Maximizing Cassandra's Potential: Tips on Schema, Queries, Parallel Access, and Reactive Programming

Hartmut Armbruster • Thriving.dev

Hartmut Armbruster

Software Architect, Developer,
Independent Consultant

# Architecture • Data • Cloud-native •
Distributed Systems • High-load/Scalability •
Stream Processing • Backend • Web Front-End •
Reactive Programming • Kotlin/Java • TS/JS •
Vue.js • Nuxt.js • Kubernetes • GitOps

Agenda

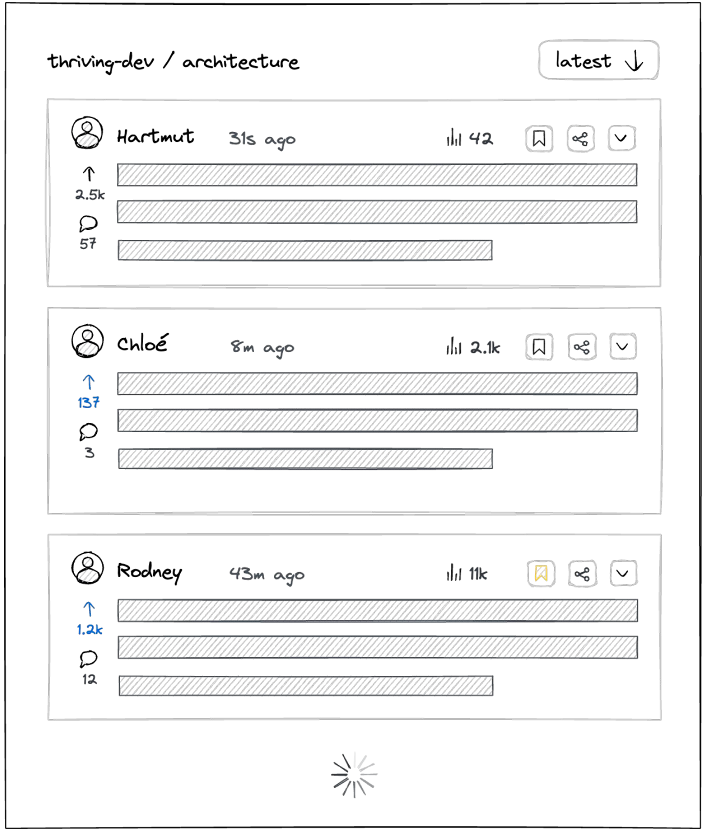
- Use Case: 'Social Platform Feed'
- Access Patterns
  - Queries & Schema
  - Process Flow
- Iterative Refinement Process
- Non-blocking, Reactive Programming
- Conclusions

Functional Requirements

RESTful API

Social Platform Feed

- 'feeds' have 'posts'
- posts can be
  - listed in a given order
  - liked, bookmarked, commented
- "endless scroll"

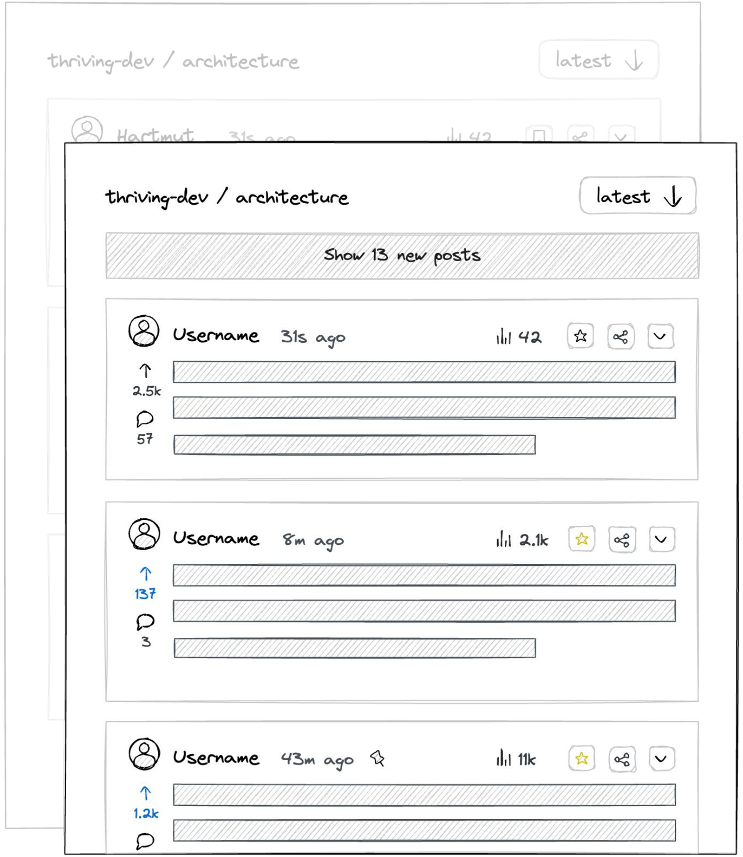


Functional Requirements

RESTful API

Social Platform Feed

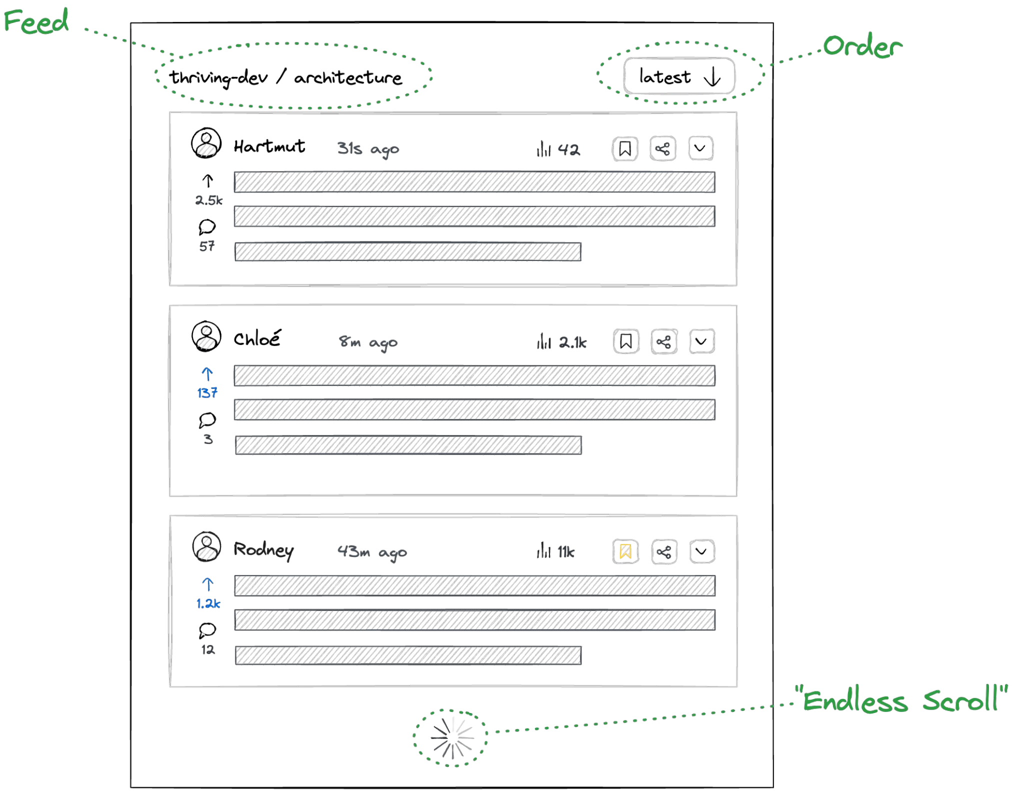
- 'feeds' have 'posts'
- posts can be
  - listed in a given order
  - liked, bookmarked, commented
- "endless scroll"
- "pinned pagination" (client-side state)
- "new posts" indicator



Data

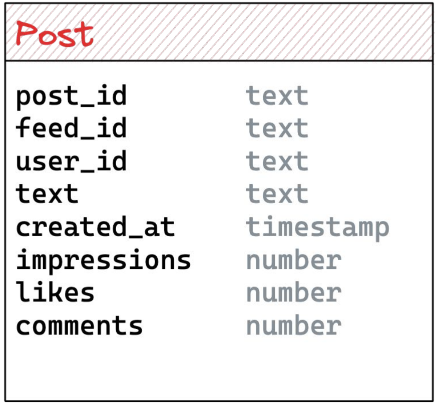
Entities & Relationships

Feed	
feed_id	text
name	text
created_at	timestamp

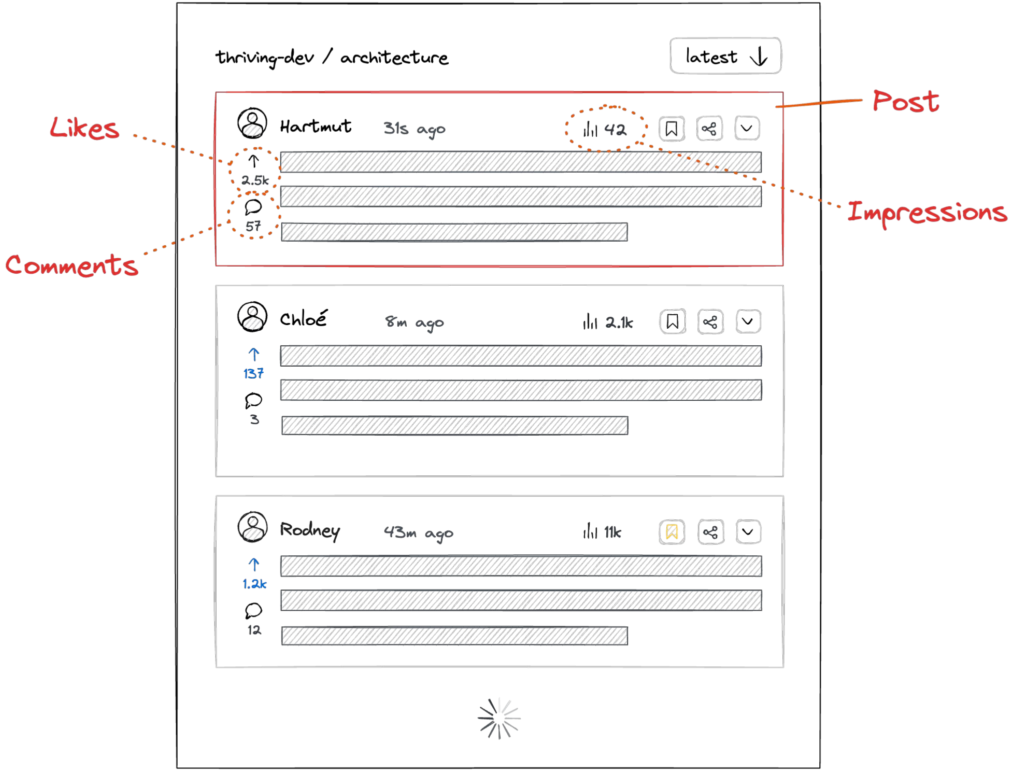


Data

Entities & Relationships



Post	
post_id	text
feed_id	text
user_id	text
text	text
created_at	timestamp
impressions	number
likes	number
comments	number

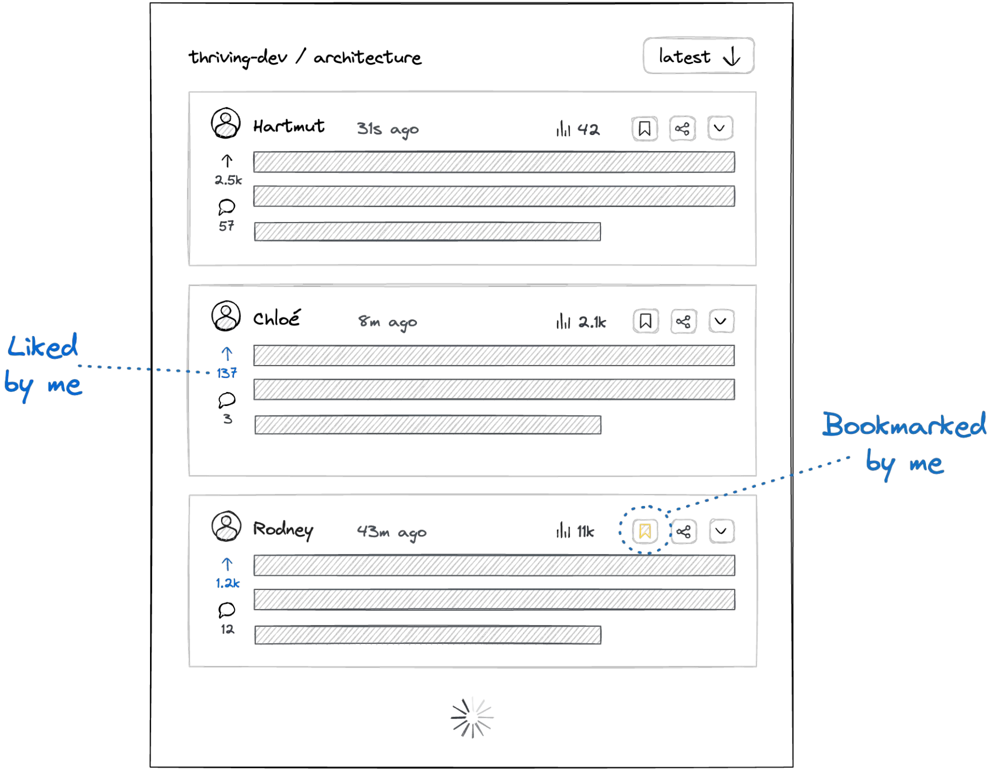


Data

Entities & Relationships

User Likes Post	
user_id	text
post_id	text
created_at	timestamp

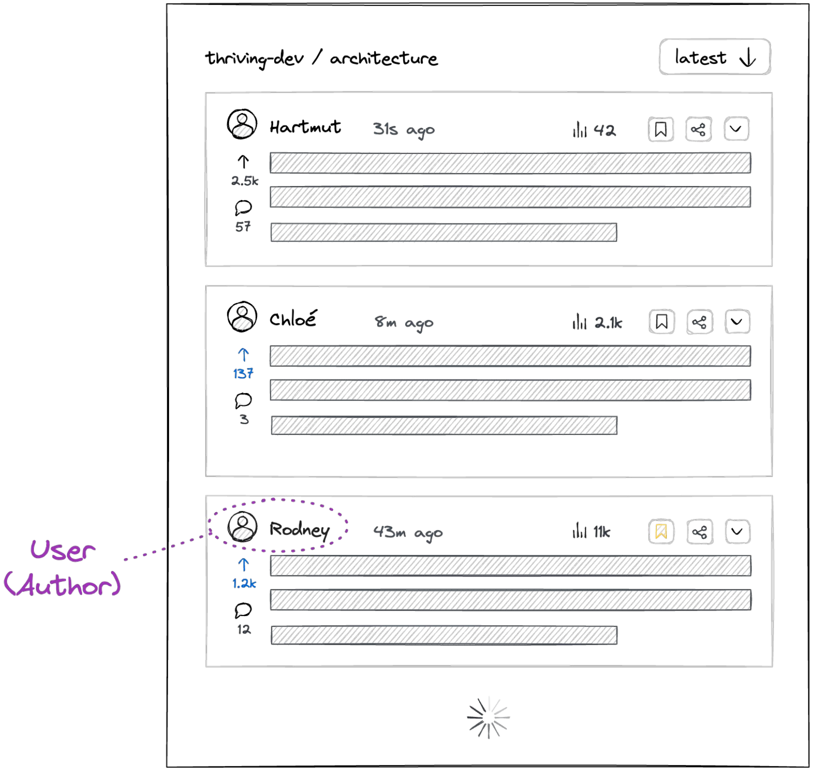
User Bookmarked Post	
user_id	text
post_id	text
created_at	timestamp



