ИТМО

Факультет программной инженерии и компьютерной техники

Базы данных

Лабораторная работа №5

Преподаватель: Николаев Владимир Вячеславович

Выполнил:

Мельников Никита P33222

Санкт-Петербург

2023

Описание задания:

Добавить в ранее созданную базу данных (лр №4) триггеры для обеспечения комплексных ограничений целостности.

Реализовать функции и процедуры на основе описания бизнес-процессов, определенных при описании предметной области (лр №1). Должна быть обеспечена проверка корректности вводимых данных для созданных функций и процедур.

Необходимо произвести анализ использования созданной базы данных, выявить наиболее часто используемые объекты базы данных, виды запросов к ним. Результаты должны быть представлены в виде текстового описания.

На основании полученного описания требуется создать подходящие индексы и доказать, что они будут полезны для представленных в описании случаев использования базы данных.

Комплексные ограничения целостности:

Для начала, добавим в БД триггеры для обеспечения комплексных ограничений целостности. Эти триггеры были описаны ранее в лабораторной работе 3.

1. Инвайт может отправлять только человек с ролью организатора.

2. Инвайт может получать только человек с ролью игрока.

3. Ответ может посылать только человек с ролью игрока.

4. Ответ может получать только человек с ролью организатора.

5. Взнос может посылать только человек с ролью игрока.

6. Взнос может принимать только человек с ролью организатора.

7. Принимать участие в играх могут только игроки.

8. Следить за играми могут только арбитры.

9. Подготавливать игры может только персонал.

10. Быть ответственным за инвентарь может быть только персонал.

11. Атрибут человека has_accepted может принимать значения true/false только, если у человека роль игрока, у людей с другими ролями has_accepted может быть только null.

1, 2 *Инвайт может отправлять только человек с ролью организатора*

Инвайт может получать только человек с ролью игрока:

```
create or replace function check_invite_sending_funk() returns trigger
    as
    $$
    declare
        current_player_id int := new.player_id;
        current_org_id int := new.org_id;
    begin
        if (select role.name from person join role on person.role_id =
role.id where person.id = current_player_id) != 'Player'
            then
                raise exception 'Invite could be send only to players!';
        elseif (select role.name from person join role on person.role_id =
role.id where person.id = current_org_id) != 'Organizator'
            then
                raise exception 'Invite could be send only by organizators!';
        end if;
        return new;
    end;
    $$
    language 'plpgsql';


create or replace trigger check_invite_sending
    before insert or update on invite
    for each row
    execute function check_invite_sending_funk();
```

3, 4 *Ответ может посылать только человек с ролью игрока*

Ответ может получать только человек с ролью организатора:

```
create or replace function check_response_handling_funk() returns trigger
    as
    $$
    declare
        current_player_id int := new.player_id;
        current_org_id int := new.org_id;
    begin
        if (select role.name from role join person p on role.id = p.role_id
where p.id = current_player_id) != 'Player'
            then
                raise exception 'Only players could give responses!';
        elseif (select role.name from role join person p2 on role.id =
p2.role_id where p2.id = current_org_id) != 'Organizator'
            then
                raise exception 'Only organizators could receive responses!';
        end if;
        return new;
    end;
    $$
    language 'plpgsql';

create or replace trigger check_response_handling
    before insert or update on response
    for each row
    execute function check_response_handling_funk();
```

5, 6 *Взнос может посылать только человек с ролью игрока.*

Взнос может принимать только человек с ролью организатора:

```
create or replace function check_entrance_fee_handling_funk() returns trigger
    as
    $$
    declare
        current_player_id int := new.player_id;
        current_org_id int := new.org_id;
    begin
        if (select role.name from role join person p on role.id = p.role_id
where p.id = current_player_id) != 'Player'
            then
                raise exception 'Only players could pay entrance fees!';
        elseif (select role.name from role join person p2 on role.id =
p2.role_id where p2.id = current_org_id) != 'Organizator'
            then
                raise exception 'Only organizators could receive entance
fees!';
        end if;
        return new;
    end;
    $$
    language 'plpgsql';

create or replace trigger check_entrance_fee_handling
    before insert or update on entrance_fee
    for each row
    execute function check_entrance_fee_handling_funk();
```

