A close up of a logo

Description automatically generated

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.

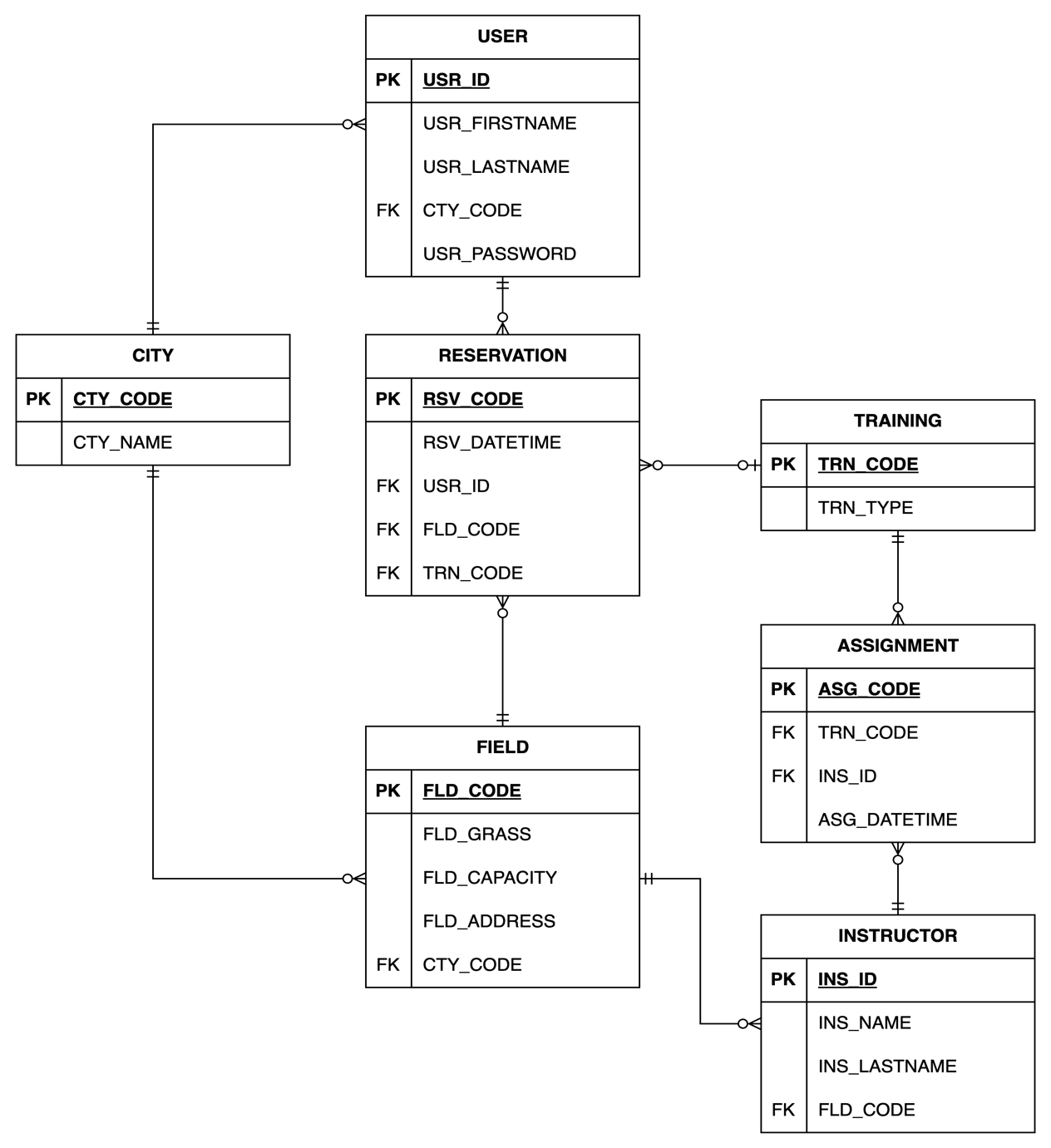
They also offer training services. You can choose to work on attack, defense, tactics, fitness, and more. Each field has instructors who can do any type of training.

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.



1. 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:

A black background with white text

Description automatically generated

1. My ERD design appears to be in 3NF. There are no apparent partial or transitive dependencies.
2. 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.
3. As
   1. 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

)

* 1. 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()

* 1. 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()

* 1. 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()

* 1. The methods I choose that are useful for my application:
     1. An update method for contact information fo the user. Users may need to update their information, such as changing their email or phone number.
     2. 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()

1. I will run tests for the following scenarios:
   1. Insert User Test
      1. Description: Testing the insertion of a new user with valid data.
      2. Input: ('USR003', 'Alice', 'Wonderland', 'CT002', 'alice@example.com', '1238734890', 'password123')
      3. Expected Output: Successful insertion of the user.
      4. Actual Output:

A screenshot of a computer

Description automatically generated

* 1. Insert Field Test
     1. Description: Testing the insertion of a new field with valid data including a path for an image.
     2. Input: ('FLD003', 'artificialTurf.jpeg', 12, '456 George St.', 'CT002')
     3. Expected Output: Successful insertion of the field.
     4. Actual Output:

A screenshot of a computer

Description automatically generated

* 1. Insert User with Invalid Email Test
     1. Description: Testing the insertion of a new user with an invalid email format.
     2. Input: ('USR004', 'Bob', 34566, 'CT001', 'bob@example.com', '9894567890', 'password123')
     3. Expected Output: Graceful handling of the error.
     4. Actual Output:

A screenshot of a computer program

Description automatically generated

* 1. Remove Assignment Test
     1. Description: Testing the removal of an assignment by its primary key.
     2. Input: ‘ASG002’
     3. Expected Output: Successful removal of the assignment.
     4. Actual Output:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

* 1. Get Available Fields Test on Reserved Datetime
     1. Description: Testing retrieval of available fields for a given city and reserved datetime.
     2. Input: ('CT001', '2024-07-27 10:00:00')
     3. Expected Output: No available fields.
     4. Actual Output:

A screenshot of a computer program

Description automatically generated

* 1. Get Available Fields Test on Free Datetime
     1. Description: Testing retrieval of available fields for a given city and free datetime.
     2. Input: ('CT001', '2024-07-27 11:00:00')
     3. Expected Output: List of available fields.
     4. Actual Output:

A screenshot of a computer program

Description automatically generated

* 1. Update the email of a User
     1. Description: Testing update of user email.
     2. Input: (‘USR001’, email=’tempmail@example.com’)
     3. Expected Output: Successful user update.
     4. Actual Output:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated