

Mohamed Shamir | 17110084

Note: Please visit the [github repository](#) for codes. [Link](#).

Contents

- [Questions:](#)
 - [Problem 1](#)
 - [1](#)
 - [2](#)
 - [3](#)
 - [5.](#)
 - [6](#)
 - [13](#)
 - [16](#)
 - [19](#)
 - [20](#)
 - [21](#)
 - [Problem 2](#)
 - [References:](#)
 - [Useful Materials:](#)
 - [Problems Bumped in and Fixes:](#)
-

Questions:

Problem 1

Implement the backend connectors for the E-Commerce database created in Assignment 2.

1. Ensure that input sanitization checks are performed.
2. Submit a method definition for Question [1, 2, 3, 5, 6, 13, 16, 19, 20, 21] of the SQL queries implemented in Assignment 2, i.e., you have to holi_Deals a python function as a solution for each question. It must perform the task as per the question.

Ex: If a query is " insert name and roll number in STUDENT table. "

```
# Solution
```

```
def insert_data(name, roll_num, tablename, dbconnector, dbname):  
  
    dummy_query = 'insert into {0}(name, rollnum) values({1},  
{2})'.format(tablename, name, roll_num)
```

```
dbconnector.update(dbname, dummy_query)
# Here input sanitization is not performed
```

Solution:

- [Jupyter Notebook Link](#)

Python Functions:

1

Fill the tables with at least 20 dummy records.

```
class fill_tables:

    def __init__(self, mydb):
        self.cursor=mydb.cursor()
        self.mydb = mydb

    def
insert_into_user_table(self, user_id, user_name, email, pasw, city, state, country
, phone, doj, toj):

        query= """
        INSERT INTO Users VALUES(%s,%s,
        %s,%s,%s,
        %s,%s,%s, %s,%s);
        """

        self.cursor.execute(query,
        (user_id, user_name, email, pasw, city, state, country, phone, doj, toj))
        self.mydb.commit()

    def insert_into_retailer_table(self, user_name, email, city, state, country):

        query= """
        INSERT INTO Retailer VALUES(%s,%s,%s,%s,%s);

        """

        self.cursor.execute(query, (user_name, email, city, state, country))
        self.mydb.commit()

    def
insert_into_delivery_address(self, user_ID, city, country, india, phone, pin):

        query= """
        INSERT INTO Delivery_Address(user_ID, city, state, country, Phone, Pin)
        VALUES(%s,%s,%s,%s,%s,%s);
```

```
        """

        self.cursor.execute(query, (user_ID, city, country, india, phone, pin))
        self.mydb.commit()

    def
insert_into_order_table(self, product_ID, user_ID, order_Date, order_time, address_ID, quantity, rating, review):

        query= """
        INSERT INTO
Orders(product_ID, user_ID, order_Date, order_Time, address_ID, quantity, rating,
review) VALUES(%s,%s,%s,%s,%s,%s,%s,%s);

        """

        self.cursor.execute(query,
        (product_ID, user_ID, order_Date, order_time, address_ID, quantity, rating, review
        ))
        self.mydb.commit()

    def insert_into_history_table(self, product_ID, user_ID, No_of_visites):

        query="""
        INSERT INTO History(user_ID, product_ID, No_of_visits)
VALUES(%s,%s,%s);
        """

        self.cursor.execute(query, (product_ID, user_ID, No_of_visites))
        self.mydb.commit()

    def insert_into_cart_table(self, user_ID, product_ID, quantity):
        query = """
        INSERT INTO Cart(user_ID, product_ID, quantity)
VALUES(%s,%s,%s);
        """

        self.cursor.execute(query, (user_ID, product_ID, quantity))
        self.mydb.commit()

    def
insert_into_product_details(self, tablename, product_name, category_id, date_added, rating, price, retailer_id, quantity_stock):

        query="""
        INSERT INTO %s VALUES(%s,%s,%s,%s,%s,%s,%s,%s);
        """

        self.cursor.execute(query,
```

```
(tablename, product_name, category_id, date_added, rating, price, retailer_id, quantity_stock))  
    self.mydb.commit()
```

```
# How to call the function  
import mysql.connector  
  
mydb = mysql.connector.connect(  
    host="localhost",  
    user="temp",  
    passwd="password",  
    database = "ecommerce"  
  
)  
  
mycursor = mydb.cursor()  
  
insert = fill_tables(mydb)  
insert.insert_into_user_table('10000', 'shamir', 'shamir@gmail.com', 'passw', 'Ahmedabad', 'Gujarat', 'India', '1234567890', '2021-02-23', '2330')
```

```
[17] ▶ ML  
query = "select * from Users where user_ID=10000"  
mycursor.execute(query)  
  
for line in mycursor:  
    print(line)  
  
(10000, 'shamir', 'shamir@gmail.com', 'passw', 'Ahmedabad', 'Gujarat', 'India', '1234567890', datetime.date(2021, 2, 23), datetime.timedelta(seconds=1410))
```

2

Delete a user from the database. After deleting the user update name of the user as 'Anonymous' in all the ratings and reviews written by that user.

Solution:

Here user with user_ID=88888 is anonymous. When a particular user is deleted from the database, their order details are updated to user_ID 88888 before deletion to preserve the ratings and reviews as anonymous user.

```
def delete_from_user(mydb,user_ID):

    query0="""
    SET foreign_key_checks = 0;
    """
    query1="""
    update Orders set user_ID=88888 where user_ID = %s;"""

    query2 = """
    delete from Users where user_ID= %s;"""

    query3="""
    SET foreign_key_checks = 1;
    """
    mycursor= mydb.cursor()
    mycursor.execute(query0)
    mycursor.execute(query1,(user_ID,))
    mycursor.execute(query2,(user_ID,))
    mycursor.execute(query3)
    mydb.commit()
```

```
"""
Example on how to call
Here 4 is the user_ID of the user who is getting deleted.
"""

delete_from_user(mydb,4)
```

3

Increment the price of all products priced below Rs. 5000 by 10%, which were viewed by more than 10 users in the last 3 months.

Solution:

- Original SQL query:

```
update Products
inner join
(select B.product_ID,B.price,count(B.date_visited) as Views
from
(select A.product_ID,A.date_visited,A.price
from
(select Products.product_ID, Products.price,
History.user_ID,History.date_visited from
History inner join Products
on Products.product_ID =History.product_ID
```

```

and Products.price<5000) as A
where A.date_visited> now()- INTERVAL 3 month) as B
group by B.product_ID) as C
on Products.product_ID=C.product_ID
and C.Views>10
set Products.price=1.1*Products.price;

```

- Python Mysql

```

def
increase_price(mydb,price_limit,interval,view_limit,percentage_increase):

    query="""

        update Products
        inner join
        (select B.product_ID,B.price,count(B.date_visited) as Views
        from
        (select A.product_ID,A.date_visited,A.price
        from
        (select Products.product_ID, Products.price,
        History.user_ID,History.date_visited from
        History inner join Products
        on Products.product_ID =History.product_ID
        and Products.price<%s) as A
        where A.date_visited> now()- INTERVAL %s month) as B
        group by B.product_ID) as C
        on Products.product_ID=C.product_ID
        and C.Views>%s
        set Products.price=%s*Products.price;
        """

    test_query="""
select C.product_ID,C.price,C.Views from
Products inner join
(select B.product_ID,B.price,count(B.date_visited) as Views
    from
    (select A.product_ID,A.date_visited,A.price
    from
    (select Products.product_ID, Products.price,
    History.user_ID,History.date_visited from
    History inner join Products
    on Products.product_ID =History.product_ID
    and Products.price<%s) as A
    where A.date_visited> now()- INTERVAL %s month) as B
    group by B.product_ID) as C
    on Products.product_ID=C.product_ID
    and C.Views>%s
    """

    percentage_increase = 1+ percentage_increase/100
    mycursor=mydb.cursor()
    mycursor.execute(test_query,(price_limit,interval,view_limit))

```

```

print("Price Before")
print("=====")
print("Product_ID | Price | Views")

print("-----")

for line in mycursor:
    print(line)
mycursor.execute(query,
(price_limit,interval,view_limit,percentage_increase))
mycursor.execute(test_query,(price_limit,interval,view_limit))

print("\n\nPrice After")
print("=====")

print("Product_ID | Price | Views")

print("-----")
for line in mycursor:
    print(line)
mydb.commit()

```

- Calling Python Function

```
increase_price(mydb,5000,3,10,10)
```

The screenshot shows a Jupyter Notebook cell with the following content:

```
[52] ▶ M1
increase_price(mydb,5000,3,10,10)
```

The output of the cell is as follows:

```

Price Before
=====
Product_ID | Price | Views
-----
(27, 1549.23, 20)
(52, 1223.79, 20)

Price After
=====
Product_ID | Price | Views
-----
(27, 1704.15, 20)
(52, 1346.17, 20)

```

5.

