Name: Xiaotian Liu
ID:300364582

SWEN304

Question 1 Relational Algebra

a)

1.

$\pi_{Name}(\sigma_{Category='food' \vee Category='healthcare'}(r(Manufacturer)* (r(Products)* r(Produced\_By))))$

2.

$\pi_{Name}(r(Manufacturer)* (r(Products)* r(Produced\_By)))- \pi_{Name}(\sigma_{Category! ='drink'}(r(Manufacturer)* (r(Products)* r(Produced\_By))))$

3.

$\pi_{Description}(r(Products*\pi_{PId}(\sigma_{MId! =MId2}(\pi_{PId, MId}(r(Produced\_By)) * \delta_{PID, MID\rightarrow MID2}(r(Produced\_By))))))$

4.

$\pi_{Description, Name}(\sigma_{Category='food'}(r(Manufacturer)* (r(Products)* r(Produced\_By))))$

b)

1.

Retrieve the Phone of all Manufacturers who produce some products that amounts are greater than 50.

```
SELECT Phone

FROM Products NATURAL JOIN Produced_By NATURAL JOIN Manufacturer

WHERE Amount>50;
```

2.

Retrieve the Id of all manufacturers who produce some products of category ‘Muffin’ that amounts are greater than 50.

```
SELECT MId

FROM Produced_By NATURAL JOIN Products

WHERE Amount>50 AND Description='Muffin';
```

Name: Xiaotian Liu
ID:300364582

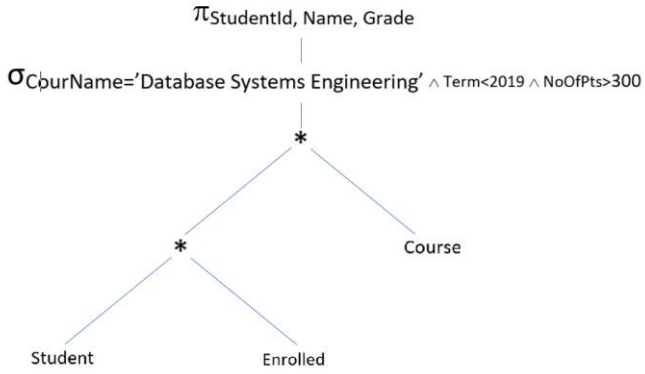
Question 2 Heuristic and Cost-Based Query Optimization

a)

1.

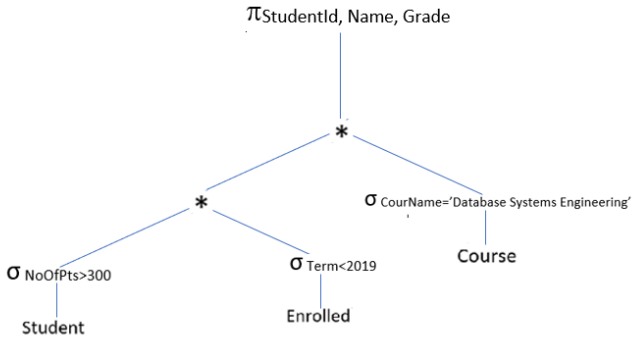
$\pi_{StudentId, Name, Grade}(\sigma_{CourName='Database\ Systems\ Engineering' \wedge Term<2019 \wedge NoOfPts>300}((r(Student) * r(Enrolled)) * r(Course)))$

2.

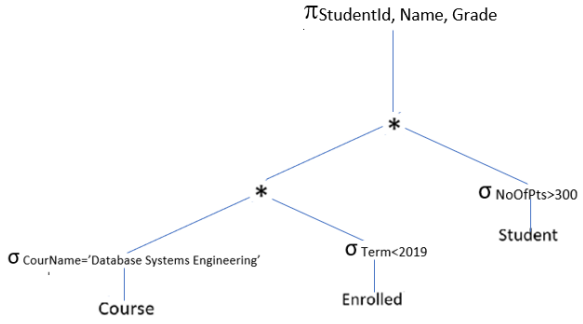


3.

