IT CAREER HUB

Урок 42.
MySQL. Транзакции

Подключение без указания базы

При подключении к MySQL через `pymysql.connect()` **указание базы данных (database=...) не является обязательным**.

Это может быть полезно, если:

- вы хотите сначала подключиться, а потом выбрать базу вручную;
- работаете с административными задачами (создание/удаление баз);
- вам нужно переключаться между разными базами в рамках одного соединения.

Пример

```python
import pymysql

config = {'host': 'ich-db.edu.itcareerhub.de',
          'user': 'ich1',
          'password': 'password',
          }

connection = pymysql.connect(**config)

with connection.cursor() as cursor:
    cursor.execute("SHOW DATABASES")
    for db in cursor:
        print(db)
```

Выбор базы данных после подключения

Если база не указана при подключении, её можно выбрать вручную с помощью SQL-команды:

```python
cursor.execute("USE yourdatabase")
```

После этого все последующие запросы будут выполняться в рамках выбранной базы.

Пример: выбор базы данных вручную

```python
import pymysql

config = {'host': 'ich-db.edu.itcareerhub.de',
          'user': 'ich1',
          'password': 'password',
          }

connection = pymysql.connect(**config)

with connection.cursor() as cursor:
    cursor.execute("USE hr")       # выбрать нужную базу
    cursor.execute("SHOW TABLES")  # запрос внутри этой базы
    for row in cursor:
        print(row)
```

Особенности:

- Команда USE выбирает базу данных на время текущего соединения.
- Если база не будет выбрана, запросы к таблицам вызовут ошибку `No database selected`.
- Можно выбирать другую базу в любой момент, выполнив новый `USE <database>`.

DictCursor

По умолчанию курсор в `pymysql` возвращает строки в виде **кортежей**. Чтобы получать результаты в виде **словарй** `dict` **с названиями колонок в качестве ключей**, можно использовать `DictCursor`.

Подключение с DictCursor

```python
import pymysql
from pymysql.cursors import DictCursor  # импорт класса

config = {'host': 'ich-db.edu.itcareerhub.de',
          'user': 'ich1',
          'password': 'password',
          'database': 'hr',
          'cursorclass': DictCursor,  # ссылка на класс
          }

connection = pymysql.connect(**config)
```

Пример

```python
with connection.cursor() as cursor:
    cursor.execute("SELECT * FROM employees")
    result = cursor.fetchone()
    print(result)
    print(result["first_name"])  # доступ по имени столбца
```

IT CAREER HUB

Задания для закрепления 1

1. Почему может быть полезно не указывать базу данных при подключении?

a. Чтобы сократить время соединения
b. Чтобы создавать или удалять базы
c. Чтобы ускорить выборку данных
d. Чтобы выбирать базу позже вручную

Посмотреть ответ

2. Что произойдёт, если выполнить запрос к таблице без выбора базы данных?

a. Будет использована первая база
b. Будет использована стандартная база
c. Возникнет ошибка `No database selected`
d. Вернётся пустой результат

Посмотреть ответ

IT CAREER HUB

Метод commit

Чтобы **добавить, изменить или удалить данные** в базе, используются запросы `INSERT`, `UPDATE`, `DELETE`. У них есть важная особенность — данные сначала попадают во **временное хранилище транзакции**.

Чтобы окончательно записать изменения в базу данных, необходимо вызвать метод:

```python
connection.commit()
```

Зачем это нужно?

- Без `commit()` изменения **не сохраняются** и будут **потеряны** при закрытии соединения.
- Это поведение обеспечивает **безопасность**: изменения применяются только после явного подтверждения.

Создание базы данных и таблиц (`CREATE DATABASE`, `CREATE TABLE`) **не требует вызова** `commit()` — эти команды сохраняются сразу.

Подключение к серверу с правами на изменения

Чтобы создавать базы и таблицы, подключитесь к серверу с правами на изменения:

```python
config = {'host': 'ich-edit.edu.itcareerhub.de',
          'user': 'ich1',
          'password': 'ich1_password_ilovedbs',
          }

connection = pymysql.connect(**config)
```

Пример: создание базы и таблицы sales

```python
with connection.cursor() as cursor:
    # Создание новой базы данных (если ещё не существует)
    cursor.execute("CREATE DATABASE IF NOT EXISTS market")
    cursor.execute("USE market")

    # Создание таблицы продаж
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS sales (
            id INT AUTO_INCREMENT PRIMARY KEY,
            item_name VARCHAR(100),
            quantity INT,
            price DECIMAL(10, 2),
            sale_date DATE
        )
    """)
```

Пример: вставка данных

```python
cursor.execute(
     "INSERT INTO sales (item_name, quantity, price, sale_date) VALUES (%s, %s, %s, %s)",
    ("Keyboard", 2, 45.50, "2024-06-15")
)
connection.commit()  # Сохраняем изменения
```

Пример: обновление данных

```python
cursor.execute(
    "UPDATE sales SET quantity = %s WHERE item_name = %s",
    (3, "Keyboard")
)
connection.commit()
```

Пример: удаление данных

```python
cursor.execute(
    "DELETE FROM sales WHERE item_name = %s",
    ("Keyboard",)
)
connection.commit()
```

IT CAREER HUB

Метод executemany

Метод `executemany()` используется для многократного выполнения одного и того же SQL-запроса с разными данными. Это удобно и эффективно при массовом добавлении, обновлении или удалении строк.

Пример

```python
cursor.executemany(
     "INSERT INTO sales (item_name, quantity, price, sale_date) VALUES (%s, %s, %s, %s)",
    [
        ("Notebook", 3, 19.99, "2024-06-15"),
        ("Pen", 10, 1.99, "2024-06-16"),
        ("Bag", 1, 49.90, "2024-06-17")
    ]
)
connection.commit()
```

Особенности:

- Работает только с **изменяющими запросами** (`INSERT, UPDATE, DELETE`)
- Вторым аргументом передаётся **список кортежей**
- Экономит ресурсы, т.к. запрос компилируется один раз, а выполняется много

IT CAREER HUB

Транзакции

 **Транзакция — это последовательность запросов, которая выполняется как единое целое.**

Если хотя бы один из шагов завершился ошибкой — **все изменения отменяются**. Это позволяет сохранить **целостность данных** в базе.

Зачем нужны транзакции?

- Гарантируют **атомарность**: всё или ничего
- Защищают от частичных обновлений при сбоях
- Полезны при **нескольких связанных изменениях** (например, перемещение денег между счетами)

IT CAREER HUB

Метод rollback

Если в процессе транзакции возникает ошибка, можно использовать метод:

```python
connection.rollback()
```

Этот метод **отменяет все изменения**, сделанные с начала текущей транзакции. Он используется в связке с `try-except` и `commit()`, чтобы **сохранить целостность данных**.

Если транзакция не подтверждена вызовом `commit()`, то `rollback()` отменяет все действия, выполненные **с момента начала транзакции** — то есть с момента последнего `commit()` или с открытия соединения.

Пример: транзакция покупки товара клиентом

Создадим необходимые таблицы:

- `clients` — список клиентов и их баланс
- `sales` — информация о товарах
- `purchases` — записи о покупках

Затем **попробуем провести оплату**. Клиент попытается купить товар:

- Если у него **хватит денег**, операция завершится успешно.
- Если денег **недостаточно**, база останется без изменений.

1. Подготовка таблиц

Создаём три таблицы:

- `customers` — список клиентов с балансом,
- `products` — товары с ценой и остатком,
- `purchases` — фиксация покупок.

```python
cursor.execute("""
CREATE TABLE IF NOT EXISTS customers (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    balance DECIMAL(10, 2) NOT NULL CHECK (balance >= 0)
)
""")

cursor.execute("""
CREATE TABLE IF NOT EXISTS products (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    price DECIMAL(10, 2),
    stock INT NOT NULL CHECK (stock >= 0)
)
""")

cursor.execute("""
CREATE TABLE IF NOT EXISTS purchases (
    id INT AUTO_INCREMENT PRIMARY KEY,
    customer_id INT,
    product_id INT,
    purchase_date DATE,
    FOREIGN KEY (customer_id) REFERENCES customers(id),
    FOREIGN KEY (product_id) REFERENCES products(id)
)
""")
```

2. Заполнение данными

Добавим двух клиентов и два товара. Один клиент с маленьким балансом, другой — с достаточным.

```python
cursor.execute("DELETE FROM purchases")
cursor.execute("DELETE FROM customers")
cursor.execute("DELETE FROM products")

cursor.executemany(
    "INSERT INTO customers (name, balance) VALUES (%s, %s)",
```

```
    [("Alice", 20.00), ("Bob", 200.00)]
)

cursor.executemany(
    "INSERT INTO products (name, price, stock) VALUES (%s, %s, %s)",
    [("Headphones", 99.99, 3), ("Mouse", 25.00, 5)]
)

connection.commit()
```

3. Попытка покупки с недостаточным балансом

Если у клиента недостаточно денег, никакие изменения (списание денег, уменьшение количества товара, запись в покупки) не произойдут. Всё будет отменено.

Python

```
try:
    # Получаем товар
      cursor.execute("SELECT id, price, stock FROM products WHERE name = %s",
("Headphones",))
    product_id, price, stock = cursor.fetchone()

    # Если товар в наличии списываем его
    if stock < 1:
        raise ValueError("Out of stock")
      cursor.execute("UPDATE products SET stock = stock - 1 WHERE id = %s",
(product_id,))


    # Получаем клиента
       cursor.execute("SELECT id, balance FROM customers WHERE name = %s",
("Alice",))
    customer_id, balance = cursor.fetchone()

    # Если баланса хватает списываем оплату
    if balance < price:
        raise ValueError("Insufficient funds")
     cursor.execute("UPDATE customers SET balance = balance - %s WHERE id = %s",
(price, customer_id))
```

```
    # Фиксируем покупку в таблице purchases
          cursor.execute("INSERT INTO purchases (customer_id, product_id,
purchase_date) VALUES (%s, %s, CURDATE())",
                  (customer_id, product_id))

    connection.commit()  # При отсутствии ошибок завершаем транзакцию
    print("Purchase successful.")
except Exception as e:
    connection.rollback()  # Откатываем если что-то пошло не так
    print("Transaction failed:", e)
```

4. Покупка с достаточным балансом

Здесь у клиента хватает денег, и покупка проходит успешно.

Python

```
try:
    # Получаем товар
      cursor.execute("SELECT id, price, stock FROM products WHERE name = %s",
("Mouse",))
    product_id, price, stock = cursor.fetchone()

    # Если товар в наличии списываем его
    if stock < 1:
        raise ValueError("Out of stock")
      cursor.execute("UPDATE products SET stock = stock - 1 WHERE id = %s",
(product_id,))


    # Получаем клиента
       cursor.execute("SELECT id, balance FROM customers WHERE name = %s",
("Bob",))
    customer_id, balance = cursor.fetchone()

    # Если баланса хватает списываем оплату
    if balance < price:
        raise ValueError("Insufficient funds")
     cursor.execute("UPDATE customers SET balance = balance - %s WHERE id = %s",
(price, customer_id))
```

```
    # Фиксируем покупку в таблице purchases
          cursor.execute("INSERT  INTO  purchases  (customer_id,  product_id,
purchase_date) VALUES (%s, %s, CURDATE())",
                  (customer_id, product_id))

    connection.commit()  # при отсутствии ошибок завершаем транзакцию
    print("Purchase successful.")
except Exception as e:
    connection.rollback()  # откатываем если что-то пошло не так
    print("Transaction failed:", e)
```

IT CAREER HUB

Задания для закрепления 2

1. После какого действия нужно вызывать commit()?

a. `SELECT`
b. `INSERT`
c. `UPDATE`
d. `DELETE`

Посмотреть ответ

2. Что делает метод executemany()?

a. Выполняет SQL-запрос указанное количество раз
b. Выполняет SQL-запрос указанное количество раз или до первого успешного выполнения
c. Позволяет выполнять один SQL-запрос с разными данными

Посмотреть ответ

3. Что произойдёт, если после INSERT не вызвать commit()?

a. Запрос не выполнится
b. Изменения будут сохранены автоматически
c. Изменения будут потеряны после закрытия соединения

Посмотреть ответ

IT CAREER HUB

Ответы на задания

Задания на закрепление 1	Вернуться к заданиям
1. Указание базы данных при подключении	Ответ: b, d
2. Запрос к таблице без выбора базы данных	Ответ: c
Задания на закрепление 2	Вернуться к заданиям
1. Вызов `commit()`	Ответ: b, c, d
2. Метод `executemany()`	Ответ: c
3. Отсутствие вызова commit() после INSERT	Ответ: c

IT CAREER HUB

🔎 Практическая работа

1. Добавление товаров в таблицу

Напишите программу, которая подключается к базе данных `market`, затем:

- создаёт таблицу `products`, если она ещё не существует
- добавляет в неё несколько товаров (название и цена)
- выводит список товаров с их ценами

Данные:

```python
products = [
    ("Notebook", 10.00),
    ("Pencil", 1.00),
    ("Bag", 25.00)
]
```

Пример вывода:

```python
Products added:
1. Notebook — $10.00
2. Pencil — $1.00
3. Bag — $25.00
```

Решение:

```python
from pymysql.cursors import DictCursor
import pymysql

config = {
    'host': 'ich-edit.edu.itcareerhub.de',
    'user': 'ich1',
    'password': 'ich1_password_ilovedbs',
}

with pymysql.connect(**config, cursorclass=DictCursor) as connection:
    with connection.cursor() as cursor:
        cursor.execute("CREATE DATABASE IF NOT EXISTS market")
        cursor.execute("USE market")

        cursor.execute("""
            CREATE TABLE IF NOT EXISTS products (
                id INT AUTO_INCREMENT PRIMARY KEY,
                name VARCHAR(100),
                price DECIMAL(10, 2)
            )
        """)

        products = [
            ("Notebook", 10.00),
            ("Pencil", 1.00),
            ("Bag", 25.00)
        ]
        cursor.executemany(
            "INSERT INTO products (name, price) VALUES (%s, %s)",
            products
        )
        connection.commit()

        print("Products added:")
        cursor.execute("SELECT id, name, price FROM products")
        for row in cursor.fetchall():
            print(f"{row['id']}. {row['name']} – ${row['price']:.2f}")
```

IT CAREER HUB

2. Массовое повышение цен

Продолжите предыдущую программу. Теперь:

- увеличьте цену всех товаров на **20%**;
- выполните обновление с помощью одного `UPDATE`;
- выведите список товаров после изменения цен.

Пример вывода:

```
Python
Prices updated.

Products after update:
1. Notebook — $12.00
2. Pencil — $1.20
3. Bag — $30.00
```

Решение:

```python
with pymysql.connect(**config, cursorclass=DictCursor) as connection:
    with connection.cursor() as cursor:
        cursor.execute("USE market")
        cursor.execute("UPDATE products SET price = price * 1.2")
        connection.commit()

        print("Prices updated.\n\nProducts after update:")
        cursor.execute("SELECT id, name, price FROM products")
        for row in cursor.fetchall():
            print(f"{row['id']}. {row['name']} — ${row['price']:.2f}")
```