COMS W4111: Introduction to Databases
Spring 2023, Sections 002

Homework 2, Part 1
Continuing Core Concepts, ER Modeling, Relational Algebra, SQL

Introduction and Overview

HW Objectives

- HW 2, part 1 is for **both tracks.**
- We have covered additional concepts since HW 1. HW 2, part 1 tests and reinforces learning the new concepts.

Submission Instructions

Complete all the tests in this notebook and submit only this notebook as a PDF to GradeScope. To convert the jupyter notebook into a pdf you can use either of the following methods:

- File --> Print Preview --> Print --> Save to PDF
- File --> Download As HTML --> Print --> Save to PDF

Due date: March 1, 11:59 PM EDT on GradeScope

It is recommended that you put the screenshots into the same folder as this notebook so you do not have to alter the path to include your images.

Please read all the instructions thoroughly!

Guidelines

You may not work with or collaborate with anyone in any way to complete the homework. You may speak with the professor and TAs. You may ask **private** questions on Ed if you need clarification.

You may use lecture slides, the textbook slides, the textbook or public information on the web to help you answer your questions. You may not "cut and past" information. Your answer must be in your own words and demonstrate the you understand the concept. If you use information for sources other than lectures, lecture slides, textbook slides or the textbook, you MUST provide a URL to the source you used.

Add Student Information

1. Replace my name with your full name.
2. Replace my UNI with your UNI.
3. Replace "Cool Track" with either "Programming" or "Non-programming."

In [1]:
```
# Print your name, uni, and track below

name = "Chaofan Song"
uni = "cs4185"
track = "Non-programming"

print(name)
print(uni)
print(track)
```

```
Chaofan Song
cs4185
Non-programming
```

Testing Environment

Run the following cells to ensure that your environment is set up.

You may need to change passwords.

General Packages

In [24]:
```
import json
```

In [25]:
```
import csv
```

In [26]:
```
import pandas
```

In [27]:
```
import os
```

pymysql

In [28]:
```
import pymysql
```

In [29]:
```
#
# Run this cell but change your user ID and password.
#
pymysql_conn = pymysql.connect(
    user="root",
    password="dbuserdbuser",
    host="localhost",
    port=3306,
    autocommit=True,
    cursorclass=pymysql.cursors.DictCursor
)
```

In [30]:
```
# You must have installed the db_book DB for this to work.
#
cursor = pymysql_conn.cursor()
sql = "select * from db_book.student"
res = cursor.execute(sql)
result = cursor.fetchall()
```

In [31]:
```
#
# You must have installed the db_book DB for this to work.
#
df1 = pandas.DataFrame(result)
df1
```

Out[31]:

	ID	name	dept_name	tot_cred
0	00128	Zhang	Comp. Sci.	102
1	12345	Shankar	Comp. Sci.	32
2	19991	Brandt	History	80
3	23121	Chavez	Finance	110
4	44553	Peltier	Physics	56
5	45678	Levy	Physics	46
6	54321	Williams	Comp. Sci.	54
7	55739	Sanchez	Music	38
8	70557	Snow	Physics	0
9	76543	Brown	Comp. Sci.	58
10	76653	Aoi	Elec. Eng.	60
11	98765	Bourikas	Elec. Eng.	98
12	98988	Tanaka	Biology	120

ipython-SQL

In [32]:
```
%load_ext sql
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
```

In [33]:
```
#
# Remember to change your user ID and password.
#
%sql mysql+pymysql://root:dbuserdbuser@localhost
```

In [34]:
```
%sql select * from db_book.student where ID=12345
```

```
 * mysql+pymysql://root:***@localhost
1 rows affected.
```

Out[34]:

ID	name	dept_name	tot_cred
12345	Shankar	Comp. Sci.	32

SQLAlchemy

In [35]:
```
from sqlalchemy import create_engine
```

In [36]:
```
#
# Remember to change your user ID and password.
#
sql_url = "mysql+pymysql://root:dbuserdbuser@localhost"
```

In [37]:
```
engine = create_engine(sql_url)
```

In [38]:
```
sql = "select * from db_book.student"
df = pandas.read_sql(sql, con=engine)
```

In [39]:
```
df
```

Out[39]:

	ID	name	dept_name	tot_cred
0	00128	Zhang	Comp. Sci.	102.0
1	12345	Shankar	Comp. Sci.	32.0
2	19991	Brandt	History	80.0
3	23121	Chavez	Finance	110.0
4	44553	Peltier	Physics	56.0
5	45678	Levy	Physics	46.0
6	54321	Williams	Comp. Sci.	54.0
7	55739	Sanchez	Music	38.0
8	70557	Snow	Physics	0.0
9	76543	Brown	Comp. Sci.	58.0
10	76653	Aoi	Elec. Eng.	60.0
11	98765	Bourikas	Elec. Eng.	98.0
12	98988	Tanaka	Biology	120.0

Relational Algebra

Instructions

- Use the schema and data (https://dbis-uibk.github.io/relax/calc/gist/4f7866c17624ca9dfa85ed2482078be8/relax-silberschatz-english.txt/0) associated with the recommended textbook. Clicking the link should take you directly to the RelaX page configured with the DB.
- Your submission format for each question is:
  - A Markdown cell with a copy of your relational algebra statement.
  - A screen shot of the execution. If the expression returns several result pages, you only need to show the first page.
  - There is an example below.

Example

Question:

Write a relational statement that produces a relation of the form:

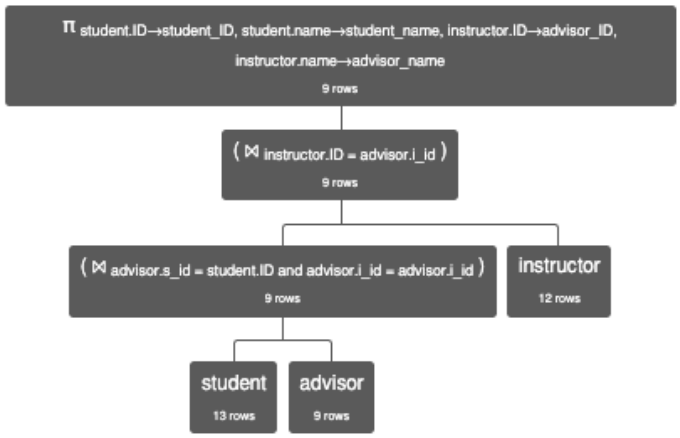
- `student_id` is `student.ID`
- `student_name` is `student.name`
- `advisor_id` is `instructor.ID`
- `advisor_name` is `instructor.name`

The `advisor` table is an associative entity for `student` and `instructor`.

Relational statement:

```
    advisor_ID←instructor.ID, advisor_name←instructor.name
((
    student ⋈
        advisor.s_id=student.ID ∧ advisor.i_id = advisor.i_id
    advisor
)
⋈ instructor.ID=advisor.i_id
instructor
)
```

Capture



π student.ID→student_ID, student.name→student_name, instructor.ID→advisor_ID, instructor.name→advisor_name ((student ⋈ advisor.s_id = student.ID and advisor.i_id = advisor.i_id advisor) ⋈ instructor.ID = advisor.i_id instructor)

Execution time: 2 ms

student_ID	student_name	advisor_ID	advisor_name
128	'Zhang'	45565	'Katz'
12345	'Shankar'	10101	'Srinivasan'
23121	'Chavez'	76543	'Singh'
44553	'Peltier'	22222	'Einstein'
45678	'Levy'	22222	'Einstein'
76543	'Brown'	45565	'Katz'
76653	'Aoi'	98345	'Kim'
98765	'Bourikas'	98345	'Kim'
98988	'Tanaka'	76766	'Crick'

Question R1: Courses and Prereq

Question

- Please run the following relational algebra statement. The statement produces a table of course that do not have preqs and courses that are not prereq.

```
(π course_id←course.course_id, prereq_id←'None' (course ▷ prer
eq))
∪
(π
    course_id←'None', prereq_id←course.course_id
    (course ▷ course.course_id=prereq.prereq_id prereq)
)
```

- Please write an equivalent query that does not use `anti-join.`

Answer

(π course_id←course.course_id, prereq_id←'None'
(σ prereq.prereq_id = null (course ⟕ prereq)))

∪

(π course_id←'None', prereq_id←course.course_id (σ prereq.prereq_id = null (course ⟕ course.course_id=prereq.prereq_id prereq)))

Type *Markdown* and LaTeX: α^2

Question R2: Instructors and Credits

Question

Please write a relational algebra statement that produces a relation of the form:

- Columns
  - `instructor_name` is `instructor.name`
  - `instructor_id` is `instructor.ID`
  - `teaching_credits_yearly` is the total course credits that the instructor taught in a year. The information needed is in `teaches` and `course` .
  - `year` is from `teaches` .
- Sort the result by `instructor_name` and then ``year```.

Answer

τ instructor.name, teaches.year (γ instructor.name, year,ID; sum(course.credits) → teaching_credits_yearly (instructor ⋈ teaches ⋈ course))

ER Modeling and Implementation

Data

- We will do bottom up modeling and implementation starting with data.
- There are two entity sets for reengineering:
  1. `customers`
  2. `reservations` is a single set containing three types of reservation:
     A. `Flight`
     B. `Rental car`
     C. `Hotel`
- The input data is:

In [61]:
```
customers_df = pandas.read_csv('/Users/songchaofan/Desktop/S23-W4111-H
```

In [62]:
```
customers_df
```

Out[62]:

	customer_id	first_name	last_name	email
0	4fc2a5e7-859c-494f-9f82-9ef3da9c6265	Lew	Jagiela	ljagiela0@comsenz.com
1	e649dc5e-5570-43f6-837c-89379800179f	Karin	De Ruggiero	kderuggiero1@linkedin.com
2	a20afabf-86af-4070-9c7c-4b4643202013	Georgena	Gapp	ggapp2@ed.gov
3	12d17b18-f009-49fd-a887-eaab9cd60274	Enrika	Thripp	ethripp3@newsvine.com
4	d9a14dd2-0000-4227-bf30-6907efb4caa7	Edin	Seyler	eseyler4@geocities.jp
...	...	...	...	...
995	bbe5ca2a-b5e1-4ff6-a1d6-450bdf41f6ec	Sean	Wagen	swagenr8@abc.net.au
996	a6632fc7-316b-4ad5-9b2c-977f7a732d54	Delmar	Benoy	dbenoyr9@multiply.com
997	3f05996b-5c79-429f-aacb-2367255cc495	Crin	Smooth	csmoothra@blogger.com
998	8e88da67-2e28-4330-bfa2-17c4369c96cf	Riobard	Ricket	rricketrb@va.gov
999	493ca6a1-b2c7-4c1d-817f-3a0d73f8f24c	Neville	Blowen	nblowenrc@etsy.com

1000 rows × 4 columns

In [63]:
```
reservations_df = pandas.read_csv("/Users/songchaofan/Desktop/S23-W411
```

In [64]:
```
import numpy as np
```

In [65]:
```
reservations_df = reservations_df.replace({np.nan: None})
```

In [66]:
```
reservations_df
```

Out[66]:

