23-1 Database System Team Project

phase2

21900156 김예준 / 22000796 함상훈 / 22100579 이진주

1. Introduce

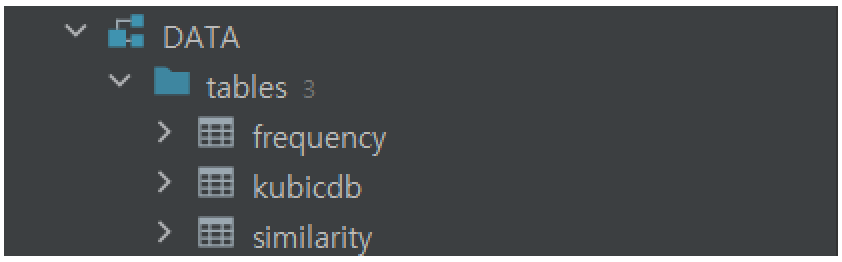
Phase 2 of the team project provides a new challenge for row data in KUBIC, which was covered in phase 1, with time complexity as a priority.

In phase2, we consider the new data tables together.

- frequency: contains the TF-IDF(term frequency-inverse document frequency) scores for selected words for each document. 877490 records and 4 columns. We can measure how relevant a word is to a document in a collection of documents by them.

- similarity: Cosine similarity analysis of the service documents(between each pair of data instances). 10,000,000 records and 3 columns. We use it to measure the similarity between each pair of documents.

source of this phase



The purpose of our assignment is to optimize the database using denormalization and indexing techniques. As databases grow in complexity and size, it becomes crucial to find ways to improve their performance and efficiency. Denormalization involves strategically duplicating and storing data in multiple places to reduce the need for complex joins and improve query performance. Indexing, on the other hand, involves creating data structures that allow for quick and efficient data retrieval.

In this assignment, we aim to explore the benefits of denormalization and indexing in optimizing our database which could have been traded off the normalization process. By implementing denormalization, we can reduce the number of tables and simplify the data model, leading to faster query execution. Additionally, by carefully creating indexes on frequently accessed columns, we can significantly improve the speed of data retrieval operations.

Through this assignment, we hope to demonstrate the advantages of denormalization and indexing in enhancing the performance of our database and make best performance in solving the given tasks. By optimizing our database, we can achieve faster query execution times.

We aim to define a database and provide queries with optimal performance for the given 11 tasks after denormalization and indexing. We proceeded according to the procedure below.

First, we denormalized the normalized tables and added indexes to make them in an optimal form for better performance.

Then, based on the resulting database, we wrote some queries that solve each of the given tasks.

The written queries are modified by utilizing various strategies to result in more efficient time performance.

To explicitly evaluate the improvement measure, we first designed 10 queries from the given questions. We computed the average query speed for each query out of 5 times executions from the schema constructed in our last assignment.

2. Denormalization

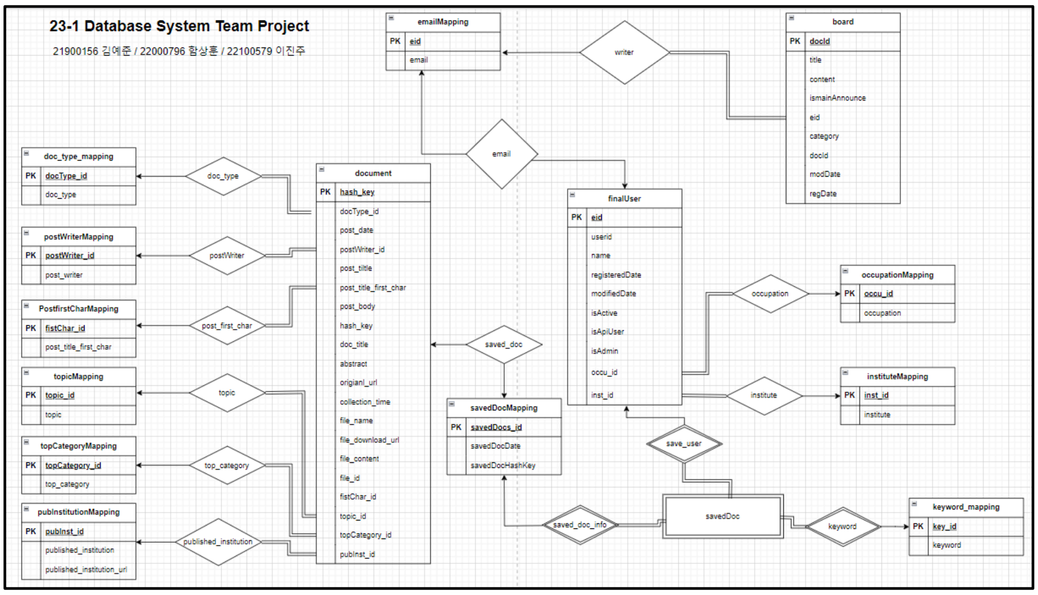


Figure 1. ER diagram constructed in the previous assignment (Phase 1)

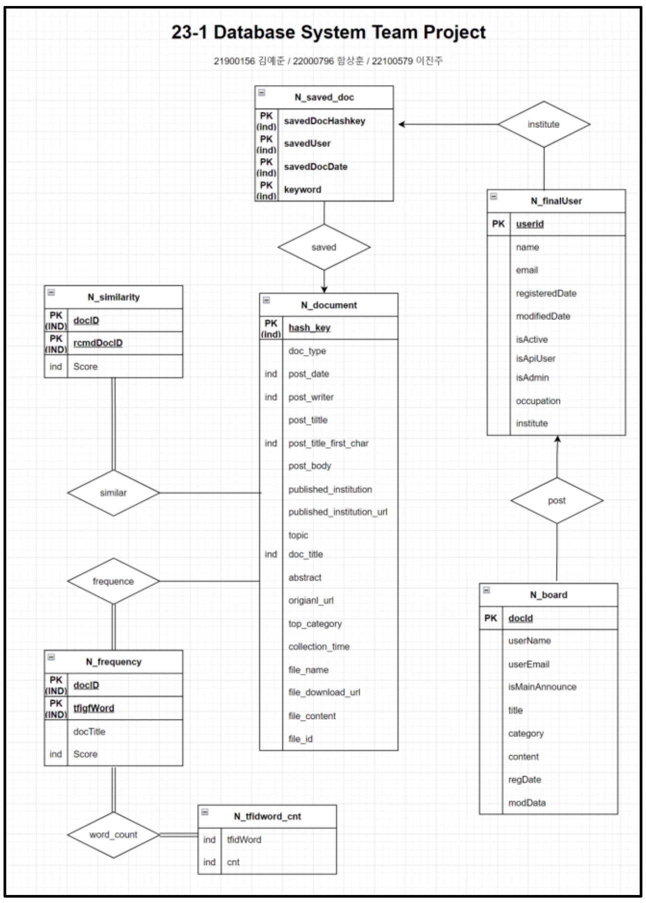


Figure 2 . New ER diagram constructed in this assignment by denormalized

We denormalized the tables that were normalized in phase 1(Figure 1) and returned them to the four tables of the big topic. The denormalized tables are Figure 2.

N_finalUser: This is where data about the user is stored.
N_document: Overall data about the document is stored.
N_board: Data related to the board is stored
N_savedDoc: Stores data for documents bookmarked by the user.

In addition to these four tables, we imported the frequency and similarity tables added in phase 2, named **N_freqency** and **N_similarity**, respectively, and used them together.

Using the N_frequency table, we created an additional table, **N_tfiword_cnt** , for use in several future tasks. This table is useful in solving given tasks and it will be explained further later.

The following DDL queries were used to complete each table by denormalization.

2.1 N_finalUser denormalization

```
create table N_finalUser
as select * from finalUser;

alter table N_finalUser add column email varchar(255);
update N_finalUser n, emailMapping x
set n.email = x.email
where n.eid = x.eid;
alter table N_finalUser drop eid;

alter table N_finalUser add column occupation varchar(255);
update N_finalUser n, occupationMapping x
set n.occupation = x.occupation
where n.occu_id = x.occu_id;
alter table N_finalUser drop occu_id;

alter table N_finalUser add column institute varchar(255);
update N_finalUser n, instituteMapping x
set n.institute = x.institute
where n.inst_id = x.inst_id;
alter table N_finalUser drop inst_id;
```

Firstly, we considered combining “finalUser”, “emailMapping” ,“occpuationMappping” and “instituteMapping” tables. For denormalizing this tables, we proceeded in the form of adding columns and then adding rows according to the conditions and dropping column which is executed like JOIN. By combining, we make it to access information about “finaluser” more quickly. Information of “finaluser” is same with attributes of “N_finalUser” in Figure 2.

2.2 N_document denormalization

```
create table N_document
as select * from document;

# postTitleFirstChar------------------------------------
alter table N_document add post_title_firt_char varchar(1);

update N_document n, PostfirstCharMapping x
set n.post_title_first_char = x.post_title_first_char
where n.firstChar_id = x.firstChar_id;

alter table N_document drop firstChar_id;

# topic------------------------------------
alter table N_document add topic varchar(255);

update N_document n, topicMapping x
set n.topic = x.topic
where n.topic_id = x.topic_id;

alter table N_document drop topic_id;

# topCategory------------------------------------
alter table N_document add top_category varchar(255);

update N_document n, topCategoryMapping x
set n.top_category = x.top_category
where n.topCategory_id = x.topCategory_id;

alter table N_document drop topCategory_id;

# publishedInstitution------------------------------------
alter table N_document add published_institution varchar(255);
alter table N_document add published_institution_url varchar(255);

update N_document n, pubInstitutionMapping x
set n.published_institution = x.published_institution
where n.pubInst_id = x.pubInst_id;

update N_document n, pubInstitutionMapping x
set n.published_institution_url = x.published_institution_url
where n.pubInst_id = x.pubInst_id;

alter table N_document drop pubInst_id;

# postWriter------------------------------------
alter table N_document add post_writer varchar(255);
```

```
update N_document n, postWriterMapping x
set n.post_writer = x.post_writer
where n.postWriter_id = x.postWriter_id;

alter table N_document drop postWriter_id;
```

Secondly, we considered combining “PostfirstCharMapping”, “topicMapping” ,“topCategoryMapping” and “pubInstitutionMapping” and “postWriterMapping” tables. For denormalizing this tables, we proceeded in the form of adding columns and then adding rows according to the conditions and dropping column which is executed like JOIN. By combining, we make it to access information about “documents” more quickly. Information of “document” is same with attributes of “N_document” in Figure 2.

2.3 N_board - board denormalization

```
CREATE TABLE N_board as(
SELECT docId, title, content, isMainAnnounce, board.eid as eid,
category, modDate, regDate, email
FROM DB16.board left join DB16.emailMapping on board.eid =
emailMapping.eid);
```

Thirdly, we considered combining “emailMapping” and “board” tables. For denormalizing this tables, we proceeded using JOIN. By combining, we make it to access information about “board” more quickly. Information of “board” is same with attributes of “N_board” in Figure 2.

2.4 N_savedDoc - savedDoc denormalization

```
CREATE TABLE N_savedDoc as (
SELECT eid, savedDocDate, savedDocHashKey, keyword
FROM DB16.savedDoc natural join DB16.savedDocsMapping natural join
DB16.keyword_mapping natural join DB16.finalUser);
```

Fourthly, we considered combining “savedDocMappingMapping” and “savedDoc” tables. For denormalizing this tables, we proceeded using Natural JOIN. By combining, we make it to access information about “savedDoc” more quickly. Information of “savedDoc” is same with attributes of “N_savedDoc” in Figure 2.

2.5 N_frequency and N_similarity from source DATA

```
create table N_frequency as
select * from DATA.frequency;

create table N_similarity as
select * from DATA.similarity;
```

Fifthly, we have to use the given ‘similarity’ table that contains the similarity between each pair of data instances and the given ‘frequency’ table that contains the TF-IDF (term frequency-inverse document frequency) scores for selected words for each document. For using those tables, we bring all of the datum from ‘similarity’ table and ‘frequency’ table on our schema by using ‘select *’ clause.

2.6 N_tfiword_cnt : table from task5

```
CREATE TABLE N_tfiword_cnt AS(
   select tfidfWord, count(tfidfWord) from N_frequency group by
tfidfWord
order by count(tfidfWord) desc);
```

Lastly, we create the table ‘N_tfiword_cnt’ that contains tfidfWord and the number of tfidfWord used which is arranged in descending order. After looking at the given tasks, we decided that we would need this table, so we made it .

3. indexing

We set up appropriate indexes on the tables to improve performance on certain search queries.

The indexes were set on the primary key of each table by default, and as we wrote the queries to solve the tasks, we set additional indexes on the appropriate columns for each query.

We find that queries with 'group by' are slow to execute. Therefore, using sorted data through index is expected to speed up execution.

Similarly, 'order by' was often used in queries, which could also be expected to improve performance by using index.

Frequently referenced data could also be expected to improve performance using index, but in this given task, frequently referenced data are usually pk, so index was automatically generated.

3.1 N_document

(PK) hash_key
post_date
post_title_first_char
doc_title, post_writer

```
create index doc_index1 on N_document (hash_key);
create index doc_index2 on N_document (post_date);
create index doc_index3 on N_document (post_title_first_char);
create index doc_index4 on N_document (doc_title, post_writer );
```

In order to further optimize the N_document table, we have created four indexes: doc_index1, doc_index2, doc_index3, and doc_index4. Each index is designed to improve the efficiency of specific types of queries by creating a data structure that allows for quick data retrieval based on the indexed columns. The index doc_index1 has been created on the column hash_key of the N_document table. This index is particularly useful when querying for specific documents based on their hash_key values. The index doc_index2 has been created on the column post_date of the N_document table. This index is beneficial for queries that involve filtering or ordering documents based on their post_date values. The index doc_index3 is built on the column post_title_first_char of the N_document table. This index is valuable when searching for documents based on the first character of their post_title. The index doc_index4 is a composite index created on the columns doc_title and post_writer of the N_document table. This composite index is useful for queries that involve filtering or sorting documents based on both the doc_title and post_writer values simultaneously.

3.2 N_savedDoc

(PK) savedDocHashKey, email, savedDocDate, keyword

```
create index savedDoc_index on N_savedDoc (savedDocHashKey, email,
savedDocDate, keyword);
```

To further optimize the N_savedDoc table, we have created an index called savedDoc_index. This index is designed to improve the efficiency of queries involving multiple columns: savedDocHashKey, email, savedDocDate, and keyword. The purpose of this index is to enable fast data retrieval based on combinations of these columns. By creating this index, the database engine can efficiently locate and retrieve saved documents based on the values stored in these columns.

3.3 N_frequency

(PK) docID, tfidfWord
Score desc

```
create index freq_index1 on N_frequency (docID, tfidfWord);
create index freq_index2 on N_frequency (Score desc);
```

In order to optimize the N_frequency table, we have created two indexes: freq_index1 and freq_index2. The index freq_index1 has been created on the columns docID and tfidfWord of the N_frequency table. This index is particularly useful for queries that involve searching or filtering frequency records based on both the document ID (docID) and the TF-IDF word (tfidfWord). The index freq_index2 has been created on the column Score in descending order (desc) of the N_frequency table. This index is valuable for queries that require sorting frequency records based on the Score column in descending order.

