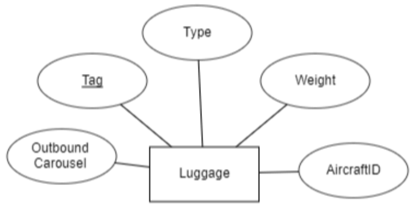
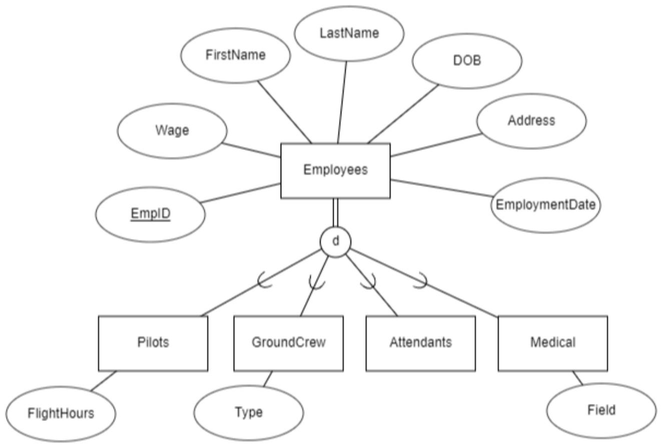
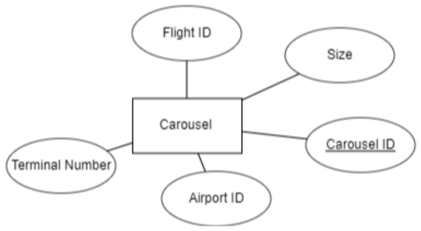
Eliyahu Masinter
Eden Amiga