Find phone numbers and email IDs of all users who reside in the city 'Madrid' and have made a total purchase greater than or equal to Rs. 10000 in the past.

Solution:

```
def find_data(mydb,city,total_purchase):

    query="""
    select C.user_name,C.email,C.phone from
    (select B.user_name,B.email,B.phone,
    SUM(B.price) total_purchase from
    (select A.*,Products.price from
    (select Users.user_ID,Users.user_name,Users.email,
    Users.phone,Orders.product_ID
    from Users inner join Orders
    on Users.user_ID = Orders.user_ID
    and Users.city=%s) as A
    inner join Products
    on A.product_ID = Products.product_ID) as B
    group by B.user_name,B.email,B.phone) as C
    where C.total_purchase>%s;
    """

    dbcursor=mydb.cursor()
    dbcursor.execute(query,(city,total_purchase))

    for line in dbcursor:
        print(line)
```

```
find_data(mydb,'Madrid',10000)
```



```
[74] find_data(mydb,'Madrid',10000)

('Carl', 'Carl@gmail.com', '1666898865')
('Carla', 'Carla@gmail.com', '2658844727')
```

6

Find all products in the database whose name contains the string 'mi'. E.g. Xiaomi, etc, and all users who bought them at least once.

Solution:

```
def find_data(mydb,word):
    query ="""
    select Name,product_ID
    from Electronics
    where
    Name LIKE %s union all
    select Name,product_ID
```



```

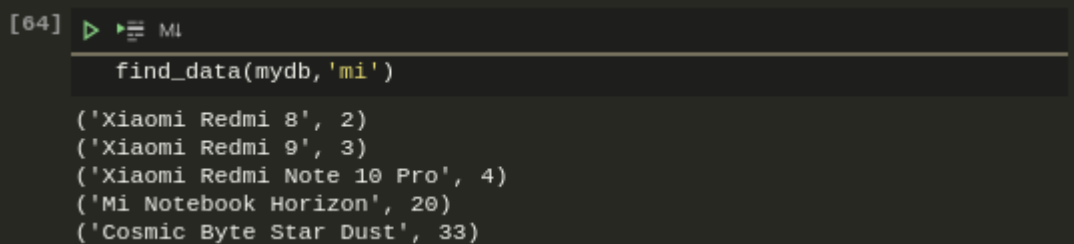
from Novels
where Name LIKE %s union all
select Name,product_ID
from Clothes where Name LIKE %s ;
"""

word = "%" + word + "%"
dbcursor = mydb.cursor()
dbcursor.execute(query, (word, word, word))

for line in dbcursor:
    print (line)

```

```
find_data(mydb, 'mi')
```



```

[64] In [64]: find_data(mydb, 'mi')
Out[64]: ('Xiaomi Redmi 8', 2)
('Xiaomi Redmi 9', 3)
('Xiaomi Redmi Note 10 Pro', 4)
('Mi Notebook Horizon', 20)
('Cosmic Byte Star Dust', 33)

```

13

Sort all laptops according to the price in increasing order.