3.4 N_similarity

(PK) docID, rcmdDocID
Score desc

```
create index simil_index1 on N_similarity (docID, rcmdDocID);
create index simil_index2 on N_similarity (Score desc);
```

In order to optimize the N_similarity table, we have created two indexes: simil_index1 and simil_index2. The index simil_index1 has been created on the columns docID and rcmdDocID of the N_similarity table. This index is particularly useful for queries that involve searching or filtering similarity records based on both the source document ID (docID) and the recommended document ID (rcmdDocID). The index simil_index2 has been created on the column Score in descending order (desc) of the N_similarity table. This index is valuable for queries that require sorting similarity records based on the Score column in descending order.

3.5 N_tfiword_cnt

cnt desc, tfidWord

```
create index word_index on N_tfiword_cnt (cnt desc, tfidfWord);
```

To optimize the N_tfiword_cnt table, we have created an index called word_index. This index is designed to improve the performance of queries involving the cnt and tfidfWord columns. The index is constructed to allow for efficient data retrieval based on the combination of these columns. First of indexes is the cnt column in descending order (desc) and second is the tfidfWord column. This indexing enables the database to quickly search, filter, and sort records based on both the cnt and tfidfWord values.

4. task solutions

We solved the task to the best of our ability, using our database which was created by the above method.

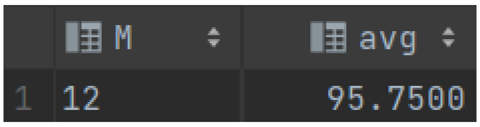
The queries used to solve each task, the results, and the execution time shown below.

task1
“ On average, in which month are the most publications released (posted)? Submit your solution along with the query that works on your database schema.”

query

```
with month as(
   select substring(post_date,6,2) as M
   from N_document
),
year as(
   select count(distinct left(post_date,4)) as Y
   from N_document
)

select M, COUNT(*) / (select * from year) as avg
from month
group by M
order by avg desc
limit 1;
```

result



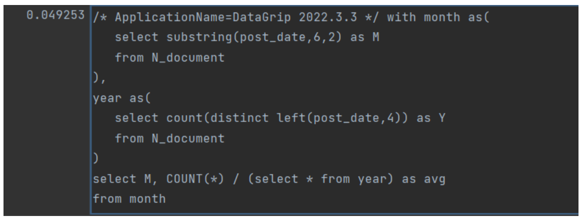
```
     M      avg
1    12     95.7500
```

execution time(output)

(execution: 55 ms, fetching: 21 ms)

duration(profiling)

0.049253



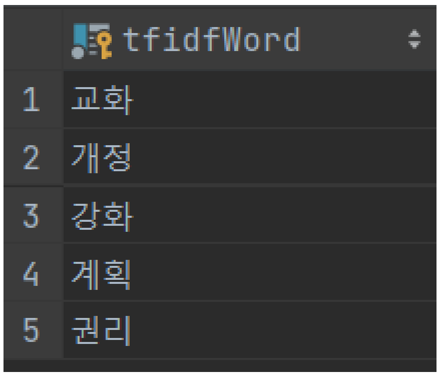
```
0.049253 /* ApplicationName=DataGrip 2022.3.3 */ with month as(
            select substring(post_date,6,2) as M
            from N_document
         ),
         year as(
            select count(distinct left(post_date,4)) as Y
            from N_document
         )
         select M, COUNT(*) / (select * from year) as avg
         from month
```

task2
“Find the five most important keywords (in terms of TF-IDF) in the document that was published in 2011 and has been bookmarked (saved) by the highest number of users.”

query

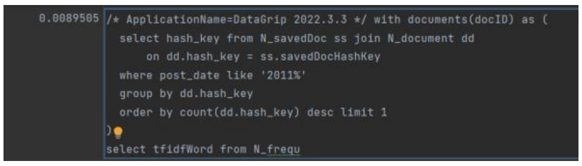
```sql
with documents(docID) as (
 select hash_key from N_savedDoc ss join N_document dd
     on dd.hash_key = ss.savedDocHashKey
 where post_date like '2011%'
 group by dd.hash_key
 order by count(dd.hash_key) desc limit 1
)

select tfidfWord from N_frequency f
WHERE f.docID in (SELECT * FROM documents)
order by Score desc limit 5;
```

result



	tfidfWord
1	교화
2	개정
3	강화
4	계획
5	권리

execution time(output)	duration(profiling)
(execution: 17 ms, fetching: 26 ms)	0.0089505



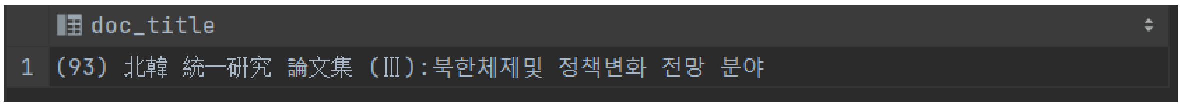
```
0.0089505 /* ApplicationName=DataGrip 2022.3.3 */ with documents(docID) as (
            select hash_key from N_savedDoc ss join N_document dd
                on dd.hash_key = ss.savedDocHashKey
            where post_date like '2011%'
            group by dd.hash_key
            order by count(dd.hash_key) desc limit 1
          )
          select tfidfWord from N_frequ
```

task3
“Give the title of the most similar document to the document that is saved least frequently by the users in the handong.ac.kr domain.”

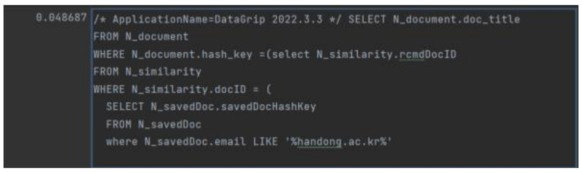
query

```
SELECT N_document.doc_title
FROM N_document
WHERE N_document.hash_key =(
   select N_similarity.rcmdDocID
   FROM N_similarity
   WHERE N_similarity.docID = (
     SELECT N_savedDoc.savedDocHashKey
     FROM N_savedDoc
     where N_savedDoc.email LIKE '%handong.ac.kr%'
     GROUP BY N_savedDoc.savedDocHashKey
     ORDER BY COUNT(*)
     LIMIT 1
   )
ORDER BY N_similarity.Score DESC
LIMIT 1,1);
```

result



```
  doc_title
1 (93) 北韓 統一研究 論文集 (Ⅲ):북한체제및 정책변화 전망 분야
```

execution time(output)	duration(profiling)
(execution: 49 ms, fetching: 22 ms)	0.048687 

```
0.048687 /* ApplicationName=DataGrip 2022.3.3 */ SELECT N_document.doc_title
         FROM N_document
         WHERE N_document.hash_key =(select N_similarity.rcmdDocID
         FROM N_similarity
         WHERE N_similarity.docID = (
           SELECT N_savedDoc.savedDocHashKey
           FROM N_savedDoc
           where N_savedDoc.email LIKE '%handong.ac.kr%'
```

