cristian.cartofeanu@gmail.com

Rent a car - Semstral Work

Description

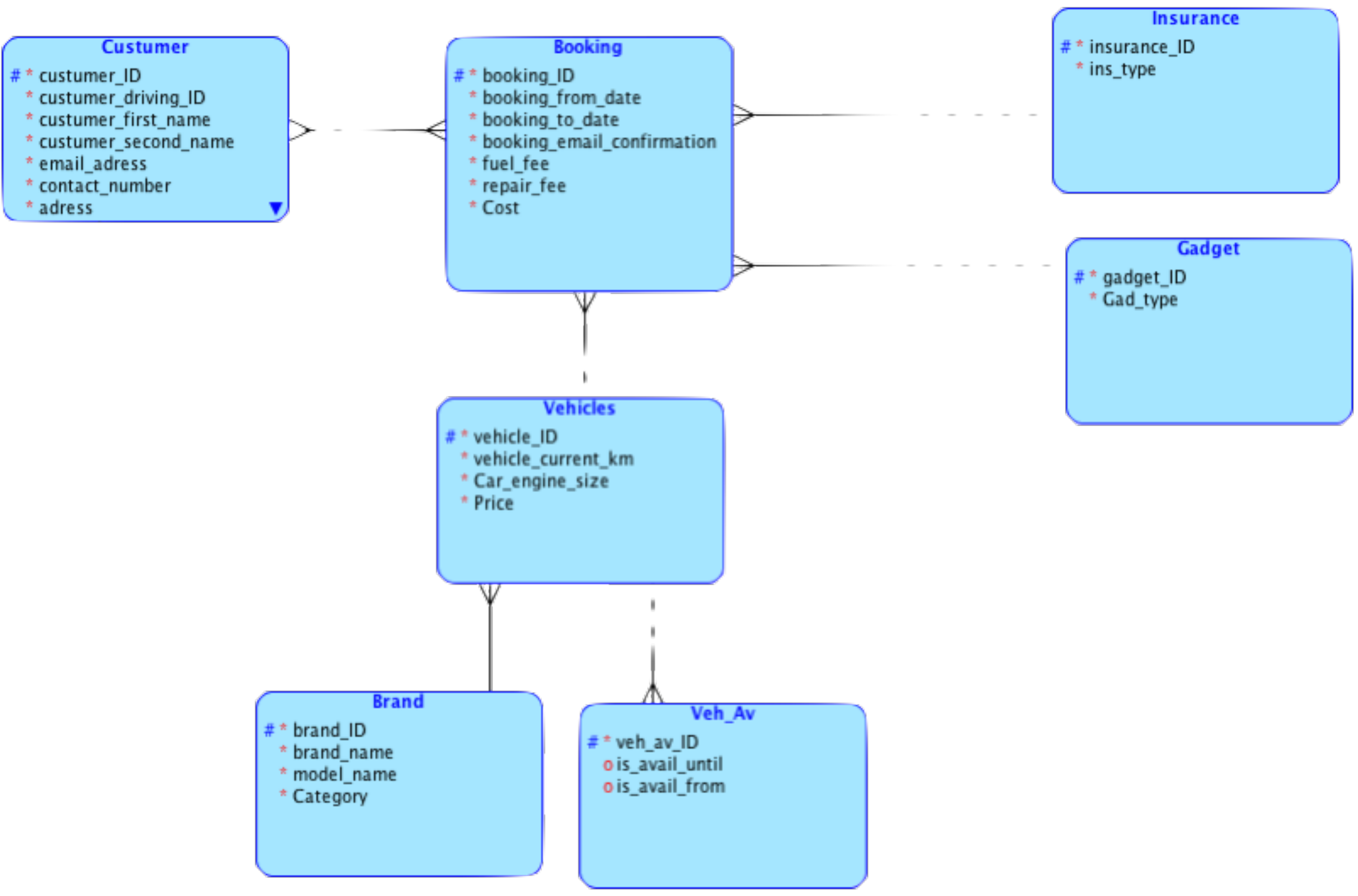
There is a rent car company which provides car renting. The car renting is possible only with customer_ID, driving_ID and personal data.

Each car is identified by vehicle_ID. It has it's brand, model, category and availability status, all this four attributes are mandatory. Car category represents the class, type of the car (convertible, sedan, etc...). We store also information about vehicle current km, vehicle engine size and the price for each vehicle separatly depending on it's mileage .

Each booking procedure requires from the customer to select from the option pickup and return date, vehicle category, additional adding some gadgets to the car (GPS, child seat and satellite radio), adding our type of insurance (full covering and partial covering), fill in personal data(name, surname, e-mail, etc...).