	reservation_id	customer_id	reservation_type	price	reservation_date	city	res
0	b097458e-898a-40a0-85eb-cedb0ea47466	4fc2a5e7-859c-494f-9f82-9ef3da9c6265	Car	$126.30	2/23/2022	Granja	
1	67070e81-81f0-4d95-9302-98486a08bf73	e649dc5e-5570-43f6-837c-89379800179f	Hotel	$792.89	1/30/2023	Kavalerovo	
2	f164e71e-4874-4c42-beb4-27c2c6f5af0f	a20afabf-86af-4070-9c7c-4b4643202013	Hotel	$660.09	9/30/2022	Daxi	
3	7b088670-9567-4068-8571-946dd085dab5	12d17b18-f009-49fd-a887-eaab9cd60274	Flight	$423.59	1/9/2023	None	
4	ca849908-b88e-49d8-b52b-2aeb88b3c949	d9a14dd2-0000-4227-bf30-6907efb4caa7	Hotel	$552.07	11/5/2022	Albuquerque	
...	...	...	...	...	...	...	
995	16b955d3-0181-4ad3-9654-0fea52bf1094	bbe5ca2a-b5e1-4ff6-a1d6-450bdf41f6ec	Hotel	$169.73	2/3/2023	Cibunar	
996	caa55cf1-3032-42b4-94d5-2f8ac70585ce	a6632fc7-316b-4ad5-9b2c-977f7a732d54	Hotel	$517.19	6/12/2022	Quiling	
997	e99b7c4f-1e14-42e1-9598-5010172f3454	3f05996b-5c79-429f-aacb-2367255cc495	Car	$636.63	2/11/2023	Hukeng	
998	c54b4398-3eb7-46a0-9906-d2acc04f0e90	8e88da67-2e28-4330-bfa2-17c4369c96cf	Car	$174.76	11/10/2022	Pravdinsk	
999	4c942bd8-2d80-4d57-9281-0e6387ed7f51	493ca6a1-b2c7-4c1d-817f-3a0d73f8f24c	Car	$662.24	11/10/2022	Kongolo	

1000 rows × 15 columns

Model

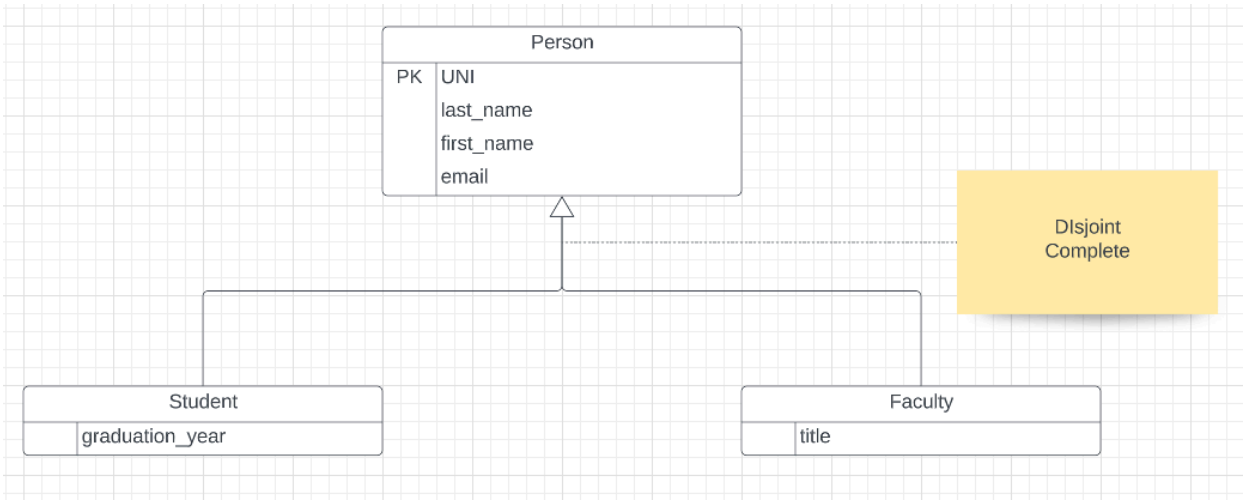
- You are going to model *interitance/specialization.*
  - There is a base entity type *Reservation.*
  - *CarReservation* **isA** *Reservation.*
  - *FlightReservation* **isA** *Reservation.*
  - *HotelReservation* **isA** *Reservation.*
  - The specialization is *complete* and *disjoin.*

- The following table shows which properties in the input data apply to which reservation type.