7 *Принимать участие в играх могут только игроки*:

```
create or replace function check_game_participants_funk() returns trigger
    as
    $$
    declare
        first_player_id int := new.player1_id;
        second_player_id int := new.player2_id;
    begin
        if (select r.name from person join role r on person.role_id = r.id
where person.id = first_player_id) != 'Player'
            then
                raise exception 'First person is not a player!';
        elseif (select r.name from person join role r on person.role_id =
r.id where person.id = second_player_id) != 'Player'
            then
                raise exception 'Second person is not a player!';
        elseif first_player_id = second_player_id
            then
                raise exception 'Player can not play with himself!';
        end if;
        return new;
    end;
    $$
    language 'plpgsql';

create or replace trigger check_game_participants
    before insert or update on game
    for each row
    execute function check_game_participants_funk();
```

8 *Следить за играми могут только арбитры*:

```
create or replace function check_game_observer_funk() returns trigger
    as
    $$
    declare
        arbitr_id int := new.arb_id;
    begin
        if (select r.name from person join role r on person.role_id = r.id
where person.id = arbitr_id) != 'Arbitr'
            then
                raise exception 'Only arbitres could observe games!';
        end if;
        return new;
    end;
    $$
    language 'plpgsql';

create or replace trigger check_game_observer
    before insert or update on game
    for each row
    execute function check_game_observer_funk();
```

9 *Подготавливать игры может только персонал.*:

```
create or replace function check_game_preparators_funk() returns trigger
    as
    $$
    declare
        staff_id int := new.staff_id;
    begin
        if (select r.name from person join role r on person.role_id = r.id
where person.id = staff_id) != 'Staff'
            then
                raise exception 'Only staff could prepare games!';
        end if;
        return new;
    end;
    $$
    language 'plpgsql';

create or replace trigger check_game_preparators
    before insert or update on game_staff
    for each row
    execute function check_game_preparators_funk();
```

10 *Быть ответственным за инвентарь может быть только персонал*:

```
create or replace function check_inventory_responsible_funk() returns trigger
    as
    $$
    declare
        staff_id int := new.staff_id;
    begin
        if (select r.name from person join role r on person.role_id = r.id
where person.id = staff_id) != 'Staff'
            then
                raise exception 'Only staff could prepare games!';
        end if;
        return new;
    end;
    $$
    language 'plpgsql';

create or replace trigger check_inventory_responsible
    before insert or update on inventory
    for each row
    execute function check_inventory_responsible_funk();
```

11 *Атрибут человека has_accepted может принимать значения true/false только, если у человека роль игрока, у людей с другими ролями has_accepted может быть только null*:

```
create or replace function check_player_flag_funk() returns trigger
    as
    $$
    declare
        role_id int := new.role_id;
    begin
        if (select role.name from role where role.id = role_id) = 'Player'
            and new.has_accepted IS NULL
            then
                raise exception 'Players must have non-null has_accepted
attribute';
        elseif (select role.name from role where role.id = role_id) !=
'Player'
            and new.has_accepted IS NOT NULL
            then
                raise exception 'Non-players must have null has_accepted
atttribute';
        end if;
        return new;
    end;
    $$
    language 'plpgsql';

create or replace trigger check_player_flag
    before insert or update on person
    for each row
    execute function check_player_flag_funk();
```

Функции и процедуры:

Теперь, нам необходимо создать функции и процедуры, основанные на бизнес-процессах нашей предметной области.

1). Добавление нового человека с определенной ролью:

Было бы здорово иметь удобный способ добавить человека с той или иной ролью в базу данных.

2). Выслать игроку приглашение для участия в турнире;

Чтобы игрок смог сыграть в нашем турнире, он должен об этом самом турнире как-то узнать. Для этого один из организаторов турнира должен выслать приглашение данному игроку.

3). Получить ответа от игрока;

Игрок, если все же ответит на приглашение от организатора, может либо согласиться на участие в турнире.

4). Получить вступительный взнос от игрока;

Игрок, который хочет принять участие в турнире, должен оплатить вступительный взнос.

5). Зарегистрировать игрока на турнире;

Если все формальности, указанные в предметной области соблюдены, игрока регистрируют для участия в турнире.

6). Вывести всех игроков конкретного турнира;

Для отчетности нам важно понимать, какие игроки участвуют в том или ином турнире.

7). Показать все ответы игроков для конкретного турнира, где игрок ответил в срок;

Организаторам нужно знать все ответы игроков, где они ответили в срок, для их регистрации в турнире.

8). Показать все приглашения и все ответы (если они есть) для игрока на всех турнирах.

