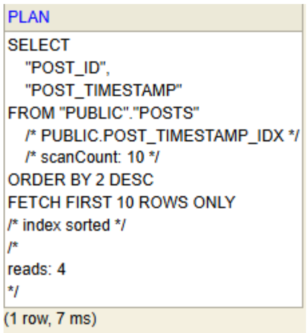
HW4 Oliver Li

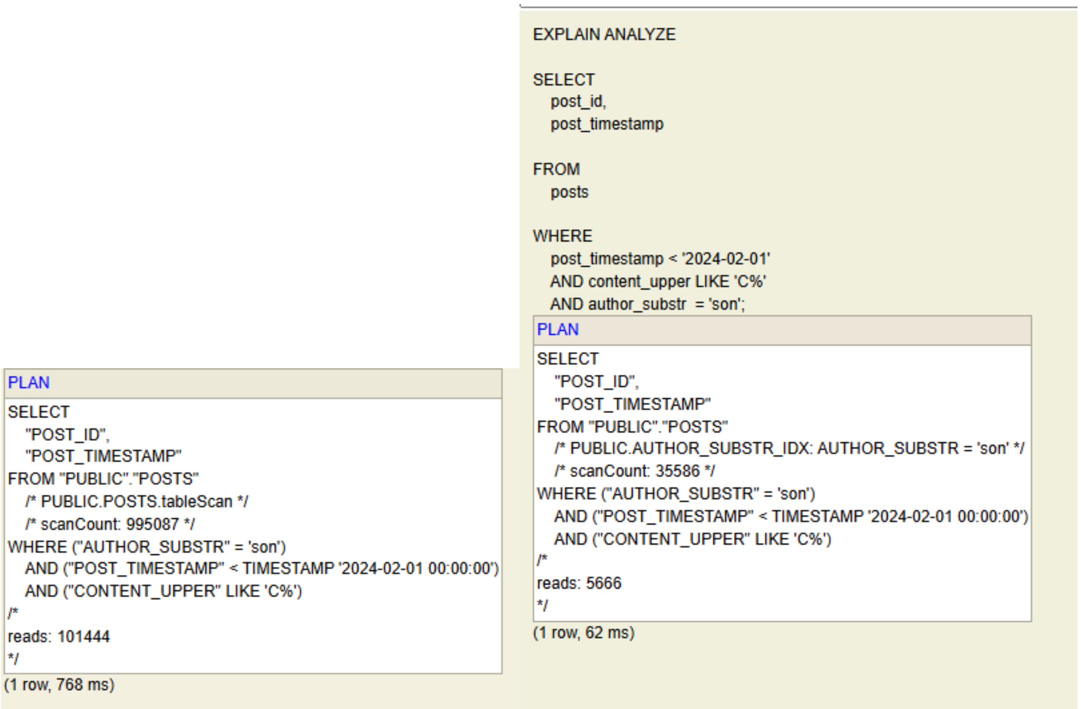
```
## Problem 1 - Recent Posts

<change you made>

CREATE INDEX post_timestamp_idx ON posts(post_timestamp);
SELECT
    post_id,
    post_timestamp

FROM
    posts

ORDER BY
    post_timestamp DESC

LIMIT 10

<screenshot of EXPLAIN ANALYZE>
```



```
PLAN
SELECT
    "POST_ID",
    "POST_TIMESTAMP"
FROM "PUBLIC"."POSTS"
    /* PUBLIC.POST_TIMESTAMP_IDX */
    /* scanCount: 10 */
ORDER BY 2 DESC
FETCH FIRST 10 ROWS ONLY
/* index sorted */
/*
reads: 4
*/
(1 row, 7 ms)
```

Adding an index to improve performance

```
## Problem 2 - Somewhat Strange Query

<change you made>
ALTER TABLE posts
        ADD COLUMN author_substr VARCHAR(3) AS SUBSTR(author, 3, 3);

ALTER TABLE posts
        ADD COLUMN content_upper VARCHAR(255) AS UPPER(content);

CREATE INDEX content_idx ON posts (content);
CREATE INDEX post_timestamp_idx ON posts (post_timestamp);
CREATE INDEX author_substr_idx ON posts (author_substr);
CREATE INDEX content_upper_idx ON posts (content_upper);
SELECT
        post_id,
        post_timestamp
FROM
        posts
WHERE
        post_timestamp < '2024-02-01'
        AND content_upper LIKE 'C%'
        AND author_substr = 'son';

<screenshot of EXPLAIN ANALYZE>
```



```
EXPLAIN ANALYZE

SELECT
   post_id,
   post_timestamp

FROM
   posts

WHERE
   post_timestamp < '2024-02-01'
   AND content_upper LIKE 'C%'
   AND author_substr  = 'son';
```

```
PLAN
SELECT
   "POST_ID",
   "POST_TIMESTAMP"
FROM "PUBLIC"."POSTS"
   /* PUBLIC.POSTS.tableScan */
   /* scanCount: 995087 */
WHERE ("AUTHOR_SUBSTR" = 'son')
   AND ("POST_TIMESTAMP" < TIMESTAMP '2024-02-01 00:00:00')
   AND ("CONTENT_UPPER" LIKE 'C%')
/*
reads: 101444
*/
(1 row, 768 ms)
```

```
PLAN
SELECT
   "POST_ID",
   "POST_TIMESTAMP"
FROM "PUBLIC"."POSTS"
   /* PUBLIC.AUTHOR_SUBSTR_IDX: AUTHOR_SUBSTR = 'son' */
   /* scanCount: 35586 */
WHERE ("AUTHOR_SUBSTR" = 'son')
   AND ("POST_TIMESTAMP" < TIMESTAMP '2024-02-01 00:00:00')
   AND ("CONTENT_UPPER" LIKE 'C%')
/*
reads: 5666
*/
(1 row, 62 ms)
```

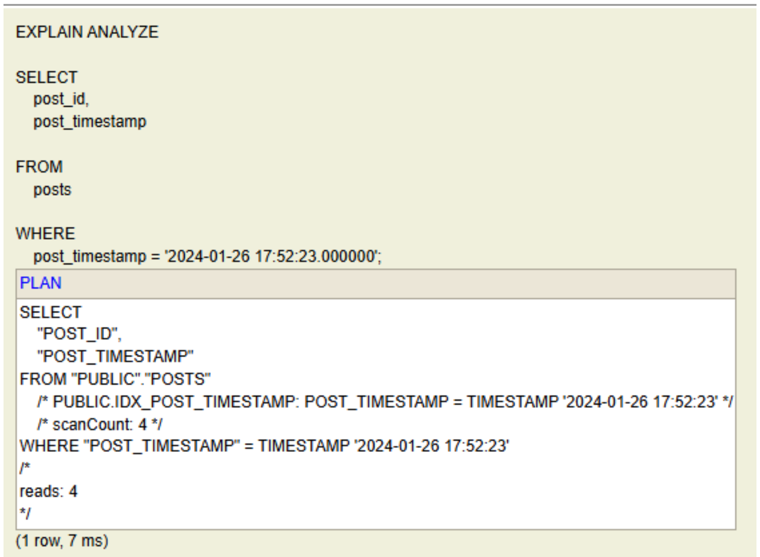
Creating an index on all the provided queries allows faster performance speed. Also creating subrows to provide the indices.

Problem 3 - Really Fast Single Row Responses

Problem 3.1

<What index does H2DB end up using? Explain the pros and cons of each index that you created.> The H2DB ends up using the B-Tree indexing. For both of the indices: B-Trees work fast on range queries and sorting, perform well (O(logN)) in lookup tasks/exact match queries, and more general to use. However B-Trees take up more space and memory in practice, also performs slower in insets and updates due to re-balancing in data. On the other hand Hash Indices use less space and performs fastest on lookup time. However it is neither useful for range queries or sorting queries.

```
SET TRACE_LEVEL_SYSTEM_OUT 3;
SET CACHE_SIZE 0;
SET MAX_MEMORY_ROWS 0;
SET MAX_MEMORY_UNDO 0;
EXPLAIN ANALYZE

SELECT
   post_id,
   post_timestamp

FROM
   posts

WHERE
   post_timestamp = '2024-01-26 17:52:23.000000'

CREATE INDEX post_timestamp_idx ON posts (post_timestamp);

DROP INDEX post_timestamp_idx;
```



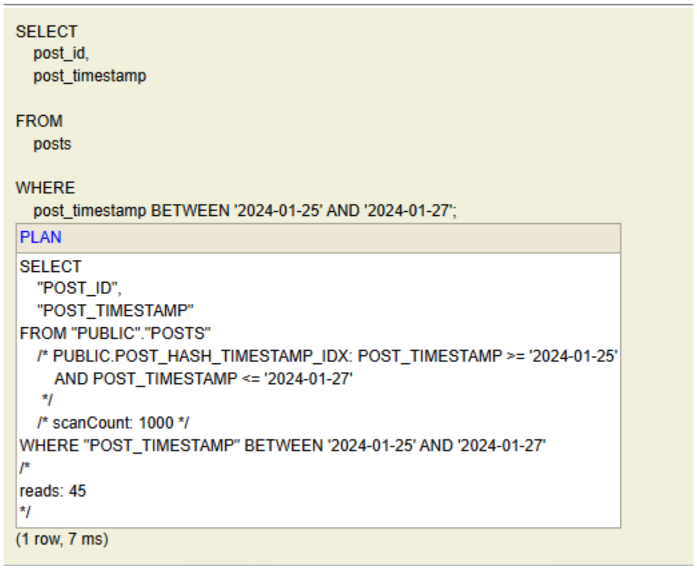
```
EXPLAIN ANALYZE

SELECT
   post_id,
   post_timestamp

FROM
   posts

WHERE
   post_timestamp = '2024-01-26 17:52:23.000000';
```

```
PLAN
SELECT
   "POST_ID",
   "POST_TIMESTAMP"
FROM "PUBLIC"."POSTS"
   /* PUBLIC.IDX_POST_TIMESTAMP: POST_TIMESTAMP = TIMESTAMP '2024-01-26 17:52:23' */
   /* scanCount: 4 */
WHERE "POST_TIMESTAMP" = TIMESTAMP '2024-01-26 17:52:23'
/*
reads: 4
*/
(1 row, 7 ms)
```

Problem 3.2

<Which of the indexes that you created for 3.1 would you expect to be used now. Please explain.> It is expected to use a B-Tree Index. A BETWEEN AND operation is not supported by a hash index, as hash indices do not have order.

```
SET TRACE_LEVEL_SYSTEM_OUT 3;
SET CACHE_SIZE 0;
SET MAX_MEMORY_ROWS 0;
SET MAX_MEMORY_UNDO 0;
EXPLAIN ANALYZE

SELECT
   post_id,
   post_timestamp

FROM
   posts

WHERE
   post_timestamp BETWEEN '2024-01-25' AND '2024-01-27';

CREATE HASH INDEX post_hash_timestamp_idx ON posts (post_timestamp);
CREATE INDEX post_btree_timestamp_idx ON posts (post_timestamp);

DROP INDEX post_btree_timestamp_idx ;
DROP INDEX post_hash_timestamp_idx ;
```



```
SELECT
   post_id,
   post_timestamp

FROM
   posts

WHERE
   post_timestamp BETWEEN '2024-01-25' AND '2024-01-27';
```

PLAN

```
SELECT
   "POST_ID",
   "POST_TIMESTAMP"
FROM "PUBLIC"."POSTS"
   /* PUBLIC.POST_HASH_TIMESTAMP_IDX: POST_TIMESTAMP >= '2024-01-25'
      AND POST_TIMESTAMP <= '2024-01-27'
    */
   /* scanCount: 1000 */
WHERE "POST_TIMESTAMP" BETWEEN '2024-01-25' AND '2024-01-27'
/*
reads: 45
*/
```

(1 row, 7 ms)

Problem 3.3

<Can you modify one of the indexes from 3.2 to make this query even faster? Explain why your change to the index made the query even faster.> CREATE INDEX post_timestamp_btree_idx ON posts (post_timestamp, post_id, content); This is a composite index that allow he coverage of all the columns in the query, where the database can fetch data directly from the index without having to access the full table.

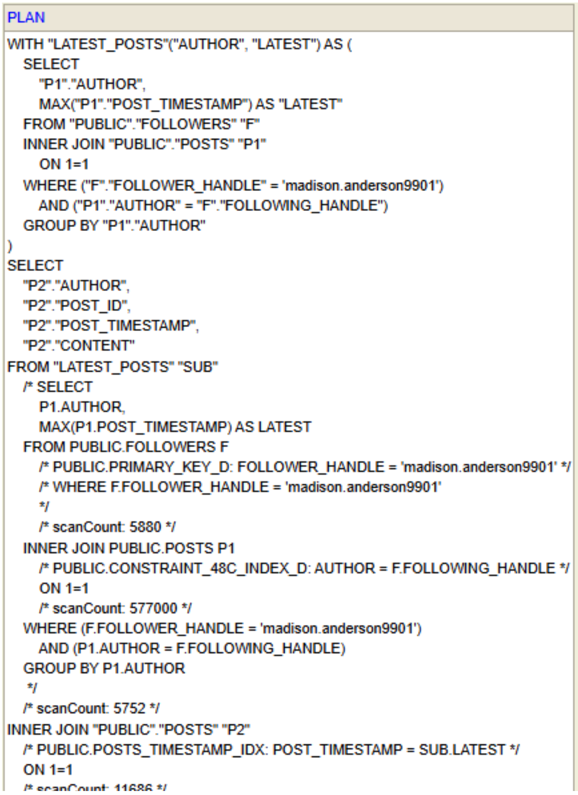
```
## Problem 4 - Table Join Order
### Problem 4.1

<Your modified query here>

SELECT
      COUNT(1)

FROM
        users
        JOIN followers ON users.handle = followers.follower_handle
        JOIN posts ON followers.following_handle = posts.author

WHERE
        users.last_name = 'Anderson'
        AND users.first_name = 'Abigail';
```

Problem 4.2

We have 3 tables to join hence we have 3!-2=4 possible joins possible, and known that the size of each table is: users = 10000; followers = 995040; posts = 995086; Note that the performance can only be optimized when each operation eliminates most elements first, hence given possible joins:

```
FROM
        posts
        JOIN followers ON posts.author = followers.following_handle
        JOIN users ON followers.follower_handle = users.handle
### Gives 205 in 103754 ms
### Likely to take the longest time as the operations take the longest table first, then the second longest and the shortest.
### Elimination of rows is minimized.

FROM
        users
        JOIN followers ON users.handle = followers.follower_handle
        JOIN posts ON followers.following_handle = posts.author
### Gives 205 in 10 ms
### Likely to take the shortest time as the operations take the shortest table first, then the second shortest and the longest.
### Elimination of rows is maximized as the table is short in the first place.

FROM
        followers
        JOIN posts ON followers.following_handle = posts.author
        JOIN users ON followers.follower_handle = users.handle
### Gives 205 in 61401 ms
### Better than the first JOIN order, as the second longest table is given at first, then the longest and the shortest
### Elimination of rows is not maximized.

FROM
        followers
        JOIN users ON followers.follower_handle = users.handle
        JOIN posts ON followers.following_handle = posts.author
### Gives 205 in 672 ms
### Better than the previous JOIN order, as the second longest table is given at first, then the shortest and the longest
### Elimination of rows is not maximized but slightly better than the previous operation order, as 995040->10000 is better than 99
```

```
## Problem 5 - Putting it All Together - Fast Most Recent Posts

<your query here>
DROP INDEX IF EXISTS posts_author_idx;
DROP INDEX IF EXISTS followers_follower_idx;
DROP INDEX IF EXISTS followers_following_idx;
DROP INDEX IF EXISTS posts_timestamp_idx;

CREATE HASH INDEX followers_following_idx ON followers (following_handle);
CREATE HASH INDEX followers_follower_idx ON followers (follower_handle);
CREATE HASH INDEX posts_author_idx ON posts (author);

CREATE INDEX posts_timestamp_idx ON posts (post_timestamp DESC);

EXPLAIN ANALYZE
WITH latest_posts AS (
      SELECT
              p1.author,
              MAX(p1.post_timestamp) AS latest
      FROM followers f
      JOIN posts p1
              ON p1.author = f.following_handle
      WHERE f.follower_handle = 'madison.anderson9901'
      GROUP BY p1.author
)
SELECT
        p2.author,
        p2.post_id,
        p2.post_timestamp,
        p2.content
FROM latest_posts sub
JOIN posts p2
        ON p2.author = sub.author
        AND p2.post_timestamp = sub.latest
ORDER BY p2.post_timestamp DESC;

```



```
PLAN
WITH "LATEST_POSTS"("AUTHOR", "LATEST") AS (
   SELECT
      "P1"."AUTHOR",
      MAX("P1"."POST_TIMESTAMP") AS "LATEST"
   FROM "PUBLIC"."FOLLOWERS" "F"
   INNER JOIN "PUBLIC"."POSTS" "P1"
      ON 1=1
   WHERE ("F"."FOLLOWER_HANDLE" = 'madison.anderson9901')
      AND ("P1"."AUTHOR" = "F"."FOLLOWING_HANDLE")
   GROUP BY "P1"."AUTHOR"
)
SELECT
   "P2"."AUTHOR",
   "P2"."POST_ID",
   "P2"."POST_TIMESTAMP",
   "P2"."CONTENT"
FROM "LATEST_POSTS" "SUB"
   /* SELECT
      P1.AUTHOR,
      MAX(P1.POST_TIMESTAMP) AS LATEST
   FROM PUBLIC.FOLLOWERS F
      /* PUBLIC.PRIMARY_KEY_D: FOLLOWER_HANDLE = 'madison.anderson9901' */
      /* WHERE F.FOLLOWER_HANDLE = 'madison.anderson9901'
      */
      /* scanCount: 5880 */
   INNER JOIN PUBLIC.POSTS P1
      /* PUBLIC.CONSTRAINT_48C_INDEX_D: AUTHOR = F.FOLLOWING_HANDLE */
      ON 1=1
      /* scanCount: 577000 */
   WHERE (F.FOLLOWER_HANDLE = 'madison.anderson9901')
      AND (P1.AUTHOR = F.FOLLOWING_HANDLE)
   GROUP BY P1.AUTHOR
    */
   /* scanCount: 5752 */
INNER JOIN "PUBLIC"."POSTS" "P2"
   /* PUBLIC.POSTS_TIMESTAMP_IDX: POST_TIMESTAMP = SUB.LATEST */
   ON 1=1
   /* scanCount: 11686 */
```