **SQLAlchemy** — мощная и гибкая библиотека, которая предлагает ORM, а также возможность работы с "сырыми" SQL-запросами. Хороший выбор для крупных проектов с более сложной архитектурой базы данных.

3. Выполнение прямых SQL-запросов из приложений.

Выполнение прямых SQL-запросов из приложений

Прямые SQL-запросы — это запросы, которые напрямую передаются приложением в базу данных для выполнения операций, таких как выборка данных, вставка, обновление или удаление.

Эти запросы могут быть выполнены с помощью стандартных SQL-операторов (SELECT, INSERT, UPDATE, DELETE).

Преимущества прямых SQL-запросов:

- Простота: Написание SQL-запросов напрямую в коде позволяет быстро реализовать логику работы с базой данных без дополнительного слоя абстракции.
- Гибкость: Прямые SQL-запросы предоставляют полный контроль над тем, какие данные и как будут извлечены или изменены в базе данных.
- Производительность: В некоторых случаях прямое выполнение запросов может быть быстрее, так как отсутствуют дополнительные уровни, как в ORM.

Недостатки прямых SQL-запросов:

- Уязвимость к SQL-инъекциям: Если запросы неправильно параметризованы, приложение может быть подвержено атакам типа SQL-инъекций.
- Читаемость и поддерживаемость: Наличие множества прямых SQL-запросов в коде может затруднить его чтение и поддержку.
- Сложность масштабирования: В больших проектах управление SQL-запросами напрямую может стать сложной задачей, особенно при работе с несколькими базами данных или изменении их структуры.

Работа с параметризованными запросами для безопасности (SQL-инъекции).

SQL-инъекции — это тип атак, при которых злоумышленник внедряет вредоносный SQL-код в запрос. Чтобы предотвратить такие атаки, важно использовать параметризованные запросы.

Пример уязвимого запроса (без параметризации):

```
user_input = "admin' --"
query = f"SELECT * FROM users WHERE username = '{user_input}'"
cursor.execute(query)
```

Если злоумышленник введет строку admin' --, запрос станет:

```
SELECT * FROM users WHERE username = 'admin' --'
```

Комментарий (--) завершит запрос, и проверка пароля не выполнится.

Пример параметризации (безопасный подход):

```
sql = "SELECT * FROM users WHERE username = %s AND password = %s"
cursor.execute(sql, (username, password))
```

Вместо того, чтобы напрямую подставлять пользовательские данные в строку запроса, используются специальные плейсхолдеры (%s), которые затем связываются с реальными значениями.

4. Использование ORM для работы с базой данных

ORM (Object-Relational Mapping) — это метод программирования, который позволяет разработчикам взаимодействовать с базами данных, используя объектно-ориентированный подход.

ORM автоматически отображает объекты программного кода на таблицы в базе данных и наоборот.

Принципы работы ORM:

- Отображение объектов в таблицы: Каждый класс в приложении соответствует таблице в базе данных, а атрибуты классов — столбцам таблиц.
- Автоматическое управление связями: ORM обрабатывает связи между объектами, позволяя легко устанавливать отношения между таблицами (например, один-к-одному, один-ко-многим и многие-ко-многим).
- Абстракция SQL: ORM позволяет разработчикам работать с базами данных и не писать SQL-запросы напрямую. Вместо этого используется синтаксис, понятный для языка программирования.

Преимущества использования ORM:

- Абстракция базы данных: Разработчики могут сосредоточиться на бизнес-логике, а не на SQL.
- Упрощение кода: Меньше кода для написания и понимания, так как ORM автоматически генерирует запросы.
- Уменьшение числа ошибок: ORM снижает риск ошибок, связанных с ручным написанием SQL-запросов и управления соединениями.

Популярные ORM-фреймворки:

Python: SQLAlchemy — один из самых популярных ORM-фреймворков в Python. Он предоставляет мощные инструменты для работы с базами данных, поддерживает различные диалекты SQL и позволяет гибко настраивать взаимодействие с базами данных.

5. CRUD

CRUD — это аббревиатура, обозначающая четыре основных операции, выполняемые с данными в приложении, работающем с базами данных:

1. **Create (Создание)**: добавление новых записей в базу данных.
2. **Read (Чтение)**: извлечение данных из базы, получение информации о записях.
3. **Update (Обновление)**: изменение существующих записей в базе данных.
4. **Delete (Удаление)**: удаление записей из базы данных.

Эти операции являются основой большинства приложений, работающих с базами данных, и позволяют управлять данными в системе.

6. Заключение

Используйте ORM для упрощения разработки и повышения читаемости кода, особенно для стандартных операций с базой данных.

Прямые запросы лучше применять для сложных операций или когда требуется оптимизация производительности и полный контроль над SQL.

Домашнее задание:

1. Повторить материал лекции.

Контрольные вопросы:

- Какие основные параметры необходимы для подключения к базе данных MySQL?
- Что такое параметризация запросов, и как она помогает защититься от SQL-инъекций?
- Опишите процесс выполнения SQL-запросов в приложении: этапы и используемые методы.
- Какие типы SQL-запросов существуют и для чего они используются (DML и DDL)?
- В чем заключаются преимущества и недостатки прямых SQL-запросов в приложении?
- Какие библиотеки используются для подключения к MySQL в Python, и чем они отличаются?
- Что такое ORM, и как оно упрощает работу с базой данных?
- Какие преимущества предоставляет использование ORM по сравнению с прямыми SQL-запросами?
- Как правильно обрабатывать результаты SQL-запросов в приложении?
- Почему важно закрывать соединение с базой данных после работы?

Список литературы:

1. В. Ю. Кара-ушанов SQL — язык реляционных баз данных
2. А. Б. ГРАДУСОВ. Введение в технологию баз данных
3. А.Мотеев. Уроки MySQL

Материалы лекций:

https://github.com/ShViktor72/Education

Обратная связь:

colledge20education23@gmail.com