Lecture : OOP and ORM

OOP (object oriented programming)

(when we create our flight object...)

Classes

```
class Flight:

    def __init__(self, origin, destination, duration)

        self.or   = origin
        self.dest = destination
        self.dur  = duration
```

init method

save data as attributes of our class.

```
def method_1(self):
```

create an instance of our Flight class

```
A = Flight('NewYork', 'London', 185)
A.method_1()
```

A.

invoke method-1

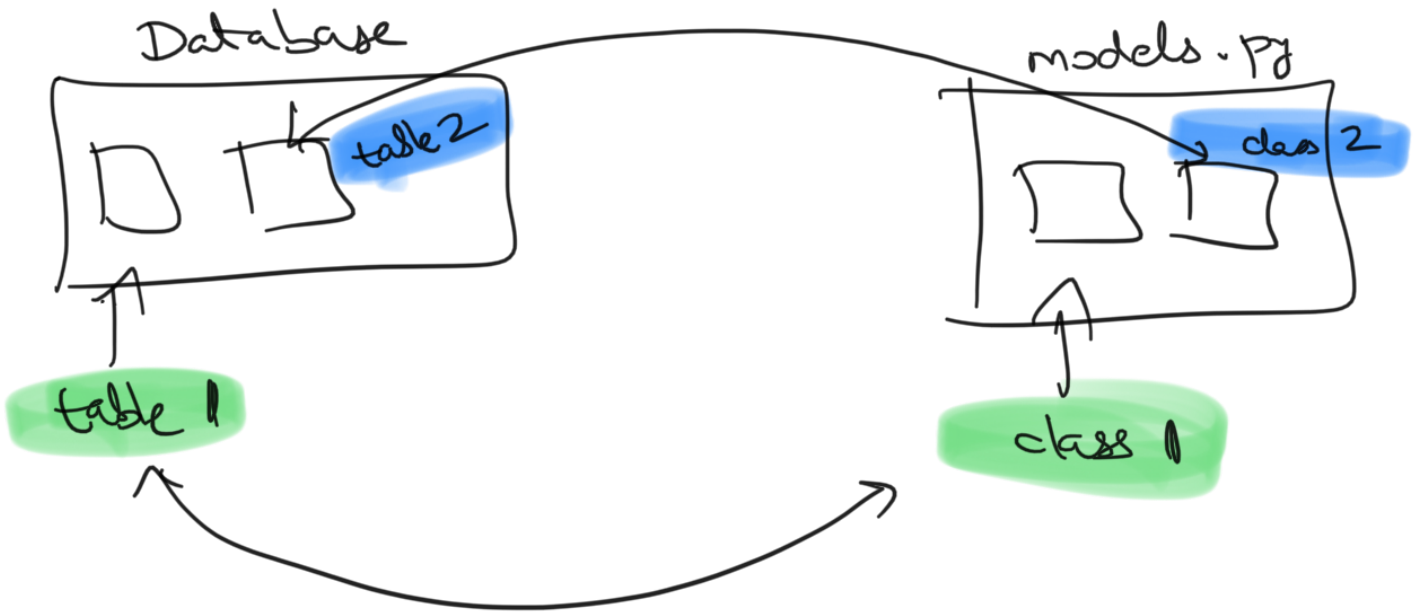
ORM (object relational mapping)

Python ← Flask-SQLAlchemy → SQL

interact

Models.py

from flask_sqlalchemy import SQLAlchemy



db = SQLAlchemy()

inherting from db.Model

class Flight(db.Model):

__tablename__

= "flights"

can interact with tables in db

Flight class ⟷ flights table in database

Add cols to our table...

```
id = db.Column(db.Integer,
                primary_key=True)

origin = db.Column(db.String,
                   nullable=False)

destination = db.Column(db.String,
                        nullable=False)

duration = db.Column(db.Integer,
                     nullable=False)
```

Can create Foreign key also as

```
new_col = db.Column(db.Integer,
          db.ForeignKey("old_table.col name"),
          nullable=False)
```

col in other table which it is referencing

`db.create_all()` ← creates all tables associated to classes...

(have all classes ↔ tables in models.py)

from flask import Flask, render_template, request

from models import * ← import our classes...

app = Flask(__name__)

app.config["SQLALCHEMY_DATABASE_URI"] = os.getenv("DATABASE_URL")

environment variable tells Flask which database to use

app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False

db.init_app(app) ← tie db database to our Flask app...

def main():

```
db.create_all()
```

```
if __name__ == "__main__":
    with app.app_context():
        main()
```

to interact on the command line, with Flask

if you run this python file, creates database - -

Inserting data

```
flight = Flight('NY', 'Paris', 540)
db.session.add(flight)
```

At the end, `db.session.commit()`

Query

```
Flight.query.all()
Flight.query.filter_by(origin='NY').all()
```

← returns a list of Flight obj

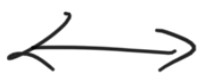
`.first()`

returns a Flight obj--

.count( ) ← how many rows??

Querying for particular id

```
Flight.query.filter_by(id=2).first()
```



```
Flight.query.get(2)
```

Updating data

```
flight = Flight.query.get(6)
flight.duration = new_duration
```

```
db.session.delete(flight)
```

↑ this deletes that row.

more commands

```
Flight.query.order_by(Flight.origin).all()
```

Flight.origin.desc()

```
Flight.query.filter( )
```

↑ boolean expr.

Picks all rows for which booker expr evaluates to True

. all ( )

matches regex ...

ex of Boolean expr

Flight . origin . like (" %a% ")

Flight . origin . in_ (["T", "P"])

and_ (Bool expr_1 , Bool_expr_2)

or_ (" " , " ")

need to import these from SQLAlchemy

Join 2 tables and query

db. session . query (Flight, Passenger)

. filter (Flight . id == Passenger . flight_id)

. all ( )

Add relationships bet tables

passengers = db.relationship("Passenger", backref="flight", lazy=True)

Can do flight.passengers for a flight = Flight()

object--