➢ Problems for Making Equivalent SQL Queries

Given the following queries, transform each query into another equivalent query (i.e., providing the same results) while satisfying the stated requirements.

(Note: the transformed query should not be worked by two steps. That is, it is not allowed to use the results of executing the nested query in the original query.)

➢ Submission

Read carefully this guidance for submission.

Submission components for given each query: 1) transformed query, 2) actual results executing an original query on Oracle databases(captured image), 3) actual results executing the transformed query on Oracle databases(captured image), and 4) the explanations why the transformed query is equivalent to the original query.

When you capture the screen to show actual results, you MUST contain a cmd window to show your unique IP address (type ipconfig in cmd windows) as follows. This means you show two windows in a screen where one is SQL Plus to show the results and the other is cmd windows to show your IP address of PC you were working.



Make one file named “HW#1\_STUDENTID.doc” (ppt, or pdf) including all the things as stated above and submit the file into e-class system.

1. Transform the following nested query into unnested query

|  |
| --- |
| SELECT pack\_id, speed , sector\_id  FROM acdb\_packages  WHERE sector\_id =  (SELECT sector\_id FROM acdb\_packages WHERE pack\_id = 10); |

* Transformed Query

select A.PACK\_ID, A.SPEED, A.SECTOR\_ID

from ACDB\_PACKAGES a, ACDB\_PACKAGES b

where B.PACK\_ID = 10 and A.SECTOR\_ID = B.SECTOR\_ID;

* Result of Original Query

텍스트이(가) 표시된 사진

자동 생성된 설명

* Result of Transformed Query

텍스트, 스크린샷, 전자기기, 디스플레이이(가) 표시된 사진

자동 생성된 설명

* Explanation

Using self-join. Get two relations with different nicknames a and b. Then, we make join with sector\_id of a and b. And add a condition b.pack\_id = 10.

2. Transform the following nested query into unnested query

|  |
| --- |
| SELECT first\_name , last\_name , join\_date  FROM acdb\_customers  WHERE extract(year from join\_date) =  (SELECT extract(year from join\_date) FROM acdb\_customers WHERE customer\_id = 372)  AND  extract (month from join\_date) =  (SELECT extract (month from join\_date) FROM acdb\_customers WHERE customer\_id = 372); |

* Transformed Query

select a.first\_name , a.last\_name , a.join\_date

from acdb\_customers a, acdb\_customers b

where extract(year from a.join\_date) = extract(year from b.join\_date) and b.customer\_id = 372

intersect

select a.first\_name, a.last\_name, a.join\_date

from acdb\_customers a, acdb\_customers b

where extract(month from a.join\_date) = extract(month from b.join\_date) and b.customer\_id = 372;

* Result of Original Query

텍스트이(가) 표시된 사진

자동 생성된 설명

* Result of Transformed Query

텍스트이(가) 표시된 사진

자동 생성된 설명

* Explanation

Using self-join and introduce intersect to get common part of both tables. One table is consisted of same people who have birth year which is same with customer\_id 372’s birth year. And the other is consisted of same people who have birth month which is same with customer\_id 372’s birth month. Using intersect operator, we can filter the results and find out those who are having same birth year and month which is same with customer\_id 372.

3. Transform the following nested query into unnested query

|  |
| --- |
| SELECT first\_name , last\_name , city , state, pack\_id  FROM acdb\_customers  WHERE pack\_id IN (SELECT pack\_id FROM acdb\_packages WHERE speed = '5Mbps'); |

* Transformed Query

select c.first\_name, c.last\_name, c.city, c.state, c.pack\_id

from acdb\_customers c, acdb\_packages p

where c.pack\_id = p.pack\_id and p.speed = '5Mbps';

* Result of Original Query

텍스트, 모니터, 스크린샷, 화면이(가) 표시된 사진

자동 생성된 설명

* Result of Transformed Query

텍스트, 모니터, 스크린샷, 컴퓨터이(가) 표시된 사진

자동 생성된 설명

* Explanation

To unnest query, we can make join in main query instead of sub query of IN operator.

Join the two relations with ‘=’. And add condition speed = ‘5Mbps’.

4. Transform the following nested query into unnested query

|  |
| --- |
| SELECT first\_name , monthly\_discount , pack\_id , main\_phone\_num ,  secondary\_phone\_num  FROM acdb\_customers  WHERE pack\_id IN  (SELECT pack\_id  FROM acdb\_packages  WHERE sector\_id IN  (SELECT sector\_id  FROM acdb\_sectors  WHERE sector\_name = 'Business')); |

* Transformed Query

select c.first\_name, c.monthly\_discount, c.pack\_id, c.main\_phone\_num, c.secondary\_phone\_num

from acdb\_customers c, acdb\_packages p, acdb\_sectors s

where c.pack\_id = p.pack\_id and p.sector\_id = s.sector\_id and s.sector\_name = 'Business';

* Result of Original Query

텍스트이(가) 표시된 사진

자동 생성된 설명

* Result of Transformed Query

텍스트이(가) 표시된 사진

자동 생성된 설명

* Explanation

Multiple nested query with IN operator with different three relations. Then, we can use multiple joins with three tables by FROM and WHERE clause. Join with the foreign key for each relation and then add the condition for our target.

5. Transform the following nested query into unnested query

|  |
| --- |
| SELECT customer\_id , first\_name , city , state ,birth\_date , monthly\_discount  FROM acdb\_customers  WHERE birth\_date =  (SELECT birth\_date FROM acdb\_customers WHERE customer\_id = 179)  AND  monthly\_discount >  (SELECT monthly\_discount FROM acdb\_customers WHERE customer\_id = 107); |

* Transformed Query

select a.customer\_id, a.first\_name, a.city, a.state, a.birth\_date, a.monthly\_discount

from acdb\_customers a, acdb\_customers b, acdb\_customers c

where a.birth\_date = b.birth\_date and b.customer\_id = 179

and a.monthly\_discount > c.monthly\_discount and c.customer\_id = 107;

* Result of Original Query

텍스트이(가) 표시된 사진

자동 생성된 설명

* Result of Transformed Query

텍스트이(가) 표시된 사진

자동 생성된 설명

* Explanation

Using self-join. Get the same relations with different three alias name.

Change sub query into (a.birth\_date = b.birth\_date and b.customer\_id = 179), and (a.monthly\_discount > c.monthly\_discount and c.customer\_id = 107)

6. Transform the following query into another query so as to use EXISTS or NOT EXISTS operator

|  |
| --- |
| SELECT pack\_id, speed, monthly\_payment  FROM acdb\_packages  WHERE monthly\_payment > ALL(SELECT monthly\_payment FROM acdb\_packages WHERE speed = '5Mbps'); |

* Transformed Query

select a.pack\_id, a.speed, a.monthly\_payment

from acdb\_packages a

where not exists(select a.monthly\_payment from acdb\_packages b where a.monthly\_payment <= b.monthly\_payment and b.speed='5Mbps');

* Result of Original Query

텍스트이(가) 표시된 사진

자동 생성된 설명

* Result of Transformed Query

텍스트이(가) 표시된 사진

자동 생성된 설명

* Explanation

To get ALL things that satisfy the condition is equal to find and remove ALL things that do not satisfy the condition. So, ‘> ALL’ can be replaced with ‘NOT EXISTS’. In NOT EXISTS’ sub query, conditions should be opposite from the condition of ALL’s sub query.

7. Transform the following query into another query so as to use EXISTS or NOT EXISTS operator

|  |
| --- |
| SELECT pack\_id ,speed , monthly\_payment  FROM acdb\_packages  WHERE monthly\_payment > ANY (SELECT monthly\_payment FROM acdb\_packages WHERE speed = '5Mbps'); |

* Transformed Query

select a.pack\_id, a.speed, a.monthly\_payment

from acdb\_packages a

where exists(select a.monthly\_payment from acdb\_packages b where b.speed='5Mbps' and a.monthly\_payment > b.monthly\_payment);

* Result of Original Query

텍스트이(가) 표시된 사진

자동 생성된 설명

* Result of Transformed Query

텍스트이(가) 표시된 사진

자동 생성된 설명

* Explanation

The specified query must select a monthly payment greater than the user at 5 Mbps speed. So I used the EXITES operator to transform a given query.

