Database Tutorial 3: Relational Algebra and SQL

November 24, 2017

1 Repetition

1.1 Relational Algebra

$\sigma_C(R)$ Select matching condition C which is a boolean expression

$\pi_A(R)$ Project with attribute list A, can be used for renaming and some computations

$\rho_S(R)$ Rename according to schema $S = \{S_1 \to T_1 \dots S_n \to T_n\}$

$\gamma_{X,G}(R)$ Group according to X and aggregate with G, usually involving renaming

$\tau_A(R)$ Sort according to attributes A

Set operations Union, Intersection, Difference, Cross Product

Joins Natural Join ($\bowtie$)[1], Conditional Join ($\bowtie_C$), Division ($\div$)[2]

1.2 SQL

CREATE TABLE name (attributes);

INSERT INTO table (attributes) VALUES (values);

UPDATE table SET name=value WHERE condition;

DELETE FROM table WHERE condition;

SELECT attributes FROM table;

Set Operations UNION, INTERSECT, EXCEPT

1.3 Translation

```
SELECT   X, G
FROM     T
WHERE    C
GROUP BY Y
HAVING   D
ORDER BY Z;
```

$$\tau_Z(\pi_{X,G}(\sigma_D(\gamma_{Y,G}(\sigma_C(T)))))$$

[1] $R \bowtie S = \{r \cup s \mid r \in R \wedge s \in S \wedge Fun(r \cup s)\}$ with predicate Fun(t) true if t is a function

[2] $R \div S = \{t[a_1 \dots a_n] \mid t \in R \wedge \forall s \in S(t[a_i \dots a_n] \cup s \in R)\}$ with $t[a_1 \dots a_n]$ subset of all attributes in R

2 Exercises

1. (8 points) (a) (4 points) Given the relation

   `Planets(star, name, position, distance, mass, atmosphere, oxygen, water)`

   write a relational algebra query that returns, for each star with more than 5 planets, the total combined mass of all planets with an atmosphere. The query should return tuples of the form `(star, totalMass)`

   Solution:

   In relational Algebra

   $\pi_{star,totalmass}(\sigma_{atmosphere\&planetcount>5}$
   $(\gamma_{star,COUNT(*)\rightarrow planetcount,SUM(mass)\rightarrow totalmass}(Planets)))$

   In SQL

   ```
   SELECT Star,SUM(Mass) AS TotalMass FROM
   Planets WHERE Atmosphere GROUP BY Star HAVING COUNT(*) > 5 ;
   ```

   This solution has several problems.

   - In relational algebra is the result of the aggregation because you first sum and afterwards try to filter.
   - In the SQL that the condition in the WHERE and the HAVING are evaluated in the wrong order.

   Alternative Solution:

   In relational Algebra

   $\pi_{star,totalMass}((\sigma_{planetcount>5}(\gamma_{star,COUNT(*)\rightarrow planetcount}(Planets))) \bowtie$
   $(\gamma_{star,SUM(mass)\rightarrow totalMass}(\sigma_{atmosphere}(Planets))))$

   In SQL

   ```
   SELECT Star,TotalMass FROM -- π_{star,totalMass}
   -- γ_{star,COUNT(*)→planetcount}(Planets)
   (SELECT Star AS PlanetCount FROM Planets
   GROUP BY Star HAVING COUNT(*) > 5) AS P
   NATURAL JOIN -- ⋈
   -- γ_{star,SUM(mass)→totalMass}
   (SELECT Star, SUM(Mass) as totalMass FROM
     (SELECT * FROM Planets WHERE Atmosphere) AS Q -- σ_{atmosphere}(Planets)
   GROUP BY Star)
   ```

   (b) (4 points) Given the relations

   `P(star, position, distance, mass, atmosphere, oxygen, water)` and

   `G(star, position, gravity)`,

   translate the following relational algebra query to SQL:

   $\tau_{maxg}(\pi_{position,maxg}(\gamma_{position,AVG(mass)\rightarrow avgm,MAX(gravity)\rightarrow maxg}(P \bowtie G)))$

