SWEN304
Database System Engineering
Assignment 2

David Thomsen, 300052209

Question 1. Relational Algebra

a)

i) Retrieve the IDs and names of musicians who don't play instruments.

```
Musician {MusicianId, Mus_Name}
MINUS (Musician {MusicianId, Mus_Name}
JOIN Played_By {MusicianId})
```

ii) For all instruments that are played by a musician, return the IDs, names and a count of how many musicians it is played by.

```
(SUMMARIZE Played_By BY {InstrumentId} ADD (COUNT() AS
COUNT_MusicianId))
JOIN Instrument {InstrumentId, Inst_Name}
```

b)

i) $\pi Mus\_Name(\sigma Mus\_Type = "Conductor"(Musician))$

```
(Musician WHERE Mus_Type = "Conductor") {Mus_Name}
```

ii) $\pi Mus\_Name(((\pi MusicianId (\pi InstrumentId(\sigma Inst\_Name = "Piano" \vee Inst\_Name = "Violin"(Instrument))) a * r(Played\_B)) - (\pi MusicianId (\pi InstrumentId(\sigma Inst\_Name = "Guitar"(Instrument))) * r(Played\_B))) * r(Musician))$

```
((((Instrument
WHERE Inst_Name = "Piano" OR Inst_Name = "Violin")
{InstrumentId}
JOIN Played_By) {MusicianId}
MINUS ((Instrument WHERE Inst_Name = "Guitar")
{InstrumentId}
JOIN Played_By) {MusicianId})
JOIN Musician) {Mus_Name}
```

iii) $\pi Mus\_Name, Inst\_Name((\pi Mus\_Name, InstrumentId (\pi MusicianId, Mus\_Name(\sigma Mus\_Type = \text{"}Singer\text{"}(Musician)) * r(Played\_By))) * r(Instrument))$

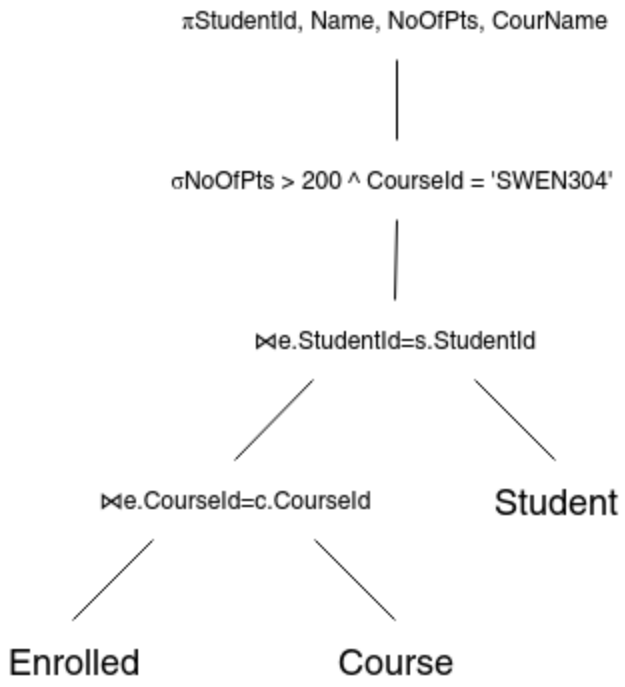
```
(((Musician WHERE Mus_Type = "Singer") {MusicianId,
Mus_Name}
JOIN Played_By) {Mus_Name, InstrumentId}
JOIN Instrument) {Mus_Name, Inst_Name}
```