Solution:

- SQL Query:

```

select Products.product_Name,Products.price,
Products.product_ID
from Products,Electronics
where (Products.product_ID = Electronics.product_ID
and Electronics.sub_category="Laptop")
ORDER BY price DESC;

```

- Python Version:

```

def sort_electronics(mydb,sub_category):

query = """
select Products.product_Name,Products.price

```

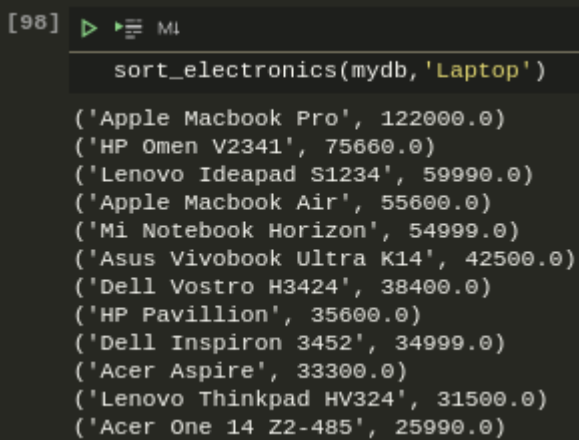
```

from Products,Electronics
where (Products.product_ID = Electronics.product_ID
and Electronics.sub_category=%s)
ORDER BY price DESC;

"""
dbcursor=mydb.cursor()
dbcursor.execute(query,(sub_category,))
for line in dbcursor:
    print(line)

```

```
sort_electronics(mydb, 'Laptop')
```



```

[98] In [98]: sort_electronics(mydb, 'Laptop')
Out[98]: ('Apple Macbook Pro', 122000.0)
('HP Omen V2341', 75660.0)
('Lenovo Ideapad S1234', 59990.0)
('Apple Macbook Air', 55600.0)
('Mi Notebook Horizon', 54999.0)
('Asus Vivobook Ultra K14', 42500.0)
('Dell Vostro H3424', 38400.0)
('HP Pavillion', 35600.0)
('Dell Inspiron 3452', 34999.0)
('Acer Aspire', 33300.0)
('Lenovo Thinkpad HV324', 31500.0)
('Acer One 14 Z2-485', 25990.0)

```

16

Print the UserId, mobile number, and Email Id of all users who have saved a product in the cart, whose quantity is less than 5.

Solution:

```

def list_product_cart(mydb,quantity):

    query = """
    select Cart.user_ID,Users.phone,
    Users.email
    from Cart,Users where
    Cart.quantity<%s
    and Users.user_ID=Cart.user_ID;

    """

    dbcursor=mydb.cursor()
    dbcursor.execute(query,(quantity,))
    for line in dbcursor:
        print(line)

```

```
list_product_cart(mydb,5)
```

```
[104] ▶ MI
list_product_cart(mydb,5)
(3, '3390336017', 'Aaron@gmail.com')
(12, '1991496793', 'Abdiel@gmail.com')
(20, '2816479673', 'Abigale@gmail.com')
(22, '1603080649', 'Abner@gmail.com')
(42, '8629118901', 'Adda@gmail.com')
(43, '7992054653', 'Addie@gmail.com')
(44, '2998110255', 'Addison@gmail.com')
(46, '5992239524', 'Addyson@gmail.com')
(48, '4662736739', 'Adela@gmail.com')
(53, '3946900720', 'Adelia@gmail.com')
(67, '5402619420', 'Adline@gmail.com')
(70, '1163981320', 'Adolfo@gmail.com')
(77, '4326194082', 'Adriana@gmail.com')
(79, '2725591347', 'Adrianna@gmail.com')
(97, '6133214625', 'Ah@gmail.com')
```

19

List all retailer ids whose products, user_id = 1 have purchased.

Solution:

```
def list_retailers(mydb,user):

    query = """
    select Products.retailer_ID
    from Products inner join
    (select *
    from Orders
    where Orders.user_ID= %s ) as A
    on A.product_ID=Products.product_ID;

    """
    dbcursor=mydb.cursor()
    dbcursor.execute(query,(user,))
    print("Retailer Ids")
    for line in dbcursor:
        print(line)
```

```
list_retailers(mydb,1)
```

```
[116] ▶ MI
list_retailers(mydb,1)

Retailer Ids
(19,)
(22,)
(48,)
(47,)
(47,)
(10,)
(48,)
(40,)
(10,)
```

20

Write a query to update the discount on all new products by 15% and store it as a new table holi_Deals.
Note: Any product that is added to the database in the last 100 days is considered to be a new product.

