

한동대학교 24-1학기

Database_한국어1분반 Team 6

Phase 2

ECE30030-01

2024.06.24



	학번	이름
1	22000758	최윤성
2	22100511	이선훈
3	22200527	이성현

For each problems,

solution queries, results, the execution time

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

solution queries

```
SELECT
    MID(post_date, 6, 2) AS post_month,
    COUNT(post_date) / COUNT(DISTINCT LEFT(post_date, 4)) AS post_count
FROM
    document
WHERE
    post_date IS NOT NULL AND LENGTH(post_date) > 4
GROUP BY
    post_month
ORDER BY
    post_count DESC
LIMIT 1;
```

results

	post_month	post_count
1	03	122.1579

the execution time

	1st	2nd	3rd	4th	5th	Total_AVG
before indexing	1.474	1.71	1.99	1.156	1.102	1.4864(s)
after indexing	1.139	1.114	1.77	1.138	1.82	1.3962(s)

2. Find the number of documents per each category (in terms of top_category)

solution queries

```
SELECT top_category, COUNT(hash_key) AS num_doc FROM document WHERE  
top_category IS NOT NULL GROUP BY top_category;
```

results

	☐ top_category ↕	☐ num_doc ↕
1	북한방송	52
2	전체자료	5396
3	탈북자 증언수기	244
4	연구자료	7980
5	정부자료	3640
6	참고자료	500

·
·
·

134	선언문	1
135	인도	1
136	성명서및보도자료	2
137	통일-해외사례	1
138	북한정세	1
139	기타자료실	2

the execution time

1st	2nd	3rd	4th	5th	Total_AVG
1.83	1.67	1.85	1.89	1.77	1.802

3. Find the years in which the words "corona" and "North Korea" are mentioned simultaneously, and sort these years in descending order based on the number of times each word is mentioned per year. (Hint: Use the 'frequency' table.)

solution queries

```
SELECT
    c.year,
    c.mention AS corona,
    n.mention AS north
FROM (
    SELECT
        LEFT(d.timestamp, 4) AS year,
        COUNT(d.hash_key) AS mention
    FROM
        document d
    JOIN
        frequency f ON d.hash_key = f.docId
    WHERE
        f.tfidfWord = '코로나' AND f.score > 0
    GROUP BY
        LEFT(d.timestamp, 4)
) c
JOIN (
    SELECT
        LEFT(d.timestamp, 4) AS year,
        COUNT(d.hash_key) AS mention
    FROM
        document d
    JOIN
        frequency f ON d.hash_key = f.docId
    WHERE
        f.tfidfWord = '북한' AND f.score > 0
    GROUP BY
        LEFT(d.timestamp, 4)
) n ON c.year = n.year
ORDER BY
    c.mention DESC, n.mention DESC;
```

results

	year	corona	north
1	2022	312	7065
2	2023	10	37
3	2024	1	25

the execution time

	1st	2nd	3rd	4th	5th	Total_AVG
before indexing	29.978	29.857	28	28.125	27.581	28.7082(s)
after indexing	1.443	1.412	1.297	1.281	1.454	1.3774(s)

4. Give the title of the most similar document to the document that is saved most frequently by the users in the handong.ac.kr or handong.edu domain.

solution queries

```
SELECT doc_title
FROM document
JOIN (
    SELECT rcmdDocId AS doc_id, score
    FROM similarity
    JOIN (
        SELECT savedDocHashKey
        FROM saved_doc
        JOIN user ON saved_doc.userId = user.userId AND user.email LIKE
        '%@handong%' AND saved_doc.savedDocHashKey IS NOT NULL
        GROUP BY savedDocHashKey
        ORDER BY COUNT(user.userId) DESC
        LIMIT 1
    ) AS handong_freq ON similarity.docId = handong_freq.savedDocHashKey
    ORDER BY score DESC
    LIMIT 1, 1
) AS similarity_freq ON document.hash_key = similarity_freq.doc_id;
```

results

	doc_title
1	“의약품의 음성적 거래로 인한 피해는 국민에게 전가된다”

the execution time

	1st	2nd	3rd	4th	5th	Total_AVG
before indexing	20.228	20.598	20.401	21.21	19.898	20.467(s)
after indexing	2.547	2.517	2.169	2.293	2.121	2.3294(s)

5. Write a query to find the most recently created document based on the 'post_date,' and then display the similarity scores, titles, and post dates of the top five documents that are most similar to it, excluding the document itself.

solution queries

```
#5
SELECT
    ts.score,
    d.doc_title,
    d.post_date
FROM
    document d
JOIN
    (
        SELECT
            s.rcmdDocID,
            s.score
        FROM
            similarity s
        WHERE
            s.docID = (
                SELECT
                    doc.hash_key
                FROM
                    document doc
                ORDER BY
                    doc.post_date DESC
                LIMIT 1
            )
        ORDER BY
            s.score DESC

    ) ts ON d.hash_key = ts.rcmdDocID
ORDER BY
    ts.score DESC
LIMIT 1, 5;
```

results

	□ score ↕	□ doc_title ↕	□ post_date ↕
1	0.561957	‘남북연합’ 구상의 역사와 전망	2008-09-19
2	0.500735	김대중의 통일철학과 햇볕정책	2009-11-17
3	0.486079	북한의 대남정책과 통일정책: 지속과 변	2008-01-18
4	0.446067	통일환경의 변화와 통일방안의 재검토	2014-08-26
5	0.417367	이명박 정부 대북정책 4년 성과	2012-06-05

the execution time

	1st	2nd	3rd	4th	5th	Total_AVG
before indexing	22.109	21.571	21.548	22.238	21.725	21.8382(s)
after indexing	1.53	1.731	1.809	1.735	1.889	1.7388(s)

6. Construct a query to sort the frequency of words in descending order that appear in articles authored by non-profit Korean institutions, where the titles of the articles start with Korean characters ('first_char_title' starting from 'ㄱ' to 'ㅎ').

solution queries

```
SELECT
    f.tfidfWord,
    COUNT(DISTINCT f.docId) AS frequency
FROM
    frequency f
JOIN
    document d1 ON f.docId = d1.hash_key
JOIN
    document d2 ON d1.hash_key = d2.hash_key
WHERE
    (d1.published_institution_url LIKE '%org' OR d1.published_institution_url
    LIKE '%or.kr')
    AND d2.first_char_title BETWEEN 'ㄱ' AND 'ㅎ'
    AND f.score > 0
GROUP BY
    f.tfidfWord
ORDER BY
    frequency DESC;
```

results

	tfidfWord	frequency
1	북한	887
2	정책	617
3	관계	450
4	미국	440
5	문제	431

.
.
.

7718	對美	1
7719	小康	1
7720	常態	1
7721	東京	1
7722	經濟	1
7723	韓美	1

the execution time

	1st	2nd	3rd	4th	5th	Total_AVG
before indexing	19.505	20.005	20.334	19.962	19.261	19.8134(s)
after indexing	9.313	9.635	9.802	9.514	8.884	9.4296(s)

7. Among the words that are used the 8th most frequently in the word frequency analysis, locate the document with the 150th highest score. Next, identify the document with the 4th highest similarity to the above-found document. Please provide the ID, title, and author name for the document.

solution queries

```
SELECT hash_key AS ID ,doc_title AS title ,post_writer AS author
FROM document
WHERE hash_key= (SELECT rcmdDocId
                  FROM similarity
                  WHERE docId =
                    (SELECT docId
                     FROM frequency
                     WHERE tfidfWord = (SELECT tfidfWord
                                          FROM frequency
                                          WHERE score != 0
                                          GROUP BY tfidfWord
                                          ORDER BY count(*) DESC
                                          LIMIT 1 OFFSET 7)
                     ORDER BY score DESC
                     LIMIT 1 OFFSET 149)
                  AND docId != similarity.rcmdDocId
                  ORDER BY score DESC
                  LIMIT 1 OFFSET 3)
```

results

	ID	Title	Author_name
1	3860612892340944	사회보험과 노동조합의 역할 - 한국·독일·일본 비교	윤조덕

the execution time

	1st	2nd	3rd	4th	5th	Total_AVG
before indexing	48.631	47.833	48.371	48.309	48.334	48.2956(s)
after indexing	9.2	9.482	8.793	8.269	8.717	8.8922(s)

8. Among the documents whose titles start with "□", find the ID, title, and author name of the document with the highest tfidf importance for the keyword "고찰".

solution queries

```
SELECT
    f.docId,
    d.post_title,
    d.post_writer
FROM
    document d
JOIN frequency f ON d.hash_key = f.docId AND d.first_char_title='□' AND
f.tfidfWord = '고찰'
ORDER BY f.score DESC
LIMIT 1;
```

results

	□ docId	□ post_title	□ post_writer
1	17111155196474576508	민주주의와 시민사회 가치의 재정	김영래

the execution time

	1st	2nd	3rd	4th	5th	Total_AVG
before indexing	13.468	13.827	13.331	13.749	13.626	13.6002(s)
after indexing	1.53	1.4	1.33	1.3	1.36	1.384(s)

9. In database indexing, the difference in tfidf scores between consecutive words can indicate how well the preceding word describes the document. This difference, termed "predominance," suggests that a larger gap between the tfidf scores of two adjacent words means the first word is more significant in describing the document content.

solution queries

```
CREATE FUNCTION PREDOMINANCE(docId varchar(35))
    RETURNS JSON DETERMINISTIC
RETURN (
    SELECT JSON_ARRAYAGG(
        JSON_OBJECT (
            'docId', docId,
            'tfidfWord', tfidfWord,
            'tfidfScore', score,
            'predominanceScore', score - second_score
        )
    )
    FROM (
        SELECT docId,
            tfidfWord,
            score,
            LEAD(score, 1) OVER (PARTITION BY docId ORDER BY score DESC) AS
second_score
        FROM frequency
        WHERE frequency.docId = docId AND score <> 0
    ) pred_frequency
);

CREATE FUNCTION PREDOMINANCE_TOP(docId varchar(35))
    RETURNS JSON DETERMINISTIC
RETURN (
    SELECT
        JSON_OBJECT(
            'docId', docId,
            'tfidfWord', tfidfWord,
            'tfidfScore', tfidfScore,
            'predominanceScore', predominanceScore
        )
)
```

```

FROM JSON_TABLE(
    PREDOMINANCE(docId),
    '$[*]' COLUMNS (
        docId VARCHAR(35) PATH '$.docId',
        tfidfWord VARCHAR(35) PATH '$.tfidfWord',
        tfidfScore DOUBLE PATH '$.tfidfScore',
        predominanceScore DOUBLE PATH '$.predominanceScore'
    )
) predominance_list
ORDER BY predominanceScore DESC
LIMIT 1
);

DELIMITER //
CREATE PROCEDURE PREDOMINANCE_TOP10()
BEGIN
    DECLARE valid JSON //
    DECLARE pred JSON //

    SET valid = (
        SELECT JSON_ARRAYAGG(
            JSON_OBJECT(
                'docId', hash_key,
                'post_bodyLength', LENGTH(post_body)
            )
        )
        FROM document
        WHERE LENGTH(post_body) > 150
    ) //

    SET pred = (
        SELECT JSON_ARRAYAGG(PREDOMINANCE_TOP(docId))
        FROM JSON_TABLE(
            valid,
            '$[*]' COLUMNS (
                docId VARCHAR(35) PATH '$.docId',
                post_bodyLength DOUBLE PATH '$.post_bodyLength'
            )
        ) valid_doc

```

```

) //

SELECT
    pred_score.docId AS docId,
    post_bodyLength,
    tfidfWord,
    tfidfScore,
    predominanceScore
FROM
    JSON_TABLE(
        pred,
        '$[*]' COLUMNS (
            docId VARCHAR(35) PATH '$.docId',
            tfidfWord VARCHAR(35) PATH '$.tfidfWord',
            tfidfScore DOUBLE PATH '$.tfidfScore',
            predominanceScore DOUBLE PATH '$.predominanceScore'
        )
    ) pred_score
JOIN
    JSON_TABLE(
        valid,
        '$[*]' COLUMNS (
            docId VARCHAR(35) PATH '$.docId',
            post_bodyLength DOUBLE PATH '$.post_bodyLength'
        )
    ) valid_doc
    ON pred_score.docId = valid_doc.docId
ORDER BY predominanceScore DESC
LIMIT 10 //
END //
DELIMITER ;

CALL PREDOMINANCE_TOP10();

```

results

	docId	post_bodyLength	tfidfWord	tfidfScore	predominanceScore
1	55966172767997587	682	바우처	0.9320793151855469	0.8197694048285484
2	2528904465743786239	367	부패	0.9453839063644409	0.8088950365781784
3	13391431084610703904	219	석면	0.9403188228607178	0.807667925953865
4	1907754711089199822	682	바우처	0.9351223707199097	0.8075327128171921
5	18245145984153656468	175	발음	0.9152495265007019	0.77899569272995
6	1141827547106462950	409	무용	0.9173531532287598	0.7749816179275513
7	17383625392319235185	700	열병식	0.8788763284683228	0.7719349414110184
8	3880724392256936776	178	정세	0.9317888021469116	0.7704636454582214
9	13082642792880212232	589	교수	0.8987947702407837	0.7678698152303696
10	6294022130571297194	210	시각	0.9180716872215271	0.7650965452194214

the execution time

1st	2nd	3rd	4th	5th	Total_AVG
49.438	49.189	47.662	51.228	49.727	49.4488(s)

10. For any given Document A, we define the "Representativeness" of a word as the ratio of its tfidf score to the sum of tfidf scores in Document A, which indicates how representative the word is of Document A.

solution queries

```
DELIMITER //
```

```
CREATE FUNCTION REPRESENTATION(docId VARCHAR(35))  
    RETURNS JSON DETERMINISTIC  
BEGIN  
    DECLARE sum_score DOUBLE //
```

```
    SELECT SUM(score) INTO sum_score  
    FROM frequency  
    WHERE frequency.docId = docId //
```

```
    RETURN (  
        SELECT JSON_ARRAYAGG(  
            JSON_OBJECT(  
                'tfidfWord', tfidfWord,  
                'docId', docId,  
                'representation', score / sum_score  
            )  
        )  
        FROM frequency  
        WHERE frequency.docId = docId  
    ) //
```

```
END //
```

```
DELIMITER ;
```



```
DELIMITER //
```

```
CREATE FUNCTION REPRESENTATION_AUTHOR(name VARCHAR(35))  
    RETURNS JSON DETERMINISTIC  
BEGIN  
    DECLARE result JSON DEFAULT JSON_ARRAY() //
```

```
    DECLARE array JSON //
```

```
    DECLARE idx INT DEFAULT 0 //
```

```
    DECLARE len INT //
```

```

DECLARE element JSON //

SET array = (
    SELECT JSON_ARRAYAGG(REPRESENTATION(hash_key))
    FROM document
    WHERE document.post_writer = name
) //

SET len = JSON_LENGTH(array) //

WHILE idx < len DO
    SET element = JSON_EXTRACT(array, CONCAT('$[', idx, ']')) //
    SET result = JSON_MERGE_PRESERVE(result, element) //
    SET idx = idx + 1 //
end while //

RETURN result //
END //
DELIMITER ;

SELECT tfidfWord, docId, representation
FROM JSON_TABLE(
    REPRESENTATION_AUTHOR('김준형'),
    '$[*]' COLUMNS (
        tfidfWord VARCHAR(35) PATH '$.tfidfWord',
        docId VARCHAR(35) PATH '$.docId',
        representation DOUBLE PATH '$.representation'
    )
) REPRESENTATION
ORDER BY representation DESC;

```

results

	tfidfWord	docId	representation
1	실용주의	6842315935094808813	0.301484930680958
2	동맹	17616469537819167592	0.23888018832292615
3	운전대	5525275171169302295	0.22681774313737568
4	마마	5525275171169302295	0.2198681611960841
5	이상주의	6842315935094808813	0.21219261621992772
6	이삭줍기	18435181389713959202	0.2120289342822084
7	무임승차	18435181389713959202	0.198341145149885
8	조급증	18435181389713959202	0.198341145149885
9	이미지	15953384081720205028	0.19384470978011178
10	패스	7626594766852771031	0.19237546468798727
11	보고서	18030222497015784634	0.18717705019181705
12	오바마	5043516459819264334	0.18365975980621416
13	착각	18435181389713959202	0.15857450909444556
14	뜯구름	15953384081720205028	0.15230542137581263
15	발포	12229844749278475923	0.15161281708525626
16	공생	17385708078545315528	0.15027481285283106
17	강경책	17385708078545315528	0.14341626403766933
18	슬픔	17385708078545315528	0.14017908218191147
19	엇박자	15953384081720205028	0.13969187281471104
20	공백	17385708078545315528	0.13896647662386716

...

493	발사	15565125055489991014	0.012257766483135757
494	행사	3899600976102071924	0.012248626922298904
495	양면	5613589032615304747	0.012237355980241043
496	희망	3899600976102071924	0.012219750455515564
497	기존	14093936613680502847	0.012219017001339864
498	도발	15565125055489991014	0.012165965729327486
499	빌미	2777159163859864429	0.012134619511886798
500	동북아	14093936613680502847	0.012124706060509433

the execution time

1st	2nd	3rd	4th	5th	Total_AVG
0.84	0.835	0.832	0.915	0.89	0.8624(s)

Description of how the team used the EXPLAIN, SHOW
PROFILE/PROFILES for the optimization is preferred

de-/re-normalization

먼저 메타 데이터의 종류별로 데이터를 정규화했다.

총 8개의 테이블로 분류하였고 다음과 같다

```
post { post_title , post_body , post_writer , post_date }
institution{ , published_institution , published_institution_url }
file { file_download_url , file_name , file_id_in_fsfiles , file_extracted_content }
doc { first_char_title , top_category , doc_type , doc_title , original_url , timestamp , hash_key , topic }
saved_doc { _id , savedUserName , savedUserEmail , keyword , savedDocHashKey , abstract , docURL }
KUBic { title , content , isMainAnnounce }
QnA { userName , regDate , modeDate , question , answer , category , postId }
user { isAdmin , isApiUser , isActive , name , email , inst , status , userId , registeredDate , modifiedDate }
```

의미 없는 **cartesian product** 되어있던 테이블 **KUBic, QnA** 와 유저 정보를 포함하고 있는 **saved_doc , user** 들을 제외한 다음 4개의 테이블을 하나의 테이블로 비 정규화를 시켰다.

Post : 게시물의 제목, 본문, 작성자, 게시 날짜 정보를 저장.

- post_title
- post_body
- post_writer
- post_date

Institution : 게시물을 발행한 기관에 대한 정보를 저장.

- published_institution
- published_institution_url

File : 게시물과 관련된 파일의 다운로드 링크, 파일 이름, ID, 추출된 내용을 저장.

- file_download_url
- file_name
- file_id_in_fsfiles

- file_extracted_content

Doc : 문서에 대한 메타데이터를 포함하며, 제목, 카테고리, 유형, 원본 URL 등의 정보를 저장.

- first_char_title
- top_category
- doc_type
- doc_title
- original_url
- timestamp
- hash_key
- topic

위 4개의 테이블은 게시물과 문서에 관해서 **hash_key**를 키로 나타낼 수 있고

문서 정보를 조회하기 위해서 너무 많은 **join**이 발생하기 때문에

하나의 **document** 테이블로 **de-normalization** 진행하였다.

Explain Profiling and Indexing steps

#각 문항의 **EXPLAIN** 결과는 **indexing** 전인 상황으로, **document**의 **primary key**인 **hash_key**만 인덱스로 사용된 상황에서 체크되었다.

#1

explain 결과

before index
-> Limit: 1 row(s) (actual time=53.2..53.2 rows=1 loops=1)
-> Sort: post_count DESC, limit input to 1 row(s) per chunk (actual time=53.2..53.2 rows=1 loops=1)
-> Stream results (cost=6141 rows=168) (actual time=41..53.2 rows=12 loops=1)
-> Group aggregate: count(distinct left(document.post_date,4)), count(document.post_date) (cost=6141 rows=168) (actual time=41..53.2 rows=12 loops=1)
-> Sort: post_month (cost=3323 rows=28178) (actual time=39.4..40.8 rows=25882 loops=1)
-> Filter: ((document.post_date is not null) and (length(document.post_date) > 4)) (cost=3323 rows=28178) (actual time=0.054..24.7 rows=25882 loops=1)
-> Table scan on document (cost=3323 rows=28178) (actual time=0.0504..20.7 rows=30317 loops=1)

인덱스 목록 :

1) document 테이블의 post_date

효과 :

1) 검색 및 필터링 속도 향상:

- post_date 의 null 값 조회와 length 필터링이 빨라졌다 .
- count() 함수의 조회가 빨라졌다.

2) 그룹화 성능 개선:

- MID() 함수의 개산 속도 상승으로 인해 GROUP BY의 성능이 향상되었다.

#2

explain 결과

before index
-> Table scan on <temporary> (actual time=39.3..39.3 rows=140 loops=1)
-> Aggregate using temporary table (actual time=39.3..39.3 rows=140 loops=1)
-> Table scan on document (cost=3323 rows=28178) (actual time=0.0498..16.4 rows=30317 loops=1)

인덱스 목록 :

- 1) document 테이블의 hash_key (Primary Key, 자동 인덱싱됨)
- 2) document 테이블의 top_category

효과 :

- 1) 검색 및 필터링 속도 향상:
 - hash_key는 Primary Key로 이미 인덱스화되어 있어, 이 필드를 기준으로 하는 검색과 필터링이 빨라졌다.

#3

explain 결과

before index
-> Limit: 1 row(s) (actual time=53.2..53.2 rows=1 loops=1)
-> Sort: post_count DESC, limit input to 1 row(s) per chunk (actual time=53.2..53.2 rows=1 loops=1)
-> Stream results (cost=6141 rows=168) (actual time=41..53.2 rows=12 loops=1)
-> Group aggregate: count(distinct left(document.post_date,4)), count(document.post_date) (cost=6141 rows=168) (actual time=41..53.2 rows=12 loops=1)
-> Sort: post_month (cost=3323 rows=28178) (actual time=39.4..40.8 rows=25882 loops=1)
-> Filter: ((document.post_date is not null) and (length(document.post_date) > 4)) (cost=3323 rows=28178) (actual time=0.054..24.7 rows=25882 loops=1)
-> Table scan on document (cost=3323 rows=28178) (actual time=0.0504..20.7 rows=30317 loops=1)

인덱스 목록 :

- 1) document 테이블의 hash_key (Primary Key, 자동 인덱싱됨)
- 2) frequency 테이블의 docId
- 3) frequency 테이블의 tfidfWord
- 4) frequency 테이블의 score

효과 :

- 1) 검색 속도 향상 :
 - tfidfWord와 score 필드에 인덱스가 추가되면서, 해당 필드를 기준으로 하는 검색 속도가 빨라졌다.
- 2) 조인 성능 개선 :
 - hash_key와 docId의 조인은 이미 인덱스화 되어서 속도가 빨라졌다.

#4

explain 결과

before index
-> Nested loop inner join (cost=3.43 rows=2) (actual time=22577..22577 rows=1 loops=1)
-> Filter: (similarity_freq.doc_id is not null) (cost=245256..2.73 rows=2) (actual time=22577..22577 rows=1 loops=1)
-> Table scan on similarity_freq (cost=490511..490511 rows=1) (actual time=22577..22577 rows=1 loops=1)
-> Materialize (cost=490509..490509 rows=1) (actual time=22577..22577 rows=1 loops=1)
-> Limit/Offset: 1/1 row(s) (cost=490509 rows=1) (actual time=22577..22577 rows=1 loops=1)
-> Sort: similarity.score DESC, limit input to 2 row(s) per chunk (cost=490509 rows=43.9e+6) (actual time=22577..22577 rows=2 loops=1)
-> Filter: (similarity.docId = '809917129479975178') (cost=490509 rows=43.9e+6) (actual time=567..22577 rows=523 loops=1)
-> Table scan on similarity (cost=490509 rows=43.9e+6) (actual time=0.0133..19817 rows=44.1e+6 loops=1)
-> Filter: (document.hash_key = similarity_freq.doc_id) (cost=0.3 rows=1) (actual time=0.0162..0.0163 rows=1 loops=1)
-> Single-row index lookup on document using PRIMARY (hash_key=similarity_freq.doc_id) (cost=0.3 rows=1) (actual time=0.0151..0.0152 rows=1 loops=1)

인덱스 목록 :

- 1) document 테이블의 hash_key (Primary Key, 자동 인덱싱됨)
- 2) similarity 테이블의 score
- 3) saved_doc 테이블의 userId
- 4) saved_doc 테이블의 savedDocHashKey

효과 :

- 1) 검색 및 필터링 속도 향상:
 - 조인 조건에 필터링 조건을 포함시켜 조인 시점에서 불필요한 레코드를 제거함으로써, 전체적으로 처리해야 할 데이터 양을 줄였다.
 - saved_doc 테이블의 userId에 인덱스를 추가함으로써, @handong 을 기준으로 하는 검색 속도가 빨라졌다.
 - saved_doc 테이블의 savedDocHashKey에 인덱스를 추가함으로써, Null값이 아닌 것을 찾기 위한 검색 속도가 빨라졌다.
 -
- 2) 조인 성능 개선 :

- **hash_key**를 기준으로 하는 **document** 테이블과 **similarity_freq** 서브쿼리 간의 조인은 인덱스를 활용하여 빨라졌다.
- **saved_doc**와 **user** 테이블 간의 조인도 **userId** 인덱스를 활용하여 빨라졌다.

3) **WHERE**절을 사용하지 않은 효과 :

- **WHERE** 절 대신 **JOIN** 절에서 조건을 포함시켜 조인 시점에서 필터링을 수행함으로써, 조인 전에 조건에 맞지 않는 레코드를 제외하였다. 조인 시 처리해야 할 레코드 수를 줄여, 쿼리 성능을 향상시켰다.

4) 실행 시간 비교

- **WHERE**절을 사용하지 않음으로써 실행시간이 평균 **0.6초**가량 줄어들었다. (3.8초 -> 3.2초)

#5

explain 결과

before index
-> Limit/Offset: 5/1 row(s) (cost=9.94e+6 rows=5) (actual time=23298..23298 rows=5 loops=1)
-> Nested loop inner join (cost=9.94e+6 rows=43.9e+6) (actual time=23298..23298 rows=6 loops=1)
-> Sort: score DESC (cost=4.45e+6 rows=43.9e+6) (actual time=23298..23298 rows=10 loops=1)
-> Filter: ((s.docId = (select #3)) and (s.rcmdDocId is not null)) (cost=4.45e+6 rows=43.9e+6) (actual time=22463..23293 rows=6644 loops=1)
-> Table scan on s (cost=4.45e+6 rows=43.9e+6) (actual time=0.0323..19187 rows=44.1e+6 loops=1)
-> Select #3 (subquery in condition; run only once)
-> Limit: 1 row(s) (cost=3323 rows=1) (actual time=25.6..25.6 rows=1 loops=1)
-> Sort: doc.post_date DESC, limit input to 1 row(s) per chunk (cost=3323 rows=28178) (actual time=25.6..25.6 rows=1 loops=1)
-> Table scan on doc (cost=3323 rows=28178) (actual time=0.0346..19.1 rows=30317 loops=1)
-> Filter: (d.hash_key = s.rcmdDocId) (cost=0.25 rows=1) (actual time=0.00716..0.0072 rows=0.6 loops=10)
-> Single-row index lookup on d using PRIMARY (hash_key=s.rcmdDocId) (cost=0.25 rows=1) (actual time=0.0068..0.00682 rows=0.6 loops=10)

인덱스 목록 :

- 1) document 테이블의 hash_key (Primary Key, 자동 인덱싱됨)
- 2) similarity 테이블의 score
- 3) document 테이블의 post_date

효과 :

- 1) 검색 및 필터링 속도 향상:
 - document 테이블의 post_date에 인덱스를 추가함으로써, 최신 문서를 검색하는 속도가 빨라졌다.
 - ORDER BY doc.post_date DESC LIMIT 1 절의 성능도 향상되었다.
 - similarity 테이블의 score 필드에 인덱스를 추가하여, 점수 순으로 정렬하는 작업이 빨라졌다.
- 2) 조인 성능 개선 :
 - hash_key를 기준으로 하는 document 테이블과 similarity 서브쿼리 간의 조인은 인덱스를 활용하여 빨라졌다.

#6

explain 결과

before index
-> Sort: frequency DESC (actual time=20957..20958 rows=7723 loops=1)
-> Stream results (cost=8.92e+6 rows=5231) (actual time=14545..20953 rows=7723 loops=1)
-> Group aggregate: count(distinct f.docId) (cost=8.92e+6 rows=5231) (actual time=14545..20950 rows=7723 loops=1)
-> Nested loop inner join (cost=8.86e+6 rows=638001) (actual time=14542..20911 rows=74680 loops=1)
-> Nested loop inner join (cost=7.8e+6 rows=5.74e+6) (actual time=14537..20623 rows=117054 loops=1)
-> Sort: f.tfidfWord (cost=2.79e+6 rows=27.4e+6) (actual time=14536..14661 rows=1.94e+6 loops=1)
-> Filter: ((f.score > 0) and (f.docId is not null)) (cost=2.79e+6 rows=27.4e+6) (actual time=0.0349..13004 rows=1.94e+6 loops=1)
-> Table scan on f (cost=2.79e+6 rows=27.4e+6) (actual time=0.0318..11646 rows=26.1e+6 loops=1)
-> Filter: (((d1.published_institution_url like '%org') or (d1.published_institution_url like '%or.kr')) and (f.docId = d1.hash_key)) (cost=0.25 rows=0.21) (actual time=0.003..0.003 rows=0.0604 loops=1.94e+6)
-> Single-row index lookup on d1 using PRIMARY (hash_key=f.docId) (cost=0.25 rows=1) (actual time=0.00243..0.00245 rows=0.854 loops=1.94e+6)
-> Filter: ((d2.hash_key = d1.hash_key) and (d2.first_char_title between 'ㄱ' and 'ㅎ')) (cost=0.25 rows=0.111) (actual time=0.0023..0.00234 rows=0.638 loops=117054)
-> Single-row index lookup on d2 using PRIMARY (hash_key=f.docId) (cost=0.25 rows=1) (actual time=0.00186..0.00188 rows=1 loops=117054)

인덱스 목록 :

- 1) document 테이블의 hash_key (Primary Key, 자동 인덱싱됨)
- 2) frequency 테이블의 tfidfWord
- 3) frequency 테이블의 score
- 4) document 테이블의 first_char_title
- 5) frequency 테이블의 docId
- 6) document 테이블의 published_institution_url

효과 :

- 1) 검색 및 필터링 속도 향상:
 - 조인 조건에 필터링 조건을 포함시켜 조인 시점에서 불필요한 레코드를 사전에 제거함으로써, 전체적으로 처리해야 할 데이터 양을 줄였다. 이를 통하여 조인 과정에서 데이터 양을 줄여 쿼리 성능을 향상시켰다.
 - published_institution_url 필드에 인덱스를 추가함으로써, LIKE 조건(%org, %or.kr)을 사용하는 검색 속도가 빨라졌다.

- **first_char_title** 필드에 인덱스를 추가하여, 특정 범위('ㄱ' AND 'ㅎ')로 필터링하는 작업이 빨라졌다.

2) 조인 성능 개선 :

- **hash_key**를 기준으로 하는 **document** 테이블과 **frequency** 테이블 간의 조인은 인덱스를 활용하여 빨라졌다.
- **docId**와 **hash_key** 간의 조인도 인덱스를 통해 빨라졌다.

3) WHERE절을 사용하지 않은 효과 :

- **WHERE** 절 대신 **JOIN** 절에서 조건을 포함시켜 조인 시점에서 필터링을 수행하면, 조인 전에 조건에 맞지 않는 레코드를 제외할 수 있습니다. 이는 조인 시 처리해야 할 레코드 수를 줄여, 쿼리 성능을 향상시킵니다..

4) 실행 시간 비교 :

- **first_char_title**, **docId**, **published_institution_url**이 인덱스로 사용되지 않았을 때에 비하여 실행시간이 평균적으로 8초 가량 줄어들었다.(16초 -> 7초)
- **WHERE**절을 사용하지 않음으로써 실행시간이 평균 1초가량 줄어들었다. (7초 -> 6초)

#7

explain 결과

before index
-> Rows fetched before execution (cost=0..0 rows=1) (actual time=65e-6..120e-6 rows=1 loops=1)

인덱스 목록 :

- 1) document 테이블의 hash_key (Primary Key, 자동 인덱싱됨)
- 2) frequency 테이블의 tfidfWord
- 3) frequency 테이블의 score
- 4) similarity 테이블의 score
- 5) frequency 테이블의 docId

효과 :

- 1) 검색 및 필터링 속도 향상:

tfidfWord 필드에 인덱스를 추가함으로써, 특정 단어를 검색하는 속도가 빨라졌다. score 필드에 인덱스를 추가하여, 특정 조건(score != 0)을 기준으로 필터링하는 작업이 빨라졌다.

- 2) 조인 성능 개선:

- hash_key를 기준으로 하는 document 테이블과 서브쿼리 간의 조인이 빨라졌다.
- docId를 기준으로 하는 frequency 테이블과 similarity 테이블 간의 조인이 빨라졌다.

- 3) 정렬 성능 향상:

score 필드에 인덱스를 추가하여, score 값에 따라 내림차순으로 빠르게 정렬되었다.

#8

explain 결과

before index
-> Limit: 1 row(s) (cost=6.21e+6 rows=1) (actual time=14739..14739 rows=1 loops=1)
-> Nested loop inner join (cost=6.21e+6 rows=2.74e+6) (actual time=14739..14739 rows=1 loops=1)
-> Sort: f.score DESC (cost=2.79e+6 rows=27.4e+6) (actual time=14739..14739 rows=12 loops=1)
-> Filter: ((f.tfidfWord = '고찰') and (f.docId is not null)) (cost=2.79e+6 rows=27.4e+6) (actual time=0.0753..14738 rows=166 loops=1)
-> Table scan on f (cost=2.79e+6 rows=27.4e+6) (actual time=0.0354..12136 rows=26.1e+6 loops=1)
-> Filter: ((d.first_char_title = 'ㄷ') and (d.hash_key = f.docId)) (cost=0.25 rows=0.1) (actual time=0.00912..0.00912 rows=0.0833 loops=12)
-> Single-row index lookup on d using PRIMARY (hash_key=f.docId) (cost=0.25 rows=1) (actual time=0.00855..0.00856 rows=0.833 loops=12)

인덱스 목록 :

- 1) document 테이블의 hash_key (Primary Key, 자동 인덱싱됨)
- 2) frequency 테이블의 tfidf_word
- 3) document 테이블의 first_char_title
- 4) frequency 테이블의 score
- 5) frequency 테이블의 docId
- 6) document 테이블의 post_title

효과 :

- 1) 검색 및 필터링 속도 향상:
 - first_char_title 필드에 인덱스를 추가함으로써, 특정 값('ㄷ')으로 필터링하는 속도가 빨라졌다.
 - tfidf_word 필드에 인덱스를 추가하여, 특정 단어('고찰')를 기준으로 검색하는 작업이 빨라졌다.
 - post_title 필드에 인덱스를 추가함으로써, 이 필드에 대한 검색 및 정렬 속도가 빨라졌다.
- 2) 조인 성능 개선:
 - hash_key와 docId 필드에 인덱스를 추가하여 document 테이블과 frequency 테이블 간의 조인이 빨라졌다.
- 3) 정렬 성능 향상:
 - score 필드에 인덱스를 추가하여, 결과를 score 값에 따라 내림차순으로 정렬하는 작업이 빨라졌다.

Summary of the database size and table sizes

Size of the Database :

DatabaseName	Size(KB)
db_proj_06	3291312.0

Size of the tables :

TABLE_SCHEMA	TABLE_SIZE	data(KB)	idx(KB)
db_proj_06	KUBiC	16.0	0.0
db_proj_06	QnA	16.0	0.0
db_proj_06	document	32336.0	10832.0
db_proj_06	frequency	3259392.0	0.0
db_proj_06	saved_doc	3600.0	880.0
db_proj_06	similarity	3269632.0	0.0
db_proj_06	user	64.0	0.0

###이상현상

인덱스 용량이 계속하여 감지 되지 않는 버그가 있습니다
이 부분에 대해서 교수님과 TA님께 연락드렸으며 원인은 innoDB와
관련된 오류문제라고 들었습니다. 그리고 알려주신 피드백 대로
새로고침 혹은 **index name**의 수정을 여러번 시도하였지만 여전히
문제는 해결되지 않았습니다. 평가요소에는 반영이 안된다고 TA님께
연락받았으나, 만약 평가요소에 반영이 된다면 이 점 참고해주시면
감사하겠습니다.

	TABLE_SCHEMA	TABLE_NAME	`data(KB)`	`idx(KB)`
1	db_proj_06	KUBiC	16.0	0.0
2	db_proj_06	QnA	16.0	0.0
3	db_proj_06	document	32336.0	10832.0
4	db_proj_06	frequency	3259392.0	0.0
5	db_proj_06	saved_doc	3600.0	880.0
6	db_proj_06	similarity	3269632.0	0.0
7	db_proj_06	user	64.0	0.0