Домашнее задание № 8

Кокорин Илья, M3439

11 ноября 2019 г.

Весь код писался и тестировался в PostgreSQL 11.1 on x86_64-apple-darwin17.7.0, compiled by Apple LLVM version 10.0.0 (clang-1000.11.45.5), 64-bit

1 Схема базы данных

```
CREATE TABLE Planes
(
  plane_id            INT        NOT NULL PRIMARY KEY,
  registration_number VARCHAR(7) NOT NULL
);

CREATE TABLE Seats
(
  plane_id INT NOT NULL,
  seat_no  INT NOT NULL,
  PRIMARY KEY (plane_id, seat_no),
  FOREIGN KEY (plane_id) REFERENCES Planes (plane_id)
);

CREATE TABLE Flights
(
  flight_id               INT       NOT NULL PRIMARY KEY,
  flight_time             TIMESTAMP NOT NULL,
  plane_id                INT       NOT NULL,
  closed_by_administrator BOOLEAN   NOT NULL DEFAULT FALSE,
  FOREIGN KEY (plane_id) REFERENCES Planes (plane_id)
);

CREATE TABLE Users
(
  user_id    INT         NOT NULL PRIMARY KEY,
  first_name VARCHAR(50) NOT NULL,
  surname    VARCHAR(50) NOT NULL
);

CREATE TABLE Reservation
(
  flight_id             INT       NOT NULL,
  seat_no               INT       NOT NULL,
  reservation_timestamp TIMESTAMP NOT NULL,
  user_id               INT       NOT NULL,
  FOREIGN KEY (user_id) REFERENCES Users (user_id),
  FOREIGN KEY (flight_id) REFERENCES Flights (flight_id),
  PRIMARY KEY (flight_id, seat_no)
);
```

```
CREATE TABLE Bought
(
  flight_id INT NOT NULL,
  seat_no   INT NOT NULL,
  user_id   INT NOT NULL,
  FOREIGN KEY (user_id) REFERENCES Users (user_id),
  FOREIGN KEY (flight_id) REFERENCES Flights (flight_id),
  PRIMARY KEY (flight_id, seat_no)
);
```

В таблице Reservation будем хранить бронирования, а в таблице Bought - покупки.

2 Скрипты для бронирования, покупки и отмены покупки администратором

```
-- Проверяет, что рейс flight_id_arg существует, в самолёте,
-- совершающем данный рейс, есть место seat_no_arg,
-- бронирование (или покупка) на рейс flight_id_arg может быть
-- совершено во время action_timestamp_arg. (то есть осталось
-- больше суток/двух часов до вылета и
-- продажа на рейс не запрещена администратором)
CREATE OR REPLACE FUNCTION
  seat_exists_and_can_be_processed(flight_id_arg INT,
                                   seat_no_arg INT,
                                   action_timestamp_arg TIMESTAMP,
                                   interval_arg INTERVAL)
  RETURNS BOOLEAN AS
$$
BEGIN
  RETURN EXISTS(
      SELECT *
      FROM Flights
             NATURAL JOIN Seats
      WHERE Flights.flight_id = flight_id_arg
        AND Seats.seat_no = seat_no_arg
        AND action_timestamp_arg + interval_arg < Flights.flight_time
        AND (NOT Flights.closed_by_administrator)
    );
END;
$$
  LANGUAGE plpgsql;

-- Проверяет, что место не куплено.
CREATE OR REPLACE FUNCTION
  check_not_bought(flight_id_arg INT,
                   seat_no_arg INT)
  RETURNS BOOLEAN AS
$$
BEGIN
  RETURN NOT EXISTS(
      SELECT *
      FROM Bought
      WHERE Bought.flight_id = flight_id_arg
        AND Bought.seat_no = seat_no_arg
```

```
    );
END;
$$
  LANGUAGE plpgsql;

-- Выполняет бронирование места или продление брони.
-- Если есть истёкшие брони на данное место на данный рейс,
-- удаляет их из таблицы. Гарантируется, что указанный рейс существует,
-- данное место есть в самолёте, совершающем данный рейс, и не куплено.
-- Если броней на данное место нет, создаёт новую.
-- Если есть бронь данного человека на данное место, обновляет её.
-- Возвращает TRUE, если бронь удалось продлить или создать,
-- FALSE иначе (если данное место забронировано другим человеком).
CREATE OR REPLACE FUNCTION
  do_reservation(flight_id_arg INT,
                 seat_no_arg INT,
                 reservation_timestamp_arg TIMESTAMP,
                 user_id_arg INT)
  RETURNS BOOLEAN AS
$$
DECLARE
  user_that_reserved INT;
BEGIN
  DELETE
  FROM Reservation
  WHERE Reservation.flight_id = flight_id_arg
    AND Reservation.seat_no = seat_no_arg
    AND Reservation.reservation_timestamp
          + INTERVAL '1 Day' < reservation_timestamp_arg;

  SELECT Reservation.user_id INTO user_that_reserved
  FROM Reservation
  WHERE Reservation.flight_id = flight_id_arg
    AND Reservation.seat_no = seat_no_arg;

  IF user_that_reserved IS NULL THEN
    INSERT INTO Reservation(flight_id,
                            seat_no,
                            reservation_timestamp,
                            user_id)
    VALUES (flight_id_arg,
            seat_no_arg,
            reservation_timestamp_arg,
            user_id_arg);

    RETURN TRUE;
  ELSEIF user_that_reserved = user_id_arg THEN
    UPDATE Reservation
    SET reservation_timestamp = reservation_timestamp_arg
    WHERE Reservation.flight_id = flight_id_arg
      AND Reservation.seat_no = seat_no_arg;

    RETURN TRUE;
  ELSE
    RETURN FALSE;
  END IF;
```

```
END;
$$
  LANGUAGE plpgsql;

-- Попробовать создать новое или продлить старое бронирование для места seat_no_arg
-- на рейсе flight_id_arg для пользователя user_id_arg.
-- Для этого проверяется, что место существует, не куплено
-- и не забронировано, на рейс не закрыта продажа
-- администратором (иначе нет смысла бронировать)
-- а бронирование доступно (должно быть больше суток до рейса).
-- Возвращает TRUE, если удалось создать бронирование, FALSE иначе.
CREATE OR REPLACE FUNCTION
  create_or_update_reservation(flight_id_arg INT,
                               seat_no_arg INT,
                               user_id_arg INT) RETURNS BOOLEAN AS
$$
DECLARE
  reservation_timestamp TIMESTAMP := now();
BEGIN
  IF seat_exists_and_can_be_processed(flight_id_arg,
                                      seat_no_arg,
                                      reservation_timestamp,
                                      INTERVAL '1 Day') AND
     check_not_bought(flight_id_arg, seat_no_arg) THEN
    RETURN do_reservation(
        flight_id_arg,
        seat_no_arg,
        reservation_timestamp,
        user_id_arg
      );
  ELSE
    RETURN FALSE;
  END IF;
END;
$$
  LANGUAGE plpgsql;

-- Запрещает покупку мест на рейс flight_id_arg по запросу администратора
CREATE OR REPLACE PROCEDURE close_buy(flight_id_arg INT) AS
$$
BEGIN
  UPDATE Flights
  SET closed_by_administrator = TRUE
  WHERE flight_id = flight_id_arg;
END;
$$
  LANGUAGE plpgsql;

-- Выполняет покупку места.
-- Если есть истёкшие брони на данное место на данный рейс,
-- удаляет их из таблицы. Гарантируется, что указанный рейс существует,
-- данное место есть в самолёте, совершающем данный рейс, и не куплено.
-- Если есть действительная бронь данного места данным человеком,
-- удаляет её из таблицы бронирования.
-- Возвращает TRUE, если бронь удалось купить,
-- FALSE иначе (если данное место забронировано другим человеком).
```

```
CREATE OR REPLACE FUNCTION
  do_buy(flight_id_arg INT,
         seat_no_arg INT,
         buy_timestamp_arg TIMESTAMP,
         user_id_arg INT)
  RETURNS BOOLEAN AS
$$
DECLARE
  user_that_reserved INT;
BEGIN
  DELETE
  FROM Reservation
  WHERE Reservation.flight_id = flight_id_arg
    AND Reservation.seat_no = seat_no_arg
    AND Reservation.reservation_timestamp
          + INTERVAL '1 Day' < buy_timestamp_arg;

  SELECT Reservation.user_id INTO user_that_reserved
  FROM Reservation
  WHERE Reservation.flight_id = flight_id_arg
    AND Reservation.seat_no = seat_no_arg;

  IF user_that_reserved IS NULL THEN
    INSERT INTO Bought(flight_id, seat_no, user_id)
    VALUES (flight_id_arg, seat_no_arg, user_id_arg);

    RETURN TRUE;
  ELSEIF user_that_reserved = user_id_arg THEN
    DELETE
    FROM Reservation
    WHERE reservation.flight_id = flight_id_arg
      AND Reservation.seat_no = seat_no_arg;

    INSERT INTO Bought(flight_id, seat_no, user_id)
    VALUES (flight_id_arg, seat_no_arg, user_id_arg);

    RETURN TRUE;
  ELSE
    RETURN FALSE;
  END IF;
END;
$$
  LANGUAGE plpgsql;


-- Попробовать купить место seat_no_arg
-- на рейсе flight_id_arg для пользователя user_id_arg.
-- Для этого проверятся, что место существует, не куплено
-- и не забронировано другим пользователем, на рейс не закрыта продажа
-- администратором,
-- а покупка доступна (должно быть больше двух часов до рейса).
-- Если существует бронирование данного места, выполненное данным человеком,
-- удаляет его из таблицы бронирований
-- (т.к.  нет смысла держать бронь, если место куплено)
-- Возвращает TRUE, если удалось купить, FALSE иначе.
CREATE OR REPLACE FUNCTION
```

```
  buy(flight_id_arg INT,
      seat_no_arg INT,
      user_id_arg INT) RETURNS BOOLEAN AS
$$
DECLARE
  buy_timestamp TIMESTAMP := now();
BEGIN
  IF seat_exists_and_can_be_processed(flight_id_arg,
                                      seat_no_arg,
                                      buy_timestamp,
                                      INTERVAL '2 Hours') AND
     check_not_bought(flight_id_arg, seat_no_arg) THEN
    RETURN do_buy(flight_id_arg, seat_no_arg, buy_timestamp, user_id_arg);
  ELSE
    RETURN FALSE;
  END IF;
END;
$$
  LANGUAGE plpgsql;
```

3 Добавление индексов

Потенциально часто используемыми операциями являются:

1. Естественное объединение Flights и Seats (используя plane_id). Выполяется по plane_id, который является префиксом первичного ключа Seats. В PostgreSQL для первичных ключей автоматически добавляется B-tree индекс, а значит, поиск подходящих записей в таблице Seats будет осуществляться быстро, так как с использованием B-tree индекса мы умеем быстро искать по префиксу индексного набора полей (в данном случае - plane_id)

2. Выборка из таблицы Flights по критерий равенства определённому flight_id. flight_id - первичный ключ таблицы Flights, по причинам, аналогичным описанным выше, запрос будет исполняться эффективно с использованием стандартных индексов PostgreSQL, которые СУБД создаёт по умолчанию.

3. Выборка конкретного места seat_no, но только в определённом самолёте с определённым plane_id. Для этого индекс не нужен, так как число мест даже в самых больших самолётах не превышает нескольких сотен. Если же понадобится, индекс можно добавить командой

   ```
   CREATE INDEX seats_idx ON Seats USING btree (seat_no, plane_id);
   ```

4. Выборки из таблиц Reservation и Bought по паре (flight_id, seat_id) выполняются по первичному ключу, и, в силу описаннх выше причин, делаются быстро с использованием индексов, создаваемых СУБД по умолчанию.

Обращения к closed_by_administrator и reservation_timestamp всегда выполняются только к одной записи (выбранной с помощью первичного ключа, и потому единственной) и не являются "горячими"операциями.

4 Пусть частым запросом является определение средней заполненности самолёта по рейсу. Какие индексы могут помочь при исполнении данного запроса?

Запрос можно написать, например, так:

```sql
-- Средняя заполненность самолёта по рейсу (среднее количество мест, которые
-- были куплены в самолёте с определённым plane_id)s
WITH FlightsWithBoughtCount AS (
  SELECT Flights.flight_id,
         count(Bought.seat_no) AS bought_count
  FROM Flights
         LEFT OUTER JOIN Bought USING (flight_id)
  WHERE Flights.plane_id = ?
  GROUP BY Flights.flight_id
)
SELECT avg(FlightsWithBoughtCount.bought_count)
FROM FlightsWithBoughtCount;
```

Быстро выполнять соединение Flights и Bought используя flight_id нам поможет B-tree индекс на flight_id в обеих соединяемых таблицах, который будет создан СУБД автоматически (потому что flight_id - первичный ключ в Flights и префикс первичного ключа в Bought.)

Аналогично, СУБД автоматически создаст индекс по flight_id, который позволит быстро делать группировку.

Но нам нужно быстро делать выборку из таблицы Flights по plane_id. Для этого добавим следующий индекс:

```sql
CREATE INDEX plane_id_idx ON Flights USING hash (plane_id);
```