Solution:

- SQL Query:

```
CREATE TABLE holi_Deals (
    product_ID int,
    product_Name varchar(50),
    category_ID int,
    date_added date,
    price float(2),
    retailer_ID int);

INSERT INTO holi_Deals
(product_ID, product_Name, category_ID, date_added, price, retailer_ID)
SELECT
product_ID, product_Name, category_ID, date_added, price*0.85, retailer_ID
from Products
WHERE date_added >=(DATE(NOW()) - INTERVAL 100 DAY)
ORDER BY date_added DESC;
```

- Python Version:

```
def new_deals(mydb, tablename, discount):

    table_create = """
    CREATE TABLE if not exists {table} (
    product_ID int,
    product_Name varchar(50),
    category_ID int,
    date_added date,
    price float(2),
```

```

retailer_ID int);
"""format(table=tablename)

dbcursor = mydb.cursor()
dbcursor.execute(table_create)
mydb.commit()

discount_factor = 1-int(discount)/100
query="""

INSERT INTO {table}
(product_ID,product_Name,category_ID,date_added,price,retailer_ID)
SELECT
product_ID,product_Name,category_ID,date_added,price*%s,retailer_ID
from Products
WHERE date_added >=(DATE(NOW()) - INTERVAL 100 DAY)
ORDER BY date_added DESC;

"""format(table=tablename)

dbcursor.execute(query,(discount_factor,))
mydb.commit()
dbcursor.execute("select * from {table}".format(table=tablename))
for line in dbcursor:
    print(line)

```

```
new_deals(mydb, 'holi_Deals', '15')
```

```

[186] > ML
new_deals(mydb, 'holi_Deals', '15' )
(55, 'The Intelligent Investor', 3, datetime.date(2021, 2, 1), 95.2, 44)
(57, 'The Kite Runner', 3, datetime.date(2021, 1, 28), 255.0, 39)
(56, 'The lean Startup', 3, datetime.date(2021, 1, 24), 212.5, 40)
(58, 'The Mechanic', 3, datetime.date(2021, 1, 20), 340.0, 39)
(54, 'Cut and Sew Denim Jeans', 1, datetime.date(2020, 12, 13), 727.6, 49)
(53, 'Slim Fit Denim Jeans', 1, datetime.date(2020, 12, 12), 449.65, 48)
(48, 'Mens Bomber Jacket', 1, datetime.date(2020, 12, 10), 859.35, 48)
(55, 'The Intelligent Investor', 3, datetime.date(2021, 2, 1), 95.2, 44)
(57, 'The Kite Runner', 3, datetime.date(2021, 1, 28), 255.0, 39)
(56, 'The lean Startup', 3, datetime.date(2021, 1, 24), 212.5, 40)
(58, 'The Mechanic', 3, datetime.date(2021, 1, 20), 340.0, 39)
(54, 'Cut and Sew Denim Jeans', 1, datetime.date(2020, 12, 13), 727.6, 49)
(53, 'Slim Fit Denim Jeans', 1, datetime.date(2020, 12, 12), 449.65, 48)
(48, 'Mens Bomber Jacket', 1, datetime.date(2020, 12, 10), 859.35, 48)
(55, 'The Intelligent Investor', 3, datetime.date(2021, 2, 1), 95.2, 44)
(57, 'The Kite Runner', 3, datetime.date(2021, 1, 28), 255.0, 39)
(56, 'The lean Startup', 3, datetime.date(2021, 1, 24), 212.5, 40)
(58, 'The Mechanic', 3, datetime.date(2021, 1, 20), 340.0, 39)
(54, 'Cut and Sew Denim Jeans', 1, datetime.date(2020, 12, 13), 727.6, 49)
(53, 'Slim Fit Denim Jeans', 1, datetime.date(2020, 12, 12), 449.65, 48)
(48, 'Mens Bomber Jacket', 1, datetime.date(2020, 12, 10), 859.35, 48)
(55, 'The Intelligent Investor', 3, datetime.date(2021, 2, 1), 95.2, 44)

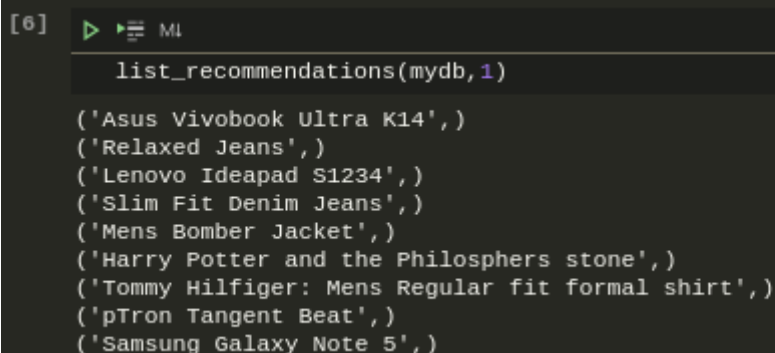
```