Rule 6: Move select operations down the tree

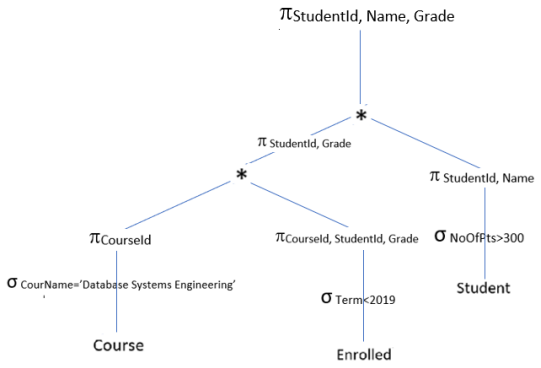


Rule 9: Switch Course and Student – the number of students who have more than 300 points must be greater than the number of courses called “Database System Engineering”, to restrict the select operation as early as possible the position of course and student should be switched



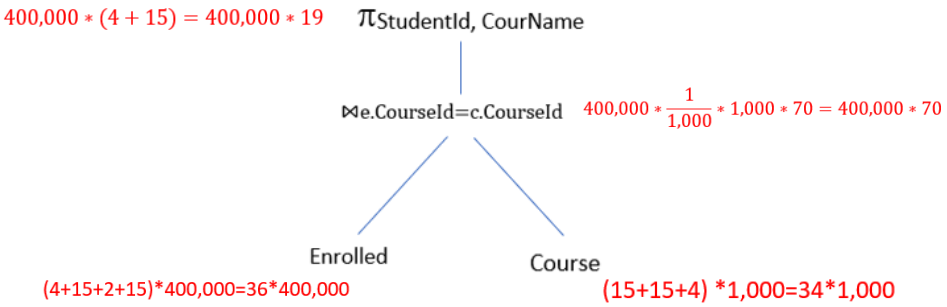
Name: Xiaotian Liu
ID:300364582

Rule 7: Apply π as early as possible to make sure only the needed attributes are selected – we only need CourseId (Course) and StudentId (Student) to join the Enrolled table and restrict the number of students in final result, however, we need to print the name and grade of each selected student, in this case, the optimized tree will be like this:



b)

1.



$$\frac{S_1}{r_1} * P * \frac{S_2}{r_2} * (r_1 + r_2 - r) = \frac{400{,}000 * 36}{36} * P * \frac{1{,}000 * 34}{34} * (36 + 34 - 0)$$

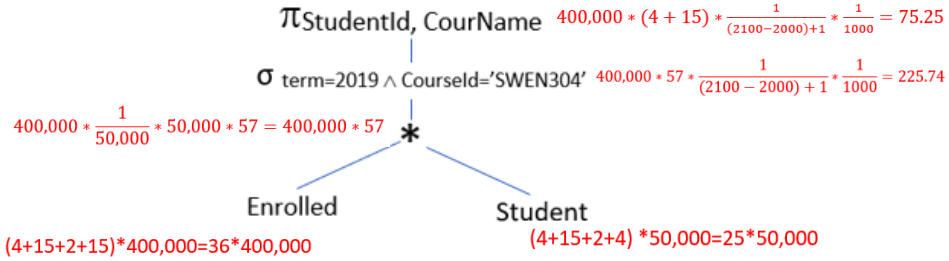
$$= 400{,}000 * P * 1{,}000 * 70$$

$$\mathrm{P} = \frac{400{,}000 * (36 + 34)}{400{,}000 * 1{,}000 * 70} = \frac{1}{1{,}000}$$

Total cost= 400,000*(19+70+36) +1,000*34=400,000*125+34,000=50,034,000

Name: Xiaotian Liu
ID:300364582

2.



$$\frac{S_1}{r_1} * P * \frac{S_2}{r_2} * (r_1 + r_2 - r) = \frac{400{,}000 * 36}{36} * P * \frac{50{,}000 * 25}{25} * (36 + 25 - 4)$$

$$= 400{,}000 * P * 50{,}000 * 57$$

$$\mathrm{P} = \frac{400{,}000*(36+25-4)}{400{,}000*50{,}000*57} = \frac{1}{50{,}000}$$