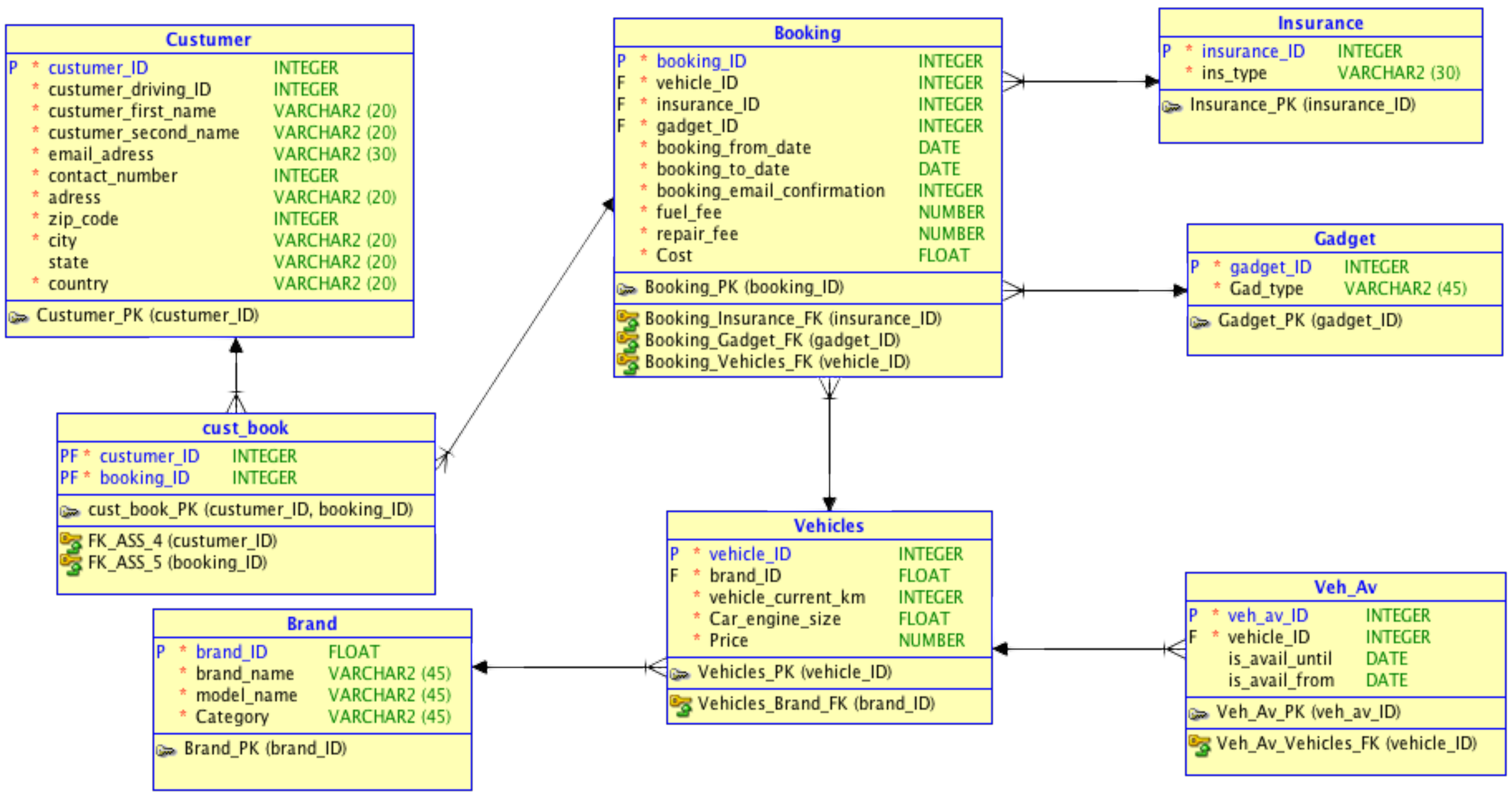
After the car was returned, possible repair_fee or fuel_fee may appear if the car was damaged or the tank fuel is not full, all this information is stored in booking procedure.

Data model



There are no loops in data model.