8. Transform the following query into another query so as to use COUNT(\*) operator

|  |
| --- |
| SELECT pack\_id ,speed , monthly\_payment  FROM acdb\_packages p1  WHERE EXISTS (select \* FROM acdb\_packages p2 WHERE p2.speed = '5Mbps'  and p1.monthly\_payment <= p2.monthly\_payment); |

* Transformed Query

select a.pack\_id, a.speed, a.monthly\_payment

from acdb\_packages a

where (select count(\*) from acdb\_packages b where b.speed='5Mbps' and a.monthly\_payment <= b.monthly\_payment)>0;

* Result of Original Query

텍스트이(가) 표시된 사진

자동 생성된 설명

* Result of Transformed Query

텍스트이(가) 표시된 사진

자동 생성된 설명

* Explanation

EXISTS means that there is at least one data that meets the criteria. So, I used 'COUNT' aggregation to count the number of data that fit the conditions. It has something, then it should be more than zero.

9. Transform the following query into another query so as to use EXISTS or NOT EXISTS operator

|  |
| --- |
| SELECT pack\_id ,speed , monthly\_payment  FROM acdb\_packages p1  WHERE p1.pack\_id IN (select p2.pack\_id FROM acdb\_packages p2 WHERE p2.speed = '5Mbps' AND p1.monthly\_payment <= p2.monthly\_payment); |

* Transformed Query

select a.pack\_id, a.speed, a.monthly\_payment

from acdb\_packages a

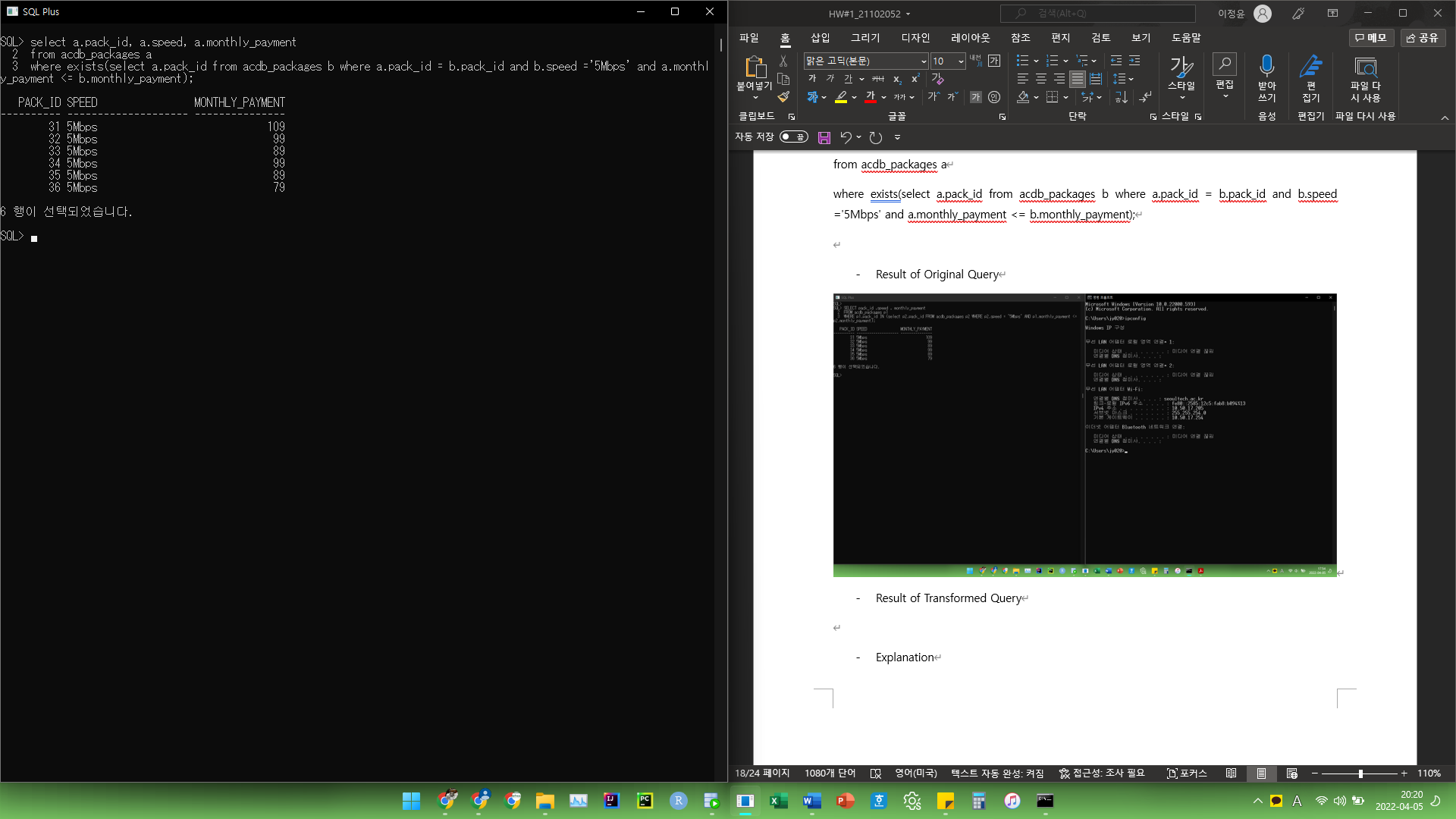
where exists(select a.pack\_id from acdb\_packages b where a.pack\_id = b.pack\_id and b.speed ='5Mbps' and a.monthly\_payment <= b.monthly\_payment);