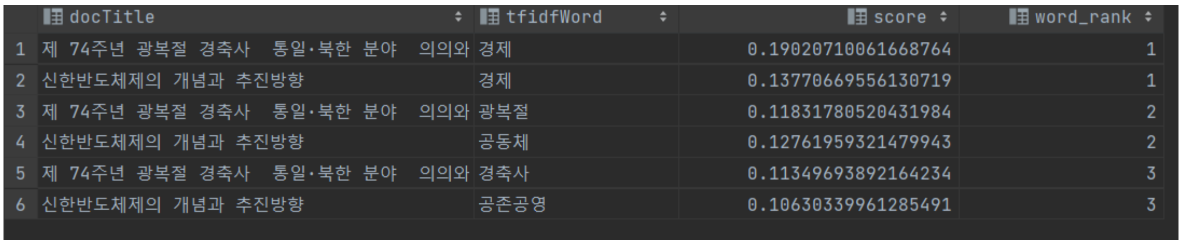
task4
“Find the three most important keywords (in terms of tf-idf) in the second most frequently bookmarked (saved by the users) document amongst the articles authored by “조한범”.”

query

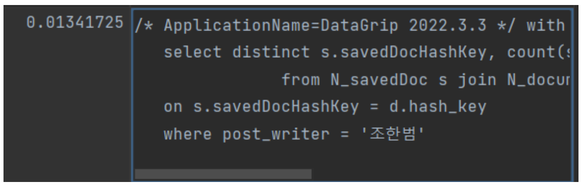
```
with joesDoc(hash_key, cnt) as (
  select distinct s.savedDocHashKey, count(savedDocHashKey) over
(partition by savedDocHashKey) cnt
                from N_savedDoc s join N_document d
  on s.savedDocHashKey = d.hash_key
  where post_writer = '조한범'
  order by cnt desc
)
```

```
select docTitle, tfidfWord, score, row_number() over (partition by docID
order by Score desc) word_rank
from N_frequency where docID in (
  select tmp.hash_key from (
      select hash_key, cnt, dense_rank() over(order by cnt desc) as
cnt_rank
      from joesDoc
      order by cnt desc
  ) as tmp
  where tmp.cnt_rank = 2
)
order by word_rank limit 6;
```

result



	docTitle	tfidfWord	score	word_rank
1	제 74주년 광복절 경축사 통일·북한 분야 의의와	경제	0.19020710061668764	1
2	신한반도체제의 개념과 추진방향	경제	0.13770669556130719	1
3	제 74주년 광복절 경축사 통일·북한 분야 의의와	광복절	0.11831780520431984	2
4	신한반도체제의 개념과 추진방향	공동체	0.12761959321479943	2
5	제 74주년 광복절 경축사 통일·북한 분야 의의와	경축사	0.11349693892164234	3
6	신한반도체제의 개념과 추진방향	공존공영	0.10630339961285491	3

execution time(output)	duration(profiling)
(execution: 20 ms, fetching: 25 ms)	0.01341725 

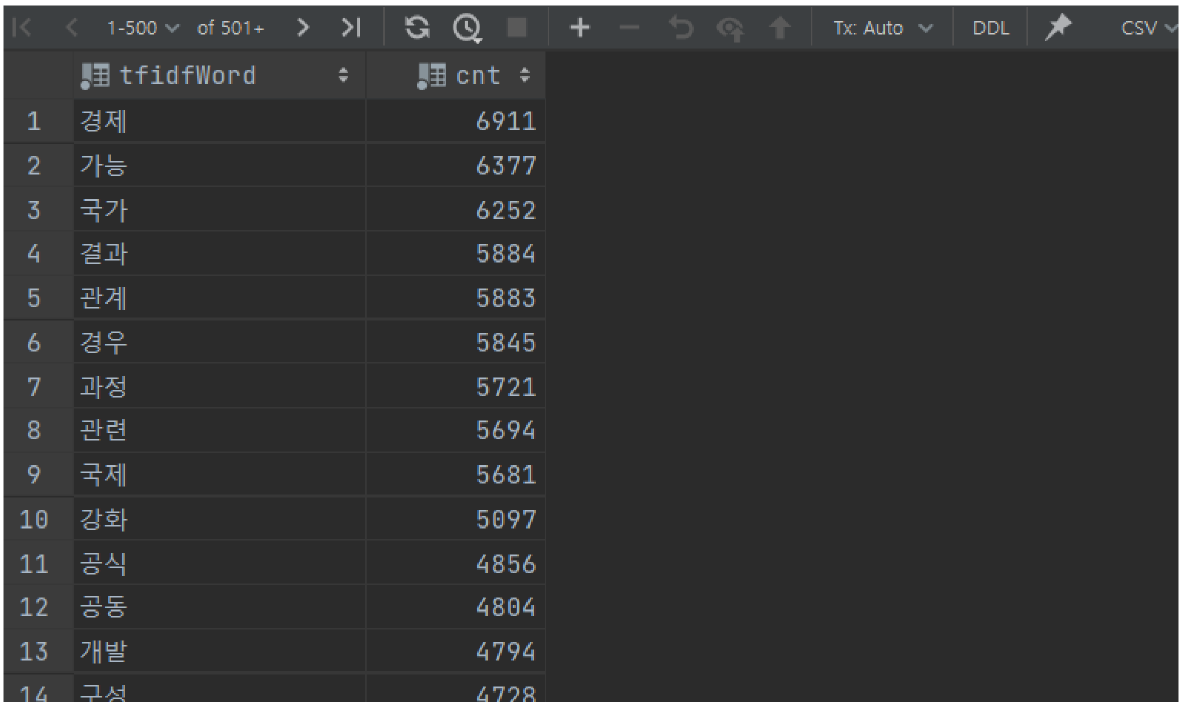
```
0.01341725 /* ApplicationName=DataGrip 2022.3.3 */ with
              select distinct s.savedDocHashKey, count(s
                          from N_savedDoc s join N_docur
              on s.savedDocHashKey = d.hash_key
              where post_writer = '조한범'
```

task5
“For all words that are used in the frequency analysis, show how many times each word has been used in the analysis (how many times each words has been used in the frequency table)”

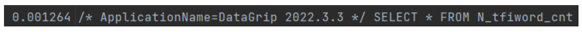
query

```
## DDL of N_tfiaord_cnt table :
# CREATE TABLE T_tfiword_cnt AS(
#    select tfidfWord, count(tfidfWord) as cnt
#    from N_frequency group by tfidfWord
#     order by count(tfidfWord) desc
# );

SELECT * FROM N_tfiword_cnt;
```

result

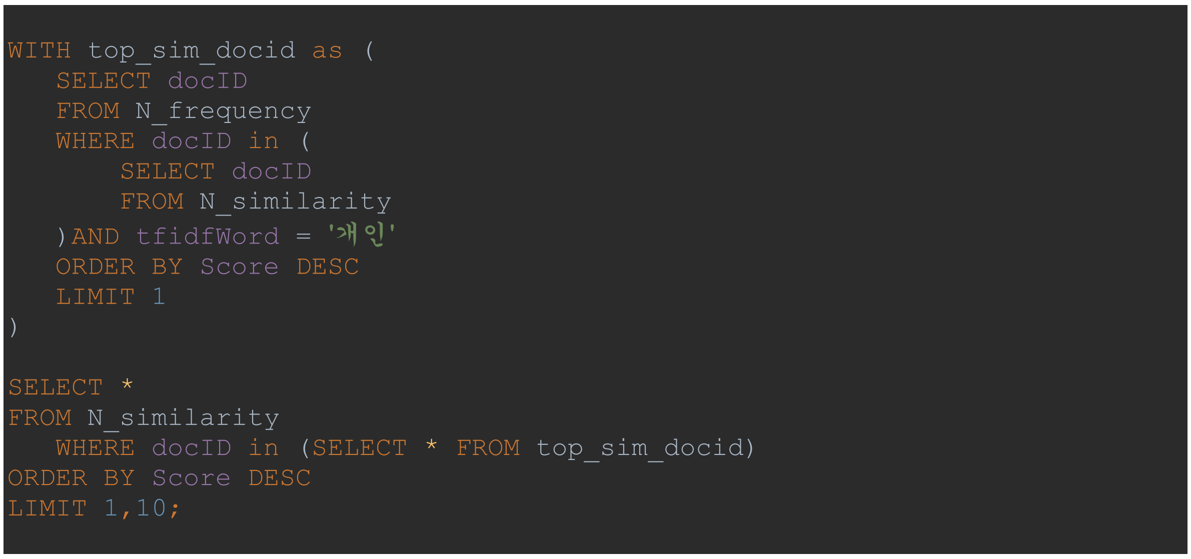


```
1-500 of 501+                Tx: Auto   DDL   CSV
    tfidfWord     cnt
1   경제          6911
2   가능          6377
3   국가          6252
4   결과          5884
5   관계          5883
6   경우          5845
7   과정          5721
8   관련          5694
9   국제          5681
10  강화          5097
11  공식          4856
12  공동          4804
13  개발          4794
14  구성          4728
```