Такая функция позволит отслеживать активность игрока и его вовлеченность в участии в турнирах.

9). Запланировать игру в рамках определенного турнира;

Нам нужен способ планировать игры между двумя игроками в рамках определенного турнира.

10). Вывести расписание игр для конкретного турнира;

Важно понимать расписание игр для каждого отдельного турнира.

11). Вывести все игры в определенном турнире для конкретного игрока;

Необходимо понимать расписание игрока в рамках того или иного турнира.

12). Выставить и обработать результат игры;

После того, как игроки сыграли игру, необходимо записать соответствующий результат в таблицу счета, а также в саму игру.

13). Вывести таблицу счета игроков для конкретного турнира;

Так как шахматы – игра соревновательная, нам необходим способ для того, чтобы узнать, кто победил в том или ином турнире.

14). Назначить человека из персонала ответственным за ту или иную игру;

За каждую игру должен быть ответственный человек из персонала, который подготавливает игру (ставит стулья, выставляет фигуры на доске и т.д.).

Код всех функций и процедур:

```
/* functions and procedures */
/* 1 */
create or replace procedure add_new_player(name VARCHAR, surname varchar, age
int)
    as
    $$
    declare
        player_id int := (select id+1 from person order by id desc limit 1);
        role_id int := (select id from role where role.name = 'Player');
    begin
        insert into person values (player_id, name, surname, age, false,
role_id);
    end;
    $$
    language 'plpgsql';

create or replace procedure add_new_org(name VARCHAR, surname varchar, age
int)
    as
    $$
    declare
        org_id int := (select id+1 from person order by id desc limit 1);
        role_id int := (select id from role where role.name = 'Organizator');
    begin
        insert into person values (org_id, name, surname, age, null,
role_id);
    end;
    $$
    language 'plpgsql';

create or replace procedure add_new_arb(name VARCHAR, surname varchar, age
int)
    as
    $$
```

```
    declare
        arb_id int := (select id+1 from person order by id desc limit 1);
        role_id int := (select id from role where role.name = 'Arbitr');
    begin
        insert into person values (arb_id, name, surname, age, null,
role_id);
    end;
    $$
    language 'plpgsql';

create or replace procedure add_new_staff(name VARCHAR, surname varchar, age
int)
    as
    $$
    declare
        staff_id int := (select id+1 from person order by id desc limit 1);
        role_id int := (select id from role where role.name = 'Staff');
    begin
        insert into person values (staff_id, name, surname, age, null,
role_id);
    end;
    $$
    language 'plpgsql';



/* 2 */
create or replace procedure send_invite(org_id integer, player_id integer,
comp_info text, tournament_id integer)
    as
    $$
    declare
        cur_id int = (select id+1 from invite order by id desc limit 1);
        cur_date date = (select current_date);
        date_to_response date = (select cast(TO_CHAR((SELECT current_date +
(interval '1 month')), 'YYYY-MM-DD') as date));

    begin
        insert into invite values (cur_id, comp_info, cur_date,
date_to_response, org_id, player_id, tournament_id);
    end;
    $$
    language 'plpgsql';



/* 3 */
create or replace procedure get_response(org_id integer, player_id integer,
answer text, tournament_id integer)
    as
    $$
    declare
        cur_id int = (select id+1 from response order by id desc limit 1);
        cur_date date = (select current_date);

    begin
        insert into response values (cur_id, answer, cur_date, org_id,
player_id, tournament_id);
    end;
    $$
    language 'plpgsql';
```

```
/* 4 */
create or replace procedure get_entrance_fee(org_id integer, player_id
integer, money int, tournament_id integer)
    as
    $$
    declare
        cur_id int = (select id+1 from entrance_fee order by id desc limit
1);
    begin
        insert into entrance_fee values (cur_id, money, org_id, player_id,
tournament_id);
    end;
    $$
    language 'plpgsql';



/* 5 */
create or replace procedure register_player(player_id int, tournament_id int)
    as
    $$
    begin
      update person set has_accepted = true where id = player_id;
      insert into tournament_person values (tournament_id, player_id);
    end;
    $$
    language 'plpgsql';




/* 6 */
create or replace function show_all_players(t_id int)
    returns table(
        player_id int,
        name varchar,
        surname varchar,
        age int,
        role varchar,
        tournament varchar,
        tournament_description text
    )
    as
    $$
    begin
        return query
        select p.id, p.name, p.surname, p.age, r.name, t.name, t.description
        from person p join tournament_person tp on p.id = tp.person_id
        join tournament t on t.id = tp.tournament_id
        join role r on r.id = p.role_id
        where t.id = t_id and r.name = 'Player';

    end;
    $$
    language 'plpgsql';




/* 7 */
create or replace function show_all_in_time_responses(t_id int)
    returns table(
        response_id int,
        answer text,
```

```
        player_id int,