* Result of Original Query

텍스트이(가) 표시된 사진

자동 생성된 설명

* Result of Transformed Query



* Explanation

IN operator and EXISTS operator can make same result. In each WHERE clause, the query requires to get pack\_id of main query’s acdb\_packages. So p1.pack\_id or a.pack\_id in sub query.

In transformed query, I used join to connect two relations, a and b. And then, add some same conditions from the original sub query in my own sub query’s WHERE clause.

10. Transform the following query into another query so as to use EXISTS or NOT EXISTS operator

|  |
| --- |
| SELECT pack\_id ,speed , monthly\_payment  FROM acdb\_packages p1  WHERE p1.pack\_id NOT IN (select p2.pack\_id FROM acdb\_packages p2 WHERE p2.speed = '5Mbps'  AND p1.monthly\_payment <= p2.monthly\_payment); |

* Transformed Query

select a.pack\_id, a.speed, a.monthly\_payment

from acdb\_packages a

where not exists(select a.pack\_id from acdb\_packages b where a.pack\_id = b.pack\_id and b.speed='5Mbps' and a.monthly\_payment <= b.monthly\_payment);

* Result of Original Query

텍스트이(가) 표시된 사진

자동 생성된 설명

* Result of Transformed Query

텍스트이(가) 표시된 사진

자동 생성된 설명

* Explanation

NOT IN operator and NOT EXISTS operator can make same result. The original query asks to get pack\_id which is not fit in the sub query’s condition. Then, we can use NOT EXISTS with sub query which has join operation between a and b. After that, add some conditions from the condition original sub query.

11. Transform the following query into another query so as to use MAX operator without ALL operator

|  |
| --- |
| SELECT pack\_id ,speed , monthly\_payment  FROM acdb\_packages  WHERE monthly\_payment  > ALL(SELECT monthly\_payment FROM acdb\_packages WHERE speed = '5Mbps'); |

* Transformed Query

select a.pack\_id, a.speed, a.monthly\_payment

from acdb\_packages a

where a.monthly\_payment > (select max(b.monthly\_payment) from acdb\_packages b where b.speed='5Mbps');

* Result of Original Query

텍스트이(가) 표시된 사진

자동 생성된 설명

* Result of Transformed Query

텍스트이(가) 표시된 사진

자동 생성된 설명

* Explanation

‘Attribute > ALL()’ means that we want to find values in attribute, which are having higher value than other all results of the sub query’s of ALL, i.e., higher than maximum value of sub query. So, we can introduce MAX aggregation in SELECT clause of nested query, then we can get same result of original query.

12. Transform the following query into another query so as to use MIN operator without SOME operator

|  |
| --- |
| SELECT pack\_id ,speed , monthly\_payment  FROM acdb\_packages  WHERE monthly\_payment  > ANY(SELECT monthly\_payment FROM acdb\_packages WHERE speed = '5Mbps'); |

\*SOME == ANY

* Transformed Query

select a.pack\_id, a.speed, a.monthly\_payment

from acdb\_packages a

where a.monthly\_payment > (select min(b.monthly\_payment) from acdb\_packages b where b.speed='5Mbps')

* Result of Original Query

텍스트, 스크린샷, 디스플레이이(가) 표시된 사진

자동 생성된 설명

* Result of Transformed Query

텍스트이(가) 표시된 사진

자동 생성된 설명

* Explanation

‘Attribute > ANY()’ means that we want to find values in attribute, which are having higher value than other any results of the sub query’s of ANY, i.e., higher than minimum value of sub query. So, we can introduce MIN aggregation in SELECT clause of nested query, then we can get same result of original query.