# Database Programming

## Agenda

1. PyMySQL
2. CRUD queries
3. Exercise: “TODO application”
4. SQLAlchemy
5. *pymongo

## PyMySQL

## Installing PyMySQL

- PyMySQL is a library providing easy access to MySQL databases
- Installation: `pip install pymysql`

## Connecting to the database

- Connection is achieved using `connect` function and should be closed before application finishes
- `connect` takes the following parameters:
    - database host (`127.0.0.1` if you are using a local database)
    - user
    - password
    - default schema/database to use (empty if there is no schema yet)
- Queries should be executed using `cursor` objects

## Example 01-pymysql/example-01.py

```python
import pymysql

db = pymysql.connect("127.0.0.1", "root", "password", "")

with db.cursor() as c:
    c.execute("SELECT VERSION()")
    version = c.fetchone()
    print(f"Database version: {version[0]}")
db.close()
```

```
$ python3 01-pymysql/example-01.py
```

```
Database version: 5.7.28
```

## Creating default schema/db

- It is a good practice to create a new schema for every project
- We will use PyMySQL to create a schema and then connect to it
- The SQL statement:
  ```
  CREATE SCHEMA IF NOT EXISTS default DEFAULT CHARACTER SET
  utf8;
  ```

## Example 01-pymysql/example-02.py

```python
import pymysql

db = pymysql.connect("127.0.0.1", "root", "password", "")
with db.cursor() as c:
   c.execute("CREATE SCHEMA IF NOT EXISTS `default` DEFAULT CHARACTER SET utf8;")
db.close()

db = pymysql.connect("127.0.0.1", "root", "password", "default")
db.close()
```

```
$ python3 01-pymysql/example-02.py
```

## CRUD queries

## Inserting values

- You can insert new rows using INSERT statement
- However, you should never format SQL query with values yourself
- Instead of that, SQL queries support printf-like formatting for the values
- Syntax:
  - `INSERT INTO table (col1, col2) VALUES (%s, %s)`
  - `cursor.execute(parametrized_sql, parameter_tuple)`

## Example 01-pymysql/example-03.py

```python
import pymysql

db = pymysql.connect("127.0.0.1", "root", "password", "default")
with db.cursor() as c:
    c.execute("CREATE TABLE IF NOT EXISTS `example3` (a integer, b varchar(255));")
    a = int(input("Please provide integer value for a: "))
    b = input("Please provide value for b: ")[:254]
    c.execute("INSERT INTO `example3` (a, b) VALUES (%s, %s);", (a, b))
    db.commit()
db.close()
```

```
$ python3 01-pymysql/example-03.py
```

```
Please provide integer value for a: 1
Please provide value for b: Sample string
```

## Exercise 1

Using code provided in Example 2, create `bikesharing` table in `default` database with the following columns:

- tstamp **timestamp**
- cnt **integer**
- temperature **decimal(5, 2)**
- temperature_feels **decimal(5, 2)**
- humidity **decimal(4, 1)**
- wind_speed **decimal(5,2)**
- weather_code **integer**
- is_holiday **boolean**
- is_weekend **boolean**
- season **integer**

## Exercise 2

- load values from `london-bikes.csv` and insert it into the database, one by one
- * commit after every 100 inserts, not after every one

## Fetching values

- To fetch values execute a select statement in cursor
- Then use cursor’s `fetchone` to fetch a single row or `fetchall` to fetch all resulting rows
- Cursor also stores row count in a `rowcount` property and
- Column names in `description` property

## Example 01-pymysql/example-04.py

```python
import pymysql
import datetime

db = pymysql.connect("127.0.0.1", "root", "password", "default")
with db.cursor() as c:
    c.execute(
        "SELECT tstamp,cnt FROM bikesharing WHERE tstamp BETWEEN  %s AND %s",
        (datetime.datetime(2016, 1, 1, 0), datetime.datetime(2016, 1, 1, 5)),
    )
    print(f"Column names: {[d[0] for d in c.description]}")
    print(c.fetchone())  # first row
    print(c.fetchall())  # remaining rows
    print(f"Got {c.rowcount} rows")

db.close()
```

```
$ python3 01-pymysql/example-04.py
```

```
Column names: ['tstamp', 'cnt']
(datetime.datetime(2016, 1, 1, 0, 0), 786)
((datetime.datetime(2016, 1, 1, 1, 0), 660), (datetime.datetime(2016, 1, 1, 2, 0), 387), (datetime.datetime(2016, 1, 1, 3, 0), 294),
(datetime.datetime(2016, 1, 1, 4, 0), 219), (datetime.datetime(2016, 1, 1, 5, 0), 153))
Got 6 rows
```