   Solution:

   ```
   SELECT Position, MAX(Gravity) AS MaxG
   FROM P NATURAL JOIN G GROUP BY Position
   ORDER BY MaxG;

   -- More verbose
   SELECT Position, MaxG -- π_{position,maxg}
   ```

```
-- $\gamma_{position,AVG(mass)\rightarrow avgm,MAX(gravity)\rightarrow maxg}$
(SELECT Position, AVG(Mass) AS AvgM, MAX(Gravity) AS MaxG FROM
P NATURAL JOIN G) -- $P \bowtie G$
ORDER BY MaxG; -- $\tau_{maxg}$
```

2. (8 points) A multi-national company uses a relational database to manage information about its offices in different cities, and its employees. This database has the following relations:

```
Offices(city, supplement)
Departments(city, dname, departmentHead)
Employees(empId, name, salary, dept, city)
```

The company has one office in each city, and several departments can be located at each office. Attribute supplement is the monthly salary supplement that each employee working at that office receives (e.g. employees at the London office might receive a supplement of 1000 SEK per month to cover higher living costs in London). The default city supplement is 0 SEK. Attribute dname describes the departments function (e.g. sales or personnel). Attribute departmentHead is the employee identifier of the head of the department. Employee identifiers (empId) are unique. Attribute salary is an employees basic monthly salary. The total monthly salary for an employee can be calculated by adding the city supplement to the employees basic monthly salary.

(a) (4 points) Write a relational algebra expression that finds the employee identifier, name and total monthly salary of all employees (recall that the total monthly salary for an employee can be calculated by adding the city supplement to the employees basic monthly salary). The results should be sorted by employee name.

Solution:
In Relational Algebra:
$\tau_{name}(\pi_{empId,name,salary+suplement\rightarrow totalSalary}(Employees \bowtie Offices)$
In SQL

```
SELECT empId,name,salary+suplement AS totalSalary FROM
Employees NATURAL JOIN Offices
ORDER BY name;
```

(b) (4 points) Write a relational algebra expression that finds the names of cities where there is a sales department (named "sales") and, for each of these departments, the average basic salary of the employees in that department. You can assume that every department has at least one employee.

Solution:
In Relational Algebra:
$\pi_{city,avgsalary}(\gamma_{city,AVG(salary)\rightarrow avgsalary}(\sigma_{dept=\text{"}sales\text{"}}(Employees)))$
In SQL

```
SELECT City,AVG(Salary) AS AvgSalary FROM Employees WHERE Dept = 'Sales'
GROUP BY City;
```

Consider the relation `Planets(star, name, distance, mass, atmosphere, oxygen, water)`.

3. (12 points) (a) (4 points) Write an SQL table definition with reasonable types and constraints. Store distance in millions of km (For Earth, you would store the value 149.6)

Solution:

```
CREATE TABLE Planets
(
  Star TEXT,
  Name TEXT,
  Distance REAL NOT NULL CHECK (Distance > 0),
  Mass REAL NOT NULL CHECK (Mass > 0),
  Atmosphere BOOLEAN NOT NULL,
  Oxygen SMALLINT NOT NULL CHECK ((Oxygen = 0 AND NOT Atmosphere ) OR (
Atmosphere AND Oxygen >= 0 AND Oxygen <= 100)),
  Water SMALLINT NOT NULL CHECK (Water >= 0 AND Water <= 100),
  PRIMARY KEY (Star,Name),
  UNIQUE (Star,Distance)
  )
```

(b) (4 points) Write an SQL query to determine how many planets are in orbits larger than the orbit of the fictional planet "Duna" of the fictional star "Kerbol"?

Solution:

```
SELECT COUNT (*) FROM Planets WHERE
Distance > ( SELECT Sistance FROM Planets WHERE
Star = 'Kerbol' AND name = 'Duna') ;
```

(c) (4 points) We define a planet "habitable" if it satisfies all these conditions:

- orbits at a distance (in millions of km) between 100 and 200 (inclusive) from its star,
- has an atmosphere and it has an oxygen percentage between 15% and 25% (inclusive),
- has water on its surface.

