AMOD 5450 – Intro to Database

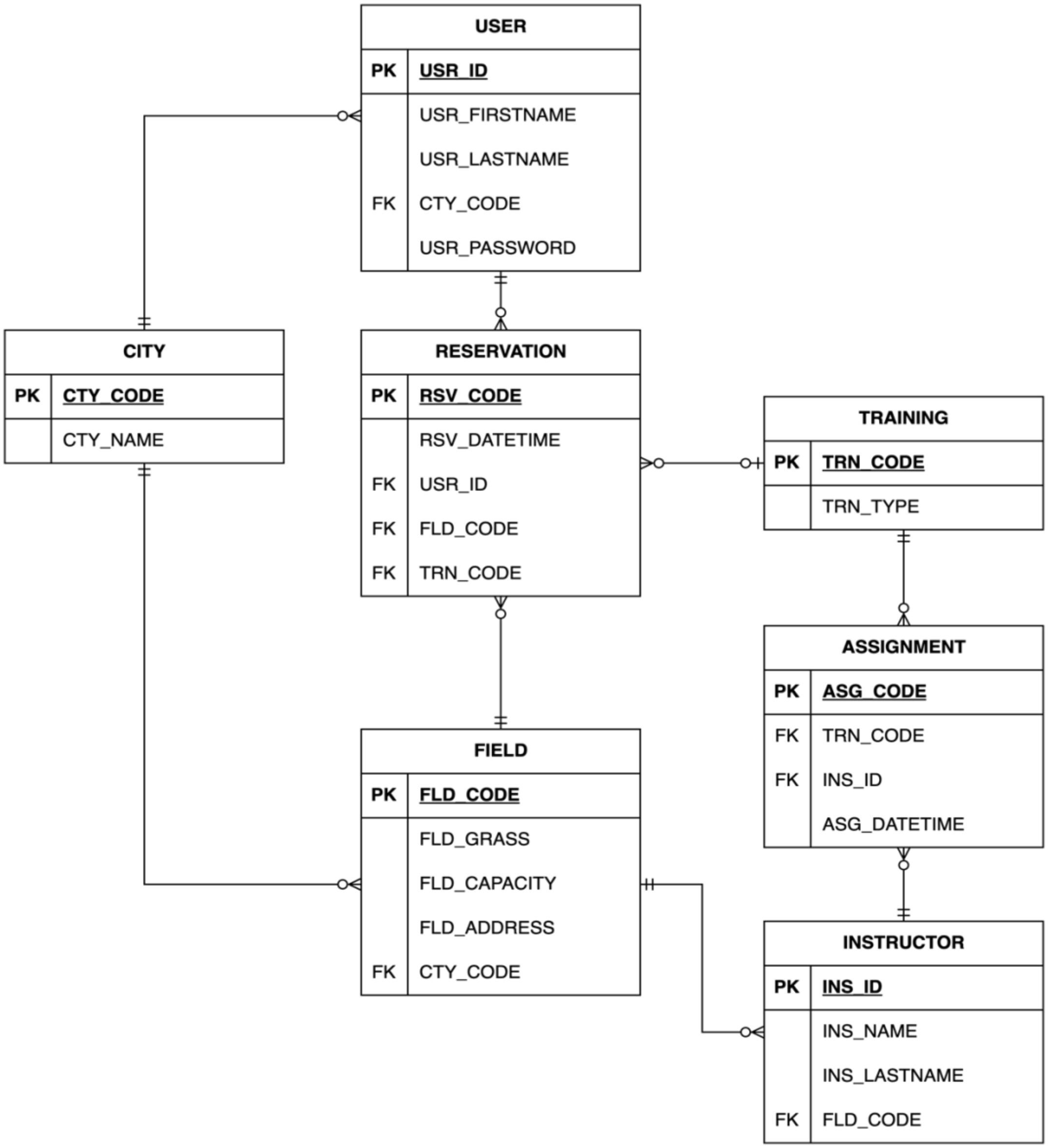
Diego Brito – 0814117

Assignment 3

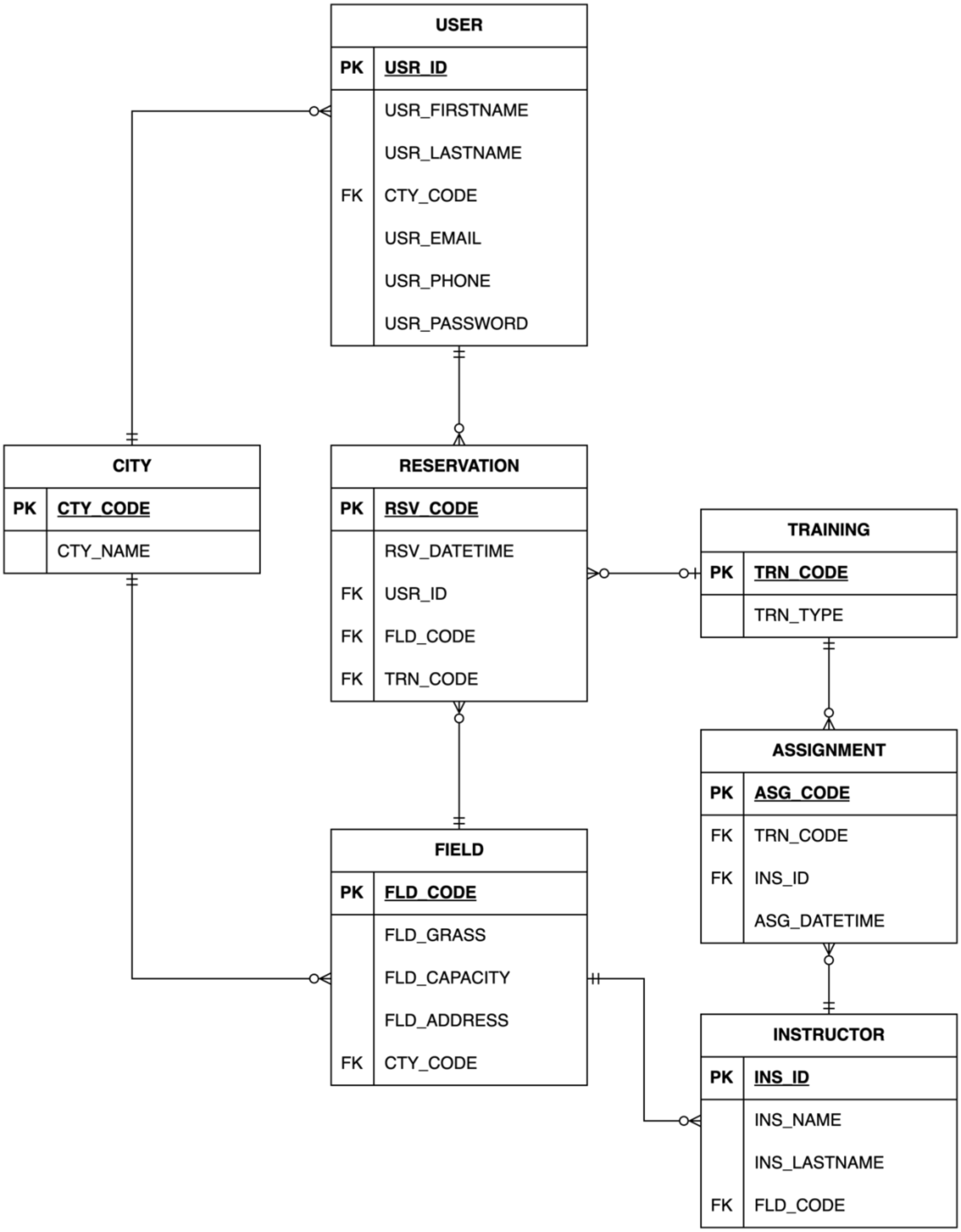
1. This app lets you book soccer fields across Ontario. The company has many fields with different sizes and grass types in popular cities.
   They also offer training services. You can choose to work on attack, defense, tactics, fitness, and more. Each field has instructors who can do any type of training.
   If you just want to play with friends, you can book a field without training.
   To use the app:
   - Log in
   - Pick your city
   - Choose a field
   - Select training (if you want it) or just book for playing

   It's simple to find and reserve a field for your game or practice.

2.



3. My ERD seems to be well constructed; it keeps referential integrity as foreign keys are properly established and ensure data consistency across tables. The relationships between entities are well-defined and meets all the requirements of the application. However, I believe that adding more contact information to the user table will add a more complete profile of the user and will allow to use an email as the username to login. The resulting ERD:



4. My ERD design appears to be in 3NF. There are no apparent partial or transitive dependencies.

5. The current 3NF design have a good balance between normalization, performance and practical use for the application. The “benefits” of moving to higher normal forms don’t outweigh the extra complexity and potential performance issues.

6. As

   a. Ss

```
ssh_host = 'loki.trentu.ca'#'192.197.151.116'
ssh_port = 22  # Default SSH port
ssh_username = 'diegobrito'  # Enter your username
ssh_key_path = '/Users/dalonsobc/.ssh/diegobrito.private'  # Enter your private key path, if your private key is in the same directory as your script, all you have to provide is the name of the file.
ssh_password = 'lok1P@ssphrase'  # Enter your password
mysql_host = '127.0.0.1'  # This should be '127.0.0.1' because you're connecting via the tunnel
mysql_port = 3306  # Default MySQL port
mysql_user = 'diegobrito'  # Enter your phpMyAdmin User
mysql_password = 'myTren$0' # Enter your password
mysql_db = 'diegobrito'  # Enter your db name (should be named after you)
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',  # This is where pymysql connects
        user=mysql_user,
        password=mysql_password,
        database=mysql_db,
        port=tunnel.local_bind_port,
        # Use the local port assigned by sshtunnel
    )
```

   b. As

```
create_statements = [
"""
CREATE TABLE CITY (
    CTY_CODE CHAR(5) PRIMARY KEY,
    CTY_NAME VARCHAR(50) NOT NULL
);
""",
"""
CREATE TABLE USER (
    USR_ID CHAR(10) PRIMARY KEY,
    USR_FIRSTNAME VARCHAR(50) NOT NULL,
    USR_LASTNAME VARCHAR(50) NOT NULL,
    CTY_CODE CHAR(5),
    USR_EMAIL VARCHAR(100) NOT NULL,
    USR_PHONE VARCHAR(20),
    USR_PASSWORD VARCHAR(255) NOT NULL,
    FOREIGN KEY (CTY_CODE) REFERENCES CITY(CTY_CODE)
);
""",
"""
CREATE TABLE FIELD (
    FLD_CODE CHAR(10) PRIMARY KEY,
    FLD_GRASS LONGBLOB,
    FLD_CAPACITY INT,
    FLD_ADDRESS VARCHAR(100),
    CTY_CODE CHAR(5),
    FOREIGN KEY (CTY_CODE) REFERENCES CITY(CTY_CODE)
);
""",
"""
CREATE TABLE TRAINING (
    TRN_CODE CHAR(10) PRIMARY KEY,
    TRN_TYPE VARCHAR(50) NOT NULL
);
""",
"""
CREATE TABLE RESERVATION (
    RSV_CODE CHAR(10) PRIMARY KEY,
        RSV_DATETIME DATETIME NOT NULL,
        USR_ID CHAR(10),
        FLD_CODE CHAR(10),
        TRN_CODE CHAR(10),
        FOREIGN KEY (USR_ID) REFERENCES USER(USR_ID),
        FOREIGN KEY (FLD_CODE) REFERENCES FIELD(FLD_CODE),
        FOREIGN KEY (TRN_CODE) REFERENCES TRAINING(TRN_CODE)
    );
    """,
    """
    CREATE TABLE INSTRUCTOR (
        INS_ID CHAR(10) PRIMARY KEY,
        INS_NAME VARCHAR(50) NOT NULL,
        INS_LASTNAME VARCHAR(50) NOT NULL,
        FLD_CODE CHAR(10),
        FOREIGN KEY (FLD_CODE) REFERENCES FIELD(FLD_CODE)
    );
    """,
    """
    CREATE TABLE ASSIGNMENT (
        ASG_CODE CHAR(10) PRIMARY KEY,
        TRN_CODE CHAR(10),
        INS_ID CHAR(10),
        ASG_DATETIME DATETIME NOT NULL,
        FOREIGN KEY (TRN_CODE) REFERENCES TRAINING(TRN_CODE),
        FOREIGN KEY (INS_ID) REFERENCES INSTRUCTOR(INS_ID)
    );
    """
]
    try:
        with connection.cursor() as cursor:

            for statement in create_statements:
                cursor.execute(statement)
            print("Tables created successfully")
    except Exception as e:
        print(e)
    finally:
        connection.close()
```

c. Dd

```
# -------------------------- HELPER METHODS ------------------------------

def convert_to_binary(file_path):
    with open(file_path, 'rb') as file:
        binary_data = file.read()
    return binary_data

def hash_password(password):
    salt = hashlib.sha256(os.urandom(60)).hexdigest().encode('ascii')
    pwdhash = hashlib.pbkdf2_hmac('sha512', password.encode('utf-8'), salt, 100000)
    pwdhash = binascii.hexlify(pwdhash)
    return (salt + pwdhash).decode('ascii')

# -------------------------- INSERT METHODS ------------------------------

def insert_city(CTY_CODE, CTY_NAME):
    if not isinstance(CTY_CODE, str) or not isinstance(CTY_NAME, str):
        print("Invalid data type")
        raise Exception("Invalid data type")

    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, autocommit=True)
        try:
            with connection.cursor() as cursor:
                sql = "INSERT INTO CITY (CTY_CODE, CTY_NAME) VALUES (%s, %s)"
                cursor.execute(sql, (CTY_CODE, CTY_NAME))
            connection.commit()
        except Exception as e:
            print(f"Error: {e}")
        finally:
            connection.close()
```

```
def insert_user(USR_ID, USR_FIRSTNAME, USR_LASTNAME, CTY_CODE, USR_EMAIL, USR_PHONE,
USR_PASSWORD):
    if not all(isinstance(arg, str) for arg in [USR_ID, USR_FIRSTNAME, USR_LASTNAME, CTY_CODE, USR_EMAIL,
USR_PHONE, USR_PASSWORD]):
        print("Invalid data type")
        raise Exception("Invalid data type")
    hashed_password = hash_password(USR_PASSWORD)
    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, autocommit=True)
        try:
            with connection.cursor() as cursor:
                sql = "INSERT INTO USER (USR_ID, USR_FIRSTNAME, USR_LASTNAME, CTY_CODE, USR_EMAIL,
USR_PHONE, USR_PASSWORD) VALUES (%s, %s, %s, %s, %s, %s, %s)"
                cursor.execute(sql, (USR_ID, USR_FIRSTNAME, USR_LASTNAME, CTY_CODE, USR_EMAIL,
USR_PHONE, hashed_password))
            connection.commit()
        except Exception as e:
            print(f"Error: {e}")
        finally:
            connection.close()

def insert_field(FLD_CODE, FLD_GRASS, FLD_CAPACITY, FLD_ADDRESS, CTY_CODE):
    if not all(isinstance(arg, (str, int)) for arg in [FLD_CODE, FLD_CAPACITY, CTY_CODE]):
        print("Invalid data type")
        raise Exception("Invalid data type")
    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, autocommit=True)
        try:
            binary_image = convert_to_binary(FLD_GRASS)
            with connection.cursor() as cursor:
                sql = "INSERT INTO FIELD (FLD_CODE, FLD_GRASS, FLD_CAPACITY, FLD_ADDRESS, CTY_CODE)
VALUES (%s, %s, %s, %s, %s)"
```

```
                cursor.execute(sql, (FLD_CODE, binary_image, FLD_CAPACITY, FLD_ADDRESS, CTY_CODE))
            connection.commit()
        except Exception as e:
            print(f"Error: {e}")
        finally:
            connection.close()

def insert_training(TRN_CODE, TRN_TYPE):
    if not all(isinstance(arg, str) for arg in [TRN_CODE, TRN_TYPE]):
        print("Invalid data type")
        raise Exception("Invalid data type")
    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, autocommit=True)
        try:
            with connection.cursor() as cursor:
                sql = "INSERT INTO TRAINING (TRN_CODE, TRN_TYPE) VALUES (%s, %s)"
                cursor.execute(sql, (TRN_CODE, TRN_TYPE))
            connection.commit()
        except Exception as e:
            print(f"Error: {e}")
        finally:
            connection.close()

def insert_reservation(RSV_CODE, RSV_DATETIME, USR_ID, FLD_CODE, TRN_CODE):
    if not all(isinstance(arg, str) for arg in [RSV_CODE, USR_ID, FLD_CODE, TRN_CODE]):
        print("Invalid data type")
        raise Exception("Invalid data type")
    if not isinstance(RSV_DATETIME, str):  # Assuming RSV_DATETIME is passed as a string
        print("Invalid data type")
        raise Exception("Invalid data type")
    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, autocommit=True)
        try:
            with connection.cursor() as cursor:
                sql = "INSERT INTO RESERVATION (RSV_CODE, RSV_DATETIME, USR_ID, FLD_CODE, TRN_CODE)
VALUES (%s, %s, %s, %s, %s)"
                cursor.execute(sql, (RSV_CODE, RSV_DATETIME, USR_ID, FLD_CODE, TRN_CODE))
            connection.commit()
        except Exception as e:
            print(f"Error: {e}")
        finally:
            connection.close()

def insert_instructor(INS_ID, INS_NAME, INS_LASTNAME, FLD_CODE):
    if not all(isinstance(arg, str) for arg in [INS_ID, INS_NAME, INS_LASTNAME, FLD_CODE]):
        print("Invalid data type")
        raise Exception("Invalid data type")
    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, autocommit=True)
        try:
            with connection.cursor() as cursor:
                sql = "INSERT INTO INSTRUCTOR (INS_ID, INS_NAME, INS_LASTNAME, FLD_CODE) VALUES (%s,
%s, %s, %s)"
                cursor.execute(sql, (INS_ID, INS_NAME, INS_LASTNAME, FLD_CODE))
            connection.commit()
        except Exception as e:
            print(f"Error: {e}")
        finally:
            connection.close()

def insert_assignment(ASG_CODE, TRN_CODE, INS_ID, ASG_DATETIME):
    if not all(isinstance(arg, str) for arg in [ASG_CODE, TRN_CODE, INS_ID, ASG_DATETIME]):
        print("Invalid data type")
        raise Exception("Invalid data type")
    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, autocommit=True)
```

```
        try:
            with connection.cursor() as cursor:
                sql = "INSERT INTO ASSIGNMENT (ASG_CODE, TRN_CODE, INS_ID, ASG_DATETIME) VALUES (%s, %s, %s, %s)"
                cursor.execute(sql, (ASG_CODE, TRN_CODE, INS_ID, ASG_DATETIME))
            connection.commit()
        except Exception as e:
            print(f"Error: {e}")
        finally:
            connection.close()
```

d. Ff

```
# -------------------------- DELETE METHODS ------------------------------

def remove_city(cty_code):
    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, autocommit=True)
        try:
            with connection.cursor() as cursor:
                deleteQuery = "DELETE FROM CITY WHERE CTY_CODE = %s"
                cursor.execute(deleteQuery, (cty_code,))
        finally:
            connection.close()

def remove_user(usr_id):
    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, autocommit=True)
        try:
            with connection.cursor() as cursor:
                deleteQuery = "DELETE FROM USER WHERE USR_ID = %s"
                cursor.execute(deleteQuery, (usr_id,))
        finally:
            connection.close()
```

```
def remove_field(fld_code):
    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, autocommit=True)
        try:
            with connection.cursor() as cursor:
                deleteQuery = "DELETE FROM FIELD WHERE FLD_CODE = %s"
                cursor.execute(deleteQuery, (fld_code,))
        finally:
            connection.close()

def remove_training(trn_code):
    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, autocommit=True)
        try:
            with connection.cursor() as cursor:
                deleteQuery = "DELETE FROM TRAINING WHERE TRN_CODE = %s"
                cursor.execute(deleteQuery, (trn_code,))
        finally:
            connection.close()

def remove_reservation(rsv_code):
    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, autocommit=True)
        try:
            with connection.cursor() as cursor:
                deleteQuery = "DELETE FROM RESERVATION WHERE RSV_CODE = %s"
                cursor.execute(deleteQuery, (rsv_code,))
        finally:
            connection.close()
```

```
def remove_instructor(ins_id):
    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, autocommit=True)
        try:
            with connection.cursor() as cursor:
                deleteQuery = "DELETE FROM INSTRUCTOR WHERE INS_ID = %s"
                cursor.execute(deleteQuery, (ins_id,))
        finally:
            connection.close()


def remove_assignment(asg_code):
    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, autocommit=True)
        try:
            with connection.cursor() as cursor:
                deleteQuery = "DELETE FROM ASSIGNMENT WHERE ASG_CODE = %s"
                cursor.execute(deleteQuery, (asg_code,))
        finally:
            connection.close()
```

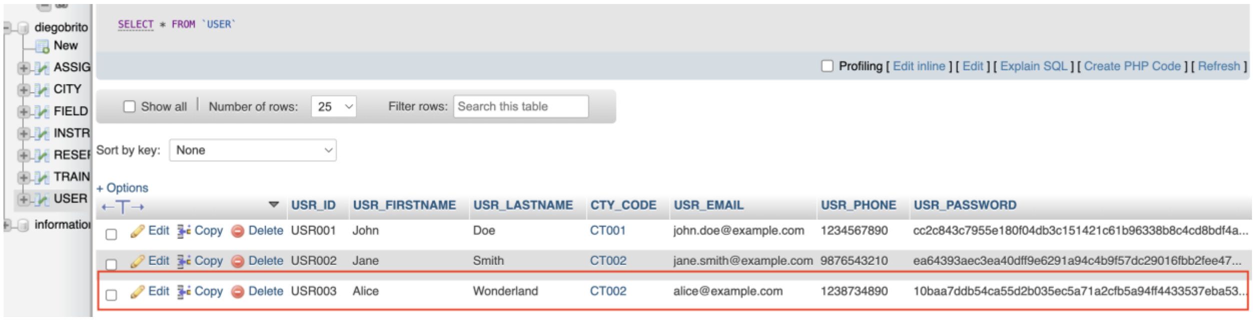
e. The methods I choose that are useful for my application:
    i. An update method for contact information fo the user. Users may need to update their information, such as changing their email or phone number.
    ii. Get available fields method. It is crucial to know which fields are available in a city given a date and time

```
# -------------------------- ADDITIONAL METHODS ------------------------------


def update_user(usr_id, first_name=None, last_name=None, cty_code=None, email=None, phone=None,
password=None):
    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, autocommit=True)
        try:
            with connection.cursor() as cursor:
                update_fields = []
                update_values = []

                if first_name:
                    update_fields.append("USR_FIRSTNAME = %s")
                    update_values.append(first_name)
                if last_name:
                    update_fields.append("USR_LASTNAME = %s")
                    update_values.append(last_name)
                if cty_code:
                    update_fields.append("CTY_CODE = %s")
                    update_values.append(cty_code)
                if email:
                    update_fields.append("USR_EMAIL = %s")
                    update_values.append(email)
                if phone:
                    update_fields.append("USR_PHONE = %s")
                    update_values.append(phone)
                if password:
                    hashed_password = hashlib.sha256(password.encode()).hexdigest()
                    update_fields.append("USR_PASSWORD = %s")
                    update_values.append(hashed_password)

                update_values.append(usr_id)

                updateQuery = f"UPDATE USER SET {', '.join(update_fields)} WHERE USR_ID = %s"
                cursor.execute(updateQuery, update_values)
        finally:
            connection.close()

def get_available_fields(city_code, datetime):
    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, autocommit=True)
    try:
        with connection.cursor() as cursor:
            query = """
            SELECT FIELD.FLD_CODE, FIELD.FLD_GRASS, FIELD.FLD_CAPACITY, FIELD.FLD_ADDRESS
            FROM FIELD
            LEFT JOIN RESERVATION ON FIELD.FLD_CODE = RESERVATION.FLD_CODE AND
RESERVATION.RSV_DATETIME = %s
            WHERE FIELD.CTY_CODE = %s AND RESERVATION.RSV_CODE IS NULL
            """
            cursor.execute(query, (datetime, city_code))
            result = cursor.fetchall()
            return result
    finally:
        connection.close()
```

7. I will run tests for the following scenarios:

   a. Insert User Test
      i. Description: Testing the insertion of a new user with valid data.
      ii. Input: ('USR003', 'Alice', 'Wonderland', 'CT002', 'alice@example.com', '1238734890', 'password123')
      iii. Expected Output: Successful insertion of the user.
      iv. Actual Output:



```
SELECT * FROM `USER`
```

diegobrito
New
ASSIG
CITY
FIELD
INSTR
RESEF
TRAIN
USER
informatio
□ Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP Code] [Refresh]
□ Show all
+ Options
←T→

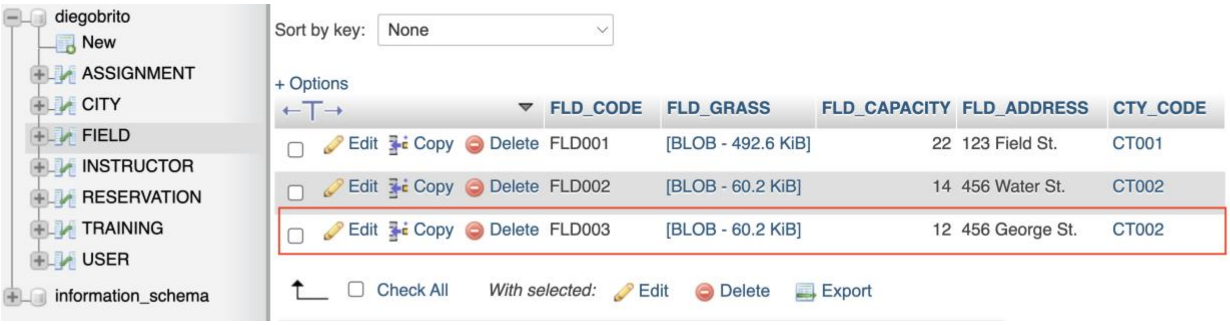
Number of rows: 25 | Filter rows: Search this table

Sort by key: None

	USR_ID	USR_FIRSTNAME	USR_LASTNAME	CTY_CODE	USR_EMAIL	USR_PHONE	USR_PASSWORD
□ Edit Copy Delete	USR001	John	Doe	CT001	john.doe@example.com	1234567890	cc2c843c7955e180f04db3c151421c61b96338b8c4cd8bdf4a...
□ Edit Copy Delete	USR002	Jane	Smith	CT002	jane.smith@example.com	9876543210	ea64393aec3ea40dff9e6291a94c4b9f57dc29016fbb2fee47...
□ Edit Copy Delete	USR003	Alice	Wonderland	CT002	alice@example.com	1238734890	10baa7ddb54ca55d2b035ec5a71a2cfb5a94ff4433537eba53...

   b. Insert Field Test
      i. Description: Testing the insertion of a new field with valid data including a path for an image.
      ii. Input: ('FLD003', 'artificialTurf.jpeg', 12, '456 George St.', 'CT002')
      iii. Expected Output: Successful insertion of the field.

iv. Actual Output:



Sort by key: None

	FLD_CODE	FLD_GRASS	FLD_CAPACITY	FLD_ADDRESS	CTY_CODE
□ Edit Copy Delete	FLD001	[BLOB - 492.6 KiB]	22	123 Field St.	CT001
□ Edit Copy Delete	FLD002	[BLOB - 60.2 KiB]	14	456 Water St.	CT002
□ Edit Copy Delete	FLD003	[BLOB - 60.2 KiB]	12	456 George St.	CT002

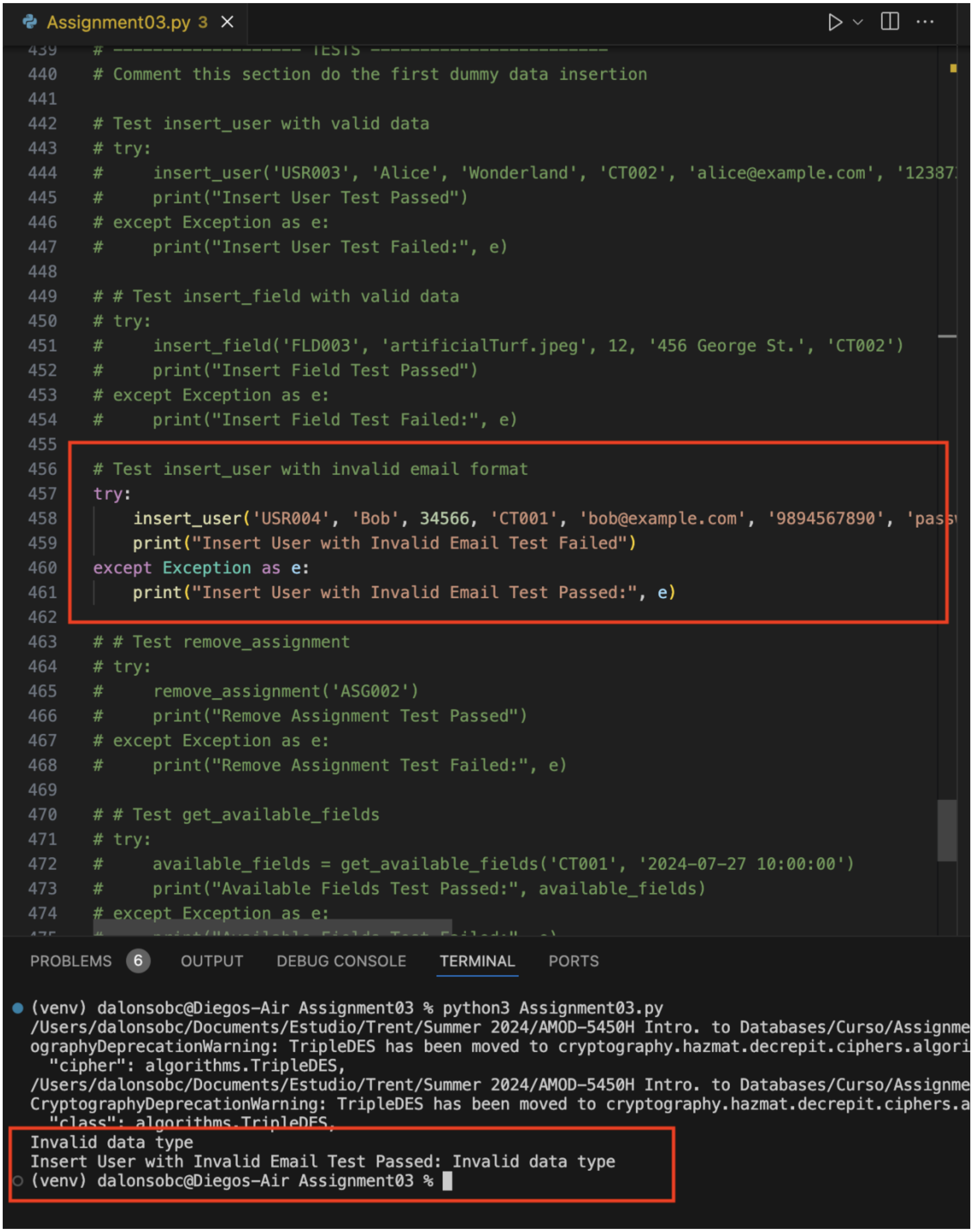
c. Insert User with Invalid Email Test

i. Description: Testing the insertion of a new user with an invalid email format.

ii. Input: ('USR004', 'Bob', 34566, 'CT001', 'bob@example.com', '9894567890', 'password123')

iii. Expected Output: Graceful handling of the error.

iv. Actual Output:



```
# -------------------- TESTS -------------------------
# Comment this section do the first dummy data insertion

# Test insert_user with valid data
# try:
#     insert_user('USR003', 'Alice', 'Wonderland', 'CT002', 'alice@example.com', '12387
#     print("Insert User Test Passed")
# except Exception as e:
#     print("Insert User Test Failed:", e)

# # Test insert_field with valid data
# try:
#     insert_field('FLD003', 'artificialTurf.jpeg', 12, '456 George St.', 'CT002')
#     print("Insert Field Test Passed")
# except Exception as e:
#     print("Insert Field Test Failed:", e)

# Test insert_user with invalid email format
try:
    insert_user('USR004', 'Bob', 34566, 'CT001', 'bob@example.com', '9894567890', 'pass
    print("Insert User with Invalid Email Test Failed")
except Exception as e:
    print("Insert User with Invalid Email Test Passed:", e)

# # Test remove_assignment
# try:
#     remove_assignment('ASG002')
#     print("Remove Assignment Test Passed")
# except Exception as e:
#     print("Remove Assignment Test Failed:", e)

# # Test get_available_fields
# try:
#     available_fields = get_available_fields('CT001', '2024-07-27 10:00:00')
#     print("Available Fields Test Passed:", available_fields)
# except Exception as e:
#     print("Available Fields Test Failed:", e)
```

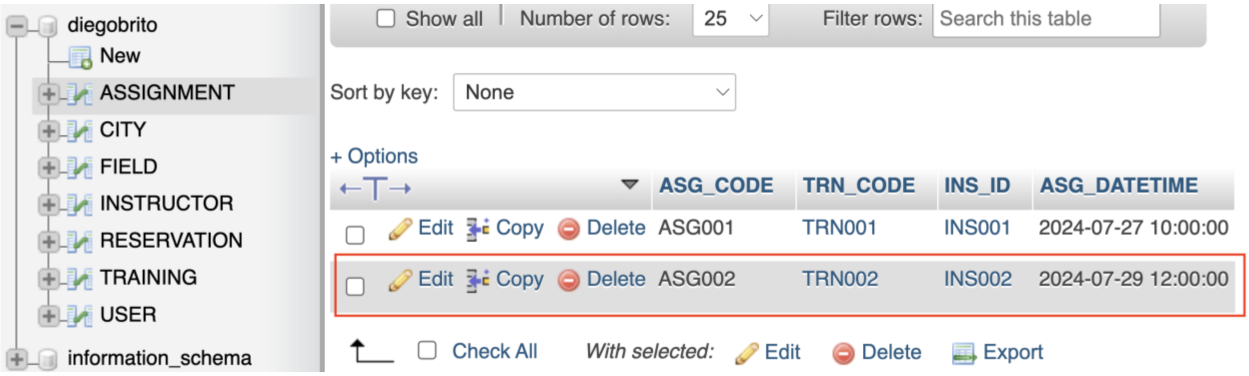
```
(venv) dalonsobc@Diegos-Air Assignment03 % python3 Assignment03.py
/Users/dalonsobc/Documents/Estudio/Trent/Summer 2024/AMOD-5450H Intro. to Databases/Curso/Assignme
ographyDeprecationWarning: TripleDES has been moved to cryptography.hazmat.decrepit.ciphers.algori
  "cipher": algorithms.TripleDES,
/Users/dalonsobc/Documents/Estudio/Trent/Summer 2024/AMOD-5450H Intro. to Databases/Curso/Assignme
CryptographyDeprecationWarning: TripleDES has been moved to cryptography.hazmat.decrepit.ciphers.a
  "class": algorithms.TripleDES,
Invalid data type
Insert User with Invalid Email Test Passed: Invalid data type
(venv) dalonsobc@Diegos-Air Assignment03 %
```

d. Remove Assignment Test
   i. Description: Testing the removal of an assignment by its primary key.

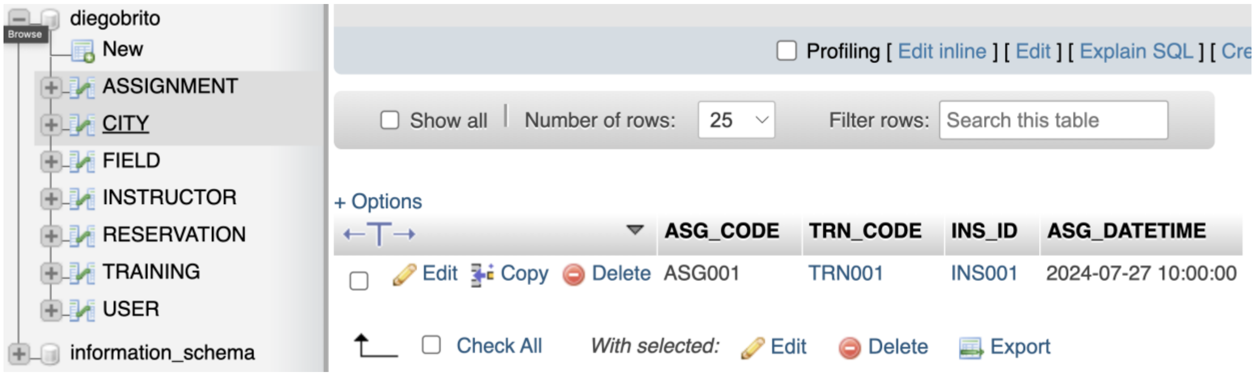
ii. Input: ‘ASG002’

iii. Expected Output: Successful removal of the assignment.

iv. Actual Output:



	ASG_CODE	TRN_CODE	INS_ID	ASG_DATETIME
□ Edit Copy Delete	ASG001	TRN001	INS001	2024-07-27 10:00:00
□ Edit Copy Delete	ASG002	TRN002	INS002	2024-07-29 12:00:00



	ASG_CODE	TRN_CODE	INS_ID	ASG_DATETIME
□ Edit Copy Delete	ASG001	TRN001	INS001	2024-07-27 10:00:00

