Lab Exercises

Exercise 1. SQLALCHEMY

Overview	
Running the prepared script to create our database schema and test data	
Objective	• Optionally Create the database structure for our application • Create a Knight class that uses SQLAlchemy to save and retrieve instances • Create convenience functions to drop and create the table • Create a form function that accepts user input to add Knights to the database
Builds on Previous Labs	Setup
Time to Complete	30-45 minutes

This exercise is designed to support MySQL/MariaDB or SQLite. If you're using MySQL/MariaDB, you may need to create and populate the database for classroom use.

If necessary, make sure you've installed SQLAlchemy:

```
pip3 install sqlalchemy
```

Or if you're using Anaconda:

```
conda install sqlalchemy
```

1. Create a new script called "knight_alchemy.py". First import the key modules:

```
import sys
from sqlalchemy import create_engine, Column, Integer, String
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker
```

2. Set up a db_url variable. Notice we have the uncommented version for MySQL/MariaDB. The commented version is for SQLite. Flip the commented version if you're using SQLite:

```
mysql_user = "student"
mysql_password = "Tr!v3raT3ch"
db_url = "mysql+mysqlconnector://%s:%s@localhost/pydemo" % (mysql_user,
mysql_password)
# db_url = "sqlite:///:memory: # For SQLite
# print(db_url)
```

3. Let's create the session using the db_url, then a Session factory bound to that engine. From there we'll create a session instance (which we might do in an individual user request/response thread in a web app). Our "Base" class will use SQLAlchemy's "declarative_base":


```
engine = create_engine(db_url, echo=False) # set echo to True for verbose
logging
Session = sessionmaker(bind=engine)
session = Session()
Base = declarative_base()
```

4. Create a Knight class that extends the base. We'll need the __tablename__ field to use the name of the table in the DB. The 4 fields are defined using the Column constructor. The __init__ method and __repr__ methods are conveniences for us later and not strictly required by SQLAlchemy.

```
class Knight(Base):
    __tablename__ = 'knights'

    id = Column(Integer, primary_key=True)
    name = Column(String(50))
    quest = Column(String(255))
    color = Column(String(50))
    comment = Column(String(255))

    def __init__(self, name = "", quest = "", color = "", comment = ""):
        self.name = name
        self.quest = quest
        self.color = color
        self.comment = comment

    def __repr__(self):
        return "<Knight(id=%s, name=%s, quest=%s, color=%s, comment=%s)>"
% (self.id, self.name, self.quest, self.color, self.comment)
```

5. Let's create a convenience function that prompts for a Knight's field values and returns a constructed Knight instance. Note our use of default values (input "or" default):

```
def KnightForm(_name = "", _quest = "", _color = "", _comment = ""):
    print("Stop! Who would cross the Bridge of Death must answer me these questions
three, ere the other side he see.")
    name = input("What... is your name? [%s]" % _name) or _name
    quest = input("What... is your quest? [%s]" % _quest) or _quest
    if "Robin" in name:
        color = input("What... is the capital of Assyria? [%s]" % _color) or _color
        comment = input("Comment: [%s] " % _comment) or _comment
    elif "Arthur" in name:
        color = input("What... is the air-speed velocity of an unladen swallow? [%s]"
% _color) or _color
        print("Huh? I-- I don't know that. Auuuuuuuugh!")
        comment = input("How do know so much about swallows? [%s]" % _comment) or
_comment
    else:
        color = input("What... is your favorite color? [%s]" % _color) or _color
        print("Off you go!")
        comment = input("Comment: [%s] " % _comment) or _comment
    return Knight(name, quest, color, comment)
```

6. This function will "describe" the Knight table with the SQL schema:

```
def print_knight_table():
    Knight.__table__
```


7. This is a convenience function that won't add much value for the moment. Internally it will just use the __repr__ function, so this is a simple wrapper function. We might add other qualities later that aren't available in the normal print() function:

```
def print_knight(knight):
    print(knight)
```

8. Create a function called "create_table" that uses the Base.metadata.create_all call - this will infer the schema from the declared columns in the class:

```
def create_table():
    Base.metadata.create_all(engine)
```

9. Let's create a convenience function to drop the table so we can start fresh. This probably won't be used in a production system.

```
def drop_table():
    Knight.__table__.drop(engine)
    session.flush()
    session.commit()
```

10. One of our key functions will be to list all of the Knights in the database - effectively a "SELECT * FROM knights ORDER BY id":

```
def list_all():
    for knight in session.query(Knight).order_by(Knight.id):
        print_knight(knight)
```

11. Another key function is passed a Knight instance and saves it to the DB. No SQL for us to write - SQLAlchemy does it for us:

```
def add_knight(knight):
    session.add(knight)
    session.flush()
    session.commit()
```

12. When our program ends we'll want to make sure that we at least close the SQLAlchemy session. Let's create a function to do that:

```
def quit():
    session.close()
    sys.exit()
```


13. Our main will use a menu that lists options to use our functions. We'll use a function to do that.

```
def print_menu():
    menu_items=[
            ("Add Knight"),
            ("Add Sir Lancelot"),
            ("Add Sir Robin"),
            ("Add King Arthur"),
            ("List Knights")
    ]
    for idx, menu_item in enumerate(menu_items):
        print("%s) %s" % (str(idx+1), menu_item))
    print("q) Quit")
    # Prompt for a menu item number, default to '5' for 'List Knights' if we get
nothing
    mi_num = input("Menu Item # ") or '5'
    # print(mi_num)

    # Instead of a switch-case, we create an anonymous dictionary with lambda
functions to invoke and execute based on the input value we just retrieved
    {
            '1': lambda: add_knight(KnightForm()),
            '2': lambda: add_knight(KnightForm(_name = "Sir Lancelot of Camelot",
_quest = "To Seek the Holy Grail", _color = "Blue", _comment = "Oh, thank you. Thank
you very much.")),
            '3': lambda: add_knight(KnightForm(_name = "Sir Robin of Camelot", _quest
= "To Seek the Holy Grail", _color = "I don't know that!", _comment =
"Auuuuuuuugh!")),
            '4': lambda: add_knight(KnightForm(_name = "Arthur, King of the Britons",
_quest = "To Seek the Holy Grail", _color = "What do you mean? An African or European
swallow?", _comment = "Well, you have to know these things when you're a king, you
know.")),
            '5': lambda: list_all(),
            'q': lambda: quit()
    }[mi_num]()
```

14. Lastly we'll create our main. We'll want command-line options to destroy and create a table, but the main's primary purpose is to run "print_menu" in an infinite loop:

```
# Invoke to interact with existing data and be able to add more:
#   python3 knight_alchemy.py
# Invoke with "drop" to destroy the table (and the data in it):
#   python3 knight_alchemy.py drop
# Invoke with "init" to create the table:
#   python3 knight_alchemy.py init
# Start Fresh; drop and re-create the table (parameter ordering doesn't
matter - drop will occur before init):
#   python3 knight_alchemy.py drop init
if __name__ == '__main__':
    if 'drop' in sys.argv:
        drop_table()
    if 'init' in sys.argv:
        create_table()
    while True:
        print_menu()
```


15. **Overtime**: Add a function and menu item to find a Knight by name, possibly using the "LIKE" or "ILIKE" functions
16. **Overtime**: Add menu items to destroy and create the database
17. **Overtime**: Add a menu item to export the current database to a CSV file
18. **Overtime**: Add a menu item to import a CSV file into the database

Congratulations! You've just begun to exploit the power of an ORM tool like SQLAlchemy. From here you'll want to tackle One-to-One relationships, One-to-Many relationships, Many-to-Many relationships, and cascade deletes.