Write an SQL query which returns the star and name of a planet, as well as a column status with value ''habitable'' if the planet is habitable, otherwise ''uninhabitable''. (This means, return 3 values per row)

Solution:

```
(SELECT Star, Name, 'habitable' FROM Planets WHERE
  Distance >= 100 AND Distance <= 200 AND
  Atmosphere AND Oxygen >= 15 AND Oxygen <= 25 AND
  Water > 0)
UNION
(SELECT Star, Name, 'uninhabitable' FROM Planets WHERE NOT (
  Distance >= 100 AND Distance <= 200 AND
  Atmosphere AND Oxygen >= 15 AND Oxygen <= 25 AND
  Water > 0) ) ;
```

Alternative Solution:

```
WITH Habitables AS (SELECT Star, Name FROM Planets WHERE
  Distance >= 100 AND Distance <= 200 AND
  Atmosphere AND Oxygen >= 15 AND Oxygen <= 25 AND
  Water > 0)
SELECT Star, Name, 'habitable' FROM Planets WHERE
  (Star, Name) IN (SELECT Star , Name from Habitables)
UNION
SELECT Star, Name, 'unhabitable' FROM Planets WHERE
  (Star, Name) NOT IN (SELECT Star , name from Habitables)
```

4. (10 points) A database system used by a company's personnel department has the following relations:

```
Employees(empId, name, year, salary, entitlement, branch)
ParentalLeave(employee, startDay, startYear, endDay, endYear)
```

Employee identifiers (empId) are unique. Attribute year is the employee's year of birth. Attribute entitlement is the number of annual vacation days to which the employee is entitled. Employees have 30 days of annual vacation entitlement by default. Branch is the name of the city where the branch is located (assume that there is only one branch in each city). The personnel department keeps a record of all periods of parental leave taken by employees. The attributes startDay and endDay are integers in the range 1-366 that represent the day within the year. For each period of parental leave, the start date must be before the end date.

(a) (4 points) Suggest keys and references for these relations. Write SQL statements that create these relations with reasonable constraints in PostgreSQL.

Solution:

```
CREATE TABLE Employees (
  EmpId INT PRIMARY KEY,
  Name TEXT NOT NULL,
  Year INT NOT NULL CHECK (year > 1900),
  Salary FLOAT NOT NULL CHECK (Salary >= 0),
  Entitlement INT NOT NULL DEFAULT 30 CHECK (Entitlement >= 0),
  Branch TEXT NOT NULL
);

CREATE TABLE ParentalLeave (
  Employee INT REFERENCES Employees,
  StartDay INT NOT NULL CHECK (startDay >= 0 AND startDay <= 366),
  StartYear INT NOT NULL CHECK (year > 1900),
  EndDay INT NOT NULL CHECK (( endYear = startYear AND startDay <= endDay ) OR
  EndYear > startYear ) ,
  EndYear INT NOT NULL CHECK ( endYear >= startYear )
);
```

(b) (3 points) Show an SQL query to get the amount of employees per branch that have/had parental leave spanning a period in two different calendar years (this means startYear and endYear are not the same). Example output row if the Stockholm branch has had 3 such employees: ('Stockholm', 3).

Solution:

```
SELECT Branch , COUNT (*)
  FROM Employees e JOIN ParentalLeave p ON e.EmpId = p.Employee
    WHERE p.StartYear <> p.EndYear
    GROUP BY e.Branch ;
```

(c) (3 points) Show an SQL query to get the branch name and average of salaries in that branch, for those branches that only have employees born after 1987. Sort the output by average salary. Do not show information about branches that have employees born in or before 1987.

Solution:

```
SELECT Branch, AVG (Salary) AS AvgSalary
  FROM Employees
  GROUP BY Branch
  HAVING MIN (Year) >= 1987
  ORDER BY AvgSalary ;
```