## Exercise 3

- fetch total sum of new shares by season
- fetch total sum of new shares during thunderstorms
- fetch the date and hour with the most new shares

## Updating values - Exercise 4

- Add 10 to cnt column for all 2015-01-09 entries

## Deleting values - Exercise 5

- Delete all entries from 2017-01-03 in `bikesharing` table

## Exercise: “TODO application”

## Exercise 6

The goal: a “To Do” application

- Create a `todo_app` database
- Create a `tasks` table with the following schema
  - id **int not null auto_increment**
  - task **text not null**
  - done **boolean**
  - **primary key** - id
- In a loop:
  - ask user what to do using `input()`
  - show task list
  - mark task as done
  - add new task
  - exit application
- Implement functions which perform the above actions using the database as task storage

- For `show tasks`:
  - print all open tasks and their ids in order of ids
- For `mark as done`
  - ask user which `id` to mark as done
  - update the done field in the table for given `id`
- For `add new task`
  - ask for task name/description
  - insert a new record to the tasks db

# SQLAlchemy

## What is an ORM?

- Until now we have been using pure SQL queries to interact with the database
- ORM, or Object Relational Mapping, is a layer of abstraction on top of relational database and its query language
- Instead of operating directly on databases, queries and rows, we use objects, filters and iterators to represent the underlying data.

## What is SQLAlchemy?

- SQLAlchemy is one of the most popular Python ORMs
- It uses purely Pythonic objects to represent the data:
  - class represents a table
  - instance represents a row
  - instance attribute represents a column
- Session is the interface for communicating with the database in SQLAlchemy
- `pip install sqlalchemy`

## Creating a table

To create a table you have to:

- Connect to the database using `create_engine()`
- Create a declarative base class
- Use it as a superclass for classes defining tables
- Use `Base.metadata.create_all(engine)` to create all our tables

# Example 02-sqlalchemy/example-01.py

```python
from sqlalchemy import create_engine
from models import Base, Student

CONNECTION_STRING = "mysql+pymysql://{user}:{password}@{host}/{db}"

eng = create_engine(
    CONNECTION_STRING.format(
        user="root", password="password", host="127.0.0.1", db="default"
    )
)
Base.metadata.create_all(eng)
```

```python
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import Column, Integer, String, ForeignKey, DateTime


Base = declarative_base()


class Student(Base):
    __tablename__ = "students"
    id = Column(Integer, primary_key=True, autoincrement=True)
    first_name = Column(String(255))
    last_name = Column(String(255))

    def __str__(self):
        return f"<Student #{self.id} {self.first_name} {self.last_name}>"
```

```
$ python3 02-sqlalchemy/example-01.py
```

## Adding data

- As mentioned before, you need a session object
- To do so, use `sessionmaker` function to create Session base class
- Then create a `Session` instance
- Use the class we just created to add some new students with `add_all(list)` or `add(object)`

## Example 02-sqlalchemy/example-02.py

```python
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from models import Base, Student

CONNECTION_STRING = "mysql+pymysql://{user}:{password}@{host}/{db}"

eng = create_engine(
    CONNECTION_STRING.format(
        user="root", password="password", host="127.0.0.1", db="default"
    )
)
Session = sessionmaker(bind=eng)
s = Session()

s.add_all(
    [
        Student(first_name="Mike", last_name="Wazowski"),
        Student(first_name="Netti", last_name="Nashe"),
        Student(first_name="Jessamine", last_name="Addison"),
        Student(first_name="Brena", last_name="Bugdale"),
        Student(first_name="Theobald", last_name="Oram"),
    ]
)
s.commit()
```

```
$ python3 02-sqlalchemy/example-02.py
```

## Querying data

- To query the data, use session object like so: `session.query(<class>).all()`
- You can use existing classes which inherit after the Base class
- You can use a `filter(<class>.property == <value>)` function to create SQL like clauses (similar to `WHERE`)

# Example 02-sqlalchemy/example-03.py

```python
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from models import Base, Student
CONNECTION_STRING = "mysql+pymysql://{user}:{password}@{host}/{db}"
eng = create_engine(
    CONNECTION_STRING.format(
        user="root", password="password", host="127.0.0.1", db="default"
    )
)
Session = sessionmaker(bind=eng)
s = Session()

rows = s.query(Student).all()
for row in rows:
    print(row)

print("---")
total = s.query(Student).count()
print(f"Total: {total}")

print("---")
query_result = s.query(Student).filter(Student.id >= 2,
Student.first_name.like("Bre%"))
print("Found students:")
for row in query_result:
    print(row)
```

```
$ python3 02-sqlalchemy/example-03.py
```

```
<Student #1 Mike Wazowski>
<Student #2 Netti Nashe>
<Student #3 Jessamine Addison>
<Student #4 Brena Bugdale>
<Student #5 Theobald Oram>
---
Total: 5
---
Found students:
<Student #4 Brena Bugdale>
```

## Foreign keys

- to define foreign keys, add `ForeignKey(<class.attribute>)` to the `Column` declaration
- to query with joins on tables using foreign keys, use `session.query(class1).join(class2)`
- to get data from both tables, use `session.query(class1, class2).join(class2)`

# Example 02-sqlalchemy/example-04.py

```python
from sqlalchemy.exc import IntegrityError, InvalidRequestError
(...)
try:
    s.add_all([
            Locker(number=1, student=4),
            Locker(number=2, student=1),
            Locker(number=3, student=5),
            Locker(number=4, student=2),
            Locker(number=5, student=3),
    ])
    s.commit()
except IntegrityError:
    s.rollback()
    print("Lockers already created!")

rows = s.query(Student, Locker).join(Locker).filter(Locker.number == 4)
for row in rows:
    student, locker = row
    print(f"Student with locker #{locker.number}: {student}")
```

```python
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import Column, Integer, String, ForeignKey, DateTime


Base = declarative_base()


class Locker(Base):
    __tablename__ = "lockers"
    number = Column(Integer, primary_key=True)
    student = Column(Integer, ForeignKey(Student.id), primary_key=True)

    def __str__(self):
        return f"<Locker {self.number}: {self.student}>"
```

```
$ python3 02-sqlalchemy/example-04.py
```

```
Student with locker #4: <Student #2 Netti Nashe>
```

## Exercise 1

- Create a new table called `address`
- It should include the following fields:
  - student **int, foreign key, primary key**
  - street_name **string**
  - number **int**
  - city **string**
- Add an `Address` for each student
- Print out all students along with their addresses using a `join()`

## Exercise 2

- Create a table called `grades`
- It should include the following fields:
    - id **int, primary key, autoincrement**
    - student **int, foreign key**
    - grade **int** or **string -** whichever you prefer
    - date_created **datetime**
- Add some grades for each student
- Print out all grades per each student using `filter`

## *pymongo

## What is pymongo?

- Pymongo is an official MongoDB ORM
- `pip install pymongo`

# Example 03-mongo/example-01.py

```python
import pymongo

client = pymongo.MongoClient( "127.0.0.1", 27017)
db = client.test_db
print(db)

collection = db.test_collection
print(collection)

response = collection.insert_one({ "test": "data", "number": 3})
print(response.inserted_id)

print(collection.find_one())
print(collection.find_one({ "number": {"$gt": 1}}))
```

```
$ python3 03-mongo/example-01.py
```

```
Database(MongoClient(host=['127.0.0.1:27017'], document_class=dict, tz_aware=False, connect=True), 'test_db')
Collection(Database(MongoClient(host=['127.0.0.1:27017'], document_class=dict, tz_aware=False, connect=True), 'test_db'),
'test_collection')
5de65708fdb7b8cad1eee524
{'_id': ObjectId('5de656a2b61c7741d2fbfc58'), 'test': 'data', 'number': 3}
{'_id': ObjectId('5de656a2b61c7741d2fbfc58'), 'test': 'data', 'number': 3}
```

## Exercise 1

Using pymongo:

- Use data in `medical-data.json` to create a new collection: medicaldata
- Find all rows with `procedure_code` equal `0F1F4ZC`
- Find patient with patient_id equal 74, print his full name
- Find a procedure performed on `2019-05-24T01:52:37.000Z` and update its procedure code to `0F1F4ZC`