WILFRID LAURIER UNIVERSITY
LAURIER
Inspiring Lives.

CP465 – Database II Assignment #2

Instruction:

1- You must use the Answer template (see the assignment folder) to write down your answer to the theoretical questions.
2- Carefully read the entire assignment first and focus on the rubrics and submission requirements.

Part 1: Theoretical Questions (40 Points)

Chapter 17: Database-System Architectures (10 Points)

1. **Compare and contrast centralized, client-server, parallel, and distributed database architectures.**
    a) Provide real-world examples of each model. (5 points)
2. **Explain the different types of parallel database architectures.**
    a) Discuss shared-memory, shared-disk, and shared-nothing architectures. (5 points)

Chapter 20: Data Warehousing and Data Mining (10 Points)

3. **What is the difference between OLAP and OLTP?**
    a) Explain how data warehousing supports OLAP. (5 points)
4. **Describe the concept of association rule mining.**
    a) Provide an example of how market basket analysis is used in business intelligence. (5 points)

Chapter 21: Information Retrieval (10 Points)

5. **Explain how relevance ranking works using term frequency (TF) and inverse document frequency (IDF).**
    a) Show the mathematical formulas and provide an example calculation. (5 points)

6. **Describe how web search engines use hyperlink-based ranking algorithms like PageRank.**

    a) Provide a real-world example. (5 points)

Chapter 22: Object-Based Databases (10 Points)

7. **What are the advantages and disadvantages of object-relational databases (ORDBMS) compared to relational databases (RDBMS)?**

    a) Provide examples of object-oriented features in SQL. (5 points)

8. **Discuss the concept of Object-Relational Mapping (ORM).**

    a) How does ORM help developers integrate databases with object-oriented programming languages? (5 points)

Part 2: Practical Exercises (30 Points)

Chapter 20: Implementing a Simple Data Warehouse (15 Points)

Task: Design and implement a simple **star schema** for a retail business.

Instructions:

1) Create **four tables** in SQL:
    a. **Fact table:** `sales` (Transaction ID, Product ID, Date, Amount).
    b. **Dimension tables:** `customers`, `products`, `time`.
2) Insert at least **10 sample records**.
3) Write SQL queries for:
    a. Finding the **total sales** per product.
    b. Identifying the **most valuable customers**.
    c. Computing **monthly sales trends**.

Example SQL Query:

```
SELECT p.product_name, SUM(s.amount) AS total_sales
FROM sales s
JOIN products p ON s.product_id = p.product_id
GROUP BY p.product_name;
```

Chapter 21: Information Retrieval - Implementing TF-IDF (15 Points)

Task: Implement **TF-IDF ranking** for a small collection of documents using Python.

Instructions:

a) Use a **set of 5 text documents** as the dataset. A document can contain 3-5 sentences.
b) Implement **TF-IDF** ranking to compute term importance.
c) Rank documents based on a user-given **query**.
d) Display the **top-ranked documents**.

Python Implementation:

```
from sklearn.feature_extraction.text import TfidfVectorizer

# Sample documents
documents = [
    "Databases are essential for managing data.",
    "Data mining helps uncover patterns in databases.",
    "Information retrieval focuses on search and ranking.",
    "Object databases store complex data structures.",
    "Parallel databases allow distributed processing."
]

# Initialize TF-IDF Vectorizer
vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(documents)

# Display feature names and matrix
print("Feature Names:", vectorizer.get_feature_names_out())
print("TF-IDF Matrix:", tfidf_matrix.toarray())
```

Part 3: Application-Based Task (30 Points)

Chapter 22: Implementing Object-Relational Mapping (ORM) with SQLAlchemy

Objective:

To design and implement a **database system** using **Object-Relational Mapping (ORM)** with **SQLAlchemy** in Python. This task will demonstrate how **object-oriented programming (OOP)** concepts integrate with **relational databases**.

