Project Report for 23-1 Semester Database Class 2

Phase 2. Data Optimizing

21300024 Hyeon-myeong Kang

21400659 Jae-geun Jang

21900543 Won-bin Lee

1. Direction of attacking this problem

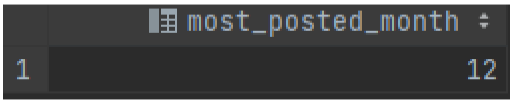
   We attacked this problem with the following steps.

   1) Implementing Queries
   2) Analyzing queries for indexing
      a) Which attribute is used frequently, especially in WHERE, ORDER clause and so on.
      b) Dilemma; similarity and frequency tables
   3) Indexing
      a) List of indexes
      b) Executing time comparison

2. Query Implementation
   i. 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 :

```
SELECT month most_posted_month
FROM (SELECT month,
            RANK() OVER (ORDER BY AVG(cnt) DESC) RANKING
     FROM (SELECT YEAR(post_date) year, MONTH(post_date) month, count(*)
cnt
           FROM DB34.post
           GROUP BY year, month
           HAVING year IS NOT NULL) y_m_posted
     GROUP BY month) temp
WHERE RANKING = 1;
```

      ○ Result :

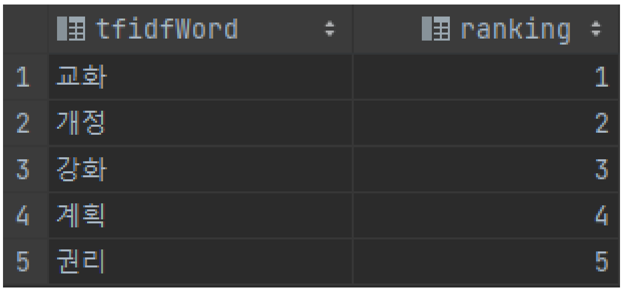


	most_posted_month
1	12

   ii. Find the 5 most important keywords (in terms of TFIDF) in the document that is bookmarked (saved) by the most users.
      ○ Query:

```
SELECT tfidfWord, ranking
FROM (
   SELECT tfidfWord, score, rank() over (order by Score desc) as ranking
   FROM frequency
   WHERE docID in (
       SELECT savedDocHashKey
       FROM (
           SELECT savedDocHashKey, rank() OVER (ORDER BY COUNT(*) DESC) as
ranking, years
           FROM DB34.savedPost
           LEFT JOIN (SELECT hash_key, Year(post_date) as years
                      FROM DB34.post) as D
               on D.hash_key = savedDocHashKey
           WHERE email IS NOT NULL and years = 2011
           GROUP BY savedDocHashKey) C
       WHERE ranking = 1)
   ) B
WHERE ranking <= 5;
```

- Result :



	tfidfWord	ranking
1	교화	1
2	개정	2
3	강화	3
4	계획	4
5	권리	5

iii. 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.

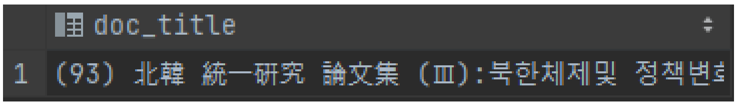
- Query :

```
SELECT doc_title
FROM DB34.post
WHERE hash_key IN (SELECT rcmdDocID
                  FROM (SELECT rcmdDocID, RANK() OVER (ORDER BY Score DESC)
Score_RANK
                        FROM similarity
                                 JOIN (SELECT savedDocHashKey
                                       FROM (SELECT savedDocHashKey,
                                                    count(*)
savedCNT,
                                                    RANK() OVER (ORDER BY
count(*)) LEAST_RANKING
                                             FROM DB34.savedPost
                                             WHERE email IS NOT NULL
                                               AND savedUser LIKE
'%@handong.ac.kr'

                                             GROUP BY savedDocHashKey
                                             ORDER BY savedCNT) a
                                       WHERE LEAST_RANKING = 1) b
```

```
                                      ON similarity.docID =
b.savedDocHashKey AND Score <> 0) b
                              WHERE Score_RANK = 2 # Since the first rank is for the
same document with taget document (document to be compared)
);
```

- Result :



