# **Student ID: 300363269 First Name: Christopher Last Name: Straight**

## School of Engineering and Computer Science

### SWEN 304 **Database System Engineering**

### **Assignment 2**

Due: 23:59, Friday, 30 April 2021

The objective of this assignment is to test your understanding of Relational Algebra and Query Processing and Optimization. It is worth 10% of your final grade. The Assignment is marked out of 100.

In Appendix 1, you will find short recapitulation of formulae needed for cost-based optimization. Appendix 2 contains an abbreviated instruction for using PostgreSQL.

**Submission Instructions:**

* Please submit your project in **pdf** with your **student ID** and **Name** via the submission system.
* Submissions not in **pdf** will incur **3 marks** deduction from the total marks.

**Question 1. Relational Algebra [40 marks]**

Consider the Suppliers database schema given below.

Set of relation schemas:

*Products* ({*PId*, *Description, Category*}, {*PId*}),

*Company* ({*CId*, *Name*, *Phone, Location*}, {*CId*})

*Supplied\_By* ({*PId*, *CId, Amount, Year, Price*}, {*PId* + *CId + Year*})

Set of referential integrity constraints:

*Supplied\_By* [*PId*] ⊆ *Products* [*PId*],

*Supplied\_By* [*CId*] ⊆ *Company* [*CId*]

In this question, you will be given queries on the Suppliers database above in two ways. Firstly, queries are given in plain English and you must answer them in Relational Algebra. Secondly, queries are given in Relational Algebra and you must answer them in plain English and in SQL. Submit all your answers in printed form.

1. **[25 marks]** Translate the following query into Relational Algebra:
2. [5 marks] Retrieve the names of all companies who always supply products of category ‘food’.  
   π Name (σ Category = 'food' (Products \* Supplied\_By) \* Company)
3. [5 marks] Retrieve the descriptions of all products that are supplied by two or more companies.  
   π Description (σ numS ≥ 2 (Products \* (σ Products.PId = C.PId ρ C (π PId, numS γ PId; COUNT(CId)→numS (Supplied\_By)))))
4. [5 marks] For all products of category ‘food’ list their descriptions and the names of their supplying companies.  
   π Description, Name ((σ Category = 'food' (Products))\* Supplied\_By \* Company)
5. [5 marks] Retrieve the names of companies who have **not** supplied any product in 2021.  
   π Name (Company ▷ (σ Year = 2021 Supplied\_By))
6. [5 marks] Retrieve the description of products that have been supplied by companies in Wellington who **always** supply products with price higher than $50.00.

π Description ((σ Location = 'Wellington' (Company ▷ (σ Price <= 50 Supplied\_By))) \* Supplied\_By \* Products)

1. **[15 marks]** Translate the following queries into plain English and into SQL:
2. π*name, Description* (σ*price<10* (*Products* \* (*Supplied\_By* \* *Company* ) ))  
   Retrieve the description of every product with price less than 10 and the name of every company that supplies it at that price.

select Name, Description  
from Products natural join  
 (select \* from Supplied\_By natural join Company) as SBC  
where Price < 10;

1. π*Name ,Phone* (*Products* \* (σ*Amount>1000* (*Supplied\_By*) \* *Company*))  
   Retrieve the name and phone number of every company that has supplied more than 1000 of a product in one year.

select Name, Phone  
from Products natural join  
 (select \* from Supplied\_By natural join Company  
 where Amount > 1000) as SBC;

1. π*CId* (σ*Amount>100* (*Supplied\_By*)) ∩ π*CId*(*Supplied\_By* \* (σ*Description=’Cake’* (*Products*)))

Retrieve the ID of every company the has supplied more than 100 of any product and has supplied cake

select SBPCI.CId  
from  
 (select CId from Supplied\_By where Amount > 100) as SBCI  
 ,  
 (select CId from Supplied\_By natural join Products  
 where Description = 'Cake') as SBPCI  
where SBCI.CId = SBPCI.CId;

**Question 2. Heuristic and Cost-Based Query Optimization [40 marks]**

The DDL description of a part of the University database schema is given below.

CREATE DOMAIN StudIdDomain AS int NOT NULL CHECK (VALUE >= 30000000 AND VALUE <= 300099999);

CREATE DOMAIN CharDomain AS char(15) NOT NULL;

CREATE DOMAIN NumDomain AS smallint NOT NULL CHECK (VALUE BETWEEN 0 AND 10000);

CREATE TABLE Student (

StudentId StudIdDomain PRIMARY KEY,

Name CharDomain,

NoOfPts NumDomain CHECK (NoOfPts < 1000),

Tutor StudIdDomain REFERENCES Student(StudentId)

);

CREATE TABLE Course (

CourseId CharDomain PRIMARY KEY,

CourName CharDomain,

ClassRep StudIdDomain REFERENCES Student(StudentId)

);

CREATE TABLE Enrolled (

StudentId StudentIdDomain REFERENCES Student,

CourseId CharDomain REFERENCES Course,

Term NumDomain CHECK(Term BETWEEN 2000 AND 2100),

Grade CharDomain CHECK (Grade IN (‘A+’, ‘A’, ‘A-‘, ‘B+’, ‘B’, ‘B-‘, ‘C+’, ‘C’)),

PRIMARY KEY (StudentId, CourseId, Term)

);

The corresponding EER is shown below (for your reference):



1. **[20 marks] Heuristic query optimization**
2. **[5 marks]** Transfer the following query into Relational Algebra.

SELECT StudentId, Name, Grade

FROM Student NATURAL JOIN Enrolled NATURAL JOIN Course

WHERE CourName = ‘Database Systems’ AND Term = 2020   
 AND NoOfPts > 360;

πStudentId, Name, Grade ( σCourName = ‘Database Systems’ ˄ Term = 2020 ((Student \* Enrolled) \* Course)))

**[3 marks]** Draw a query tree for the relational algebra query from **1**).

**ANSWER**

πStudentId, Name, Grade

σCourName = ‘Database Systems’ ˄ Term = 2020

\*

Course

\*

Student

Enrolled

1. **[12 marks]** Transfer the following query treeinto an optimized query tree using the query optimization heuristics.

****

**ANSWER**

πStudentId, Name, Tutor

\*

\*

σNoOfPts>160

σClassRep = “Mark”

σGrade = ‘A’

Student

Course

Enrolled

* 1. **[20 marks] Query cost calculation**

Suppose the following:

* The *Student* relation contains data about *ns* = *50000* students (enrolled during the past *10* years),
* The *Course* relation contains data about *nc* = *1000* courses,
* *The Enrolled relation contains data about ne = 400,000 enrollments,*
* *All data distributions are uniform (i.e. each year approximately the same number of students enrolls into each course),*
* *The intermediate results of the query evaluation are materialized,*
* *The final result of the query is materialized.*

**Note:** If you feel that some information is missing, please make a reasonable assumption and make you assumption explicit in your answer.

For each of the given two queries below draw a query tree and calculate the cost of executing query.

πStudentId, Name, Grade (σterm = 2019 ˄ CourseId =‘SWEN304’ (Student \* Enrolled ) )

**ANSWER**

Ci = 0

S3 = (1 – Ci) \* S2 \* (SStudentIdDomain + SCharDomain + SCharDomain) / (rStudent + rEnrolled - SStudentIdDomain)

= 254.8

\*

πStudentId, Name, Grade

σterm = 2019 ˄ CourseId=’SWEN304’

αC = (1/num years on record \* 1/ nc) / 100 = 0.000001

S2 = αC \* S1 = 2942.4

p = 1 / ns = 0.00002

S1 = SStudent / rStudent \* p \* SEnrolled / rEnrolled \* (rStudent + rEnrolled SStudentIdDomain)