Stage 1

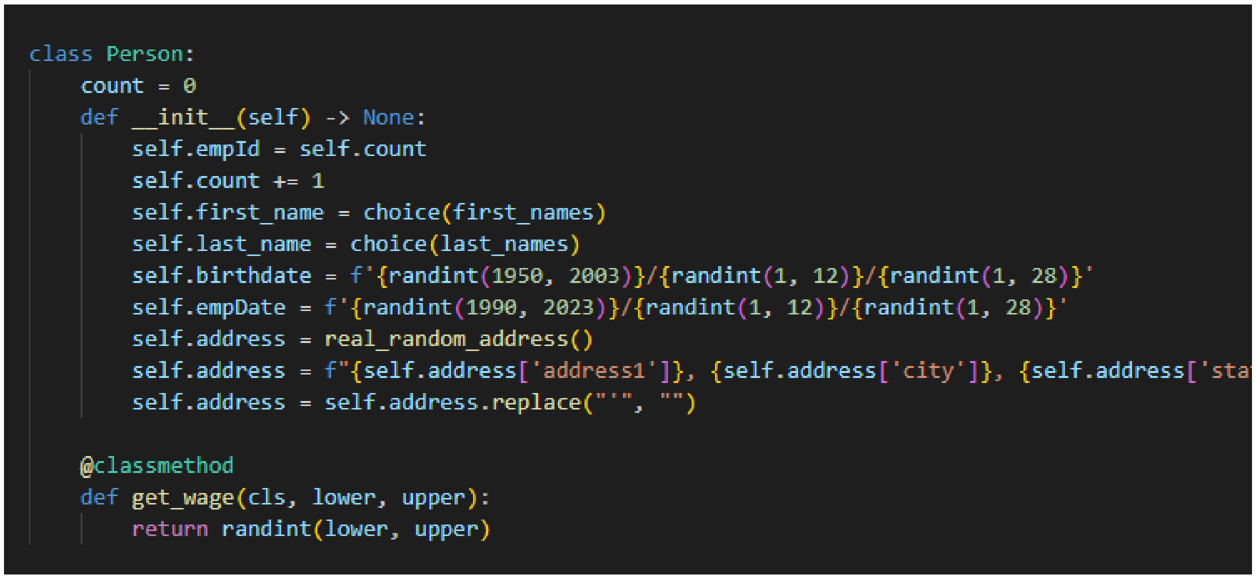






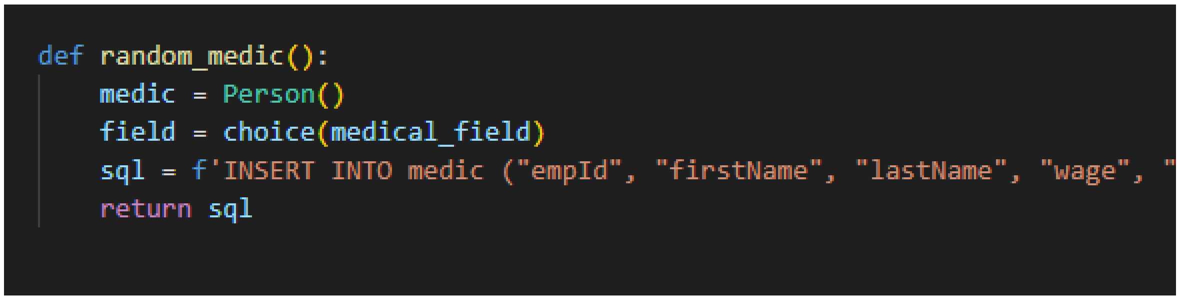
We have chosen to build the part of the database to handle airline employees, luggage, and luggage retrieval. We made sure that each relation is at least in 3nf. The use cases for these particular relations would be to have information on tracking passengers' luggage, making sure the luggage gets sent to the right carousel and that there is a carousel available, in addition to keeping track of all current airline employees. Of course, there are things even within these categories that aren't covered.

In terms of generating the data I used a python script to generate sql commands for each of the tables. Since our tables deal mostly with employees I created a person class which will generate random data about any employee. This is then used in addition to more specific information as needed for each specific employee. Note the use of the real_random_address function. That is from a python module I found to generate random addresses.



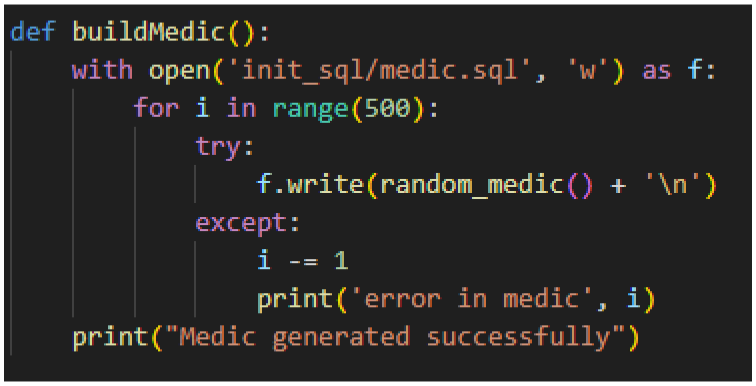
```
class Person:
    count = 0
    def __init__(self) -> None:
        self.empId = self.count
        self.count += 1
        self.first_name = choice(first_names)
        self.last_name = choice(last_names)
        self.birthdate = f'{randint(1950, 2003)}/{randint(1, 12)}/{randint(1, 28)}'
        self.empDate = f'{randint(1990, 2023)}/{randint(1, 12)}/{randint(1, 28)}'
        self.address = real_random_address()
        self.address = f"{self.address['address1']}, {self.address['city']}, {self.address['sta
        self.address = self.address.replace("'", "")

    @classmethod
    def get_wage(cls, lower, upper):
        return randint(lower, upper)
```

Then for each entity we created a function to generate a random one (example below is a random medic)



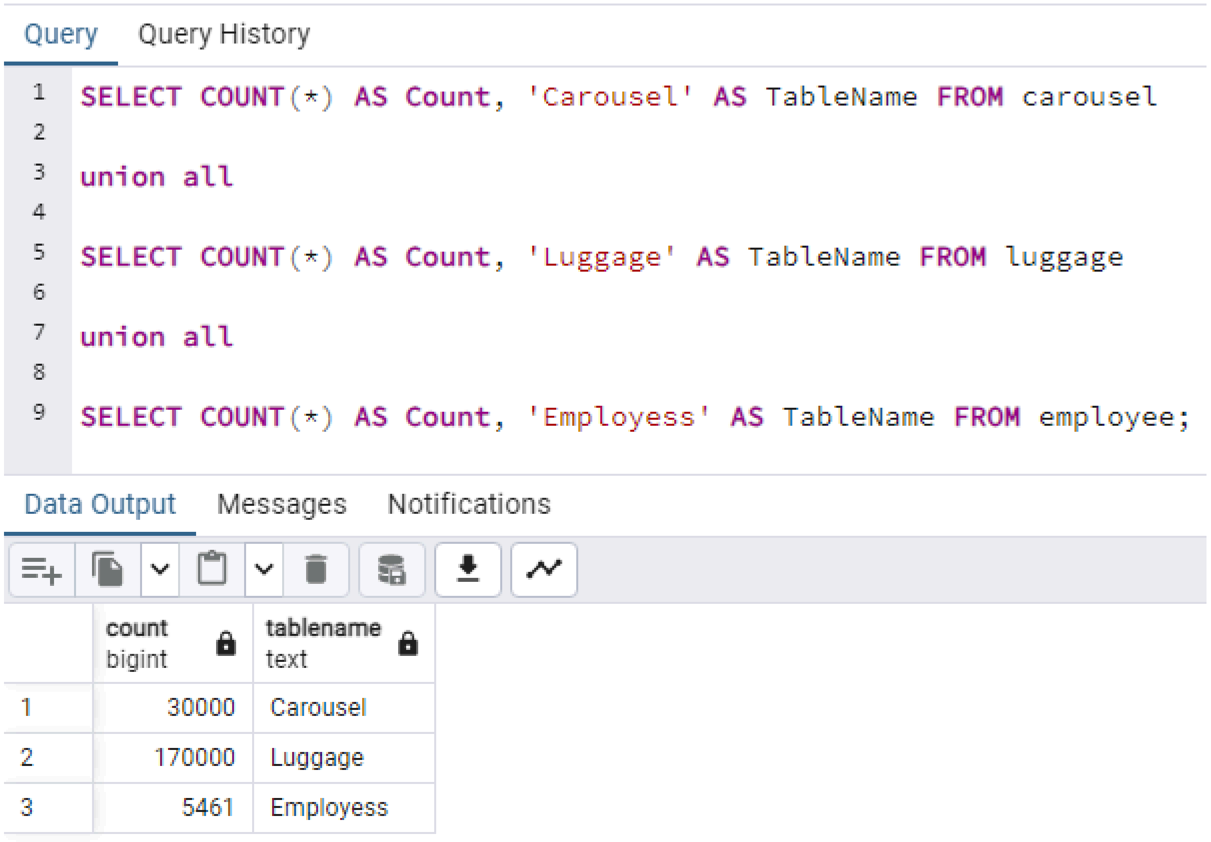
```
def random_medic():
    medic = Person()
    field = choice(medical_field)
    sql = f'INSERT INTO medic ("empId", "firstName", "lastName", "wage", "
    return sql
```

Then I had another function for each entity to generate the desired number of entities. The try-except block is due to the random address generator module which seemed to fail every so often.



```
def buildMedic():
    with open('init_sql/medic.sql', 'w') as f:
        for i in range(500):
            try:
                f.write(random_medic() + '\n')
            except:
                i -= 1
                print('error in medic', i)
    print("Medic generated successfully")
```

Then we called each of these functions which created a sql file for each entity.



Query | Query History

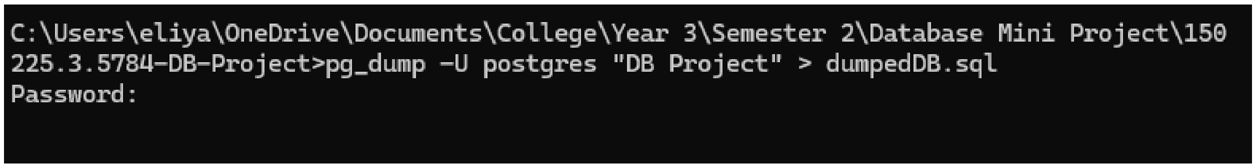
```sql
SELECT COUNT(*) AS Count, 'Carousel' AS TableName FROM carousel

union all

SELECT COUNT(*) AS Count, 'Luggage' AS TableName FROM luggage

union all

SELECT COUNT(*) AS Count, 'Employess' AS TableName FROM employee;
```

Data Output | Messages | Notifications

	count bigint	tablename text
1	30000	Carousel
2	170000	Luggage
3	5461	Employess

Here's the SQL query showing the number of elements in each table. Note that employees is the generic table of Attendant, Ground Crew, Medic, and Pilot. You can't add an employee directly, you must add one of the specific employee but all will show up in the employee table. The number of records (as shown) is more than 200,000.

Dump



```
C:\Users\eliya\OneDrive\Documents\College\Year 3\Semester 2\Database Mini Project\150
225.3.5784-DB-Project>pg_dump -U postgres "DB Project" > dumpedDB.sql
Password:
```

The dump command pg_dump -U postgres "DB Project" > dumpedDB.sql worked (See the dumped file in the repo). The command had no output as the instructions indicated it would.