execution time(output)	duration(profiling)
(execution: 12 ms, fetching: 28 ms)	0.001264  `0.001264 /* ApplicationName=DataGrip 2022.3.3 */ SELECT * FROM N_tfiword_cnt`

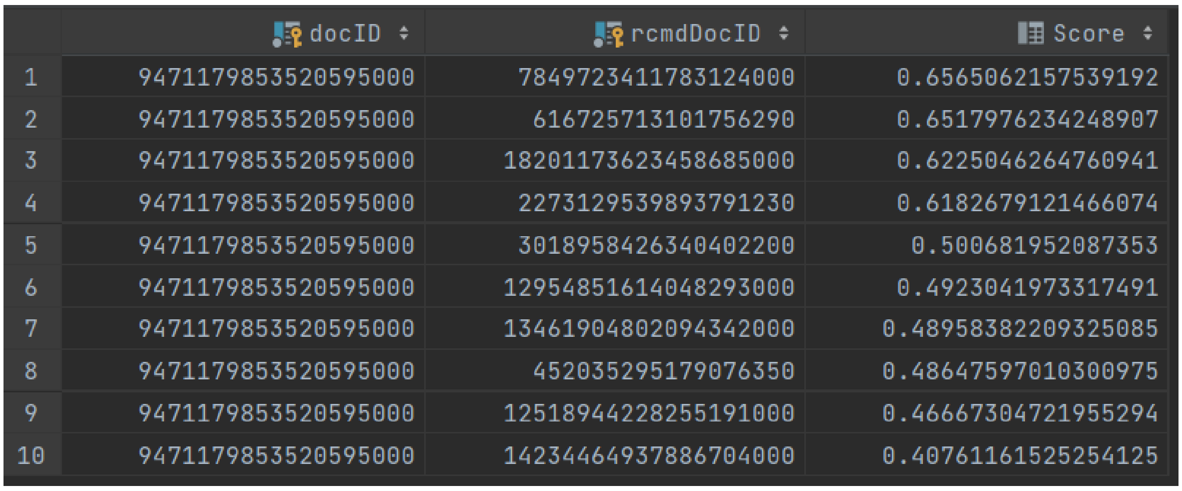
task6
“Find the ten most similar documents to those of the author who holds the most representative document for keyword “개인.” + Consider only the documents who have records in the frequency and similarity tables”

query



```
WITH top_sim_docid as (
   SELECT docID
   FROM N_frequency
   WHERE docID in (
       SELECT docID
       FROM N_similarity
   )AND tfidfWord = '개인'
   ORDER BY Score DESC
   LIMIT 1
)

SELECT *
FROM N_similarity
   WHERE docID in (SELECT * FROM top_sim_docid)
ORDER BY Score DESC
LIMIT 1,10;
```

result



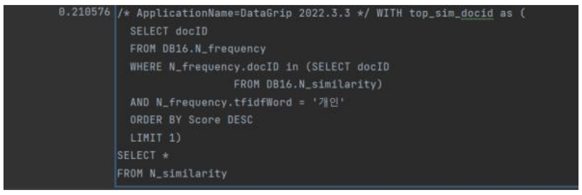
	docID	rcmdDocID	Score
1	9471179853520595000	7849723411783124000	0.6565062157539192
2	9471179853520595000	616725713101756290	0.6517976234248907
3	9471179853520595000	1820117362345868500	0.6225046264760941
4	9471179853520595000	2273129539893791230	0.6182679121466074
5	9471179853520595000	3018958426340402200	0.500681952087353
6	9471179853520595000	1295485161404829300	0.4923041973317491
7	9471179853520595000	1346190480209434200	0.48958382209325085
8	9471179853520595000	452035295179076350	0.48647597010300975
9	9471179853520595000	1251894422825519100	0.46667304721955294
10	9471179853520595000	1423446493788670400	0.40761161525254125

execution time(output)

(execution: 217 ms, fetching: 22 ms)

duration(profiling)

0.210576



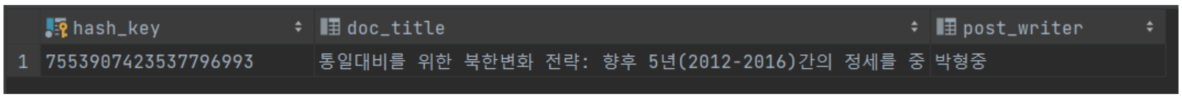
```
0.210576 /* ApplicationName=DataGrip 2022.3.3 */ WITH top_sim_docid as (
           SELECT docID
           FROM DB16.N_frequency
           WHERE N_frequency.docID in (SELECT docID
                               FROM DB16.N_similarity)
           AND N_frequency.tfidfWord = '개인'
           ORDER BY Score DESC
           LIMIT 1)
         SELECT *
         FROM N_similarity
```

task7
“Among the words that are used the 10th most frequently in the word frequency analysis, locate the document with the 200th highest score. Next, identify the document with the 7th highest similarity to the abovefound document. Please provide the ID, title, and author name for both documents. (You may use the UNION command to combine both results)”

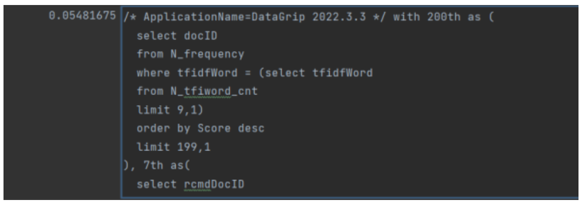
query

```
with 200th as (
 select docID
 from N_frequency
 where tfidfWord = (
     select tfidfWord
     from N_tfiword_cnt
     limit 9,1
 )order by Score desc
 limit 199,1
), 7th as(
 select rcmdDocID
 from N_similarity
 where docID in (select * from 200th)
 order by Score DESC
```

```sql
 limit 6,1
)

select hash_key, doc_title, post_writer
from N_document
where hash_key in (select * from 7th)
union
select hash_key, doc_title, post_writer
from N_document
where hash_key in (select * from 200th);
```



	hash_key	doc_title	post_writer
1	7553907423537796993	통일대비를 위한 북한변화 전략: 향후 5년(2012-2016)간의 정세를 중	박형중

execution time(output)	duration(profiling)
(execution: 77 ms, fetching: 24 ms)	0.05481675 

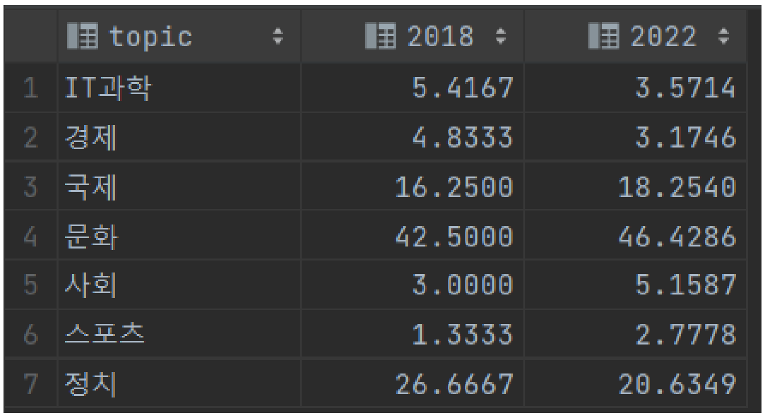
```
0.05481675 /* ApplicationName=DataGrip 2022.3.3 */ with 200th as (
             select docID
             from N_frequency
             where tfidfWord = (select tfidfWord
             from N_tfiword_cnt
             limit 9,1)
             order by Score desc
             limit 199,1
           ), 7th as(
             select rcmdDocID
```

task8
“Compare the topic distribution among the documents published in 2018 and 2022, respectively.”

query

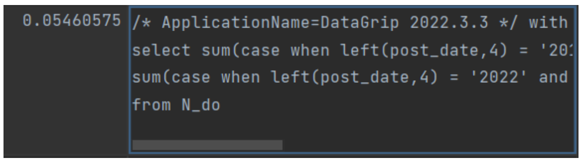
```sql
with T_sum as(
   select sum(case when left(post_date,4) = '2018' and topic is not null
and topic <> '' then 1 else 0 end) as 'sum_2018',
   sum(case when left(post_date,4) = '2022' and topic is not null and
topic <> '' then 1 else 0 end ) as 'sum_2022'
   from N_document
)

select * from(
   select topic, (count(*)/ sum_2018) * 100 as '2018'
   from N_document,T_sum
   where post_date like '2018%' and topic is not null and topic <> ''
   group by topic,sum_2018
   order by topic) as A join(
   select topic, (count(*) / sum_2022) * 100 as '2022'
   from N_document,T_sum
   where post_date like '2022%' and topic is not null and topic <> ''
   group by topic,sum_2022
   order by topic) as B using (topic);
```

result



	topic	2018	2022
1	IT과학	5.4167	3.5714
2	경제	4.8333	3.1746
3	국제	16.2500	18.2540
4	문화	42.5000	46.4286
5	사회	3.0000	5.1587
6	스포츠	1.3333	2.7778
7	정치	26.6667	20.6349

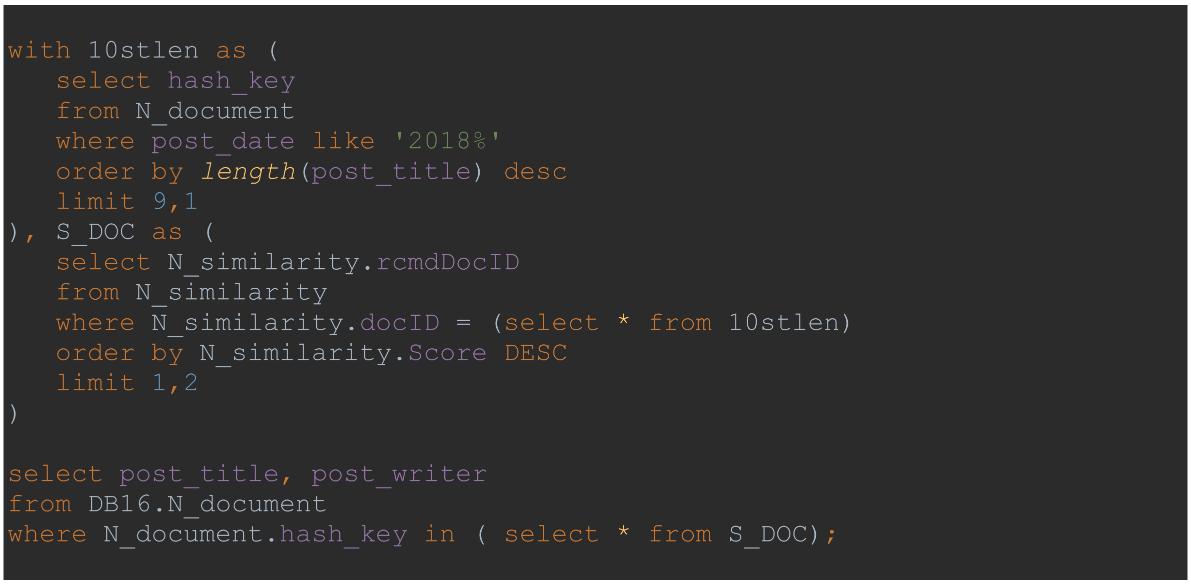
execution time(output)	duration(profiling)
(execution: 66 ms, fetching: 23 ms)	0.05460575



```
0.05460575 /* ApplicationName=DataGrip 2022.3.3 */ with
           select sum(case when left(post_date,4) = '20
           sum(case when left(post_date,4) = '2022' and
           from N_do
```

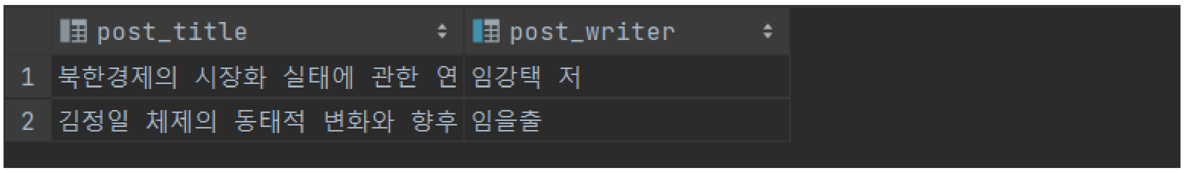
task9
“Find the titles (post_title) and authors (post_writer) of the two most similar documents (regardless of the published year) to the one with the 10th longest title among those published in 2018.”

query

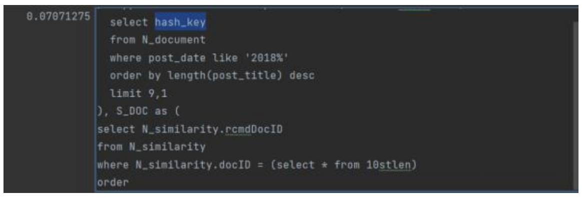


```
with 10stlen as (
   select hash_key
   from N_document
   where post_date like '2018%'
   order by length(post_title) desc
   limit 9,1
), S_DOC as (
   select N_similarity.rcmdDocID
   from N_similarity
   where N_similarity.docID = (select * from 10stlen)
   order by N_similarity.Score DESC
   limit 1,2
)

select post_title, post_writer
from DB16.N_document
where N_document.hash_key in ( select * from S_DOC);
```

result



	post_title	post_writer
1	북한경제의 시장화 실태에 관한 연	임강택 저
2	김정일 체제의 동태적 변화와 향후	임을출

execution time(output)	duration(profiling)
(execution: 70 ms, fetching: 20 ms)	0.07071275 