        player_name varchar,
        player_surname varchar
        )
    as
    $$
    begin
        return query
        select r.id, r.answer, p.id, p.name, p.surname
        from response r join person p on r.player_id = p.id
        join invite i on i.player_id = p.id
        where r.tournament_id = t_id and r.send_date <= i.date_to_response;
    end;
    $$
    language 'plpgsql';

select r.id, r.answer, p.id, p.name, p.surname from response r join person p
on r.player_id = p.id join invite i on i.player_id = p.id
         where r.tournament_id = 1 and r.send_date <= i.date_to_response;



/* 8 */
create or replace function show_player_activity(p_id int)
    returns table(
        player_id int,
        competition_info text,
        answer text
        )
    as
    $$
    begin
        if (select role_id from person p where p.id = p_id) = 1
        then
        return query
        select p.id, i.competition_info, r.answer
        from invite i join person p on p.id = i.player_id
            left join response r on r.player_id = p.id where i.player_id =
p_id;
        else
            raise exception 'Person with id % is not a player!', p_id;
        end if;
    end;
    $$
    language 'plpgsql';


/* 9 */
create or replace procedure create_game(p1_id int, p2_id int, a_id int, t_id
int, tr int)
    as
    $$
    declare
        cur_id int = (select id+1 from game order by id desc limit 1);
        cur_date date := (select current_date);
    begin
        insert into game values (cur_id, p1_id, p2_id, a_id, t_id, null, tr,
cur_date);
    end;
    $$
    language 'plpgsql';


/* 10 */
```

```
create or replace function show_tournament_schedule(t_id int)
    returns table(
        game_id int,
        first_name varchar,
        first_surname varchar,
        second_name varchar,
        second_surname varchar,
        start_date date
    )
    as
    $$
    begin
        return query
        select g.id, p1.name, p1.surname, p2.name, p2.surname, g.start_date
        from tournament t join game g on t.id = g.tournament_id
        join person p1 on p1.id=g.player1_id join
        person p2 on p2.id = g.player2_id where t.id = t_id;
    end;
    $$
    language 'plpgsql';



/* 11 */
create or replace function show_player_schedule(t_id int, p_id int)
    returns table(
        game_id int,
        first_name varchar,
        first_surname varchar,
        second_name varchar,
        second_surname varchar,
        start_date date
    )
    as
    $$
    begin
        return query
        select g.id, p1.name, p1.surname, p2.name, p2.surname, g.start_date
        from tournament t join game g on t.id = g.tournament_id
        join person p1 on p1.id=g.player1_id join
        person p2 on p2.id = g.player2_id where t.id = t_id
        and (p1.id = p_id or p2.id = p_id) order by g.tour;
    end;
    $$
    language 'plpgsql';



/* 12 */
/* result: 0 - first player win, 1 - draw, 2 - second player win */
create or replace procedure calculate_game_result(g_id int, res int)
    as
    $$
    declare
        p1_id int := (select player1_id from game where id = g_id);
        p2_id int := (select player2_id from game where id = g_id);
    begin
        if res > 2 or res < 0
            then
                raise exception 'result: 0 - first player win, 1 - draw, 2 -
second player win, % - is not an option', res;
        elseif res = 0
            then
```

```
                update game set result = res where id = g_id;
                update score set score = score+1 where player_id = p1_id;
        elseif res = 2
            then
                update game set result = res where id = g_id;
                update score set score = score+1 where player_id = p2_id;
        else
                update game set result = res where id = g_id;
                update score set score = score+0.5 where player_id = p1_id;
                update score set score = score+0.5 where player_id = p2_id;
        end if;
    end;
    $$
    language 'plpgsql';



/* 13 */
create or replace function show_players_scores(t_id int)
    returns table(
        score_id int,
        player_name varchar,
        player_surname varchar,
        score decimal
    )
    as
    $$
    begin
        return query
        select s.id, p.name, p.surname, s.score
        from score s join tournament t on s.tournament_id=t.id
        join person p on p.id = s.player_id
        where t.id = t_id order by s.score desc;
    end;
    $$
    language 'plpgsql';




/* 14 */
create or replace procedure set_responsible_for_game(staff_id int, game_id
int)
    as
    $$
    declare
        tgame_id int = (select t.id from tournament t join game g on t.id =
g.tournament_id where g.id = game_id);
        tstaff_id int = (select t.id from tournament t join tournament_person
tp on t.id = tp.tournament_id join person p on p.id = tp.person_id where p.id
= staff_id);
    begin
        if tgame_id = tstaff_id
        then
            insert into game_staff values (game_id, staff_id);
        else
            raise exception 'Staff member and game are in different
tournaments';
        end if;
    end;
    $$
    language 'plpgsql';
```