= 2942400000

Total = SStudent + SEnrolled + S1 + S2 + S3

= 49,10,103,197.2

Enrolled

Student

SNumDomain = 16 bits

SEnrolled = ne \* rEnrolled

= 1,955,200,000

SStudent = ns \* rStudent

= 125,000,000

SStudentIdDomain = 32 bits

rEnrolled = SStudentIdDomain + SCharDomain + SNumDomain + SCharDomain

= 4888 bits

SCharDomain = ((15 \* 32 + 4) + (15 \* 8 + 1)) / 2

= 302.5 bytes = 2420 bits

assuming UTF-8 encoding

rStudent = SStudentIdDomain + SCharDomain + SNumDomain + SStudentIdDomain

= 2500 bits

πStudentId, Name, Grade (Student \* σterm = 2019 ˄ CourseId =‘SWEN304’ (Enrolled) )

**ANSWER**

Ci = 0

S3 = (1 – Ci) \* S2 \* (SStudentIdDomain + SCharDomain + SCharDomain) / (rStudent + rEnrolled - SStudentIdDomain)

= 1948.8

πStudentId, Name, Grade

p = 1 / ns = 0.00002

S2 = SStudent / rStudent \* p \* S1 / rEnrolled \* (rStudent + rEnrolled - SStudentIdDomain)

= 2942.4

\*

σterm = 2019 ˄ CourseId=’SWEN304’

Student

Enrolled

αC = (1/num years on record \* 1/ nc) / 100 = 0.000001

S1 = αC \* SEnrolled = 1955.2

SStudent = ns \* rStudent

= 125,000,000

SEnrolled = ne \* rEnrolled

= 1,955,200,000

Total = SStudent + SEnrolled + S1 + S2 + S3

= 2,080,206,846.4

* + 1. Which of the above two trees has a smaller query cost and why?

**ANSWER**

Query 2 is smaller because the number of tuples is cut down before the join rather than after

***Hint:*** *To find out about the sizes of attributes in PostgreSQL please consult the documentation (*[*www.postgresql.org/docs/9.2/static/datatype.html*](http://www.postgresql.org/docs/9.2/static/datatype.html)*) or check this tutorial (*[*www.tutorialspoint.com/postgresql/postgresql\_data\_types.htm*](http://www.tutorialspoint.com/postgresql/postgresql_data_types.htm)*).*

***Note:*** *Use the formulae introduced in the lecture notes (also in Appendix) to compute the estimated query costs. Total query cost of a query tree is the sum of the costs of all leaves, the intermediate notes and the root of a query tree.*

# **Question 3. PostgreSQL and Query Optimization [20 marks]**

You are asked here to improve efficiency of two database queries. The only condition is that after making improvements your queries produce the same results as the original ones, and your databases contain the same information as before.

For the optimization purposes, you will use two databases. A database that was dumped into the file

GiantCustomer.data

And the other database that was dumped into the file

Library.data

Both files are accessible from the course Assignments web page. Copy both files into your private directory. You are to:

1. Use PostgreSQL in order to create a database and to execute the command

psql –d <database\_name> -f ~/<file\_name>

This command will execute the CREATE TABLE and INSERT commands stored in the file <file\_name>, and make a database for you.

1. Execute the following commands:
   * VACUUM ANALYZE customer;

on the database containing GiantCustomer.data file, and

* + VACUUM ANALYZE customer;
  + VACUUM ANALYZE loaned\_book;

on database containing Library.data file.

These commands will initialize the catalog statistics of your database <*database\_name\_x*>, and allow the query optimizer to calculate costs of query execution plans.

1. Read the PostgreSQL Manual and learn about EXPLAIN command, since you will need it when optimizing queries. Note that a PostgreSQL answer to EXPLAIN <*query*> command looks like:

NOTICE: QUERY PLAN:

Merge Join (cost=6.79..7.10 rows=1 width=24)

-> Sort (cost=1.75..1.75 rows=23 width=12)

-> Seq Scan on cust\_order o (cost=0.00..1.23 rows=23 width=12)

-> Sort (cost=5.04..5.04 rows=2 width=12)

-> Seq Scan on order\_detail d (cost=0.00..5.03 rows=2 width=12)