Total Cost=75.25+225.74+400,000*(57+36) +50,000*25=38,450,300.99

Question 3 PostgreSQL and Query Optimization

a)

```
indus% need comp302tools
indus% need postgres
indus% createdb xiaotianliu304a2
indus% psql -d xiaotianliu304a2 -f /home/liuxiao21/Desktop/304A2/GiantCustomer.data
SET
CREATE TABLE
COPY 4980
```

```
indus% psql -d xiaotianliu304a2
psql (10.9)
Type "help" for help.

xiaotianliu304a2=> VACUUM ANALYZE customer;
VACUUM
xiaotianliu304a2=> EXPLAIN select count(*) from customer where no_borrowed = 6;
```

Original:

```
xiaotianliu304a2=> EXPLAIN ANALYZE select count(*) from customer where no_borrowed=6;
                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Aggregate  (cost=114.41..114.42 rows=1 width=8) (actual time=2.698..2.720 rows=1 loops=1)
   ->  Seq Scan on customer  (cost=0.00..114.25 rows=63 width=0) (actual time=0.045..1.947 rows=63 loops=1)
         Filter: (no_borrowed = 6)
         Rows Removed by Filter: 4917
 Planning time: 1.605 ms
 Execution time: 3.278 ms
```

Improvement:

```
xiaotianliu304a2=> EXPLAIN ANALYZE select count(*) from customer where no_borrowed = 6;
                                                            QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------
 Aggregate  (cost=5.54..5.55 rows=1 width=8) (actual time=2.036..2.063 rows=1 loops=1)
   ->  Index Only Scan using customer_no_borrowed_idx on customer  (cost=0.28..5.38 rows=63 width=0) (actual time=0.494..1.236 rows=63 loops=1)
         Index Cond: (no_borrowed = 6)
         Heap Fetches: 0
 Planning time: 16.825 ms
 Execution time: 3.048 ms
(6 rows)
```

Name: Xiaotian Liu
ID:300364582

Cost improvement: (114.42-5.55)/114.42=95.149%

Average execution time improvement: (3.428-2.076)/3.428 = 39.44%

In the original query, it searched the whole table which means too many unnecessary information are read by the query, in this case I **added index** for all tuples, so that the query only needs to read the index to do count(*) instead of everything in the table. This will improve the performance include both cost and execution time because the cost of reading data was reduced.

b)

```
xiaotianliu304a2=> EXPLAIN ANALYZE select * from customer where customerid = 4567;
                                              QUERY PLAN
-----------------------------------------------------------------------------------------------------
 Seq Scan on customer  (cost=0.00..114.25 rows=1 width=56) (actual time=0.697..0.793 rows=1 loops=1)
   Filter: (customerid = 4567)
   Rows Removed by Filter: 4979
 Planning time: 0.165 ms
 Execution time: 0.893 ms
(5 rows)
```

```
xiaotianliu304a2=> ALTER TABLE customer ADD PRIMARY KEY (customerid);
ALTER TABLE
xiaotianliu304a2=> EXPLAIN ANALYZE select * from customer where customerid = 4567;
                                              QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 Index Scan using customer_pkey on customer  (cost=0.28..8.30 rows=1 width=56) (actual time=6.749..6.773 rows=1 loops=1)
   Index Cond: (customerid = 4567)
 Planning time: 1.169 ms
 Execution time: 6.875 ms
```

Cost improvement: (114.25-8.3)/114.25 = 92.73%

In this question I didn't count the average execution time because it was not accurate, for the original query I didn't restart the terminal so that the planning time and execution time were reduced to less than 1 ms.

To improve performance of this query, I set the customerid to be primary key, because fast query performance will be benefits from the NOT NULL optimization, the table data is physically organized to do fast lookups and sorts based on the primary key column. Since the table is quite big, have a primary key will be helpful to improve the performance.