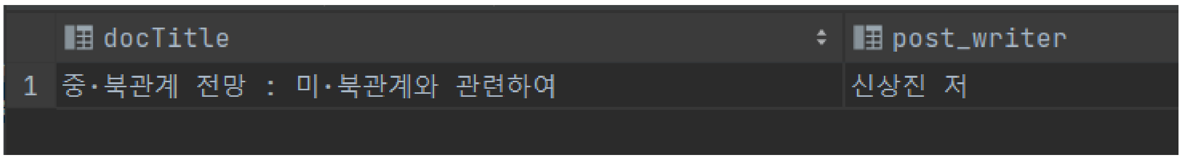
```
0.07071275   select hash_key
               from N_document
               where post_date like '2018%'
               order by length(post_title) desc
               limit 9,1
             ), S_DOC as (
             select N_similarity.rcmdDocID
             from N_similarity
             where N_similarity.docID = (select * from 10stlen)
             order
```

task10
“Among the documents whose titles start with "ㅈ”, Find the ID, title, and author name of the document with the highest tfidf importance for the keyword "관계". • Consider only the documents who have records in the frequency and similarity tables”

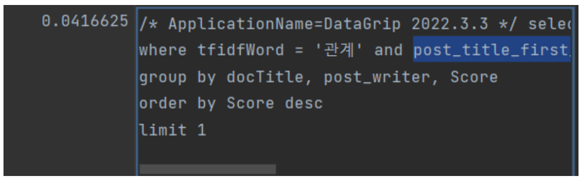
query

```
select docTitle, post_writer from N_frequency f join N_document d on
f.docID = d.hash_key
where tfidfWord = '관계' and post_title_first_char = 'ㅈ'
group by docTitle, post_writer, Score
order by Score desc
limit 1;
```

result



	docTitle	post_writer
1	중·북관계 전망 : 미·북관계와 관련하여	신상진 저

execution time(output)	duration(profiling)
(execution: 50 ms, fetching: 22 ms)	0.0416625 

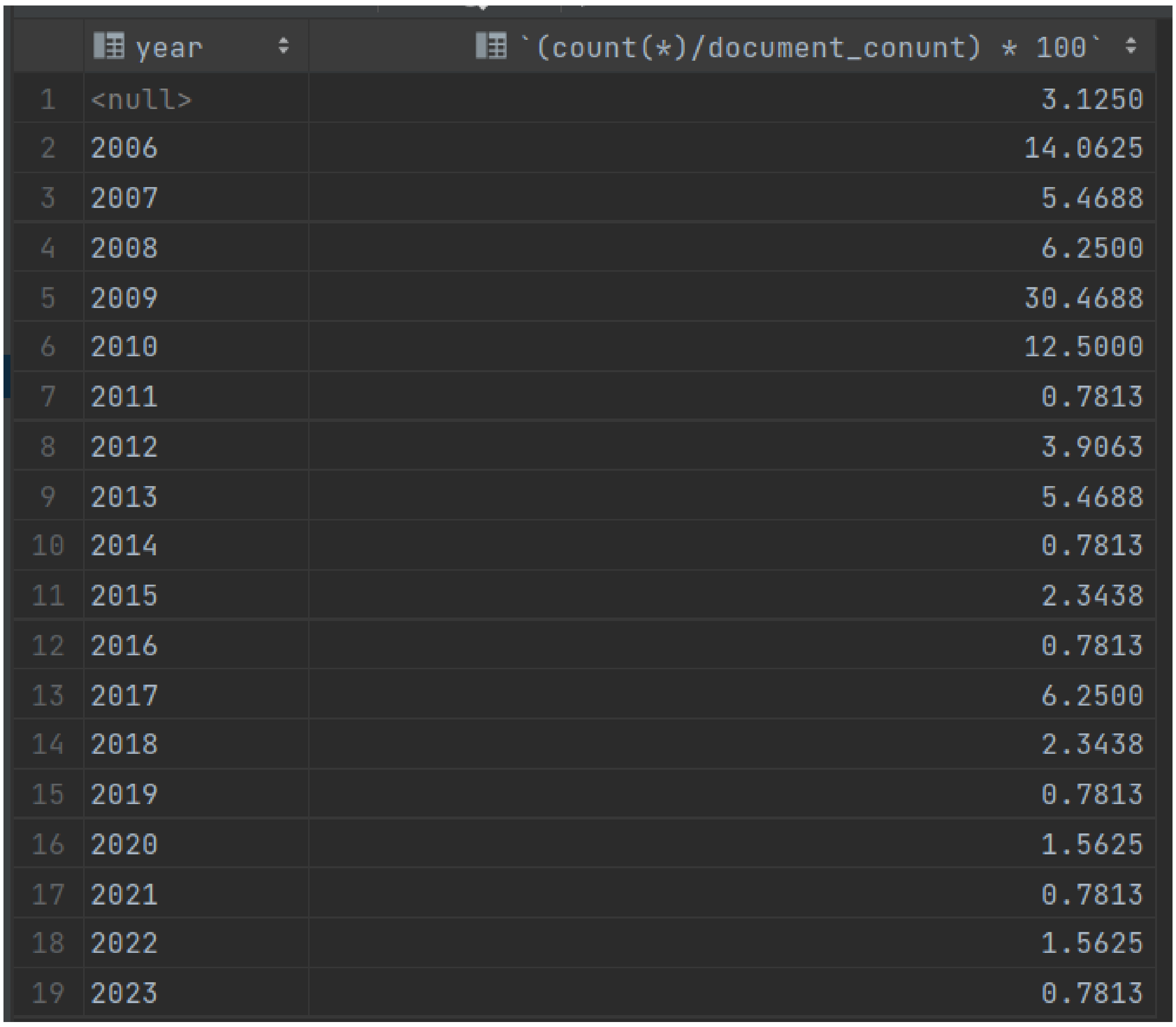
```
0.0416625  /* ApplicationName=DataGrip 2022.3.3 */ selec
           where tfidfWord = '관계' and post_title_first
           group by docTitle, post_writer, Score
           order by Score desc
           limit 1
```

task11
“Show and compare the yearly distribution of the document counts whose post_body contains “강경” and that of “대화””

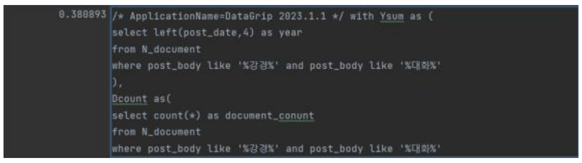
query

```
with Ysum as (
   select left(post_date,4) as year
   from N_document
   where post_body like '%강경%' and post_body like '%대화%'
),Dcount as(
   select count(*) as document_conunt
   from N_document
   where post_body like '%강경%' and post_body like '%대화%'
)

select year , (count(*)/document_conunt) * 100
from Ysum,Dcount
group by year,document_conunt
order by year;
```

result



	year	\`(count(*)/document_conunt) * 100\`
1	<null>	3.1250
2	2006	14.0625
3	2007	5.4688
4	2008	6.2500
5	2009	30.4688
6	2010	12.5000
7	2011	0.7813
8	2012	3.9063
9	2013	5.4688
10	2014	0.7813
11	2015	2.3438
12	2016	0.7813
13	2017	6.2500
14	2018	2.3438
15	2019	0.7813
16	2020	1.5625
17	2021	0.7813
18	2022	1.5625
19	2023	0.7813

execution time(output)	duration(profiling)
(execution: 392 ms, fetching: 20 ms)	0.380893 

```
0.380893 /* ApplicationName=DataGrip 2023.1.1 */ with Ysum as (
         select left(post_date,4) as year
         from N_document
         where post_body like '%강경%' and post_body like '%대화%'
         ),
         Dcount as(
         select count(*) as document_conunt
         from N_document
         where post_body like '%강경%' and post_body like '%대화%'
```

Regarding the execution time, since the duration of the task was such that there were always an unspecified number of concurrent users on the server, there was an error in the execution time for each run, so we performed about 10-20 iterations per task and chose the lowest execution time to record.