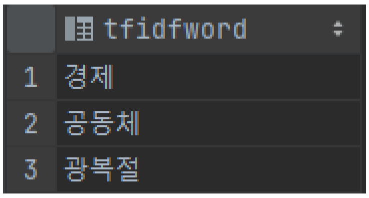
	doc_title
1	(93) 北韓 統一研究 論文集 (Ⅲ):북한체제및 정책변

iv. 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 :

```
SELECT tfidfWord, Score
FROM frequency
WHERE docID = (
   SELECT hash_key
   FROM DB34.post
   LEFT JOIN DB34.savedPost ON hash_key = savedDocHashKey
   WHERE post_writer = '조한범'
   GROUP BY hash_key
   ORDER BY COUNT(savedDocHashKey) DESC
   LIMIT 1, 1
)
ORDER BY Score DESC
LIMIT 3;
```

- Result : In the process of solving the problem, when we wrote a subquery to extract the second most frequently saved document among '조한범's posts using the rank() function, I confirmed that two results were output. Consequently, we decided to identify the three most important keywords for these two documents. However, during this process, we encountered a keyword duplication issue where common keywords were found in both documents. To address this, we used the window function's partition to retain only the duplicated keywords with the highest Score. Through this, we were able to implement a query that outputs only the keyword with the higher Score when a duplicated Keyword appears.



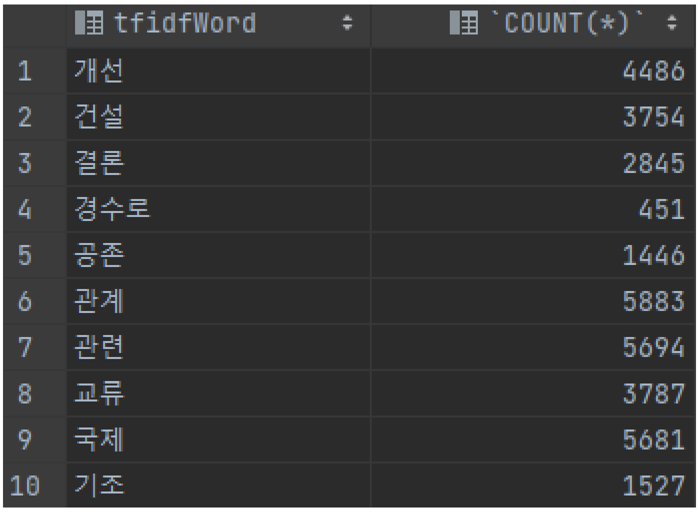
	tfidfword
1	경제
2	공동체
3	광복절

v. 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 :

```
SELECT tfidfWord, COUNT(*)
FROM DATA.frequency
GROUP BY tfidfWord;
```

- Result :



	tfidfWord	`COUNT(*)`
1	개선	4486
2	건설	3754
3	결론	2845
4	경수로	451
5	공존	1446
6	관계	5883
7	관련	5694
8	교류	3787
9	국제	5681
10	기조	1527

(and so on)

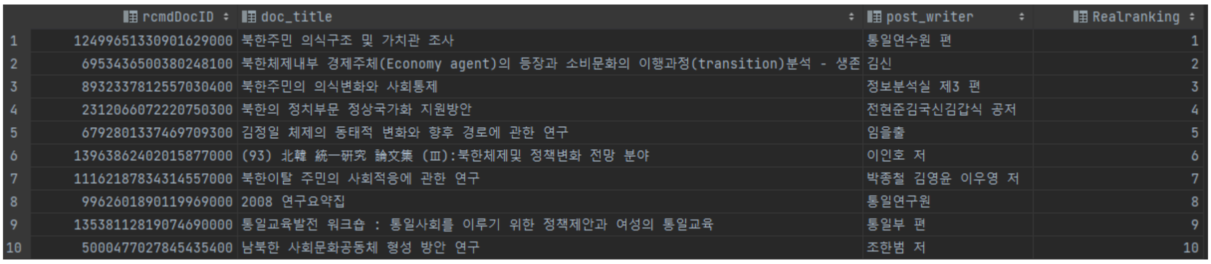
vi. Find the ten most similar documents to those of the author who holds the most representative document for keyword “개인.” • Consider only the documents who have records in the frequency and similarity tables

- Query :

```
WITH compare_table AS (
 SELECT hash_key, doc_title, post_writer, tfidfWord, score,
   RANK() OVER (ORDER BY Score DESC) as ranking
 FROM DB34.post
 JOIN (
   SELECT docID, docTitle, tfidfWord, Score
   FROM frequency
   JOIN (
     SELECT distinct docID as simID
     FROM similarity
   ) C ON frequency.docID = simID
   WHERE tfidfWord = '개인'
 ) D ON D.docID = DB34.post.hash_key
)
SELECT rcmdDocID, doc_title, post_writer, RANK() OVER (ORDER BY Rranking)
as Realranking
```

```
FROM DB34.post
JOIN (
 SELECT rcmdDocID, RANK() OVER (ORDER BY SCORE DESC) as Rranking
 FROM similarity
 WHERE docID IN (
   SELECT hash_key
   FROM DB34.post
   WHERE post_writer IN (
     SELECT post_writer
     FROM compare_table
     WHERE compare_table.ranking = 1
   )
   AND hash_key IN (
     SELECT hash_key
     FROM compare_table
   )
 )
 AND docID <> similarity.rcmdDocID
) D ON post.hash_key = rcmdDocID
LIMIT 10;
```

- ○ Result :



	rcmdDocID	doc_title	post_writer	Realranking
1	12499651330901629000	북한주민 의식구조 및 가치관 조사	통일연수원 편	1
2	6953436500380248100	북한체제내부 경제주체(Economy agent)의 등장과 소비문화의 이행과정(transition)분석 - 생존	김신	2
3	8932337812557030400	북한주민의 의식변화와 사회통제	정보분석실 제3 편	3
4	2312066072220750300	북한의 정치부문 정상국가화 지원방안	전현준김국신김갑식 공저	4
5	6792801337469709300	김정일 체제의 동태적 변화와 향후 경로에 관한 연구	임을출	5
6	13963862402015877000	(93) 北韓 統一硏究 論文集 (Ⅲ):북한체제및 정책변화 전망 분야	이인호 저	6
7	11162187834314557000	북한이탈 주민의 사회적응에 관한 연구	박종철 김영윤 이우영 저	7
8	9962601890119969000	2008 연구요약집	통일연구원	8
9	13538112819074690000	통일교육발전 워크숍 : 통일사회를 이루기 위한 정책제안과 여성의 통일교육	통일부 편	9
10	5000477027845435400	남북한 사회문화공동체 형성 방안 연구	조한범 저	10

vii. 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.

- ○ Query :

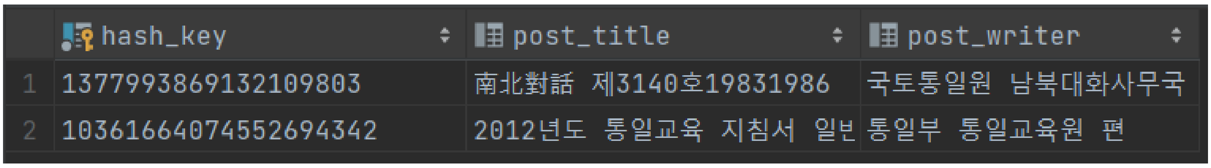
```
WITH select_Two AS (SELECT docID, rcmdDocID
FROM (SELECT docID, rcmdDocID, RANK() OVER (ORDER BY SCORE desc) as rnk
FROM similarity
WHERE docID in (SELECT docID #result = 16277291468130386000
FROM (SELECT docID, score, rank() over (order by Score DESC) as ranking2
FROM (SELECT *
FROM frequency
WHERE frequency.docID in (SELECT hash_key FROM DB34.post)
AND frequency.docID in (SELECT docID FROM similarity)) B
WHERE tfidfWord in (SELECT tfidfWord
FROM (SELECT tfidfWord,
count(*),
rank() over (ORDER BY count(*) DESC) as ranking
FROM DATA.frequency
GROUP BY tfidfWord) C
WHERE ranking = 10)) D
WHERE ranking2 = 200)) E
WHERE rnk = 7)

SELECT hash_key, post_title, post_writer
FROM DB34.post
WHERE hash_key in (SELECT docID FROM select_Two)

UNION

SELECT hash_key, post_title, post_writer
FROM DB34.post
WHERE hash_key in (SELECT rcmdDocID FROM select_Two);
```

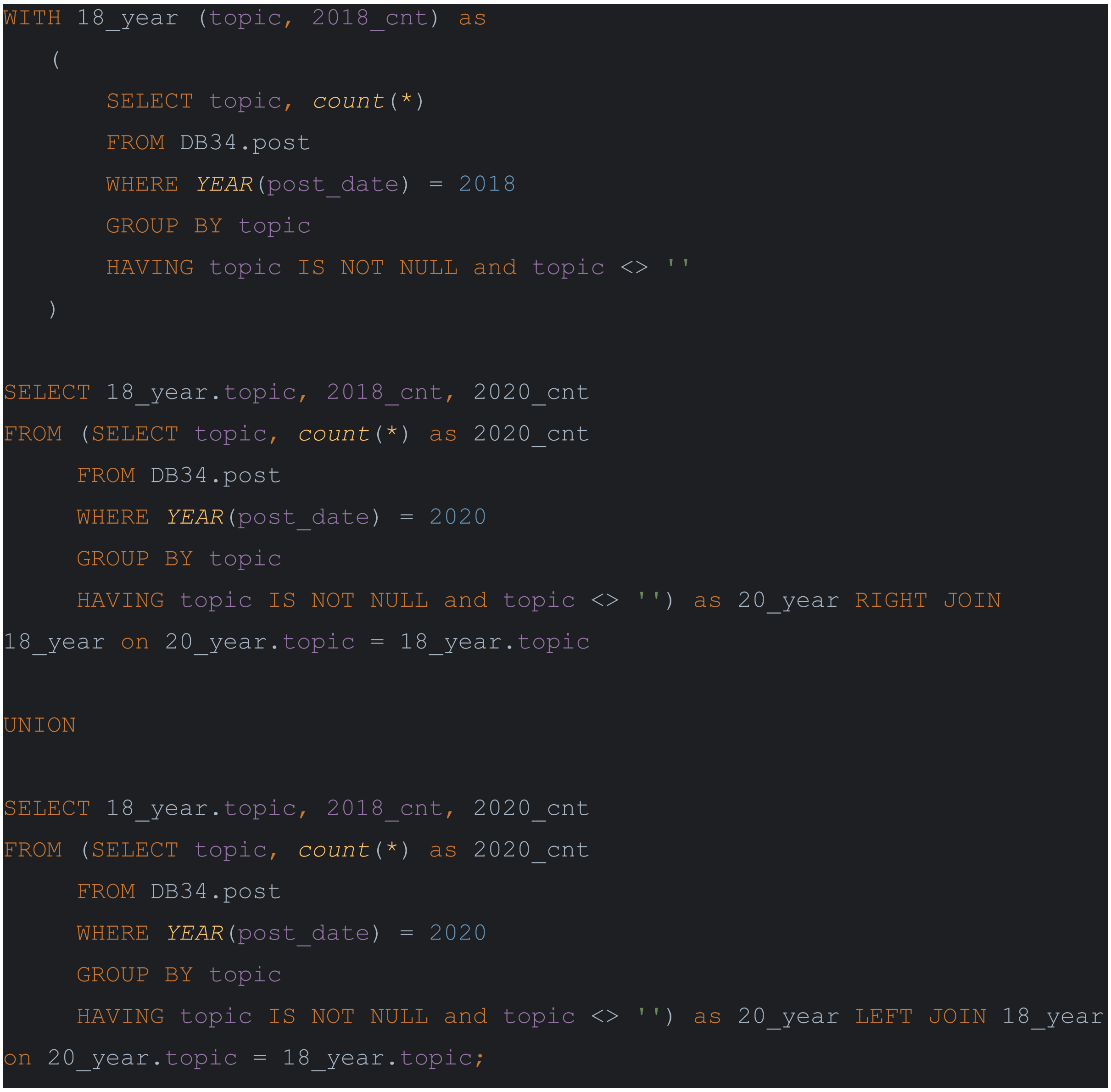
- Result : When simply extracting the 7th and 200th values from the frequency table, an issue arose where these values did not exist in the post table. As a result, I obtained results under the assumption that values existed in both the frequency table and the post table.



	hash_key	post_title	post_writer
1	1377993869132109803	南北對話 제3140호19831986	국토통일원 남북대화사무국
2	10361664074552694342	2012년도 통일교육 지침서 일반	통일부 통일교육원 편

viii. Compare the topic distribution among the documents published in 2018 and 2022, respectively.

