Term Project Phase 2 Report

Team 14
21400714 조세형
21700147 김은택
21900395 신소은

1. Introduction

The purpose of this report is to describe how to design and implement a database instance that is efficient in time. In phase 1, normalization was performed to increase the write performance, but in this case, the read performance is decreased because we should perform many join operations when we read the data. Even if we give up some of the write performance, we can consider denormalization which reduces join operations by allowing duplicate data in the table. Denormalization increases read performance because it performs relatively few join operations but requires extra storage space and additional execution time for updates. It also decreases write performance due to extra coding work and increases the possibility of errors in extra code. Our group members proceeded with phase 2 in the following order.

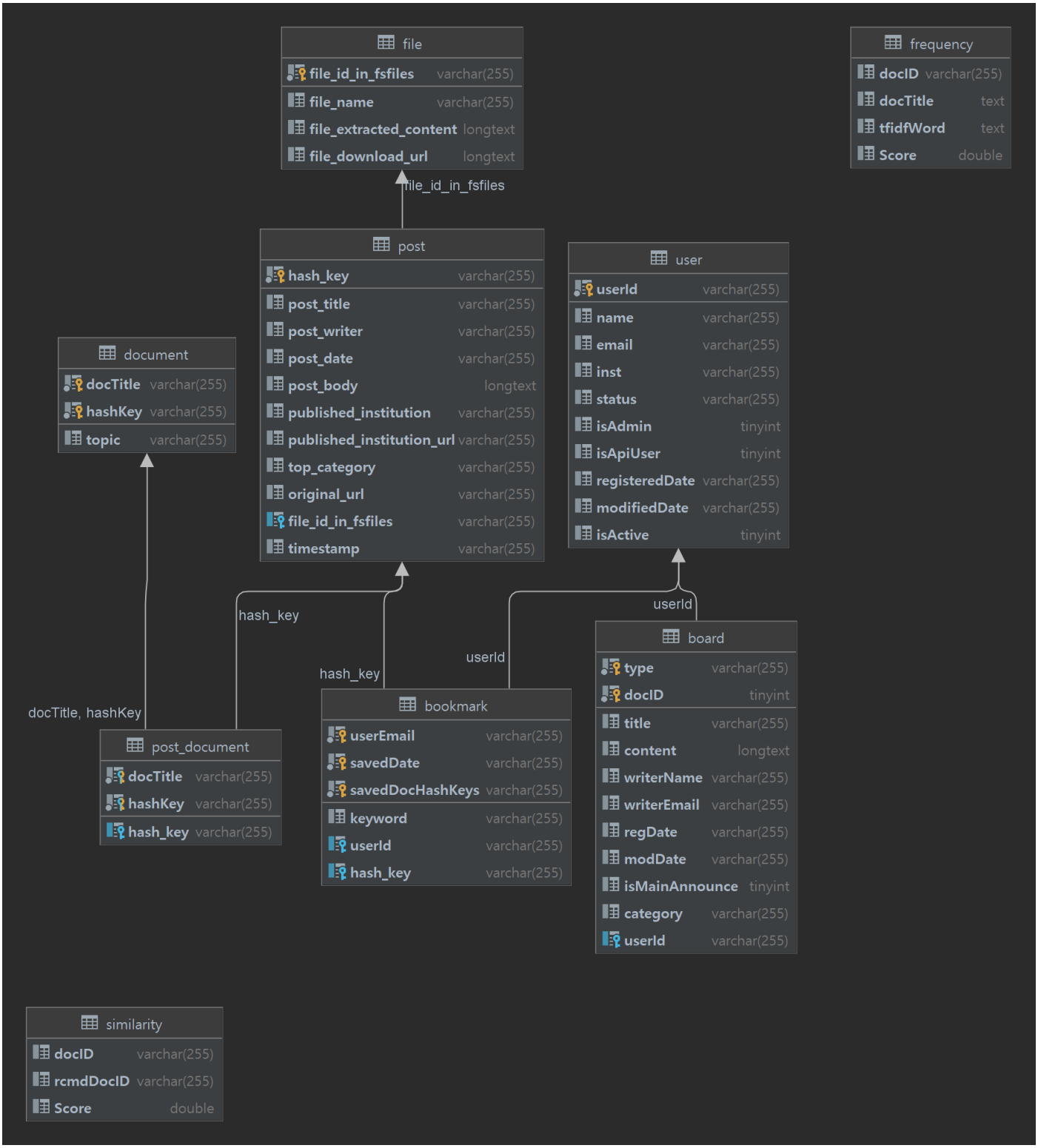
1. We reflected on the feedback we received in phase 1.
2. We implemented 10 queries presented in phase 2.
3. We considered and designed how to perform denormalization efficiently through queries we created.
4. We considered and designed how to create indexes efficiently through queries we created.

We divided phase 2 into 3 steps of work. Reflect feedback, perform denormalization, and create indexes. We will present queries in step 1, and step 2 respectively because the structures of the tables and queries have been changed after denormalization. The details will be explained below.

2. Step 1: Reflect Feedback

First, we reflected on the feedback we received in phase 1 before performing denormalization and creating indexes. We received feedback that the table related to the *bookmark* table is unnecessarily separated, and the *board* table can be connected directly to the *user* table, so there is no need to make a separate table. Reflecting the feedback, we merged the *bookmark_post* table with the *bookmark* table and the *user_board* table with the *board* table. Through the process of reflecting on the feedback, we were able to check the normalization that we performed in phase 1 and design a better database. The E-R diagrams after reflecting on the feedback are in [Figure 1].

Next, we implemented 10 queries presented in phase 2. The queries described here are the queries before performing denormalization. After performing denormalization, since the structures of the tables have been changed, the queries also have to be changed. We will introduce the solution queries, results, and execution times on the COSS' DBMS from [Figure 2] to [Figure 32].



[Figure 1] E-R diagram after reflecting on the feedback

```sql
select topic, count(*) as topic_count,
       rank() over (order by count(*) desc) as topic_rank
from document
where topic is not null
group by topic;
```

[Figure 2] Solution query of problem 1

	topic	topic_count	topic_rank
1	문화	5204	1
2	정치	1430	2
3	경제	761	3
4	사회	580	4
5	국제	536	5
6	IT_과학	449	6
7	스포츠	58	7

[Figure 3] Result of problem 1

```
[2022-06-20 08:57:56] 7 rows retrieved starting from 1 in 227 ms (execution: 50 ms, fetching: 177 ms)
```

[Figure 4] Execution time of problem 1

```sql
select count(hash_key) as document_count
from post
where post_date between '2019' and '2022-12-31';
```

[Figure 5] Solution query of problem 2

	document_count
1	2595

[Figure 6] Result of problem 2

```
[2022-06-20 09:00:04] 1 row retrieved starting from 1 in 118 ms (execution: 79 ms, fetching: 39 ms)
```

[Figure 7] Execution time of problem 2

```sql
select userEmail, count(*) as bookmark_count
from bookmark
where savedDate like '2022%'
group by userEmail
order by bookmark_count desc
limit 1;
```

[Figure 8] Solution query of problem 3

	userEmail	bookmark_count
1	jerrySahara@hgu.db2022.com	80

[Figure 9] Result of problem 3

```
[2022-06-20 09:00:47] 1 row retrieved starting from 1 in 111 ms (execution: 78 ms, fetching: 33 ms)
```

[Figure 10] Execution time of problem 3

```
select keyword, keyword_count
from (select keyword, keyword_count,
             rank() over (order by keyword_count desc) as keyword_rank
      from (select keyword, count(*) as keyword_count
            from bookmark
            where userEmail = 'jerrySahara@hgu.db2022.com'
            group by keyword) as A) as B
where B.keyword_rank <= 5;
```

[Figure 11] Solution query of problem 4

	keyword	keyword_count
1	평화	50
2	국제	35
3	고려	29
4	계획	29
5	기획	28
6	과제	28
7	개방	28

[Figure 12] Result of problem 4

```
[2022-06-20 09:01:17] 7 rows retrieved starting from 1 in 83 ms (execution: 45 ms, fetching: 38 ms)
```

[Figure 13] Execution time of problem 4

In problem 4, we listed the 7 keywords because 3 keywords are found the same number of times. If we list only 5 keywords, there is no standard for selecting among the 3 keywords, so we listed them based on rank.

```
with longest_title_2014 as
(select hash_key, post_title, post_writer
from post
where length(post_title) = (select max(length(post_title))
                            from post
                            where post_date like '2014%')
      and post_date like '2014%'
group by hash_key)
select post_title, post_writer
from similarity join post on similarity.rcmdDocID = post.hash_key
where docID = (select hash_key
               from longest_title_2014) and docID <> rcmdDocID
order by Score desc
limit 3;
```

[Figure 14] Solution query of problem 5

	post_title	post_writer
1	中國-臺灣間 交流.協力 現況	孫仁燮
2	북한 외화벌이 추세와 전망	김석진
3	주간국방논단 제1624-2호: 국방 분야 민간지원 활용에 있어, 호주의 '성과기반계약(PBC)' 제도 분	장지홍, 김진호

[Figure 15] Result of problem 5

```
[2022-06-20 09:02:03] 3 rows retrieved starting from 1 in 777 ms (execution: 741 ms, fetching: 36 ms)
```

[Figure 16] Execution time of problem 5

```
select name, inst, email, status
from user
where status in (select status
                 from user
                 group by status
                 having count(*) = 1);
```

[Figure 17] Solution query of problem 6

	name	inst	email	status
1	Charles Johnson	한동대학교	charlesJohnson@hgu.db2022.com	박사
2	Reginald Spaulding	한동대학교	reginaldSpaulding@hgu.db2022.com	연구원
3	Jesse Kubik	재단법인통일과ㄴ	jesseKubik@hgu.db2022.com	기타
4	David Dugue	한동대학교	davidDugue@hgu.db2022.com	석사

[Figure 18] Result of problem 6

```
[2022-06-20 09:02:36] 4 rows retrieved starting from 1 in 61 ms (execution: 22 ms, fetching: 39 ms)
```

[Figure 19] Execution time of problem 6

```
select tfidfWord
from post join frequency on post.hash_key = frequency.docID
where post_writer like '%송인호%'
order by Score desc
limit 5;
```

[Figure 20] Solution query of problem 7

	tfidfWord
1	급속
2	결론
3	가치
4	개념
5	개선

[Figure 21] Result of problem 7

```
[2022-06-20 09:03:38] 5 rows retrieved starting from 1 in 968 ms (execution: 938 ms, fetching: 30 ms)
```

[Figure 22] Execution time of problem 7

```
select post_title, post_writer, post_date
from similarity join post on similarity.rcmdDocID = post.hash_key
where docID in (select hash_key
                from post
                where instr(post_writer, (select post_writer
                                          from frequency join post
                                               on frequency.docID = post.hash_key
                                          where tfidfWord = '국민'
                                          order by Score desc
                                          limit 1)) > 0) and docID <> rcmdDocID
order by Score desc
limit 5;
```

[Figure 23] Solution query of problem 8

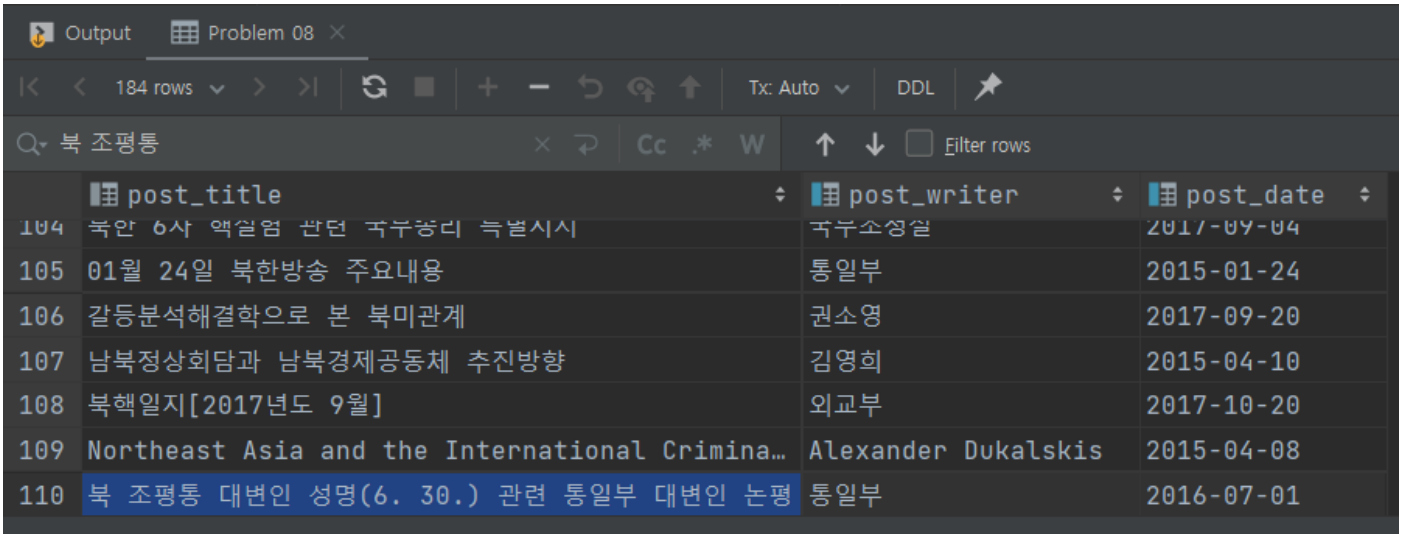
	post_title	post_writer	post_date
1	개성공단 기업 경협보험금 조기 지급을 위한 교추협 의결	통일부	2016-02-21
2	07월 22일 북한방송 주요내용	통일부	2015-07-22
3	[영문주제강좌] North Korea´s Planned Economy and Marketization	통일부 통일교육원	2015-09-17
4	06월 21일 북한방송 주요내용	통일부	2015-06-22
5	평화통일 국민공감대 인천세미나 인사말	통일준비위원회	2015-11-26

[Figure 24] Result of problem 8

```
[2022-06-20 09:04:21] 5 rows retrieved starting from 1 in 1 s 892 ms (execution: 1 s 863 ms, fetching: 29 ms)
```

[Figure 25] Execution time of problem 8

In problem 8, our result is different from the sample result. To resolve this problem, find the author who holds the most representative document for the keyword '국민' first. The author we found is '김준형'. Second, find the documents of the author '김준형'. We can find 5 documents and the hash keys of the documents which are '1337465753066586767', '18008246883532771190', '5795239884110089394', '623850667958864873', and '8441526156624636806'. Among these documents, only '5795239884110089394' is in the *similarity* table. However, there are 184 most similar documents because the scores of the 184 documents are all the same by 1. If we remove the *limit* clause, we can see the documents in the sample result. An example is represented in [Figure 26].



	post_title	post_writer	post_date
104	북한 6차 핵실험 관련 국무총리 특별지시	국무조정실	2017-09-04
105	01월 24일 북한방송 주요내용	통일부	2015-01-24
106	갈등분석해결학으로 본 북미관계	권소영	2017-09-20
107	남북정상회담과 남북경제공동체 추진방향	김영희	2015-04-10
108	북핵일지[2017년도 9월]	외교부	2017-10-20
109	Northeast Asia and the International Crimina…	Alexander Dukalskis	2015-04-08
110	북 조평통 대변인 성명(6. 30.) 관련 통일부 대변인 논평	통일부	2016-07-01

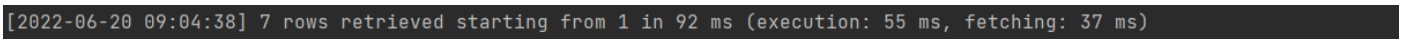
[Figure 26] An example of the most similar documents in problem 8

```
select document.topic,
       count(case when post.post_date like '2015%' then 1 end) as 'CNT2015',
       count(case when post.post_date like '2020%' then 1 end) as 'CNT2020'
from post join document on post.hash_key = document.hashKey
where document.topic is not null
group by document.topic;
```

[Figure 27] Solution query of problem 9

	topic	CNT2015	CNT2020
1	문화	100	623
2	사회	4	5
3	IT_과학	27	86
4	정치	114	50
5	국제	7	8
6	스포츠	2	0
7	경제	20	4

[Figure 28] Result of problem 9



[Figure 29] Execution time of problem 9

```
select tfidfWord, count(*) as word_count,
       rank() over (order by count(*) desc) word_rank
from frequency
group by tfidfWord;
```

[Figure 30] Solution query of problem 10

	tfidfWord	word_count	word_rank
1	경제	6911	1
2	가능	6377	2
3	국가	6252	3
4	결과	5884	4
5	관계	5883	5
6	경우	5845	6

[Figure 31] Result of problem 10

[Figure 32] Execution time of problem 10

3. Step 2: Perform Denormalization

3.1. Design How to Perform Denormalization

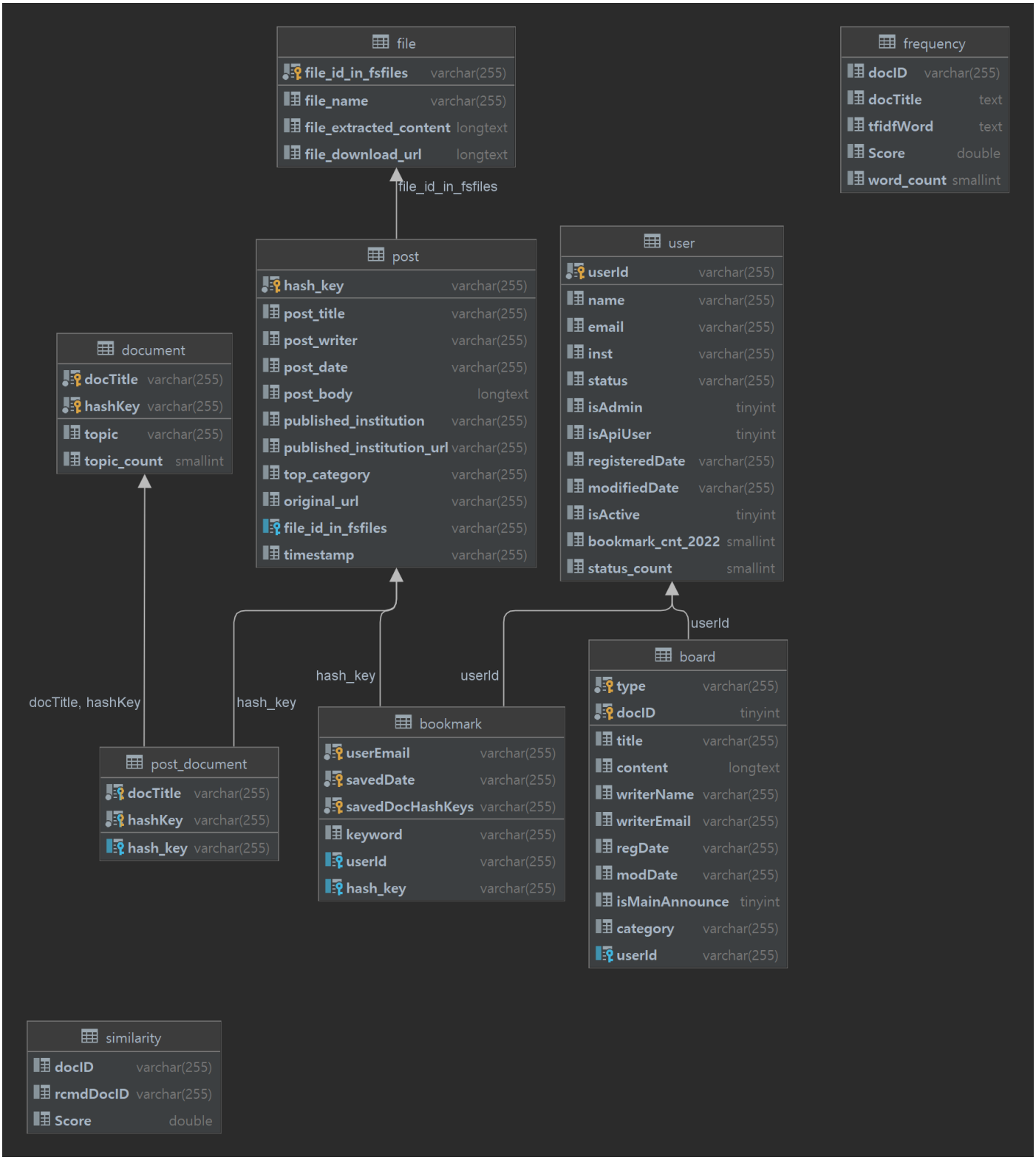
Next, we considered and designed how we could perform denormalization efficiently through the queries we implemented. Among the methods of denormalization, we used two ways to reduce join operations and calculate operations by manipulating columns. Firstly, for example, when a user uses the *university* database and needs the information about *course_id*, *title*, and *prereq_id*, the *course* table and *prereq* table should be joined because of the *title*. However, if the *prereq_id* column of the *prereq* table is added to the *course* table, the user can make a query by using only the *course* table. Secondly, when a user wants to make a query to show the number of sections open in each year in the *university* database, they should perform **group by** and **count** operations on the *year* column in the *section* table. However, if we add a column called *year_section_count*, which stores the number of sections open in each year, to the *section* table, the user can use *year*, *year_section_count* to query the desired result without performing operations. These are the two methods that we used to perform denormalization, and the detailed application process for each query is as follows.

- Problem 1: Create a *topic_count* column and add it to the *document* table.
- Problem 3: Create a *bookmark_cnt_2022* column and add it to the *user* table.
- Problem 6: Create a *status_count* column and add it to the *user* table.
- Problem 10: Create a *word_count* column and add it to the *frequency* table.

The above process decreases write performance because the user should update the new column whenever new data is added into the table. However, the read performance increases because the desired data can be queried immediately without performing extra operations.

3.2. Implement Denormalization

After performing denormalization with the above process, we can get the new E-R diagram because the structure of the tables is changed. The new E-R diagram is represented in [Figure 33].



[Figure 33] The new E-R diagram by performing denormalization

We can check that the new columns were added through the new E-R diagram in [Figure 33]. Therefore, the solution queries of the problems 1, 3, 6, and 10 should be changed. The new solution queries of them are represented in [Figure 34] to [Figure 45].

```
select distinct topic, topic_count,
       dense_rank() over (order by topic_count desc) as topic_rank
from document
where topic is not null;
```

[Figure 34] Solution query of problem 1 after performing denormalization

	topic	topic_count	topic_rank
1	문화	5204	1
2	정치	1430	2
3	경제	761	3
4	사회	580	4
5	국제	536	5
6	IT_과학	449	6
7	스포츠	58	7

[Figure 35] Result of problem 1 after performing denormalization

```
[2022-06-20 09:15:48] 7 rows retrieved starting from 1 in 100 ms (execution: 65 ms, fetching: 35 ms)
```

[Figure 36] Execution time of problem 1 after performing denormalization (227ms to 100ms)

```
select email, bookmark_cnt_2022
from user
order by bookmark_cnt_2022 desc
limit 1;
```

[Figure 37] Solution query of problem 3 after performing denormalization

	email	bookmark_cnt_2022
1	jerrySahara@hgu.db2022.com	80

[Figure 38] Result of problem 3 after performing denormalization

```
[2022-06-20 09:16:17] 1 row retrieved starting from 1 in 63 ms (execution: 31 ms, fetching: 32 ms)
```

[Figure 39] Execution time of problem 3 after performing denormalization (111ms to 63ms)

```
select name, inst, email, status
from user
where status_count = 1;
```

[Figure 40] Solution query of problem 6 after performing denormalization

	name	inst	email	status
1	Charles Johnson	한동대학교	charlesJohnson@hgu.db2022.com	박사
2	Reginald Spaulding	한동대학교	reginaldSpaulding@hgu.db2022.com	연구원
3	Jesse Kubik	재단법인통일과ㄴ	jesseKubik@hgu.db2022.com	기타
4	David Dugue	한동대학교	davidDugue@hgu.db2022.com	석사

[Figure 41] Result of problem 6 after performing denormalization

```
[2022-06-20 09:16:54] 4 rows retrieved starting from 1 in 57 ms (execution: 31 ms, fetching: 26 ms)
```

[Figure 42] Execution time of problem 3 after performing denormalization (61ms to 57ms)

```
select distinct tfidfWord, word_count,
       dense_rank() over (order by word_count desc) word_rank
from frequency;
```

[Figure 43] Solution query of problem 10 after performing denormalization

	tfidfWord	word_count	word_rank
1	경제	6911	1
2	가능	6377	2
3	국가	6252	3
4	결과	5884	4
5	관계	5883	5
6	경우	5845	6

[Figure 44] Result of problem 10 after performing denormalization

```
[2022-06-20 09:17:05] 500 rows retrieved starting from 1 in 1 s 568 ms (execution: 1 s 518 ms, fetching: 50 ms)
```

[Figure 45] Execution time of problem 10 after performing denormalization (4s 94ms to 1s 568ms)

4. Step 3: Create Indexes

4.1. Design How to Create Indexes

Finally, we considered and designed how to add indexes efficiently through the queries we created. The index is used to speed up searches in tables using certain conditions, so it would be efficient to add the index to the column frequently used in the **where** clause. And in the b+ tree, the data is automatically sorted, so if we add an index to a column that is often used in the **order by** clause, we can look up the index without referring to the table, so it will be efficient. Based on these two points, we created and added an index to the table. The process of adding indexes and the execution time is represented in [Figure 46] to [Figure 58].

4.2. Create indexes

```
create index post_idx_1 on post (post_date);
```

[Figure 46] Index of problem 2 (*post_date* column is used in the **where** clause)

```
[2022-06-20 09:30:57] 1 row retrieved starting from 1 in 92 ms (execution: 36 ms, fetching: 56 ms)
```

[Figure 47] Execution time of problem 2 after adding the index (118ms to 92ms)

```
create index user_idx_1 on user (bookmark_cnt_2022);
```

[Figure 48] Index of problem 3 (*bookmark_cnt_2022* column is used in the **order by** clause)

```
[2022-06-20 09:31:40] 1 row retrieved starting from 1 in 58 ms (execution: 26 ms, fetching: 32 ms)
```

[Figure 49] Execution time of problem 3 after adding the index (63ms to 58ms)

```
create index similarity_idx_1 on similarity (docID);
create index similarity_idx_2 on similarity (Score);
```

[Figure 50] Index of problem 5 (*docID* column is used in the **where** clause and *Score* column is used in the **order by** clause)

```
[2022-06-20 09:32:07] 3 rows retrieved starting from 1 in 82 ms (execution: 41 ms, fetching: 41 ms)
```

[Figure 51] Execution time of problem 5 after adding the index (777ms to 82ms)

```
create index user_idx_2 on user (status_count);
```

[Figure 52] Index of problem 6 (*status_count* column is used in the **where** clause)

```
[2022-06-20 09:32:48] 4 rows retrieved starting from 1 in 45 ms (execution: 16 ms, fetching: 29 ms)
```

[Figure 53] Execution time of problem 6 after adding the index (57ms to 45ms)

```
create index post_idx_2 on post (post_writer);
create index frequency_idx_1 on frequency (Score);
```

[Figure 54] Index of problem 7 (*post_writer* column is used in the **where** clause and *Score* column is used in the **order by** clause)

```
[2022-06-20 09:33:25] 5 rows retrieved starting from 1 in 453 ms (execution: 432 ms, fetching: 21 ms)
```

[Figure 55] Execution time of problem 7 after adding the index (968ms to 453ms)

```
[2022-06-20 09:34:01] 5 rows retrieved starting from 1 in 125 ms (execution: 99 ms, fetching: 26 ms)
```

[Figure 56] Execution time of problem 8 after adding the index (1s 892ms to 125ms, indexes are already created in the previous problems)

```
create index frequency_idx_3 on frequency (word_count);
```

[Figure 57] Index of problem 10 (*word_count* column is used in the **order by** clause)

```
[2022-06-20 09:34:37] 500 rows retrieved starting from 1 in 1 s 438 ms (execution: 1 s 413 ms, fetching: 25 ms)
```

[Figure 58] Execution time of problem 10 after adding the index (1s 568ms to 1s 438ms)

5. Conclusion

From the above information, we can check that read performance increases when denormalization and index processes are performed. All the execution times are represented in [Figure 59].

	Step 1: Reflect Feedback	Step 2: Perform Denormalization	Step 3: Create Indexes	Conclusion
Problem 1	227ms	100ms		100ms
Problem 2	118ms		92ms	92ms
Problem 3	111ms	63ms	58ms	58ms
Problem 4	83ms			83ms
Problem 5	777ms		82ms	82ms
Problem 6	61ms	57ms	45ms	45ms
Problem 7	968ms		453ms	453ms
Problem 8	1s 892ms		125ms	125ms
Problem 9	92ms			92ms
Problem 10	4s 94ms	1s 568ms	1s 438ms	1s 438ms

[Figure 59] Execution times

There are some queries in [Figure 59] that have a short run time, so there are some queries that do not have a significant difference in run time through denormalization and index process. However, if this query operation is repeated numerous times, it will make a big difference in overall read performance. Finally, the database size and table sizes in Kilobytes are represented in [Figure 60] and [Figure 61].

	DatabaseName	`Size(KB)`
1	2022_itp30010_1_14	616400.0

[Figure 60] Database size

	TABLE_SCHEMA	TABLE_NAME	`data(KB)`	`idx(KB)`
1	2022_itp30010_1_14	board	80.0	16.0
2	2022_itp30010_1_14	bookmark	17008.0	33088.0
3	2022_itp30010_1_14	document	1552.0	0.0
4	2022_itp30010_1_14	file	223792.0	0.0
5	2022_itp30010_1_14	frequency	114320.0	55424.0
6	2022_itp30010_1_14	post	12848.0	4656.0
7	2022_itp30010_1_14	post_document	2576.0	1552.0
8	2022_itp30010_1_14	similarity	72304.0	77088.0
9	2022_itp30010_1_14	user	64.0	32.0

[Figure 61] Table size

Based on [Figure 60] and [Figure 61], we can check that the denormalization and indexing process cause extra storage space and increase the size of the database.

While performing phase 2 tasks, we came to think about the criteria for denormalization and the criteria for using the index. Denormalization is meaningful only after performing normalization elaborately. This is because the *core* table, which is a chunk of data, is not a desirable form of a database. From a developer's point of view, we will have to think about how to compromise read performance and write performance. Randomly denormalized to increase read performance will result in a database like the *core* table, which will increase capacity rapidly, slow insert, update speed, and difficulty in maintaining the database. Even though we were students, it was a meaningful experience to design the database from the perspective of the developer through what we learned in the class. In this process, we were able to understand the abstract E-R diagram, normalization, denormalization, and index concretely.