5. strategies

When we first tackled the task, we focused on getting a result somehow.

After obtaining the results, we went through the process of modifying the query to find and replace a statement that gives the same result but is more efficient in terms of execution time.

In order to determine which one has a lower execution time, we did theoretical investigations and direct experiments.

Here are some of the cases we found that were replaced to improve performance.

LIMIT n OFFSET m -> LIMIT p, q
We've tried writing and executing queries for the same result myself and found that there is a performance benefit to not using offset and taking the form of limit p,q.

= -> IN
We've seen by reference that there are benefits to using an IN statement instead of using the = operator in a WHERE clause. ref: MySQL IN 절을 통한 성능 개선 방법

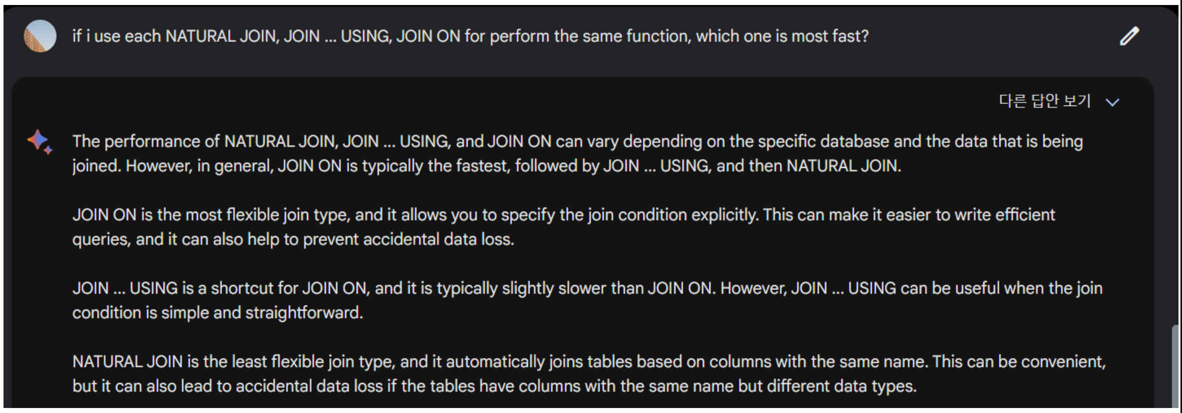
TEXT type -> VARCHAR
Since the TEXT type is referenced once more to get the actual data, it is less time-efficient than VARCHAR, so we converted it to VARCHAR type.

Minimize DISTINCT
DISTINCT statements cause a performance penalty, so we've removed them unless absolutely necessary.

ref: [SQL 성능을 높이는 5 가지 방법]

NATURAL JOIN, JOIN … USING, JOIN ON, -> JOIN ON

While NATURAL JOIN, JOIN ... USING, JOIN ON can perform the same function
Based on GoogleAi Bard's answer, we decided that it is more efficient to use JOIN... ON.
ref:



if i use each NATURAL JOIN, JOIN ... USING, JOIN ON for perform the same function, which one is most fast?

The performance of NATURAL JOIN, JOIN ... USING, and JOIN ON can vary depending on the specific database and the data that is being joined. However, in general, JOIN ON is typically the fastest, followed by JOIN ... USING, and then NATURAL JOIN.

JOIN ON is the most flexible join type, and it allows you to specify the join condition explicitly. This can make it easier to write efficient queries, and it can also help to prevent accidental data loss.

JOIN ... USING is a shortcut for JOIN ON, and it is typically slightly slower than JOIN ON. However, JOIN ... USING can be useful when the join condition is simple and straightforward.

NATURAL JOIN is the least flexible join type, and it automatically joins tables based on columns with the same name. This can be convenient, but it can also lead to accidental data loss if the tables have columns with the same name but different data types.

COUNT(*), COUNT(column), COUNT(DISTINCT(column)) ->COUNT(*)

When using COUNT, there was a performance benefit to using it with the * symbol rather than a specific column.

ref: [MySQL] COUNT 의 잘못된 인식과 속도 차이

6. summary of DB

As a result, our database which was used to generate all the results, has the following size.

Since we did not delete the normalized tables in phase 1 for stability, we supplemented the query with the naming policy of the tables (starting with ‘N_’) that we applied earlier for accurate size measurement in phase 2.

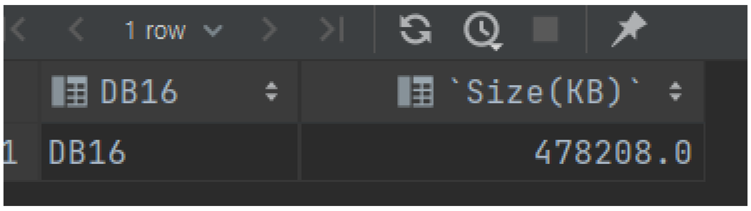
6.1 Database size

: 478208.0 KB

query

```
SELECT table_schema AS 'DB16',
ROUND(SUM(data_length+index_length)/1024, 1) AS 'Size(KB)'
FROM information_schema.tables
WHERE table_schema = 'DB16'
 AND TABLE_NAME LIKE 'N\_%';
```

result



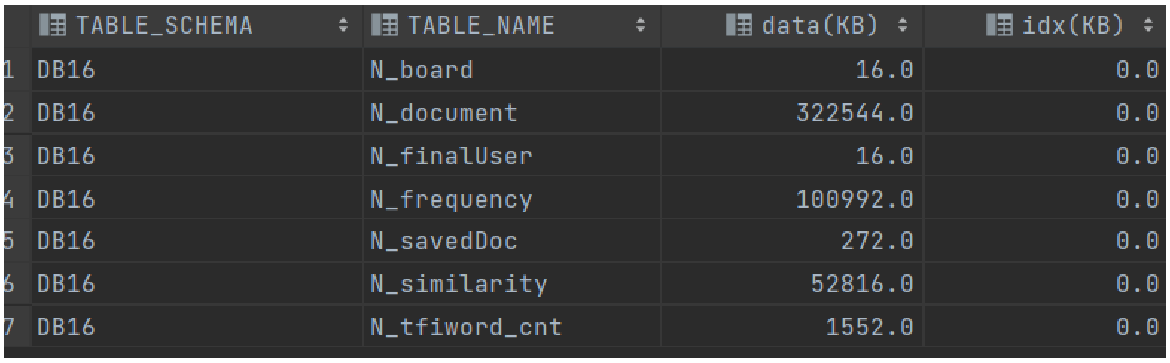
```
1 row
   DB16        `Size(KB)`
1  DB16          478208.0
```

6.2 Table size

query

```
select table_schema,
      table_name,
      round(data_length / (1024), 1)  as 'data(KB)',
      round(index_length / (1024), 1)  as 'idx(KB)'
from information_schema.tables
where table_type = 'BASE TABLE'
 and table_schema = 'DB16'
 AND TABLE_NAME LIKE 'N\_%';
```

result



	TABLE_SCHEMA	TABLE_NAME	data(KB)	idx(KB)
1	DB16	N_board	16.0	0.0
2	DB16	N_document	322544.0	0.0
3	DB16	N_finalUser	16.0	0.0
4	DB16	N_frequency	100992.0	0.0
5	DB16	N_savedDoc	272.0	0.0
6	DB16	N_similarity	52816.0	0.0
7	DB16	N_tfiword_cnt	1552.0	0.0

Although the index was used, we guess it was not reflected in the information schema.