Question 2. Heuristic and Cost-Based Query Optimization

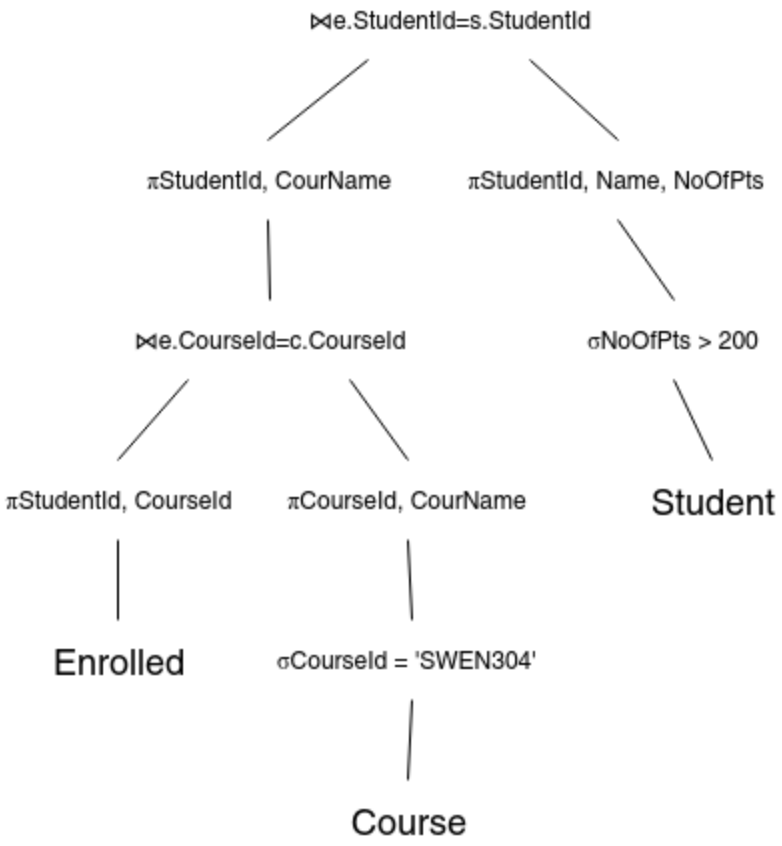
a)

i) $\pi StudentId, Name, NoOfPoints, CourName (\sigma NoOfPoints > 200 \wedge CourseId = \text{"}SWEN304\text{"} (r(Student) * (r(Enrolled) * r(Course))))$

ii)

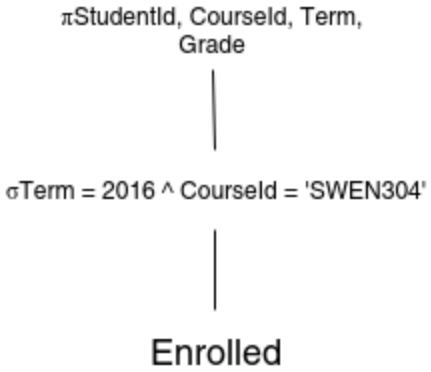


iii)



b)

i)



$s(Y) = 300,000\ enrolments\ /\ (500\ courses\ *\ 10\ years)\ =\ 60$

$L = int + char(15) + smallInt + char(15) = 4 + 15 + 2 + 15 = 36\ bytes$

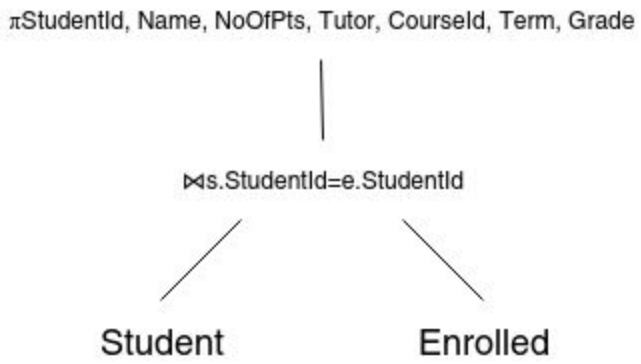
$f = floor[500/36] = 13\ bytes$

$B = 500\ bytes$

$b = ceiling[300,000/13] = 23,077$

$C = 23,077 + ceiling[60/13] = 23,082$

ii)



Question 2. . PostgreSQL and Query Optimization

a)

Analysis before modifying table:

```
EXPLAIN SELECT COUNT(*) FROM customer WHERE no_borrowed = 6;
                          QUERY PLAN
---------------------------------------------------------------------
 Aggregate  (cost=114.41..114.42 rows=1 width=0)
   ->  Seq Scan on customer  (cost=0.00..114.25 rows=63 width=0)
         Filter: (no_borrowed = 6)
(3 rows)
```

Modification made to table:

```
CREATE INDEX ON customer(no_borrowed);
```

Analysis after modifying table:

```
EXPLAIN SELECT COUNT(*) FROM customer WHERE no_borrowed = 6;
                                              QUERY PLAN
-----------------------------------------------------------------------------
 Aggregate  (cost=5.54..5.55 rows=1 width=0)
   ->  Index Only Scan using customer_no_borrowed_idx on customer
(cost=0.28..5.38 rows=63 w
idth=0)
         Index Cond: (no_borrowed = 6)
(3 rows)
```

Improvement = 5.55/114.42 = 0.05 = 95%

This modification improves the speed of the query greatly, however I do not think it is worth the improvement in query speed to perform this modification as it creates a substantial overhead to the database structure, both in terms of the space it takes up and the extra time required to add things to and remove things from the table.

