Проект по базам данных

Бурлака Полина, Кокорева Кира, Вороненко Егор, ИАД - 13

Описание

Разработка базы данных для агрегатора такси, которую будут использовать аналитики компании для получения данных о поездках, пользователях, таксистах. При этом требуется создать базу, содержащую только успешно выполненные заказы.

Функциональные требования

1. Хранение информации о поездках:

- Данные о каждой успешной поездке, включая:
  - Id поездки
  - Id пользователя
  - Id водителя
  - Id машины
  - Дату и время начала и окончания поездки
  - Место посадки (адрес)
  - Место высадки (адрес)
  - Стоимость поездки
  - Способ оплаты (наличные, карта)
  - Километры в пути
  - Время ожидания клиента (от момента уведомления на телефон клиента о прибытии до посадки в авто)
  - Коэффициент, на который умножается стоимость поездки (1 для нормальных условий, 1.5 для плохой погоды/повышенного спроса, 2 для плохой погоды И повышенного спроса)

2. Хранение информации о пользователях:

- Данные о каждом пользователе, включая:
  - Id
  - Фио
  - Пол
  - Дата рождения
  - Номер телефона
  - Электронная почта
  - Дата первой поездки

3. Хранение информации о машинах:

- Данные о каждом авто, включая:
  - Id - он же уникальный VIN номер
  - Марка
  - Название модели
  - Год выпуска
  - Класс (эконом/комфорт/бизнес)

    - Госномер
    - Состояние (в работе/на ремонте/списана)

4. Хранение информации о водителях:

- Данные о каждом водителе, включая:
    - Id водителя
    - Фио
    - Паспорт
    - Номер в/у
    - Пол
    - Номер телефона
    - Дата начала работы
    - Дата рождения
    - Активен ли водитель

5. Хранение информации об отзывах:

- Данные о каждом отзыве, включая:
    - Id заказа
    - Оценка пользователем водителя
    - Оценка водителем пользователя

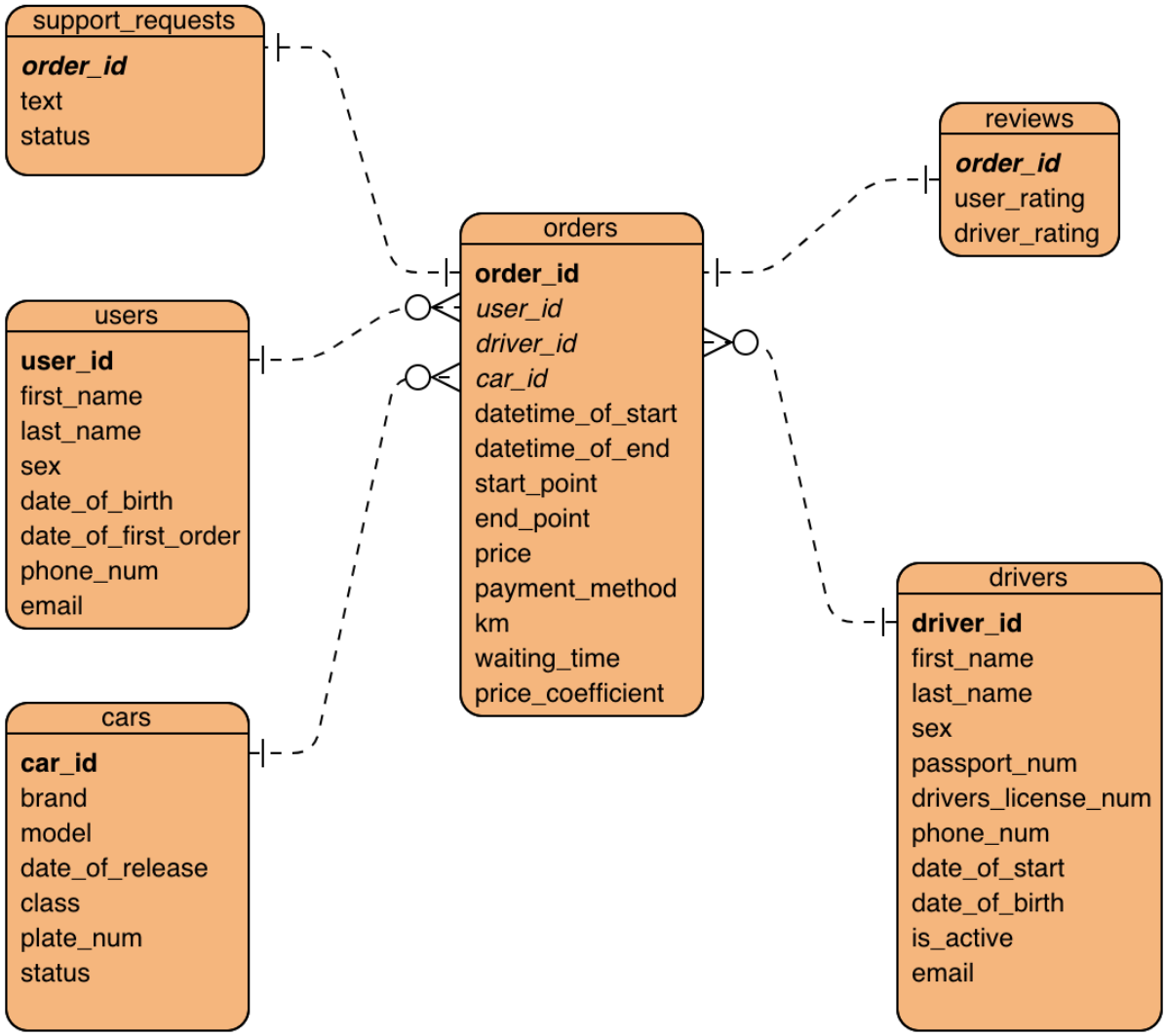
6. Хранение информации об обращениях в поддержку:

- Данные о каждом обращении, включая:
    - Id заказа
    - Текст обращения
    - Статус обращения (в работе/обработано)

7. Сбор статистики по работе сервиса:

- Возможность агрегировать данные для аналитики, к примеру:
    - Распределение машин по классам;
    - Распределение пользователей по тратам (активности) в месяц;
    - Ранжирование водителей по разным показателям;
    - Динамика развития сервиса в материальном эквиваленте (рост/снижение дохода);
    - Выявление некомпетентных сотрудников по оценкам пользователей.

ER – диаграмма



Ограничения данных:

1. Все поля таблицы ‘orders’ должны быть заполнены.
2. Все поля таблицы ‘cars’ должны быть заполнены, поле ‘status’ должны быть в трех состояниях {в работе, на ремонте, списана}, поле ‘plate_num’ должно быть уникально
3. Все поля таблицы ‘drivers’ должны быть заполнены, поля ‘passport_num’, ‘drivers_license_num’ должны быть уникальны, ‘is_active’ должно быть в двух состояниях {активен/ не активен}
4. В таблице ‘support_requests’ поле ‘status’ должно быть в двух состояниях { в работе/ решено}
5. В таблице ‘users’ должны быть заполнены поля ‘email’, ‘phone_num’
6. Во всех таблицах поля вида ‘_id’ должны быть уникальны.
7. Ценовой коэффициент должен быть одинаковый для заказов, выполняемых в один час
8. Время окончания поездки должно быть позже времени начала поездки
9. Один водитель в одно время может иметь не больше одного заказа
10. Один пользователь в одно время может иметь не больше одного заказа
11. Одна машина в одно время может участвовать только в одном заказе
12. В одном заказе должен быть один водитель, один пользователь, одна машина
13. В заказах могут участвовать только машины со статусом на время поездки «в работе»

14. В заказах могут участвовать только активные на момент поездки водители
15. Место посадки не может быть таким же как место высадки
16. Стоимость поездки однозначно определяется расстоянием пути, ценовым коэффициентом, временем ожидания и классом машины

Функциональные зависимости:

1. Orders:
    a. order_id → user_id, driver_id, car_id, datetime_of start, datetime_of_end, start_point, end_point, price, payment_method, km, waiting_time, price_coefficient
    b. car_id, km, price_coefficient, weiting_time → price
    c. datetime_of_start, start_point → price_coefficient
2. Users
    a. usesr_id → first_name, last_name, sex, date_od_birth, date_of_first_order, phone_num, email
3. Support_requests
   order_id → text, status
4. Reviews
    a. order_id → uder_rating, driver_rating
5. Cars
    a. car_id → brand, model, date_of_release, class, plate_num, status
    b. plate_num → brand, model, date_of_release, class, status, car_id
    c. brand, model → class
6. Drivers
    a. driver_id → first_name, last_name, sex, passport_num, drivers_license_num, phone_num, date_of_start, date_of_birth, is_active, email
    b. passport_num → driver_id, first_name, last_name, sex, drivers_license_num, phone_num, date_of_start, date_of_birth, is_active, email
    c. drivers_license_num → driver_id, first_name, last_name, sex, passport_num, phone_num, date_of_start, date_of_birth, is_active, email
    d. phone_num → driver_id, first_name, last_name, sex, passport_num, drivers_license_num, date_of_start, date_of_birth, is_active, email

не в BCNF: Orders, Cars

Cars разбивается на две таблицы:
{brand, model, class}
{car_id, brand, model, date_of_release, plate_num, status}

Orders разбивается на три таблицы:
{car_id, km, price_coefficient, weiting_time, price}
{datetime_of_start, start_point, price_coefficient}
{order_id, user_id, driver_id, car_id, datetime_of start, datetime_of_end, start_point, end_point, payment_method, km, waiting_time}

(Проблема что в таблице {car_id, km, price_coefficient, weiting_time, price} надо хранить информацию для всех km, а они не дискретные и получится очень длинная таблица, по сути для каждого заказа будет своя строка в этой таблице. Это можно решить тем, что будут храниться значения с точностью например до 1 км, что при большом количестве поездок уменьшит объем таблицы)

В недонормализованной таблице может например возникнуть проблема, что при необходимости изменить класс какой-либо модели машины, это надо будет сделать много раз.

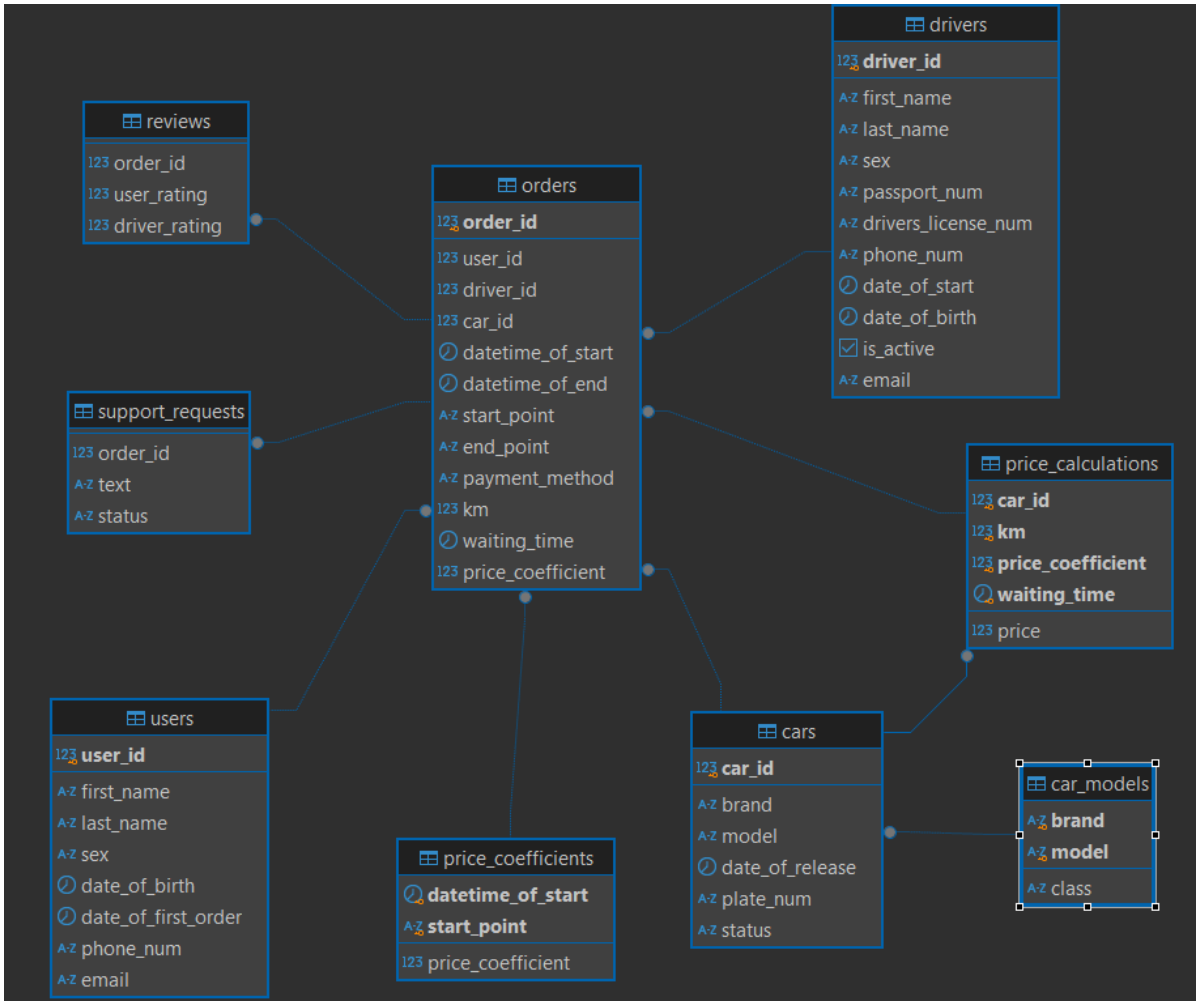
Создание таблиц.

```
-- Создание базы данных
CREATE DATABASE taxi_aggregator;
```

```
-- Подключение к базе данных
\c taxi_aggregator;

-- Таблица Users
CREATE TABLE Users (
    user_id SERIAL PRIMARY KEY,
    first_name VARCHAR(255) NOT NULL,
    last_name VARCHAR(255) NOT NULL,
    sex CHAR(1) CHECK (sex IN ('M', 'F')),
    date_of_birth DATE NOT NULL,
    date_of_first_order TIMESTAMP WITH TIME ZONE,
    phone_num VARCHAR(20) UNIQUE,
    email VARCHAR(255) UNIQUE
);

-- Таблица Drivers
CREATE TABLE Drivers (
    driver_id SERIAL PRIMARY KEY,
    first_name VARCHAR(255) NOT NULL,
    last_name VARCHAR(255) NOT NULL,
    sex CHAR(1) CHECK (sex IN ('M', 'F')),
    passport_num VARCHAR(50) UNIQUE NOT NULL,
    drivers_license_num VARCHAR(50) UNIQUE NOT NULL,
    phone_num VARCHAR(20) UNIQUE NOT NULL,
    date_of_start DATE NOT NULL,
    date_of_birth DATE NOT NULL,
    is_active BOOLEAN NOT NULL DEFAULT TRUE,
    email VARCHAR(255) UNIQUE
);

-- Таблица Car_models
CREATE TABLE Car_models (
    brand VARCHAR(255) NOT NULL,
    model VARCHAR(255) NOT NULL,
    class VARCHAR(255) NOT NULL,
    PRIMARY KEY (brand, model)
);

-- Таблица Cars
CREATE TABLE Cars (
    car_id SERIAL PRIMARY KEY,
    brand VARCHAR(255) NOT NULL,
    model VARCHAR(255) NOT NULL,
    date_of_release DATE NOT NULL,
    plate_num VARCHAR(20) UNIQUE NOT NULL,
    status VARCHAR(50) NOT NULL CHECK (status IN ('active', 'inactive')),
    FOREIGN KEY (brand, model) REFERENCES Car_models(brand, model) ON DELETE
CASCADE
);
```

```
-- Таблица Price_coefficients
CREATE TABLE Price_coefficients (
    datetime_of_start TIMESTAMP WITH TIME ZONE NOT NULL,
    start_point VARCHAR(255) NOT NULL,
    price_coefficient NUMERIC(5, 2) NOT NULL,
    PRIMARY KEY (datetime_of_start, start_point)
);

-- Таблица Price_calculations
CREATE TABLE Price_calculations (
    car_id INTEGER REFERENCES Cars(car_id) ON DELETE CASCADE NOT NULL,
    km NUMERIC(10, 2) NOT NULL,
    price_coefficient NUMERIC(5, 2) NOT NULL,
    waiting_time INTERVAL NOT NULL,
    price NUMERIC(10, 2) NOT NULL,
    PRIMARY KEY (car_id, km, price_coefficient, waiting_time)
);


-- Таблица Orders
CREATE TABLE Orders (
    order_id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES Users(user_id) ON DELETE CASCADE NOT NULL,
    driver_id INTEGER REFERENCES Drivers(driver_id) ON DELETE CASCADE NOT NULL,
    car_id INTEGER REFERENCES Cars(car_id) ON DELETE CASCADE NOT NULL,
    datetime_of_start TIMESTAMP WITH TIME ZONE NOT NULL,
    datetime_of_end TIMESTAMP WITH TIME ZONE NOT NULL,
    start_point VARCHAR(255) NOT NULL,
    end_point VARCHAR(255) NOT NULL,
    payment_method VARCHAR(50) NOT NULL,
    km NUMERIC(10, 2) NOT NULL,
    waiting_time INTERVAL NOT NULL,
    price_coefficient NUMERIC(5, 2) NOT NULL,
    FOREIGN KEY (datetime_of_start, start_point) REFERENCES
Price_coefficients(datetime_of_start, start_point) ON DELETE CASCADE,
    FOREIGN KEY (car_id, km, price_coefficient, waiting_time) REFERENCES
Price_calculations(car_id, km, price_coefficient, waiting_time) ON DELETE CASCADE

);


-- Таблица Support_requests
CREATE TABLE Support_requests (
    order_id INTEGER REFERENCES Orders(order_id) ON DELETE CASCADE NOT NULL,
    text TEXT NOT NULL,
    status VARCHAR(50) NOT NULL
);

-- Таблица Reviews
CREATE TABLE Reviews (
    order_id INTEGER REFERENCES Orders(order_id) ON DELETE CASCADE NOT NULL,
    user_rating INTEGER CHECK (user_rating BETWEEN 1 AND 5),
```

```
    driver_rating INTEGER CHECK (driver_rating BETWEEN 1 AND 5)
);
```



Запросы

1) Доход водителей за период.

```
SELECT
  d.first_name || ' ' || d.last_name AS driver_name,
  SUM(o.price) AS total_income
FROM
  Orders AS o
JOIN
  Drivers AS d ON o.driver_id = d.driver_id
WHERE
  o.datetime_of_start BETWEEN '2023-10-26' AND '2023-10-31'
GROUP BY
  driver_name
ORDER BY
  total_income DESC;
```

2) Среднее время ожидания по водителям.

```
SELECT
  d.first_name || ' ' || d.last_name AS driver_name,
  AVG(EXTRACT(EPOCH FROM o.waiting_time) / 60) AS avg_waiting_time_minutes
```

```
FROM
  Orders AS o
JOIN
  Drivers AS d ON o.driver_id = d.driver_id
GROUP BY
  driver_name
ORDER BY
  avg_waiting_time_minutes DESC;
```

3) Активность пользователей (количество заказов за последние 30 дней).

```
SELECT
    user_id,
    first_name,
    last_name,
    COUNT(*) OVER (PARTITION BY user_id ORDER BY datetime_of_start DESC
ROWS BETWEEN 29 PRECEDING AND CURRENT ROW) as orders_last_30_days
FROM
    Users
JOIN
    Orders ON Users.user_id = Orders.user_id
WHERE
    datetime_of_start >= date('now', '-30 days')
ORDER BY
    orders_last_30_days DESC;
```

4) Рейтинг водителей по количеству заказов.

```
SELECT
    d.first_name || ' ' || d.last_name AS driver_name,
    COUNT(o.order_id) AS total_orders,
    RANK() OVER (ORDER BY COUNT(o.order_id) DESC) AS driver_rank
FROM
    Drivers d
JOIN
    Orders o ON d.driver_id = o.driver_id
GROUP BY
    driver_name
ORDER BY
    driver_rank;
```

5) Динамика дохода водителей в течение года. Вычисляет доход водителей за каждый месяц с прошлого года. `LAG()` позволяет сравнить текущий доход с доходом предыдущего месяца для анализа тенденций.

```
SELECT
  d.first_name || ' ' || d.last_name AS driver_name,
  DATE_TRUNC('month', o.datetime_of_start) AS month,
  SUM(o.price) AS monthly_income,
  LAG(SUM(o.price), 1, 0) OVER (PARTITION BY d.driver_id ORDER BY
DATE_TRUNC('month', o.datetime_of_start)) AS previous_month_income
FROM
  Orders o
JOIN
  Drivers d ON o.driver_id = d.driver_id
WHERE
  o.datetime_of_start >= date('now', '-1 year')
GROUP BY
```

```
  driver_name, month
ORDER BY
  driver_name, month;
```

6) Популярные классы автомобилей по городам

```
SELECT
  c.class,
  o.start_point,
  COUNT(*) AS num_rides
FROM
  Orders AS o
JOIN
  Cars AS c ON o.car_id = c.car_id
GROUP BY
  c.class,
  o.start_point
ORDER BY
  num_rides DESC
LIMIT 10;
```

7) Топ 10 водителей с самыми частыми оценками ниже 4 (используется `LEFT JOIN` для того, чтобы в запросе участвовали все водители, даже если у них нет отзывов. Это важно для корректного подсчёта водителей, у которых низкий рейтинг, но отзывы могут отсутствовать. `WHERE` условие также уточнено для исключения null-значений.)

```
SELECT
  d.first_name || ' ' || d.last_name AS driver_name,
  COUNT(r.order_id) AS num_low_rated_orders
FROM
  Drivers AS d
JOIN
  Orders AS o ON d.driver_id = o.driver_id
LEFT JOIN
  Reviews AS r ON o.order_id = r.order_id
WHERE
  r.driver_rating IS NOT NULL AND r.driver_rating < 4
GROUP BY
  driver_name
ORDER BY
  num_low_rated_orders DESC
LIMIT 10;
```