- Query :



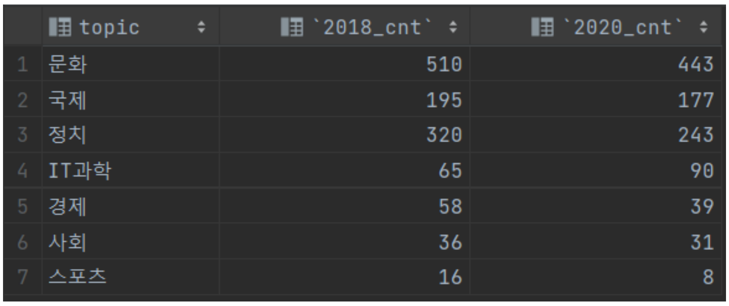
```
WITH 18_year (topic, 2018_cnt) as
   (
       SELECT topic, count(*)
       FROM DB34.post
       WHERE YEAR(post_date) = 2018
       GROUP BY topic
       HAVING topic IS NOT NULL and topic <> ''
   )

SELECT 18_year.topic, 2018_cnt, 2020_cnt
FROM (SELECT topic, count(*) as 2020_cnt
     FROM DB34.post
     WHERE YEAR(post_date) = 2020
     GROUP BY topic
     HAVING topic IS NOT NULL and topic <> '') as 20_year RIGHT JOIN
18_year on 20_year.topic = 18_year.topic

UNION

SELECT 18_year.topic, 2018_cnt, 2020_cnt
FROM (SELECT topic, count(*) as 2020_cnt
     FROM DB34.post
     WHERE YEAR(post_date) = 2020
     GROUP BY topic
     HAVING topic IS NOT NULL and topic <> '') as 20_year LEFT JOIN 18_year
on 20_year.topic = 18_year.topic;
```

- Result : During the process of deriving the results, I found two cases: where the topic was null, and where it was blank. I considered these cases unrelated to the output of the topic and hence excluded them from the final values.



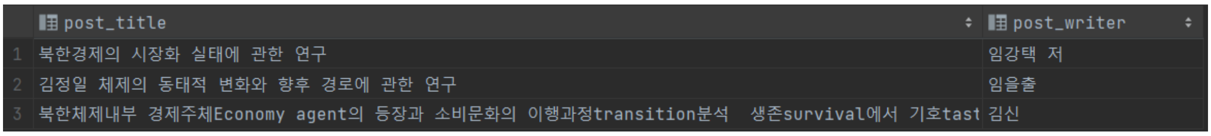
topic	`2018_cnt`	`2020_cnt`
문화	510	443
국제	195	177
정치	320	243
IT과학	65	90
경제	58	39
사회	36	31
스포츠	16	8

ix. Find the titles (post_title) and authors (post_writer) of the three most similar documents (regardless of the published year) to the document with the longest title among those published in 2018

- Query :

```
SELECT post_title, post_writer
FROM DB34.post
JOIN (
      SELECT rcmdDocID
      FROM similarity
      WHERE docID = (
            SELECT hash_key
            FROM DB34.post
            WHERE YEAR(post_date) = 2018
            ORDER BY LENGTH(post_title) DESC
            LIMIT 9, 1
            )
      ORDER BY score DESC
      LIMIT 3
) AS sub
ON hash_key = rcmdDocID;
```

- Result :



	post_title	post_writer
1	북한경제의 시장화 실태에 관한 연구	임강택 저
2	김정일 체제의 동태적 변화와 향후 경로에 관한 연구	임을출
3	북한체제내부 경제주체Economy agent의 등장과 소비문화의 이행과정transition분석 생존survival에서 기호tast	김신

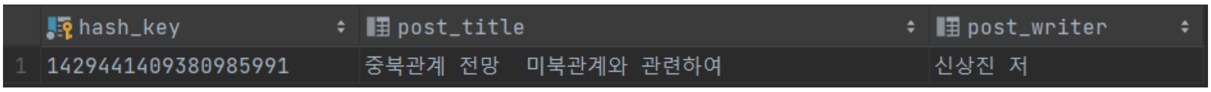
x. Among the documents whose titles start with "ㅈ”, Find the ID, title, and author name of the document with the highest tfidf importance for the keyword "관계".