Анализ созданной базы данных и создание индексов:

Необходимо провести анализ созданной базы данных для того, чтобы понять, в каких местах стоит создать индексы для повышения производительности базы данных.

Стоит помнить, что индексы будут наиболее эффективны в тех случаях, когда запись в таблицу происходит относительно реже, чем поиск по ней. В противном случае создание индексов может привести не к улучшению производительности, базы данных, а наоборот, к её снижению.

Следует также помнить, что не во всех таблицах имеется достаточно большое количество строк, а, поэтому, скорее всего не получится увидеть прирост производительности.

Рассмотрим запрос из функции 7 (Показать все ответы игроков для конкретного турнира, где игрок ответил в срок;)

Здесь мы можем создать следующий индекс:

create index response_tournament_indx on response (tournament_id);

Т.к. ответы игроки присылают не так часто(следовательно, и новые записи в таблице ответов появляются не так часто), а организаторам необходимо постоянно просматривать приходящие ответы, для того, чтобы регистрировать игроков, можно считать, что данный индекс не понесет больших потерь в производительности для БД.

Результат анализа до создания индекса:

```
studs=> explain analyze select r.id, r.answer, p.id, p.name, p.surname
       from response r join person p on r.player_id = p.id
       join invite i on i.player_id = p.id
       where r.tournament_id = 3 and r.send_date <= i.date_to_response;
                                                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------------------
 Nested Loop  (cost=76.51..308.72 rows=232 width=27) (actual time=1.512..4.123 rows=708 loops=1)
   Join Filter: (r.player_id = p.id)
   ->  Hash Join  (cost=76.22..223.85 rows=237 width=15) (actual time=1.486..2.733 rows=708 loops=1)
         Hash Cond: (i.player_id = r.player_id)
         Join Filter: (r.send_date <= i.date_to_response)
         Rows Removed by Join Filter: 2
         ->  Seq Scan on invite i  (cost=0.00..120.00 rows=5000 width=8) (actual time=0.009..0.793 rows=5000 loops=1)
         ->  Hash  (cost=67.35..67.35 rows=710 width=15) (actual time=0.776..0.776 rows=710 loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 42kB
               ->  Seq Scan on response r  (cost=0.00..67.35 rows=710 width=15) (actual time=0.018..0.629 rows=710 loops=1)
                     Filter: (tournament_id = 3)
                     Rows Removed by Filter: 2838
   ->  Index Scan using person_pkey on person p  (cost=0.28..0.35 rows=1 width=20) (actual time=0.001..0.001 rows=1 loops=708)
         Index Cond: (id = i.player_id)
 Planning Time: 1.750 ms
 Execution Time: 4.288 ms
(16 строк)
```

После создания индекса:

```
studs=> explain analyze select r.id, r.answer, p.id, p.name, p.surname
       from response r join person p on r.player_id = p.id
       join invite i on i.player_id = p.id
       where r.tournament_id = 3 and r.send_date <= i.date_to_response;
                                                                  QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------------------------------------------
 Nested Loop  (cost=40.93..273.14 rows=232 width=27) (actual time=0.790..2.925 rows=708 loops=1)
   Join Filter: (r.player_id = p.id)
   ->  Hash Join  (cost=40.64..188.27 rows=237 width=15) (actual time=0.784..1.662 rows=708 loops=1)
         Hash Cond: (i.player_id = r.player_id)
         Join Filter: (r.send_date <= i.date_to_response)
         Rows Removed by Join Filter: 2
         ->  Seq Scan on invite i  (cost=0.00..120.00 rows=5000 width=8) (actual time=0.005..0.446 rows=5000 loops=1)
         ->  Hash  (cost=31.77..31.77 rows=710 width=15) (actual time=0.336..0.336 rows=710 loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 42kB
               ->  Index Scan using response_tournament_indx on response r  (cost=0.28..31.77 rows=710 width=15) (actual time=0.007..0.204 rows=710 loops=1)
                     Index Cond: (tournament_id = 3)
   ->  Index Scan using person_pkey on person p  (cost=0.28..0.35 rows=1 width=20) (actual time=0.001..0.001 rows=1 loops=708)
         Index Cond: (id = i.player_id)
 Planning Time: 0.218 ms
 Execution Time: 2.991 ms
(15 строк)
```

