# Database Exercise I

TDA357/DIT60
SQL Exercises

2019-01-30

**Exercise 1: Basics.** Based on the *Departments* table below, create a **SQL** table called *Employees* that stores employee number, employee name, department and salary information. The `primary key` for the *Employees* table should be the employee number. Create a `foreign key` on the *Employees* table that references the *Departments* table based on the `department_id` field.

```
CREATE TABLE Departments (
  department_id INT NOT NULL,
  department_name TEXT NOT NULL,
  -- Here we use the explicit constraint form for primary keys
  CONSTRAINT department_pk PRIMARY KEY (department_id)
);
```

**Solution to Exercise 1.** Is satisfied by the following table:

```
CREATE TABLE Employees (
  employee_number INT NOT NULL,
  employee_name TEXT NOT NULL,
  employee_department INT NOT NULL,
  employee_salary INT NOT NULL,
  CONSTRAINT employee_pk PRIMARY KEY (employee_number),
  CONSTRAINT department_member FOREIGN KEY (employee_department)
                               REFERENCES Departments (department_id));
```

**Exercise 2: Suppliers.** Consider the following relational schema:

`Suppliers(`**sid:integer**`, sname:string, city:string, street:string)`
`Parts(`**pid:integer**`, pname:string,color:string)`
`Catalog(`**sid:integer**`,`**pid:integer**`,cost:real)`

Find the names of all suppliers who have supplied a non-blue part.

**Solution to Exercise 2.** Assuming we encode the information in the following tables:

```
CREATE TABLE Suppliers (
  sid INT PRIMARY KEY,
  sname TEXT NOT NULL,
  city TEXT NOT NULL,
  street TEXT NOT NULL);

CREATE TABLE Parts (
  pid INT PRIMARY KEY,
  pname TEXT NOT NULL,
  color TEXT NOT NULL,
  CONSTRAINT color_name CHECK (color in ('red', 'blue', 'green')));

CREATE TABLE Catalog (
  sid INT NOT NULL REFERENCES Suppliers (sid),
  pid INT NOT NULL REFERENCES Parts (pid),
  cost FLOAT NOT NULL,
  CONSTRAINT catalog_pk PRIMARY KEY (sid,pid));
```

The solution would then be given by:

```
SELECT sname from Suppliers
  where sid IN (SELECT sid from Catalog
                  where pid IN (SELECT pid from Parts where color!='blue'));
```

Or the more idiomatic:

```
SELECT DISTINCT S.sname
FROM Catalog as C, Parts as P, Suppliers as S
WHERE C.pid = P.pid
  AND S.sid = C.sid
  AND color != 'blue';
```

**Exercise 3: Employees.** Consider the following table:

`Employees(`**`empId`**`, name, department, salary)`

The columns `empId` and `name` are of type `text`, while `department` and `salary` are of type integer.

a) Write a query to find the employees who get a higher salary than anyone in department 5 (you may assume that department 5 has more than one employee).

b) Write a query to find the maximum salary from each department.

c) Find all employee records containing the word „Joe", regardless of whether it was stored as JOE, Joe or joe.

**Solution to Exercise 3.** We can encode the table in SQL in the following way:

```
CREATE TABLE Employees (
        empId INT PRIMARY KEY,
        name TEXT NOT NULL,
        department INT NOT NULL,
        salary INT NOT NULL);
```

A solution will then be:

```
-- We start by doing b), and then re-use it in a):
-- b) is given by the following:
SELECT department, MAX(salary) from Employees GROUP BY department;

-- By reusing b) we can get a):
-- a)
SELECT name from Employees where salary > (
 SELECT MAX(salary) from Employees where department=5 GROUP BY department);

-- c). Note the use of LOWER, to make sure we catch all variants.
SELECT empId, name from Employees where LOWER(name) LIKE '%joe%';
```

**Exercise 4: Company.** Consider following relational schema:

```
Employees(employeeId, employeeName, street, city)
Companies(companyId, companyName, city)
Works(employee, company, salary)
Manages(manager, employee)
```

The information on which company an employee works for and their current salary is stored in a relation *Works*. Assume that all people work for at most one company. The information on which employees have manager roles and who they manage is stored in the relation *Manages*.

