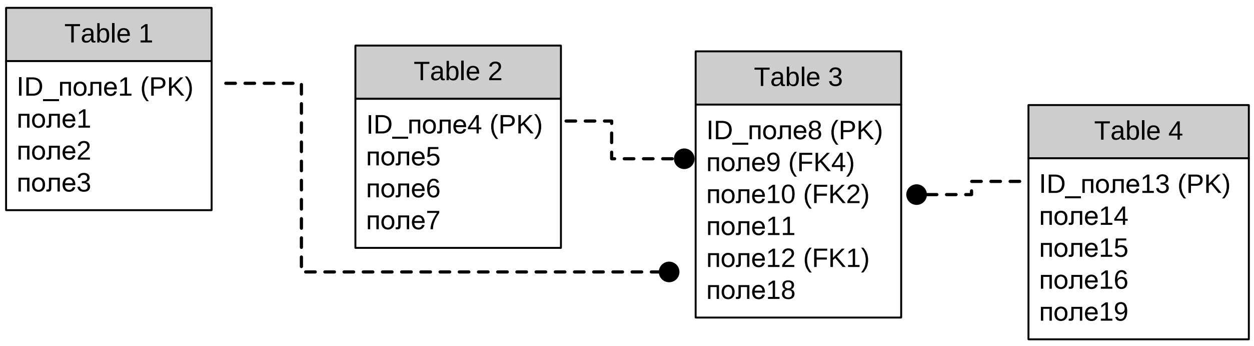
Оглавление

1 Задание

79	Ведение базы данных пункта проката видеопродукции	Ведение классификаторов и информационной базы по видеофильмам и их пользователям в пункте проката. Учет приема, выдачи видеофильмов, расчет размера оплаты.	2

Вариант 2



2 Введение

В настоящее время существует довольно много ПО(программного обеспечения) для реалезации коммерческой деятельности, в том числе, и для проката фильмов. Но его аренда стоит довольно дорого и продается в виде долгосрочной подписки, что довольно невыгодно мелким предприятиям. Актуальность моей работы заключается в том, что современным “малым” предпринимателям необходимо недорогостоящее ПО для ведения своей коммерческой деятельности.

Целью моей работы является создание бюджетного, портативного кроссплатформенного ПО для обеспечения нужд малых предпринимателей по выдаче фильмов в аренду. Для реалезации данной задачи необходимо реализовать как сторону клиента, которой будет пользоваться предприниматель для выдачи и приема заказов на аренду фильмов, так и сторону сервера, на которой находится база данных хранящяя в себе всю информацию о пользователях сервиса и данные о транзакции по аренде фильмов.

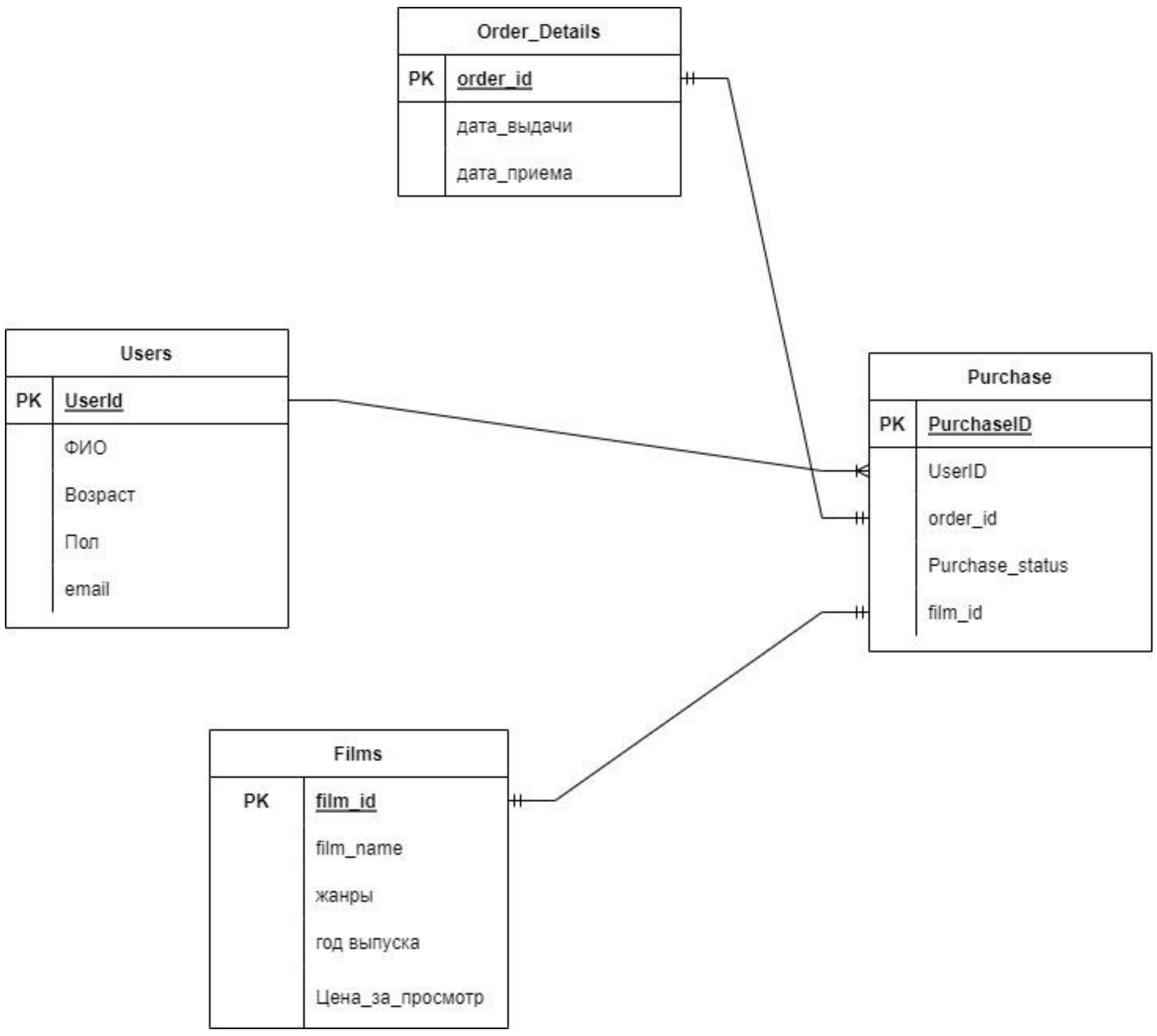
В качестве стека технологий предлагается использовать фреймворк PySide6 для реалезации интерфейса приложения, а так же СУБД PostgresSQL в качестве “движка” для базы данных. Базу данных решено было создать из четырех таблиц, связанных превичными и вторичными ключами. Описание функций таблиц, составляющих базу данных приложения:

1) Таблица Users хранит в себе данные пользователей сервиса проката фильмов
2) Таблица Films хранит в себе данные о фильмах, выдаваемых на прокат данным сервисом.
3) Таблица Purchase содержит информацию о заказах пользователей сервиса проката фильмов.
4) Таблица Order_Details хранит в себе данные о длительности выдачи фильма в прокат.

3 База данных

3.1 Создание Таблиц

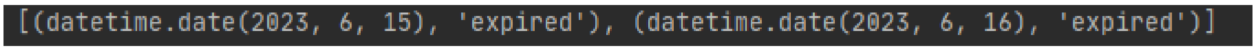
Ниже представлена структура таблиц БД:



```
CREATE TABLE Users (
        Id Serial PRIMARY KEY,
        First_name CHARACTER VARYING(30),
        Last_name CHARACTER VARYING(30),
        Email CHARACTER VARYING(30),
        Age int
);
CREATE TABLE IF NOT EXISTS Films (
        Film_id Serial PRIMARY KEY,
        Film_name CHARACTER VARYING(50),
        Genres CHARACTER VARYING(100),
        Year_of date,
        Cost_for_watch int2
    );
    CREATE TABLE IF NOT EXISTS Order_Details (
        Order_id Serial PRIMARY KEY,
        Date_of_issue date,
        Date_of_return date
    );
    CREATE TABLE IF NOT EXISTS Purchase (
        Purchase_id Serial PRIMARY KEY,
        UserId int,
        OrderId int,
        FilmId int,
        Purchase_status bool,
        Foreign Key (UserId) REFERENCES Users (Id) ON DELETE
CASCADE ON UPDATE CASCADE,
        Foreign Key (OrderId) REFERENCES Order_Details (Order_id) ON
DELETE CASCADE ON UPDATE CASCADE,
        Purchase_status bool,
        Foreign Key (FilmId) REFERENCES Films (Film_id) ON DELETE
CASCADE ON UPDATE CASCADE
    );
```