Здесь можно увидеть, что стоимость снизилась на 12%, а скорость увеличилась на 47%.

Стоит отметить, что это достаточно скромный результат, однако, чем больше у нас будет турниров в системе, тем сильнее результат будет улучшаться.

Теперь рассмотрим запрос из функции 8 (Показать все приглашения и все ответы (если они есть) для игрока на всех турнирах.)

Здесь можно создать следующие индексы:

create index invite_player_id_indx on invite (player_id);

create index response_player_id_indx on response (player_id);

Приглашения рассылаются непосредственно перед самим турниром в небольшом количестве, а ответов может быть получено еще меньше, чем приглашений было отправлено, потому что игрок может банально проигнорировать приглашение и не выслать ответ. Поэтому, создание данных индексов приведет к увеличению производительности БД.

Результаты анализа до создания индексов:

```
studs=> explain analyze select p.id, i.competition_info, r.answer
    from invite i join person p on p.id = i.player_id
        left join response r on r.player_id = p.id
    where i.player_id = 1;
                                                                QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------------------------------
 Nested Loop Left Join  (cost=0.28..204.17 rows=1 width=64) (actual time=0.029..0.990 rows=1 loops=1)
   Join Filter: (r.player_id = p.id)
   ->  Nested Loop  (cost=0.28..136.81 rows=1 width=61) (actual time=0.021..0.633 rows=1 loops=1)
         ->  Seq Scan on invite i  (cost=0.00..132.50 rows=1 width=61) (actual time=0.010..0.620 rows=1 loops=1)
               Filter: (player_id = 1)
               Rows Removed by Filter: 4999
         ->  Index Only Scan using person_pkey on person p  (cost=0.28..4.30 rows=1 width=4) (actual time=0.009..0.010 rows=1 loops=1)
               Index Cond: (id = 1)
               Heap Fetches: 0
   ->  Seq Scan on response r  (cost=0.00..67.35 rows=1 width=7) (actual time=0.006..0.355 rows=1 loops=1)
         Filter: (player_id = 1)
         Rows Removed by Filter: 3547
 Planning Time: 0.426 ms
 Execution Time: 1.020 ms
(14 строк)
```

После создания индексов:

```
studs=> explain analyze select p.id, i.competition_info, r.answer
    from invite i join person p on p.id = i.player_id
        left join response r on r.player_id = p.id
    where i.player_id = 1;
                                                                 QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 Nested Loop Left Join  (cost=0.84..20.92 rows=1 width=64) (actual time=0.054..0.057 rows=1 loops=1)
   Join Filter: (r.player_id = p.id)
   ->  Nested Loop  (cost=0.56..12.61 rows=1 width=61) (actual time=0.043..0.045 rows=1 loops=1)
         ->  Index Scan using invite_player_id_indx on invite i  (cost=0.28..8.30 rows=1 width=61) (actual time=0.027..0.028 rows=1 loops=1)
               Index Cond: (player_id = 1)
         ->  Index Only Scan using person_pkey on person p  (cost=0.28..4.30 rows=1 width=4) (actual time=0.009..0.009 rows=1 loops=1)
               Index Cond: (id = 1)
               Heap Fetches: 0
   ->  Index Scan using response_player_id_indx on response r  (cost=0.28..8.30 rows=1 width=7) (actual time=0.008..0.009 rows=1 loops=1)
         Index Cond: (player_id = 1)
 Planning Time: 0.230 ms
 Execution Time: 0.095 ms
(12 строк)
```

Здесь можно увидеть, что стоимость снизилась в 10,2 раза, а время выполнения запроса снизилось в 10.7 раз.

Вывод:

Выполняя данную лабораторную работу, я разработал комплексные ограничения целостности, разработал функции и процедуры, описывающие бизнес-процессы предметной области, а также провел анализ и создал индексы для увеличения эффективности работы БД.