c)

```
                                              QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------
 Sort  (cost=86.01..86.03 rows=8 width=40) (actual time=6.199..6.236 rows=3 loops=1)
   Sort Key: clb.noofbooks DESC
   Sort Method: quicksort  Memory: 25kB
   ->  Subquery Scan on clb  (cost=3.05..85.89 rows=8 width=40) (actual time=4.695..6.071 rows=3 loops=1)
         Filter: (3 > (SubPlan 1))
         Rows Removed by Filter: 12
         ->  HashAggregate  (cost=3.05..3.28 rows=23 width=40) (actual time=1.790..1.957 rows=15 loops=1)
               Group Key: customer.f_name, customer.l_name
               ->  Hash Join  (cost=1.52..2.86 rows=26 width=32) (actual time=0.711..1.486 rows=26 loops=1)
                     Hash Cond: (loaned_book.customerid = customer.customerid)
                     ->  Seq Scan on loaned_book  (cost=0.00..1.26 rows=26 width=4) (actual time=0.032..0.293 rows=26 loops=1)
                     ->  Hash  (cost=1.23..1.23 rows=23 width=36) (actual time=0.564..0.574 rows=23 loops=1)
                           Buckets: 1024  Batches: 1  Memory Usage: 10kB
                           ->  Seq Scan on customer  (cost=0.00..1.23 rows=23 width=36) (actual time=0.028..0.276 rows=23 loops=1)
         SubPlan 1
           ->  Aggregate  (cost=3.57..3.58 rows=1 width=8) (actual time=0.230..0.240 rows=1 loops=15)
                 ->  HashAggregate  (cost=3.05..3.28 rows=23 width=40) (actual time=0.130..0.172 rows=4 loops=15)
                       Group Key: customer_1.f_name, customer_1.l_name
                       Filter: (clb.noofbooks < count(*))
                       Rows Removed by Filter: 11
                       ->  Hash Join  (cost=1.52..2.86 rows=26 width=32) (actual time=0.716..1.490 rows=26 loops=1)
                             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) (actual time=0.027..0.288 rows=26 loops=1)
                             ->  Hash  (cost=1.23..1.23 rows=23 width=36) (actual time=0.563..0.573 rows=23 loops=1)
                                   Buckets: 1024  Batches: 1  Memory Usage: 10kB
                                   ->  Seq Scan on customer customer_1  (cost=0.00..1.23 rows=23 width=36) (actual time=0.035..0.278 rows=23 loops=1)
 Planning time: 1.346 ms
 Execution time: 7.228 ms
```

Name: Xiaotian Liu
ID:300364582

```
xiaotianliu304a2=> EXPLAIN ANALYZE SELECT f_name, l_name, count(*) as nooofbooks
xiaotianliu304a2-> FROM customer NATURAL JOIN loaned_book
xiaotianliu304a2-> GROUP BY f_name, l_name
xiaotianliu304a2-> ORDER BY count(*) DESC LIMIT 3;
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=3.58..3.59 rows=3 width=40) (actual time=2.957..3.175 rows=3 loops=1)
   ->  Sort  (cost=3.58..3.64 rows=23 width=40) (actual time=2.933..2.976 rows=3 loops=1)
         Sort Key: (count(*)) DESC
         Sort Method: top-N heapsort  Memory: 25kB
         ->  HashAggregate  (cost=3.05..3.28 rows=23 width=40) (actual time=2.217..2.417 rows=15 loops=1)
               Group Key: customer.f_name, customer.l_name
               ->  Hash Join  (cost=1.52..2.86 rows=26 width=32) (actual time=0.901..1.819 rows=26 loops=1)
                     Hash Cond: (loaned_book.customerid = customer.customerid)
                     ->  Seq Scan on loaned_book  (cost=0.00..1.26 rows=26 width=4) (actual time=0.045..0.362 rows=26 loops=1)
                     ->  Hash  (cost=1.23..1.23 rows=23 width=36) (actual time=0.661..0.683 rows=23 loops=1)
                           Buckets: 1024  Batches: 1  Memory Usage: 10kB
                           ->  Seq Scan on customer  (cost=0.00..1.23 rows=23 width=36) (actual time=0.033..0.332 rows=23 loops=1)
 Planning time: 5.929 ms
 Execution time: 4.462 ms
```

Cost improvement: (85.89-3.59)/85.89 = 95.82%

To improve this, I removed those nested queries include WHEN, GROUP BY and ORDER BY which are very expensive to operate. Instead of that, choose a simpler query to select result. The only thing needs to mention is that the result was reduced from 28 rows to 14 rows. Since we already know what the table looks like, I used a tricky way which is LIMIT 3 which only choose the top 3 results to make it same as the original table.