Relational model



SQL Statements + Relational Algebra

1. List brands of cars and order it by category.

```
BRAND[CATEGORY]
```

```
SELECT DISTINCT * FROM BRAND order by CATEGORY;
```

2. The name, surname and the contact number of the customers who have curently a booking reservation.

```
(CUSTUMER × CUST_BOOK) (CUSTUMER.custumer_id=CUST_BOOK.custumer_id)
      [custumer_first_name,custumer_second_name,contact_number ]
```

```
SELECT DISTINCT CUSTUMER_FIRST_NAME, CUSTUMER_SECOND_NAME, CONTACT_NUMBER
FROM CUSTUMER CROSS JOIN CUST_BOOK
WHERE CUSTUMER.CUSTUMER_ID = CUST_BOOK.CUSTUMER_ID
```

3. The list off all vehicles and their brands

```
(VEHICLES×BRAND)(VEHICLES.brand_id=BRAND.brand_id)
      [vehicle_id,vehicle_current_km,price,brand_name,model_name,category]
```

```
SELECT DISTINCT VEHICLE_ID, VEHICLE_CURRENT_KM, PRICE, BRAND_NAME, MODEL_NAME, CATEGORY
FROM VEHICLES CROSS JOIN BRAND
WHERE VEHICLES.BRAND_ID = BRAND.BRAND_ID
```

4. The name, surname and contact number of all customers who dont have right now any booking

```
CUSTUMER[custumer_first_name,custumer_second_name,contact_number]  \ (CUSTUMER * CUST_BOOK * BOOKING)
      (custumer_id=custumer_id   ^ booking_id=booking_id)[custumer_first_name,custumer_second_name,contact_numbe
```

```
SELECT DISTINCT CUSTUMER_FIRST_NAME, CUSTUMER_SECOND_NAME, CONTACT_NUMBER
FROM CUSTUMER
MINUS
SELECT DISTINCT CUSTUMER_FIRST_NAME, CUSTUMER_SECOND_NAME, CONTACT_NUMBER
FROM CUSTUMER NATURAL JOIN CUST_BOOK NATURAL JOIN BOOKING
WHERE CUSTUMER_ID = CUSTUMER_ID AND BOOKING_ID = BOOKING_ID
```

5. How many BWM cars we have

```
select count(*) "BMW"
from Brand
where brand_name = 'BMW';
```

6. What's the average millage of all cars

```
select round(avg(vehicle_current_km),0) as "Average millage of all cars"
from Vehicles;
```

7. Update the name and surname of the customer with ID='30'

```
UPDATE CUSTUMER
SET custumer_first_name='Angelaa', custumer_second_name='Moraruu'
WHERE custumer_id='30';
```

8. Dellete custumer that have the name Angel and the surname Wilkerson from our database

```
DELETE FROM CUSTUMER
WHERE custumer_first_name='Angel' AND custumer_second_name='Wilkerson';
```

9. Find all vehicles[vehicle_id,brand_id] which had 1 or more booking

```
select vehicle_id, brand_id
from VEHICLES V
where 1 <= (select count(*)
from BOOKING B
where V.vehicle_id = B.vehicle_id);
```

10. Select the vehicle_id and the model of the cars that have a booking at the moment

```
(BOOKING * BRAND * VEHICLES)[vehicle_id,model_name]
```

```
SELECT DISTINCT VEHICLE_ID, MODEL_NAME
FROM BOOKING NATURAL JOIN BRAND NATURAL JOIN VEHICLES
```

11. How many custumers have a booking of the vehicle_id=70

```
select count(city) "Nr of custumers "
from CUSTUMER join CUST_BOOK cb using(custumer_id) join BOOKING b on (cb.booking_id=b.booking_id)
where b.vehicle_id='70'
```

12. For each custumer find the number of booking it has.

```
select c.custumer_id, custumer_first_name "CUSTUMER", count(cb.custumer_id) "Nr of booking"
from CUSTUMER c left outer join CUST_BOOK cb on (c.custumer_id=cb.custumer_id)
         join BOOKING using (booking_id)
group by c.custumer_id, custumer_first_name;
```

13. Find vehicles which were not booked.

```
select vehicle_id
from VEHICLES ve
where not exists (select *
               from BOOKING b
               where ve.vehicle_id=b.vehicle_id)
order by vehicle_id;
```

14. Find vehicles which were not booked.

```
select vehicle_id
from VEHICLES
where vehicle_id not in ( select vehicle_id
                 from BOOKING)
order by vehicle_id;
```

15. Find vehicles which were not booked.

```
VEHICLES[vehicle_id]   \   BOOKING[vehicle_id]
```

```
SELECT DISTINCT VEHICLE_ID
FROM VEHICLES
MINUS
SELECT DISTINCT VEHICLE_ID
FROM BOOKING
```

16. Find vehicle (vehicle_id, brand_id) which has more then 1 booking at the moment

```
select vehicle_id, brand_id
from VEHICLES join BOOKING using (vehicle_id)
group by vehicle_id, brand_id
having count(*) >= 2;
```