3.2 Составной многотабличный запрос с CASE-выражением

```
1)SELECT Date_of_return,
  CASE
      WHEN Date_of_return < (SELECT current_date)   THEN 'active'
      ELSE 'expired'
  END AS "rent_date_status"
FROM Order_Details;
```



```
[(datetime.date(2023, 6, 15), 'expired'), (datetime.date(2023, 6, 16), 'expired')]
```

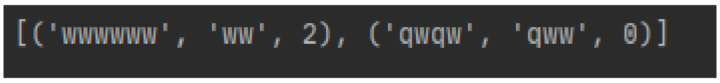
3.3 Многотабличный VIEW, с возможностью его обновления

```
CREATE OR REPLACE VIEW full_purchase AS SELECT usr.first_name,
    usr.last_name,
    f.film_name,
    f.cost_for_watch,
          pr.purchase_status
   FROM purchase pr
     JOIN users usr ON pr.userid = usr.id
     JOIN films f ON pr.filmid = f.film_id;


          CREATE OR REPLACE FUNCTION update_full_purchase_view()
RETURNS TRIGGER AS $$
              BEGIN
                    IF (TG_OP = 'DELETE') THEN
                              DELETE FROM films WHERE film_name = OLD.film_name;
                              DELETE FROM users WHERE first_name = OLD.first_name
AND last_name=OLD.last_name;
                              IF NOT FOUND THEN RETURN NULL; END IF;
                              RETURN OLD;
                    ELSIF (TG_OP = 'UPDATE') THEN
```

```
                        UPDATE films SET cost_for_watch = NEW.cost_for_watch
WHERE film_name = OLD.film_name;
                        UPDATE purchase SET purchase_status=true WHERE userid =
(SELECT id FROM users WHERE first_name=NEW.first_name AND
last_name=NEW.last_name) AND filmid =
        (SELECT film_id FROM films WHERE film_name=NEW.film_name);
                        IF NOT FOUND THEN RETURN NULL; END IF;
                        RETURN NEW;
                ELSIF (TG_OP = 'INSERT') THEN
                        INSERT INTO films (film_name,cost_for_watch) VALUES
(NEW.film_name,NEW.cost_for_watch);
                        IF NOT FOUND THEN RETURN NULL; END IF;
                        RETURN NEW;
                END IF;
                END;
        $$ LANGUAGE plpgsql;

        CREATE OR REPLACE TRIGGER update_full_purchase_trigger
        INSTEAD OF INSERT OR UPDATE OR DELETE ON full_purchase
           FOR EACH ROW EXECUTE PROCEDURE
update_full_purchase_view();
```

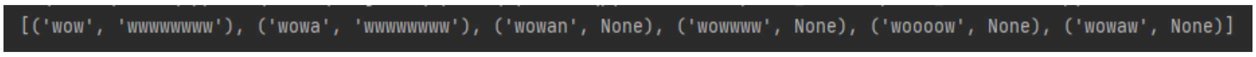
3.4 Запросы, содержащие подзапрос в разделах SELECT, FROM и WHERE

```
        SELECT first_name,last_name, (SELECT COUNT(UserID) cnt FROM
Purchase pr WHERE pr.UserID=usr.Id ) FROM Users usr;
```



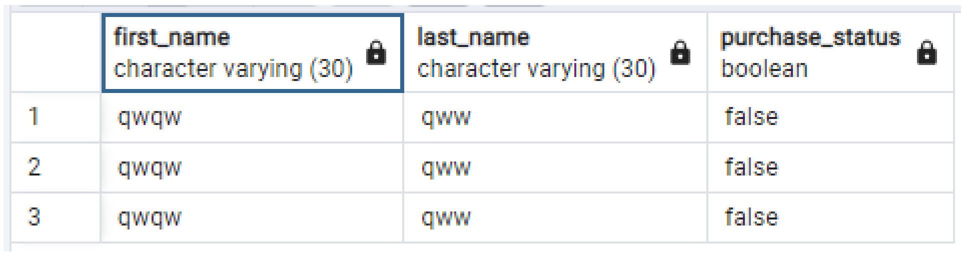
```
[('wwwwww', 'ww', 2), ('qwqw', 'qww', 0)]
```

SELECT film_name,(SELECT first_name || last_name FROM Users WHERE Id=(SELECT UserID FROM Purchase pr WHERE fl.film_id=pr.filmid)) FROM Films fl;



```
[('wow', 'wwwwwwww'), ('wowa', 'wwwwwwww'), ('wowan', None), ('wowwww', None), ('woooow', None), ('wowaw', None)]
```

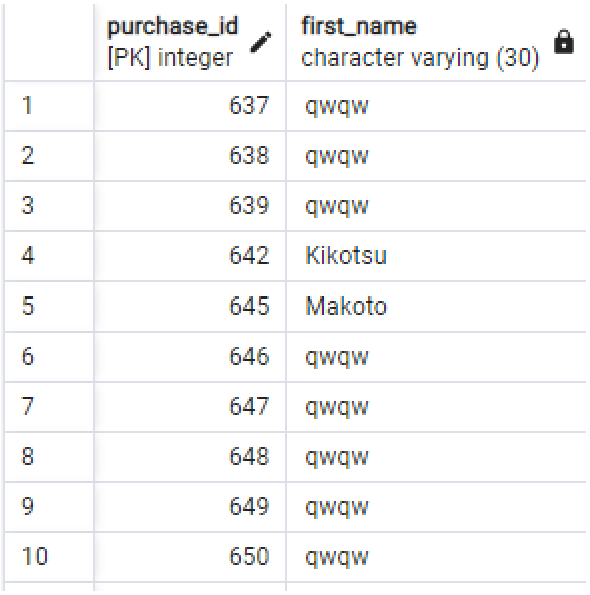
SELECT * FROM (SELECT first_name, last_name, purchase_status FROM users us JOIN purchase pr ON us.id = pr.userid) AS users_purchase_status;



	first_name character varying (30)	last_name character varying (30)	purchase_status boolean
1	qwqw	qww	false
2	qwqw	qww	false
3	qwqw	qww	false

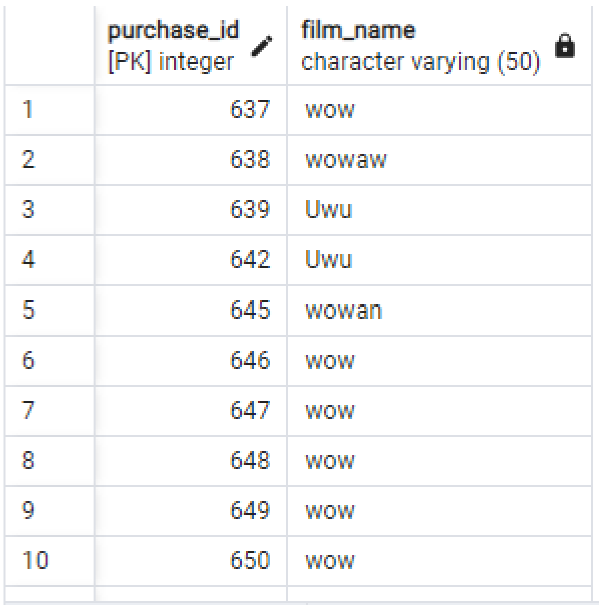
3.5 Коррелированные подзапросы

SELECT pr.purchase_id,(SELECT first_name FROM Users us WHERE us.id=pr.userid) FROM purchase pr;



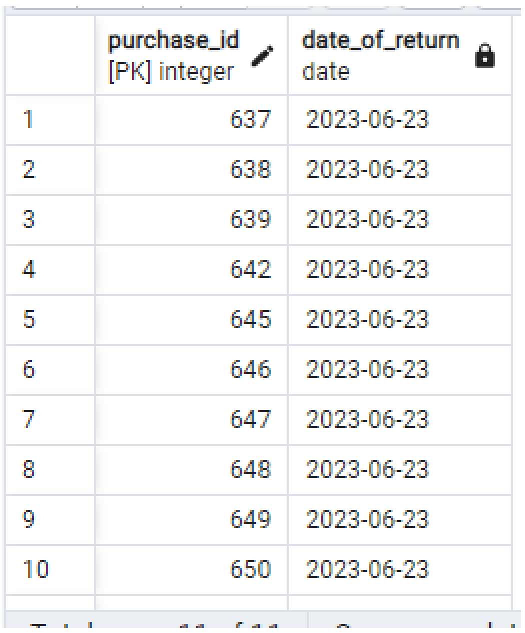
	purchase_id [PK] integer	first_name character varying (30)
1	637	qwqw
2	638	qwqw
3	639	qwqw
4	642	Kikotsu
5	645	Makoto
6	646	qwqw
7	647	qwqw
8	648	qwqw
9	649	qwqw
10	650	qwqw

SELECT pr.purchase_id,(SELECT film_name FROM Films fl WHERE fl.film_id=pr.filmid) FROM purchase pr;



	purchase_id [PK] integer	film_name character varying (50)
1	637	wow
2	638	wowaw
3	639	Uwu
4	642	Uwu
5	645	wowan
6	646	wow
7	647	wow
8	648	wow
9	649	wow
10	650	wow