Task Overview:

Develop an ORM-based database for an e-commerce system that manages Customers, Orders, and Products.

1 Database Design Requirements:

The database should have the following tables:

1. Customer Table

- `customer_id` (Primary Key)
- `name` (VARCHAR)
- `email` (VARCHAR)
- `created_at` (DATETIME, Default: Current Timestamp)
- `orders` (One-to-Many Relationship with Orders)

2. Order Table

- `order_id` (Primary Key)
- `customer_id` (Foreign Key → Customer Table)
- `order_date` (DATETIME, Default: Current Timestamp)
- `products` (Many-to-Many Relationship with Products)

3. Product Table

- `product_id` (Primary Key)
- `name` (VARCHAR)
- `price` (FLOAT)
- `orders` (Many-to-Many Relationship with Orders)

4. Order_Product Table (Many-to-Many Relationship Table)

- `order_id` (Foreign Key → Order Table)
- `product_id` (Foreign Key → Product Table)
- `quantity` (INT)

2 Implementation Details

1. **Use Python and SQLAlchemy** to create ORM models for all four tables.
2. **Insert sample data** (at least **3 customers, 5 products, and 5 orders**).
3. **Implement queries** to:

    a) Retrieve all **orders** for a given **customer**.
    b) Retrieve all **products** in a specific **order**.
    c) Calculate **total revenue per customer**.
    d) **Generate a report** of **customers and their total spending.**
    e) **Use a bar chart** to visualize the **top 5 customers by spending.**

SQLAlchemy Code Implementation

```
from sqlalchemy import create_engine, Column, Integer, String,
Float, DateTime, ForeignKey, Table
from sqlalchemy.orm import relationship, sessionmaker,
declarative_base
import datetime
import matplotlib.pyplot as plt

# Initialize database
Base = declarative_base()
engine = create_engine('sqlite:///ecommerce.db')
Session = sessionmaker(bind=engine)
session = Session()

# Many-to-Many Association Table (Orders & Products)
order_product_table = Table(
    'order_product', Base.metadata,
    Column('order_id', Integer, ForeignKey('orders.order_id')),
    Column('product_id', Integer,
ForeignKey('products.product_id')),
    Column('quantity', Integer)
)

# Customer Model
class Customer(Base):
    __tablename__ = 'customers'
    customer_id = Column(Integer, primary_key=True)
    name = Column(String)
    email = Column(String)
    created_at = Column(DateTime,
default=datetime.datetime.utcnow)
    orders = relationship('Order', back_populates='customer')

# Order Model
class Order(Base):
    __tablename__ = 'orders'
    order_id = Column(Integer, primary_key=True)
    customer_id = Column(Integer,
ForeignKey('customers.customer_id'))
    order_date = Column(DateTime,
default=datetime.datetime.utcnow)
    customer = relationship('Customer', back_populates='orders')
    products = relationship('Product',
secondary=order_product_table, back_populates='orders')

# Product Model
class Product(Base):
    __tablename__ = 'products'
```

```
    product_id = Column(Integer, primary_key=True)
    name = Column(String)
    price = Column(Float)
    orders = relationship('Order',
secondary=order_product_table, back_populates='products')

# Create database tables
Base.metadata.create_all(engine)
```

Insert Sample Data

```
# Insert Customers
customer1 = Customer(name="Alice Johnson",
email="alice@example.com")
customer2 = Customer(name="Bob Smith", email="bob@example.com")
customer3 = Customer(name="Charlie Brown",
email="charlie@example.com")

session.add_all([customer1, customer2, customer3])
session.commit()

# Insert Products
product1 = Product(name="Laptop", price=1200.00)
product2 = Product(name="Smartphone", price=800.00)
product3 = Product(name="Headphones", price=150.00)
product4 = Product(name="Monitor", price=300.00)
product5 = Product(name="Keyboard", price=50.00)

session.add_all([product1, product2, product3, product4,
product5])
session.commit()

# Insert Orders
order1 = Order(customer_id=customer1.customer_id,
products=[product1, product3])
order2 = Order(customer_id=customer1.customer_id,
products=[product2, product5])
order3 = Order(customer_id=customer2.customer_id,
products=[product4])
order4 = Order(customer_id=customer3.customer_id,
products=[product2, product3, product5])

session.add_all([order1, order2, order3, order4])
session.commit()
```