Property	Meaning	Car	Flight	Hotel
reservation_id	Unique ID, primary key	X	X	X
customer_id	Unique ID, foreign key	X	X	X
reservation_type	Enum for type of reservation	X	X	X
price	Price to charge	X	X	X
reservation_date	Date reservation made/changed	X	X	X
city	City for reservation	X		X
reservation_start	Start date	X		X
reservation_end	End date	X		X
car_category	Enum	X		
room_category	Enum			X
bed_category	Enum			X
flight_departure_time	Takeoff time		X	
flight_arrival_time	Arrival time		X	
departure_airport	Departure airport code		X	
arrival_airport	Arrival airport code		X	

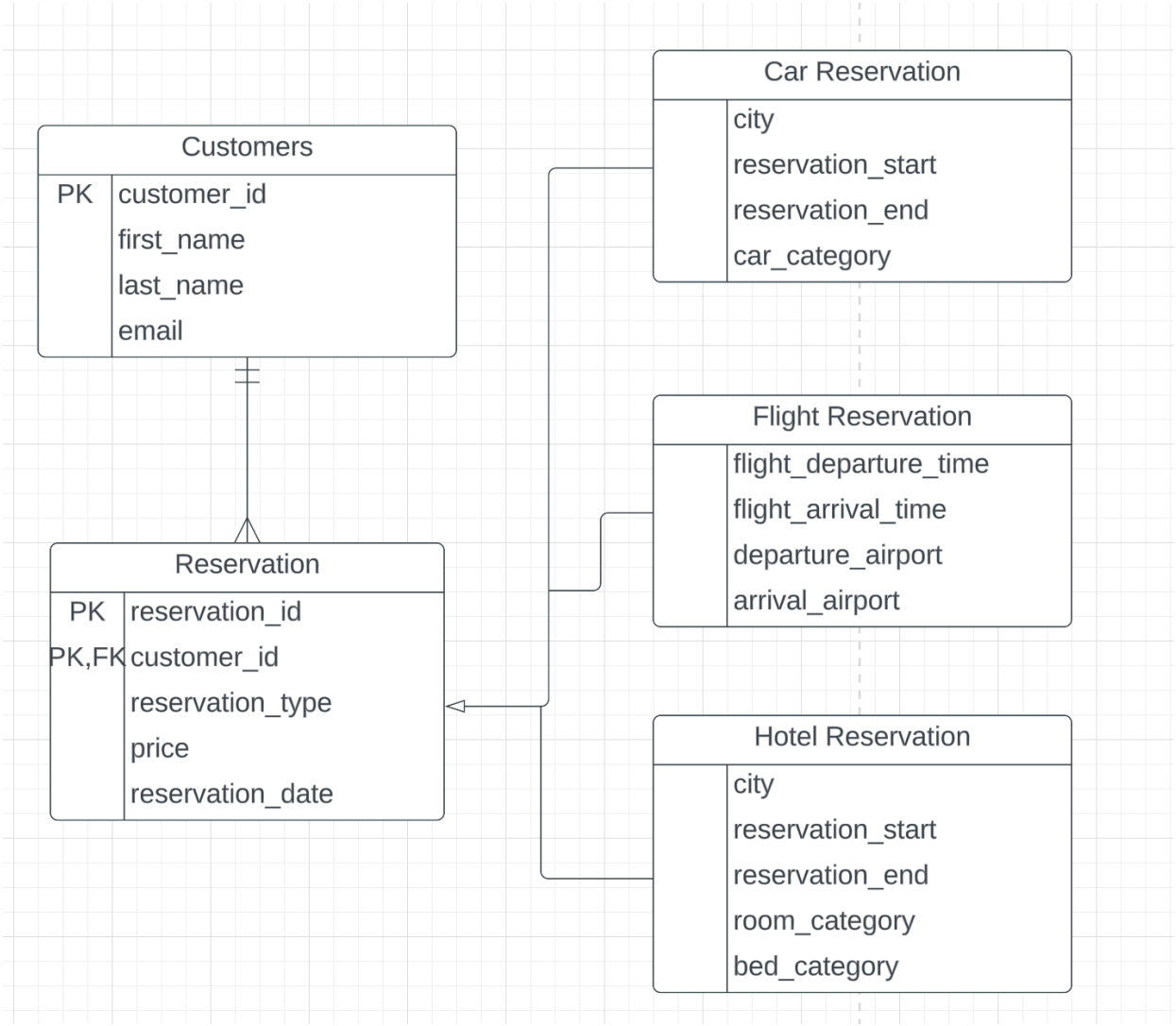
- The implementation pattern for inheritance will be:
  - *Reservations* is a view.
  - There is a table for each of *CarReservation, FlightReservation, HotelReservation.*

