1 Relational Algebra and SQL (6 points)

Consider the following relational schemas.

[persons] : {[pid: int, name: varchar, birth_year: int]}
[libraries] : {[lid: int, city: varchar]}
[members] : {[mid: (persons), favorite_author: (authors), late_fees: int]}
[membership] : {[member: (members), library: (libraries)]}
[authors] : {[aid: (persons), salary: int]}
[books] : {[bid: int, author: (authors), title: varchar, year: int, genre: varchar]}
[borrow] : {[member: (members), library: (libraries), book: (books), borrow_date: date, due_date: date]}
[reserve] : {[member: (members), library: (libraries), book: (books), reservation_date: date]}

(a) Translate the following relational algebra expressions into SQL queries without subqueries:

1. $\pi_{\text{title}}\ ((\sigma_{\text{year} > 1989}\ \text{books}) \bowtie_{\text{author} = \text{aid}}\ (\sigma_{\text{salary} \neq 1500}\ \text{authors}))$

2. $R := (\sigma_{\text{borrow\_date} = \text{'16.05.2024'}}\ \text{borrow}) \bowtie_{\text{book} = \text{bid}}\ \text{books}$
   $\pi_{\text{genre}}\ (\sigma_{\text{count}(*) > 10}\ (\gamma_{\text{genre,count}(*)}\ R))$

(b) Translate the following SQL queries into relational algebra expressions. If possible, replace Cartesian products with joins:

1.
```
SELECT   DISTINCT *
FROM     membership, members, libraries
WHERE    late_fees = 0
         AND city <> 'Saarbrücken'
         AND member = mid
         AND library = lid;
```

2.
```
SELECT   DISTINCT AVG(late_fees)
FROM     members, authors, reserve
WHERE    salary = 2000
         AND favorite_author = aid
         AND member = mid;
GROUP BY reservation_date
HAVING   SUM(late_fees) = 100;
```

2 Natural Language and SQL (6 points)

Consider the following relational schemas.

[persons] : {[id: int, firstname: varchar, surname: varchar, age: int]}
[cars] : {[id: int, manufacturer: varchar, horsepower: int, model: varchar]}
[mechanics] : {[mid: (persons), city: varchar, salary: int]}
[repair] : {[mechanic: (mechanics), car: (cars)
start: timestamp, end: timestamp, reason: varchar]}
[buy] : {[person: (persons), car: (cars), date_of_purchase: timestamp]}

(a) Translate the following natural language expressions into SQL queries without subqueries. Your outputs should not contain any duplicates:

1. The id of cars that were purchased before 2015 but have never been repaired due to a broken clutch.
2. The id of mechanics that repaired cars more than 5 times, where each repair took less than 6 months and each car has 250 horsepower.

Hint: To read the year from a date you can use the ***`YEAR(date)`*** *function and to compare two dates on the span in months you can use the* ***`DATEDIFF(month, start, end)`*** *function. Here, 'date', 'start', and 'end' are arbitrary dates.*

(b) Translate the following SQL queries into natural language:

1.
```sql
SELECT   DISTINCT m.city, m.salary
FROM     mechanics AS m
         JOIN repair AS r
              ON m.mid = r.mechanic
         JOIN car AS c
              ON r.car = c.id
WHERE    c.manufacturer = 'VW'
         AND c.model LIKE 'Golf GT_'
GROUP BY m.mid
HAVING   COUNT(*) > 1;
```

2.
```sql
SELECT   DISTINCT firstname, surname, age
FROM     mechanic
         JOIN persons
              ON mid = id
WHERE    salary > 4000 AND surname <> 'Smith'
ORDER BY age DESC
LIMIT    10;
```

3 SQL Debugging (3 points)

In the following SQL queries, errors - in the sense of violations of the syntax and semantics formally introduced in the lecture and of the SQL standard - crept in. Name them and briefly describe how they can be fixed (the query results do not necessarily have to remain the same after fixing the queries). The queries are based on the PhotoDB schema known from the lecture and the SQL.ipynb notebook.

[persons] : {[id: int, lastname: string, firstname: string, birthday: string]}
[employees] : {[personId:(persons), salary: int, experience: int]}
[seniors] : {[employeeId:(employees), numGreyHairs: int, bonus: int]}
[salespersons] : {[employeeId:(employees), areaOfExpertise:string]}
[photographers] : {[employeeId:(employees)]}
[cameras] : {[id: int, brand: string, model: string]}
[photos] : {[id: int, location: string, unix_time: int,
photographerId:(photographers), cameraId:(cameras)]}

1.
```
SELECT   NULL AS NOT_NULL, location
FROM     cameras AS c JOIN photos AS p ON c.id = p.id
GROUP BY location
WHERE    location LIKE '%bruecken';
```

2.
```
SELECT   SUM(numGreyHairs) AS measureOfAge
FROM     seniors
WHERE    experience > 42
GROUP BY measureOfAge;
```

3.
```
SELECT   salary
FROM     employees
HAVING   SUM(experience) > 5;
```

4 PhotoDB and SQL (5 points)

The following relational schemas based on the PhotoDB known from the lecture and the corresponding SQL.ipynb notebook are given.

[persons] : {[id: int, lastname: string, firstname: string, birthday: string]}
[employees] : {[personId:(persons), salary: int, experience: int]}
[seniors] : {[employeeId:(employees), numGreyHairs: int, bonus: int]}
[salespersons] : {[employeeId:(employees), areaOfExpertise:string]}
[photographers] : {[employeeId:(employees)]}
[cameras] : {[id: int, brand: string, model: string]}
[photos] : {[id: int, location: string, unix_time: int,
photographerId:(photographers), cameraId:(cameras)]}

Your job is to translate the following natural language queries into appropriate SQL queries. You may use views and subqueries. You can test your queries by pasting them in the designated place in the enclosed *assignment03.ipynb* notebook. Please make sure to clearly separate both queries.

1. The bonus of each senior photographer, that has taken more than two photos with a camera of the brand 'Nikon'. (2 points)
2. The brands of the cameras that were used by a photographer that earns a higher salary than the average of all employees. Sort your output in descending order based on the brand and only return the first 2 tuples. (3 points)

Submission

Solutions must be submitted in teams of 3 to 4 students by May, 23 2024, 10:00 a.m. via your personal status page in CMS using the Team Groupings functionality. Late submissions will not be graded!

Please note that there are two submissions, under **Theoratical** you submit your solutions to exercises 1, 2 and 3 as a PDF file.

Your solution to exercise 4 must be handed in under **Practical** as txt file.

Make sure that you only copy the complete Jupyter cells you want to add and that indentation and formatting are correct.