b)
Analysis before modifying table:

```
EXPLAIN SELECT * FROM customer WHERE customerid = 4567;
                        QUERY PLAN
-----------------------------------------------------------------
 Seq Scan on customer  (cost=0.00..114.25 rows=1 width=56)
   Filter: (customerid = 4567)
(2 rows)
```

Modification made to table:

```
ALTER TABLE customer ADD CONSTRAINT primary_key PRIMARY KEY
(customerid);
```

Analysis after modifying table:

```
EXPLAIN SELECT * FROM customer WHERE customerid = 4567;
                                QUERY PLAN
-----------------------------------------------------------------------
 Index Scan using primary_key on customer  (cost=0.28..8.30 rows=1
width=56)
   Index Cond: (customerid = 4567)
(2 rows)
```

Improvement = 8.30/114.42 = 0.07 = 93%

Unlike the previous question, adding a primary key to customer on customerid makes perfect sense and is a constraint that should have been in place already. This also means that no customers can have the same customerid.

c)
Analysis before modifying table:

```
EXPLAIN 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;
                                                      QUERY PLAN
---------------------------------------------------------------------------
 Sort  (cost=94.08..94.10 rows=8 width=46)
   Sort Key: clb.noofbooks
   ->  Subquery Scan on clb  (cost=3.28..93.96 rows=8 width=46)
         Filter: (3 > (SubPlan 1))
         ->  HashAggregate  (cost=3.28..3.51 rows=23 width=38)
               Group Key: customer.f_name, customer.l_name
               ->  Hash Join  (cost=1.52..3.10 rows=23 width=38)
                     Hash Cond: (loaned_book.customerid =
customer.customerid)
                     ->  Seq Scan on loaned_book  (cost=0.00..1.26
rows=26 width=4)
                     ->  Hash  (cost=1.23..1.23 rows=23 width=42)
                           ->  Seq Scan on customer
(cost=0.00..1.23 rows=23 width=42)
         SubPlan 1
           ->  Aggregate  (cost=3.91..3.92 rows=1 width=0)
                 ->  HashAggregate  (cost=3.34..3.62 rows=23
width=38)
                       Group Key: customer_1.f_name,
customer_1.l_name
                       Filter: (clb.noofbooks < count(*))
                       ->  Hash Join  (cost=1.52..3.10 rows=23
width=38)
                             Hash Cond: (loaned_book_1.customerid =
customer_1.customerid)
                             ->  Seq Scan on loaned_book
loaned_book_1  (cost=0.00..1.26 rows
=26 width=4)
                             ->  Hash  (cost=1.23..1.23 rows=23
width=42)
                                   ->  Seq Scan on customer
customer_1  (cost=0.00..1.23 rows
=23 width=42)
(21 rows)
```

To make my query faster I will add a column to my customer table to store the number of books they have:

```
ALTER TABLE customer ADD COLUMN noofbooks INT NOT NULL DEFAULT 0;
```

And I will populate that column with the number of books for each customer:

```
UPDATE customer SET noofbooks = loaned_no.noofbooks FROM (SELECT
customerid, COUNT(customerid) AS noofbooks FROM loaned_book GROUP
BY customerid) AS loaned_no WHERE customer.customerid =
loaned_no.customerid;
```

My query is the same as the original one, except instead of calling on the loaned_book table, my query will just use the new noofbooks column in the customer table.

```
EXPLAIN SELECT clb.f_name, clb.l_name, noofbooks FROM (SELECT
f_name, l_name, noofbooks FROM customer) AS clb WHERE 3 > (SELECT
COUNT(*) FROM (SELECT noofbooks FROM customer) AS clb1 WHERE
clb.noofbooks<clb1.noofbooks);
                                    QUERY PLAN
-------------------------------------------------------------------
 Seq Scan on customer  (cost=0.00..31.59 rows=8 width=36)
   Filter: (3 > (SubPlan 1))
   SubPlan 1
     ->  Aggregate  (cost=1.31..1.32 rows=1 width=0)
           ->  Seq Scan on customer customer_1  (cost=0.00..1.29
rows=8 width=0)
                 Filter: (customer.noofbooks < noofbooks)
(6 rows)
```

Improvement = 31.59/114.42 = 0.28 = 72%

This is a large improvement, but not as large as the requirement of the assignment. The original query was inefficient because it called an extra table where my new query does not. However for my query to continue to be useful, the schema of the database would have to be modified so the noofbooks a customer has is always updated whenever they loan or return a book, a modification I have not made here.