Student ID: **First Name:** **Last Name:**

School of Engineering and Computer Science

SWEN 304 **Database System Engineering**

Assignment 2

Due: 23:59, Friday, 3 May 2024

The objective of this assignment is to test your understanding of Relational Algebra and Query Processing and Optimization. It is worth 5**%** 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 [35 marks]

Consider the Suppliers database schema given below.

Set of relation schemas:

Items ({*Id*, *Description*, *Category*}, {*Id*}),
Suppliers ({*SId*, *Name*, *Phone*, *Location*}, {*SId*})
Supplied_By ({*Id*, *SId*, *Amount*, *Year*, *Price*}, {*Id* + *SId* + *Year*})

Set of referential integrity constraints:
Supplied_By [*Id*] $\subseteq$ *Items* [*Id*],
Supplied_By [*SId*] $\subseteq$ *Suppliers* [*SId*]

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.

a) [20 marks] Translate the following queries into Relational Algebra:

1) **[5 marks]** Retrieve the names of all suppliers who *always* supply items of category ‘bread’.

2) **[5 marks]** Retrieve the descriptions of all items that are supplied by *two or more* suppliers.

3) **[5 marks]** Retrieve the names of all suppliers who have *not* supplied any item in 2024.

4) **[5 marks]** Retrieve the descriptions of all items that have been supplied by suppliers in Napier who *never* supply items with a price higher than $100.

b) [15 marks] Translate the following queries into plain English and into SQL:

1) **[5 marks]** $\pi_{Name,\ Phone}(Items * (\sigma_{Amount>1000}(Supplied\_By) * Suppliers))$

2) **[5 marks]** $\pi_{Description,\ Name}(\sigma_{price<10}(Items * (Supplied\_By * Suppliers)))$

3) **[5 marks]** $\pi_{SId}(\sigma_{Amount>1000}(Supplied\_By)) \cap \pi_{SId}(Supplied\_By * (\sigma_{Description='Carrot\ Cake'}(Items)))$

Question 2. Heuristic and Cost-Based Query Optimization [43 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)
);
```

a) [18 marks] Heuristic query optimization

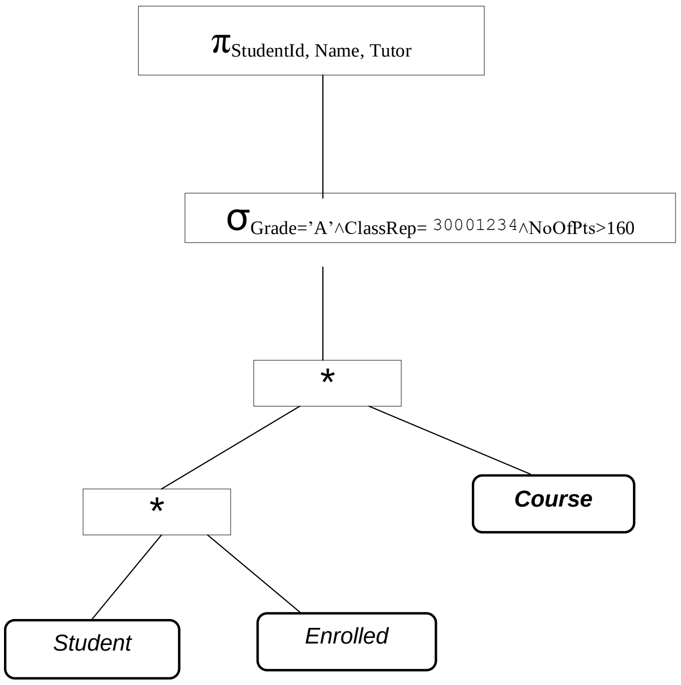
1) **[4 marks]** Transfer the following query into Relational Algebra.

```
SELECT StudentId, Name, Grade

FROM Student NATURAL JOIN Enrolled NATURAL JOIN Course

WHERE CourName = ‘Database Systems’ AND Term = 2024
                AND Tutor = ‘Tom’;
```

2)[14 marks] Transfer the following query tree into an optimized query tree using the query optimization heuristics. Draw the optimised query tree, and list the heuristic rules you applied.


$\pi_{\text{StudentId, Name, Tutor}}$
$\sigma_{\text{Grade='A'} \wedge \text{ClassRep= 30001234} \wedge \text{NoOfPts>160}}$
*
*
Course
Student
Enrolled

ANSWER

b) [25 marks] Query cost calculation

Suppose the following:

- The *Student* relation contains data about n_s = *50000* students (enrolled during the past *10* years),
- The *Course* relation contains data about n_c = *1000* courses,
- *The Enrolled relation contains data about n_e = 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.***

(i) $\pi_{\text{StudentId, Name, Grade}}$ ($\sigma_{\text{term = 2024} \wedge \text{CourseId ='SWEN304'}}$ (Student $\ast$ Enrolled))

ANSWER

(ii) $\pi_{\text{StudentId, Name, Grade}}(\text{Student} \ast \sigma_{\text{term} = 2024 \wedge \text{CourseId} = \text{'SWEN304'}}(\text{Enrolled}))$

ANSWER

iii) Which of the above two trees has a smaller query cost and why?

ANSWER

Hint: *To find out about the sizes of attributes in PostgreSQL please consult the documentation (www.postgresql.org/docs/9.2/static/datatype.html) or check this tutorial (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 [22 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:

i. 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.

ii. 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.

iii. 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***.

a) **[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

b) [4 marks] Improve the efficiency of the following query:

```
select * from customer where customerid = 2446;
```

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.

c) **[12 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

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

Appendix 1: Formulae for Computing a Query Cost Estimate

For a relation with schema $R = \{A_1,...,A_k\}$, the average size of a tuple is: $r = \sum_{j=1}^{k} l_j$

The size of relation is $s = n \cdot r$, with n as the average number of tuples in the relation,

Select: for a selection node σ_C the assigned size is $a_C \cdot s$, where s is the size assigned to the successor and $100 \cdot a_C$ is the average percentage of tuples satisfying C

Project: for a projection node π_{R_i} the assigned size is $(1 - C_i) \cdot s \cdot r_i / r$, where r_i (r) is the average size of a tuple in a relation over R_i (R), s is the size assigned to the successor and C_i is the probability that two tuples coincide on R_i

Join: for a join node the assigned size is $s_1/r_1 \cdot p \cdot s_2/r_2 \cdot (r_1 + r_2 - r)$, where s_i are the sizes of the successors, r_i 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 $s_1 + s_2 - p \cdot s_1$ with the probability p for tuple of R_1 to coincide with a tuple over R_2

Difference: for a difference node the assigned size is $s_1 \cdot (1 - p)$, where $(1 - p)$ is probability that tuple from R_1-relation does not occur as tuple in R_2-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!