Queries and Data Analysis

Retrieve all orders for a given customer

```
def get_customer_orders(customer_name):
    customer =
session.query(Customer).filter_by(name=customer_name).first()
    if customer:
        print(f"\nOrders for {customer.name}:")
        for order in customer.orders:
            print(f"Order ID: {order.order_id}, Date:
{order.order_date}")
    else:
        print("Customer not found.")

get_customer_orders("Alice Johnson")
```

Retrieve all products in a specific order

```
def get_order_products(order_id):
    order =
session.query(Order).filter_by(order_id=order_id).first()
    if order:
        print(f"\nProducts in Order {order_id}:")
        for product in order.products:
            print(f"{product.name} - ${product.price}")
    else:
        print("Order not found.")

get_order_products(1)
```

Calculate total revenue per customer

```
def get_total_spent_per_customer():
    results = session.query(Customer, Order).join(Order).all()
    customer_spending = {}

    for customer, order in results:
        total_spent = sum([product.price for product in
order.products])
        if customer.name in customer_spending:
            customer_spending[customer.name] += total_spent
        else:
            customer_spending[customer.name] = total_spent

    print("\nTotal Revenue per Customer:")
    for customer, total in customer_spending.items():
```

```
        print(f"{customer}: ${total:.2f}")

    return customer_spending

spending_data = get_total_spent_per_customer()
```

Data Visualization (Bar Chart of Top 5 Customers by Spending)

```
def plot_top_customers(spending_data):
    sorted_data = dict(sorted(spending_data.items(), key=lambda
item: item[1], reverse=True)[:5])
    plt.bar(sorted_data.keys(), sorted_data.values(),
color=['blue', 'green', 'red', 'purple', 'orange'])
    plt.xlabel("Customers")
    plt.ylabel("Total Spending ($)")
    plt.title("Top 5 Customers by Spending")
    plt.show()

plot_top_customers(spending_data)
```

Deliverables

a) **Python script** for ORM implementation.
b) **SQL dump file** if needed for testing in raw SQL.
c) **Report including:**
   Schema design explanation.
   Sample queries and results.
   Screenshots of database tables.
   Data analysis and insights from revenue calculations.
   Visualization of top customers using Matplotlib.

Students should use the code snippets to:

a) Modify and expand the ORM models (e.g., add new fields like shipping_address).
b) Run queries on real-world datasets (e.g., extend with more customer orders).
c) Experiment with new queries (e.g., find the most purchased product).
d) Improve data visualization (e.g., add pie charts, trend graphs).

Grading Rubric Part 3

Criteria	Points	Description
ORM Implementation	10	Correct definition of tables, relationships, and data insertion.
Query Execution & Results	10	Accurate query retrievals for orders, products, and revenue.

Data Analysis & Visualization	10	Proper revenue calculations and graphical representation of customer spending.

Grading Rubric: All assignment

Criteria	Points	Description
Theoretical Questions	40	Completeness, clarity, and depth of explanation.
Data Warehouse Implementation	15	Correct schema design, SQL queries, and clear explanations.
TF-IDF Implementation	15	Accurate TF-IDF ranking, well-commented Python code.
Object-Relational Mapping	30	Functional ORM with SQLAlchemy, proper relations, and correct query execution.

Submission Guidelines

Submit your assignment as follows:

1. A **PDF document** with answers, SQL outputs, and Python results.
2. Python **code files** (ipythonnotebook)
3. Ensure all code is **properly commented** and tested.