- Query :

```
SELECT hash_key, post_title, post_writer
FROM DB34.post
WHERE hash_key in (
       SELECT docID
       FROM (SELECT docID, score, RANK() OVER (ORDER BY Score DESC) as
ranking
             FROM (SELECT hash_key
                   FROM DB34.post
                   WHERE post_title_first_char = 'ㅈ') as first
                 LEFT JOIN frequency on first.hash_key = frequency.docID
             WHERE tfidfWord = '관계') C
```

```
      WHERE ranking = 1);
```

- Result:



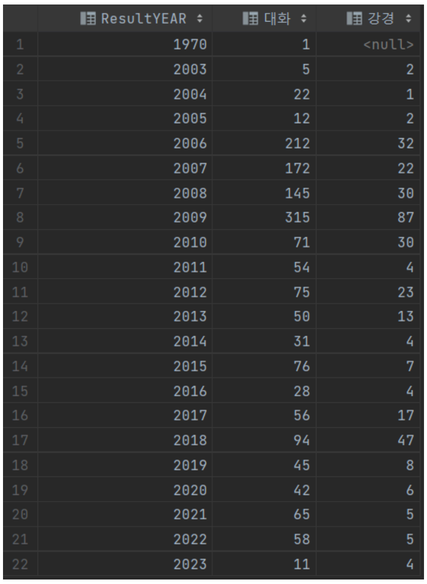
	hash_key	post_title	post_writer
1	1429441409380985991	중북관계 전망　미북관계와 관련하여	신상진 저

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

- Query :

```
SELECT speak.year as ResultYEAR, speak.cnt as 대화, strick.cnt as 강경
FROM ( SELECT YEAR(post_date) as year, COUNT(*) as cnt
      FROM DB34.post
      WHERE post_body LIKE '%대화%'
      GROUP BY YEAR(post_date)) as speak LEFT JOIN
     ( SELECT YEAR(post_date) as year, COUNT(*) as cnt
       FROM DB34.post
       WHERE post_body LIKE '%강경%'
       GROUP BY YEAR(post_date)) as strick
     ON speak.year = strick.year
WHERE speak.year is not null
ORDER BY ResultYEAR;
```

- Result : During the process of retrieving the values, there were instances where the year was null. We considered these cases irrelevant to the yearly analysis that the problem required, so despite the presence of values in the 'conversation' column, we deleted the row.



	ResultYEAR	대화	강경
1	1970	1	<null>
2	2003	5	2
3	2004	22	1
4	2005	12	2
5	2006	212	32
6	2007	172	22
7	2008	145	30
8	2009	315	87
9	2010	71	30
10	2011	54	4
11	2012	75	23
12	2013	50	13
13	2014	31	4
14	2015	76	7
15	2016	28	4
16	2017	56	17
17	2018	94	47
18	2019	45	8
19	2020	42	6
20	2021	65	5
21	2022	58	5
22	2023	11	4

3. Analyzing queries for indexing

   i. Which attribute is used frequently, especially in WHERE, ORDER clause and so on.

      ○ **‘post_date’ from the ‘post’ table**
        This column is used frequently in filtering and GROUP BY clause across several queries, so we thought indexing this column would speed up these operations.

      ○ **‘hash_key’ from the ‘post’ table**
        This column is used in multiple joins and filter operations. We thought indexing this would speed up those operations.

      ○ **‘savedDocHashKey’ from the ‘savedPost’**
        Same reason with ‘hash_key’

      ○ **‘hash_key’, ‘post_date’**
        Query1 frequently accesses these two columns together. It first groups by year and month derived from post_date and then ranks by hash_key. Having a combined index can help here.

      ○ **‘post_writer’, ‘hash_key’**
        This combination is used frequently in sub-queries.

   ii. Dilemma; similarity and frequency tables
       : We were able to copy both the similarity and frequency tables into our database so that we can index to those tables. However, we decided to omit this process because the increase in memory size outweighed the speed improvement it would provide.