Write a SQL query for the following tasks:

a) Find the names, street address, and cities for all employees who work for 'First Bank Corporation' and earn more than $10,000.

b) Find the names of all employees who:

   i) live in the same city as the company they work for.

   ii) live on the same street in the same city as their manager.

   iii) earn more than every employee of 'Small Bank Corporation'

c) Find the name of the company that has the smallest payroll (i.e. the smallest total sum of salaries).

d) Assuming that the `Employee` relation had an email column with `NULL` values, write a query to update the email values.

e) Find all employees who are not managers.

**Solution to Exercise 4.** Assume that we've encoded the tables in the following way:

```
CREATE TABLE Employees (
   employeeId INT PRIMARY KEY,
   employeeName TEXT NOT NULL,
   street TEXT NOT NULL,
   city TEXT NOT NULL);

CREATE TABLE Companies (
   companyId INT PRIMARY KEY,
   companyName TEXT NOT NULL,
   city TEXT NOT NULL);

CREATE TABLE Manages (
  manager INT NOT NULL REFERENCES Employees (employeeId),
  employee INT NOT NULL REFERENCES Employees (employeeId),
  PRIMARY KEY (manager, employee));
```

The solutions are then given by:

```
-- a)
SELECT employeeName, street, Employees.city
FROM Employees, Works, Companies
WHERE employeeId = employee
  AND company = companyId
  AND companyName = 'First Bank Corporation'
  AND salary > 10000;

-- Or using subqueries:
SELECT employeeName, street, city
 FROM Employees
 WHERE employeeId in (SELECT employee FROM Works
   WHERE company=(SELECT companyId FROM companies
                  WHERE companyName='First Bank Corporation')
     AND salary >= 10000);
```

```sql
-- b)
-- i)
SELECT employeeName
FROM Employees as E, Works as W, Companies as C
WHERE E.employeeId = W.employee
 AND C.companyId = W.company
 AND E.city = C.city;

-- ii)
SELECT EE.employeeName
FROM Employees as EE, Employees as EM, Manages as M
WHERE EM.employeeId = M.manager
  AND EE.employeeId = M.employee
  AND EE.city = EM.city
  AND EE.street = EM.street;

-- iii)
SELECT employeeName
FROM Employees, Works
WHERE employeeId = employee
  AND salary > ( SELECT MAX(salary)
                 FROM Companies, Works
                 WHERE companyId = company
                   AND companyName = 'Small Bank Corporation'
                 GROUP BY company);

-- c)
SELECT companyName FROM Companies
WHERE companyId = (SELECT company as c
                                  FROM Works
                                  GROUP BY company
                   ORDER BY SUM(salary) LIMIT 1);

-- Or by using Having:

SELECT companyName
FROM Companies as C, (SELECT company FROM Works as W
                      GROUP BY company
                      HAVING SUM(salary) <= ALL (SELECT SUM(salary)
                                                 FROM Works
                                                 GROUP BY company)
                     ) AS TEMP
WHERE C.companyId = TEMP.company;

-- d)
```

```sql
UPDATE Employee
SET employeeEmail = <insert email here>
WHERE employeeId = <insert id here>

-- e)
SELECT employeeName FROM Employees
WHERE employeeId NOT IN (SELECT manager from Manages);
```

**Exercise 5: Hospital.** A database system used by a hospital to record information about patients and wards has the following relational schema:

```
Wards(number, numBeds)
Patients(pid, name, year, sex)
PatientInWard(pid, ward)
Tests(patient, testDate, testHour, temperature, heartRate)
```

A ward is identified by its number. The `numBeds` attribute is the number of beds in that ward. Patients are identified by their personal identification number. The name, year of birth and biological sex ('M' or 'F') of each patient is stored in the *Patients* relation. The ward to which each patient is assigned is stored in the *PatientInWard* relation.

During their stay in the hospital, patients will undergo routine tests. The date, hour, and measurements (temperature and heart rate) of each test is performed is recorded, and stored in the database. A patient will normally undergo multiple tests over the course of their hospitalization.

Implement the schema in SQL with the appropriate constraints and types, and do the following:

a) Write a SQL query that finds the temperature and heart rate measured in each test carried out on patients born before 1950.

b) Create a view, `FreeBeds(ward, numBeds)` where `ward` is a ward number, and `numBeds` is the number of available beds in that ward.

**Solution to Exercise 4.** Assuming we encode the tables in the following way:

```sql
CREATE TABLE Wards (
  number INT PRIMARY KEY,
  numBeds INT NOT NULL);

CREATE TABLE Patients (
  pid VARCHAR(10) PRIMARY KEY,
  name TEXT NOT NULL,
  year INT NOT NULL,
  bsex VARCHAR(1),
  CONSTRAINT bsex_check CHECK (bsex in ('M', 'F')));
```

```
CREATE TABLE PatientInWard (
  pid VARCHAR(10) NOT NULL REFERENCES Patients(pid),
  ward INT NOT NULL REFERENCES Wards(number),
  PRIMARY KEY (pid, ward));

CREATE TABLE Tests (
  -- ID numbers are 10 digits.
  patient VARCHAR(10) NOT NULL REFERENCES Patients(pid),
   -- Note the use of DATE and TIME
  testDate DATE NOT NULL,
  testHour TIME NOT NULL,
  temperature FLOAT NOT NULL,
  heartRate INT NOT NUll);
```

A solution is then given by:

```
-- a)
SELECT temperature, heartRate FROM Tests
WHERE patient IN (SELECT pid FROM Patients where year < 1950);

-- b)
CREATE VIEW FreeBeds (ward, numBeds) AS (
  SELECT ward, (numbeds - COUNT(*))
  FROM PatientInWard, Wards
  WHERE ward=number
  GROUP BY (ward, numbeds));
```

**Exercise 6: Planets.** Consider the relation:

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

We assume that all stars have different names, and that planet names are only unique within their star system. For simplicity, we also assume that each star system has exactly one star, and that all planets have circular orbits around their star at different distances. A planet's position indicates its order within the star system, e.g. Earth is the 3rd planet around the Sun, after Mercury and Venus. If a planet has oxygen or other gases, it has an atmosphere. If it does not have an atmosphere, it has no gases. The surface of a planet is either all water, all land, or somewhere in-between.

Do the following:

a) Implement the relation in a SQL table, with reasonable types and constraints. Store distances in millions of km, e.g. for Earth the value should be 149.6.

b) Assume that the relation contains the planet ,,Duna" in orbit around a star called ,,Kerbol". Write a SQL query to find how many planets are in orbits around their star that are larger than Duna's orbit around Kerbol.

c) We say that a planet is ,,habitable" if it satisfies the following conditions:

i. it orbits at a distance between 100 and 200 million kilometers (inclusive) of its star (i.e. in the ,,Goldilocks zone"),

ii. it has has an atmosphere, and an oxygen percentage between 15 and 25% (inclusive),

iii. it has water on its surface.

Create the view `Habitable(name, star, status)`, where `status` is 'habitable' if the planet `name` around `star` is habitable and 'uninhabitable' if the planet is uninhabitable.

**Solution to Exercise 6.** We begin by encoding the relation in a SQL table:

```
-- a)
CREATE TABLE Planets (
  star TEXT NOT NULL,
  name TEXT NOT NULL,
  distance FLOAT NOT NULL CHECK (distance > 0),
  mass FLOAT NOT NULL CHECK (mass > 0),
  atmosphere BOOLEAN NOT NULL,
  oxygen INT NOT NULL
  -- We give the checks names, so the programmer knows
  -- which of them failed.
  CONSTRAINT oxygen_percent CHECK (oxygen >= 0 AND oxygen <= 100),
  CONSTRAINT oxygen_atmos CHECK ((atmosphere AND oxygen > 0)
                              OR ((NOT atmosphere) AND oxygen = 0)),
  water FLOAT NOT NULL
  CONSTRAINT water_check CHECK ((water >= 0 AND water <= 100)),
  PRIMARY KEY (star, name),
  UNIQUE (star, distance));

--b)
SELECT COUNT(*) FROM Planets
WHERE distance > (SELECT distance FROM Planets
                  WHERE star='Kerbol' AND name='Duna');

-- c)
-- Here we use a helper view to make our lives easier.
CREATE VIEW HabitableHelper (name,star) AS (
 SELECT name, star FROM Planets
  WHERE  distance >= 100 AND  distance <= 200
    AND  atmosphere
    AND  oxygen >= 15 AND  oxygen <= 25
    AND  water > 0);
```

```sql
CREATE VIEW Habitable (name, star, status) AS (
 SELECT name, star, 'habitable' FROM HabitableHelper
 UNION
 SELECT name, star, 'uninhabitable' FROM Planets
 WHERE (name, star) NOT IN (SELECT * FROM HabitableHelper)
);
```