21

List the top 10 recommended products for the user_id=1 based on the user's purchase and search history(use any recommendation algorithm).

```
def list_recommendations(mydb,user_ID):  
  
    query="""  
    select Products.product_Name from Orders inner join Products on  
    Products.product_ID = Orders.product_ID  
    and Orders.user_ID=%s  
    order by Orders.rating DESC limit 10;  
  
    """  
  
    cursor = mydb.cursor()  
    cursor.execute(query,(user_ID,))  
  
    for line in cursor:  
        print(line)
```

```
list_recommendations(mydb,1)
```



```
[6] ▶ ML  
list_recommendations(mydb,1)  
  
( 'Asus Vivobook Ultra K14', )  
( 'Relaxed Jeans', )  
( 'Lenovo Ideapad S1234', )  
( 'Slim Fit Denim Jeans', )  
( 'Mens Bomber Jacket', )  
( 'Harry Potter and the Philosphers stone', )  
( 'Tommy Hilfiger: Mens Regular fit formal shirt', )  
( 'pTron Tangent Beat', )  
( 'Samsung Galaxy Note 5', )
```

Problem 2

Assume there is a social network platform named RandomX for movie reviews, where a user can post a review about any movie and give a rating to any movie. To manage all these, we will create a database name RandomX and this database will consist of three schemas. One table will contain all the user details (username, gender, age), the second schema will consist of movie details (movie_id, movie_title, audience_rating), and in the third schema, movies review (movie_id, username, movie_rev) will be there. For your convenience, all these details are provided in the excel (.xls) files:

- **User Information:** This file consist of dummy user details.
- **Movie Information:** This file contains all the movie names along with their id and ratings.
- **Movie Reviews:** This file contains all the user reviews of the movie mentioned in the movie information file, along with the username. Note: For this question, use a jupyter notebook.

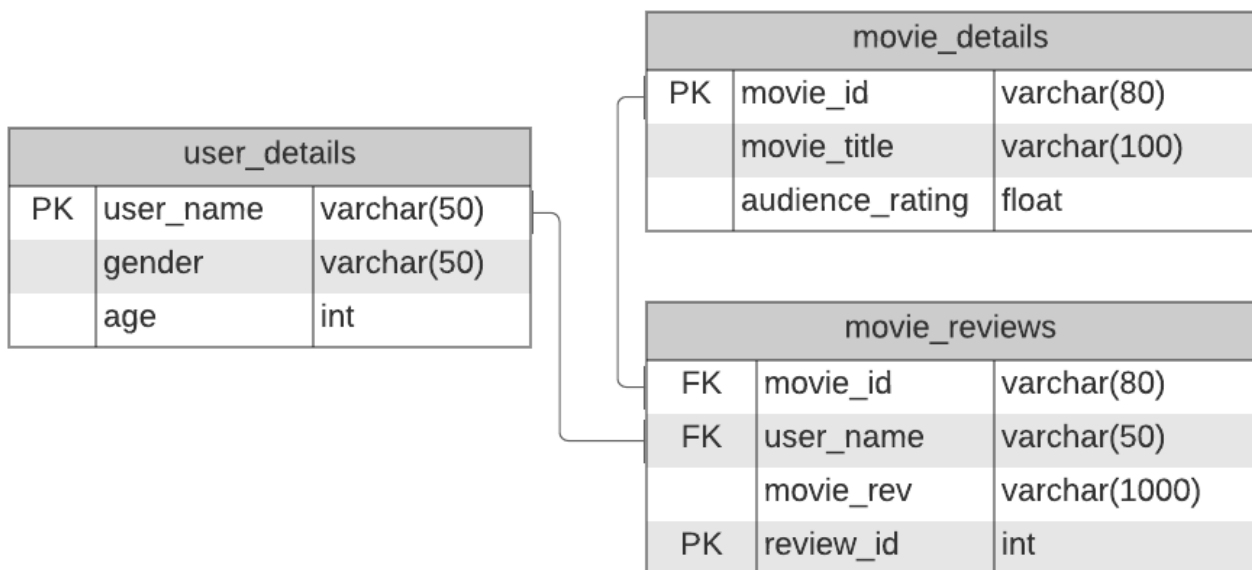
1. Identify the primary keys and all the constraints and then create a database consisting of all the schemas required as per the question. [hint](#) [2 Marks]
2. Insert all the dummy records in the created schemas by reading their respective excel data. [hint: for convenience use pandas library to read excel file] [3 Marks]
3. Add a new column to the movie review table and name it 'sentiment.' This sentiment column consists of 'positive', 'negative', and 'neutral' value based on that row's review. A review is said to be positive if the number of positive words occurring in it is more than the number of negative words, vice-versa for negative, and for neutral, positive word count is equal to negative word count. A list of positive words and negative words is given below, and your task is to update each row's sentiment cell with the 'positive', 'negative', or 'neutral' value. [7 Marks]