4. Indexing

i. List of indexes

```sql
CREATE INDEX post_date ON DB34.post(post_date);
CREATE INDEX post_hash ON DB34.post(hash_key);
CREATE INDEX savedhash ON DB34.savedPost(savedDocHashKey);
CREATE INDEX post_hashDate ON DB34.post(hash_key, post_date);
CREATE INDEX post_writerHash ON DB34.post(post_writer, hash_key);
```

ii. Executing time comparison

Query Number	Before	After
1	15	14
2	16	15
3	18	17
4	22	21
5	14	14
6	20	19
7	29	24
8	16	13
9	17	15
10	18	16
11	19	16

(scale : ms)

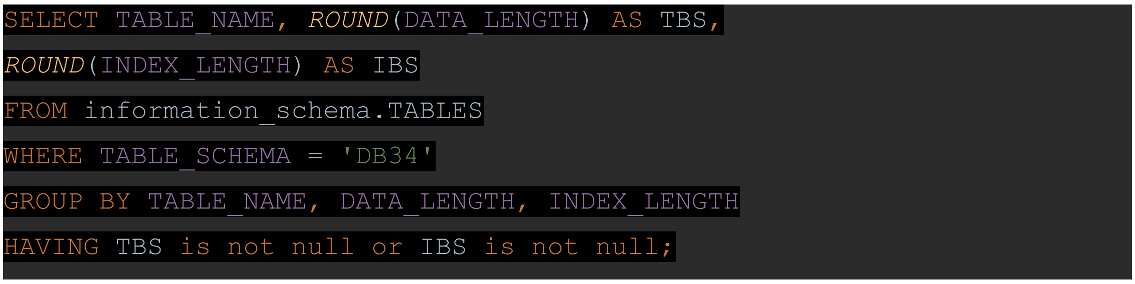
5. Result and Evaluation

i. Performance Changes before and after Indexing Implementation

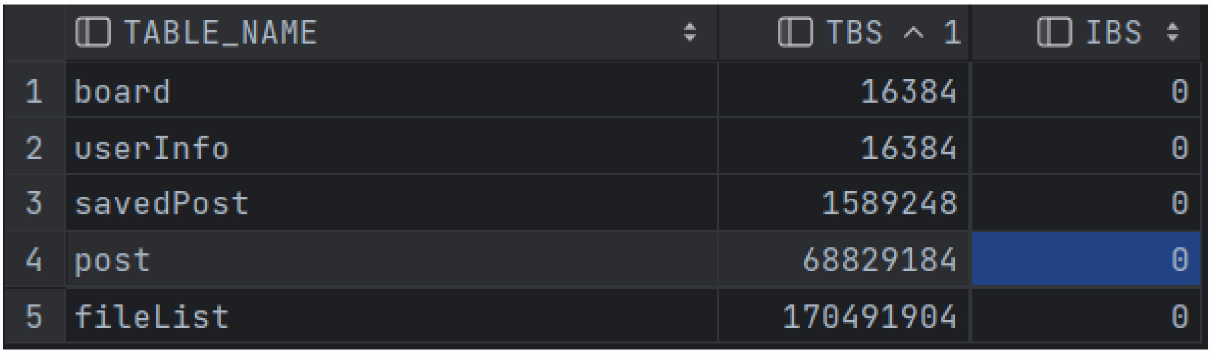
No significant time change was observed after implementing indexing, possibly due to the following reasons:

Case 1: The processing time for the query was already very quick, so no substantial speed change could be observed in the processing time before and after using the index.
Case 2: The database could not utilize the index for an unknown reason.
Case 3: Although functions like COUNT(*) and IN used indexes, substantial time was consumed for subsequent query processing. Therefore, the indexes did not efficiently reduce the processing time.

ii. Comparative Analysis of Database Size Changes and Summary of Table Sizes (KB)
We used the query below to observe the size of the indexes we implemented.



```
SELECT TABLE_NAME, ROUND(DATA_LENGTH) AS TBS,
ROUND(INDEX_LENGTH) AS IBS
FROM information_schema.TABLES
WHERE TABLE_SCHEMA = 'DB34'
GROUP BY TABLE_NAME, DATA_LENGTH, INDEX_LENGTH
HAVING TBS is not null or IBS is not null;
```



	TABLE_NAME	TBS	IBS
1	board	16384	0
2	userInfo	16384	0
3	savedPost	1589248	0
4	post	68829184	0
5	fileList	170491904	0

Judging from the above result, it seems that the size could not be displayed because the index was not recognized in the database system.