m16_assignment

December 31, 2024

1 Module 16: Database Integration with Python

1.1 Assignment

[1]:
```
import subprocess
import sys

# Step 1: Install mysql-connector-python if not installed
try:
    import mysql.connector
except ImportError:
    print("mysql-connector-python is not installed. Installing now...")
    subprocess.check_call([sys.executable, "-m", "pip", "install",␣
 ↪"mysql-connector-python"])
    import mysql.connector
```

```
mysql-connector-python is not installed. Installing now…
```

[4]:
```
# Step 1: Connect to the MySQL Server and create the "retails" database
db_connection = mysql.connector.connect(
    host="127.0.0.1",
    user="john",
    password="P@ssw0rd12345"
)
cursor = db_connection.cursor()

cursor.execute("CREATE DATABASE IF NOT EXISTS retails")
print("Database 'retails' created!")
```

```
Database 'retails' created!
```

[6]:
```
# Step 2: Connect to the "retails" database and create tables
# Switch to the "retails" database
# Use the newly created database for subsequent operations
db_connection.database = "retails"

#  Clean up existing tables if they exist
# Drop tables "orders" and "customer" if they already exist to start fresh
cursor.execute("DROP TABLE IF EXISTS orders")
```

```
cursor.execute("DROP TABLE IF EXISTS customer")
print("Existing tables dropped!")

# Create "customer" table
# Define the schema for the "customer" table
cursor.execute(
    """
    CREATE TABLE IF NOT EXISTS customer (
        customer_id INT PRIMARY KEY,
        age INT,
        city VARCHAR(50),
        gender VARCHAR(10)
    )
    """
)
print("Table 'customer' created!")

# Create "orders" table
# Define the schema for the "orders" table
cursor.execute(
    """
    CREATE TABLE IF NOT EXISTS orders (
        order_id INT AUTO_INCREMENT PRIMARY KEY,
        order_date DATE,
        amount DECIMAL(10, 2),
        customer_id INT,
        FOREIGN KEY (customer_id) REFERENCES customer(customer_id)
    )
    """
)
print("Table 'orders' created!")
```

```
Existing tables dropped!
Table 'customer' created!
Table 'orders' created!
```

[8]:
```
# Step 3: Add 'is_sale' column to the "orders" table
# Alter the "orders" table to add a new column "is_sale"
cursor.execute("ALTER TABLE orders ADD COLUMN is_sale BOOLEAN DEFAULT FALSE")
print("Column 'is_sale' added to 'orders' table.")
```

```
Column 'is_sale' added to 'orders' table.
```

[9]:
```
# Step 4: Insert data into "customer" table
# Populate the "customer" table with sample data
customer_data = [
    (1001, 34, 'Austin', 'male'),
    (1002, 37, 'Houston', 'male'),
```

```
    (1003, 25, 'Austin', 'female'),
    (1004, 28, 'Houston', 'female'),
    (1005, 22, 'Dallas', 'male')
]
cursor.executemany(
    "INSERT INTO customer (customer_id, age, city, gender) VALUES (%s, %s, %s,␣
 ↪%s)", customer_data
)
db_connection.commit()
print("Data inserted into 'customer' table.")
```

```
Data inserted into 'customer' table.
```

[11]:
```
# Step 5: Query customers in Austin
# Retrieve and display customers located in Austin
cursor.execute("SELECT * FROM customer WHERE city = 'Austin'")
print("\nCustomers in Austin:")
for row in cursor.fetchall():
    print(row)
```

```
Customers in Austin:
(1001, 34, 'Austin', 'male')
(1003, 25, 'Austin', 'female')
```

[12]:
```
# Step 6: Group customers by city
# Count and group customers by their city
cursor.execute("SELECT city, COUNT(*) AS num_customers FROM customer GROUP BY␣
 ↪city")
print("\nCustomers grouped by city:")
for row in cursor.fetchall():
    print(row)
```

```
Customers grouped by city:
('Austin', 2)
('Houston', 2)
('Dallas', 1)
```