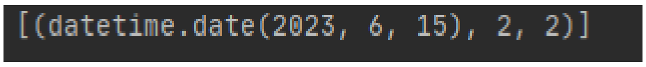
SELECT pr.purchase_id, (SELECT date_of_return FROM order_details od WHERE date_of_return>current_date AND od.order_id=pr.orderid) FROM purchase pr;



	purchase_id [PK] integer	date_of_return date
1	637	2023-06-23
2	638	2023-06-23
3	639	2023-06-23
4	642	2023-06-23
5	645	2023-06-23
6	646	2023-06-23
7	647	2023-06-23
8	648	2023-06-23
9	649	2023-06-23
10	650	2023-06-23

3.6 Многотабличный запрос, содержащий группировку записей, агрегатные функции и параметр, используемый в разделе HAVING

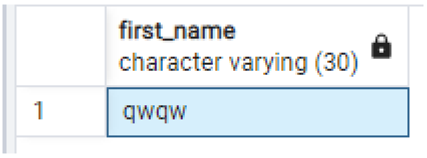
SELECT Date_of_return,COUNT(pr.Purchase_ID),COUNT(pr.userID) FROM Purchase pr JOIN Order_Details od ON pr.orderid=od.order_id GROUP BY Date_of_return HAVING COUNT(pr.Purchase_ID)>0;



```
[(datetime.date(2023, 6, 15), 2, 2)]
```

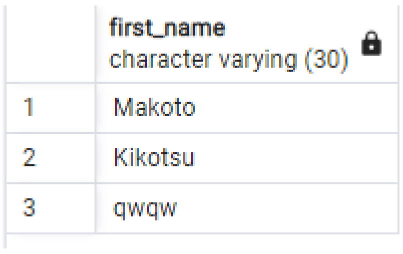
3.7 Запросы, содержащие предикаты ANY(SOME) или ALL

SELECT DISTINCT first_name FROM Users WHERE id = ANY (SELECT userid FROM Purchase);



	first_name character varying (30)
1	qwqw

SELECT DISTINCT us.first_name FROM Users us JOIN Purchase pr ON us.id=pr.userid JOIN Order_details od ON pr.orderid=od.order_id WHERE od.date_of_issue = ALL (SELECT current_date);

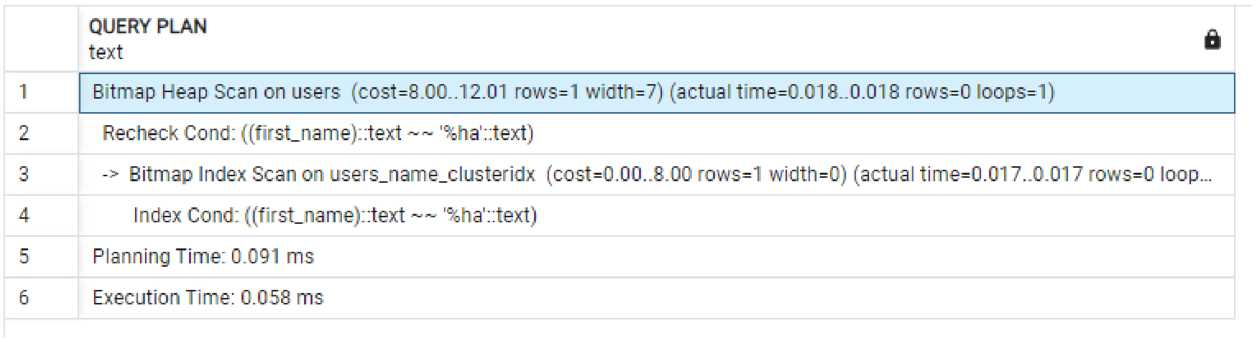


	first_name character varying (30)
1	Makoto
2	Kikotsu
3	qwqw

3.8 Создание индексов

CREATE INDEX users_name_clusteridx ON users USING gin(first_name gin_trgm_ops);

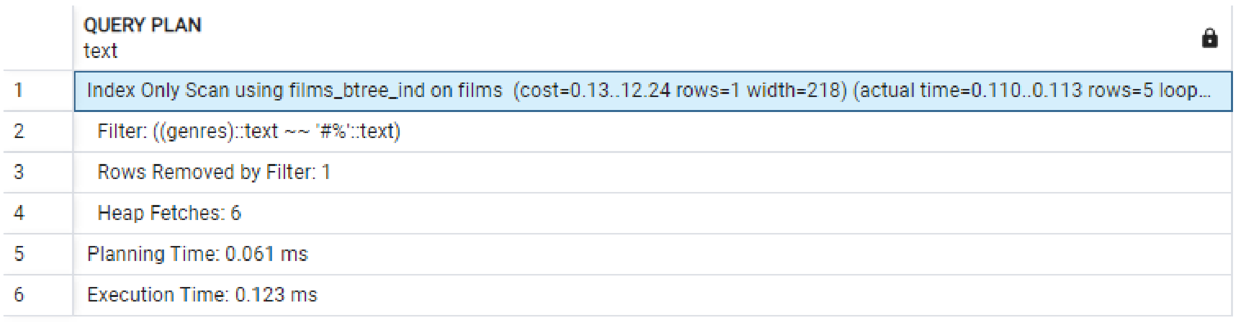
CREATE EXTENSION pg_trgm; EXPLAIN ANALYZE SELECT first_name FROM users WHERE first_name LIKE '%ha';



```
QUERY PLAN
text
1  Bitmap Heap Scan on users  (cost=8.00..12.01 rows=1 width=7) (actual time=0.018..0.018 rows=0 loops=1)
2    Recheck Cond: ((first_name)::text ~~ '%ha'::text)
3    ->  Bitmap Index Scan on users_name_clusteridx  (cost=0.00..8.00 rows=1 width=0) (actual time=0.017..0.017 rows=0 loop...
4          Index Cond: ((first_name)::text ~~ '%ha'::text)
5  Planning Time: 0.091 ms
6  Execution Time: 0.058 ms
```

CREATE INDEX films_btree_ind ON films USING btree(genres);

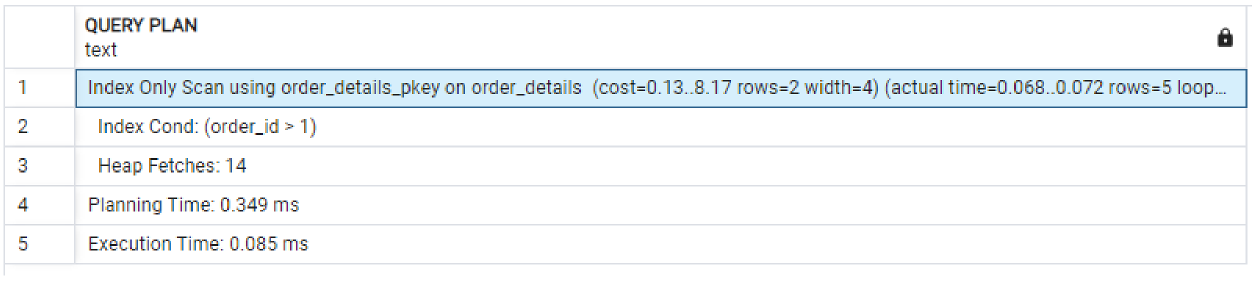
EXPLAIN ANALYZE SELECT genres FROM films WHERE genres LIKE '#%';



```
QUERY PLAN
text
1  Index Only Scan using films_btree_ind on films  (cost=0.13..12.24 rows=1 width=218) (actual time=0.110..0.113 rows=5 loop...
2    Filter: ((genres)::text ~~ '#%'::text)
3    Rows Removed by Filter: 1
4    Heap Fetches: 6
5  Planning Time: 0.061 ms
6  Execution Time: 0.123 ms
```

CREATE INDEX purchase_gist ON order_details USING hash (order_id);

EXPLAIN ANALYZE SELECT order_id FROM order_details WHERE order_id>1;



```
QUERY PLAN
text
1  Index Only Scan using order_details_pkey on order_details  (cost=0.13..8.17 rows=2 width=4) (actual time=0.068..0.072 rows=5 loop...
2    Index Cond: (order_id > 1)
3    Heap Fetches: 14
4  Planning Time: 0.349 ms
5  Execution Time: 0.085 ms
```

3.9 Автоматическое заполнение поля с тригером

В данном задании происходит заполнение поля purchase_status, хранящее в себе статус аренды заказа, по срабатывании триггера.

```
        CREATE OR REPLACE FUNCTION ins_purchase_status_func()
RETURNS TRIGGER AS $$
          BEGIN
            NEW.purchase_status := 0;
                INSERT INTO order_details(date_of_issue, date_of_return)
VALUES (current_date, current_date+7);
                NEW.orderid  = currval('order_details_order_id_seq');
            RETURN NEW;
          END;
        $$ LANGUAGE plpgsql;

        CREATE OR REPLACE TRIGGER ins_purchase_status_trigger
        BEFORE INSERT ON purchase
        FOR EACH ROW
        EXECUTE PROCEDURE ins_purchase_status_func();
```

```
        CREATE OR REPLACE FUNCTION del_purchase_func() RETURNS
TRIGGER AS $$
          BEGIN
                DELETE FROM order_details WHERE
order_details.order_id=OLD.orderid;
                RETURN OLD;
          END;
        $$ LANGUAGE plpgsql;

        CREATE OR REPLACE TRIGGER del_purchase_trigger
        AFTER DELETE ON purchase
        FOR EACH ROW
        EXECUTE PROCEDURE del_purchase_func();
```

```
CREATE OR REPLACE FUNCTION upd_purchase_func() RETURNS
TRIGGER AS $$
  BEGIN
        NEW.purchase_status := 1;
        RETURN NEW;
  END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER upd_purchase_trigger
BEFORE UPDATE ON purchase
FOR EACH ROW
EXECUTE PROCEDURE upd_purchase_func();
```

3.10 Операции добавления, удаления и обновления таблиц

```
      CREATE OR REPLACE PROCEDURE insert_users(fn CHARACTER
VARYING(30), lnm CHARACTER VARYING(30),em CHARACTER
VARYING(30),agge int)
      Language SQL AS $$
      INSERT INTO Users VALUES (0,fn,lnm,em,agge);
      $$;

      CREATE OR REPLACE PROCEDURE update_users(fn CHARACTER
VARYING(30), lnm CHARACTER VARYING(30),em CHARACTER
VARYING(30),agge int)
      Language SQL AS $$
      UPDATE Users SET first_name = fn, last_name=lnm, email =em,
age=agge;
      $$;
```

```
CREATE OR REPLACE PROCEDURE delete_users(fn CHARACTER VARYING(30), lnm CHARACTER VARYING(30),em CHARACTER VARYING(30),agge int)
Language SQL AS $$
DELETE FROM Users WHERE first_name = fn AND last_name=lnm AND email =em AND age=agge;
$$;

CREATE OR REPLACE PROCEDURE insert_films(fn CHARACTER VARYING(30), gnrs CHARACTER VARYING(30),years date, cost_for int)
Language SQL AS $$
INSERT INTO films VALUES (0,fn,gnrs,years,cost_for);
$$;

CREATE OR REPLACE PROCEDURE update_films(fn CHARACTER VARYING(30), gnrs CHARACTER VARYING(30),years date, cost_for int)
Language SQL AS $$
UPDATE films SET film_name = fn, genres=gnrs, year_of =years, cost_for_watch=cost_for;
$$;

CREATE OR REPLACE PROCEDURE delete_films(fn CHARACTER VARYING(30), gnrs CHARACTER VARYING(30),years date, cost_for int)
Language SQL AS $$
DELETE FROM films WHERE film_name = fn AND genres=gnrs AND year_of =years AND cost_for_watch=cost_for;
$$;
```

```
        CREATE OR REPLACE PROCEDURE insert_purchase(us_id int, or_id
int,flm_id int, p_s bool)
        Language SQL AS $$
        INSERT INTO purchase VALUES (0,us_id,or_id,flm_id,p_s);
        $$;

        CREATE OR REPLACE PROCEDURE update_purchase(us_id int, or_id
int,flm_id int, p_s bool)
        Language SQL AS $$
        UPDATE purchase SET userid = us_id, orderid=or_id, filmid =flm_id,
purchase_status=p_s;
        $$;

        CREATE OR REPLACE PROCEDURE delete_purchase(us_id int, or_id
int,flm_id int, p_s bool)
        Language SQL AS $$
        DELETE FROM purchase WHERE userid = us_id AND orderid=or_id
AND filmid =flm_id AND purchase_status=p_s;
        $$;

        CREATE OR REPLACE PROCEDURE insert_order_det(now date, future
date)
        Language SQL AS $$
        INSERT INTO order_details(date_of_issue,date_of_return) VALUES
(now,future);
        $$;

        CREATE OR REPLACE PROCEDURE delete_order_det(now date, future
date)
        Language SQL AS $$
```

```
	DELETE FROM order_details WHERE date_of_issue=now AND
date_of_return= future;
	$$;

	CREATE OR REPLACE PROCEDURE update_order_det(now_old date,
future_old date,now_new date, future_new date )
	Language SQL AS $$
	UPDATE order_details SET date_of_issue=now_new ,
date_of_return=future_new  WHERE date_of_issue=now_old AND
date_of_return= future_old;
	$$;
```

3.11 Функция, состоящая из нескольких операций в виде единой транзакции, которая при определенных условиях может быть откатана

```
	CREATE OR REPLACE FUNCTION check_new_order(fn CHARACTER
VARYING(30),lnm CHARACTER VARYING(30),flm_name CHARACTER
VARYING(30)) RETURNS VOID AS $$
	DECLARE
		ms int[];
		res int;
	BEGIN
	INSERT INTO Purchase(userid,filmid) VALUES((SELECT id FROM
Users WHERE first_name=fn AND last_name=lnm),(SELECT film_id FROM
Films WHERE flim_name=flm_name));

	ms= ret_cost_for_user_ms(fn,lnm);
	res=SUM(ms);

	IF res>500 THEN
```

```
		COMMIT;
	ELSE
		ROLLBACK;
	END IF;

	END;
	$$ LANGUAGE plpgsql;
```

3.12 Курсор на обновление данных

```
	CREATE OR REPLACE FUNCTION ret_cost_for_user(fn CHARACTER
VARYING(30), lnm CHARACTER VARYING(30)) RETURNS int AS $$
	DECLARE full_cost int :=0; fn CHARACTER VARYING(30); lnm
CHARACTER VARYING(30); c_f_w int;
	c_user CURSOR FOR SELECT first_name,last_name,cost_for_watch
FROM full_purchase;
	BEGIN
	OPEN c_user;
		LOOP
			FETCH c_user INTO fn,lnm,c_f_w;
			EXIT WHEN NOT FOUND;
			full_cost = full_cost+ c_f_w;
		END LOOP;
	CLOSE c_user;
	RETURN full_cost;
	END;
	$$ LANGUAGE plpgsql;
```

3.13 Скалярные и векторные функции

```
        CREATE OR REPLACE FUNCTION ret_cost_for_user(fn CHARACTER
VARYING(30), lnm CHARACTER VARYING(30)) RETURNS int AS $$
        DECLARE full_cost int :=0; fn CHARACTER VARYING(30); lnm
CHARACTER VARYING(30); c_f_w int;
        c_user CURSOR FOR SELECT first_name,last_name,cost_for_watch
FROM full_purchase;
        BEGIN
        OPEN c_user;
                LOOP
                        FETCH c_user INTO fn,lnm,c_f_w;
                        EXIT WHEN NOT FOUND;
                        full_cost = full_cost+ c_f_w;
                END LOOP;
        CLOSE c_user;
        RETURN full_cost;
        END;
        $$ LANGUAGE plpgsql;

        CREATE OR REPLACE FUNCTION ret_cost_for_user_ms (fn
CHARACTER VARYING(30), lnm CHARACTER VARYING(30)) RETURNS
int[] AS $$
        DECLARE ms_cost int[]; fn CHARACTER VARYING(30); lnm
CHARACTER VARYING(30); c_f_w int;  i int := 0;
        c_user CURSOR FOR SELECT first_name,last_name,cost_for_watch
FROM full_purchase;
        BEGIN

        OPEN c_user;
                LOOP
                FETCH c_user INTO fn,lnm,c_f_w;
                EXIT WHEN NOT FOUND;
                ms_cost[i] = c_f_w;
                i = i + 1;
            END LOOP;
    CLOSE c_user;
    RETURN ms_cost;
    END;
    $$ LANGUAGE plpgsql;
```

3.14 Распределение прав пользователей

Права доступа разграничиваются на 2 пользователей: bd_admin, который имеет полный доступ ко всем таблицам и worker, который имеет права записывать только в Purchase и Users, а также читать со всех таблиц. Ниже приведен фрагмент кода для создания данных пользователей:

```
    CREATE USER bd_admin PASSWORD 'main_user';
    GRANT ALL PRIVILEGES ON TABLE users,films,purchase,order_details
TO bd_admin;
    CREATE USER worker PASSWORD 'so_hard';
    GRANT SELECT ON TABLE users,films,order_details,purchase TO
worker;
    GRANT INSERT, UPDATE ON TABLE purchase,users TO worker;
    GRANT USAGE ON SEQUENCE
films_film_id_seq,order_details_order_id_seq,purchase_purchase_id_seq,users_id
_seq TO bd_admin;
    GRANT USAGE ON SEQUENCE
films_film_id_seq,order_details_order_id_seq,purchase_purchase_id_seq,users_id
_seq TO worker;
```

ЗАКЛЮЧЕНИЕ

Проделав данную лабораторную работу, я освоил основные навыки работы с СУБД PostgresSQL, а также научился работать с библиотекой Qt на C++ и ее аналогом PySide6 на языке Python.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- https://postgrespro.ru/
- https://doc.qt.io/qtforpython-6/gettingstarted/porting_from2.html
- https://habr.com/ru/companies/skillfactory/articles/599599/

Приложение № 1

Распечатка SQL скрипта всех созданных объектов БД:

```
CREATE TABLE Users (
        Id Serial PRIMARY KEY,
        First_name CHARACTER VARYING(30),
        Last_name CHARACTER VARYING(30),
        Email CHARACTER VARYING(30),
        Age int
);
CREATE TABLE IF NOT EXISTS Films (
        Film_id Serial PRIMARY KEY,
        Film_name CHARACTER VARYING(50),
        Genres CHARACTER VARYING(100),
        Year_of date,
        Cost_for_watch int2
);
CREATE TABLE IF NOT EXISTS Order_Details (
        Order_id Serial PRIMARY KEY,
        Date_of_issue date,
        Date_of_return date
);
CREATE TABLE IF NOT EXISTS Purchase (
        Purchase_id Serial PRIMARY KEY,
        UserId int,
        OrderId int,
        FilmId int,
        Purchase_status bool,
        Foreign Key (UserId) REFERENCES Users (Id) ON DELETE
CASCADE ON UPDATE CASCADE,
            Foreign Key (OrderId) REFERENCES Order_Details (Order_id) ON
DELETE CASCADE ON UPDATE CASCADE,
            Purchase_status bool,
            Foreign Key (FilmId) REFERENCES Films (Film_id) ON DELETE
CASCADE ON UPDATE CASCADE
      );
      CREATE INDEX users_pkey_cluster_idx ON Users USING btree (id);
      CREATE INDEX films_pkey_cluster_idx ON Films USING btree
(Film_id);
      CREATE INDEX purchase_pkey_cluster_idx ON Purchase USING hash
(Purchase_id);
```

```
      CREATE USER bd_admin PASSWORD 'main_user';
      GRANT ALL PRIVILEGES ON TABLE users,films,purchase,order_details
TO bd_admin;
      CREATE USER worker PASSWORD 'so_hard';
      GRANT SELECT ON TABLE users,films,order_details,purchase TO
worker;
      GRANT INSERT,UPDATE ON TABLE purchase,users TO worker;
```

```
      CREATE VIEW full_purchase AS SELECT usr.first_name,
        usr.last_name,
        f.film_name,
        f.cost_for_watch
```

```
          FROM purchase pr
            JOIN users usr ON pr.userid = usr.id
            JOIN films f ON pr.filmid = f.film_id;

CREATE OR REPLACE FUNCTION update_full_purchase_view() RETURNS
TRIGGER AS $$
   BEGIN
        IF (TG_OP = 'DELETE') THEN
                DELETE FROM films WHERE film_name = OLD.film_name;
                DELETE FROM users WHERE first_name = OLD.first_name AND
last_name=OLD.last_name;
                IF NOT FOUND THEN RETURN NULL; END IF;
                RETURN OLD;
        ELSIF (TG_OP = 'UPDATE') THEN
                UPDATE films SET cost_for_watch = NEW.cost_for_watch WHERE
film_name = OLD.film_name;
                IF NOT FOUND THEN RETURN NULL; END IF;
                RETURN NEW;
        ELSIF (TG_OP = 'INSERT') THEN
                INSERT INTO films (film_name,cost_for_watch) VALUES
(NEW.film_name,NEW.cost_for_watch);
                IF NOT FOUND THEN RETURN NULL; END IF;
                RETURN NEW;
        END IF;
        END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER update_full_purchase_trigger
INSTEAD OF INSERT OR UPDATE OR DELETE ON full_purchase
   FOR EACH ROW EXECUTE PROCEDURE update_full_purchase_view();
```

```
        CREATE OR REPLACE PROCEDURE insert_users(fn CHARACTER
VARYING(30), lnm CHARACTER VARYING(30),em CHARACTER
VARYING(30),agge int)
        Language SQL AS $$
        INSERT INTO Users VALUES (0,fn,lnm,em,agge);
        $$;

        CREATE OR REPLACE PROCEDURE update_users(fn CHARACTER
VARYING(30), lnm CHARACTER VARYING(30),em CHARACTER
VARYING(30),agge int)
        Language SQL AS $$
        UPDATE Users SET first_name = fn, last_name=lnm, email =em,
age=agge;
        $$;

        CREATE OR REPLACE PROCEDURE delete_users(fn CHARACTER
VARYING(30), lnm CHARACTER VARYING(30),em CHARACTER
VARYING(30),agge int)
        Language SQL AS $$
        DELETE FROM Users WHERE first_name = fn AND last_name=lnm AND
email =em AND age=agge;
        $$;

        CREATE OR REPLACE PROCEDURE insert_films(fn CHARACTER
VARYING(30), gnrs CHARACTER VARYING(30),years date, cost_for int)
        Language SQL AS $$
```

```sql
INSERT INTO films VALUES (0,fn,gnrs,years,cost_for);
$$;

CREATE OR REPLACE PROCEDURE update_films(fn CHARACTER VARYING(30), gnrs CHARACTER VARYING(30),years date, cost_for int)
Language SQL AS $$
UPDATE films SET film_name = fn, genres=gnrs, year_of =years, cost_for_watch=cost_for;
$$;

CREATE OR REPLACE PROCEDURE delete_films(fn CHARACTER VARYING(30), gnrs CHARACTER VARYING(30),years date, cost_for int)
Language SQL AS $$
DELETE FROM films WHERE film_name = fn AND genres=gnrs AND year_of =years AND cost_for_watch=cost_for;
$$;



CREATE OR REPLACE PROCEDURE insert_purchase(us_id int, or_id int,flm_id int, p_s bool)
Language SQL AS $$
INSERT INTO purchase VALUES (0,us_id,or_id,flm_id,p_s);
$$;

CREATE OR REPLACE PROCEDURE update_purchase(us_id int, or_id int,flm_id int, p_s bool)
Language SQL AS $$
UPDATE purchase SET userid = us_id, orderid=or_id, filmid =flm_id, purchase_status=p_s;
$$;
```

```
CREATE OR REPLACE PROCEDURE delete_purchase(us_id int, or_id int,flm_id int, p_s bool)
Language SQL AS $$
DELETE FROM purchase WHERE userid = us_id AND orderid=or_id AND filmid =flm_id AND purchase_status=p_s;
$$;


CREATE OR REPLACE PROCEDURE insert_order_det(now date, future date)
Language SQL AS $$
INSERT INTO order_details(date_of_issue,date_of_return) VALUES (now,future);
$$;

CREATE OR REPLACE PROCEDURE delete_order_det(now date, future date)
Language SQL AS $$
DELETE FROM order_details WHERE date_of_issue=now AND date_of_return= future;
$$;

CREATE OR REPLACE PROCEDURE update_order_det(now_old date, future_old date,now_new date, future_new date )
Language SQL AS $$
UPDATE order_details SET date_of_issue=now_new , date_of_return=future_new  WHERE date_of_issue=now_old AND date_of_return= future_old;
$$;
```

```
        CREATE OR REPLACE FUNCTION ins_purchase_status_func()
RETURNS TRIGGER AS $$
          BEGIN
            NEW.purchase_status := 0;
                INSERT INTO order_details(date_of_issue, date_of_return)
VALUES (current_date, current_date+7);
                NEW.orderid  = currval('order_details_order_id_seq');
            RETURN NEW;
          END;
        $$ LANGUAGE plpgsql;

        CREATE OR REPLACE TRIGGER ins_purchase_status_trigger
        BEFORE INSERT ON purchase
        FOR EACH ROW
        EXECUTE PROCEDURE ins_purchase_status_func();
```

```
        CREATE OR REPLACE FUNCTION del_purchase_func() RETURNS
TRIGGER AS $$
          BEGIN
                DELETE FROM order_details WHERE
order_details.order_id=OLD.orderid;
                RETURN OLD;
          END;
        $$ LANGUAGE plpgsql;
        CREATE OR REPLACE TRIGGER del_purchase_trigger
        AFTER DELETE ON purchase
        FOR EACH ROW
        EXECUTE PROCEDURE del_purchase_func();
```

```
CREATE OR REPLACE FUNCTION upd_purchase_func() RETURNS
TRIGGER AS $$
  BEGIN
         NEW.purchase_status := 1;
         RETURN NEW;
  END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER upd_purchase_trigger
BEFORE UPDATE ON purchase
FOR EACH ROW
EXECUTE PROCEDURE upd_purchase_func();
```

```
        CREATE OR REPLACE FUNCTION ret_cost_for_user(fnn CHARACTER
VARYING(30), lnmm CHARACTER VARYING(30)) RETURNS int AS $$
        DECLARE full_cost int :=0; fn CHARACTER VARYING(30); lnm
CHARACTER VARYING(30); c_f_w int;
        c_user CURSOR FOR SELECT first_name,last_name,cost_for_watch
FROM full_purchase WHERE purchase_status = false AND first_name=fnn AND
last_name=lnmm;
        BEGIN
        OPEN c_user;
        LOOP
            FETCH c_user INTO fn,lnm,c_f_w;
            EXIT WHEN NOT FOUND;
            full_cost = full_cost+ c_f_w;
        END LOOP;
    CLOSE c_user;
    RETURN full_cost;
    END;
    $$ LANGUAGE plpgsql;

    CREATE OR REPLACE FUNCTION ret_cost_for_user_ms (fnn
CHARACTER VARYING(30), lnmm CHARACTER VARYING(30)) RETURNS
int[] AS $$
    DECLARE ms_cost int[]; fn CHARACTER VARYING(30); lnm
CHARACTER VARYING(30); c_f_w int;  i int := 0;
    c_user CURSOR FOR SELECT first_name,last_name,cost_for_watch
FROM full_purchase WHERE purchase_status = false AND first_name=fnn AND
last_name=lnmm;
    BEGIN

    OPEN c_user;
        LOOP
            FETCH c_user INTO fn,lnm,c_f_w;
            EXIT WHEN NOT FOUND;
            ms_cost[i] = c_f_w;
            i = i + 1;
        END LOOP;
    CLOSE c_user;
    RETURN ms_cost;
    END;
```

```
$$ LANGUAGE plpgsql;
```

Приложение № 2

Листинг исходного кода клиента:

```
      import sys
from PySide6.QtCore import Qt
from PySide6.QtWidgets import *
from Main_w import Ui_MainWindow
from conn import Ui_Dialog
from add_purchase import Ui_Form
from Users_widget import Ui_Form as f2
import psycopg2


class Change_Users(QDialog):
    def __init__(self):
        super().__init__()
        self.ui=f2()
        self.ui.setupUi(self)
        #self.ui.tableView.

class AddPurchase(QDialog):
    def __init__(self):
        super().__init__()
        self.ui=Ui_Form()
        self.ui.setupUi(self)
        self.ui.accept_btn.clicked.connect(self.add_film_in_collection)

    def add_film_in_collection(self):
        try:
            conn = psycopg2.connect(host="127.0.0.1", user="bd_admin",
password="main_user", database="test")
            with conn.cursor() as curs:
                curs.execute("INSERT INTO purchase VALUES (0,1,1,1);")

                #res = curs.fetchall()
                #print(res, end=' ')
                #ms = [i[0] for i in res]
        except Exception as ex:
            print(ex)
        finally:
            print()
            if conn:
                conn.close()
                self.close()


class ConnDialog(QDialog):
    def __init__(self):
        super().__init__()
        self.ui = Ui_Dialog()
        self.ui.setupUi(self)
        self.ui.pushButton.clicked.connect(self.collect_info)
        self.conn_status=False

    def collect_info(self):
        self.address=self.ui.lineEdit.text()
        self.bdname=self.ui.lineEdit_2.text()
        self.login = self.ui.lineEdit_3.text()
        self.paswd = self.ui.lineEdit_4.text()
        print(self.address)
        self.conn_db()


    def conn_db(self):
        try:
            conn = psycopg2.connect(host="127.0.0.1", user=self.login,
password=self.paswd, database=self.bdname)
            with conn.cursor() as curs:
                curs.execute("Select version();")
                print(curs.fetchone())
        except Exception as ex:
            print(ex)
        finally:
            if conn:
                conn.close()
                print("closed")
                self.close()
                self.conn_status=True

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.cd = None
        self.purchase_windows=None
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)
        self.ui.connect_2.triggered.connect(self.open_dia)
        self.ui.action.triggered.connect(self.check_conn_data)
        self.ui.give_film_btn.clicked.connect(self.open_AddPurchase)


    def open_dia(self):
        if self.cd is None:
            self.cd = ConnDialog()
        self.cd.show()

    def check_conn_data(self):
        print(self.cd.address) #данные подключения

    def open_AddPurchase(self):
        if self.purchase_windows is None:
            self.purchase_windows = AddPurchase()
            conn=None
            try:
                conn = psycopg2.connect(host="127.0.0.1", user=self.cd.login,
password=self.cd.paswd, database=self.cd.bdname)
                with conn.cursor() as curs:
                    curs.execute("SELECT film_name FROM Films;")
                    res = curs.fetchall()
                    self.purchase_windows.film_names = [i[0] for i in res]
            except Exception as ex:
                print(ex)
            finally:
                if conn:
                    conn.close()
                    print("closed")
                    self.conn_status = True

self.purchase_windows.ui.ch_film.addItems(self.purchase_windows.film_names)
        self.purchase_windows.show()


if __name__ == '__main__':
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    app.exec()
```

(Это клиент без доп окон, в которых и работаю запросы, полная версия шире, если будет надо - прикреплю), (сложные запросы в доп окнах, их скрипты оставлю ниже). Пожалуйста, допустите, при защите все покажу.

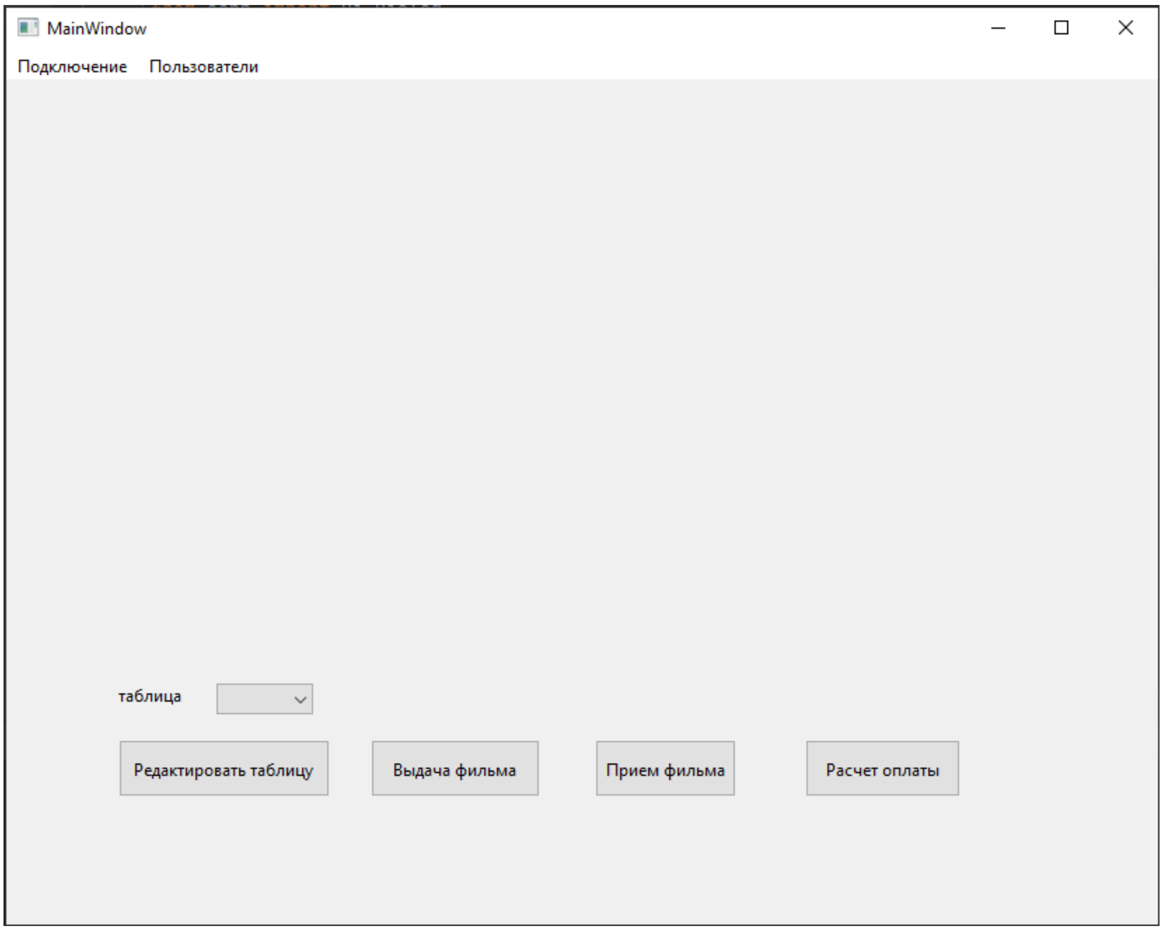


Рисунок 1 – Главная форма

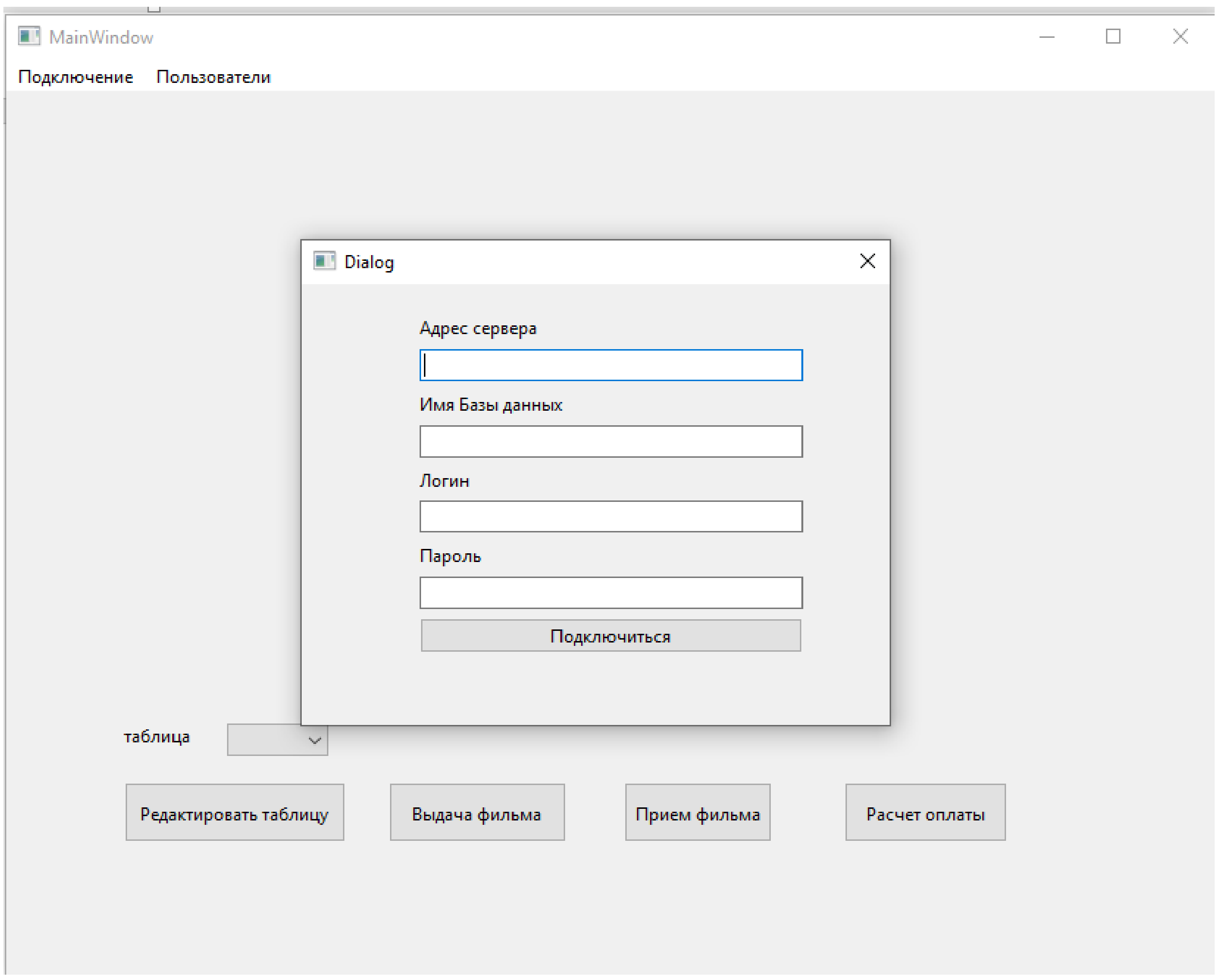


Рисунок 2 – Форма входа для пользователя

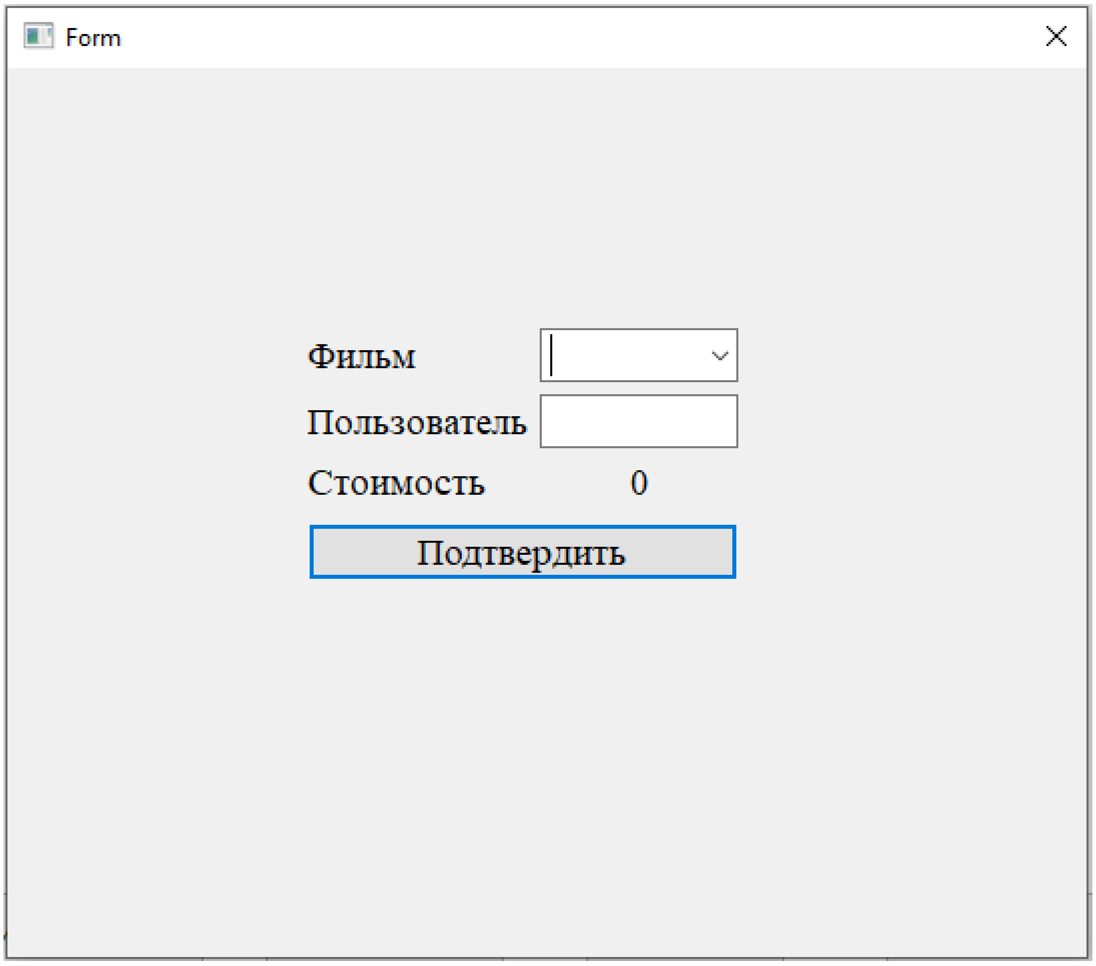
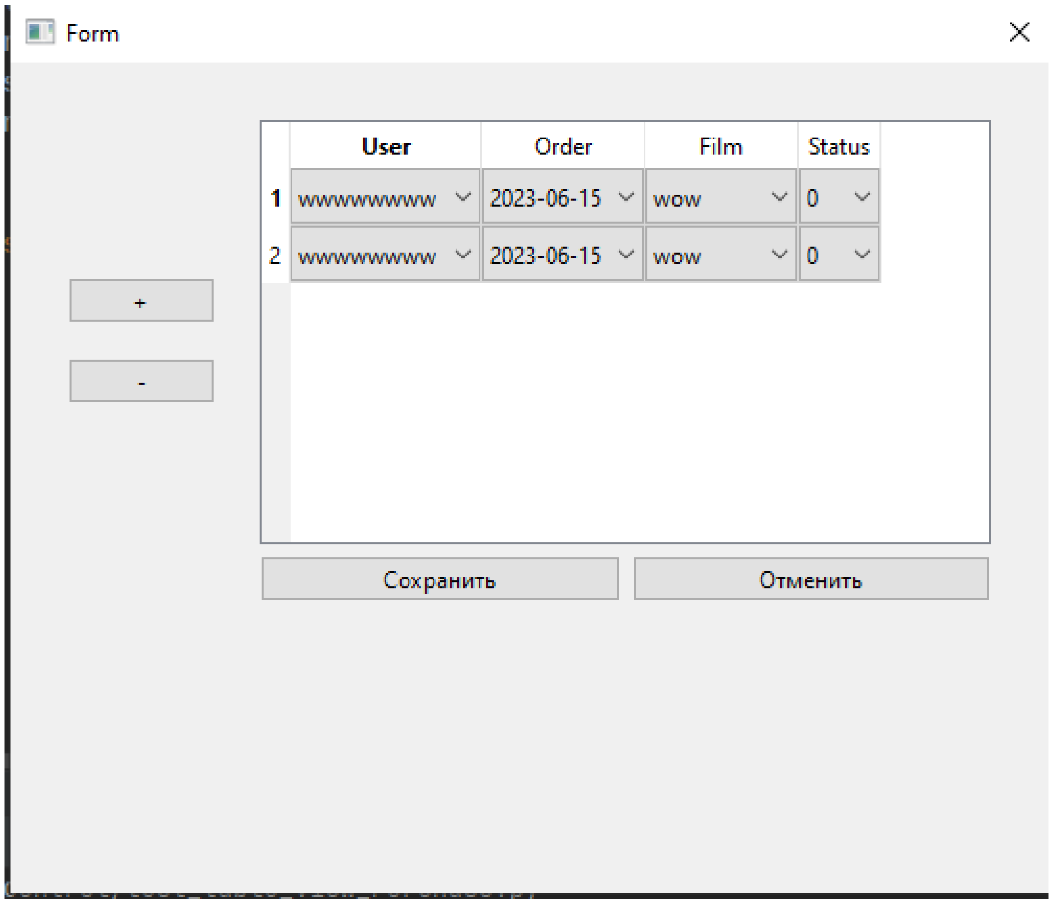
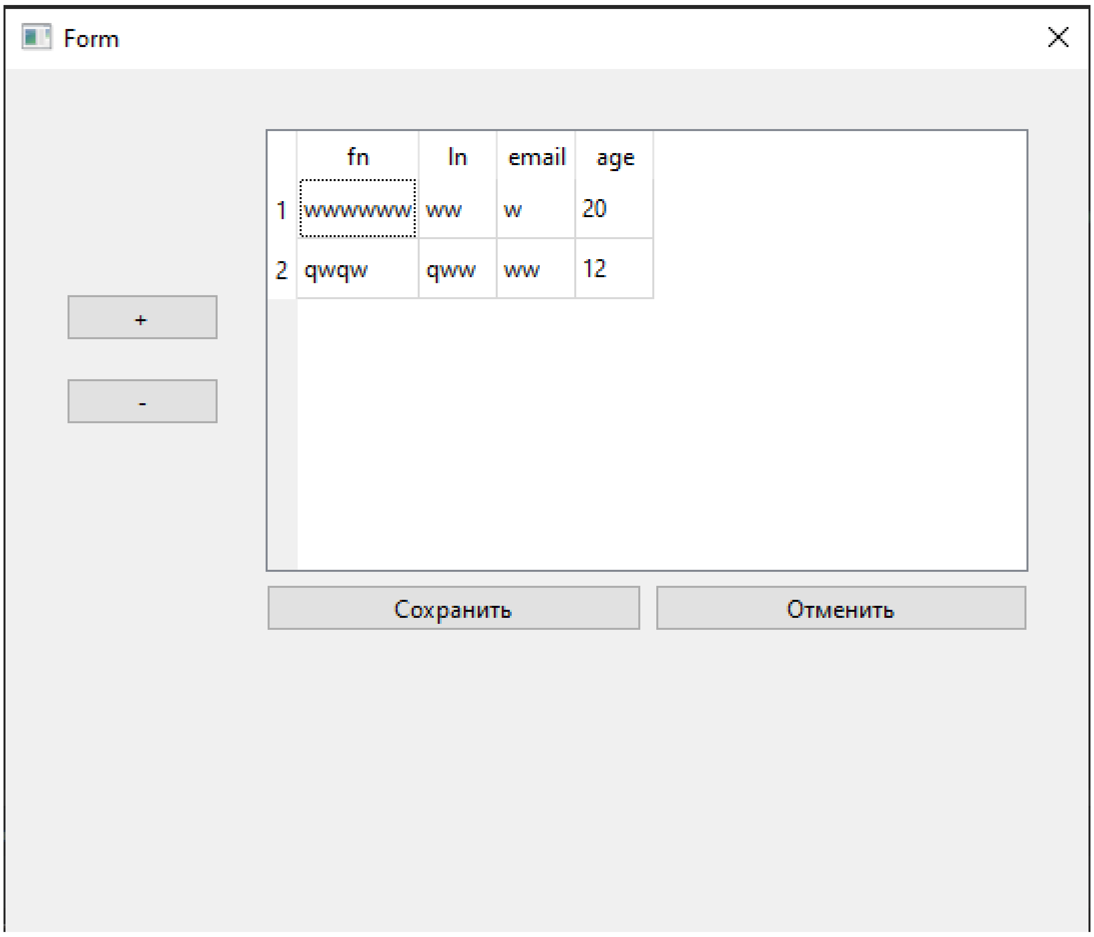


Рисунок 3 – Форма оформления аренды фильма



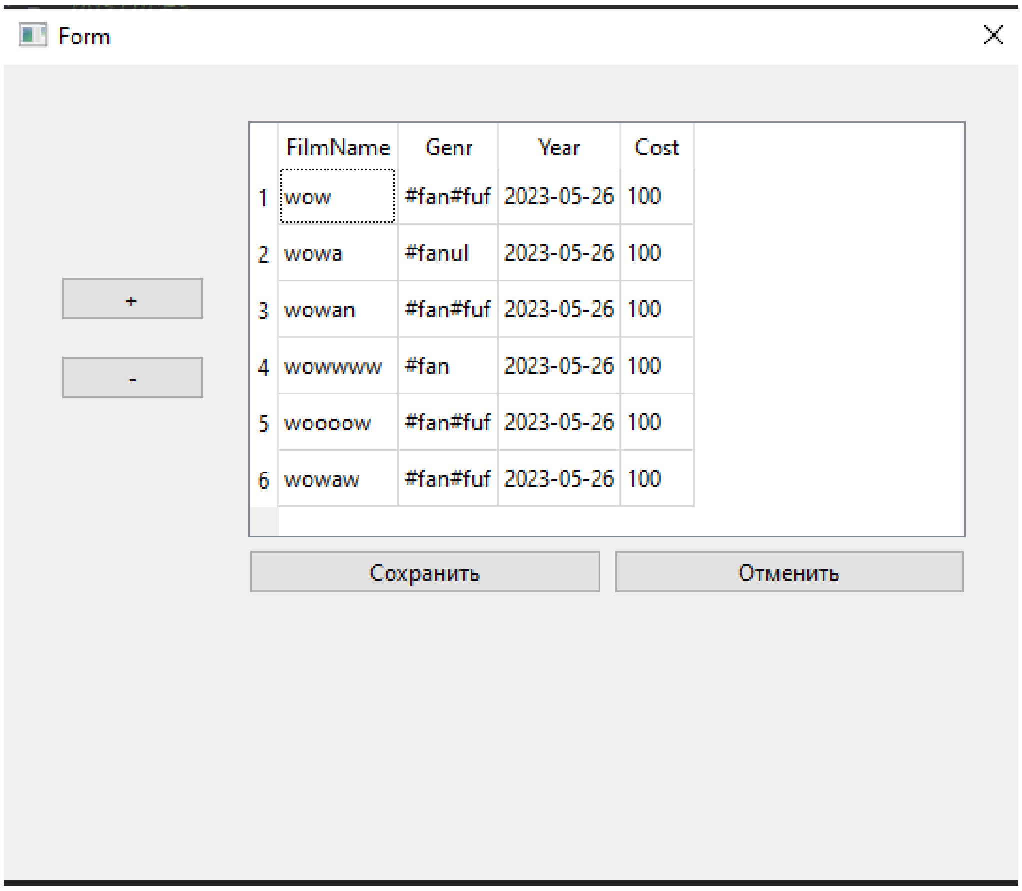
	User	Order	Film	Status
1	wwwwwwwww	2023-06-15	wow	0
2	wwwwwwwww	2023-06-15	wow	0

Рисунок 4 -Форма работы с таблицей Purchase



	fn	ln	email	age
1	wwwwwww	ww	w	20
2	qwqw	qww	ww	12

Рисунок 5 - Форма работы с таблицей Users



	FilmName	Genr	Year	Cost
1	wow	#fan#fuf	2023-05-26	100
2	wowa	#fanul	2023-05-26	100
3	wowan	#fan#fuf	2023-05-26	100
4	wowwww	#fan	2023-05-26	100
5	woooow	#fan#fuf	2023-05-26	100
6	wowaw	#fan#fuf	2023-05-26	100

Рисунок 6 - Форма работы с таблицей Films