- We covered a simple example of modeling inheritance in class. The diagram notiation we used for *Person, Student, Faculty* is:



- Draw diagram for *Reservation, CarReservation, FlightReservation, HotelReservation* using the pattern above.
- Your diagram should include a foreign key relationship from *Reservations* to *Customers.*

Diagram

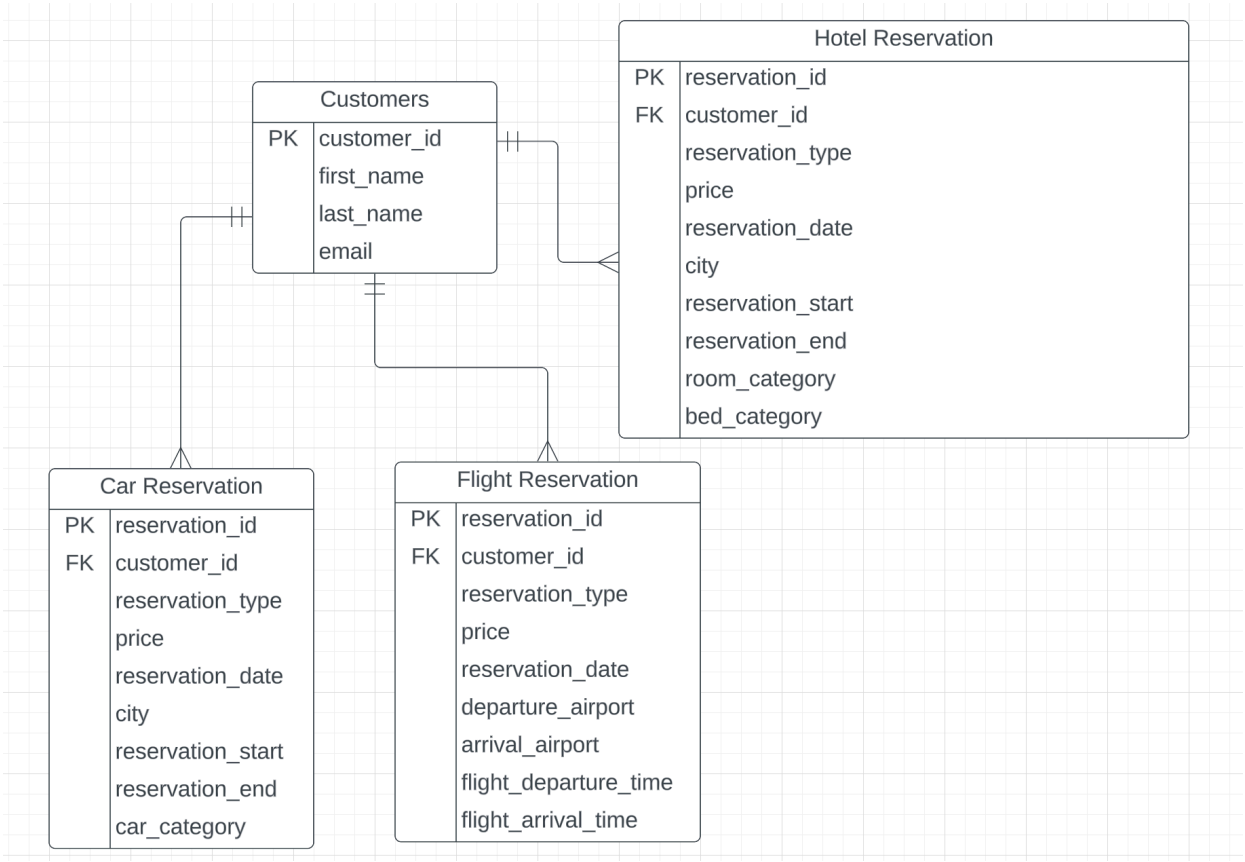


Type *Markdown* and LaTeX: α^2

Implement

- You must draw a logical ER diagram for the four tables you will create. Your diagram does not have to include the view.
- Your diagram must include primary keys and foreign keys.

Diagram



- Save the two dataframes into two tables (using pandas `to_sql` and schema `s23_w4111_hw2_p1_<uni>`)
  - `customers`
  - `reservations`
- Use SQL to create the tables and data for the three types of reservation.
- Set appropriate data types and keys for the tables.
- Show and execute your SQL for creating the tables and modifying the schema.

In [86]:
```
%sql create database if not exists W4111_HW2_p1_cs4185;
```

```
 * mysql+pymysql://root:***@localhost
1 rows affected.
```

Out[86]: []

In [87]:
```
import pandas as pd

reservations_df['price'] = reservations_df['price'].replace(
    '$', '').astype(float)
reservations_df['reservation_date'] = pd.to_datetime(
    reservations_df['reservation_date'])
reservations_df['reservation_start'] = pd.to_datetime(
    reservations_df['reservation_start'])
reservations_df['reservation_end'] = pd.to_datetime(
    reservations_df['reservation_end'])
```

In [88]:
```
customers_df.to_sql("customers",
                    schema="W4111_HW2_p1_cs4185",
                    index=False, if_exists="replace", con=engine)
reservations_df.to_sql("reservations",