17. Find all the bookings (all attributes) that contain both insurance and gadget

```
(BOOKING<*GADGET)  ∩  (BOOKING<*INSURANCE)
```

```
SELECT DISTINCT BOOKING_ID, VEHICLE_ID, INSURANCE_ID, GADGET_ID, BOOKING_FROM_DATE, BOOKING_TO_DATE,
        BOOKING_EMAIL_CONFIRMATION, FUEL_FEE, REPAIR_FEE, COST
FROM BOOKING NATURAL JOIN GADGET
INTERSECT
SELECT DISTINCT BOOKING_ID, VEHICLE_ID, INSURANCE_ID, GADGET_ID, BOOKING_FROM_DATE, BOOKING_TO_DATE,
  BOOKING_EMAIL_CONFIRMATION, FUEL_FEE, REPAIR_FEE, COST
FROM BOOKING NATURAL JOIN INSURANCE
```

18. Find all the bookings (all attributes) that contain insurance or gadget

```
(BOOKING<*GADGET)   ∪   (BOOKING<*INSURANCE)
```

```
SELECT DISTINCT BOOKING_ID, VEHICLE_ID, INSURANCE_ID, GADGET_ID, BOOKING_FROM_DATE, BOOKING_TO_DATE,
        BOOKING_EMAIL_CONFIRMATION, FUEL_FEE, REPAIR_FEE, COST
FROM BOOKING NATURAL JOIN GADGET
UNION
SELECT DISTINCT BOOKING_ID, VEHICLE_ID, INSURANCE_ID, GADGET_ID, BOOKING_FROM_DATE, BOOKING_TO_DATE,
  BOOKING_EMAIL_CONFIRMATION, FUEL_FEE, REPAIR_FEE, COST
FROM BOOKING NATURAL JOIN INSURANCE
```

19. For each custumer find the number of booking it has, including custumers withoud booking

```
select g.custumer_id, g.custumer_first_name as "Custumer",
(select count(*)
 from CUST_BOOK mg
 where g.custumer_id = mg.custumer_id) as "Nr of bookings"
from CUSTUMER g;
```

20. Create a view Car_engine which lists cars (vehicle_id) that has the engine size equal to 2.

```
CREATE OR REPLACE VIEW Car_engine_2 AS
SELECT vehicle_id,car_engine_size
FROM VEHICLES
WHERE car_engine_size = '2';
```

21. Select vehicle that has a booking with all 3 kinds of gadets.

```
select*
from booking
where vehicle_id not in(
select vehicle_id from (
select  BOOKING.vehicle_id, GADGET.gadget_id
from GADGET,BOOKING minus
select  vehicle_id, gadget_id
from BOOKING));
```

22. Select vehicle_id from the view of the cars that have engine size equal to 2.

```
SELECT DISTINCT vehicle_id
FROM CAR_ENGINE_2;
```

23. Select all vehicles that has price equal or greater to 270.

```
VEHICLES(price >='270 ')
```

```
SELECT DISTINCT *
FROM VEHICLES
WHERE PRICE >= '270 '
```

24. Select all bookings that dosent have child Seats additional order.

```
BOOKING(gadget_id !='2')
```

```
SELECT DISTINCT *
FROM BOOKING
WHERE GADGET_ID <> '2'
```

SQL Statements Cover

Category	Covered by statement num:	Description of statement category
A	1, 23	simple query (SELECT ... FROM ... WHERE)
B	2, 10	choose all satisfaying a condition
C	4, 13, 14, 15, 24	choose all do not satisfaying a condition
D	21	general quantifie query
F1	10, 11, 17, 18	join - JOIN ON
F2	11	join - NATURAL JOIN I JOIN USING
F3	3	join - CROSS JOIN (cartesian product)
F4	12	outer join - LEFT I RIGHT OUTER JOIN
G1	9, 13	subquery inside WHERE
G2	4	subquery inside FROM
G3	19	subquery inside SELECT
G4	13	correlated subquery (EXISTS I NOT EXISTS)
H1	18	set union query - UNION
H2	15	set substraction query - MINUS (in Oracle) / EXCEPT by stnadard
H3	17	set intersection - INTERSECT
I1	5, 6, 11, 12	aggregation functions (count I sum I min I maxI avg)
I2	16	aggregations with GROUP BY (HAVING) clause
J	13, 14, 15	the same query using three different formulations
L	20	VIEW
M	22	query which uses a view
M	7	UPDATE statement with subquery in WHERE/SET
N	8	DELETE statement with subquery in WHERE clause

Scripts

SQL_Developer project- Car_Rent_SQL_Developer_Project

Input data of this database Input_Data

Database creation scritpt database_creation_script

Conclussions

This semestrial project gave me some basic knowledge of constructing our own database model and how to work with it.

References

[1] Sample of the DB project - https://users.fit.cvut.cz/~valenta/bie-dbs/sem-project/index.html