Here, PosgreSQL is informing you that it decided to apply Sort Merge Join algorithm and that this join algorithm requires Sequential Scan and Sort of both relations. The shaded number 7.10 is an estimate of the query execution cost made by PostgreSQL. When making an improved query, you will compare your achievement to this figure, and compute the relative improvement using the following formula

**(original\_cost – new\_cost) / original\_cost.**

You may also want to use EXPLAIN ANALYZE <*query*> command that will give you additional information about the actual query execution time. Please note, the query execution time figures are not quiet reliable. They can vary from one execution to the other, since they strongly depend on the workload imposed on the database server by users. ***To get a more reliable query time measurement, you should run your query a number of times and then calculate the average***.

1. **[6 marks]** Improve the cost estimate of the following query:

select count(\*) from customer where no\_borrowed = 4;

issued against the database containing GiantCustomer.data. Make such changes to your database or to the query that will allow you to produce the same result as the original query, but in a more efficient way. Of course, your changes have to be fair. Analyze the output from the PostgreSQL query optimizer and make a plan on how to improve the efficiency of the query.

*Show what you have done by copying appropriate messages from the PostgreSQL prompt and explain why you have done it, calculate the improvement*. Each time you want to quit with that database, please drop it, since it occupies a lot of memory space.

**Marking schedule:**

You will receive:

* 5 marks if your query cost estimate is at least 64% better than the original one.
* between 2 and 4 marks if your query cost estimate is between 20% and 60% better than the original one and your marks will be calculated proportionally.
* up to 1 additional marks if you give reasonable explanations of what you have done.

**ANSWER**

Original cost = 115.37

1. **[4 marks]** Improve the efficiency of the following query:

select \* from customer where customerid = 4567;

issued against the database containing GiantCustomer.data. Make such changes to your database or to the query that will allow you to produce the same result as the original query, but in a more efficient way. Analyze the output from the PostgreSQL query optimizer and make a plan how to improve the efficiency of the query.

*Show what you have done by copying appropriate messages from the PostgreSQL prompt and explain why you have done it, calculate the improvement.* Each time you want to quit with that database, please drop it, since it occupies a lot of memory space.

**Marking schedule:**

You will receive

* 3 marks if your query cost estimate is 93% (or more) better than the original one.
* between 1 and 3 marks if your query cost estimate is better between 20% and 93% than the original one and your marks will be calculated proportionally to the improvement achieved.
* up to 1 additional marks if you give reasonable explanations of what you have done.

Original cost = 114.25

1. **[10 marks]** The following query is issued against the database containing the data from Library.data. It retrieves information about every customer for whom there exist less than three other customers borrowing more books than she/he did:

select clb.f\_name, clb.l\_name, noofbooks

from (select f\_name, l\_name, count(\*) as noofbooks

from customer natural join loaned\_book

group by f\_name, l\_name) as clb

where 3 > (select count(\*)

from (select f\_name, l\_name, count(\*) as noofbooks

from customer natural join loaned\_book

group by f\_name, l\_name) as clb1

where clb.noofbooks<clb1.noofbooks)

order by noofbooks desc;

Unfortunately, the efficiency of the given query is very poor. Make such changes to your database or to the query that will allow you to produce the same result as the original query, but in a more efficient way.

Show what you have done by copying appropriate messages from the PostgreSQL prompt, calculate the improvement, and briefly explain why the query given is inefficient and why your query is better.

**Marking schedule:**

You will receive:

* 3 marks if you explain in English how the query computes the answer,
* 5 marks if your query has a cost estimate 70% (or more) better than the original one (otherwise, your marks will be calculated proportionally to the improvement achieved),
* An additional 2 marks if you give reasonable explanations of why the query given is inefficient and why is your query better.

**ANSWER**

Original cost = 83.04

+++++++++++++++++++++++++++++++++

**Appendix 1: Formulae for Computing a Query Cost Estimate**

For a relation with schema*R =* {*A1,...,Ak*}, the average size of a tuple is: *r* =

The size of relation is *s = n · r*, with *n* as the average number of tuples in the relation,

**Select**: for a selection node σC the assigned size is *aC* · *s*, where *s* is the size assigned to the successor and 100 · *aC* is the average percentage of tuples satisfying *C*

**Project**: for a projection node πRi the assigned size is (*1 - Ci*) · *s* ·*ri/ r*, where *ri* (r) is the average size of a tuple in a relation over *Ri* (*R*), *s* is the size assigned to the successor and *Ci* is the probability that two tuples coincide on *Ri*

**Join**: for a join node the assigned size is *s1/r1 · p · s2/r2 ·* (*r1 + r2 - r*), where *si* are the sizes of the successors, *ri* are the corresponding tuple sizes, r is the size of a tuple over the common attributes and *p* is the matching probability

**Union**: for a union node the assigned size is *s1 + s2 - p · s1* with the probability *p* for tuple of *R1* to coincide with a tuple over *R2*

**Difference**: for a diﬀerence node the assigned size is *s1·*(*1 - p*), where (*1 - p*) is probability that tuple from *R1*-relation does not occur as tuple in *R2*-relation

**Appendix 2: Using PostgreSQL on the workstations**

We have a command line interface to PostgreSQL server from ECS, so you need to run it from a terminal.

To connect to the servers of ECS, such as **greta-pt.ecs.vuw.ac.nz** or **barretts.ecs.vuw.ac.nz**, remotely, you can access PostgreSQL server at home via SSH as below:

**> ssh** [username]**@greta-pt.ecs.vuw.ac.nz**

* If you are not asked to enter your password, type "kinit [username]" at the shell prompt and enter your password.

To enable the various applications required, type either

**> need comp302tools**

**or**

**> need postgresql**

You may wish to add either “need comp302tools”, or the “need postgresql” command to your .cshrc file so that it is run automatically. Add this command after the command need SYSfirst, which has to be the first need command in your .cshrc file.

There are several commands you can type at the unix prompt:

**> createdb** 〈database\_name〉

Creates an empty database. The database is stored in the same PostgreSQL server used by all the students in the class. Your database may have an arbitrary name, but we recommend to name it either userid or userid\_x, where userid is your ECS user name and x is a number from 0 to 9. To ensure security, you must issue the following command as soon as you log-in into your database for the first time:

REVOKE CONNECT ON DATABASE <database\_name> FROM PUBLIC;

You only need to do this once (unless you get rid of your database to start again). **Note**, your markers may check whether you have issued this command and if they find you didn’t, you may be **penalized**.

**> psql** [ **–d** 〈db name〉]

Starts an interactive SQL session with PostgreSQL to create, update, and query tables in the database. The db name is optional (unless you have multiple databases)

**> dropdb** 〈databas\_name〉

Gets rid of a database. (In order to start again, you will need to create a database again)

**> pg\_dump -i** 〈databas\_name〉>〈file\_name〉

Dumps your database into a file in a form consisting of a set of SQL commands that would reconstruct the database if you loaded that file.

> **psql –d** <database\_name> **-f** <file\_name>

Copies the file <file\_name> into your database <database\_name>.

Inside and interactive SQL session, you can type SQL commands. You can type the command on multiple lines (note how the prompt changes on a continuation line). End commands with a ‘;’

There are also many single line PostgreSQL commands starting with ‘\’ . No ‘;’ is required. The most useful are

**\?**  to list the commands,

**\i**  〈file\_name〉

loads the commands from a file (e.g., a file of your table definitions or the file of data we provide).

**\dt**  to list your tables.

**\d**〈table\_name〉to describe a table.

**\q**  to quit the interpreter

\**copy** <table\_name> **to** <file\_name>

Copy your table\_name data into the file file\_name.

\**copy** <table\_name> **from** <file\_name>

Copy data from the file file\_name into your table table\_name.

Note also that the PostgreSQL interpreter has some line editing facilities, including up and down arrow to repeat previous commands.

For longer commands, it is safer (and faster) to type your commands in an editor, then paste them into the interpreter!