Data

Entities & Relationships

User	
user_id	text
username	text
name	text
avatar_url	text
created_at	timestamp
updated_at	timestamp

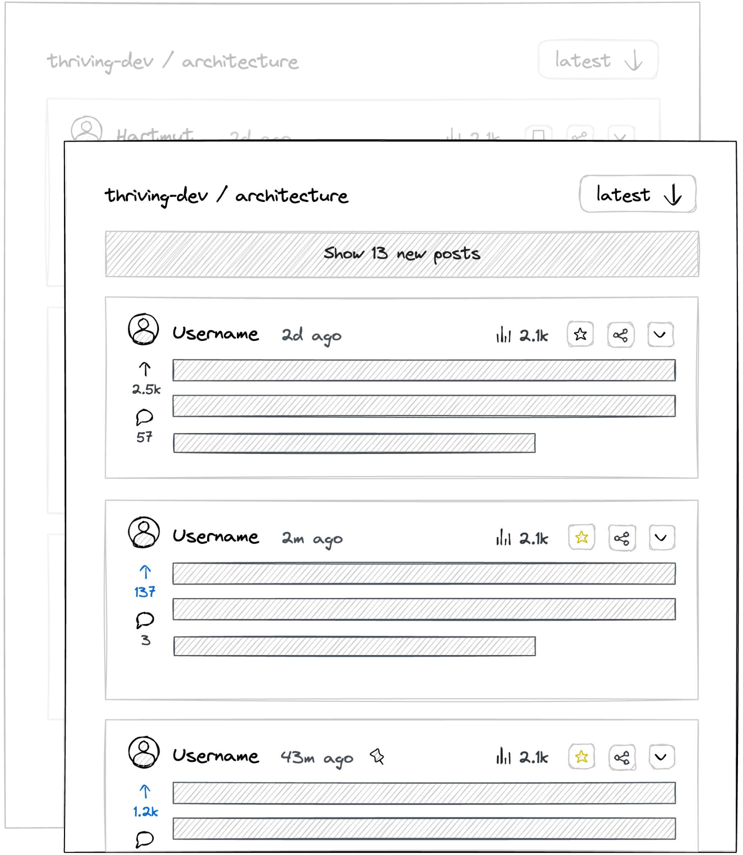


Access Patterns

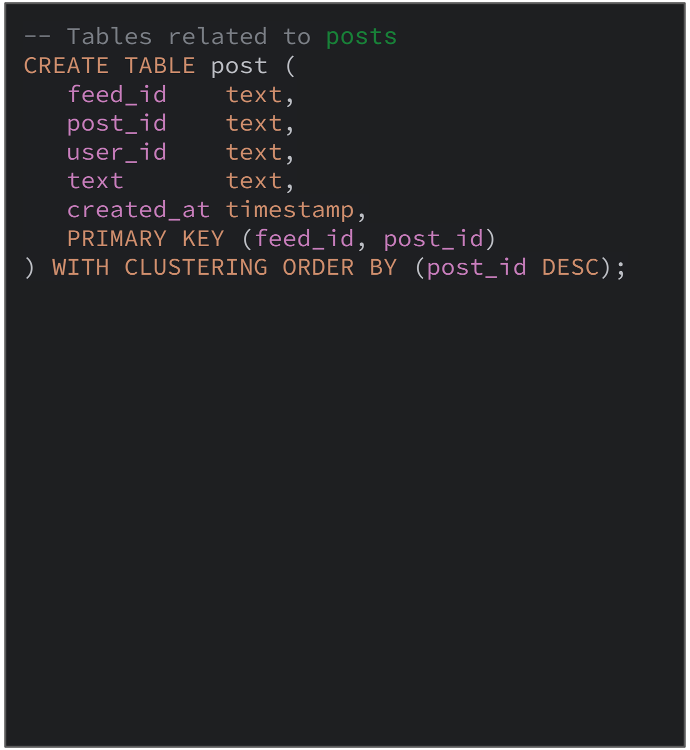
"Query-First Approach"

Access Patterns

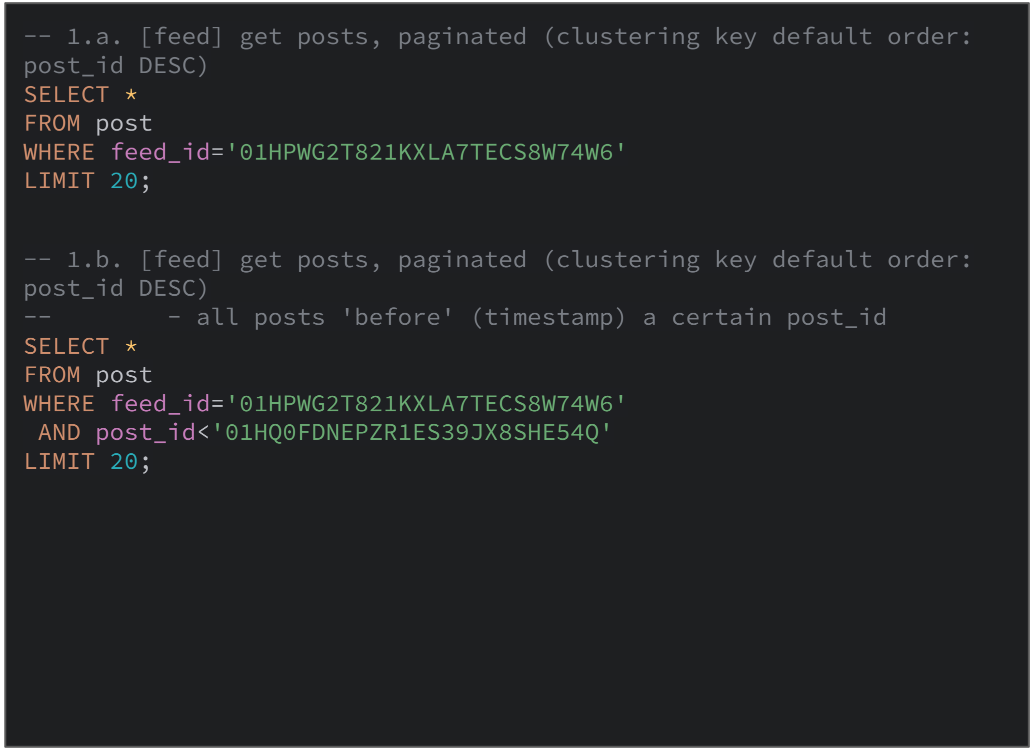
1. [feed] get posts
   - paginated, ordered (created DESC)
   - pinned query time for pagination
2. [post] get stats
   - no. of impressions, likes, comments
3. [post<>user] get session-user relations
   - likes a post
   - has bookmarked a post
4. [post] get author
5. [feed] get 'new posts' count



Data Schema (1)

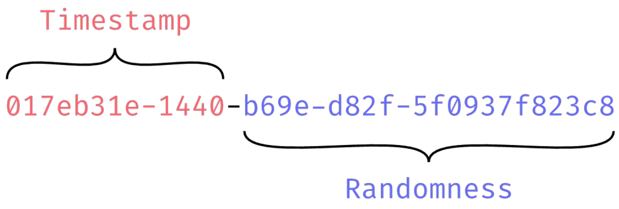


```
-- Tables related to posts
CREATE TABLE post (
   feed_id    text,
   post_id    text,
   user_id    text,
   text       text,
   created_at timestamp,
   PRIMARY KEY (feed_id, post_id)
) WITH CLUSTERING ORDER BY (post_id DESC);
```



```
-- 1.a. [feed] get posts, paginated (clustering key default order:
post_id DESC)
SELECT *
FROM post
WHERE feed_id='01HPWG2T821KXLA7TECS8W74W6'
LIMIT 20;

-- 1.b. [feed] get posts, paginated (clustering key default order:
post_id DESC)
--          - all posts 'before' (timestamp) a certain post_id
SELECT *
FROM post
WHERE feed_id='01HPWG2T821KXLA7TECS8W74W6'
 AND post_id<'01HQ0FDNEPZR1ES39JX8SHE54Q'
LIMIT 20;
```

ULID



Timestamp

`017eb31e-1440-b69e-d82f-5f0937f823c8`

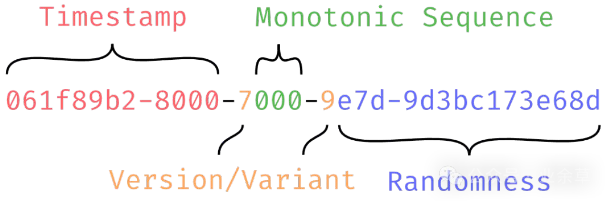
Randomness

⇒ `0GWWXY2G84DFMRVWQNJ1SRYCMC`

Universally Unique
Lexicographically Sortable Identifier

- **Canonically encoded** (26 chars)
- Case insensitive, URL safe
- 1.21e+24 unique ULIDs per millisecond

UUID v7



Timestamp

Monotonic Sequence

`061f89b2-8000-7000-9e7d-9d3bc173e68d`

Version/Variant

Randomness

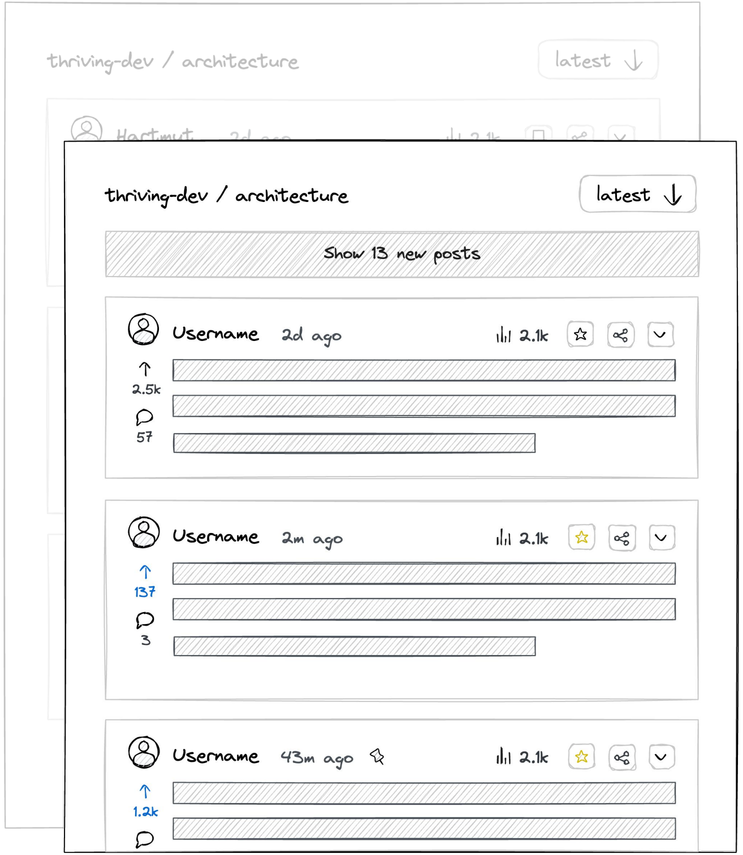
UUIDv7
(RFC 9562)

- 128-bit compatibility with prev. UUID Standards
- Part of ongoing standardization efforts

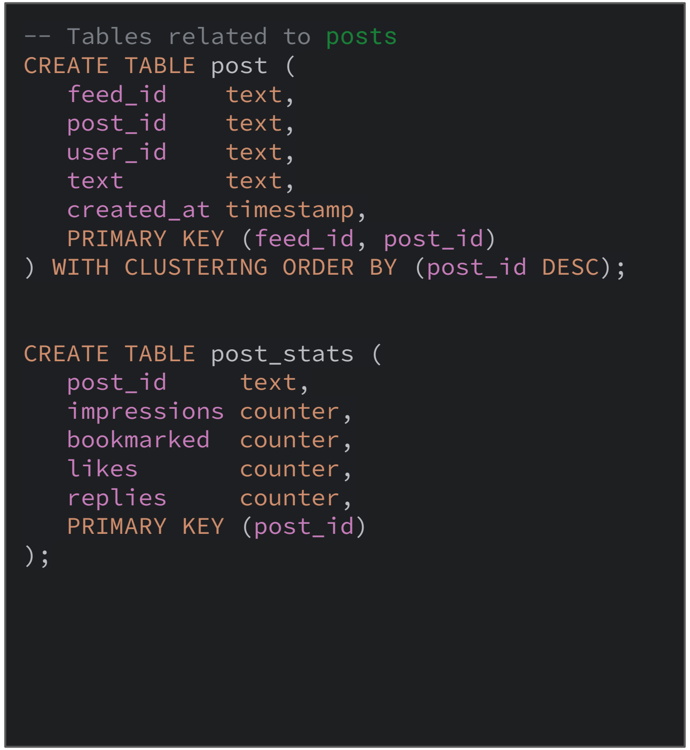
[1]: https://github.com/ulid/spec
[2]: https://datatracker.ietf.org/doc/rfc9562/
[3]: https://itnext.io/why-uuid7-is-better-than-uuid4-as-clustered-index-edb02bf70056

Access Patterns

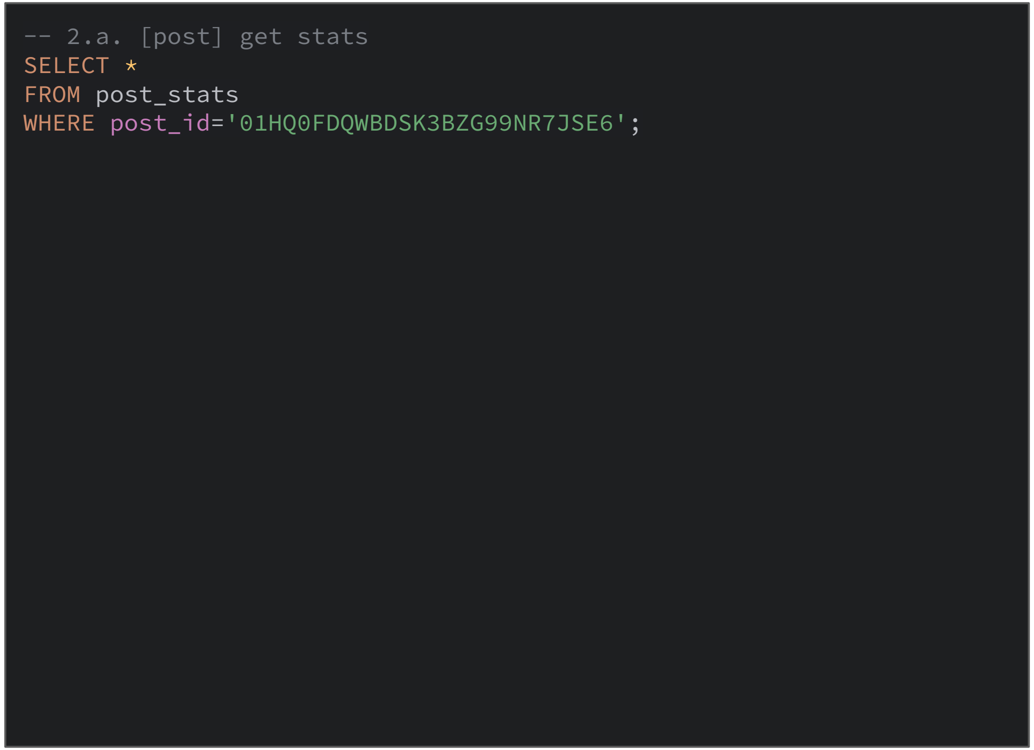
1. [feed] get posts ✅
   - paginated, ordered (created DESC)
   - pinned query time for pagination
2. [post] get stats
   - no. of impressions, likes, comments
3. [post<>user] get session-user relations
   - likes a post
   - has bookmarked a post
4. [post] get author
5. [feed] get 'new posts' count



Data Schema (2)



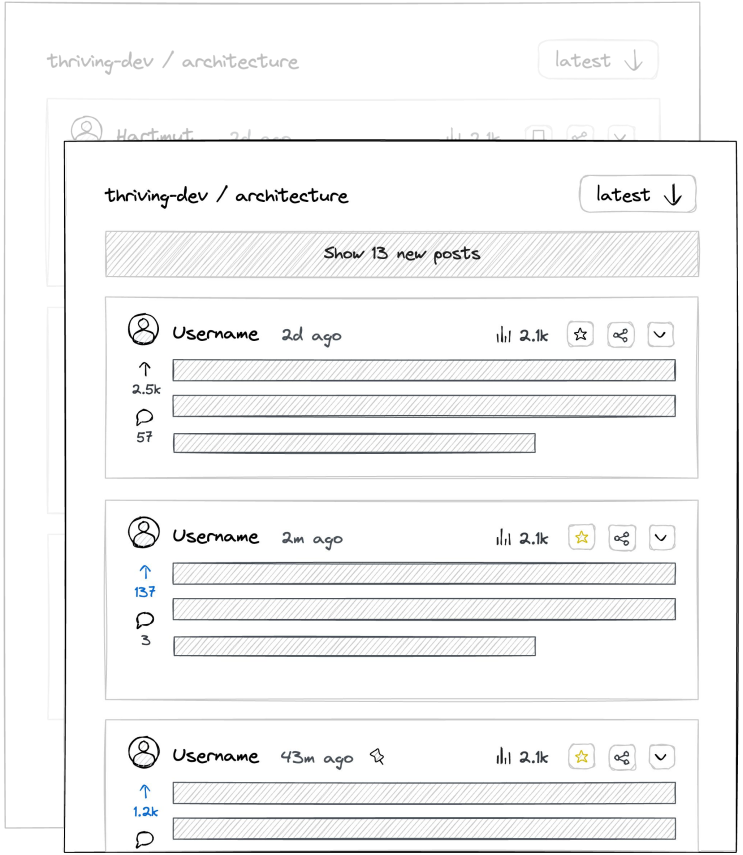
```
-- Tables related to posts
CREATE TABLE post (
   feed_id    text,
   post_id    text,
   user_id    text,
   text       text,
   created_at timestamp,
   PRIMARY KEY (feed_id, post_id)
) WITH CLUSTERING ORDER BY (post_id DESC);


CREATE TABLE post_stats (
   post_id     text,
   impressions counter,
   bookmarked  counter,
   likes       counter,
   replies     counter,
   PRIMARY KEY (post_id)
);
```



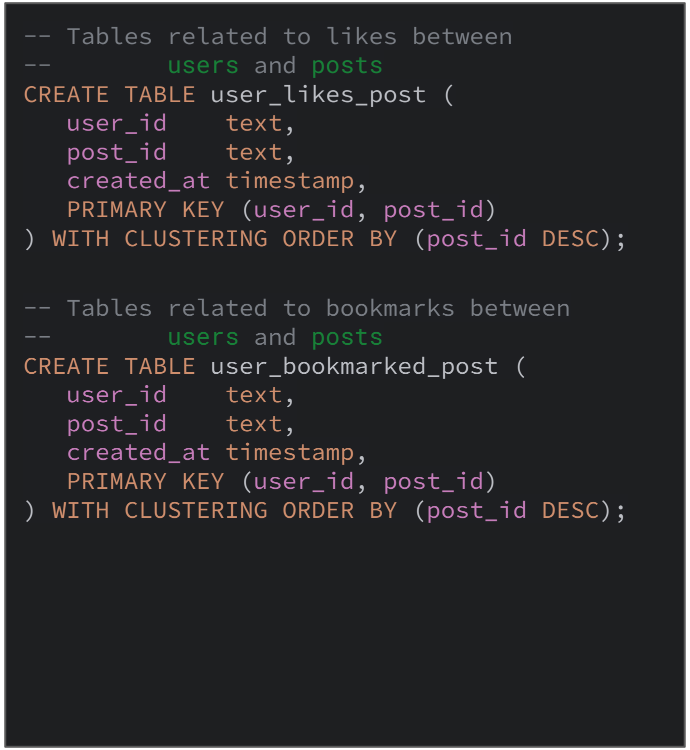
```
-- 2.a. [post] get stats
SELECT *
FROM post_stats
WHERE post_id='01HQ0FDQWBDSK3BZG99NR7JSE6';
```

Access Patterns

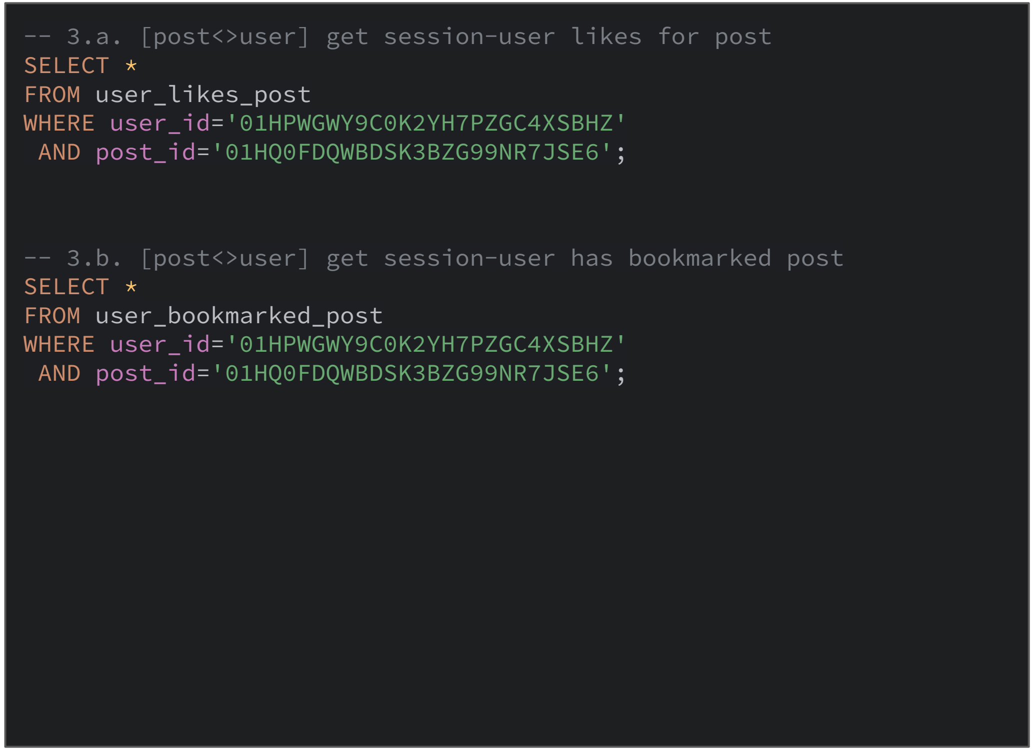
1. [feed] get posts ✅
   - paginated, ordered (created DESC)
   - pinned query time for pagination
2. [post] get stats ✅
   - no. of impressions, likes, comments
3. [post<>user] get session-user relations
   - likes a post
   - has bookmarked a post
4. [post] get author
5. [feed] get 'new posts' count



Data Schema (3)



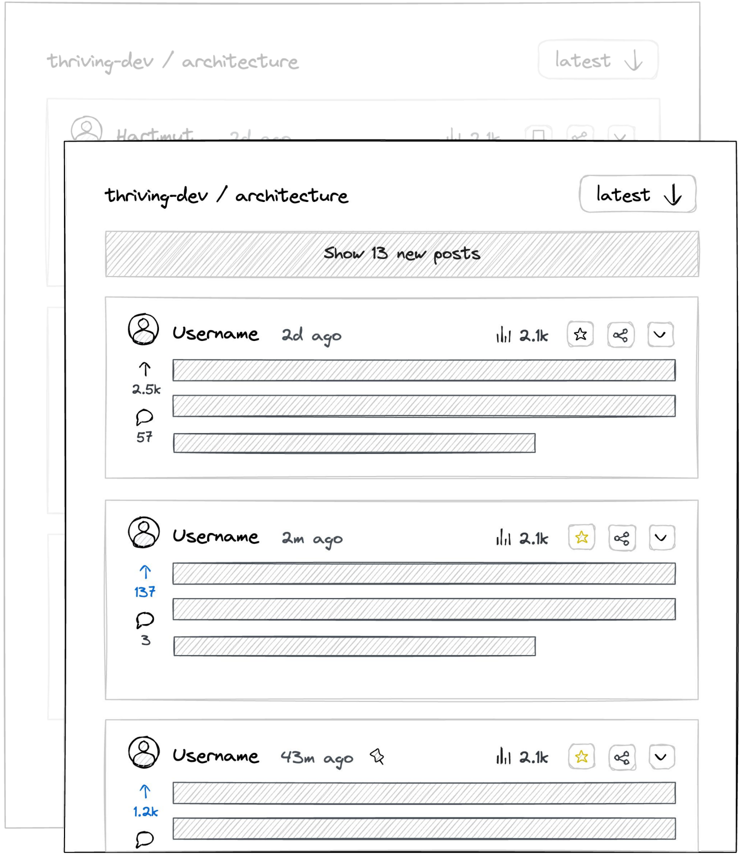
```
-- Tables related to likes between
--         users and posts
CREATE TABLE user_likes_post (
   user_id    text,
   post_id    text,
   created_at timestamp,
   PRIMARY KEY (user_id, post_id)
) WITH CLUSTERING ORDER BY (post_id DESC);

-- Tables related to bookmarks between
--         users and posts
CREATE TABLE user_bookmarked_post (
   user_id    text,
   post_id    text,
   created_at timestamp,
   PRIMARY KEY (user_id, post_id)
) WITH CLUSTERING ORDER BY (post_id DESC);
```



```
-- 3.a. [post<>user] get session-user likes for post
SELECT *
FROM user_likes_post
WHERE user_id='01HPWGWY9C0K2YH7PZGC4XSBHZ'
 AND post_id='01HQ0FDQWBDSK3BZG99NR7JSE6';


-- 3.b. [post<>user] get session-user has bookmarked post
SELECT *
FROM user_bookmarked_post
WHERE user_id='01HPWGWY9C0K2YH7PZGC4XSBHZ'
 AND post_id='01HQ0FDQWBDSK3BZG99NR7JSE6';
```

Access Patterns

1. [feed] get posts ✅
   - paginated, ordered (created DESC)
   - pinned query time for pagination
2. [post] get stats ✅
   - no. of impressions, likes, comments
3. [post<>user] get session-user relations ✅
   - likes a post
   - has bookmarked a post
4. [post] get author
5. [feed] get 'new posts' count



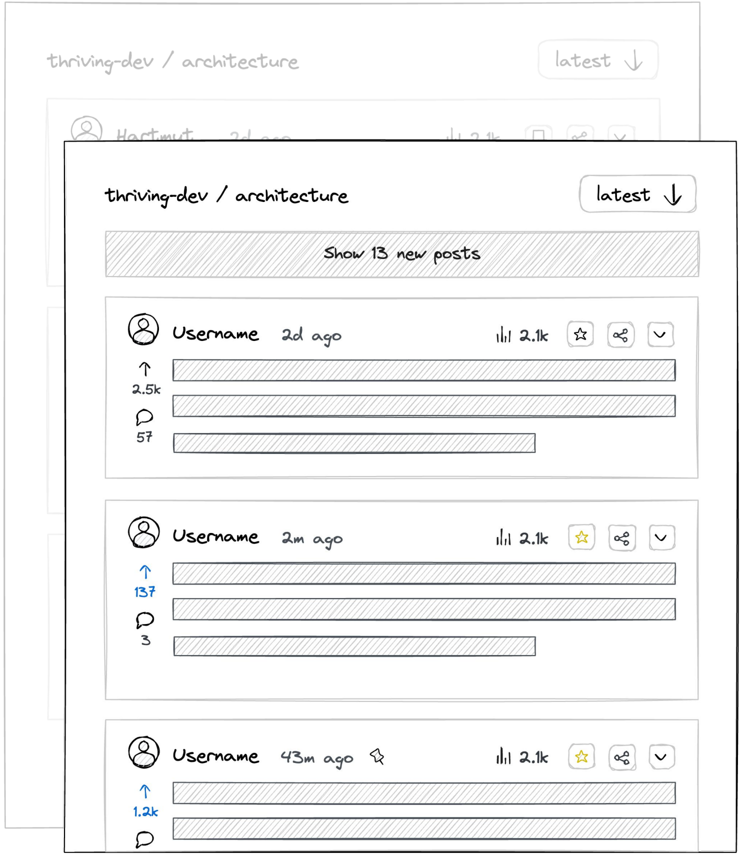
Data Schema (4)

```
-- Tables related to users
CREATE TABLE user (
   user_id       text,
   username      text,
   password_hash text,
   name          text,
   avatar_url    text,
   created_at    timestamp,
   updated_at    timestamp,
   PRIMARY KEY (user_id)
);
```

```
-- 4.a. [post] get author
SELECT *
FROM user
WHERE user_id='01HPWGWY9C0K2YH7PZGC4XSBHZ';
```

Access Patterns

1. [feed] get posts ✅
   - paginated, ordered (created DESC)
   - pinned query time for pagination
2. [post] get stats ✅
   - no. of impressions, likes, comments
3. [post<>user] get session-user relations ✅
   - likes a post
   - has bookmarked a post
4. [post] get author ✅
5. [feed] get 'new posts' count



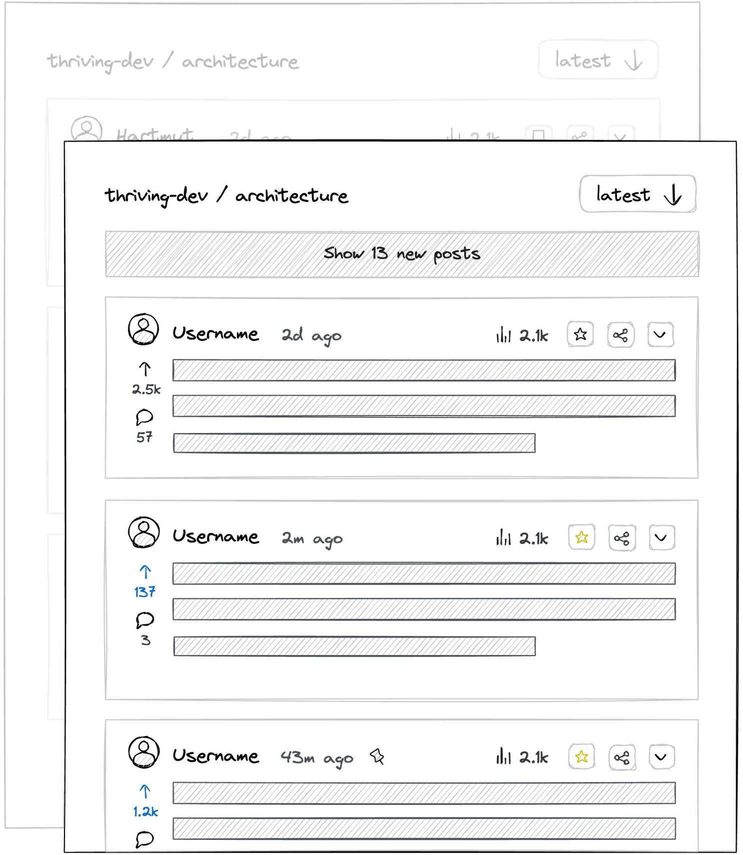
Data Schema (5)

```
-- Tables related to posts
CREATE TABLE post (
   feed_id    text,
   post_id    text,
   user_id    text,
   text       text,
   created_at timestamp,
   PRIMARY KEY (feed_id, post_id)
) WITH CLUSTERING ORDER BY (post_id DESC);
```

```
-- 5.a. [feed] get 'new posts' count
-- ⚠️ DON'T DO THIS AT HOME: using COUNT()
--    (note: see better query 5.b. below)
~~SELECT COUNT(*)~~01HPWG2T821KXLA7TECS8W74W6'
~~FROM post~~_id>='01HQ0FDNEPZR1ES39JX8SHE54Q';
~~WHERE feed_id='01HPWG2T821KXLA7TECS8W74W6'~~
~~ AND post_id>='01HQ0FDNEPZR1ES39JX8SHE54Q'~~;


-- 5.b. [feed] get 'new posts' count
-- 🧐 avoids COUNT() -> query IDs with upper LIMIT,
--    manually count rows returned in client
SELECT post_id
FROM post
WHERE feed_id='01HPWG2T821KXLA7TECS8W74W6'
 AND post_id>='01HQ0FDNEPZR1ES39JX8SHE54Q'
LIMIT 1000; -- display like: '1'; '721'; '>1000'
```

Access Patterns

1. [feed] get posts ✅
   - paginated, ordered (created DESC)
   - pinned query time for pagination
2. [post] get stats ✅
   - no. of impressions, likes, comments
3. [post<>user] get session-user relations ✅
   - likes a post
   - has bookmarked a post
4. [post] get author ✅
5. [feed] get 'new posts' count ✅



Process Flow

& Iterative Refinement

Social Platform Feed

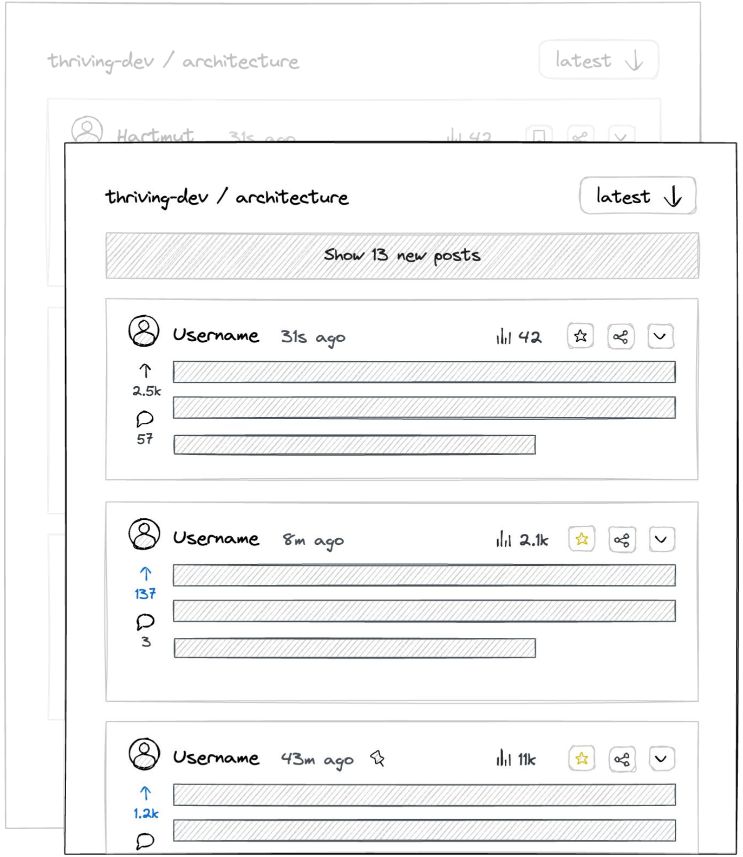
RESTful API

Specification

```
GET /feeds/{feedId}/posts
    ?pageSize=20
    &ltPostId=01HZMN4H77QBW1W8PS6W04ZH9V
```

Provides

- **all data** required to render the 'Social Platform Feed'
- for an authenticated **user** or guest
- with a **single request/response**



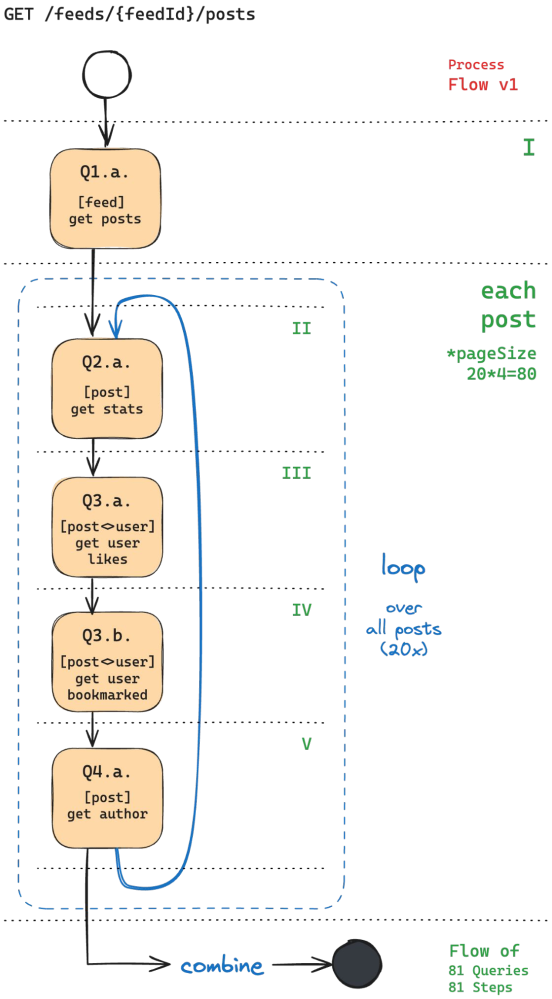
API Handler Process Flow (v1)

The most basic approach...

...fetch a 'page' of 20 posts
...iterate and enrich

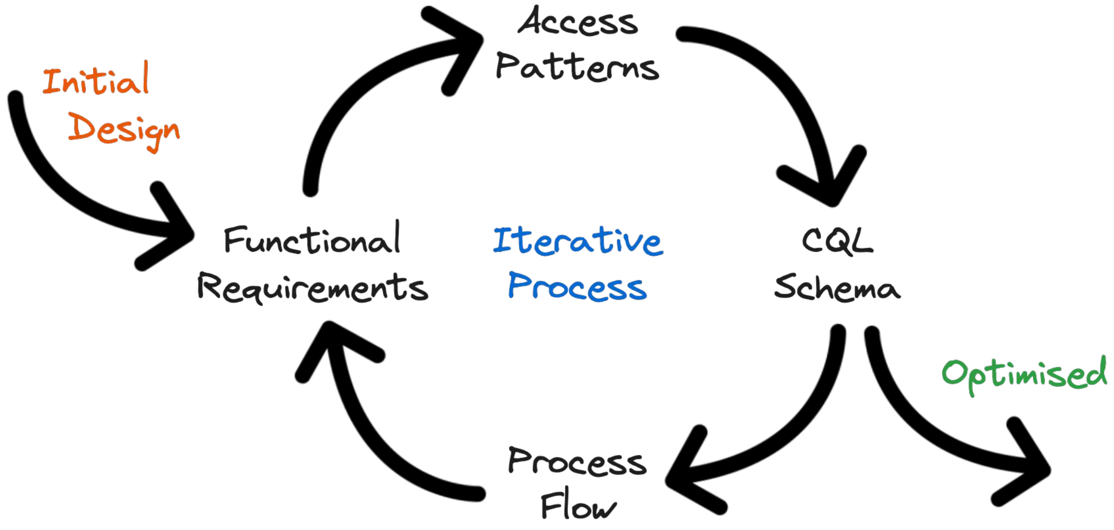
-> **81 queries** to the DB 🙀
-> **81 subsequent** steps (IO)

🤔 surely we can do better...??

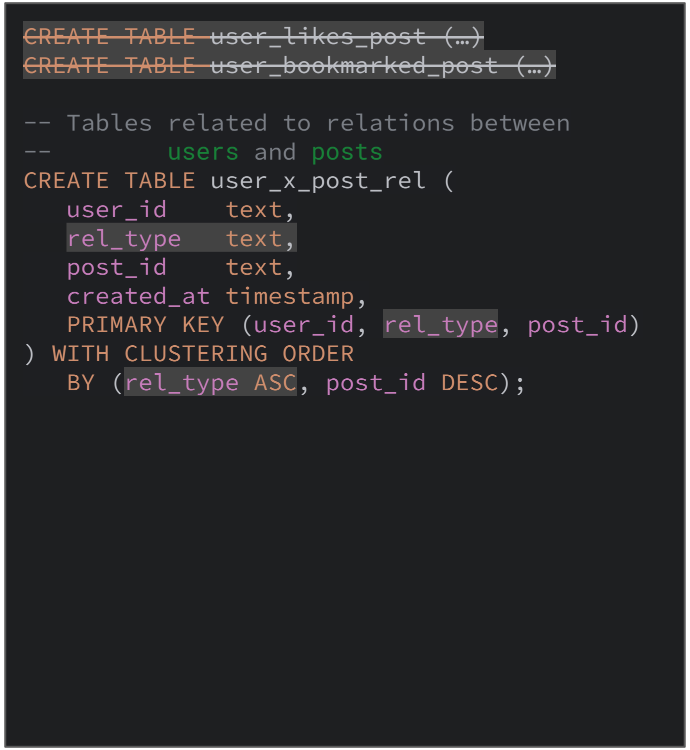


Data Modelling

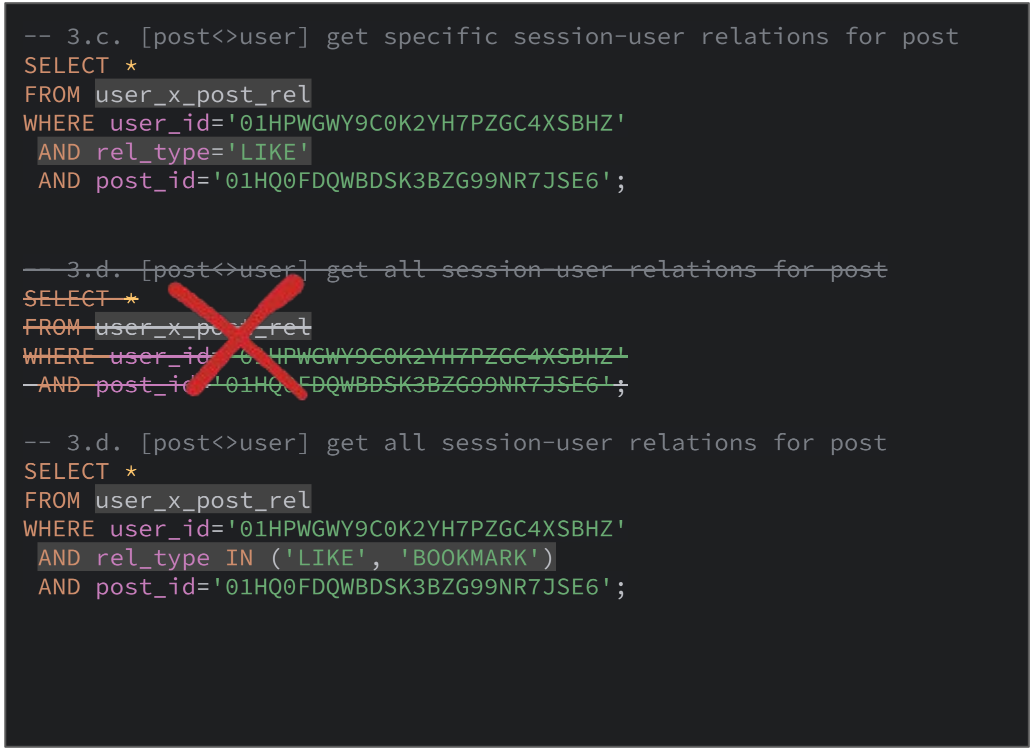
Iterative Process of Schema Refinement



Data Schema (6)



```
~~CREATE TABLE user_likes_post (…)~~
~~CREATE TABLE user_bookmarked_post (…)~~

-- Tables related to relations between
--          users and posts
CREATE TABLE user_x_post_rel (
   user_id     text,
   rel_type    text,
   post_id     text,
   created_at timestamp,
   PRIMARY KEY (user_id, rel_type, post_id)
) WITH CLUSTERING ORDER
   BY (rel_type ASC, post_id DESC);
```



```
-- 3.c. [post<>user] get specific session-user relations for post
SELECT *
FROM user_x_post_rel
WHERE user_id='01HPWGWY9C0K2YH7PZGC4XSBHZ'
 AND rel_type='LIKE'
 AND post_id='01HQ0FDQWBDSK3BZG99NR7JSE6';


~~-- 3.d. [post<>user] get all session-user relations for post~~
~~SELECT *~~
~~FROM user_x_post_rel~~
~~WHERE user_id='01HPWGWY9C0K2YH7PZGC4XSBHZ'~~
~~ AND post_id='01HQ0FDQWBDSK3BZG99NR7JSE6';~~

-- 3.d. [post<>user] get all session-user relations for post
SELECT *
FROM user_x_post_rel
WHERE user_id='01HPWGWY9C0K2YH7PZGC4XSBHZ'
 AND rel_type IN ('LIKE', 'BOOKMARK')
 AND post_id='01HQ0FDQWBDSK3BZG99NR7JSE6';
```

API Handler Process Flow (v2)

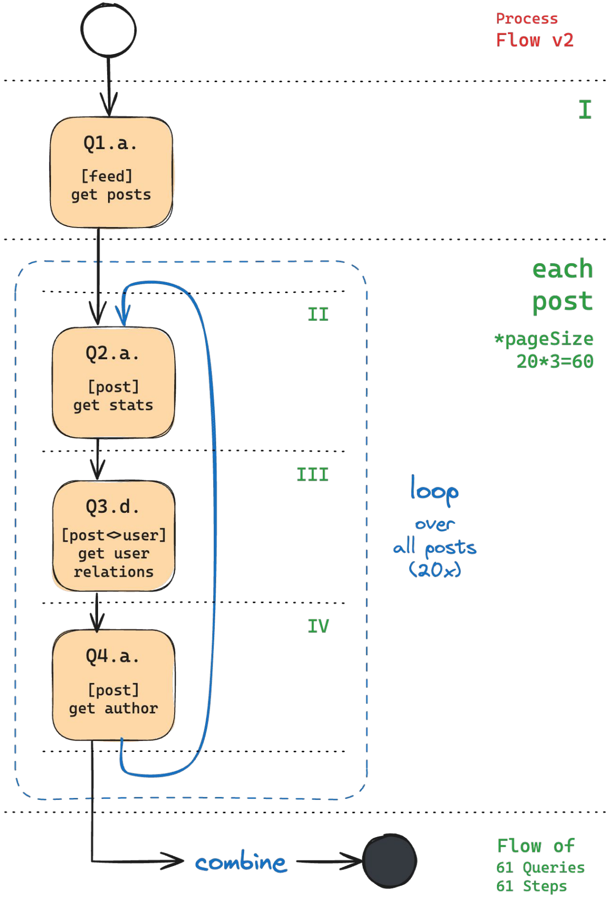
Still a basic approach...

...fetch a 'page' of 20 posts
...iterate and enrich **x3**

-> **61 queries** to the DB
-> **61 subsequent** steps (IO)

😾 ...still have to do better!!

GET /feeds/{feedId}/posts



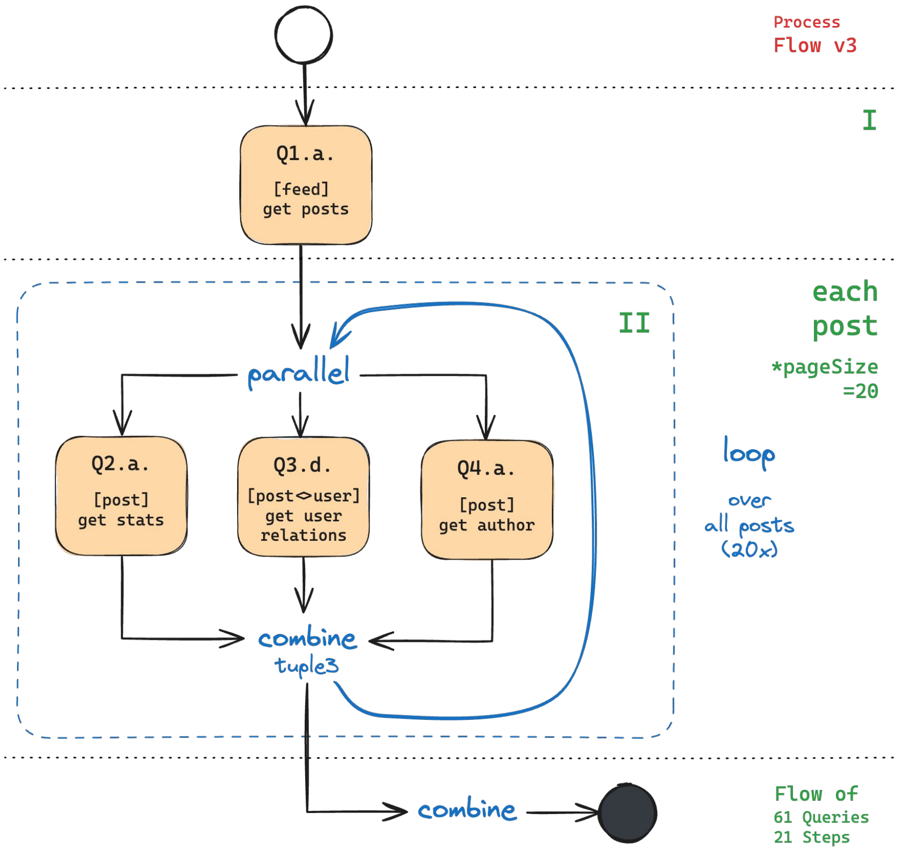
API Handler Process Flow (v3)

Parallel Processing

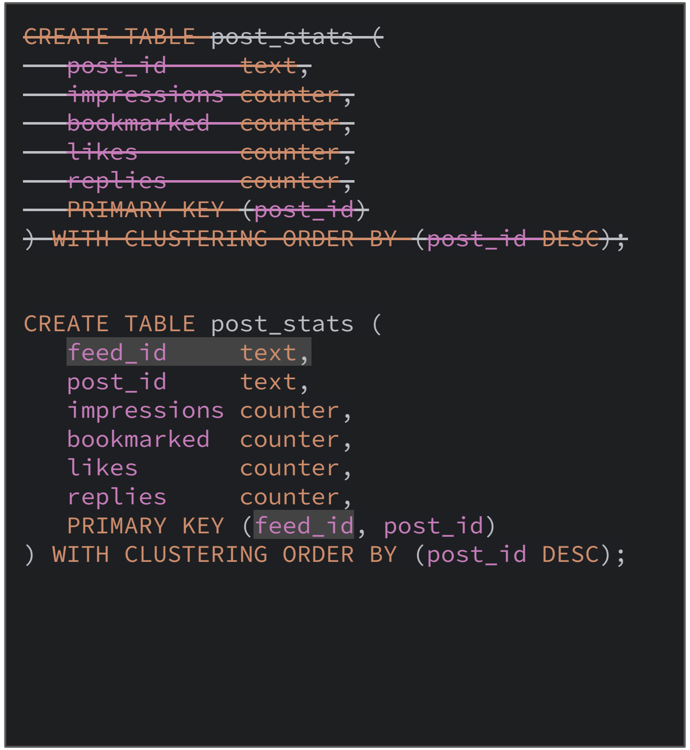
...fetch a 'page' of 20 posts
...iterate and enrich **in parallel**

-> **61 queries** to the DB
-> **21 subsequent** steps (IO)

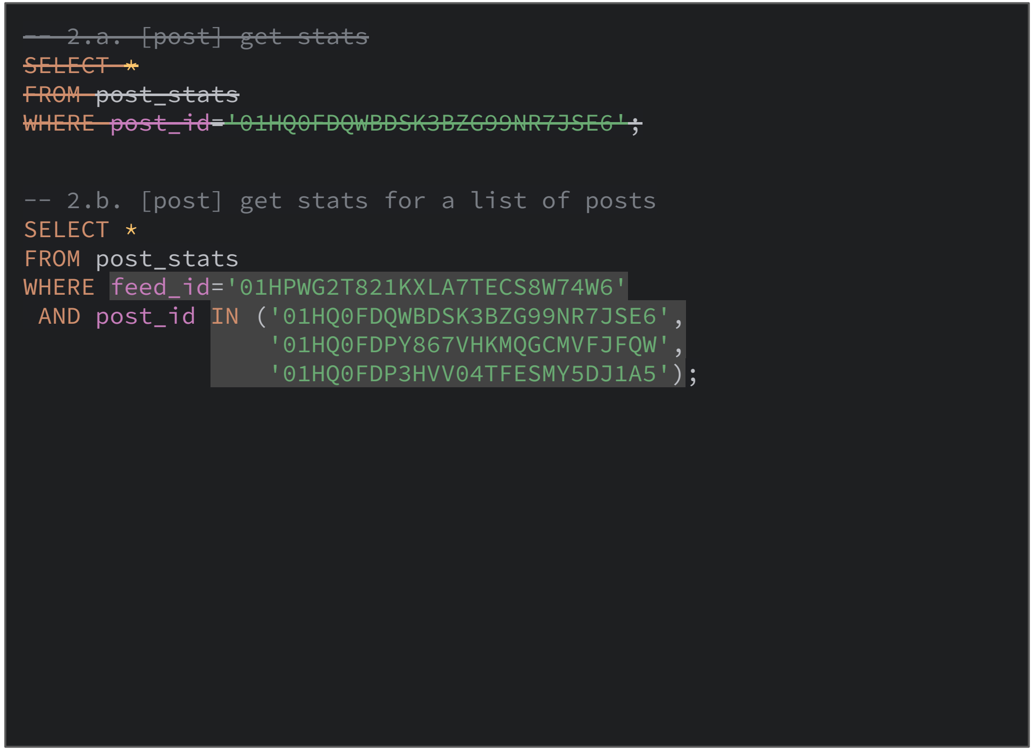
GET /feeds/{feedId}/posts



Data Schema (7)

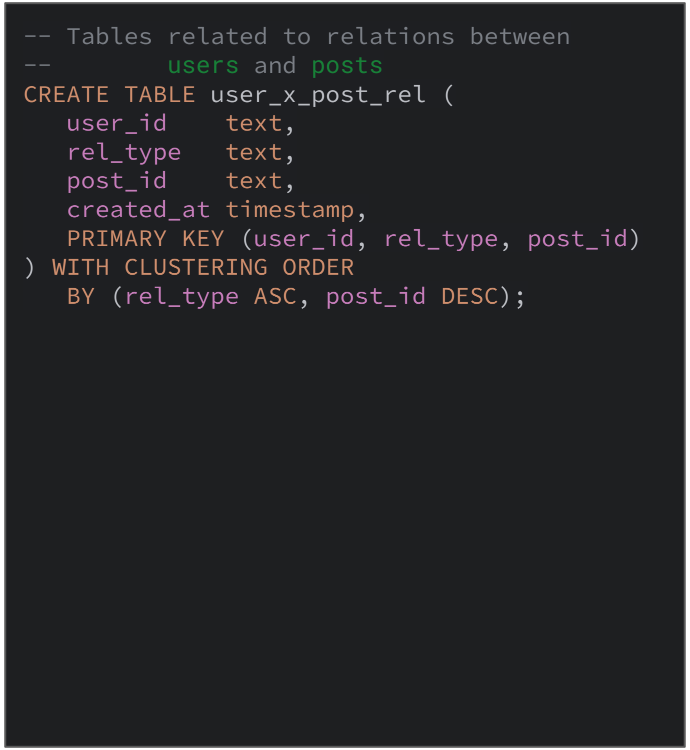


```
~~CREATE TABLE post_stats (~~
~~    post_id     text,~~
~~    impressions counter,~~
~~    bookmarked  counter,~~
~~    likes       counter,~~
~~    replies     counter,~~
~~    PRIMARY KEY (post_id)~~
~~) WITH CLUSTERING ORDER BY (post_id DESC);~~


CREATE TABLE post_stats (
   feed_id     text,
   post_id     text,
   impressions counter,
   bookmarked  counter,
   likes       counter,
   replies     counter,
   PRIMARY KEY (feed_id, post_id)
) WITH CLUSTERING ORDER BY (post_id DESC);
```

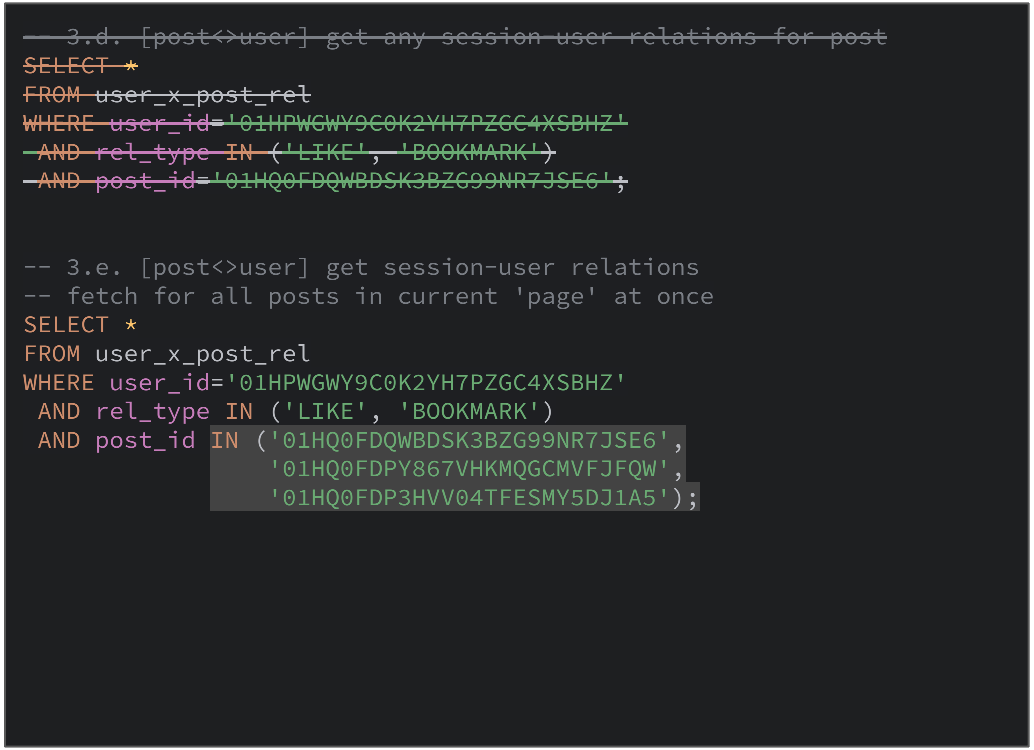


```
~~-- 2.a. [post] get stats~~
~~SELECT *~~
~~FROM post_stats~~
~~WHERE post_id='01HQ0FDQWBDSK3BZG99NR7JSE6';~~

-- 2.b. [post] get stats for a list of posts
SELECT *
FROM post_stats
WHERE feed_id='01HPWG2T821KXLA7TECS8W74W6'
 AND post_id IN ('01HQ0FDQWBDSK3BZG99NR7JSE6',
                 '01HQ0FDPY867VHKMQGCMVFJFQW',
                 '01HQ0FDP3HVV04TFESMY5DJ1A5');
```

Data Schema (8)



```
-- Tables related to relations between
--        users and posts
CREATE TABLE user_x_post_rel (
   user_id    text,
   rel_type   text,
   post_id    text,
   created_at timestamp,
   PRIMARY KEY (user_id, rel_type, post_id)
) WITH CLUSTERING ORDER
   BY (rel_type ASC, post_id DESC);
```



```
~~-- 3.d. [post<>user] get any session-user relations for post~~
~~SELECT *~~
~~FROM user_x_post_rel~~
~~WHERE user_id='01HPWGWY9C0K2YH7PZGC4XSBHZ'~~
~~ AND rel_type IN ('LIKE', 'BOOKMARK')~~
~~ AND post_id='01HQ0FDQWBDSK3BZG99NR7JSE6'~~;


-- 3.e. [post<>user] get session-user relations
-- fetch for all posts in current 'page' at once
SELECT *
FROM user_x_post_rel
WHERE user_id='01HPWGWY9C0K2YH7PZGC4XSBHZ'
 AND rel_type IN ('LIKE', 'BOOKMARK')
 AND post_id IN ('01HQ0FDQWBDSK3BZG99NR7JSE6',
                 '01HQ0FDPY867VHKMQGCMVFJFQW',
                 '01HQ0FDP3HVV04TFESMY5DJ1A5');
```

API Handler Process Flow (v4)

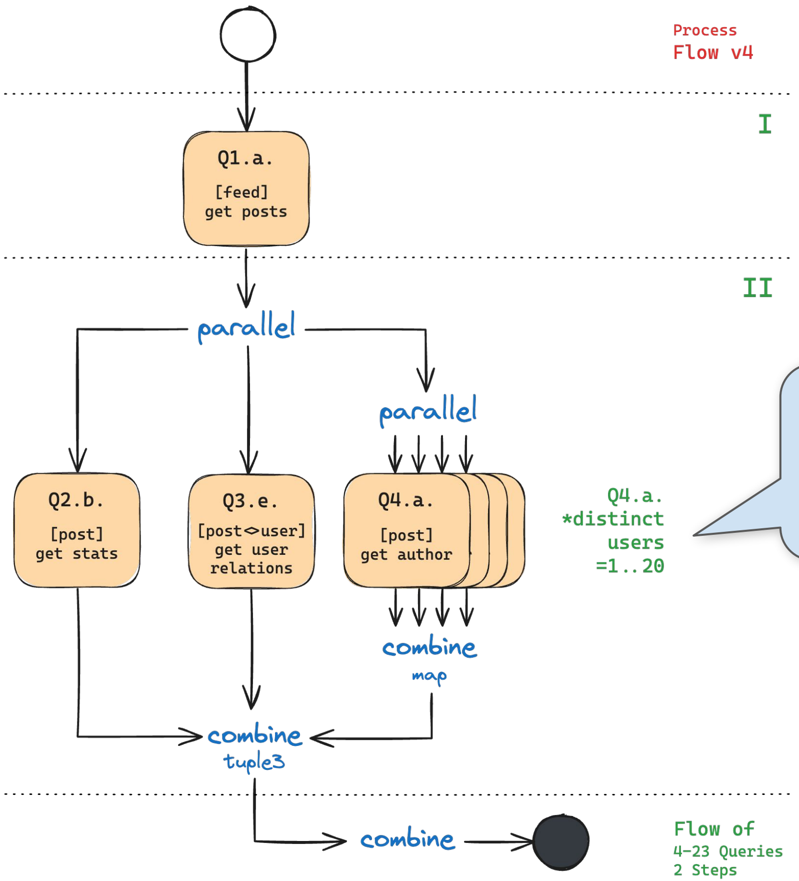
Optimal Parallelisation

...fetch a 'page' of 20 posts
...~~iterate~~ **bulk query** enrichment
...enrich **in parallel**
...lookup **distinct users**

-> **4...23 queries** to the DB
-> **2 subsequent** steps (IO)

🎉

GET /feeds/{feedId}/posts



Good candidate for (in-memory) local cache!

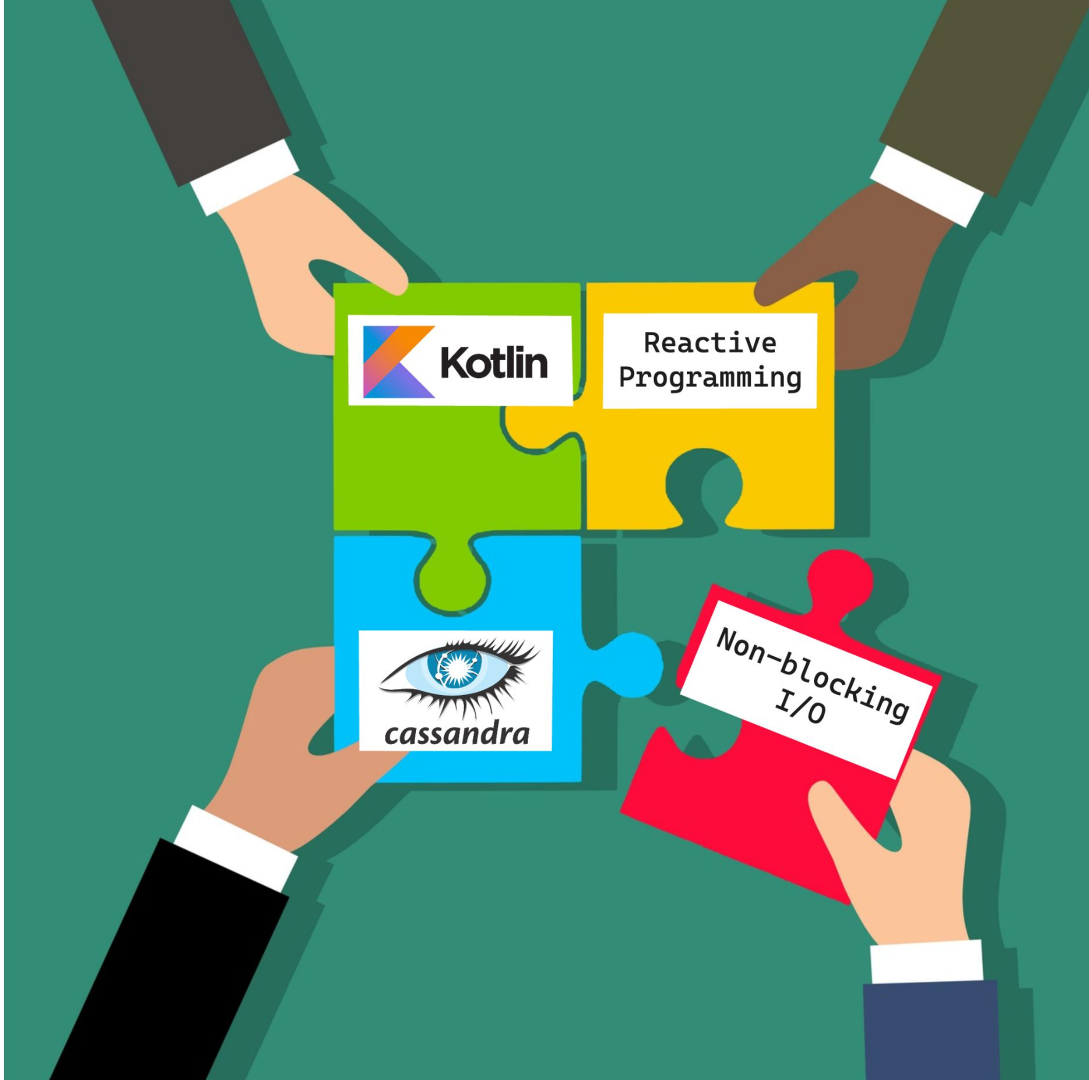
HAVING COMPLETED THE REFINEMENT PROCESS

Application Design

Backend

Application
Tech Stack

- 🚀 Non-blocking IO
- ✨ Reactive Programming
- 🦄 Kotlin



Reactive Programming

Key Benefits

- Asynchronous
  & Non-Blocking Operations
- Declarative Code
- Resilience and Fault Tolerance
- Responsive Systems

Simply put, it's
all about **Data Streams**

- Filter
- Map / Transform
- Reduce
- Collect

Project Reactor

RxJava

MUTINY!

POC

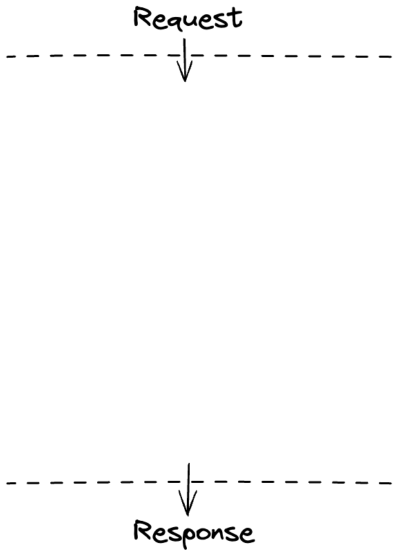
Implementation

QUARKUS MUTINY!

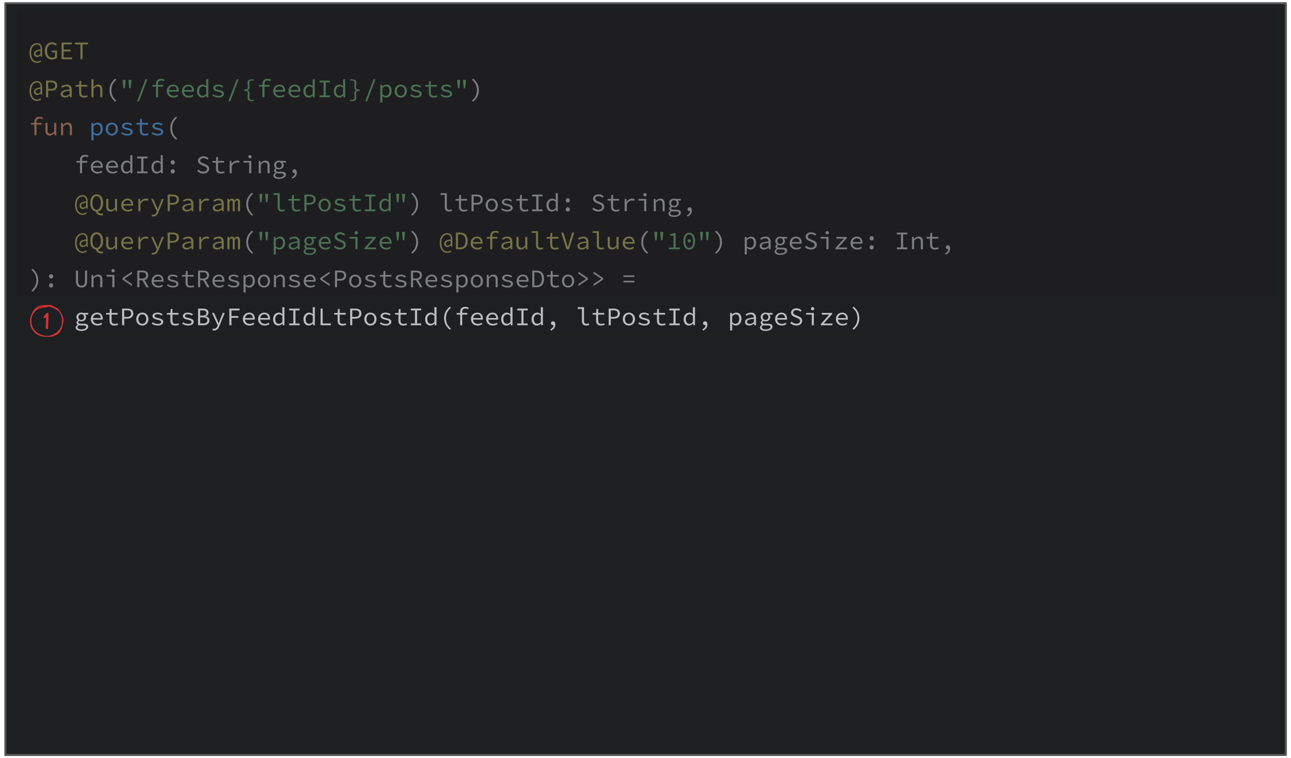
cassandra Kotlin

Code Example

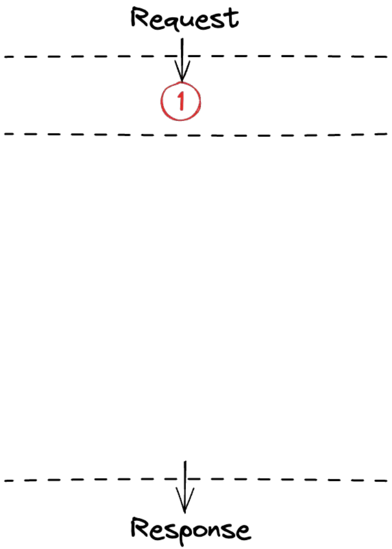
```
@GET
@Path("/feeds/{feedId}/posts")
fun posts(
   feedId: String,
   @QueryParam("ltPostId") ltPostId: String,
   @QueryParam("pageSize") @DefaultValue("10") pageSize: Int,
): Uni<RestResponse<PostsResponseDto>> =
```



Code Example

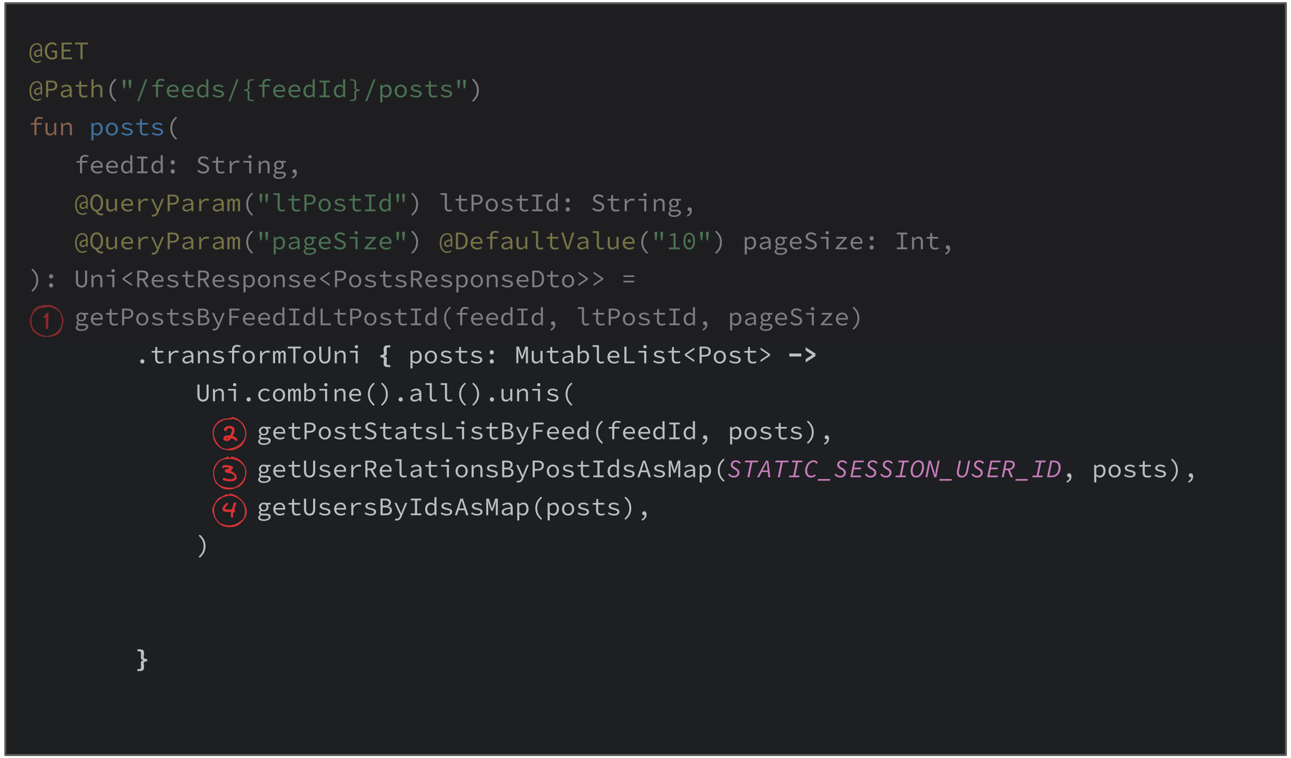


```
@GET
@Path("/feeds/{feedId}/posts")
fun posts(
   feedId: String,
   @QueryParam("ltPostId") ltPostId: String,
   @QueryParam("pageSize") @DefaultValue("10") pageSize: Int,
): Uni<RestResponse<PostsResponseDto>> =
① getPostsByFeedIdLtPostId(feedId, ltPostId, pageSize)
```

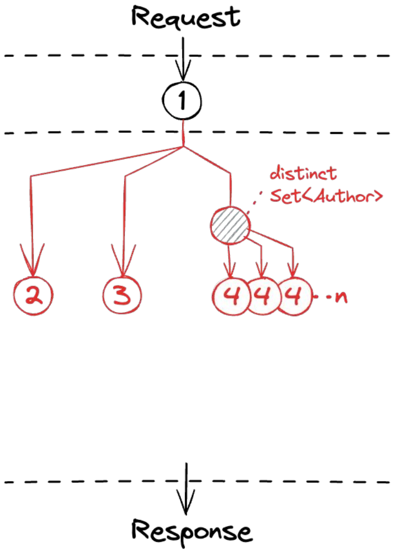


① (1.b.) SELECT post[]

Code Example



```
@GET
@Path("/feeds/{feedId}/posts")
fun posts(
   feedId: String,
   @QueryParam("ltPostId") ltPostId: String,
   @QueryParam("pageSize") @DefaultValue("10") pageSize: Int,
): Uni<RestResponse<PostsResponseDto>> =
① getPostsByFeedIdLtPostId(feedId, ltPostId, pageSize)
       .transformToUni { posts: MutableList<Post> ->
           Uni.combine().all().unis(
            ② getPostStatsListByFeed(feedId, posts),
            ③ getUserRelationsByPostIdsAsMap(STATIC_SESSION_USER_ID, posts),
            ④ getUsersByIdsAsMap(posts),
           )


       }
```



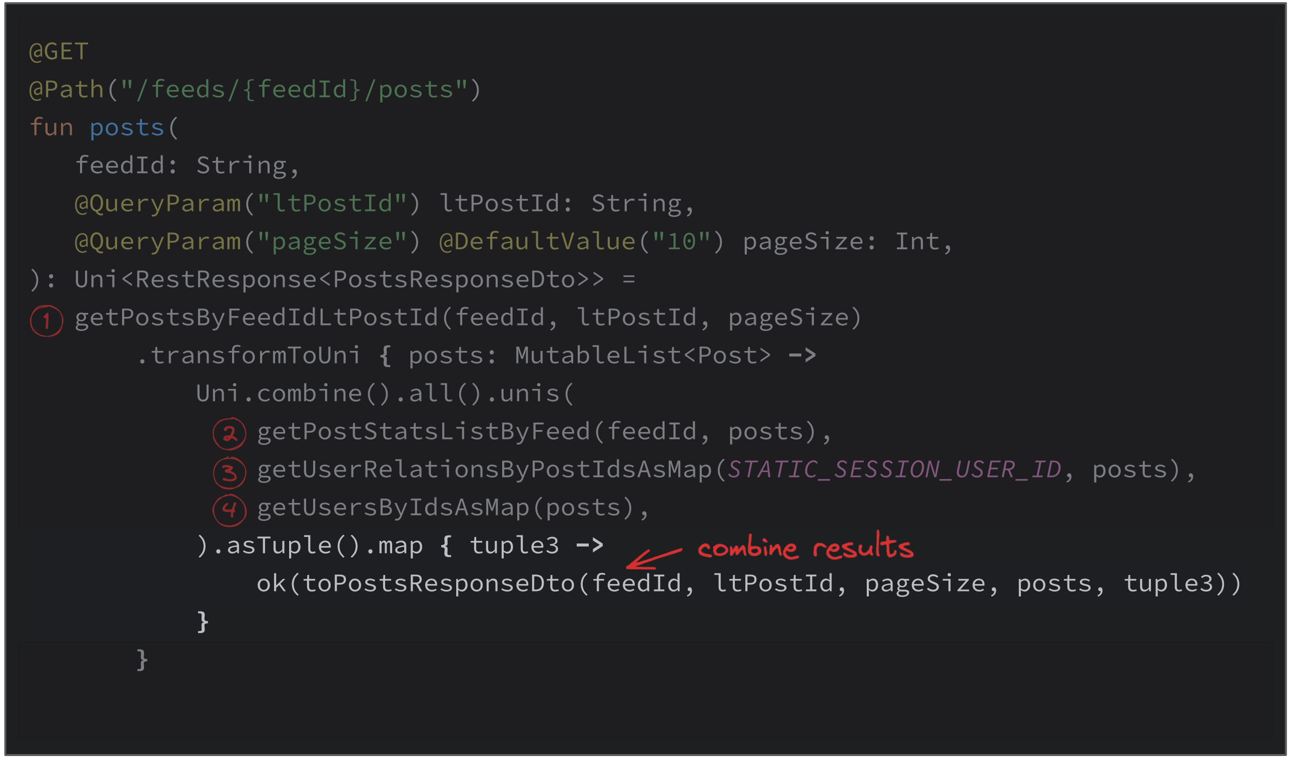
① (1.b.) SELECT post[]

② (2.b.) SELECT post_stats[]

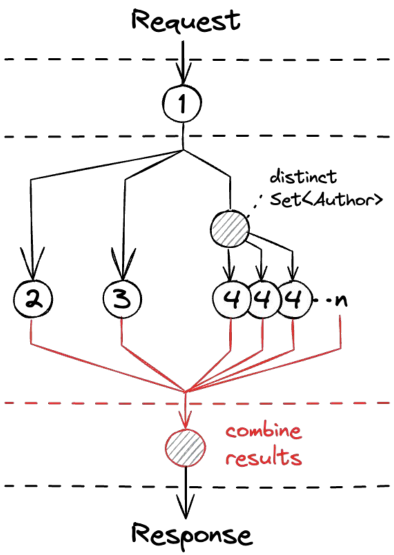
③ (3.e.) SELECT user_x_post_rel[]

④ (4.a.) SELECT user

Code Example



```
@GET
@Path("/feeds/{feedId}/posts")
fun posts(
   feedId: String,
   @QueryParam("ltPostId") ltPostId: String,
   @QueryParam("pageSize") @DefaultValue("10") pageSize: Int,
): Uni<RestResponse<PostsResponseDto>> =
① getPostsByFeedIdLtPostId(feedId, ltPostId, pageSize)
       .transformToUni { posts: MutableList<Post> ->
           Uni.combine().all().unis(
            ② getPostStatsListByFeed(feedId, posts),
            ③ getUserRelationsByPostIdsAsMap(STATIC_SESSION_USER_ID, posts),
            ④ getUsersByIdsAsMap(posts),
           ).asTuple().map { tuple3 ->  ← combine results
               ok(toPostsResponseDto(feedId, ltPostId, pageSize, posts, tuple3))
           }
       }
```



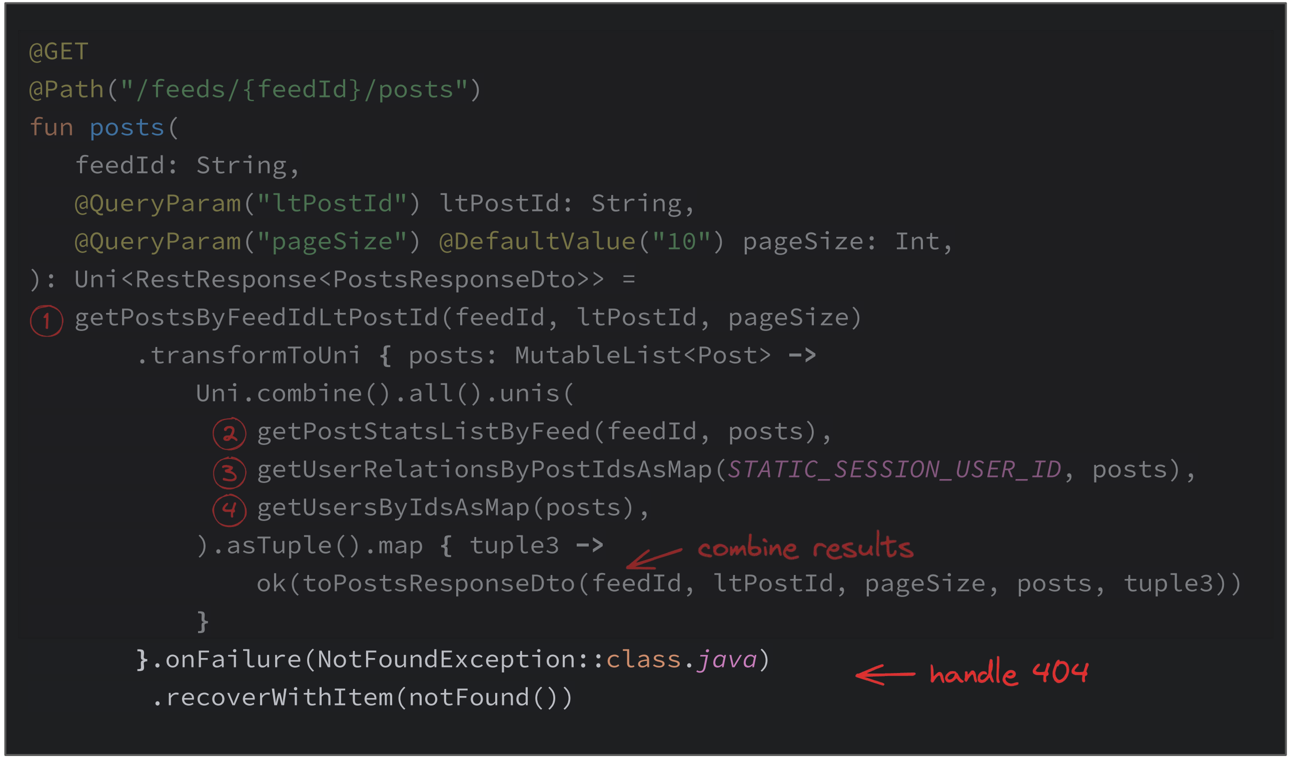
① (1.b.) SELECT post[]

② (2.b.) SELECT post_stats[]

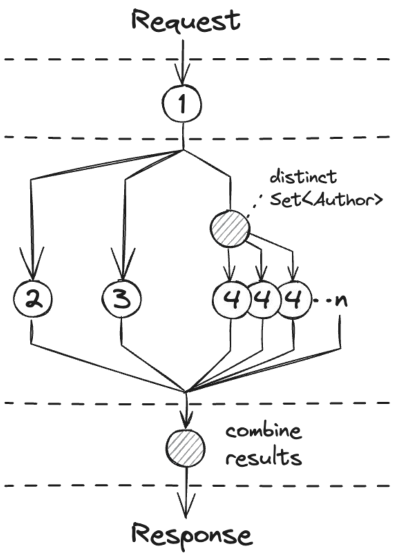
③ (3.e.) SELECT user_x_post_rel[]

④ (4.a.) SELECT user

Code Example



```
@GET
@Path("/feeds/{feedId}/posts")
fun posts(
   feedId: String,
   @QueryParam("ltPostId") ltPostId: String,
   @QueryParam("pageSize") @DefaultValue("10") pageSize: Int,
): Uni<RestResponse<PostsResponseDto>> =
① getPostsByFeedIdLtPostId(feedId, ltPostId, pageSize)
       .transformToUni { posts: MutableList<Post> ->
           Uni.combine().all().unis(
            ② getPostStatsListByFeed(feedId, posts),
            ③ getUserRelationsByPostIdsAsMap(STATIC_SESSION_USER_ID, posts),
            ④ getUsersByIdsAsMap(posts),
           ).asTuple().map { tuple3 ->   ← combine results
               ok(toPostsResponseDto(feedId, ltPostId, pageSize, posts, tuple3))
           }
       }.onFailure(NotFoundException::class.java)        ← handle 404
        .recoverWithItem(notFound())
```



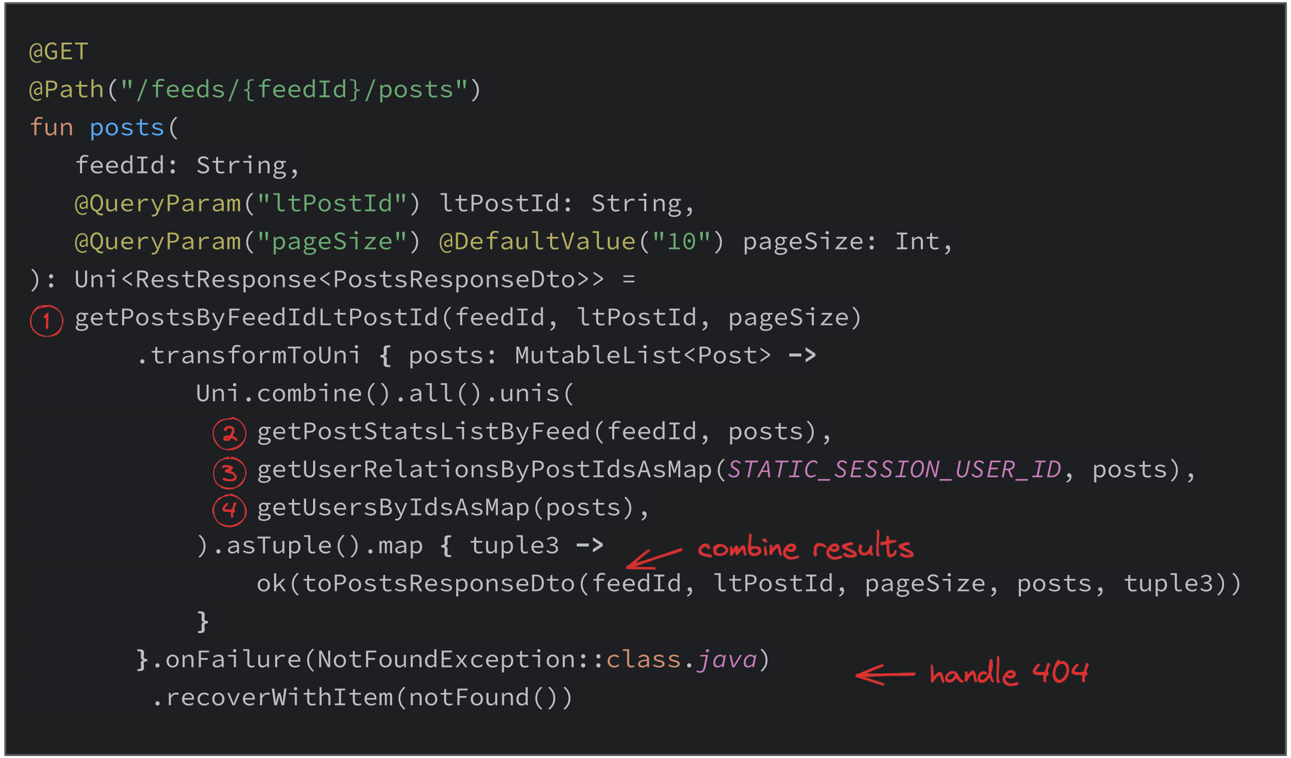
① (1.b.) SELECT post[]

② (2.b.) SELECT post_stats[]

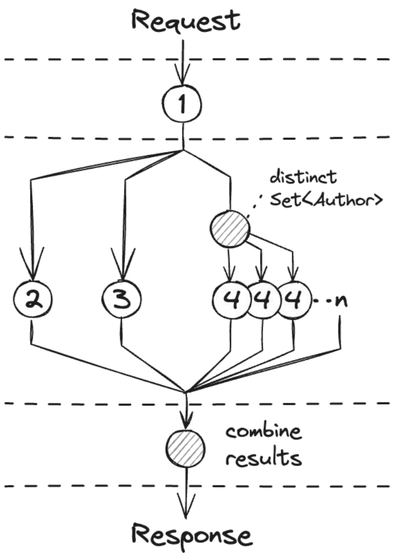
③ (3.e.) SELECT user_x_post_rel[]

④ (4.a.) SELECT user

Code Example



```
@GET
@Path("/feeds/{feedId}/posts")
fun posts(
   feedId: String,
   @QueryParam("ltPostId") ltPostId: String,
   @QueryParam("pageSize") @DefaultValue("10") pageSize: Int,
): Uni<RestResponse<PostsResponseDto>> =
① getPostsByFeedIdLtPostId(feedId, ltPostId, pageSize)
       .transformToUni { posts: MutableList<Post> ->
           Uni.combine().all().unis(
            ② getPostStatsListByFeed(feedId, posts),
            ③ getUserRelationsByPostIdsAsMap(STATIC_SESSION_USER_ID, posts),
            ④ getUsersByIdsAsMap(posts),
           ).asTuple().map { tuple3 ->   ← combine results
               ok(toPostsResponseDto(feedId, ltPostId, pageSize, posts, tuple3))
           }
       }.onFailure(NotFoundException::class.java)   ← handle 404
        .recoverWithItem(notFound())
```



① (1.b.) SELECT post[]

② (2.b.) SELECT post_stats[]

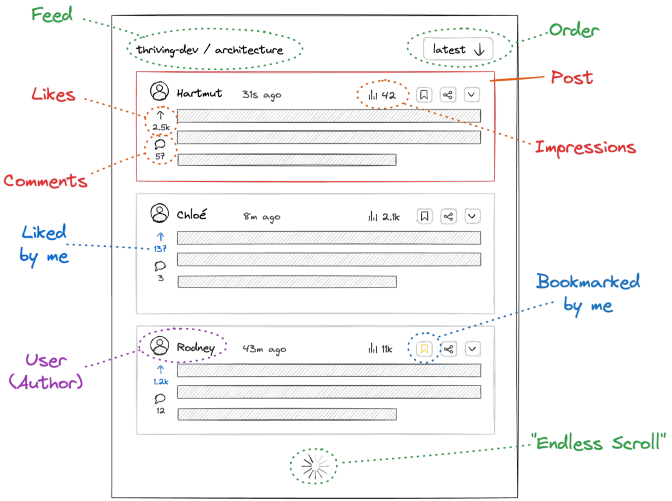
③ (3.e.) SELECT user_x_post_rel[]

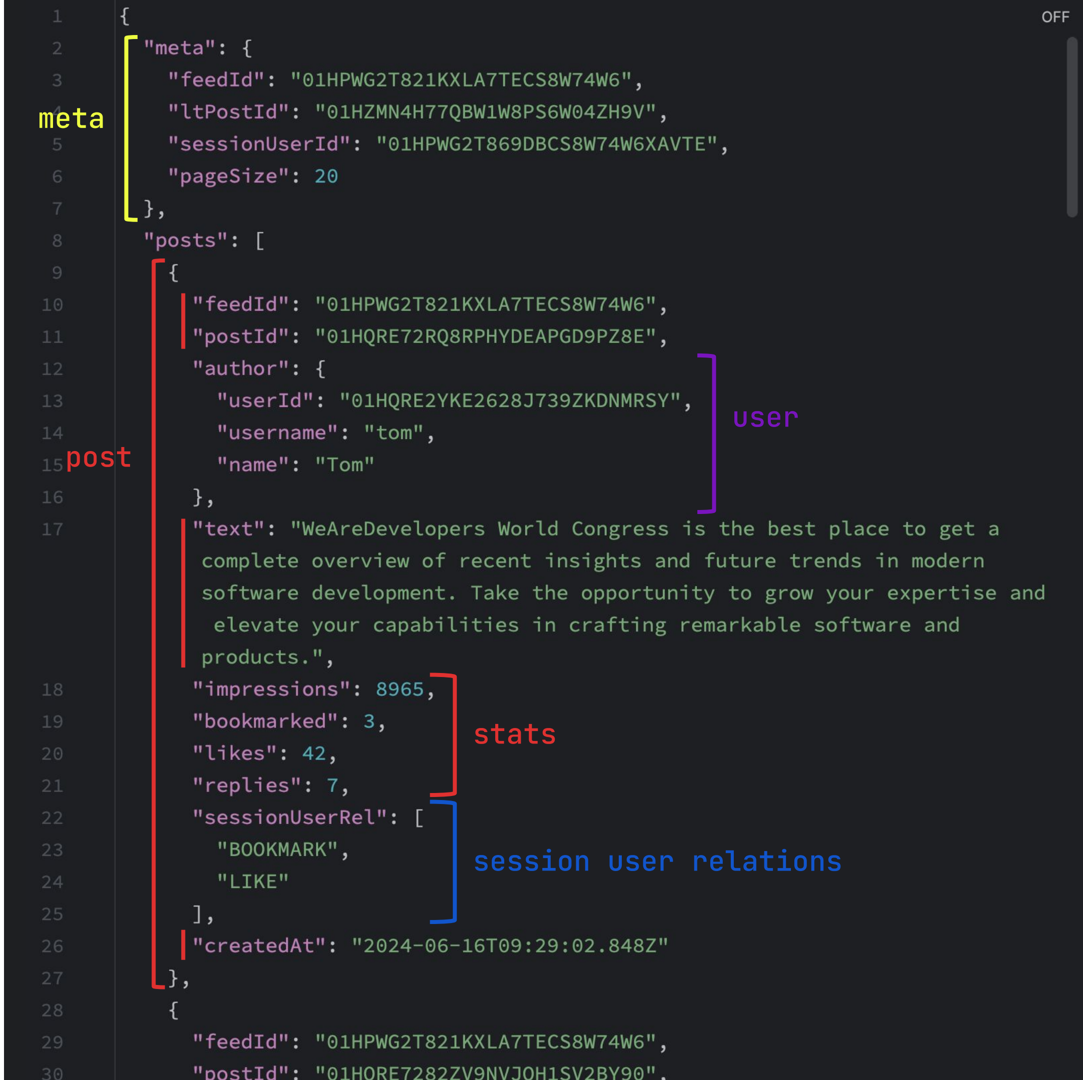
④ (4.a.) SELECT user

Social Platform Feed

RESTful API

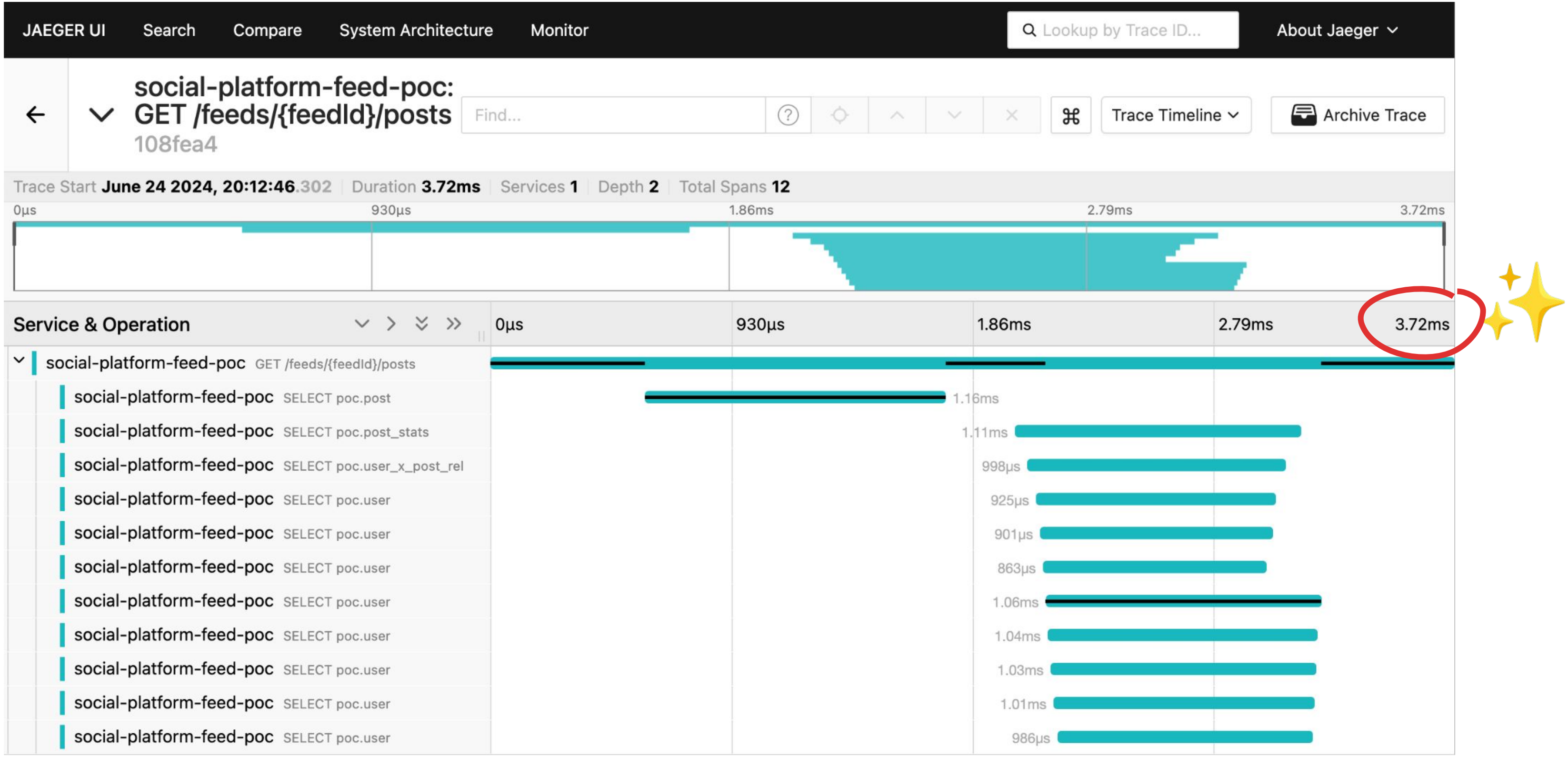
```
GET /feeds/{feedId}/posts
      ?pageSize=20
      &ltPostId=01HZMN4H77QBW1W8PS6W04ZH9V
```





```
{
  "meta": {
    "feedId": "01HPWG2T821KXLA7TECS8W74W6",
    "ltPostId": "01HZMN4H77QBW1W8PS6W04ZH9V",
    "sessionUserId": "01HPWG2T869DBCS8W74W6XAVTE",
    "pageSize": 20
  },
  "posts": [
    {
      "feedId": "01HPWG2T821KXLA7TECS8W74W6",
      "postId": "01HQRE72RQ8RPHYDEAPGD9PZ8E",
      "author": {
        "userId": "01HQRE2YKE2628J739ZKDNMRSY",
        "username": "tom",
        "name": "Tom"
      },
      "text": "WeAreDevelopers World Congress is the best place to get a complete overview of recent insights and future trends in modern software development. Take the opportunity to grow your expertise and elevate your capabilities in crafting remarkable software and products.",
      "impressions": 8965,
      "bookmarked": 3,
      "likes": 42,
      "replies": 7,
      "sessionUserRel": [
        "BOOKMARK",
        "LIKE"
      ],
      "createdAt": "2024-06-16T09:29:02.848Z"
    },
    {
      "feedId": "01HPWG2T821KXLA7TECS8W74W6",
      "postId": "01HQRE7282ZV9NVJQH1SV2BY90",
```

Sample Trace (OpenTelemetry)



social-platform-feed-poc: GET /feeds/{feedId}/posts
108fea4

Trace Start **June 24 2024, 20:12:46**.302 | Duration **3.72ms** | Services **1** | Depth **2** | Total Spans **12**

Service & Operation	0µs · 930µs · 1.86ms · 2.79ms · 3.72ms
social-platform-feed-poc GET /feeds/{feedId}/posts	
social-platform-feed-poc SELECT poc.post	1.16ms
social-platform-feed-poc SELECT poc.post_stats	1.11ms
social-platform-feed-poc SELECT poc.user_x_post_rel	998µs
social-platform-feed-poc SELECT poc.user	925µs
social-platform-feed-poc SELECT poc.user	901µs
social-platform-feed-poc SELECT poc.user	863µs
social-platform-feed-poc SELECT poc.user	1.06ms
social-platform-feed-poc SELECT poc.user	1.04ms
social-platform-feed-poc SELECT poc.user	1.03ms
social-platform-feed-poc SELECT poc.user	1.01ms
social-platform-feed-poc SELECT poc.user	986µs

Conclusions

1 Tailored DB Schema + 2 Optimised Parallel Process Flow + 3 Non-blocking I/O Reactive Code =

Conclusions; Beware of ... ☝️⚠️

- Choosing Cassandra should be a **well-informed decision**!
    - Data must be denormalized & indexed on write
    - Extra complexity of working with eventual consistency
    - Watch out for hot partitions -> *"bucketing"*
    - Migrations are expensive
- Reactive Programming
    - Training and Learning Curve
    - Testing, Debugging and Troubleshooting

Example: Bluesky

v1 Architecture -> Scaling the database layer. *

> ***Postgres was great early on*** *because we didn’t quite know exactly what questions we’d be asking of the data. It let us toss data into the database and figure it out from there.*
>
> ***Now we understand the data*** *and the types of queries we need to run, it frees us up to* ***index it in Scylla in exactly the manner we need*** *and provide APIs for the exact queries we’ll be asking.*

[1]: https://newsletter.pragmaticengineer.com/p/bluesky

Questions?

@hartmut_co_uk

@hartmut-co-uk

@hartmut-co-uk

@thriving_dev

https://thriving.dev