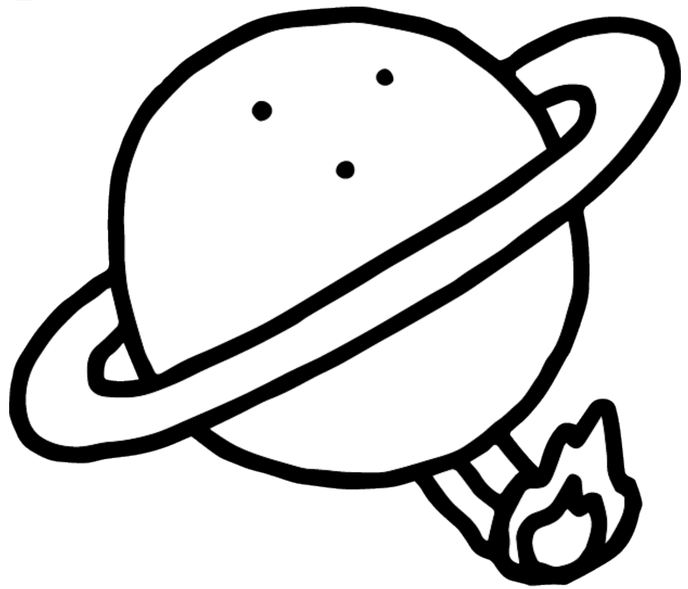
DTO Projection과 복합 인덱스로 제거한 936K Range Scan

200ms -> 10ms 성능 최적화



우아한테크코스 7기 BE 칼리

발표 목차

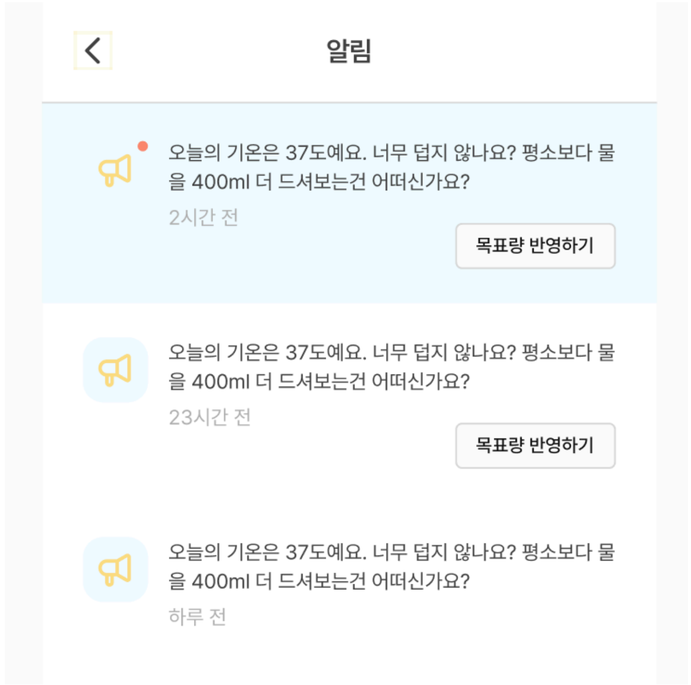
어떤 내용을 다룰까?

1. 컨텍스트 공유
2. 문제점
3. 성능 최적화
4. 궁금증

1. 컨텍스트 공유

💻 1. 컨텍스트 공유

상황



API 개요

- 목적: 알림창 무한스크롤
- 기능: 특정 회원의 최근 7일 알림을 최신순으로 20개씩 조회

=> 조건:

- 조회 대상: 특정 회원
- 시간 범위: 7일 이내
- 정렬: 최신순

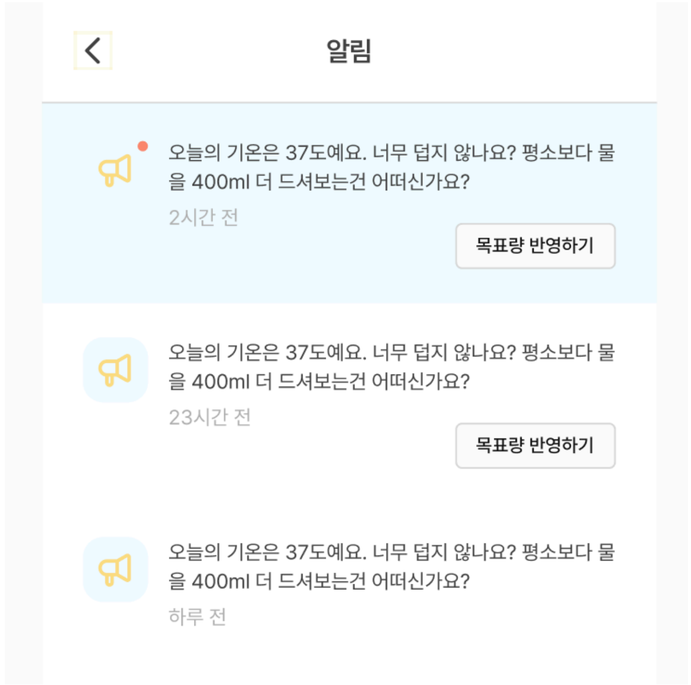
- => 실행 시간 : 201.87ms

```
HTTP
http_req_duration.............: p(90)=201.87ms p(95)=221.81ms p(99)=349.34ms
  { expected_response:true }...: p(90)=201.87ms p(95)=221.81ms p(99)=349.34ms
http_req_failed...............: 0.00% 0 out of 100
http_reqs.....................: 100   5.377451/s
```

K6 성능 측정 결과 (100회 요청, p90·p95·p99 기준)

💻 1. 컨텍스트 공유

상황



API 개요

- 목적: 알림창 무한스크롤
- 기능: 특정 회원의 최근 7일 알림을 최신순으로 20개씩 조회

=> 조건:

- 조회 대상: 특정 회원
- 시간 범위: 7일 이내
- 정렬: 최신순

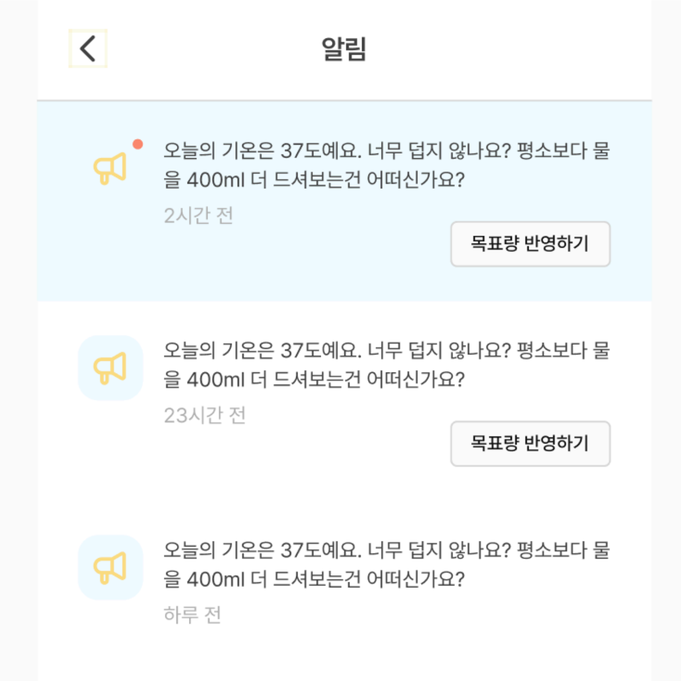
```
HTTP
http_req_duration.............: p(90)=201.87ms p(95)=221.81ms p(99)=349.34ms
  { expected_response:true }...: p(90)=201.87ms p(95)=221.81ms p(99)=349.34ms
http_req_failed...............: 0.00% 0 out of 100
http_reqs.....................: 100   5.377451/s
```

K6 성능 측정 결과 (100회 요청, p90·p95·p99 기준)

- => 실행 시간 : 201.87ms

💻 1. 컨텍스트 공유

상황



API 개요

- 목적: 알림창 무한스크롤
- 기능: 특정 회원의 최근 7일 알림을 최신순으로 20개씩 조회

=> 조건:

- 조회 대상: 특정 회원
- 시간 범위: 7일 이내
- 정렬: 최신순

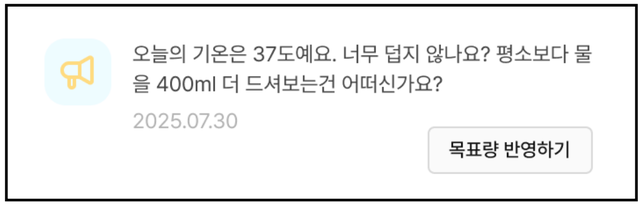
- => 실행 시간 : 201.87ms

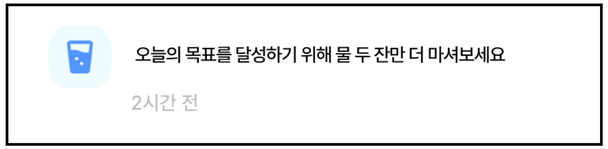
```
HTTP
http_req_duration..............: p(90)=201.87ms p(95)=221.81ms p(99)=349.34ms
  { expected_response:true }...: p(90)=201.87ms p(95)=221.81ms p(99)=349.34ms
http_req_failed................: 0.00% 0 out of 100
http_reqs......................: 100   5.377451/s
```

K6 성능 측정 결과 (100회 요청, p90·p95·p99 기준)

💻 1. 컨텍스트 공유

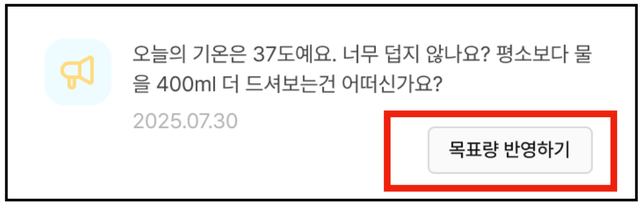
엔티티 구조 설계

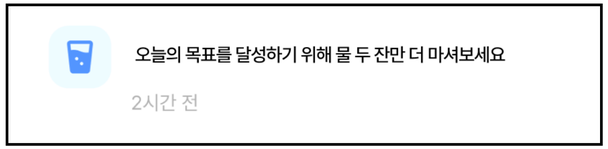




💻 1. 컨텍스트 공유

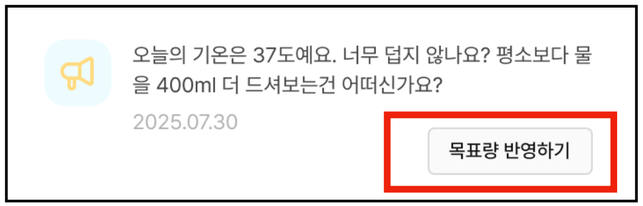
엔티티 구조 설계

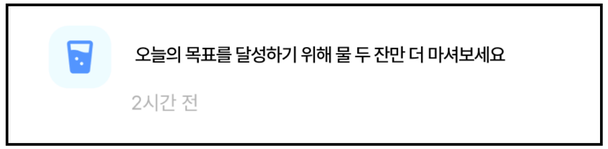


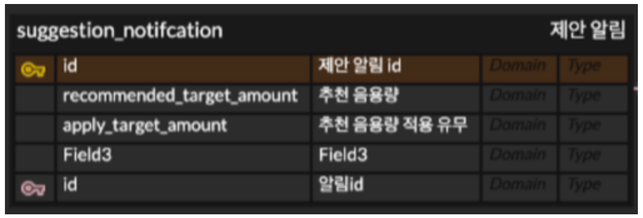


1. 컨텍스트 공유

엔티티 구조 설계

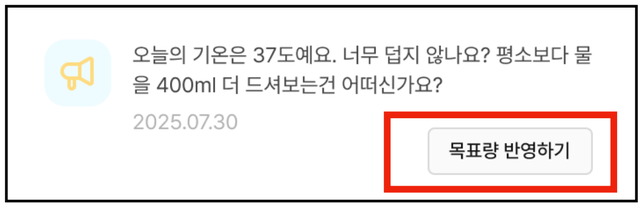


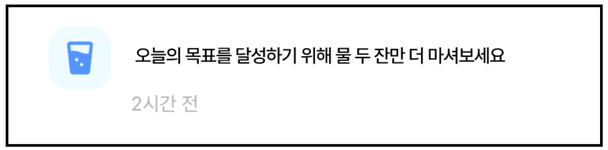




💻 1. 컨텍스트 공유

엔티티 구조 설계





SuggestionNotification (0..1) ---- (1) Notification (N) ---- (1) Member

2. 문제점

💻 2. 문제점

2.1. 조회 책임 분산

```
// NotificationReposiotry
@Query("""
            SELECT n
            FROM Notification n
            LEFT JOIN fetch SuggestionNotification s
            ON n.id = s.id
            WHERE n.id < :lastId
              AND n.createdAt >= :limitStartDateTime
              AND n.member.id = :memberId
            ORDER BY n.id DESC
            """)
```

```
// NotificationService

private NotificationResponse getNotificationResponse(Notification notification) {
        if (notification.getNotificationType() != NotificationType.SUGGESTION) {
            return new GetNotificationResponse(notification);
        }
        SuggestionNotification suggestionNotification = suggestionNotificationRepository.getSuggestionNotificationByNotification(
                notification);
        return new GetSuggestionNotificationResponse(notification, suggestionNotification);
    }
```

- 1차 쿼리: Notification 목록 조회

(LEFT JOIN FETCH SuggestionNotification 시도

- 2차 쿼리: Service 계층에서 SUGGESTION 타입마다

getSuggestionNotificationByNotification() 개별 조회

=> 목록 크기 N에 대해 추가 쿼리 최대 N회 발생 가능

💻 2. 문제점

2.2. 실행 계획

```
+---------+-----------+-------------------------------------+
| rows    | filtered  | Extra                               |
+---------+-----------+-------------------------------------+
| 464117  |    33.33  | Using where; Backward index scan    |
|       1 |   100.00  | NULL                                |
+---------+-----------+-------------------------------------+
```

<EXPLAIN>

```
---------------------------------------------------------------------------------------------------------------------+
| EXPLAIN
+-------------------------------------------------------------------------------------------------------------------------------------------------------------------
| -> Nested loop left join  (cost=594e+6 rows=15469) (actual time=0.37..351 rows=404 loops=1)
    -> Filter: ((n.member_id = 3) and (n.id < 9482291) and (n.created_at >= TIMESTAMP'2025-11-05 00:00:00'))  (cost=594e+6 rows=15469) (actual time=0.281..349 rows=404 loops=1)
        -> Index range scan on n using PRIMARY over (id < 9482291) (reverse)  (cost=594e+6 rows=464117) (actual time=0.273..292 rows=936544 loops=1)
    -> Single-row covering index lookup on s using PRIMARY (id=n.id)  (cost=1.04 rows=1) (actual time=0.00424..0.00424 rows=0 loops=404)
 |
+--------
```

<EXPLAIN ANALYZE>

"464,117개를 읽었는데 33%만 필요"
-> 인덱스 필요성 다분

💻 2. 문제점

2.2. 실행 계획

```
+---------+----------+-------------------------------------+
| rows    | filtered | Extra                               |
+---------+----------+-------------------------------------+
| 464117 |    33.33 | Using where; Backward index scan |
|       1 |   100.00 | NULL                                |
+---------+----------+-------------------------------------+
```

<EXPLAIN>

```
-----------------------------------------------------------------------------------------------------------+
| EXPLAIN
+-----------------------------------------------------------------------------------------------------------------------------------------------
| -> Nested loop left join  (cost=594e+6 rows=15469) (actual time=0.37..351 rows=404 loops=1)
    -> Filter: ((n.member_id = 3) and (n.id < 9482291) and (n.created_at >= TIMESTAMP'2025-11-05 00:00:00'))  (cost=594e+6 rows=15469) (actual time=0.281..349 rows=404 loops=1)
        -> Index range scan on n using PRIMARY over (id < 9482291) (reverse)  (cost=594e+6 rows=464117) (actual time=0.273..292 rows=936544 loops=1)
    -> Single-row covering index lookup on s using PRIMARY (id=n.id)  (cost=1.04 rows=1) (actual time=0.00424..0.00424 rows=0 loops=404)
 |
+--------
```

<EXPLAIN ANALYZE>

"464,117개를 읽었는데 33%만 필요"
-> 인덱스 필요성 다분

💻 2. 문제점

2.2. 실행 계획

```
+---------+----------+------------------------------------+
| rows    | filtered | Extra                              |
+---------+----------+------------------------------------+
| 464117  |    33.33 | Using where; Backward index scan   |
|       1 |   100.00 | NULL                               |
+---------+----------+------------------------------------+
```

<EXPLAIN>

```
--------------------------------------------------------------------------------------------------------------------------------+
| EXPLAIN
+-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
| -> Nested loop left join  (cost=594e+6 rows=15469) (actual time=0.37..351 rows=404 loops=1)
    -> Filter: ((n.member_id = 3) and (n.id < 9482291) and (n.created_at >= TIMESTAMP'2025-11-05 00:00:00'))  (cost=594e+6 rows=15469) (actual time=0.281..349 rows=404 loops=1)
        -> Index range scan on n using PRIMARY over (id < 9482291) (reverse)  (cost=594e+6 rows=464117) (actual time=0.273..292 rows=936544 loops=1)
    -> Single-row covering index lookup on s using PRIMARY (id=n.id)  (cost=1.04 rows=1) (actual time=0.00424..0.00424 rows=0 loops=404)
 |
+--------
```

<EXPLAIN ANALYZE>

"464,117개를 읽었는데 33%만 필요"
-> 인덱스 필요성 다분

💻 2. 문제점

2.3. 요약

- 문제점 1: 애플리케이션 계층의 문제 - 알림 타입에 따라 추가 쿼리 발생
- 문제점 2: DB 계층의 문제 - 불필요한 레코드 스캔

3. 성능 최적화

💻 3. 성능 최적화

3.1. 애플리케이션 계층의 문제

문제점: 알림 타입에 따라 추가 쿼리 발생

AS-IS: 목록 크기 N에 대해 추가 쿼리 최대 N회 발생 가능

TO-BE: 응답에 필요한 필드를 한번의 조회 결과로 구성

💻 3. 성능 최적화

3.1. 애플리케이션 계층의 문제

문제점: 알림 타입에 따라 추가 쿼리 발생

AS-IS: 목록 크기 N에 대해 추가 쿼리 최대 N회 발생 가능

TO-BE: 응답에 필요한 필드를 한번의 조회 결과로 구성

💻 3. 성능 최적화

3.1. 애플리케이션 계층의 문제

문제점: 알림 타입에 따라 추가 쿼리 발생

AS-IS: 목록 크기 N에 대해 추가 쿼리 최대 N회 발생 가능

TO-BE: 응답에 필요한 필드를 한번의 조회 결과로 구성

💻 3. 성능 최적화

3.1. 애플리케이션 계층의 문제

문제점: 알림 타입에 따라 추가 쿼리 발생

AS-IS: 목록 크기 N에 대해 추가 쿼리 최대 N회 발생 가능

TO-BE: 응답에 필요한 필드를 한번의 조회 결과로 구성

```
@Query(""" 2 usages  2jin2.1031 +2
    SELECT new backend.mulkkam.notification.dto.ReadNotificationRow(
        n.id,
        n.createdAt,
        n.content,
        n.notificationType,
        n.isRead,
        s.recommendedTargetAmount,
        s.applyTargetAmount
    )
    FROM Notification n
    LEFT JOIN SuggestionNotification s ON s.id = n.id
    WHERE n.createdAt >= :limitStartDateTime
      AND n.member.id = :memberId
    ORDER BY n.id DESC
""")
List<ReadNotificationRow> findLatestRows(
        Long memberId,
        LocalDateTime limitStartDateTime,
        Pageable pageable
);
```

💻 3. 성능 최적화

3.1. 애플리케이션 계층의 문제

Projection:
select 절에 대상을 지정하여 원하는 값만 뽑아오는 것

문제점: 알림 타입에 따라 추가 쿼리 발생

AS-IS: 목록 크기 N에 대해 추가 쿼리 최대 N회 발생 가능

TO-BE: 응답에 필요한 필드를 한번의 조회 결과로 구성

```
@Query("""  2 usages  2jin2.1031 +2
    SELECT new backend.mulkkam.notification.dto.ReadNotificationRow(
        n.id,
        n.createdAt,
        n.content,
        n.notificationType,
        n.isRead,
        s.recommendedTargetAmount,
        s.applyTargetAmount
    )
    FROM Notification n
    LEFT JOIN SuggestionNotification s ON s.id = n.id
    WHERE n.createdAt >= :limitStartDateTime
      AND n.member.id = :memberId
    ORDER BY n.id DESC
""")
List<ReadNotificationRow> findLatestRows(
        Long memberId,
        LocalDateTime limitStartDateTime,
        Pageable pageable
);
```

💻 3. 성능 최적화

3.1. 애플리케이션 계층의 문제

Projection:
select 절에 대상을 지정하여 원하는 값만 뽑아오는 것

문제점: 알림 타입에 따라 추가 쿼리 발생

AS-IS: 목록 크기 N에 대해 추가 쿼리 최대 N회 발생 가능

TO-BE: 응답에 필요한 필드를 한번의 조회 결과로 구성

=> 알림 타입에 따라 추가 쿼리 발생 X, 한번의 조회

```
@Query("""  2 usages  2jin2.1031 +2
    SELECT new backend.mulkkam.notification.dto.ReadNotificationRow(
        n.id,
        n.createdAt,
        n.content,
        n.notificationType,
        n.isRead,
        s.recommendedTargetAmount,
        s.applyTargetAmount
    )
    FROM Notification n
    LEFT JOIN SuggestionNotification s ON s.id = n.id
    WHERE n.createdAt >= :limitStartDateTime
      AND n.member.id = :memberId
    ORDER BY n.id DESC
""")
List<ReadNotificationRow> findLatestRows(
        Long memberId,
        LocalDateTime limitStartDateTime,
        Pageable pageable
);
```

💻 3. 성능 최적화

3.2. DB 계층의 문제

문제점: 불필요한 레코드 스캔

AS-IS: 인덱스 X

TO-BE: 복합 인덱스

💻 3. 성능 최적화

3.2. DB 계층의 문제

문제점: 불필요한 레코드 스캔

AS-IS: 인덱스 X

TO-BE: 복합 인덱스

💻 3. 성능 최적화

3.2. DB 계층의 문제 (AS-IS)

```
mysql> SHOW INDEX FROM notification;
+--------------+------------+-------------------------+
| Table        | Non_unique | Key_name                |
+--------------+------------+-------------------------+
| notification |          0 | PRIMARY                 |
| notification |          1 | fk_notification_member |
+--------------+------------+-------------------------+
```

- 더미 데이터 조건
  - 1 <= id <= 2000
  - 1 <= member_id <= 20 (한 사람 당, 100개)

PK 기반 B+Tree 구조

FK 기반 B+Tree 구조

💻 3. 성능 최적화

3.2. DB 계층의 문제 (AS-IS)

```
mysql> SHOW INDEX FROM notification;
+--------------+------------+------------------------+
| Table        | Non_unique | Key_name               |
+--------------+------------+------------------------+
| notification |          0 | PRIMARY                |
| notification |          1 | fk_notification_member |
+--------------+------------+------------------------+
```

- 더미 데이터 조건
  - 1 <= id <= 2000
  - 1 <= member_id <= 20 (한 사람 당, 100개)

PK 기반 B+Tree 구조

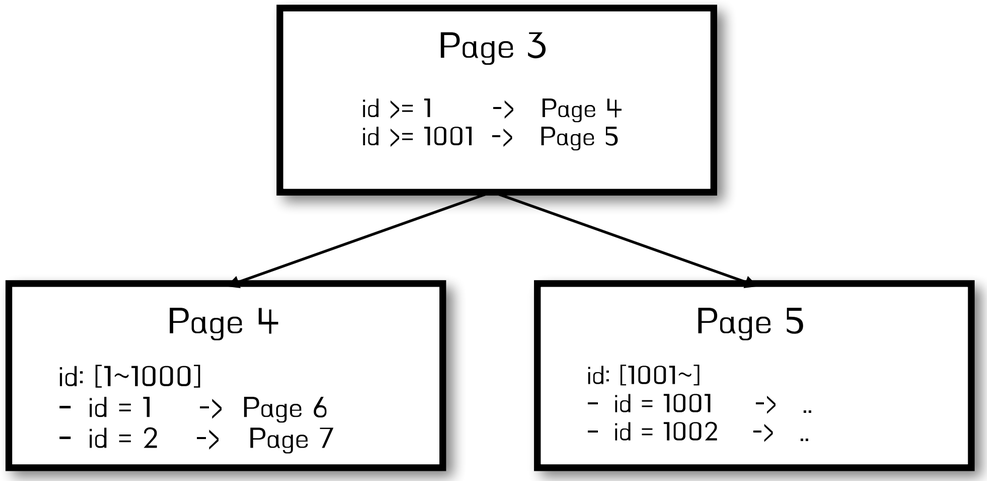
FK 기반 B+Tree 구조

💻 3. 성능 최적화

3.2. DB 계층의 문제 (AS-IS)

```
mysql> SHOW INDEX FROM notification;
+--------------+------------+-------------------------+
| Table        | Non_unique | Key_name                |
+--------------+------------+-------------------------+
| notification |          0 | PRIMARY                 |
| notification |          1 | fk_notification_member  |
+--------------+------------+-------------------------+
```

- 더미 데이터 조건
  - 1 <= id <= 2000
  - 1 <= member_id <= 20 (한 사람 당, 100개)



PK 기반 B+Tree 구조

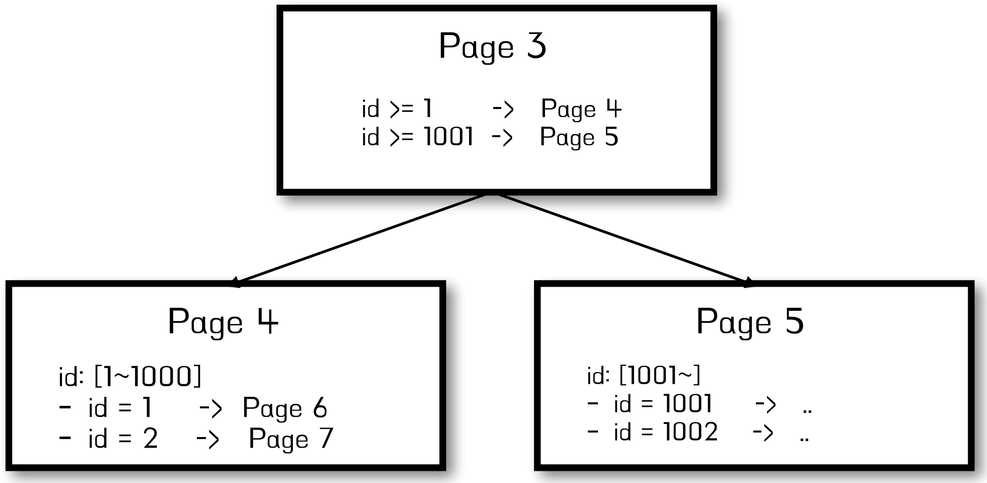
FK 기반 B+Tree 구조

💻 3. 성능 최적화

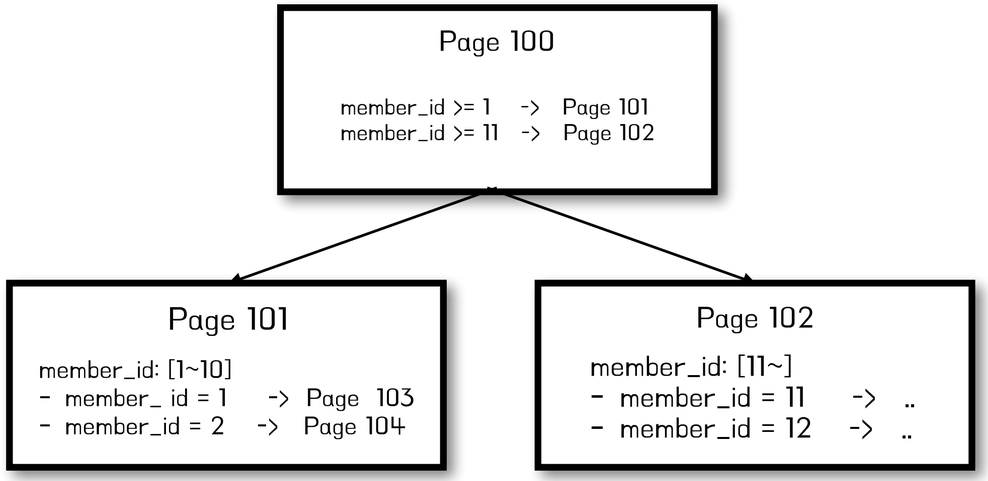
3.2. DB 계층의 문제 (AS-IS)

```
mysql> SHOW INDEX FROM notification;
+--------------+------------+------------------------+
| Table        | Non_unique | Key_name               |
+--------------+------------+------------------------+
| notification |          0 | PRIMARY                |
| notification |          1 | fk_notification_member |
+--------------+------------+------------------------+
```

- 더미 데이터 조건
  - 1 <= id <= 2000
  - 1 <= member_id <= 20 (한 사람 당, 100개)



PK 기반 B+Tree 구조



FK 기반 B+Tree 구조

💻 3. 성능 최적화

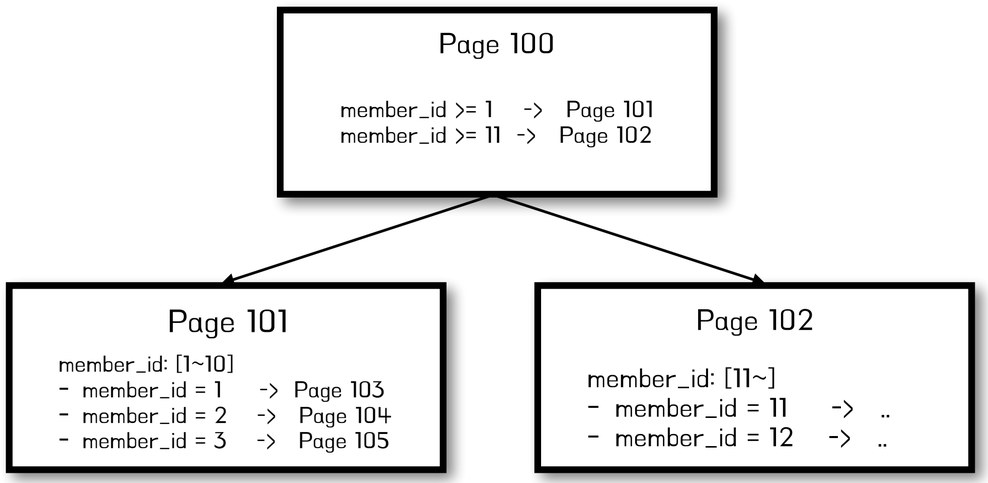
3.2. DB 계층의 문제 (AS-IS)

- 조회 데이터 조건
  - WHERE n.member_id = 3 AND n.created_at >= '2025-11-05'

💻 3. 성능 최적화

3.2. DB 계층의 문제 (AS-IS)

- 조회 데이터 조건
  - WHERE n.member_id = 3 AND n.created_at >= '2025-11-05'

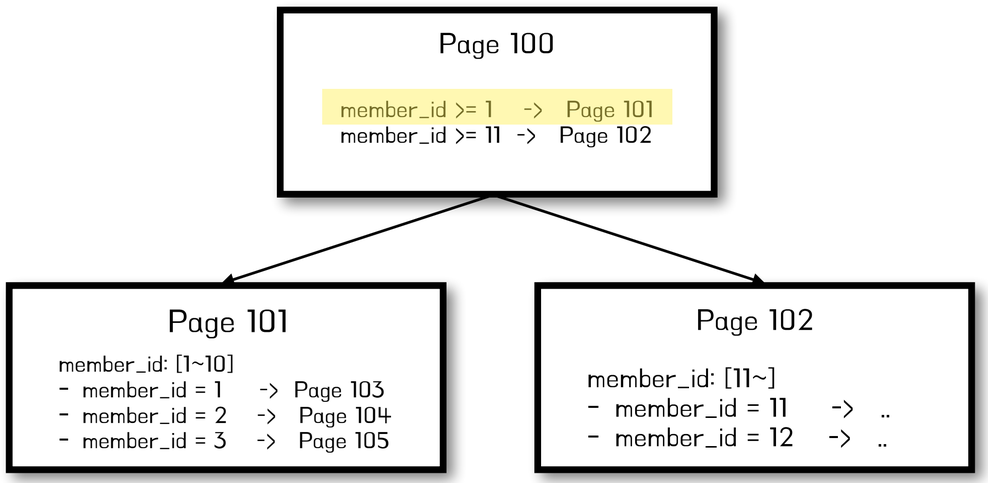


FK 기반 B+Tree 구조

💻 3. 성능 최적화

3.2. DB 계층의 문제 (AS-IS)

- 조회 데이터 조건
  - WHERE n.member_id = 3 AND n.created_at >= '2025-11-05'

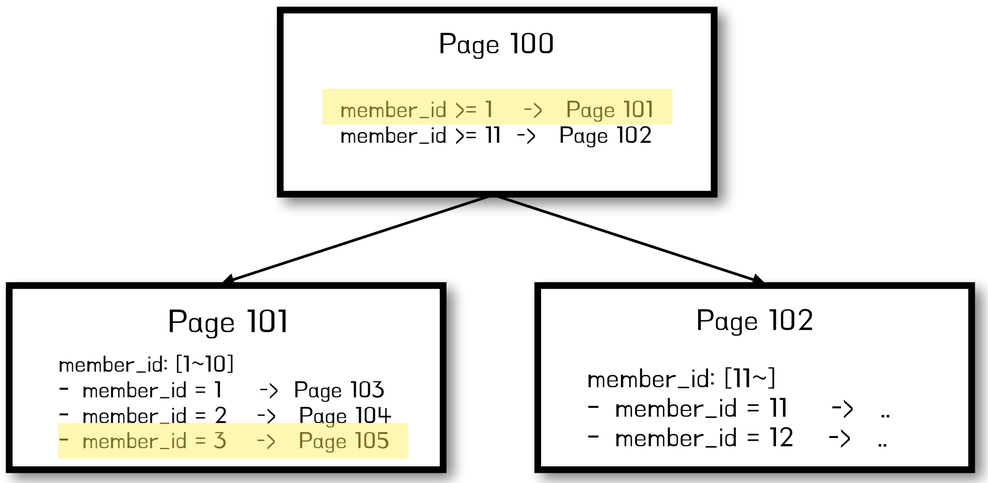


FK 기반 B+Tree 구조

💻 3. 성능 최적화

3.2. DB 계층의 문제 (AS-IS)

- 조회 데이터 조건
  - WHERE n.member_id = 3 AND n.created_at >= '2025-11-05'

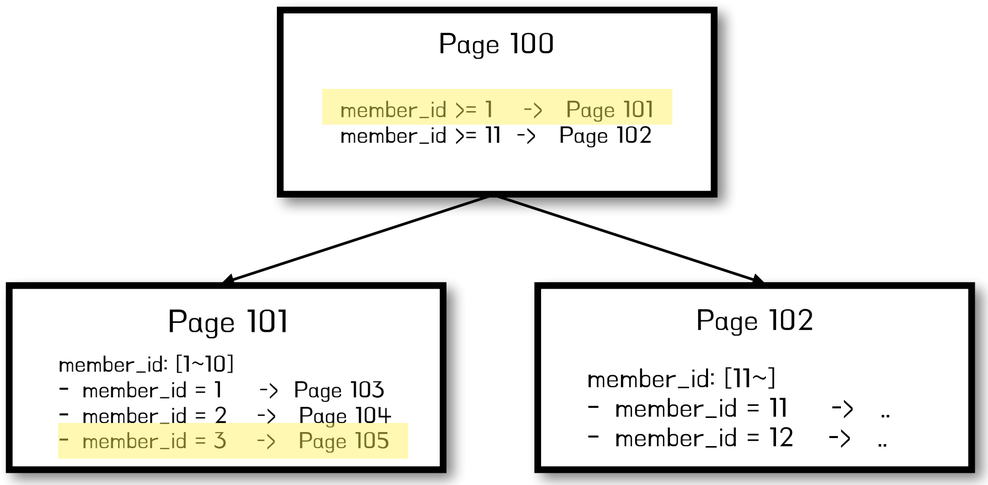


FK 기반 B+Tree 구조

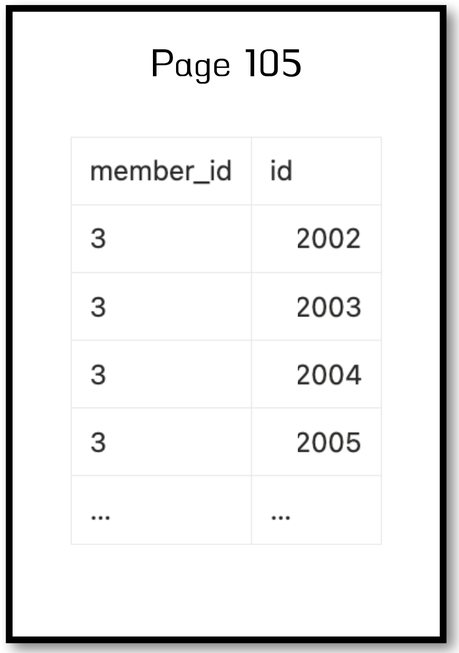
💻 3. 성능 최적화

3.2. DB 계층의 문제 (AS-IS)

- 조회 데이터 조건
  - WHERE n.member_id = 3 AND n.created_at >= '2025-11-05'



FK 기반 B+Tree 구조



💻 3. 해결

3.3. DB 계층의 문제 (TO-BE)

- 해결 방식 (인덱스 컬럼 선택의 기준)
  - where 절 모든 컬럼 + select 절 모든 컬럼
  - where 절 컬럼들로 구성 : id, member_id, created_at
    - id (기본 PK) , member_id (기본 FK)
    - id, member_id, created_at (커버링 인덱스)

💻 3. 해결

3.3. DB 계층의 문제 (TO-BE)

- 해결 방식 (인덱스 컬럼 선택의 기준)
  - ~~where 절 모든 컬럼 + select 절 모든 컬럼~~
  - where 절 컬럼들로 구성 : id, member_id, created_at
    - id (기본 PK) , member_id (기본 FK)
    - id, member_id, created_at (커버링 인덱스)

💻 3. 해결

3.3. DB 계층의 문제 (TO-BE)

- 해결 방식 (인덱스 컬럼 선택의 기준)
  - ~~where 절 모든 컬럼 + select 절 모든 컬럼~~
  - where 절 컬럼들로 구성 : id, member_id, created_at
    - id (기본 PK) , member_id (기본 FK)
    - id, member_id, created_at (커버링 인덱스)

💻 3. 해결

3.3. DB 계층의 문제 (TO-BE)

- 해결 방식 (인덱스 컬럼 선택의 기준)
  - ~~where 절 모든 컬럼 + select 절 모든 컬럼~~
  - where 절 컬럼들로 구성 : id, member_id, created_at
    - id (기본 PK) , member_id (기본 FK)
    - id, member_id, created_at (복합 인덱스)

💻 3. 해결

3.3. DB 계층의 문제 (TO-BE)

- 해결 방식 (인덱스 컬럼 선택의 기준)
  - ~~where 절 모든 컬럼 + select 절 모든 컬럼~~
  - where 절 컬럼들로 구성 : member_id, created_at
    - ~~id (기본 PK) , member_id (기본 FK)~~
    - member_id, created_at (복합 인덱스)

💻 3. 해결

3.3. DB 계층의 문제 (TO-BE)

- 해결 방식 (인덱스 컬럼 선택의 기준)
  - ~~where 절 모든 컬럼 + select 절 모든 컬럼~~
  - where 절 컬럼들로 구성 : member_id, created_at
    - ~~id (기본 PK) , member_id (기본 FK)~~
    - member_id, created_at (복합 인덱스)

💻 3. 해결

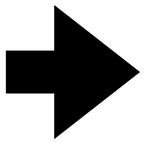
3.3. DB 계층의 문제 (TO-BE)

a) 최적화 전 코드 (: 95d8e5f3)	unread_a_1	221.81ms
b) 최적화 후 코드 (: feat 862 (= main HEAD))	unread_b_1	182.11ms
c) b + 인덱스1 (member_id, is_read, created_at)	unread_b_1	10.88ms
d) b + 인덱스2 (member_id, created_at, id DESC)	unread_b_1	10.35ms
e) b + 인덱스 3 (member_id, created_at DESC) 열기	unread_b_1	
f) b + 인덱스 4 (member_id, created_at DESC, id DESC)	unread_b_1 / unread_c	
g) 인덱스 5 (member_id, created_at, id)	unread_c	

+ 새 페이지

💻 3. 해결
3.3. DB 계층의 문제 (TO-BE)

```
+--------+----------+-----------------------------------+
| rows   | filtered | Extra                             |
+--------+----------+-----------------------------------+
| 464117 |    33.33 | Using where; Backward index scan |
|      1 |   100.00 | NULL                              |
+--------+----------+-----------------------------------+
```



SQL

```
--+------+----------+-------------------------------------------+
  | rows | filtered | Extra                                     |
--+------+----------+-------------------------------------------+
  |  500 |   100.00 | Using index condition; Backward index scan |
  |    1 |   100.00 | NULL                                      |
--+------+----------+-------------------------------------------+
```

💻 3. 해결
3.3. DB 계층의 문제 (TO-BE)

```
@Query("""  2 usages   2jin2.1031 +2
    SELECT new backend.mulkkam.notification.dto.ReadNotificationRow(
        n.id,
        n.createdAt,
        n.content,
        n.notificationType,
        n.isRead,
        s.recommendedTargetAmount,
        s.applyTargetAmount
    )
    FROM Notification n
    LEFT JOIN SuggestionNotification s ON s.id = n.id
    WHERE n.createdAt >= :limitStartDateTime
      AND n.member.id = :memberId
    ORDER BY n.createdAt DESC, n.id DESC
""")
```

💻 3. 해결
3.3. DB 계층의 문제 (TO-BE)

```
@Query("""  2 usages   2jin2.1031 +2
    SELECT new backend.mulkkam.notification.dto.ReadNotificationRow(
        n.id,
        n.createdAt,
        n.content,
        n.notificationType,
        n.isRead,
        s.recommendedTargetAmount,
        s.applyTargetAmount
    )
    FROM Notification n
    LEFT JOIN SuggestionNotification s ON s.id = n.id
    WHERE n.createdAt >= :limitStartDateTime
      AND n.member.id = :memberId
    ORDER BY n.createdAt DESC, n.id DESC
""")
```

💻 3. 해결
3.3. DB 계층의 문제 (TO-BE)

```
@Query("""  2 usages   2jin2.1031 +2
    SELECT new backend.mulkkam.notification.dto.ReadNotificationRow(
        n.id,
        n.createdAt,
        n.content,
        n.notificationType,
        n.isRead,
        s.recommendedTargetAmount,
        s.applyTargetAmount
    )
    FROM Notification n
    LEFT JOIN SuggestionNotification s ON s.id = n.id
    WHERE n.createdAt >= :limitStartDateTime
      AND n.member.id = :memberId
    ORDER BY n.createdAt DESC, n.id DESC
""")
```

```
CREATE INDEX member_id_created_at
ON notification(member_id, created_at);
```

4. 궁금증

💻 4. 궁금증

Q) B+Tree 구조에서 보조 인덱스의 마지막 컬럼에는
PK가 값으로 들어가는가, 포인터로 들어가는가?

Q) 이미 정렬된 데이터를 ORDER BY로 다시 정렬하는가?

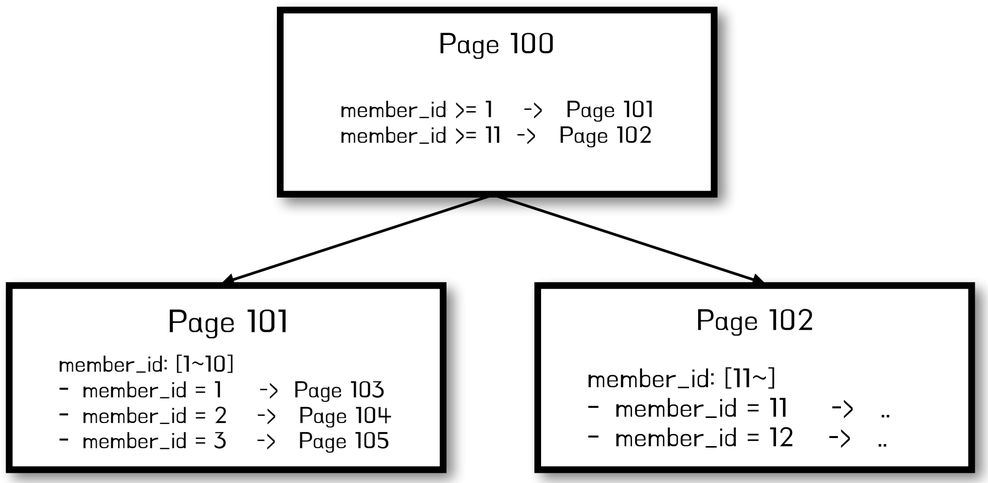
Q) 인덱스에 정렬 방식 중, ASC와 DESC 중 어떤 게 성능이 더 좋을까?

Q) (a, b) 인덱스와 (a, b, id) 인덱스는 저장 구조가 동일할까?

💻 3. 해결

3.3. DB 계층의 문제 (TO-BE)

- WHERE n.member_id = 3 AND n.created_at >= '2025-11-05'



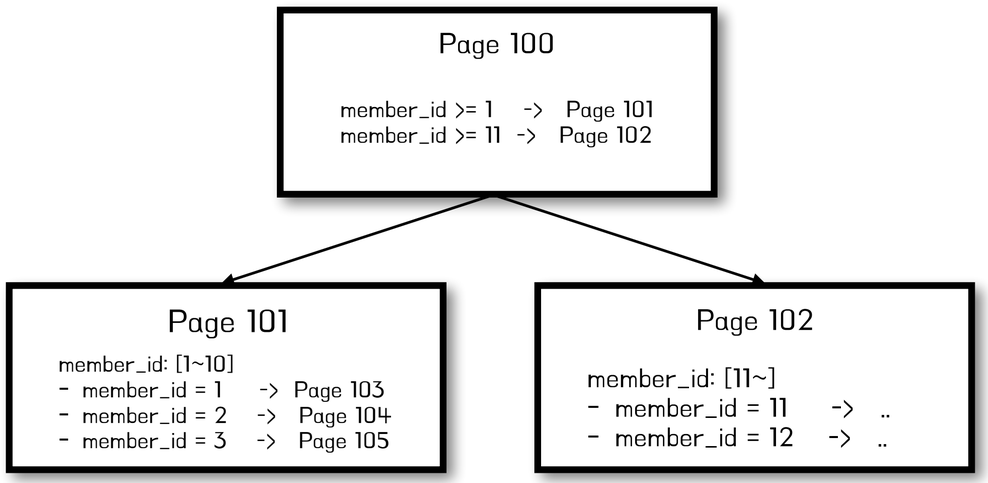
FK 기반 B+Tree 구조

Page 104

member_id	created_at	id
3	2025-08-02	@2002
3	2025-08-26	@2003
3	2025-09-01	@2004
3	2025-09-09	@2005
...	...	...

💻 3. 해결
3.3. DB 계층의 문제 (TO-BE)

- WHERE n.member_id = 3 AND n.created_at >= '2025-11-05'



FK 기반 B+Tree 구조

Page 104

member_id	created_at	id
3	2025-08-02	@2002
3	2025-08-26	@2003
3	2025-09-01	@2004
3	2025-09-09	@2005
...	...	...