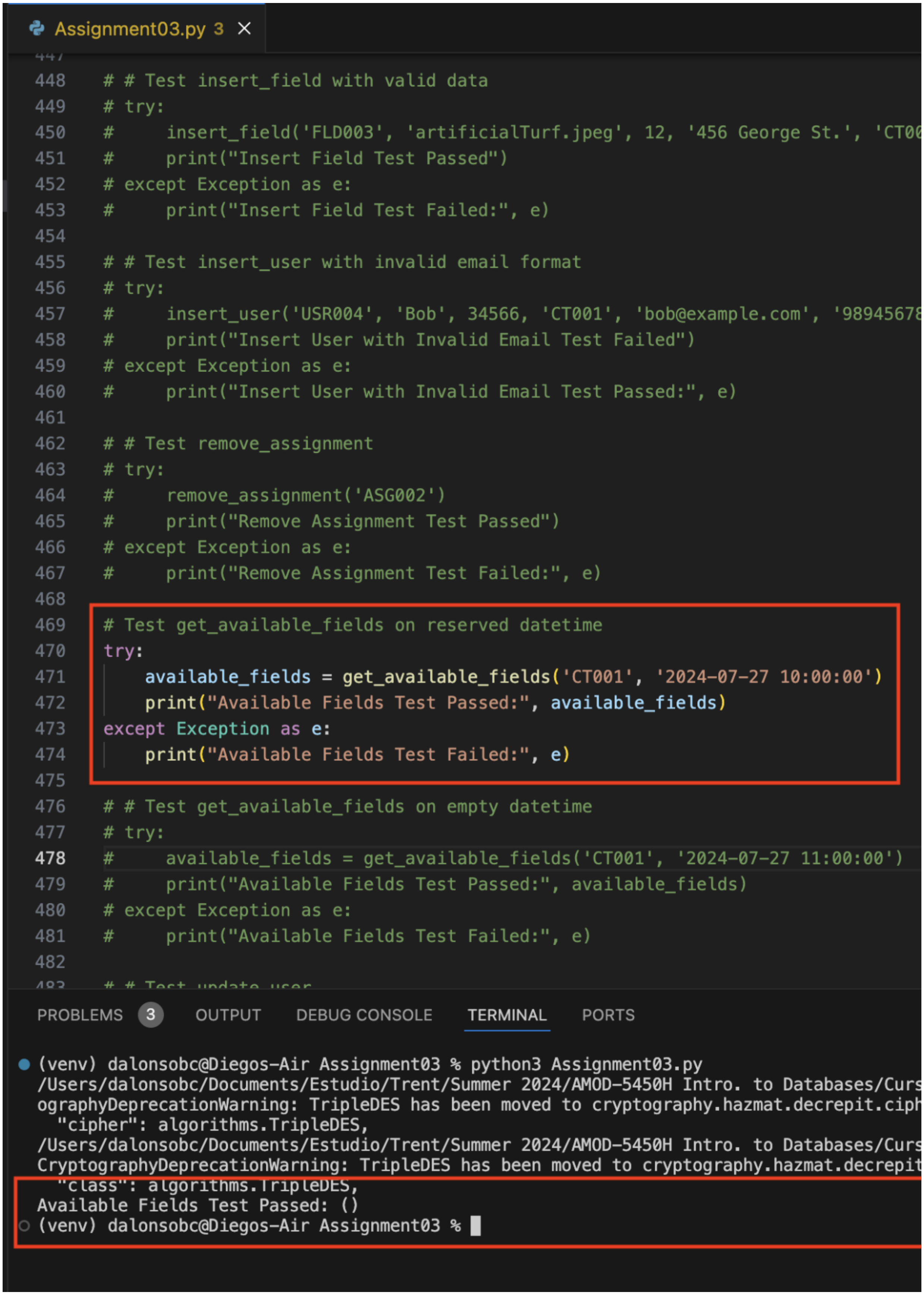
e. Get Available Fields Test on Reserved Datetime

i. Description: Testing retrieval of available fields for a given city and reserved datetime.

ii. Input: ('CT001', '2024-07-27 10:00:00')

iii. Expected Output: No available fields.

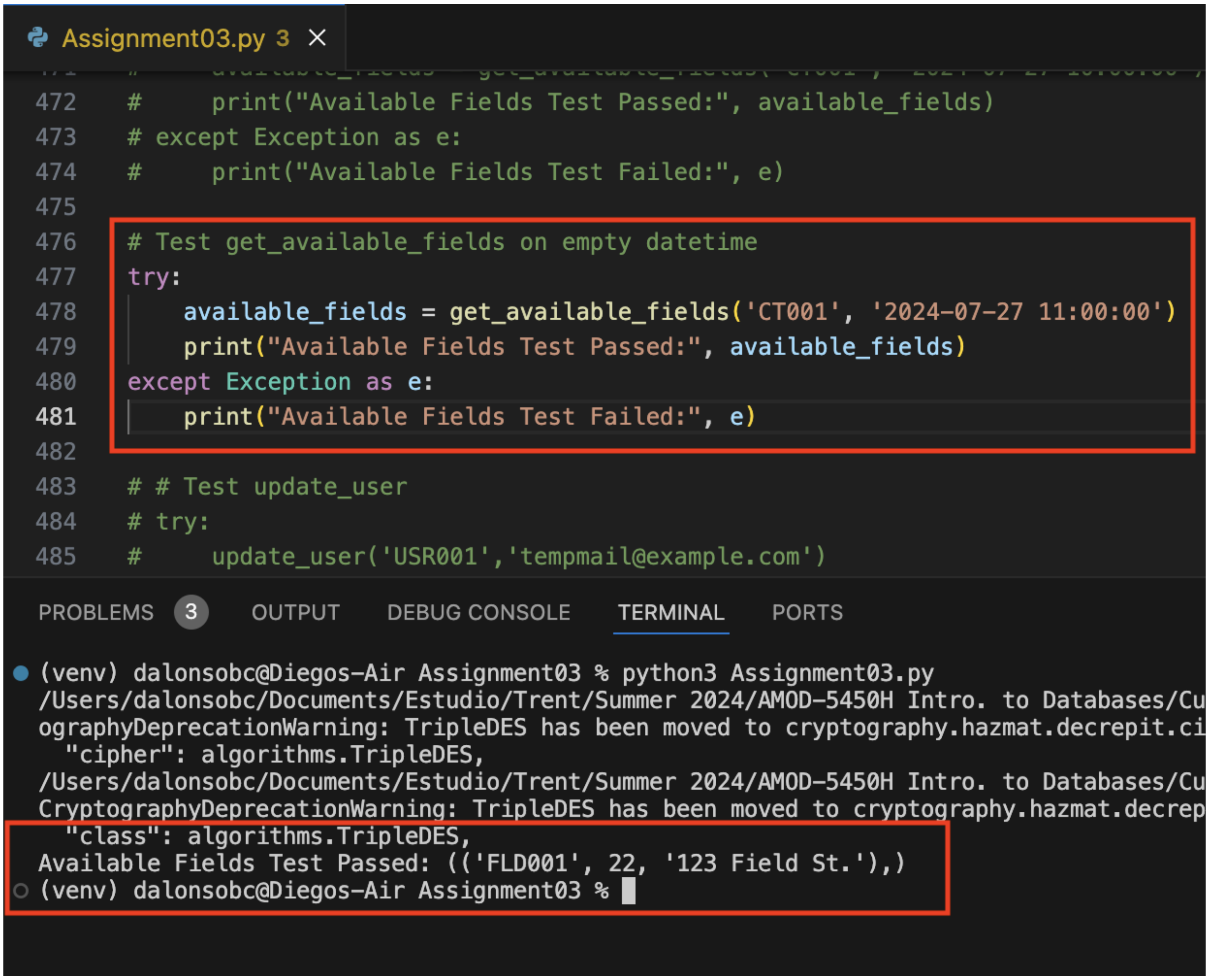
iv. Actual Output:



```python
# # Test insert_field with valid data
# try:
#     insert_field('FLD003', 'artificialTurf.jpeg', 12, '456 George St.', 'CT00
#     print("Insert Field Test Passed")
# except Exception as e:
#     print("Insert Field Test Failed:", e)

# # Test insert_user with invalid email format
# try:
#     insert_user('USR004', 'Bob', 34566, 'CT001', 'bob@example.com', '98945678
#     print("Insert User with Invalid Email Test Failed")
# except Exception as e:
#     print("Insert User with Invalid Email Test Passed:", e)

# # Test remove_assignment
# try:
#     remove_assignment('ASG002')
#     print("Remove Assignment Test Passed")
# except Exception as e:
#     print("Remove Assignment Test Failed:", e)

# Test get_available_fields on reserved datetime
try:
    available_fields = get_available_fields('CT001', '2024-07-27 10:00:00')
    print("Available Fields Test Passed:", available_fields)
except Exception as e:
    print("Available Fields Test Failed:", e)

# # Test get_available_fields on empty datetime
# try:
#     available_fields = get_available_fields('CT001', '2024-07-27 11:00:00')
#     print("Available Fields Test Passed:", available_fields)
# except Exception as e:
#     print("Available Fields Test Failed:", e)

# # Test update_user
```

```
(venv) dalonsobc@Diegos-Air Assignment03 % python3 Assignment03.py
/Users/dalonsobc/Documents/Estudio/Trent/Summer 2024/AMOD-5450H Intro. to Databases/Curs
ographyDeprecationWarning: TripleDES has been moved to cryptography.hazmat.decrepit.ciph
  "cipher": algorithms.TripleDES,
/Users/dalonsobc/Documents/Estudio/Trent/Summer 2024/AMOD-5450H Intro. to Databases/Curs
CryptographyDeprecationWarning: TripleDES has been moved to cryptography.hazmat.decrepit
  "class": algorithms.TripleDES,
Available Fields Test Passed: ()
(venv) dalonsobc@Diegos-Air Assignment03 %
```

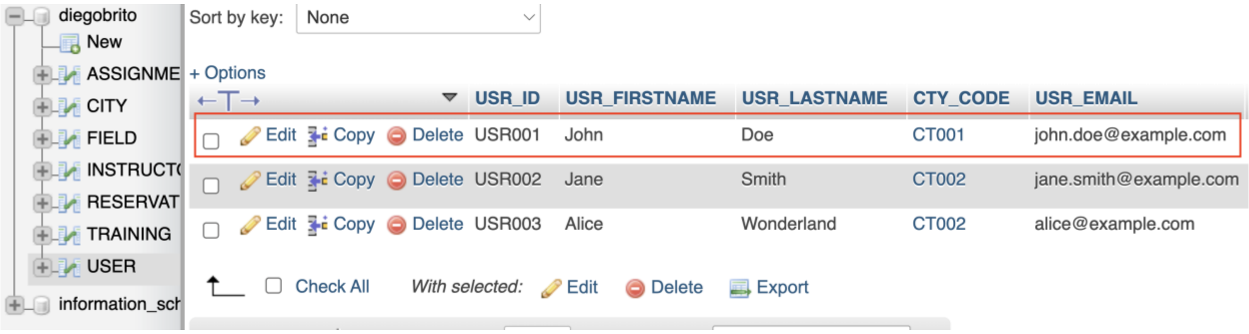
f. Get Available Fields Test on Free Datetime
    i. Description: Testing retrieval of available fields for a given city and free datetime.
    ii. Input: ('CT001', '2024-07-27 11:00:00')
    iii. Expected Output: List of available fields.
    iv. Actual Output:



```python
#     print("Available Fields Test Passed:", available_fields)
# except Exception as e:
#     print("Available Fields Test Failed:", e)

# Test get_available_fields on empty datetime
try:
    available_fields = get_available_fields('CT001', '2024-07-27 11:00:00')
    print("Available Fields Test Passed:", available_fields)
except Exception as e:
    print("Available Fields Test Failed:", e)

# # Test update_user
# try:
#     update_user('USR001','tempmail@example.com')
```

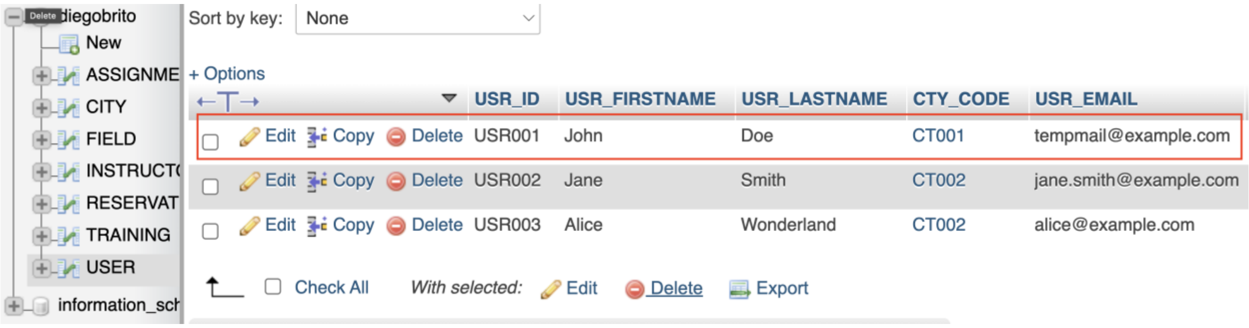
```
(venv) dalonsobc@Diegos-Air Assignment03 % python3 Assignment03.py
/Users/dalonsobc/Documents/Estudio/Trent/Summer 2024/AMOD-5450H Intro. to Databases/Cu
ographyDeprecationWarning: TripleDES has been moved to cryptography.hazmat.decrepit.ci
  "cipher": algorithms.TripleDES,
/Users/dalonsobc/Documents/Estudio/Trent/Summer 2024/AMOD-5450H Intro. to Databases/Cu
CryptographyDeprecationWarning: TripleDES has been moved to cryptography.hazmat.decrep
  "class": algorithms.TripleDES,
Available Fields Test Passed: (('FLD001', 22, '123 Field St.'),)
(venv) dalonsobc@Diegos-Air Assignment03 %
```

g. Update the email of a User
    i. Description: Testing update of user email.
    ii. Input: (‘USR001’, email=’tempmail@example.com’)
    iii. Expected Output: Successful user update.
    iv. Actual Output:



Sort by key: None

	USR_ID	USR_FIRSTNAME	USR_LASTNAME	CTY_CODE	USR_EMAIL
□ Edit Copy Delete	USR001	John	Doe	CT001	john.doe@example.com
□ Edit Copy Delete	USR002	Jane	Smith	CT002	jane.smith@example.com
□ Edit Copy Delete	USR003	Alice	Wonderland	CT002	alice@example.com



Sort by key: None

	USR_ID	USR_FIRSTNAME	USR_LASTNAME	CTY_CODE	USR_EMAIL
□ Edit Copy Delete	USR001	John	Doe	CT001	tempmail@example.com
□ Edit Copy Delete	USR002	Jane	Smith	CT002	jane.smith@example.com
□ Edit Copy Delete	USR003	Alice	Wonderland	CT002	alice@example.com