                       schema="W4111_HW2_p1_cs4185",
                       index=False, if_exists="replace", con=engine)
```

Out[88]: 1000

In [89]:
```
%%sql
use W4111_HW2_p1_cs4185;

create or replace view CarReservation as
    select reservation_id, customer_id, reservation_date, price,
    city, reservation_start, reservation_end, car_category
    from reservations
    where reservation_type = 'Car'
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
0 rows affected.
```

Out[89]: []

In [90]:
```
%%sql
use W4111_HW2_p1_cs4185;

select * from CarReservation
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
346 rows affected.
```

Out[90]:

reservation_id	customer_id	reservation_date	price	city	reservation_start	reser
b097458e-898a-40a0-85eb-cedb0ea47466	4fc2a5e7-859c-494f-9f82-9ef3da9c6265	2022-02-23 00:00:00	126.3	Granja	2022-02-21 00:00:00	
79399360-5553-4876-9058-f7f113934bd9	44a1ddc8-8ab4-490a-bfc2-7a860ad83d88	2022-08-12 00:00:00	521.83	Belfast	2022-07-11 00:00:00	
3b699a17-fef2-4207-99e1-1ac0b4178aa0	b3784afa-2e26-48b9-aa89-231569eb4989	2022-10-15 00:00:00	330.33	Ivanovo	2022-04-14 00:00:00	
07529b64-8491-4b97-	790b5b32-7455-4fb2-	2022-04-06	792.48	Mojo	2022-07-23	

In [91]:
```
%%sql
use W4111_HW2_p1_cs4185;

create or replace view FlightReservation as
    select reservation_id, customer_id, reservation_date,
    price, flight_departure_time, flight_arrival_time,
    departure_airport, arrival_airport
    from reservations
    where reservation_type = 'Flight'
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
0 rows affected.
```

Out[91]: []

In [92]:
```
%%sql
use W4111_HW2_p1_cs4185;

select * from FlightReservation
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
334 rows affected.
```