[14]:
```
# Step 7: Group customers by gender
# Count and group customers by their city
cursor.execute("SELECT gender, COUNT(*) AS num_customers FROM customer GROUP BY␣
 ↪gender")
print("\nCustomers grouped by gender:")
for row in cursor.fetchall():
    print(row)
```

```
Customers grouped by gender:
```

```
('male', 3)
('female', 2)
```

[15]:
```
# Step 8: Insert data into "orders" table
# Populate the "orders" table with sample data
orders_data = [
    ('2022-10-1', 100.25, 1001),
    ('2022-10-2', 200.75, 1002),
    ('2022-10-3', 500.00, 1003),
    ('2022-10-3', 600.00, 1004),
    ('2022-10-4', 600.00, 1005)
]
cursor.executemany(
    "INSERT INTO orders (order_date, amount, customer_id) VALUES (%s, %s, %s)",␣
 ↪orders_data
)
db_connection.commit()
print("Data inserted into 'orders' table.")
```

```
Data inserted into 'orders' table.
```

[16]:
```
# Step 9: Query orders on '2022-10-03'
# Retrieve and display orders placed on the specified date
cursor.execute("SELECT * FROM orders WHERE order_date = '2022-10-03'")
print("\nOrders on 2022-10-03:")
for row in cursor.fetchall():
    print(row)
```

```
Orders on 2022-10-03:
(8, datetime.date(2022, 10, 3), Decimal('500.00'), 1003, 0)
(9, datetime.date(2022, 10, 3), Decimal('600.00'), 1004, 0)
```

[17]:
```
# Step 10: Show orders with amount > 300
cursor.execute("SELECT * FROM orders WHERE amount > 300")
print("Orders with amount > 300:")
for row in cursor.fetchall():
    print(row)
```

```
Orders with amount > 300:
(8, datetime.date(2022, 10, 3), Decimal('500.00'), 1003, 0)
(9, datetime.date(2022, 10, 3), Decimal('600.00'), 1004, 0)
(10, datetime.date(2022, 10, 4), Decimal('600.00'), 1005, 0)
```

[19]:
```
# Step 11: Show and sort orders placed on '2022-10-03'
cursor.execute("SELECT * FROM orders WHERE order_date = '2022-10-03' ORDER BY␣
 ↪amount DESC")
print("Sorted orders on 2022-10-03:")
for row in cursor.fetchall():
```

```
    print(row)
```

```
Sorted orders on 2022-10-03:
(9, datetime.date(2022, 10, 3), Decimal('600.00'), 1004, 0)
(8, datetime.date(2022, 10, 3), Decimal('500.00'), 1003, 0)
```

[21]:
```
# Step 12: Count the number of distinct days
cursor.execute("SELECT COUNT(DISTINCT order_date) FROM orders")
distinct_days = cursor.fetchone()[0]
print(f"Number of distinct days: {distinct_days}")

cursor.execute("SELECT DISTINCT order_date FROM orders")
distinct_dates = cursor.fetchall()

#print(f"Number of distinct days: {len(distinct_dates)}")
print("Distinct days are:")
for date in distinct_dates:
    print(date[0])
```

```
Number of distinct days: 4
Distinct days are:
2022-10-01
2022-10-02
2022-10-03
2022-10-04
```

[22]:
```
# Step 13: Count the orders grouped by date
cursor.execute("SELECT order_date, COUNT(*) AS num_orders FROM orders GROUP BY␣
 ↪order_date")
print("Orders grouped by date:")
for row in cursor.fetchall():
    print(row)
```

```
Orders grouped by date:
(datetime.date(2022, 10, 1), 1)
(datetime.date(2022, 10, 2), 1)
(datetime.date(2022, 10, 3), 2)
(datetime.date(2022, 10, 4), 1)
```

[23]:
```
# Step 14: Calculate average order amount
cursor.execute("SELECT AVG(amount) AS avg_amount FROM orders")
avg_amount = cursor.fetchone()[0]
print(f"Average order amount: {avg_amount:.2f}")
```

```
Average order amount: 400.20
```

[24]:
```
# Close the connection
cursor.close()
db_connection.close()
```

[ ]: