Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

Национальный исследовательский университет «МЭИ»

Типовой расчёт

по курсу «**Базы данных**»

Симуляция базы данных сети автопарков

Выполнила:

студент группы А-12-19

Иванова С. С.

Москва

2020

Содержание:

1. **Хранящиеся данные:**

- Список всех водителей: идентификатор, фио, права, суммарная выплата, статус. (свободен, совершает рейс)

- Список всех автомобилей: идентификатор, марка, грузоподъёмность, идентификатор водителя которому назначен, статус. (доступен, недоступен)

- Список всех рейсов: идентификатор, идентификатор водителя, идентификатор транспортного средства, массу груза, пункт назначения, расстояние и цену.

Примеры:

Таблица 1. Пример списка водителей

ID	**ФИО**	**ПРАВА**	**НАКОПЛЕНИЕ**	**СТАТУС**
1	Алексей А. Е.	5604491942	250.30	свободен
2	Иван И. И.	4895321942	500.63	совершает рейс

Таблица 2. Пример списка автомобилей

ID	МАРКА	ГРУЗОПОДЪЁМНОСТЬ	ID ВОДИТЕЛЯ	**СТАТУС**
1	Лада	200	2	недоступен
2	Жигули	10000000	1	доступен

Таблица 3. Пример списка рейсов

ID	**ID ВОДИТЕЛЯ**	**ID Т.С.**	**МАССА**	**НАЗНАЧЕНИЕ**	**РАССТОЯНИЕ**	**ЦЕНА**
1	1	2	200	Москва	200	100.50
2	2	1	100	Санкт Петербург	10	32.10

2. **Группы пользователей:**
    - Администраторы

      Может изменять данные во всех отделений сети, а также может добавлять новых менаджеров.

    - Менаджеры

      Могут изменять данные только во своём отделении.

3. **Порядок работы**
    - Менаджер:

Авторизуется в системе с помощью выданного ему администратором логина и пароля (рис. 1.). При самой авторизации система автоматический определяет каком отделений он принадлежит. Дальше ему показывается интерфейс с помощью которого он может выбрать какие данные будет изменять (рис. 2.). Пример добавления водителя показан на рисунке 3. Таблица перевозок разбита на три таблицы. Первая таблица перевозок хранить новые перевозки которые только должны быть выполнены. Вторая таблица перевозок хранить перевозки которые в процессе. Третья таблица перевозок хранить законченные перевозки. Добавлять и удалять данные можно только в первую таблицу. После того как перевозка началась, её удалить или изменить нельзя, разрешено только перевести её в состояние «закончена». После окончания перевозки, деньги автоматический начисляются перевозчику который совершил эту перевозку. Сумма начисления соответствует цене указанной в перевозке. Пример добавления, переведения в состояние активно и завершения перевозки показан на рисунках 4, 5 и 6, соответственно. Начисление денег на счёт перевозчика

показано на рисунке 7. Перевозку можно добавить только если в системе уже существует хотя бы один водитель и хотя бы одно транспортное средство. Новые транспортные средства можно добавлять только если существует водитель к которому нет привязанного транспортного средства. Если к выбранному водителю уже привязано транспортное средство, то при добавлении нового т.с. этому же водителю произойдёт обновление уже существующего т.с. Интерфейс и пример добавления, обновления и удаления т.с. иллюстрирован на рисунках 8, 9 и 10, соответственно.

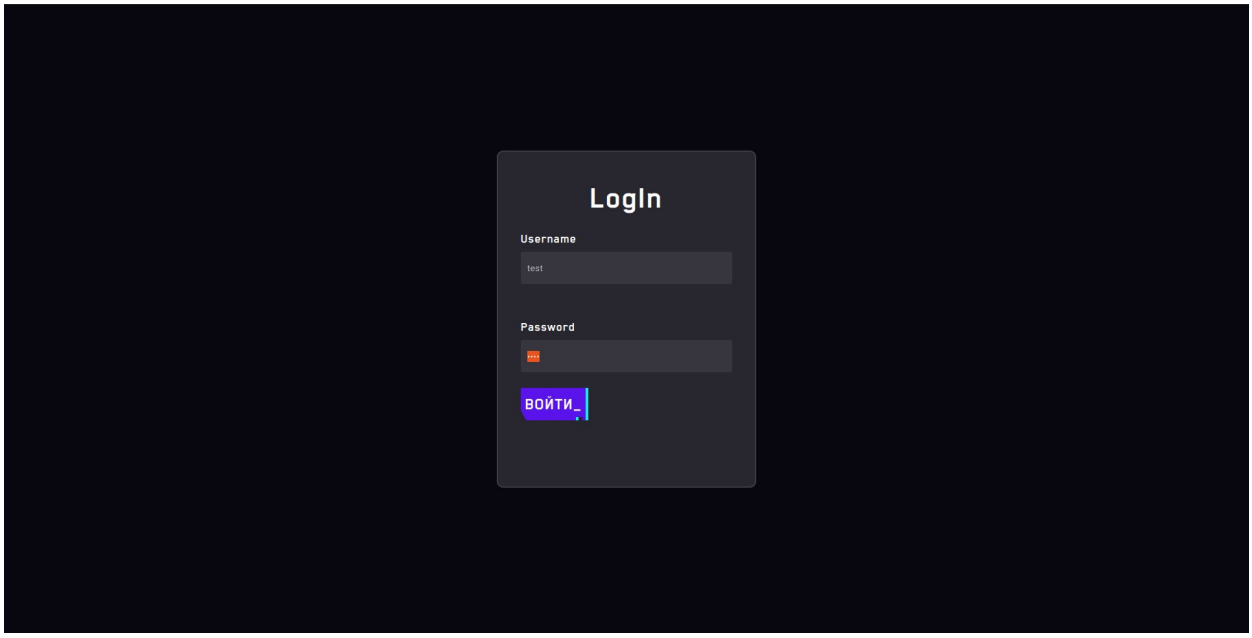

рис. 1. Форма авторизации

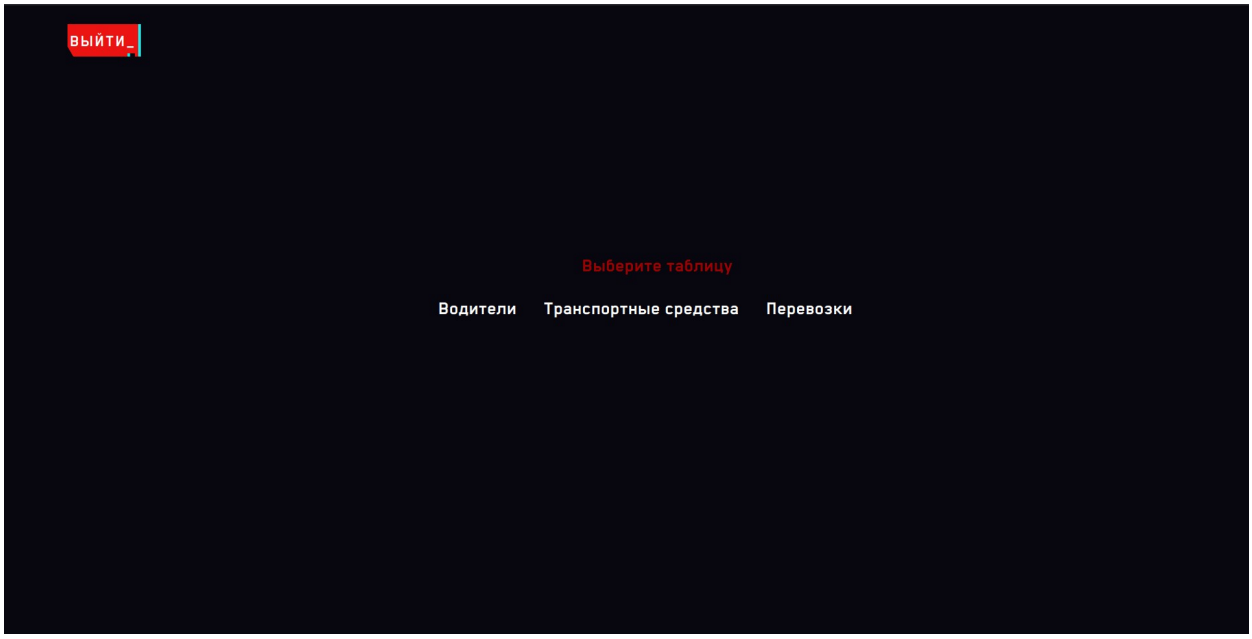

рис. 2. Интерфейс выбора таблицы

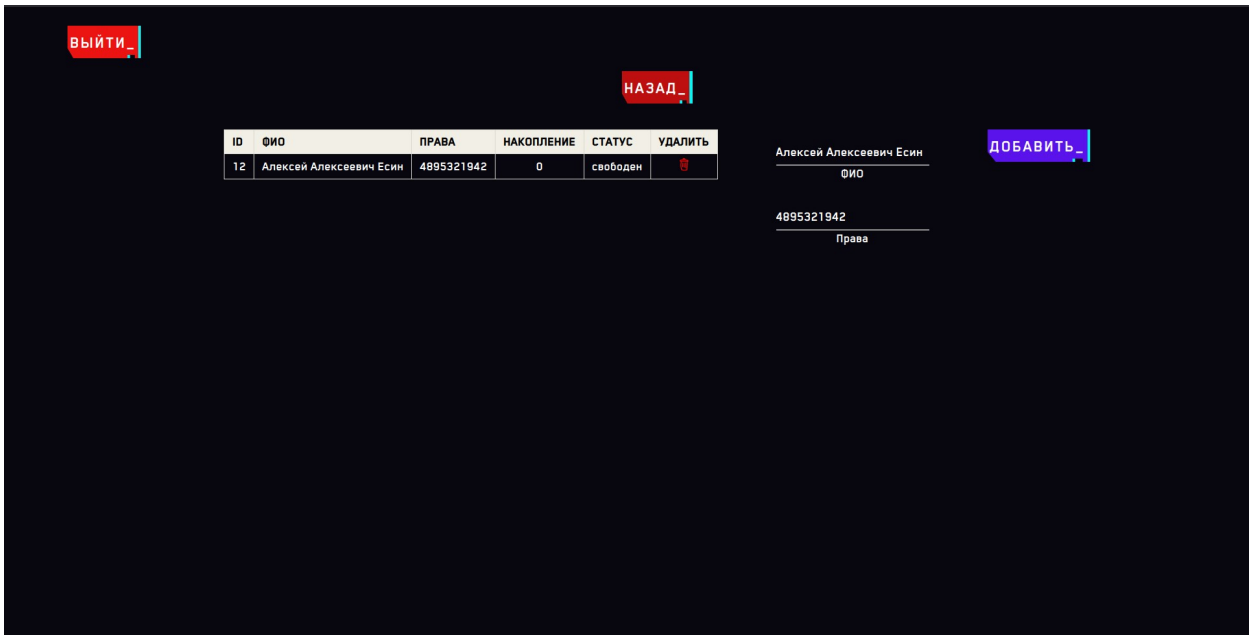

рис. 3. Пример добавления водителя

рис. 4. Пример добавления перевозки

рис. 5. Пример переведения перевозки в активное состояние

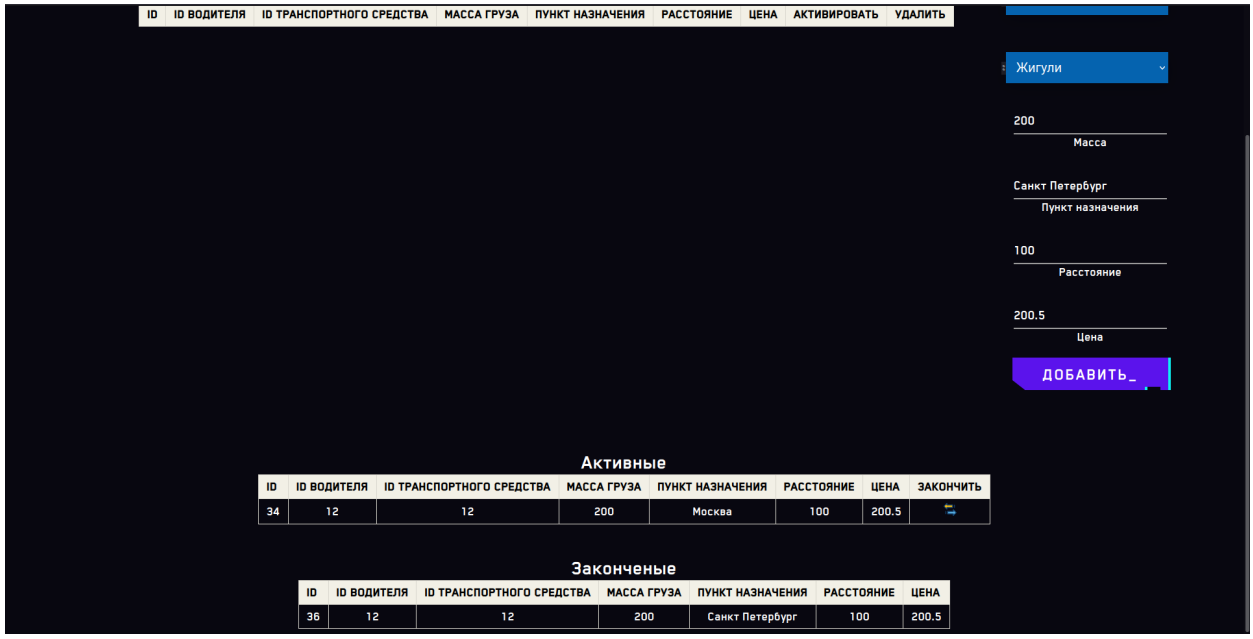

рис. 6. Пример завершения перевозки

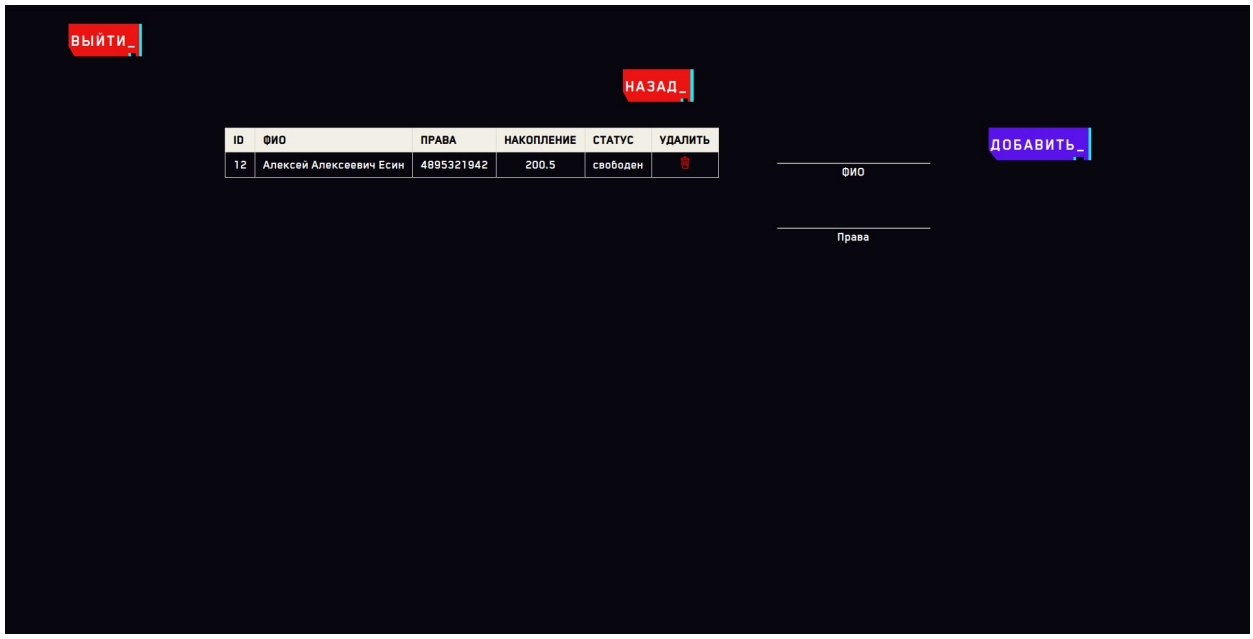

рис. 7. Пример зачисления денег на счёт перевозчика после окончания перевозки

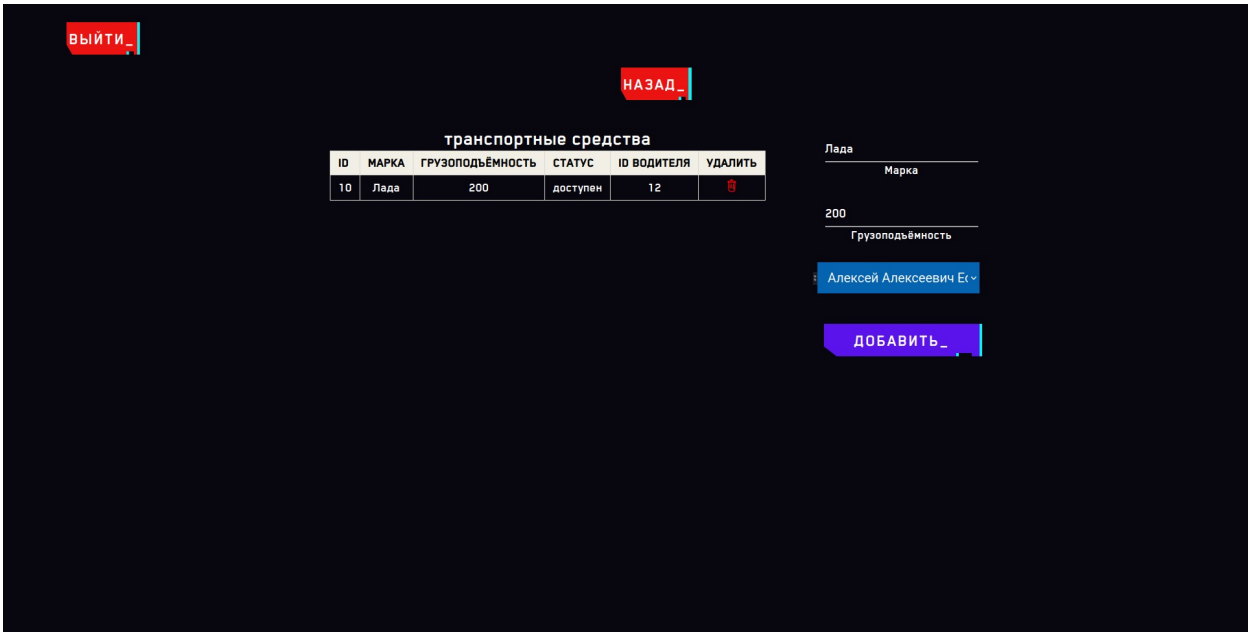

рис. 8. Пример добавления транспортного средства

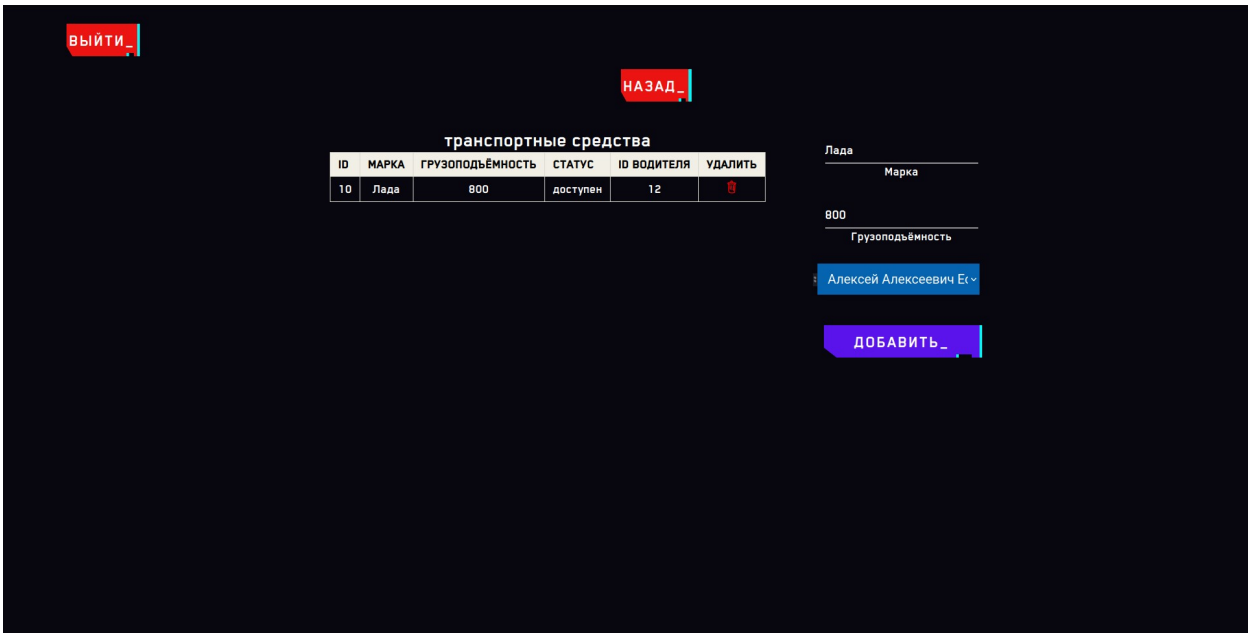

рис. 9. Пример обновления транспортного средства (грузоподъёмность)

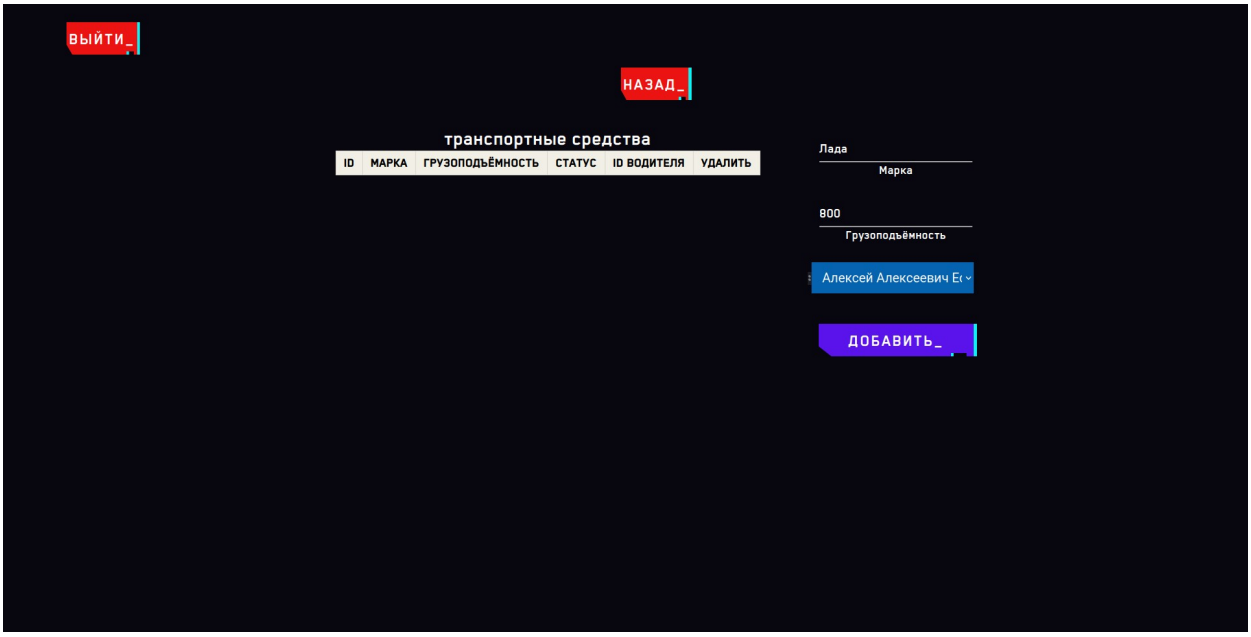

рис. 10. Пример удаления транспортного средства

- Администратор:

Отличие администратора от менаджера в том что он имеет доступ ко всем отделениям. Чтобы добиться этого, после авторизации администратора, ему предоставлен выбор отделения в котором он хочет менять данные (рис. 11.). Кроме этого, после выбора отделения, помимо выбора талицы ему даётся возможность посмотреть все менаджери в данном отделении, а также он может добавить новых менаджеров. Интерфейс добавления менаджеров показан на рисунке 12.

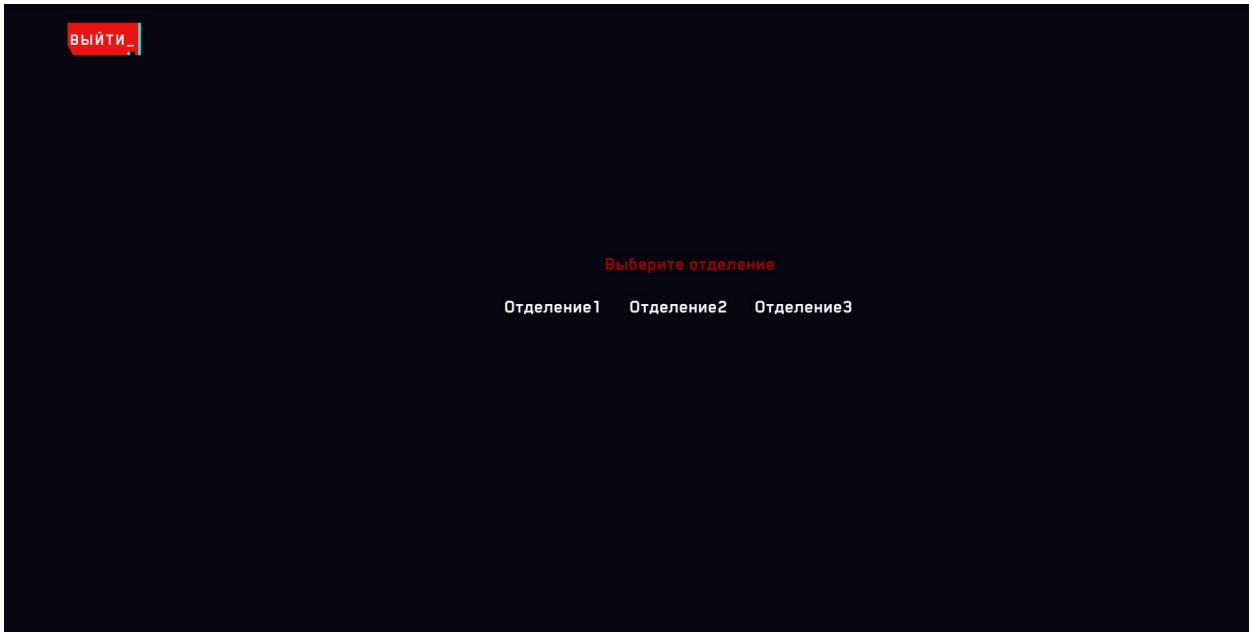

рис. 12. Главное меню администратора

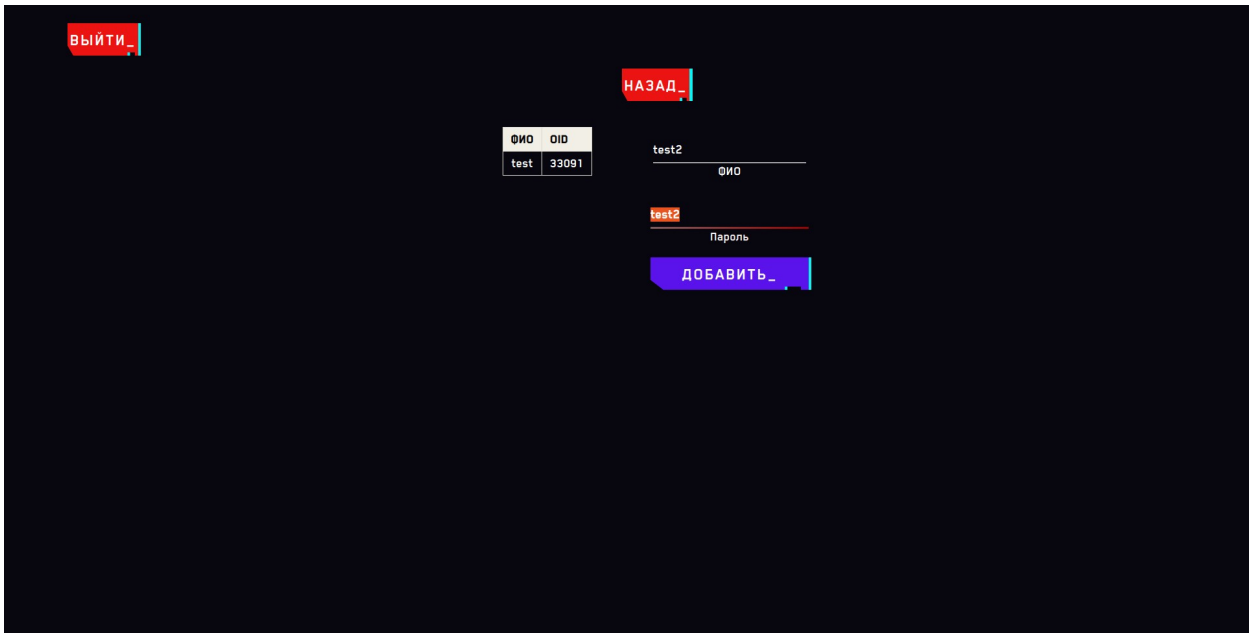

рис. 13. Выбор всех менаджеров в данном отделении и добавление нового

3. Концептуальное проектирование: ER - диаграмма. Таблицы, индексы, ключи, ограничения.

ER - диаграмма показана на рисунке 14. Так как нам понадобилось сделать симуляцию нескольких отделений, вместо создания несколько баз данных использованы схемы. ER диаграмма показывает только одну так как остальные идентичны.

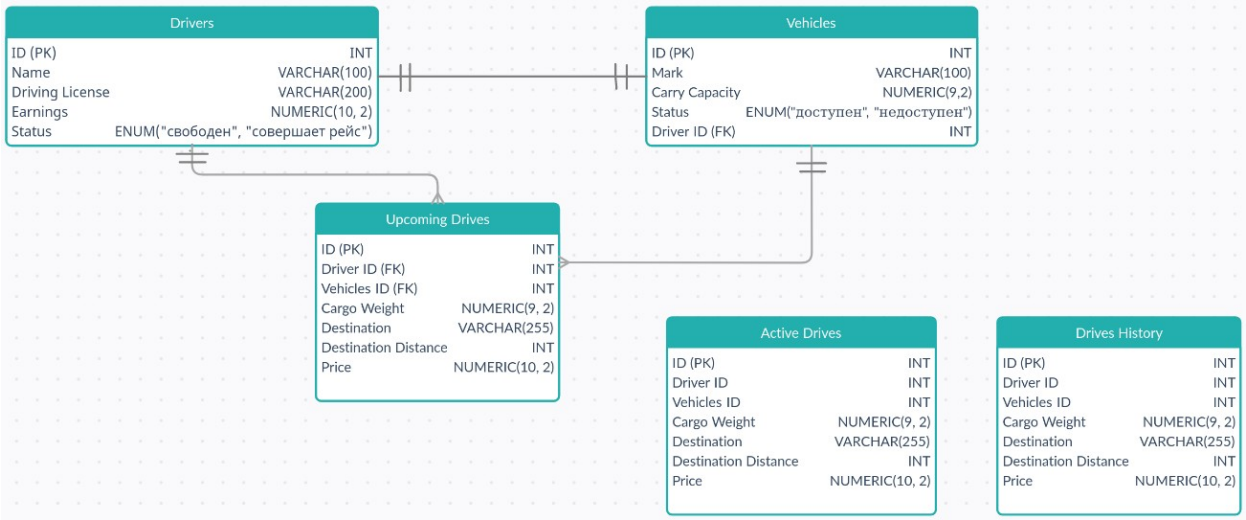

рис. 14. ER-диаграмма одной схемы

Таблицы:

- Нам понадобилось 5 таблиц. Две из которых «промежуточные» таблицы («Upcoming Drives» и «Active Drives»).
- Таблица водителей и транспортных средств хранить модифицируемую информацию.
- Таблица законченных перевозок хранить информацию которую нельзя изменять или удалять.

Индекси:

- Таблица водителей содержит индекс в виде простого числового первичного ключа. Выбор и удаление водителей происходить только с помощью этого идентификатора, поэтому не было нужды индексировать другие поля.

- Остальные таблицы также содержать индексы только на первичном ключе, так как выбор из этих таблиц также происходить только с помощью него. Но если бы расширялся функционал и скажем выбор транспортного средства происходил бы с помощью внешнего ключа водителя, то этот функционал можно было бы ускорить добавив индекс на этот внешний ключ. Аналогично с таблицами перевозок. В таблицах перевозок ещё было бы полезно проиндексировать поле «Vehicle ID».

Ключи:

- Все таблицы содержать первичный ключ в виде простого целочисленного автогенерирующегося идентификатора.

- Таблица транспортных средств содержит внешний ключ связывающий её с таблицей водителей. Это сделано потому что нам нужно было осуществить связь «водитель - т.с.»

- Таблица новых перевозок, «Upcoming Drives», содержит два внешних ключа, один из которых ссылается на идентификатор водителя, второй на идентификатор транспортного средства. Это сделано для удобства при удалении т.с. или водителя, и для

отслеживания кому нужно начислить деньги после окончания перевозки.

- Таблицы активных и законченных перевозок не содержат внешние ключи, так как при удалении т.с. или водителя информация в этих таблицах не должна меняться или удалятся.

Ограничения:

- Все таблицы имеют ограничение первичного ключа который должен быть уникален.
- Таблица водителей содержит уникальный ключ на полю «Driving License» так как эти данные должны быть уникальными для каждого пользователя, а если они повторяются то это значить что у нас два одинаковых водителей в таблице и получаем избыточность данных.
- Таблица транспортных средств содержит уникальный ключ на полю «Driver ID» так как к одному водителю может принадлежать только одно т.с. Это ограничение позволило нам не писать отдельных функции для обновления т.с. Вместо этого в случае возникновения конфликта делаем обновление данных т.с. Второе ограничение касается внешнего ключа. При удалении водителя его т.с. автоматический удаляется.
- Таблица новых перевозок содержит два ограничения на внешних ключах. При удалении т.с. или перевозчика запись в этой таблице автоматический удаляется.

4. Физическая модель БД

- Код создания БД:

```
CREATE DATABASE carpark WITH OWNER = admin ENCODING = 'UTF8' LC_CTYPE = 'ru_RU.utf8' LC_COLLATE =
'ru_RU.utf8' TABLESPACE = pg_default CONNECTION LIMIT = -1 TEMPLATE template0 ;
```

- Код создания таблиц:

```
/    CREATE SCHEMAS SIMULATING MULTIPLE BRANCHES    /
CREATE SCHEMA branch1 ;
CREATE SCHEMA branch2 ;
CREATE SCHEMA branch3 ;

/    ENUMS    /
CREATE TYPE vehicle_status_enum AS ENUM ('доступен', 'недоступен', 'в ремонте');
CREATE TYPE driver_status_enum AS ENUM ('свободен', 'совершает рейс');

/    DRIVERS    /
CREATE TABLE branch1.drivers (
id SERIAL NOT NULL,
fio VARCHAR(           ) NOT NULL,
driving_licence VARCHAR(20 )NOT NULL,
earnings NUMERIC(10, 2) NOT NULL DEFAULT 0,
status driver_status_enum NOT NULL DEFAULT 'свободен',
CONSTRAINT driver_pk PRIMARY KEY(id),
CONSTRAINT driving_licence_unique_key UNIQUE(driving_licence)
);

CREATE TABLE branch2.drivers (
id SERIAL NOT NULL,
fio VARCHAR(           ) NOT NULL,
driving_licence VARCHAR(20 )NOT NULL,
earnings NUMERIC(10, 2) NOT NULL DEFAULT 0,
status driver_status_enum NOT NULL DEFAULT 'свободен',
CONSTRAINT driver_pk PRIMARY KEY(id),
CONSTRAINT driving_licence_unique_key UNIQUE(driving_licence)
);

CREATE TABLE branch3.drivers (
id SERIAL NOT NULL,
fio VARCHAR(           ) NOT NULL,
driving_licence VARCHAR(20 )NOT NULL,
earnings NUMERIC(10, 2) NOT NULL DEFAULT 0,
status driver_status_enum NOT NULL DEFAULT 'свободен',
CONSTRAINT driver_pk PRIMARY KEY(id),
CONSTRAINT driving_licence_unique_key UNIQUE(driving_licence)
);

/    vehicles    /
CREATE TABLE branch1.vehicles (
id SERIAL NOT NULL,
mark VARCHAR(100) NOT NULL,
carry_capacity NUMERIC(9, 2) NOT NULL,
status vehicle_status_enum NOT NULL DEFAULT 'доступен',
driver_id INTEGER NOT NULL,
```

```
CONSTRAINT vehicles_pk PRIMARY KEY(id),
CONSTRAINT driver_fk FOREIGN KEY (driver_id)
REFERENCES branch1.drivers (id) MATCH SIMPLE
ON UPDATE NO ACTION
ON DELETE CASCADE,
CONSTRAINT driver_fk_unique_key UNIQUE(driver_id)
);

CREATE TABLE branch2.vehicles (
id SERIAL NOT NULL,
mark VARCHAR(100) NOT NULL,
carry_capacity NUMERIC(9, 2) NOT NULL,
status vehicle_status_enum NOT NULL DEFAULT 'доступен',
driver_id INTEGER NOT NULL,
CONSTRAINT vehicles_pk PRIMARY KEY(id),
CONSTRAINT driver_fk FOREIGN KEY (driver_id)
REFERENCES branch2.drivers (id) MATCH SIMPLE
ON UPDATE NO ACTION
ON DELETE CASCADE,
CONSTRAINT driver_fk_unique_key UNIQUE(driver_id)
);

CREATE TABLE branch3.vehicles (
id SERIAL NOT NULL,
mark VARCHAR(100) NOT NULL,
carry_capacity NUMERIC(9, 2) NOT NULL,
status vehicle_status_enum NOT NULL DEFAULT 'доступен',
driver_id INTEGER NOT NULL,
CONSTRAINT vehicles_pk PRIMARY KEY(id),
CONSTRAINT driver_fk FOREIGN KEY (driver_id)
REFERENCES branch3.drivers (id) MATCH SIMPLE
ON UPDATE NO ACTION
ON DELETE CASCADE,
CONSTRAINT driver_fk_unique_key UNIQUE(driver_id)
);



/     UPCOMING DRIVES     /
CREATE TABLE branch1.upcoming_drives(
id SERIAL NOT NULL,
driver_id INTEGER NOT NULL,
vehicles_id INTEGER NOT NULL,
cargo_weight NUMERIC(9, 2) NOT NULL,
destination VARCHAR(255),
destination_distance INTEGER NOT NULL,
price NUMERIC(     ,    )NOT NULL,
CONSTRAINT upcoming_drives_pk PRIMARY KEY (id),
CONSTRAINT driver_pk FOREIGN KEY (driver_id)
REFERENCES branch1.drivers(id) MATCH SIMPLE
ON UPDATE NO ACTION
ON DELETE CASCADE,
CONSTRAINT vehicles_fk FOREIGN KEY(vehicles_id)
REFERENCES branch1.vehicles(id) MATCH SIMPLE
ON UPDATE NO ACTION
ON DELETE CASCADE
);

CREATE TABLE branch2.upcoming_drives(
id SERIAL NOT NULL,
driver_id INTEGER NOT NULL,
vehicles_id INTEGER NOT NULL,
cargo_weight NUMERIC(9, 2) NOT NULL,
destination VARCHAR(255),
```

```
destination_distance INTEGER NOT NULL,
price NUMERIC(     ,    )NOT NULL,
CONSTRAINT upcoming_drives_pk PRIMARY KEY (id),
CONSTRAINT driver_pk FOREIGN KEY (driver_id)
REFERENCES branch2.drivers(id) MATCH SIMPLE
ON UPDATE NO ACTION
ON DELETE CASCADE,
CONSTRAINT vehicles_fk FOREIGN KEY(vehicles_id)
REFERENCES branch2.vehicles(id) MATCH SIMPLE
ON UPDATE NO ACTION
ON DELETE CASCADE
);

CREATE TABLE branch3.upcoming_drives(
id SERIAL NOT NULL,
driver_id INTEGER NOT NULL,
vehicles_id INTEGER NOT NULL,
cargo_weight NUMERIC(9, 2) NOT NULL,
destination VARCHAR(255),
destination_distance INTEGER NOT NULL,
price NUMERIC(     ,    )NOT NULL,
CONSTRAINT upcoming_drives_pk PRIMARY KEY (id),
CONSTRAINT driver_pk FOREIGN KEY (driver_id)
REFERENCES branch3.drivers(id) MATCH SIMPLE
ON UPDATE NO ACTION
ON DELETE CASCADE,
CONSTRAINT vehicles_fk FOREIGN KEY(vehicles_id)
REFERENCES branch3.vehicles(id) MATCH SIMPLE
ON UPDATE NO ACTION
ON DELETE CASCADE
);

/     ACTIVE DRIVES     /
CREATE TABLE branch1.active_drives(
id SERIAL NOT NULL,
driver_id INTEGER NOT NULL,
vehicles_id INTEGER NOT NULL,
cargo_weight NUMERIC(9, 2) NOT NULL,
destination VARCHAR(255),
destination_distance INTEGER NOT NULL,
price NUMERIC(     ,    )NOT NULL,
CONSTRAINT active_drives_pk PRIMARY KEY (id)
);

CREATE TABLE branch2.active_drives(
id SERIAL NOT NULL,
driver_id INTEGER NOT NULL,
vehicles_id INTEGER NOT NULL,
cargo_weight NUMERIC(9, 2) NOT NULL,
destination VARCHAR(255),
destination_distance INTEGER NOT NULL,
price NUMERIC(     ,    )NOT NULL,
CONSTRAINT active_drives_pk PRIMARY KEY (id)
);

CREATE TABLE branch3.active_drives(
id SERIAL NOT NULL,
driver_id INTEGER NOT NULL,
vehicles_id INTEGER NOT NULL,
cargo_weight NUMERIC(9, 2) NOT NULL,
destination VARCHAR(255),
destination_distance INTEGER NOT NULL,
price NUMERIC(     ,    )NOT NULL,
CONSTRAINT active_drives_pk PRIMARY KEY (id)
```

```
);

/     DRIVES HISTORY     /
CREATE TABLE branch1.drives_history(
id SERIAL NOT NULL,
driver_id INTEGER NOT NULL,
vehicles_id INTEGER NOT NULL,
cargo_weight NUMERIC(9, 2) NOT NULL,
destination VARCHAR(255),
destination_distance INTEGER NOT NULL,
price NUMERIC(     ,    )NOT NULL,
CONSTRAINT drives_history_pk PRIMARY KEY (id)
);

CREATE TABLE branch2.drives_history(
id SERIAL NOT NULL,
driver_id INTEGER NOT NULL,
vehicles_id INTEGER NOT NULL,
cargo_weight NUMERIC(9, 2) NOT NULL,
destination VARCHAR(255),
destination_distance INTEGER NOT NULL,
price NUMERIC(     ,    )NOT NULL,
CONSTRAINT drives_history_pk PRIMARY KEY (id)
);

CREATE TABLE branch3.drives_history(
id SERIAL NOT NULL,
driver_id INTEGER NOT NULL,
vehicles_id INTEGER NOT NULL,
cargo_weight NUMERIC(9, 2) NOT NULL,
destination VARCHAR(255),
destination_distance INTEGER NOT NULL,
price NUMERIC(     ,    )NOT NULL,
CONSTRAINT drives_history_pk PRIMARY KEY (id)
);
```

5. Функции

- Табличные

```
CREATE OR REPLACE FUNCTION get_managers_in_branch(branch VARCHAR(100))
RETURNS TABLE(
name NAME,
id OID
)
AS $$
BEGIN
RETURN QUERY(SELECT pg_roles.rolname, pg_roles.oid FROM pg_roles WHERE pg_roles.oid =
ANY(ARRAY[(SELECT grolist FROM pg_group WHERE groname = branch)]));
END $$ LANGUAGE plpgsql;
```

- Скалярные

```
/    INSERT DRIVE    /
CREATE OR REPLACE FUNCTION branch1.insert_drive(
driver_id_ INTEGER,
vehicles_id_ INTEGER,
cargo_weight_ NUMERIC(9, 2),
destination_ VARCHAR(255),
destination_distance_ INTEGER,
price_ NUMERIC(10 ,2 ))
RETURNS VOID
AS $$
BEGIN
INSERT INTO branch1.upcoming_drives(driver_id, vehicles_id, cargo_weight, destination, destination_distance,
price)
VALUES (driver_id_, vehicles_id_, cargo_weight_, destination_, destination_distance_, price_);
END $$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION branch2.insert_drive(
driver_id_ INTEGER,
vehicles_id_ INTEGER,
cargo_weight_ NUMERIC(9, 2),
destination_ VARCHAR(255),
destination_distance_ INTEGER,
price_ NUMERIC(10, 2))
RETURNS VOID
AS $$
BEGIN
INSERT INTO branch2.upcoming_drives(driver_id, vehicles_id, cargo_weight, destination, destination_distance,
price)
VALUES (driver_id_, vehicles_id_, cargo_weight_, destination_, destination_distance_, price_);
END $$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION branch3.insert_drive(
driver_id_ INTEGER,
vehicles_id_ INTEGER,
cargo_weight_ NUMERIC(9, 2),
destination_ VARCHAR(255),
destination_distance_ INTEGER,
price_ NUMERIC(10, 2))
RETURNS VOID
AS $$
BEGIN
INSERT INTO branch3.upcoming_drives(driver_id, vehicles_id, cargo_weight, destination, destination_distance,
price)
VALUES (driver_id_, vehicles_id_, cargo_weight_, destination_, destination_distance_, price_);
END $$ LANGUAGE plpgsql;



/    SET DRIVE ACTIVE    /
CREATE OR REPLACE FUNCTION branch1.set_drive_active(drive_id INTEGER)
RETURNS BOOLEAN
AS $$
BEGIN
IF (SELECT COUNT(*) FROM branch1.upcoming_drives WHERE id = drive_id) = 0 THEN
RETURN 'f';
END IF;
```

```
INSERT INTO branch1.active_drives SELECT * FROM branch1.upcoming_drives WHERE id = drive_id;
DELETE FROM branch   .upcoming_drives WHERE id = drive_id;
UPDATE branch1.drivers SET status = 'совершает рейс' WHERE id = (SELECT driver_id FROM
branch   .active_drives WHERE id = drive_id);
UPDATE branch1.vehicles SET status = 'недоступен' WHERE id = (SELECT vehicles_id FROM
branch   .active_drives WHERE id = drive_id);
RETURN 't';
END $$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION branch2.set_drive_active(drive_id INTEGER)
RETURNS BOOLEAN
AS $$
BEGIN
IF (SELECT COUNT(*) FROM branch2.upcoming_drives WHERE id = drive_id) = 0 THEN
RETURN 'f';
END IF;
INSERT INTO branch2.active_drives SELECT * FROM branch2.upcoming_drives WHERE id = drive_id;
DELETE FROM branch   .upcoming_drives WHERE id = drive_id;
UPDATE branch2.drivers SET status = 'совершает рейс' WHERE id = (SELECT driver_id FROM
branch   .active_drives WHERE id = drive_id);
UPDATE branch2.vehicles SET status = 'недоступен' WHERE id = (SELECT vehicles_id FROM
branch   .active_drives WHERE id = drive_id);
RETURN 't';
END $$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION branch3.set_drive_active(drive_id INTEGER)
RETURNS BOOLEAN
AS $$
BEGIN
IF (SELECT COUNT(*) FROM branch3.upcoming_drives WHERE id = drive_id) = 0 THEN
RETURN 'f';
END IF;
INSERT INTO branch3.active_drives SELECT * FROM branch3.upcoming_drives WHERE id = drive_id;
DELETE FROM branch   .upcoming_drives WHERE id = drive_id;
UPDATE branch3.drivers SET status = 'совершает рейс' WHERE id = (SELECT driver_id FROM
branch   .active_drives WHERE id = drive_id);
UPDATE branch3.vehicles SET status = 'недоступен' WHERE id = (SELECT vehicles_id FROM
branch   .active_drives WHERE id = drive_id);
RETURN 't';
END $$ LANGUAGE plpgsql;

/     FINISH DRIVE     /
CREATE OR REPLACE FUNCTION branch1.finish_drive(drive_id INTEGER)
RETURNS BOOLEAN
AS $$
BEGIN
IF (SELECT COUNT(*) FROM branch1.active_drives WHERE id = drive_id) = 0 THEN
RETURN 'f';
END IF;
INSERT INTO branch1.drives_history SELECT * FROM branch1.active_drives WHERE id = drive_id;
DELETE FROM branch   .active_drives WHERE id = drive_id;
RETURN 't';
END $$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION branch1.pay_driver_trigger_procedure()
RETURNS TRIGGER
AS $$
BEGIN
UPDATE branch1.drivers SET
earnings = earnings + NEW.price,
status = 'свободен'
WHERE id = NEW.driver_id;
UPDATE branch1.vehicles SET
status = 'доступен'
```

```
WHERE id = NEW.vehicles_id;
RETURN NEW;
END $$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION branch2.finish_drive(drive_id INTEGER)
RETURNS BOOLEAN
AS $$
BEGIN
IF (SELECT COUNT(*) FROM branch2.active_drives WHERE id = drive_id) = 0 THEN
RETURN 'f';
END IF;
INSERT INTO branch2.drives_history SELECT * FROM branch2.active_drives WHERE id = drive_id;
DELETE FROM branch2  .active_drives WHERE id = drive_id;
RETURN 't';
END $$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION branch2.pay_driver_trigger_procedure()
RETURNS TRIGGER
AS $$
BEGIN
UPDATE branch2.drivers SET
earnings = earnings + NEW.price,
status = 'свободен'
WHERE id = NEW.driver_id;
UPDATE branch2.vehicles SET
status = 'доступен'
WHERE id = NEW.vehicles_id;
RETURN NEW;
END $$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION branch3.finish_drive(drive_id INTEGER)
RETURNS BOOLEAN
AS $$
BEGIN
IF (SELECT COUNT(*) FROM branch3.active_drives WHERE id = drive_id) = 0 THEN
RETURN 'f';
END IF;
INSERT INTO branch3.drives_history SELECT * FROM branch3.active_drives WHERE id = drive_id;
DELETE FROM branch3  .active_drives WHERE id = drive_id;
RETURN 't';
END $$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION branch3.pay_driver_trigger_procedure()
RETURNS TRIGGER
AS $$
BEGIN
UPDATE branch3.drivers SET
earnings = earnings + NEW.price,
status = 'свободен'
WHERE id = NEW.driver_id;
UPDATE branch3.vehicles SET
status = 'доступен'
WHERE id = NEW.vehicles_id;
RETURN NEW;
END $$ LANGUAGE plpgsql;

/     DELETE DRIVERS     /
CREATE OR REPLACE FUNCTION branch1.delete_driver(driver_id INTEGER)
RETURNS BOOLEAN
AS $$
DECLARE
count_ INT;
BEGIN
WITH deleted AS (DELETE FROM branch1.drivers WHERE id = driver_id RETURNING *) SELECT count(*) INTO
```

```
count_ FROM deleted;
RETURN count_::BOOLEAN;
END $$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION branch2.delete_driver(driver_id INTEGER)
RETURNS BOOLEAN
AS $$
DECLARE
count_ INT;
BEGIN
WITH deleted AS (DELETE FROM branch2.drivers WHERE id = driver_id RETURNING *) SELECT count(*) INTO
count_ FROM deleted;
RETURN count_::BOOLEAN;
END $$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION branch3.delete_driver(driver_id INTEGER)
RETURNS BOOLEAN
AS $$
DECLARE
count_ INT;
BEGIN
WITH deleted AS (DELETE FROM branch3.drivers WHERE id = driver_id RETURNING *) SELECT count(*) INTO
count_ FROM deleted;
RETURN count_::BOOLEAN;
END $$ LANGUAGE plpgsql;

/     DELETE VEHICLES     /
CREATE OR REPLACE FUNCTION branch1.delete_vehicle(vehicle_id INTEGER)
RETURNS BOOLEAN
AS $$
DECLARE
count_ INT;
BEGIN
WITH deleted AS (DELETE FROM branch1.vehicles WHERE id = vehicle_id RETURNING *) SELECT count(*) INTO
count_ FROM deleted;
RETURN count_::BOOLEAN;
END $$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION branch2.delete_vehicle(vehicle_id INTEGER)
RETURNS BOOLEAN
AS $$
DECLARE
count_ INT;
BEGIN
WITH deleted AS (DELETE FROM branch2.vehicles WHERE id = vehicle_id RETURNING *) SELECT count(*) INTO
count_ FROM deleted;
RETURN count_::BOOLEAN;
END $$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION branch3.delete_vehicle(vehicle_id INTEGER)
RETURNS BOOLEAN
AS $$
DECLARE
count_ INT;
BEGIN
WITH deleted AS (DELETE FROM branch3.vehicles WHERE id = vehicle_id RETURNING *) SELECT count(*) INTO
count_ FROM deleted;
RETURN count_::BOOLEAN;
END $$ LANGUAGE plpgsql;

/     DELETE DRIVES     upcoming only         /
CREATE OR REPLACE FUNCTION branch1.delete_drive(drive_id INTEGER)
RETURNS BOOLEAN
AS $$
```

```
DECLARE
count_ INT;
BEGIN
WITH deleted AS (DELETE FROM branch1.upcoming_drives WHERE id = drive_id RETURNING *) SELECT count(*)
INTO count_ FROM deleted;
RETURN count_::BOOLEAN;
END $$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION branch2.delete_drive(drive_id INTEGER)
RETURNS BOOLEAN
AS $$
DECLARE
count_ INT;
BEGIN
WITH deleted AS (DELETE FROM branch2.upcoming_drives WHERE id = drive_id RETURNING *) SELECT count(*)
INTO count_ FROM deleted;
RETURN count_::BOOLEAN;
END $$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION branch3.delete_drive(drive_id INTEGER)
RETURNS BOOLEAN
AS $$
DECLARE
count_ INT;
BEGIN
WITH deleted AS (DELETE FROM branch3.upcoming_drives WHERE id = drive_id RETURNING *) SELECT count(*)
INTO count_ FROM deleted;
RETURN count_::BOOLEAN;
END $$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION user_group() RETURNS TEXT
LANGUAGE plpgsql
AS $$
BEGIN
RETURN (select rolname from pg_user
join pg_auth_members on (pg_user.usesysid=pg_auth_members.member)
join pg_roles on (pg_roles.oid=pg_auth_members.roleid)
where
pg_user.usename=CURRENT_USER);
END;
$$;
CREATE OR REPLACE FUNCTION is_super() RETURNS BOOLEAN
LANGUAGE plpgsql
AS $$
BEGIN
RETURN (SELECT usesuper FROM pg_user WHERE usename = CURRENT_USER);
END;
$$;
```

6. Процедуры

```
CREATE OR REPLACE PROCEDURE branch1.insert_driver(fio_ VARCHAR(100), driving_licence_ VARCHAR(100))
AS $$
BEGIN
INSERT INTO branch1.drivers(fio, driving_licence) VALUES(fio_, driving_licence_) ON CONFLICT (driving_licence)
DO NOTHING;
```

```
END $$ LANGUAGE plpgsql;

CREATE OR REPLACE PROCEDURE branch2.insert_driver(fio_ VARCHAR(100), driving_licence_ VARCHAR(100))
AS $$
BEGIN
INSERT INTO branch2.drivers(fio, driving_licence) VALUES(fio_, driving_licence_) ON CONFLICT (driving_licence)
DO NOTHING;
END $$ LANGUAGE plpgsql;

CREATE OR REPLACE PROCEDURE branch3.insert_driver(fio_ VARCHAR(100), driving_licence_ VARCHAR(100))
AS $$
BEGIN
INSERT INTO branch3.drivers(fio, driving_licence) VALUES(fio_, driving_licence_) ON CONFLICT (driving_licence)
DO NOTHING;
END $$ LANGUAGE plpgsql;

/     INSERT VEHICLE     /
CREATE OR REPLACE PROCEDURE branch1.insert_vehicle(mark_ VARCHAR(100), carry_capacity_ NUMERIC(9, 2),
driver_id_ INTEGER)
AS $$
BEGIN
INSERT INTO branch1.vehicles(mark, carry_capacity, driver_id) VALUES(mark_, carry_capacity_, driver_id_) ON
CONFLICT (driver_id) DO UPDATE SET mark = mark_, carry_capacity = carry_capacity_;
END $$ LANGUAGE plpgsql;

CREATE OR REPLACE PROCEDURE branch2.insert_vehicle(mark_ VARCHAR(100), carry_capacity_ NUMERIC(9, 2),
driver_id_ INTEGER)
AS $$
BEGIN
INSERT INTO branch2.vehicles(mark, carry_capacity, driver_id) VALUES(mark_, carry_capacity_, driver_id_) ON
CONFLICT (driver_id) DO UPDATE SET mark = mark_, carry_capacity = carry_capacity_;
END $$ LANGUAGE plpgsql;

CREATE OR REPLACE PROCEDURE branch3.insert_vehicle(mark_ VARCHAR(100), carry_capacity_ NUMERIC(9, 2),
driver_id_ INTEGER)
AS $$
BEGIN
INSERT INTO branch3.vehicles(mark, carry_capacity, driver_id) VALUES(mark_, carry_capacity_, driver_id_) ON
CONFLICT (driver_id) DO UPDATE SET mark = mark_, carry_capacity = carry_capacity_;
END $$ LANGUAGE plpgsql;



/    CREATE GROUP     /
CREATE OR REPLACE PROCEDURE add_group(group_name VARCHAR(50))
LANGUAGE plpgsql
AS $$
BEGIN
execute 'CREATE ROLE ' || group_name;
execute 'GRANT ALL PRIVILEGES ON SCHEMA ' || group_name || ' TO GROUP ' || group_name;
execute 'GRANT ALL ON ALL TABLES IN SCHEMA ' || group_name || ' TO GROUP ' || group_name;
execute 'GRANT USAGE ON ALL SEQUENCES IN SCHEMA ' || group_name || ' TO GROUP ' || group_name;
execute 'GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA ' || group_name || ' TO GROUP ' || group_name;
execute 'GRANT EXECUTE ON ALL PROCEDURES IN SCHEMA ' || group_name || ' TO GROUP ' || group_name;
execute 'GRANT SELECT ON ' || group_name || '_drivers_view TO GROUP ' || group_name;
execute 'GRANT SELECT ON ' || group_name || '_vehicles_view TO GROUP ' || group_name;
execute 'GRANT SELECT ON ' || group_name || '_active_drives_view TO GROUP ' || group_name;
execute 'GRANT SELECT ON ' || group_name || '_upcoming_drives_view TO GROUP ' || group_name;
execute 'GRANT SELECT ON ' || group_name || '_history_drives_view TO GROUP ' || group_name;
execute 'GRANT EXECUTE ON FUNCTION user_group() TO GROUP ' || group_name;
END;
$$;
```

```
CREATE OR REPLACE PROCEDURE add_user(username VARCHAR(50), password VARCHAR(50), group_
VARCHAR(50))
LANGUAGE plpgsql
AS $$
BEGIN
execute 'CREATE ROLE ' ||username|| ' WITH LOGIN PASSWORD ''' ||password|| ''' IN GROUP ' || group_;
END; $$;
```

7. Представления

```
/     VIEWS     /
CREATE OR REPLACE VIEW branch1_drivers_view AS
SELECT * FROM branch1.drivers;
CREATE OR REPLACE VIEW branch2_drivers_view AS
SELECT * FROM branch2.drivers;
CREATE OR REPLACE VIEW branch3_drivers_view AS
SELECT * FROM branch3.drivers;
CREATE OR REPLACE VIEW drivers_view AS /*Main admin only*/
SELECT * FROM branch1.drivers UNION ALL SELECT * FROM branch2.drivers UNION ALL SELECT * FROM
branch3.drivers;


CREATE OR REPLACE VIEW branch1_vehicles_view AS
SELECT * FROM branch1.vehicles;
CREATE OR REPLACE VIEW branch2_vehicles_view AS
SELECT * FROM branch2.vehicles;
CREATE OR REPLACE VIEW branch3_vehicles_view AS
SELECT * FROM branch3.vehicles;
CREATE OR REPLACE VIEW vehicles_view AS /*Main admin only*/
SELECT * FROM branch1.vehicles UNION ALL SELECT * FROM branch2.vehicles UNION ALL SELECT * FROM
branch3.vehicles;


CREATE OR REPLACE VIEW branch1_upcoming_drives_view AS
SELECT * FROM branch1.upcoming_drives;
CREATE OR REPLACE VIEW branch2_upcoming_drives_view AS
SELECT * FROM branch2.upcoming_drives;
CREATE OR REPLACE VIEW branch3_upcoming_drives_view AS
SELECT * FROM branch3.upcoming_drives;
CREATE OR REPLACE VIEW upcoming_drives_view AS /*Main admin only*/
SELECT * FROM branch1.upcoming_drives UNION ALL SELECT * FROM branch2.upcoming_drives UNION ALL
SELECT * FROM branch3.upcoming_drives;


CREATE OR REPLACE VIEW branch1_active_drives_view AS
SELECT * FROM branch1.active_drives;
CREATE OR REPLACE VIEW branch2_active_drives_view AS
SELECT * FROM branch2.active_drives;
CREATE OR REPLACE VIEW branch3_active_drives_view AS
SELECT * FROM branch3.active_drives;
CREATE OR REPLACE VIEW active_drives_view AS /*Main admin only*/
SELECT * FROM branch1.active_drives UNION ALL SELECT * FROM branch2.active_drives UNION ALL SELECT *
FROM branch    .active_drives;


CREATE OR REPLACE VIEW branch1_history_drives_view AS
SELECT * FROM branch1.drives_history;
CREATE OR REPLACE VIEW branch2_history_drives_view AS
SELECT * FROM branch2.drives_history;
```

```
CREATE OR REPLACE VIEW branch3_history_drives_view AS
SELECT * FROM branch3.drives_history;
CREATE OR REPLACE VIEW history_drives_view AS /*Main admin only*/
SELECT * FROM branch1.drives_history UNION ALL SELECT * FROM branch2.drives_history UNION ALL SELECT *
FROM branch    .drives_history;
```

8. CLR функция

Так как данная работа выполнена на PostgreSQL невозможно было выполнить данный пункт.

9. Триггеры

```
CREATE TRIGGER on_drive_finished_trigger AFTER INSERT ON branch1.drives_history FOR EACH ROW EXECUTE
PROCEDURE branch1.pay_driver_trigger_procedure();

CREATE TRIGGER on_drive_finished_trigger AFTER INSERT ON branch2.drives_history FOR EACH ROW EXECUTE
PROCEDURE branch2.pay_driver_trigger_procedure();

CREATE TRIGGER on_drive_finished_trigger AFTER INSERT ON branch3.drives_history FOR EACH ROW EXECUTE
PROCEDURE branch3.pay_driver_trigger_procedure();
```

10. Приложение

- Документация для VUE.js: https://vuejs.org/
- Создание http сервера: https://medium.com/@ryanblunden/create-a-http-server-with-one-command-thanks-to- python-29fcfdcd240e
- Fast API: https://fastapi.tiangolo.com/
- Uvicorn: https://www.uvicorn.org/