Ishpinder Kaur

1. The airline management system is designed to manage user information, payments, travel passes, frequent flyer details, and passport photos. This application is inspired by Scandinavian Airlines.

The application, airline management system, allows users to register, log in, manage their travel details, make payments, track their frequent flyer points, and upload passport images. The primary goal of this application is to make a basic structure for airlines which will help to streamline operations related to passenger information and loyalty programs. This will ensure efficient data handling and robust security measures.

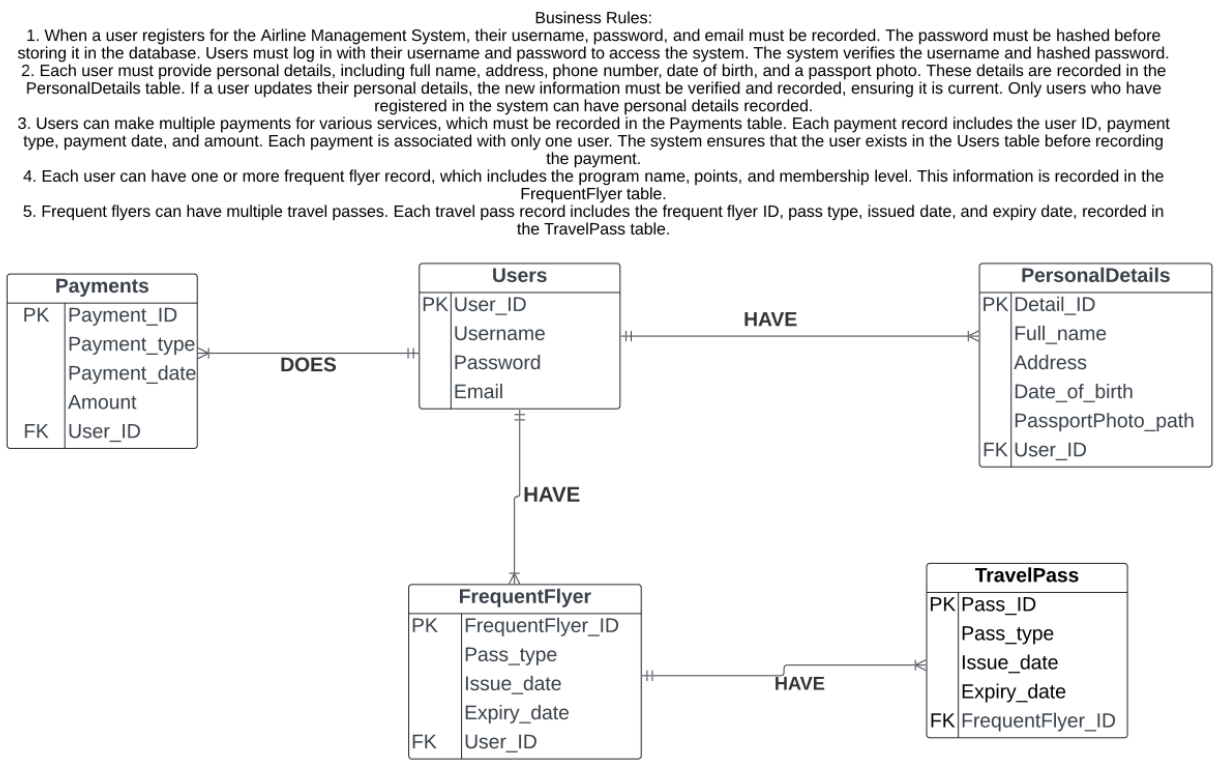
Table name	Features	Attributes
Users	User registration and login with username and password	User_ID (PK), Username, Password, Email
PersonalDetails	Management of user personal details including passport images	Detail_ID(PK), User_ID (FK), Full_name, Address, Phone_Number, Date_of_birth, PassportPhoto_path
Payments	Processing of various payment types.	Payment_ID(PK), User_ID(FK), Payment_type, Payment_date, Amount
FrequentFlyer	Tracking and management of frequent flyer points	FrequentFlyer_ID(PK), User_ID(FK), Program_name, Points, Membership_level
TravelPass	Issuance and management of travel passes for frequent travelers.	Pass_ID(PK), FrequentFlyer_ID (FK), Pass_type, Issue_date, Expiry_date

Business Rules:

1. When a user registers for the Airline Management System, their username, password, and email must be recorded. The password must be hashed before storing it in the database. Users must log in with their username and password to access the system. The system verifies the username and hashed password.
2. Each user must provide personal details, including full name, address, phone number, date of birth, and a passport photo. These details are recorded in the PersonalDetails table. If a user updates their personal details, the new information must be verified and recorded, ensuring it is current. Only users who have registered in the system can have personal details recorded.

3. Users can make multiple payments for various services, which must be recorded in the Payments table. Each payment record includes the user ID, payment type, payment date, and amount. Each payment is associated with only one user. The system ensures that the user exists in the Users table before recording the payment.
4. Each user can have one or more frequent flyer record, which includes the program name, points, and membership level. This information is recorded in the FrequentFlyer table.
5. Frequent flyers can have multiple travel passes. Each travel pass record includes the frequent flyer ID, pass type, issued date, and expiry date, recorded in the TravelPass table.

2.ERD in Crow's Foot Notation



3.Comments of database design:

- **Entity Integrity**: Each table has a primary key to uniquely identify the records. Users has User_ID, PersonalDetails has Detail_ID, Payments has Payment_ID, TravelPass has Pass_ID and FrequentFlyer has FrequentFlyer_ID
- **Referential integrity:** Each table has a foreign key that establishes its relation to other tables. This ensures data integrity.

- **Data redundancy:** The design of the tables avoids data redundancy and ensures that the data dependencies are logical. Each table is normalized to 3NF, ensuring no partial or transitive dependencies.
- **Scalability:** New tables can be added with clear relationships to existing tables through foreign keys. The structure allows for adding more detailed information or new entities (e.g. flight booking) without disrupting the existing schema.
- **Data anomalies:** Since the tables are normalized, the data anomalies are minimized.

---

4. The database is in 3NF. The tables are in 2NF and there is no transitive dependency. Hence, the database is in 3NF.

For all the tables, the attributes are fully dependent on the primary key of the table.

5.

- **BCNF:**A table is in BCNF if it is in 3NF, and every determinant is a candidate key. The database does not need BCNF as it does not have any composite keys or any other complex relationship.
- **4NF:** The database does not have any multivalued dependencies. The tables do not store any multiple values in the same record. For example, users do not store multiple email addresses or multiple phone numbers. Hence, the database does not need to change to 4NF.
- **5NF:** Since the tables do not need 3NF and 4NF, they do not need to be changed to 5NF. Each table represents a single entity. The relationships are straightforward without introducing redundancy or requiring complex joins.

The airline management system is in 3NF and it is robust. The schema prevents anomalies and maintains data integrity. Implementation of higher normal forms will not add any significant advantage to the application. It may complicate the database unnecessarily.

---

6. **a, b and c**

```
import pymysql

import csv

from sshtunnel import SSHTunnelForwarder

from passlib.hash import argon2

import pandas as pd

import os
```

```
# SSH and MySQL credentials
ssh_host = *******
ssh_port = 22
ssh_username = '*******'
ssh_key_path = *******
ssh_password = *******
mysql_host = '127.0.0.1'
mysql_port = 3306
mysql_user = *******
mysql_password = *******
mysql_db = *******

def validate_data(value, data_type):
    try:
        if data_type == int:
            return int(value)
        elif data_type == float:
            return float(value)
        elif data_type == str:
            return str(value)
        elif data_type == 'date':
            return pd.to_datetime(value).strftime('%Y-%m-%d')
        elif data_type == 'path':
            if os.path.isfile(value):
                with open(value, 'rb') as file:
                    return file.read()
            else:
                raise ValueError(f"File not found: {value}")
    except ValueError as e:
```

```
        print(f"Validation error: {e}")
        return None

# Create an SSH tunnel and connect to the database
with SSHTunnelForwarder(
        (ssh_host, ssh_port),
        ssh_username=ssh_username,
        ssh_password=ssh_password,
        ssh_pkey=ssh_key_path,
        remote_bind_address=(mysql_host, mysql_port)
) as tunnel:
    connection = pymysql.connect(
        host='127.0.0.1',
        user=mysql_user,
        password=mysql_password,
        database=mysql_db,
        port=tunnel.local_bind_port,
    )

    try:
        with connection.cursor() as cursor:


                # Create tables
            cursor.execute("DROP TABLE IF EXISTS Users")
            cursor.execute("""
                CREATE TABLE IF NOT EXISTS Users (
                    User_ID INTEGER PRIMARY KEY,
                    Username VARCHAR(255) NOT NULL UNIQUE,
                    Password TEXT NOT NULL,
```

```
        Email VARCHAR(255) NOT NULL UNIQUE
    )
""")
cursor.execute("""
    CREATE TABLE IF NOT EXISTS PersonalDetails (
        Detail_ID INTEGER PRIMARY KEY,
        User_ID INTEGER,
        Full_name TEXT NOT NULL,
        Address TEXT NOT NULL,
        Phone_number TEXT NOT NULL,
        Date_of_birth DATE NOT NULL,
        PassportPhoto_path LONGBLOB NOT NULL,
        FOREIGN KEY (User_ID) REFERENCES Users (User_ID) ON DELETE CASCADE
    )
""")
cursor.execute("""
    CREATE TABLE IF NOT EXISTS Payments (
        Payment_ID INTEGER PRIMARY KEY,
        User_ID INTEGER,
        Payment_type TEXT NOT NULL,
        Payment_date DATE NOT NULL,
        Amount REAL NOT NULL,
        FOREIGN KEY (User_ID) REFERENCES Users (User_ID) ON DELETE CASCADE
    )
""")
cursor.execute("""
    CREATE TABLE IF NOT EXISTS FrequentFlyer (
        FrequentFlyer_ID INTEGER PRIMARY KEY,
        User_ID INTEGER,
```

```
            Program_name TEXT NOT NULL,
            Points INTEGER NOT NULL,
            Membership_level TEXT NOT NULL,
            FOREIGN KEY (User_ID) REFERENCES Users (User_ID) ON DELETE CASCADE
        )
    """)
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS TravelPass (
            Pass_ID INTEGER PRIMARY KEY,
            FrequentFlyer_ID INTEGER,
            Pass_type TEXT NOT NULL,
            Issue_date DATE NOT NULL,
            Expiry_date DATE NOT NULL,
            FOREIGN KEY (FrequentFlyer_ID) REFERENCES FrequentFlyer (FrequentFlyer_ID) ON
DELETE CASCADE
        )
    """)
    connection.commit()


#users table
    with open('Users.csv', mode='r', encoding='utf-8-sig') as file:
        csv_reader = csv.reader(file)


        for row in csv_reader:
            user_id = validate_data(row[0], int)
            username = validate_data(row[1], str)
            password = validate_data(row[2], str)
            email = validate_data(row[3], str)
            if None in [user_id, username, password, email]:
```

```
            print(f"Skipping invalid row: {row}")
            continue
        hashed_password = argon2.hash(password)#hashing
        cursor.execute("""
            INSERT INTO Users (User_ID, Username, Password, Email)
            VALUES (%s, %s, %s, %s)
        """, (user_id, username, hashed_password, email))
    connection.commit()
    print("Data imported successfully for Users")


# PersonalDetails
with open('PersonalDetails.csv', mode='r', encoding='utf-8-sig') as file:
    csv_reader = csv.reader(file)


    for row in csv_reader:
        detail_id = validate_data(row[0], int)
        user_id = validate_data(row[1], int)
        full_name = validate_data(row[2], str)
        address = validate_data(row[3], str)
        phone_number = validate_data(row[4], str)
        date_of_birth = validate_data(row[5], 'date')
        passport_photo = validate_data(row[6], 'path')
        if None in [detail_id, user_id, full_name, address, phone_number, date_of_birth,
passport_photo]:
            print(f"Skipping invalid row: {row}")
            continue
        cursor.execute("""
            INSERT INTO PersonalDetails (Detail_ID, User_ID, Full_name, Address, Phone_number,
Date_of_birth, PassportPhoto_path)
```

```
            VALUES (%s, %s, %s, %s, %s, %s, %s)
        """, (detail_id, user_id, full_name, address, phone_number, date_of_birth,
passport_photo))
    connection.commit()
print("Data imported successfully for PersonalDetails")

#Payments
with open('Payments.csv', mode='r', encoding='utf-8-sig') as file:
    csv_reader = csv.reader(file)

    for row in csv_reader:
        payment_id = validate_data(row[0], int)
        user_id = validate_data(row[1], int)
        payment_type = validate_data(row[2], str)
        payment_date = validate_data(row[3], 'date')
        amount = validate_data(row[4], float)
        if None in [payment_id, user_id, payment_type, payment_date, amount]:
            print(f"Skipping invalid row: {row}")
            continue
        cursor.execute("""
            INSERT INTO Payments (Payment_ID, User_ID, Payment_type, Payment_date, Amount)
            VALUES (%s, %s, %s, %s, %s)
        """, (payment_id, user_id, payment_type, payment_date, amount))
    connection.commit()
print("Data imported successfully for Payments")

#FrequentFlyer
with open('FrequentFlyer.csv', mode='r', encoding='utf-8-sig') as file:
    csv_reader = csv.reader(file)
```

```
        for row in csv_reader:
            frequent_flyer_id = validate_data(row[0], int)
            user_id = validate_data(row[1], int)
            program_name = validate_data(row[2], str)
            points = validate_data(row[3], int)
            membership_level = validate_data(row[4], str)
            if None in [frequent_flyer_id, user_id, program_name, points, membership_level]:
                print(f"Skipping invalid row: {row}")
                continue
            cursor.execute("""
                INSERT INTO FrequentFlyer (FrequentFlyer_ID, User_ID, Program_name, Points,
Membership_level)
                VALUES (%s, %s, %s, %s, %s)
            """, (frequent_flyer_id, user_id, program_name, points, membership_level))
        connection.commit()
    print("Data imported successfully for FrequentFlyer")

    # Insert data into TravelPass table
    with open('TravelPass.csv', mode='r', encoding='utf-8-sig') as file:
        csv_reader = csv.reader(file)

        for row in csv_reader:
            pass_id = validate_data(row[0], int)
            frequent_flyer_id = validate_data(row[1], int)
            pass_type = validate_data(row[2], str)
            issue_date = validate_data(row[3], 'date')
            expiry_date = validate_data(row[4], 'date')
            if None in [pass_id, frequent_flyer_id, pass_type, issue_date, expiry_date]:
```

```
                print(f"Skipping invalid row: {row}")
                continue
            cursor.execute("""
                INSERT INTO TravelPass (Pass_ID, FrequentFlyer_ID, Pass_type, Issue_date,
Expiry_date)
                VALUES (%s, %s, %s, %s, %s)
            """, (pass_id, frequent_flyer_id, pass_type, issue_date, expiry_date))
        connection.commit()
    print("Data imported successfully for TravelPass")


finally:
    connection.close()
```

d) Delete method

```
= RESTART: C:\Users\91828\Desktop\AMOD\Semester 2\Intro to databases\Assignments
\Final assignment\Testing.py
User with User_ID 11 removed successfully
Personal details with Detail_ID 13 removed successfully
Payment with Payment_ID 13 removed successfully
```

```
import pymysql
from sshtunnel import SSHTunnelForwarder


# SSH and MySQL credentials
ssh_host = 'loki.trentu.ca'
ssh_port = 22
ssh_username = '*******'
ssh_key_path = '*******'
ssh_password = *******
mysql_host = '127.0.0.1'
mysql_port = 3306
mysql_user = *******
```

```
mysql_password = *******
mysql_db = '*******'

# remove a user
def remove_user(cursor, user_id):
    cursor.execute("DELETE FROM Users WHERE User_ID = %s", (user_id,))
    connection.commit()
    print(f"User with User_ID {user_id} removed successfully")

# remove personal details
def remove_personal_details(cursor, detail_id):
    cursor.execute("DELETE FROM PersonalDetails WHERE Detail_ID = %s", (detail_id,))
    connection.commit()
    print(f"Personal details with Detail_ID {detail_id} removed successfully")

#remove a payment
def remove_payment(cursor, payment_id):
    cursor.execute("DELETE FROM Payments WHERE Payment_ID = %s", (payment_id,))
    connection.commit()
    print(f"Payment with Payment_ID {payment_id} removed successfully")

#  remove a frequent flyer
def remove_frequent_flyer(cursor, frequentflyer_id):
    cursor.execute("DELETE FROM FrequentFlyer WHERE FrequentFlyer_ID = %s", (frequentflyer_id,))
    connection.commit()
    print(f"Frequent flyer with FrequentFlyer_ID {frequent_flyer_id} removed successfully")

# remove a travel pass
def remove_travel_pass(cursor, pass_id):
```

```
    cursor.execute("DELETE FROM TravelPass WHERE Pass_ID = %s", (pass_id,))
    connection.commit()
    print(f"Travel pass with Pass_ID {pass_id} removed successfully")

# Create an SSH tunnel
with SSHTunnelForwarder(
        (ssh_host, ssh_port),
        ssh_username=ssh_username,
        ssh_password=ssh_password,
        ssh_pkey=ssh_key_path,
        remote_bind_address=(mysql_host, mysql_port)
) as tunnel:
    connection = pymysql.connect(
        host='127.0.0.1',
        user=mysql_user,
        password=mysql_password,
        database=mysql_db,
        port=tunnel.local_bind_port,
    )

    try:
        with connection.cursor() as cursor:
            # Example usage of remove_user
            user_id_to_remove = 11  # any user id can be added from db
            remove_user(cursor, user_id_to_remove)

            # remove_personal_details example
            detail_id_to_remove = 13  # any Detail_id can be added from db
            remove_personal_details(cursor, detail_id_to_remove)
```

```
    # remove_payment example
    payment_id_to_remove = 13  # any PAyment_id can be added from db
    remove_payment(cursor, payment_id_to_remove)

    # remove_frequentFlyer example
    frequentflyer_id_to_remove = 20  # any Frequentflyer_id can be added from db
    remove_frequent_flyer(cursor, frequentflyer_id_to_remove)

    # remove_travel_pass example
    pass_id_to_remove =11   # any pass_id can be added from db
    remove_travel_pass(cursor, pass_id_to_remove)

  finally:
    connection.close()
```

e)

- **Update User Email:** User information changes often. One of the most common things that changes is the email address of the user. Implementing a method to update the email address of a user based on its User_ID will ensure that the information remains current and accurate.
- **Retrieve User Details:** To view and verify the details of the user, retrieving user details is necessary. This method can be used for various purposes, such as displaying user information in the application, verifying user credentials, or even for administrative purposes.

```
= RESTART: C:\Users\91828\Desktop\AMOD\Semester 2\Intro to databases\Assignments
\Final assignment\Testing.py
User email updated successfully for User_ID 1
User details for User_ID 1: (1, 'alice', '$argon2id$v=19$m=65536,t=3,p=4$4HwPodR
6z713LgVAqLVWyg$VsYnkPGb0PD31pgeJIVZFyfRbm+AglT/DBag+nF8JmI', 'newemail@example.
com')
```

```
import pymysql
```

```
from sshtunnel import SSHTunnelForwarder

# SSH and MySQL credentials
ssh_host = 'loki.trentu.ca'
ssh_port = 22
ssh_username = *******
ssh_key_path = *******
ssh_password = *******
mysql_host = '127.0.0.1'
mysql_port = 3306
mysql_user = *******
mysql_password = *******
mysql_db = *******

# update user email
def update_user_email(cursor, user_id, new_email):
    cursor.execute("UPDATE Users SET Email = %s WHERE User_ID = %s", (new_email, user_id))
    connection.commit()
    print(f"User email updated successfully for User_ID {user_id}")

# retrieve user details
def retrieve_user_details(cursor, user_id):
    cursor.execute("SELECT * FROM Users WHERE User_ID = %s", (user_id,))
    user_details = cursor.fetchone()
    if user_details:
        print(f"User details for User_ID {user_id}: {user_details}")
        return user_details
    else:
        print(f"No user found with User_ID {user_id}")
```

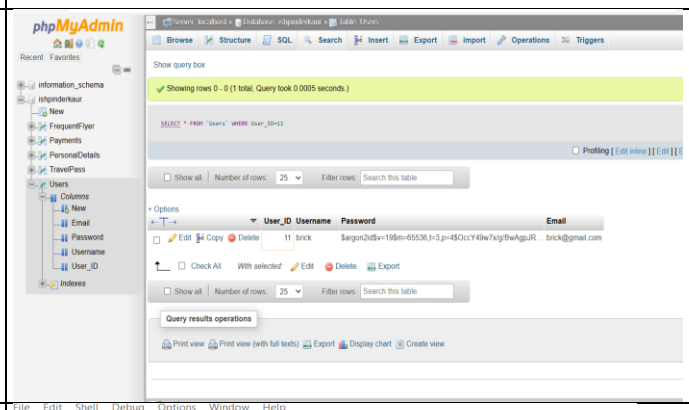
```
        return None

# Create an SSH tunnel
with SSHTunnelForwarder(
        (ssh_host, ssh_port),
        ssh_username=ssh_username,
        ssh_password=ssh_password,
        ssh_pkey=ssh_key_path,
        remote_bind_address=(mysql_host, mysql_port)
) as tunnel:
    connection = pymysql.connect(
        host='127.0.0.1',
        user=mysql_user,
        password=mysql_password,
        database=mysql_db,
        port=tunnel.local_bind_port,
    )

    try:
        with connection.cursor() as cursor:
            # Example usage of update_user_email
            user_id_to_update = 1
            new_email = "newemail@example.com"  # Replace with new email of customer(user)
            update_user_email(cursor, user_id_to_update, new_email)

            # Example usage of retrieve_user_details
            user_id_to_retrieve = 1
            user_details = retrieve_user_details(cursor, user_id_to_retrieve)
```

finally:

connection.close()

---

7. Test cases

Test Case no.	Scenario	Input	Expected output	Output
U1	Insert new user details with user id 11	user_id = 11 username = "brick" password = "abcdef12345" email = "brick@gmail.com"	Row added successfully, able to fetch row	
U2	Insert duplicate value in the table Users	user_id = 7 username = "judy ter" password = "password123" email = "terjudy@example.com"	Integrity error	

U3	Insert wrong data type in the table Users for the value user ID	user_id = 1cc username = "judy ter" password = "password123" email = "terjudy@example.com"	Validation Error	IDLE Shell 3.12.3 File Edit Shell Debug Options Window Help Python 3.12.3 (tags/v3.12.3:f6650f9, Apr 9 2024, 14:05:25) [MSC v.1938 64 bit (AMD64)] on win32 Type "help", "copyright", "credits" or "license()" for more information. >>> = RESTART: C:/Users/91828/Desktop/AMOD/Semester 2/Intro to databases/Assignments /Final assignment/Testing.py Validation error: invalid literal for int() with base 10: '1cc' Skipping invalid row: ['1cc', 'judy ter', 'password123', 'terjudy@example.com'] Data imported successfully for Users >>>
U4	Insert duplicate value for username in Users	user_id = 12 username = judy password = "pass123" email = "judy@example.com"	Integrity error	= RESTART: C:/Users/91828/Desktop/AMOD/Semester 2/Intro to databases/Assignments /Final assignment/Testing.py Traceback (most recent call last): File "C:/Users/91828/Desktop/AMOD/Semester 2/Intro to databases/Assignments/Final assignment/Testing.py", line 72, in <module> cursor.execute(""" File "C:\Users\91828\AppData\Local\Programs\Python\Python312\Lib\site-packages\pymysql\cursors.py", line 153, in execute result = self._query(query) File "C:\Users\91828\AppData\Local\Programs\Python\Python312\Lib\site-packages\pymysql\cursors.py", line 322, in _query conn.query(q) File "C:\Users\91828\AppData\Local\Programs\Python\Python312\Lib\site-packages\pymysql\connections.py", line 563, in query self._affected_rows = self._read_query_result(unbuffered=unbuffered) File "C:\Users\91828\AppData\Local\Programs\Python\Python312\Lib\site-packages\pymysql\connections.py", line 825, in _read_query_result result.read() File "C:\Users\91828\AppData\Local\Programs\Python\Python312\Lib\site-packages\pymysql\connections.py", line 1199, in read first_packet = self.connection._read_packet() File "C:\Users\91828\AppData\Local\Programs\Python\Python312\Lib\site-packages\pymysql\connections.py", line 775, in _read_packet packet.raise_for_error() File "C:\Users\91828\AppData\Local\Programs\Python\Python312\Lib\site-packages\pymysql\protocol.py", line 219, in raise_for_error err.raise_mysql_exception(self._data) File "C:\Users\91828\AppData\Local\Programs\Python\Python312\Lib\site-packages\pymysql\err.py", line 150, in raise_mysql_exception raise errorclass(errno, errval) pymysql.err.IntegrityError: (1062, "Duplicate entry 'judy' for key 'Username'")
PD 1	Insert valid personal detail	Detail_id=11 User_ID=5, Full_name="Eve Strong" address=123 Elm Strr Phone_number 555-12346 Date_of Birth=1990-01-01	Detail added successfully	IDLE Shell 3.12.3 File Edit Shell Debug Options Window Help Python 3.12.3 (tags/v3.12.3:f6650f9, Apr 9 2024, 14:05:25) [MSC v.1 AMD64)] on win32 Type "help", "copyright", "credits" or "license()" for more informat >>> = RESTART: C:\Users\91828\Desktop\AMOD\Semester 2\Intro to databases \Final assignment\Testing.py Data imported successfully for PersonalDetails >>>
PD 2	Insert Personal Detail with non existing User_ID	Detail_id=12 User_ID=, Full_name="Jibb Strong" address=123 Elm Strr Phone_number 555-1238 Date_of Birth=1990-01-01	Foreign Key constraint	IDLE Shell 3.12.3 File Edit Shell Debug Options Window Help Python 3.12.3 (tags/v3.12.3:f6650f9, Apr 9 2024, 14:05:25) [MSC v.1938 64 bit (AMD64)] on win32 Type "help", "copyright", "credits" or "license()" for more information. >>> = RESTART: C:\Users\91828\Desktop\AMOD\Semester 2\Intro to d atabases\Assignments\Final assignment\Testing.py Validation error: invalid literal for int() with base 10: '' Skipping invalid row: ['11', '', 'Eve strong', '123 Elm Strr ', '555-12346', '1/1/1990', 'C:/Users/91828/Desktop/AMOD/Sem ester 2/Intro to databases/Assignments/Final assignment/11.j pg'] >>>

P1	Insert Valid Payment details	Payment_Id=13 User_ID=6 Payment_type=Credit Card Payment_date=2025-01-10 Amount=549	Data imported successfully	File Edit Shell Debug Options Window Help Python 3.12.3 (tags/v3.12.3:f6650f9, Apr 9 2024, 14:05:25) [M AMD64)] on win32 Type "help", "copyright", "credits" or "license()" for more in >>> = RESTART: C:\Users\91828\Desktop\AMOD\Semester 2\Intro to dat \Final assignment\Testing.py >>> = RESTART: C:\Users\91828\Desktop\AMOD\Semester 2\Intro to dat \Final assignment\Testing.py >>> = RESTART: C:\Users\91828\Desktop\AMOD\Semester 2\Intro to dat \Final assignment\Testing.py Data imported successfully for Payments >>>
P2	Insert NULL for Date	Payment_Id=14 User_ID=7 Payment_type=Credit Card Payment_date=NULL Amount=549	Validation Error	IDLE Shell 3.12.3 File Edit Shell Debug Options Window Help Python 3.12.3 (tags/v3.12.3:f6650f9, Apr 9 2024, 14:05:25) [MSC v.1938 64 AMD64)] on win32 Type "help", "copyright", "credits" or "license()" for more information. >>> = RESTART: C:\Users\91828\Desktop\AMOD\Semester 2\Intro to databases\Assig \Final assignment\Testing.py Validation error: NaTType does not support strftime Skipping invalid row: ['14', '7', 'Credit Card', '', '549'] >>>
P3	Insert Payment with non-existing User_ID	Payment_Id=14 User_ID=77 Payment_type=Credit Card Payment_date=2025-01-10 Amount=549	Foreign Key constraint	IDLE Shell 3.12.3 File Edit Shell Debug Options Window Help Python 3.12.3 (tags/v3.12.3:f6650f9, Apr 9 2024, 14:05:25) [MSC v.1938 64 bit (AMD64)] on win32 Type "help", "copyright", "credits" or "license()" for more information. >>> = RESTART: C:\Users\91828\Desktop\AMOD\Semester 2\Intro to databases\Assignments\Final assignment\Test ing.py Traceback (most recent call last): File "C:\Users\91828\Desktop\AMOD\Semester 2\Intro to databases\Assignments\Final assignment\Testing .py", line 71, in <module> cursor.execute(""" File "C:\Users\91828\AppData\Local\Programs\Python\Python312\Lib\site-packages\pymysql\cursors.py", line 153, in execute result = self._query(query) File "C:\Users\91828\AppData\Local\Programs\Python\Python312\Lib\site-packages\pymysql\cursors.py", line 322, in _query conn.query(q) File "C:\Users\91828\AppData\Local\Programs\Python\Python312\Lib\site-packages\pymysql\connections.p y", line 563, in query self._affected_rows = self._read_query_result(unbuffered=unbuffered) File "C:\Users\91828\AppData\Local\Programs\Python\Python312\Lib\site-packages\pymysql\connections.p y", line 825, in _read_query_result result.read() File "C:\Users\91828\AppData\Local\Programs\Python\Python312\Lib\site-packages\pymysql\connections.p y", line 1199, in read first_packet = self.connection._read_packet() File "C:\Users\91828\AppData\Local\Programs\Python\Python312\Lib\site-packages\pymysql\connections.p y", line 775, in _read_packet packet.raise_for_error() File "C:\Users\91828\AppData\Local\Programs\Python\Python312\Lib\site-packages\pymysql\protocol.py", line 219, in raise_for_error err.raise_mysql_exception(self._data) File "C:\Users\91828\AppData\Local\Programs\Python\Python312\Lib\site-packages\pymysql\err.py", line 150, in raise_mysql_exception raise errorclass(errno, errval) pymysql.err.IntegrityError: (1452, 'Cannot add or update a child row: a foreign key constraint fails (`ishpinderkaur`.`Payments`, CONSTRAINT `Payments_ibfk_1` FOREIGN KEY (`User_ID`) REFERENCES `Users` (` User_ID`) ON DELETE CASCADE)') >>>

P4	Insert payment with wrong data type for amount	Payment_Id=14 User_ID=7 Payment_type=Credit Card Payment_date=2025-01-10 Amount="fifty"	Validation error, skipping invalid row	IDLE Shell 3.12.3 File Edit Shell Debug Options Window Help Python 3.12.3 (tags/v3.12.3:f6650f9, Apr 9 2024, 14:05:25) [MSC v.1938 64 bit (AMD64)] on win32 Type "help", "copyright", "credits" or "license()" for more information. >>> = RESTART: C:\Users\91828\Desktop\AMOD\Semester 2\Intro to databases\Assignments\Final assignment\Testing.py Validation error: could not convert string to float: 'fifty' Skipping invalid row: ['14', '7', 'Credit Card', '1/10/2025', 'fifty'] >>>
FF 1	Insert valid Frequent Flyer details into the table	FrequentFlyer_ID=11 User_ID=3 Program_name=Rapid Rewards Points=9500 Membership_level=Bronze	Data imported successfully	>>> = RESTART: C:\Users\91828\Desktop\AMOD\Semeste \Final assignment\Testing.py Data imported successfully for FrequentFlyer >>>
FF 2	Insert Frequent Flyer with non-existing User_ID	FrequentFlyer_ID=12 User_ID=33 Program_name=Instant Rewards Points=4000 Membership_level=Bronze	Foreign Key Constraint Fails	File Edit Shell Debug Options Window Help Python 3.12.3 (tags/v3.12.3:f6650f9, Apr 9 2024, 14:05:25) [MSC v.1938 64 bit (AMD64)] on win32 Type "help", "copyright", "credits" or "license()" for more information. >>> = RESTART: C:\Users\91828\Desktop\AMOD\Semester 2\Intro to databases\Assignments\Final assignment\Testing.py Traceback (most recent call last): File "C:\Users\91828\Desktop\AMOD\Semester 2\Intro to databases\Assignments\Final assignment\Testing.py", line 71, in <module> cursor.execute(""" File "C:\Users\91828\AppData\Local\Programs\Python\Python312\Lib\site-packages\pymysql\cursors.py", line 153, in execute result = self._query(query) File "C:\Users\91828\AppData\Local\Programs\Python\Python312\Lib\site-packages\pymysql\cursors.py", line 322, in _query conn.query(q) File "C:\Users\91828\AppData\Local\Programs\Python\Python312\Lib\site-packages\pymysql\connections.py", line 563, in query self._affected_rows = self._read_query_result(unbuffered=unbuffered) File "C:\Users\91828\AppData\Local\Programs\Python\Python312\Lib\site-packages\pymysql\connections.py", line 825, in _read_query_result result.read() File "C:\Users\91828\AppData\Local\Programs\Python\Python312\Lib\site-packages\pymysql\connections.py", line 1199, in read first_packet = self.connection._read_packet() File "C:\Users\91828\AppData\Local\Programs\Python\Python312\Lib\site-packages\pymysql\connections.py", line 775, in _read_packet packet.raise_for_error() File "C:\Users\91828\AppData\Local\Programs\Python\Python312\Lib\site-packages\pymysql\protocol.py", line 219, in raise_for_error err.raise_mysql_exception(self._data) File "C:\Users\91828\AppData\Local\Programs\Python\Python312\Lib\site-packages\pymysql\err.py", line 150, in raise_mysql_exception raise errorclass(errno, errval) pymysql.err.IntegrityError: (1452, 'Cannot add or update a child row: a foreign key constraint fails (`ishpinderkaur`.`FrequentFlyer`, CONSTRAINT `FrequentFlyer_ibfk_1` FOREIGN KEY (`User_ID`) REFERENCES `Users` (`User_ID`) ON DELETE CASCADE)') >>>
FF 3	Insert Frequent Flyer with wrong data type for points	FrequentFlyer_ID=22 User_ID=33 Program_name=Instant Rewards Points=fourty	Validation error, row skipped	File Edit Shell Debug Options Window Help Python 3.12.3 (tags/v3.12.3:f6650f9, Apr 9 2024, 14:05:25) [MSC v.1938 64 bi AMD64)] on win32 Type "help", "copyright", "credits" or "license()" for more information. >>> = RESTART: C:\Users\91828\Desktop\AMOD\Semester 2\Intro to databases\Assignme \Final assignment\Testing.py Validation error: invalid literal for int() with base 10: 'forty' Skipping invalid row: ['22', '33', 'Instant Rapid Rewards', 'forty', 'Bronze' Data imported successfully for FrequentFlyer >>>

		Membership_level=Bronze		
TP 1	Insert valid Travell Pass details	Pass_ID=11 FrequentFlyer_ID=12 Pass_type="10_Fly PAss" Issue_date=2024-05-01 Expiry_date=2025-04-30	Data inserted successfully	= RESTART: C:\Users\91828\Desktop\AMOD\Semester \Final assignment\Testing.py Data imported successfully for TravelPass
TP 2	Insert Travel Pass with non-existing Frequent Flyer_ID	Pass_ID=13 FrequentFlyer_ID=33 Pass_type="5_Fly PAss" Issue_date=2024-05-01 Expiry_date=2025-04-30	Foreign Key constraint fail	= RESTART: C:\Users\91828\Desktop\AMOD\Semester 2\Intro to databases\Assignments \Final assignment\Testing.py Traceback (most recent call last): File "C:\Users\91828\Desktop\AMOD\Semester 2\Intro to databases\Assignments\Final assignment\Testing.py", line 71, in <module> cursor.execute(""" File "C:\Users\91828\AppData\Local\Programs\Python\Python312\Lib\site-packages\pymysql\cursors.py", line 153, in execute result = self._query(query) File "C:\Users\91828\AppData\Local\Programs\Python\Python312\Lib\site-packages\pymysql\cursors.py", line 322, in _query conn.query(q) File "C:\Users\91828\AppData\Local\Programs\Python\Python312\Lib\site-packages\pymysql\connections.py", line 563, in query self._affected_rows = self._read_query_result(unbuffered=unbuffered) File "C:\Users\91828\AppData\Local\Programs\Python\Python312\Lib\site-packages\pymysql\connections.py", line 825, in _read_query_result result.read() File "C:\Users\91828\AppData\Local\Programs\Python\Python312\Lib\site-packages\pymysql\connections.py", line 1199, in read first_packet = self.connection._read_packet() File "C:\Users\91828\AppData\Local\Programs\Python\Python312\Lib\site-packages\pymysql\connections.py", line 775, in _read_packet packet.raise_for_error() File "C:\Users\91828\AppData\Local\Programs\Python\Python312\Lib\site-packages\pymysql\protocol.py", line 219, in raise_for_error err.raise_mysql_exception(self._data) File "C:\Users\91828\AppData\Local\Programs\Python\Python312\Lib\site-packages\pymysql\err.py", line 150, in raise_mysql_exception raise errorclass(errno, errval) pymysql.err.IntegrityError: (1452, 'Cannot add or update a child row: a foreign key constraint fails (`ishpinderkaur`.`TravelPass`, CONSTRAINT `TravelPass_ibfk_1` FOREIGN KEY (`FrequentFlyer_ID`) REFERENCES `FrequentFlyer` (`FrequentFlyer_ID`) ON DELETE CASCADE)')
TP 3	Delete user with Pass_ID =3	DELETE FROM TravelPass WHERE Pass_ID = 3	User deleted successfully	Python 3.12.3 (tags/v3.12.3:f6650f9, Apr 9 2024, 14:05:25) [MS AMD64)] on win32 Type "help", "copyright", "credits" or "license()" for more in = RESTART: C:\Users\91828\Desktop\AMOD\Semester 2\Intro to data \Final assignment\Testing.py Travel pass with Pass_ID 9 removed successfully