Positive Words [Link](#) and Negative Words [Link](#)

4. Show the top 5 movie names with a rating strictly greater than 3.5, and the count of 'positive' sentiment is greater than equal to 2 for that movie. [3 Marks]

Solution:

1. Schema:



2,3,4. [Jupyter Notebook](#).

- 3 Output

```

(1, 'm/the_two_popes', 'gamma', 'Jonathan Pryce and Anthony Hopkins engage in a theological joust, equal parts levity, humor, and spiritual exploration. There is no resolution o
f millennia-old debates, just amusing exposition and proselytizing', 'positive')
(2, 'm/the_two_popes', 'hockeren', 'Somehow the filmmakers found lightheartedness and - gasp - laughs in a story of political intrigue at the top of the notoriously buttoned-up
Catholic Church.', 'neutral')
(3, 'm/the_two_popes', 'carchery', 'There's an unquestionable appeal to the way the movie transforms a weighty and divisive topic into more approachable terms.', 'positive')
(4, 'm/the_two_popes', 'sentact', 'Despite its over reliance on unnecessary flashbacks, "The Two Popes" is a masterclass in acting delivered by two legendary stalwarts of the cr
aft: Pryce and Hopkins.', 'neutral')
(5, 'm/the_two_popes', 'oryoustald', 'Hopkins and Pryce's finely tuned performances illuminate Benedict's shrewd intelligence and Francis's deep humility.', 'positive')
  
```

- 4 Output

	Movie_Name	Audience_Rating	Count of Positive Sentiment
0	The Lord of the Rings: The Fellowship of the Ring	4.75	3
1	Room	4.65	5
2	The Man Who Shot Liberty Valance	4.60	3
3	Run Lola Run	4.50	2
4	Star Trek Into Darkness	4.45	5

References:

1. <https://www.kaggle.com/stefanoleone992/rotten-tomatoes-movies-and-critic-reviews-dataset>
2. Minqing Hu and Bing Liu. "Mining and Summarizing Customer Reviews." ; Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2004), Aug 22-25, 2004, Seattle,; Washington, USA

Useful Materials:

- Python Library
 - Python library for interfacing with MySQL (mysql-connector) [1]
 - Python Mysql Connector [1] (required)
- Tutorials
 - Python Programming [1] [2]
 - W3School (Python + Mysql) [1] (Recommended)
 - Python + Mysql [1] (Watch from 2nd video onwards)
 - Real Python (Python + Mysql) [1]
 - Mysqлтutorial [1]
- Documentation
 - MySQL Official Documentation [1]
- Input Sanitization
 - It means "checks are performed before the query is executed on the database."
 - Here is the reason why it is important to do so [1]
 - Here is the basic idea of why and how [1]
- Boiler Plate Code/ Starter Code For Database Connection And Input Sanitization
 - Before checking it, do watch tutorials or read documentation to get a complete idea.
 - [Link](#)

Problems Bumped in and Fixes:

1. Access denied to root@local host I had lower case and upper case within my original password.
Created a new user "temp" and gave all the privileges to it with a password "password"[References](#).