**COMP 353 : Assignment 3**

**Presented to :**

Professor Khaled Jababo

**Presented by :**

Jordan Hubscher, ID : 27019696

Brandon Mendez, ID : 27003606

Anthony Mendez, ID : 26998380

**Question 1**

Consider a database scheme consisting of the following relation schemes:

EMPLOYEE (eid, ename, age, salary)

DEPARTMENT(did, budget, manager\_id)

WORKS (eid, did, hours\_per\_week)

The semantics of the scheme should be clear from the identifiers used. The key attributes are underlined. Department manager\_id is a subset of employee.eid. An employee may manage more than one department.

**1**. Give an example of a foreign key constraint that involves the DEPARTMENT relation. What are the options for enforcing this constraint when a user attempts to delete a DEPARTMENT tuple?

**Answer**:

The did field of Works should be a foreign key that refers to the did of Department.

CREATE TABLE Works ( eid INTEGER NOT NULL ,

did INTEGER NOT NULL ,

hours\_per\_week INTEGER,

PRIMARY KEY (eid, did),

UNIQUE (eid),

FOREIGN KEY (did) REFERENCES Dept )

When a user deletes a Department tuple, there are several different options that can be chosen.

- Also delete all Works tuples that refer to it.

- Disallow the deletion of the Department tuple if some Works tuple refers to it.

- For every Works tuple that refers to it, set the did field to the did of some existing, default department.

- For every Works tuple that refers to it, set the did field to null.

**2.** Now, express the following queries in Relational Algebra:

**a**. Get the id’s and names of employees who have a salary more than their manager.

X = πmanager\_id, did Department

Y = πeid (Works ⋈works.did = X.did X)

πdid, ename (σEmployee.salary > X.salary (Employee ⋈ eid = Y.eidY)

**b**. Get details on employees who work in more than one department.

PEmployee1(eid1, did1)( πeid, did (Works))

PEmployee2(eid2, did2)(Employee1)

πeid, ename, age, salary(σeid1 = eid2 ^ did1 < > did2 (Employee1 x Employee2)

**c**. Who are the managers who also work in a department that they do not manage.

X = πmanager\_id, did Department

Y = πeid (Works ⋈Works.eid = X.manager\_id X)

πeid (σEmployee)

**d**. Get the id’s of employees who work in department 203 and also in department 302.

πeid (Employee) ⋈ Employee.eid = works.eid πeid(σdid = 203, Works) ∩

πeid (Employee) ⋈ Employee.eid = works.eid πeid(σdid = 302, Works)

**e**. Find those employees who do not work in the department managed by John Doe.

X = (σname = "John Doe", Employee)

Y = (πdid (Department ⋈ manager\_id < > eid X))

Z = (πeid (Works ⋈ works.did = department.did Y))

Employee ⋈ Employee.eid = works.eid Z

**f**. Find details on the employee(s) who work(s) the minimum number of hours per week in department 202.

P(R1, Works)

P(R2, Works)

Employee ⋈ (πhours\_per\_week (Works) - (πhours\_per\_week(σdid = “202” (R1.hours\_per\_week > R2.hours\_per\_week)(R1xR2)))

**g**. Get the name of the manager that manages the department with the highest budget.  
 P(Dept1(did1, budget1), Department)  
 P(Dept2(did2, budget2), Department)

πename(Employee ⋈eid=manager\_id((πmanager\_idDepartment) - (πmanager\_id(σbidget1 < budget2 ^ did1 < > did2  (Dept1 x Dept2))))

**Question 2**

Consider the following relation schemes in a database. The key attribute(s) of each scheme are underlined.

* Manufacturer (mid, mname, city)
* Produces (p#, mid, price)
* Distributor (did, dname, city)
* Supplies (did, p#, price)
* Orders (order#, cid, city, p#, did, qty, date)

The manufacturer relation contains the id, name and city of the manufacturer.

The produces relation stores information on the parts, the manufacturer and the manufacturer’s price.

The distributor and supplier relations carry similar information on distributors and the parts supplied by them.

Finally the orders relation tells us the details on each order: the id of the customer making the order, the customer’s city, the part ordered, the distributor supplying the part, the number of units of the part ordered and the date of the order.

Write the following queries in the query languages indicated in the brackets after the query (RA: Relational Algebra, TRC: Tuple Relational Calculus):

**a.** List details of all manufacturers in Montreal (TRC).

TRC :

{ m | m (mManufacturerm[city] = ‘Montreal’) }

**b.** List details on distributors who supply parts made by ‘Bombardier’ (RA, TRC).

RA :

X = πmid, did(σmname = ‘Bombardier’, Manufacturer)

Y = πp# (Produces ⋈ Produces.mid = X.mid X)

Z = πdid (Supplies ⋈ Supplies.p# = Y.p# Y)

πdid (Distributor ⋈ Distributor.did = Z.did Z)

TRC :

{d | m (mManufacturerm[mname] = ‘Bombardier’)p (pProducesp[mid] = m[mid])s (sSuppliess[p#] = p[p#])d (dDistributord[did] = s[did]) }

**c.** Find manufacturers who do not produce any parts supplied by ‘Rolls Royce’ (RA).

RA :

X = πmid, mname(σmname = ‘Rolls Royce’, Manufacturer)

Y = πmid, p#(Produces ⋈ Produces.mid = X.mid X)

Z = πmid(σmidY.mid,Produces)

πmid(σZ.midY.mid,Manufacturer)

**d.** Get those customers who have ordered only parts that are supplied and manufactured by companies located in the same city that they (ie, the customers) are located in (RA).

RA :

**e.** Find the distributor name, the part#, the purchase price (Price of the Manufacturer) and the selling price (Price of the distributor) for all parts that are sold at a price less that they are purchased. (RA).

**f.** Find details of distributors who supply all the parts produced by the manufacturer ‘Airbus’ (RA, TRC).

**g.** Find details of distributors who have never had orders for clients in the same city that

they are located (In other words all the orders that the distributor had are for clients located in a different city than the distributor). (RA).

**h.** Give Client Ids (cid) and part# of clients who have placed more than one order for the same part on the same day (RA, TRC).