Out[92]:

reservation_id	customer_id	reservation_date	price	flight_departure_time	flight_arrival_time
7b088670-9567-4068-8571-946dd085dab5	12d17b18-f009-49fd-a887-eaab9cd60274	2023-01-09 00:00:00	423.59	12:59 AM	3:06 AM
337956fb-4126-4f6a-b546-9ed743c77b3f	2177c886-4e62-4745-a470-0f690e75518d	2022-12-30 00:00:00	846.63	1:24 AM	3:36 PM
dfb58b01-8051-486d-85bf-7b820966e508	3db71dcc-b81a-4ea1-893a-9598025d1cf1	2022-03-07 00:00:00	497.46	10:48 PM	12:57 PM
c8b3f140-d002-4994-	0a70c4a7-0824-45c2-	2022-12-02	990.85	8:03 AM	3:06 PM

In [93]:
```
%%sql
use W4111_HW2_p1_cs4185;

create or replace view HotelReservation as
    select reservation_id, customer_id, reservation_date,
    price, reservation_start, reservation_end,
    room_category, bed_category, city
    from reservations
    where reservation_type = 'Hotel'
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
0 rows affected.
```

Out[93]: []

In [94]:
```
%%sql
use W4111_HW2_p1_cs4185;

select * from HotelReservation
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
320 rows affected.
```

Out[94]:

reservation_id	customer_id	reservation_date	price	reservation_start	reservation_end	roon
67070e81-81f0-4d95-9302-98486a08bf73	e649dc5e-5570-43f6-837c-89379800179f	2023-01-30 00:00:00	792.89	2022-08-23 00:00:00	2023-01-19 00:00:00	
f164e71e-4874-4c42-beb4-27c2c6f5af0f	a20afabf-86af-4070-9c7c-4b4643202013	2022-09-30 00:00:00	660.09	2022-08-20 00:00:00	2022-08-23 00:00:00	
ca849908-b88e-49d8-b52b-2aeb88b3c949	d9a14dd2-0000-4227-bf30-6907efb4caa7	2022-11-05 00:00:00	552.07	2022-08-24 00:00:00	2022-04-15 00:00:00	
0df8910a-fcb6-42d9-	2c082499-1c7f-4539-	2022-07-18	776.6	2022-08-15	2022-04-27	

In [ ]:

In [95]:
```
%%sql
use W4111_HW2_p1_cs4185;

alter table reservations
modify column reservation_id varchar(200),
modify column customer_id varchar(200),
modify column reservation_type enum('Car','Hotel','Flight'),
modify column reservation_date date,
modify column price float(50),
modify column city varchar(200),
modify column reservation_start date,
modify column reservation_end date,
modify column car_category enum('SUV','Midsize','Compact'),
modify column room_category enum('Studio','Suite','One Room'),
modify column bed_category enum('King','Two Doubles'),
modify column flight_departure_time varchar(200),
modify column flight_arrival_time varchar(200),
modify column departure_airport varchar(200),
modify column arrival_airport varchar(200);
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
1000 rows affected.
```

Out[95]: []

In [96]:
```
%%sql
use W4111_HW2_p1_cs4185;

alter table customers
modify column customer_id varchar(200);
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
1000 rows affected.
```

Out[96]: []

In [97]:
```
%%sql
use W4111_HW2_p1_cs4185;

alter table customers
add constraint pk_customers primary key (customer_id);
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
0 rows affected.
```

Out[97]: []

In [98]:
```
%%sql
use W4111_HW2_p1_cs4185;

alter table reservations
add constraint pk_reservations primary key (reservation_id);
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
0 rows affected.
```

Out[98]: []

In [99]:
```
%%sql
use W4111_HW2_p1_cs4185;

alter table reservations
add constraint fk_ReservCust
foreign key (customer_id) references customers(customer_id);
```

```
 * mysql+pymysql://root:***@localhost
0 rows affected.
1000 rows affected.
```

Out[99]: []

In [ ]: