

Напредни бази на податоци  
Фаза 4- Индекси и оптимизација на прашалници  
Проект: ArenaNet

Димитар Димов 231099  
Митко Цуклев 231236  
Теа Димитриевска 231058

**View 1: friend\_list**

1. Погледот служи за приказ на листата со пријатели во реално време. Главниот филтер ќе биде според id на најавениот корисник за да се извлечат сите негови прифатени пријателства, а дополнително може да се филтрира по username на друг корисник односно пријател за брзо пребарување низ листата.
- 2.

```
EXPLAIN ANALYZE  
SELECT *  
FROM friend_list  
WHERE user_id = 10000;
```

Timing: Generation 6.979 ms (Deform 3.382 ms), Inlining 0.000 ms, Optimization 2.988 ms, Emission 52.259 ms, Total 62.225 ms  
Execution Time: 972.765 ms

3. Иницијалното време за извршување на погледот е 972ms. Ова може да се подобри со индекси
4. Најбавните операции се Parallel scan seq на friendships табелата

```
-> Parallel Seq Scan on friendships f (cost=0.00..82796.74 rows=1 width=28) (actual time=252.664..289.126 rows=0 loops=3)
```

```
-> Parallel Seq Scan on friendships f 1 (cost=0.00..82796.74 rows=1 width=28) (actual time=448.424..451.425 rows=0 loops=2)
```

Најбавната операција е Parallel Seq Scan врз табелата friendships, бидејќи базата скенира милиони редови и ги филтрира според user\_id\_high/user\_id\_low и status = 'accepted'. Бидејќи резултатот враќа само 2 реда, ова покажува дека е потребен индекс за побрзо пронаоѓање на прифатените пријателства за конкретен корисник.

5. Времето на извршување на операциите insert и update на табелата friendships изнесува:

```

EXPLAIN ANALYZE
INSERT INTO friendships (
  id,
  user_id_low,
  user_id_high,
  requested_by_user_id,
  status
)
VALUES (
  999999999,
  10000,
  10001,
  10000,
  'accepted'
);

```

Execution Time: 0.738 ms

```

EXPLAIN ANALYZE
UPDATE friendships
SET status = 'accepted'
WHERE id = 10000;

```

Execution Time: 343.436 ms

6. Индексите се креираат врз колоните status и user\_id\_high/user\_id\_low бидејќи погледот friend\_list секогаш ги прикажува само прифатените пријателства, а главниот филтер е според id на најавениот корисник. Бидејќи корисникот може да се појави во двете колони user\_id\_high и user\_id\_low, потребни се два индекси. Со овие индекси PostgreSQL може директно да ги пронајде прифатените пријателства за конкретен корисник, наместо да прави целосно скенирање на табелата friendships.

```

CREATE INDEX idx_friendships_status_high
ON friendships(status, user_id_high);

CREATE INDEX idx_friendships_status_low
ON friendships(status, user_id_low);

ANALYZE friendships;

```

7. Со индекс:  
a. Select

```

EXPLAIN ANALYZE
SELECT *
FROM friend_list
WHERE user_id = 10000;

```

Execution Time: 0.314 ms

- b. Insert

```

INSERT INTO friendships (
  id,
  user_id_low,
  user_id_high,
  requested_by_user_id,
  status
)
VALUES (
  999999999,
  10000,
  10001,
  10000,
  'accepted'
);

```

Execution Time: 598.223 ms

### c. Update

```

EXPLAIN ANALYZE
UPDATE friendships
SET status = 'accepted'
WHERE id = 10000;

```

Execution Time: 188.505 ms

## View 2: Game catalog

1. Погледот `game_catalog` служи за приказ на каталогот на игри, заедно со нивните жанрови, девелопери, просечна оцена и број на рецензии. Во основното користење овој поглед не мора да има `WHERE` услов, бидејќи целта е да се прикажат сите игри во каталогот.

2. Иницијалното време за извршување на операцијата `select` изнесува:

```

explain analyze
select * from game_catalog gc

```

Execution Time: 0.740 ms

3. Иницијалното време на извршување на погледот изнесува `0.740 ms`, што е многу брзо и целосно прифатливо за апликацијата. Поради тоа нема потреба од креирање дополнителен индекс. Додавањето индекс во овој случај не би донело значајно подобрување, а може непотребно да ги забави `INSERT` и `UPDATE` операциите во основните табели, бидејќи индексите треба дополнително да се одржуваат при секоја промена на податоците.

## View 3: leaderboard

1. Погледот се користи за глобално рангирање на играчите. Примарно ќе се филтрира по највисок ранг и држава, овозможувајќи им на корисниците да видат кои се најдобрите играчи во светот или во нивниот регион.

```

explain analyze
select * from leaderboard l where l.country = 'Germany'

```

Planning Time: 0.022 ms

Execution Time: 2309.728 ms

2. Иницијалното време на извршување е `2309ms`, што не е прифатливо за апликацијата. Ова може да се подобри со индекси. `Insert` и `Update` операциите покажуваат добри перформанси

```

EXPLAIN ANALYZE
UPDATE profiles
SET rank_points = rank_points + 100
WHERE user_id = 9793;

```

```

•INSERT INTO users (
  id,
  username,
  email,
  password_hash,
  location_id
)
VALUES (
  999999999,
  'test_user_999999999',
  'test999999999@example.com',
  'hash',
  (SELECT id FROM locations LIMIT 1)
);

•EXPLAIN (ANALYZE, BUFFERS)
INSERT INTO profiles (
  user_id,
  avatar_url,
  bio,
  rank_points
)
VALUES (
  999999999,
  'https://example.com/avatar.png',
  'Test profile for leaderboard insert performance',
  2500
);

```

#### View 4: Unread notifications

1. Погледот `unread_notifications` служи за приказ на непочитани известувања на корисникот во реално време. Прикажува известувања заедно со корисничкото име, типот на известувањето, пораката и поврзаната игра од преференциите на корисникот.

```

EXPLAIN ANALYZE
SELECT * FROM unread_notifications
WHERE user_id = 1;

```

17	Planning Time: 1230.183 ms
18	Execution Time: 11665.446 ms

2. Најбавната операција е **Seq Scan** врз табелата `notifications`. Базата скенира сите **100,000 редови** и ги филтрира според `user_id = 1` и `is_read = false`, а враќа само **4 реда**. Ова покажува дека е потребен индекс.

```

Seq Scan on notifications n (cost=0.00..3043.00 rows=4 width=83) (actual time=2328.115..11306.573 rows=4 loc
Filter: ((NOT is_read) AND (user_id = 1))
Rows Removed by Filter: 99996

```

3. Иницијалното време на извршување на погледот изнесува **11,665ms** (Execution Time), со Planning Time од **1,230ms**. Ова не е прифатливо време за апликацијата па затоа пристапуваме кон индексирање.

```

CREATE INDEX idx_notifications_user_unread
ON notifications(user_id, is_read);

```

4. Со додавање на индексот `idx_notifications_user_unread` врз колоните (`user_id`, `is_read`), времето на извршување на `SELECT` се намали од **11,665ms** на **0.129**. PostgreSQL сега користи Bitmap Index Scan и Bitmap Heap Scan наместо Seq Scan,

директно пронаоѓајќи ги само потребните редови. Времето на INSERT и UPDATE исто така се подобри и останува прифатливо.

```
Sort Key: n.id DESC
Sort Method: quicksort Memory: 25kB
-> Nested Loop Left Join (cost=5.47..46.88 rows=4 width=109) (actual time=0.053..0.072 rows=4 loops=1)
  -> Nested Loop Left Join (cost=5.32..46.20 rows=4 width=97) (actual time=0.048..0.063 rows=4 loops=1)
    -> Nested Loop (cost=4.89..28.43 rows=4 width=105) (actual time=0.038..0.045 rows=4 loops=1)
      -> Index Scan using users_pkey on users u (cost=0.43..8.45 rows=1 width=30) (actual time=0.012..0.013 rows=1 loops=1)
        Index Cond: (id = 1)
      -> Bitmap Heap Scan on notifications n (cost=4.46..19.94 rows=4 width=83) (actual time=0.023..0.027 rows=4 loops=1)
        Recheck Cond: ((user_id = 1) AND (NOT is_read))
        Heap Blocks: exact=6
        -> Bitmap Index Scan on idx_notifications_user_unread (cost=0.00..4.46 rows=4 width=0) (actual time=0.008..0.009 rows=4 loops=1)
          Index Cond: ((user_id = 1) AND (is_read = false))
    -> Index Only Scan using preferences_pkey on preferences pr (cost=0.42..4.44 rows=1 width=16) (actual time=0.003..0.003 rows=1 loops=1)
      Index Cond: ((user_id = n.preference_user_id) AND (game_id = n.preference_game_id))
      Heap Fetches: 0
  -> Index Scan using games_pkey on games g (cost=0.15..0.17 rows=1 width=28) (actual time=0.002..0.002 rows=1 loops=4)
    Index Cond: (id = pr.game_id)
Planning Time: 0.538 ms
Execution Time: 0.129 ms
```

```
QUERY PLAN
-----+-----+
Insert on notifications (cost=0.00..0.01 rows=0 width=0) (actual time=2366.276..2366.277 rows=0 loops=1) |
-> Result (cost=0.00..0.01 rows=1 width=335) (actual time=0.006..0.007 rows=1 loops=1) |
Planning Time: 0.051 ms |
Trigger for constraint notifications_user_id_fkey: time=0.085 calls=1 |
Trigger for constraint notifications_preference_user_id_preference_game_id_fkey: time=0.056 calls=1 |
Execution Time: 2366.447 ms |
```

```
QUERY PLAN
-----+-----+
Update on notifications (cost=0.29..8.31 rows=0 width=0) (actual time=305.563..305.563 rows=0 loops=1) |
-> Index Scan using notifications_pkey on notifications (cost=0.29..8.31 rows=1 width=7) (actual time=0.025..0.028 rows=1 loops=1) |
  Index Cond: (id = 1000007) |
Planning Time: 0.100 ms |
Execution Time: 305.599 ms |
```

## View 5: user\_all\_achievements

1. Погледот `user_all_achievements` служи за приказ на сите отклучени достигнувања на корисниците. Прикажува информации за корисникот, насловот и описот на достигнувањето, играта на која припаѓа достигнувањето и времето кога било отклучено. Овој поглед е корисен за преглед на напредокот на корисниците и нивните постигнувања во различни игри.

```
explain analyze select * from user_all_achievements uaa where uaa.user_id =23233
```

```
Execution Time: 37.452 ms
```

```
INSERT INTO user_achievements (id, user_id, achievement_id, unlocked_at)
VALUES (
  1000000001,
  (SELECT id FROM users LIMIT 1),
  (SELECT id FROM achievements LIMIT 1),
  CURRENT_TIMESTAMP
);
```

```
Execution Time: 35.579 ms
```

```
EXPLAIN ANALYZE
UPDATE profiles
SET rank_points = rank_points + 100
WHERE user_id = 9793;
```

```
Execution Time: 0.151 ms
```

Времето на извршување на сите операции е прифатливо. Нема потреба од преуредување на прашалникот или креирање на индекс.

## View 6: tournament\_standings

1. Погледот `tournament_standings` служи за приказ на тимовите кои учествуваат во турнирите и нивниот напредок. Прикажува информации за турнирот, играта, датумот на започнување, наградниот фонд, тимот и бројот на одиграни/внесени натпревари. Овој поглед е корисен за следење на состојбата на турнирите и споредба на учеството на различни тимови.

```
explain analyze select * from tournament_standings ts where ts.tournament_id=1
```

Execution Time: 72.741 ms

```
EXPLAIN ANALYZE
INSERT INTO tournament_teams (tournament_id, team_id, joined_at)
VALUES (
  (SELECT MIN(id) FROM tournaments),
  (SELECT MIN(id) FROM teams),
  CURRENT_TIMESTAMP
);
```

Execution Time: 0.474 ms

```
EXPLAIN ANALYZE
UPDATE tournament_teams
SET joined_at = CURRENT_TIMESTAMP
WHERE tournament_id = (SELECT MIN(tournament_id) FROM tournament_teams)
AND team_id = (SELECT MIN(team_id) FROM tournament_teams);
```

Execution Time: 2.829 ms

Времето на извршување на сите операции е прифатливо. Нема потреба од преуредување на прашалникот или креирање на индекс.

## View 7: player\_match\_stats

1. Погледот `player_match_stats` служи за приказ на статистиките на играчите во одиграните натпревари. Прикажува информации за корисникот, натпреварот, играта, резултатот, освоените поени, промената на ELO рејтингот и времето на приклучување. Дополнително, статистиките од натпреварот се групирани во JSON формат, што овозможува полесен преглед на различни типови статистики за секој играч. Овој поглед е корисен за анализа на индивидуалниот перформанс на играчите во натпреварите.

```
explain analyze select * from player_match_stats pms where pms.user_id=370 and match_id = 10
```

Execution Time: 0.250 ms

```

EXPLAIN ANALYZE
INSERT INTO match_participants (
  id,
  match_id,
  user_id,
  team_id,
  result,
  score,
  elo_change,
  joined_at
)
VALUES
(999999991, 100, 50, 793, 'win', 24, 15, CURRENT_TIMESTAMP),
(999999992, 100, 51, 793, 'win', 18, 12, CURRENT_TIMESTAMP),
(999999993, 100, 52, 794, 'loss', 14, -12, CURRENT_TIMESTAMP),
(999999994, 100, 53, 794, 'loss', 10, -15, CURRENT_TIMESTAMP);

```

Execution Time: 119.481 ms

```

BEGIN;
EXPLAIN ANALYZE
UPDATE match_participants
SET
  score = score + 1,
  elo_change = elo_change + 5
WHERE id = 999999991;
ROLLBACK;

```

Execution Time: 204.480 ms

Времето на извршување на сите операции е прифатливо. Нема потреба од преуредување на прашалникот или креирање на индекс.

### View 8: user\_profile

1. Погледот `user_profile` служи за приказ на основните информации за корисничкиот профил. Прикажува податоци за корисникот како корисничко име, е-пошта, локација, датум на креирање, аватар, биографија, држава, ранк поени и информации за активната претплата. Овој поглед е корисен за прикажување на целосен профил на корисникот на едно место, без потреба податоците посебно да се земаат од табелите `users`, `profiles`, `locations` и `subscriptions`

```
select * from user_profile up where user_id = 100000
```

Execution Time: 649.627 ms

Најбавниот дел од оваа операција е:

```

-> Parallel Seq Scan on subscriptions s (cost=0.00..8854.50 rows=1 width=28) (never executed)
   Filter: ((user_id = 8500000) AND ((end_at IS NULL) OR (end_at >= CURRENT_DATE)) AND (start_at <= CURRENT_DATE))

```

```

EXPLAIN ANALYZE
INSERT INTO users (
  id,|
  username,
  email,
  password_hash,
  location_id,
  created_at,
  updated_at
)
SELECT
  9999999999,
  'test_user_9999999999',
  'test_user_9999999999@example.com',
  'hashed_password',
  (SELECT MIN(id) FROM locations),
  CURRENT_TIMESTAMP,
  CURRENT_TIMESTAMP;

```

Execution Time: 265.126 ms

```

EXPLAIN ANALYZE
INSERT INTO subscriptions (
  id,
  user_id,
  plan,
  start_at,
  end_at,
  is_active
)
SELECT
  9999999999,
  u.id,
  'monthly',
  CURRENT_TIMESTAMP,
  CURRENT_TIMESTAMP + interval '1 month',
  TRUE
FROM users u
WHERE NOT EXISTS (
  SELECT 1
  FROM subscriptions s
  WHERE s.user_id = u.id
  AND s.is_active = TRUE
)

```

Execution Time: 1.404 ms

За тестирање на INSERT кај погледот user\_profile, внесувањето се извршува врз основните табели users и profiles, бидејќи самиот поглед не чува податоци, туку само ги прикажува преку JOIN. Тестот симулира реално сценарио на регистрација на нов корисник, каде што прво се внесуваат основните кориснички податоци во табелата users, а потоа се креира

неговиот профил во табелата profiles. Ова е реалистичен тест, бидејќи секој нов корисник во системот мора да има и поврзан профил.

```
EXPLAIN ANALYZE
UPDATE profiles
SET
  bio = 'Updated test bio',
  rank_points = rank_points + 10
WHERE user_id = 8500000;
Execution Time: 0.322 ms
```

Времето на операцијата select не е прифатливо. Ова може да се подобри со индекс.

```
CREATE INDEX idx_subscriptions_user_dates
ON subscriptions(user_id, start_at, end_at);
```

Со индекс:

```
select * from user_profile up where user_id = 100000
Execution Time: 0.251 ms
```

### View 9: Match History

1. Погледот **match\_history** служи за приказ на историјата на завршени мечови. Прикажува информации за играта, game mode, турнир, времето на почеток и крај, траење и локација на серверот.

Иницијалното мерење на погледот без индекс покажа Execution Time од **39,071ms** со Planning Time од **1,689ms**.

2. Најбавната операција е **Seq Scan** врз табелата matches, бидејќи базата скенира сите **1,000,000 редови** и ги филтрира според ended\_at IS NOT NULL. Поради големиот број на редови е потребен индекс за побрзо извршување.

```
Seq Scan on matches m (cost=0.00..22605.00 rows=1000000 width=56) (actual time=0.102..293.360 r
  Filter: ((ended_at IS NOT NULL) AND (ended_at IS NOT NULL))
```

```
CREATE INDEX idx_matches_ended_at
ON matches(ended_at)
WHERE ended_at IS NOT NULL;
```

```
EXPLAIN ANALYZE
SELECT * FROM match_history
WHERE ended_at IS NOT NULL;
```

3. Индексот се креира врз колоната ended\_at бидејќи погледот секогаш ги филтрира само завршените мечови (ended\_at IS NOT NULL). Со овој индекс PostgreSQL може побрзо да ги пронајде завршените мечови наместо да пра ви целосно скенирање на табелата.

```
SELECT со индекс
  Planning Time: 1.153 ms
  Execution Time: 1828.103 ms
```

INSERT со индекс

```
QUERY PLAN
-----
Insert on matches (cost=0.00..0.03 rows=0 width=0) (actual time=11.262..11.262 rows=0 loops=1)
-> Result (cost=0.00..0.03 rows=1 width=68) (actual time=0.004..0.005 rows=1 loops=1)
Planning Time: 0.038 ms
Trigger for constraint matches_game_id_fkey: time=0.341 calls=1
Trigger for constraint matches_game_mode_id_fkey: time=0.174 calls=1
Trigger for constraint matches_tournament_id_fkey: time=0.163 calls=1
Trigger for constraint matches_server_id_fkey: time=0.158 calls=1
Execution Time: 12.128 ms
```

## UPDATE со индекс

```
QUERY PLAN
-----
Update on matches (cost=0.42..8.45 rows=0 width=0) (actual time=0.211..0.211 rows=0 loops=1)
-> Index Scan using matches_pkey on matches (cost=0.42..8.45 rows=1 width=14) (actual time=0.211..0.211 rows=0 loops=1)
    Index Cond: (id = 1000003)
Planning Time: 0.162 ms
Execution Time: 0.482 ms
```