Филиппов Дмитрий, M3439

Домашнее задание 8.

Схема БД: Flights(FlightId, FligtTime, PlaneId), Seats(PlaneId, SeatNo).
Использованная БД: PostgreSQL 9.4.5.

1. **Дополните эту базу данных, так что бы она поддерживала бронирование и покупку мест. При этом, бронь должна автоматически сниматься по тайм-ауту. При этом, должны поддерживаться следующие свойства:**

- Одно место не может быть продано или забронировано более чем один раз (в том числе, продано и забронировано одновременно).
- Бронь можно обновить, после чего она будет действительна ещё одни сутки.
- Бронь автоматически снимается через сутки после последнего обновления, но не позже, чем за сутки, до вылета рейса.
- Бронирование автоматически закрывается за сутки до времени рейса.
- Продажи автоматически закрываются не позднее двух часов до вылета рейса, либо при распродаже всех мест либо по запросу администратора.

Дополним базы данных *Airline* следующим образом:

- Добавим таблицу $Transaction$, содержающую информацию о покупке/бронировании места;
- А именно, в ней мы будем хранить:
  - информацию о полете — $FlightId$, $PlaneId$, которые также будут ссылаться на таблицу $Flights$;
  - информацию о месте — $PlaneId$, $SeatNo$, которые также будут ссылаться на таблицу $Seats$;
  - информацию о клиенте, покупающем/бронирующем место: серия и номер паспорта $PassportSeries$, $PassportNo$;
  - время и тип транзакции — $TransTime$, $TransType$, где $TransType$ — либо бронирование, либо покупка.
- Также нам нужно поддерживать возможность закрытия продаж по запросу администратора, для этого в таблицу $Flights$ добавим флаг $ClosedByRequest$.

Для снятия брони по тайм-ауту и поддержания всех вышеописанных свойств будем использовать триггеры, у нас есть вся информация для проверки всех этих условий, а значит все ок. Описание триггеров на SQL будет сделано ниже.

Все будет устроено следующим образом:

- Бронь не будет автоматически сниматься по тайм-ауту (не будет listener'а, который это делает), однако просто при каждом действии мы будем проверять, не прошел ли тайм-аут с последнего бронирования (или до вылета осталось мало времени), и если прошел, считаем, что бронь снялась;

- Для проверки, что одно место не продано и не забронировано несколько раз проверяем при покупке/бронировании таблицу *Transactions* и смотрим, нет ли там уже покупки/брони с неистекшим тайм-аутом на место, которое мы сейчас хотим купить;
- Для обновления брони опять же проверим, что она все еще действительна, если нет, будет считать, что обновить ее нельзя, надо делать новую, это другой запрос;
- Автоматическое снятие и закрытие брони/продажи, как писалось выше, формально мы делать не будем, однако везде будем это проверять;
- Закрытие при распродаже всех мест можно не обрабатывать — покупку отменить нельзя, поэтому уже ничего не изменится и не испортится.

2. **Запишите определение базы данных Airline на языке SQL.**
Определение таблиц и данных в них:

```
DROP DATABASE IF EXISTS airline;
CREATE DATABASE airline;
\c airline;

CREATE TABLE Flights (
  FlightId INT NOT NULL,
  FlightTime TIMESTAMP DEFAULT '1970-01-01␣00:00:01',
  PlaneId INT NOT NULL,
  ClosedByRequest BOOLEAN DEFAULT FALSE,
  PRIMARY KEY (FlightId, PlaneId)
);

CREATE TABLE Seats (
  PlaneId INT NOT NULL,
  SeatNo INT NOT NULL,
  PRIMARY KEY (PlaneId, SeatNo)
);

CREATE TABLE Transaction (
  FlightId INT NOT NULL,
  PlaneId INT NOT NULL,
  SeatNo INT NOT NULL,
  PassportSeries INT DEFAULT NULL,
  PassportNo INT DEFAULT NULL,
  TransTime TIMESTAMP DEFAULT now(),
  TransType INT DEFAULT 0,
  //0 - booking, 1 - bying
  //comment should be defined with two dashes,
  //but I have troubles with latex then :(
  FOREIGN KEY (FlightId, PlaneId) REFERENCES Flights,
  FOREIGN KEY (PlaneId, SeatNo) REFERENCES Seats
  ON DELETE CASCADE
  ON UPDATE CASCADE,
  PRIMARY KEY (FlightId, PlaneId, SeatNo)
);
```

Триггеры для проверки условий:

Функция для проверки, что бронь больше недоступна:

```
CREATE FUNCTION isBookingExpired(flightId INT, planeId INT, seatNo INT)
RETURNS BOOLEAN AS $BODY$
  DECLARE flightTime TIMESTAMP;
  DECLARE lastBookingUpdate TIMESTAMP;
  DECLARE isClosedByRequest BOOLEAN DEFAULT FALSE;
BEGIN
  SELECT (F.FlightTime, F.ClosedByRequest) INTO
    flightTime, isClosedByRequest
  FROM
    Flights as F
  WHERE
    F.FlightId = flightId AND
    F.PlaneId = planeId;

  SELECT T.TransTime INTO
    lastBookingUpdate
  FROM
    Transaction as T
  WHERE
    T.FlightId = flightId AND
    T.PlaneId = planeId AND
    T.SeatNo = seatNo AND
    T.TransType = 0;

  RETURN isClosedByRequest OR
         (lastBookingUpdate IS NOT NULL AND
         lastBookingUpdate + INTERVAL '24␣hours' < NOW()) OR
         NOW() + INTERVAL '24␣hours' > flightTime;
END;
$BODY$ LANGUAGE plpgsql;
```

Функция для обновления текущей брони (если бронь больше не доступна, она удаляется):

```
CREATE FUNCTION updateBooking()
RETURNS TRIGGER AS $BODY$
  DECLARE flightId INT;
  DECLARE planeId INT;
  DECLARE seatNo INT;
BEGIN
  flightId := NEW.FlightId;
  planeId := NEW.PlaneId;
  seatNo := NEW.SeatNo;
  IF isBookingExpired(flightId, planeId, seatNo) THEN
    //Booking is expired, we cannot update it
    DELETE FROM
      Transaction as T
    WHERE
      T.FlightId = flightId AND
```

```
      T.PlaneId = planeId AND
      T.SeatNo = seatNo AND
      T.TransType = 0;
    RETURN NEW;
  ELSE
    UPDATE
      Transaction as T
    SET
      T.TransTime = now()
    WHERE
      T.FlightId = flightId AND
      T.PlaneId = planeId AND
      T.SeatNo = seatNo AND
      T.TransType = 0;
    RETURN NEW;
  END IF;
END;
$BODY$ LANGUAGE plpgsql;
```

Процедура покупки места:

```
CREATE FUNCTION buySeat()
RETURNS TRIGGER AS $BODY$
  DECLARE flightId INT;
  DECLARE planeId INT;
  DECLARE seatNo INT;
  DECLARE passportSeries INT;
  DECLARE passportNo INT;
  DECLARE flightTime TIMESTAMP;
  DECLARE isClosedByRequest BOOLEAN DEFAULT FALSE;
  DECLARE nowPassportSeries INT DEFAULT NULL;
  DECLARE nowPassportNo INT DEFAULT NULL;
  DECLARE transType INT DEFAULT NULL;
BEGIN
  flightId := NEW.FlightId;
  planeId := NEW.PlaneId;
  seatNo := NEW.SeatNo;
  passportSeries := NEW.PassportSeries;
  passportNo := NEW.PassportNo;
  SELECT (F.FlightTime, F.isClosedByRequest) INTO
    flightTime, isClosedByRequest
  FROM
    Flights as F
  WHERE
    F.FlightId = flightId AND
    F.PlaneId = planeId;

  IF (NOT F.isClosedByRequest AND NOW() <=
     flightTime - INTERVAL '2␣hours') THEN
    SELECT (T.PassportSeries, T.PassportNo, T.TransType) INTO
      nowPassportSeries, nowPassportNo, transType
```

```
        FROM
          Transaction as T
        WHERE
          T.FlightId = flightId AND
          T.PlaneId = planeId AND
          T.SeatNo = seatNo;
        IF (nowPassportSeries IS NULL AND
            nowPassportNo IS NULL) OR
           (transType = 0 AND
            nowPassportSeries = passportSeries AND
            nowPassportNo = passportNo) OR
           (transType = 0 AND
            isBookingExpired(flightId, planeId, seatNo)) THEN
          DELETE FROM
            Transaction as T
          WHERE
            T.FlightId = flightId AND
            T.PlaneId = planeId AND
            T.SeatNo = seatNo;

          INSERT INTO Transaction
            (FlightId, PlaneId, SeatNo,
             PassportSeries, PassportNo, TransType)
          VALUES
            (flightId, planeId, seatNo,
             passportSeries, passportNo, 1);
          RETURN NEW;
        ELSE
          //seat it already bought or booked, can't␣buy␣it
␣␣␣␣␣␣␣␣␣␣RETURN␣OLD;
␣␣␣␣␣␣␣␣END␣IF;
␣␣␣␣␣␣␣ELSE
␣␣␣␣␣␣␣␣//bying␣is␣closed␣or␣it␣is␣too␣late
␣␣␣␣␣␣␣␣RETURN␣OLD;
␣␣␣␣␣␣␣END␣IF;
␣␣␣␣END;
␣␣␣␣$BODY$␣LANGUAGE␣plpgsql;
```

Процедура бронирования места:

```
  CREATE FUNCTION bookSeat()
  RETURNS TRIGGER AS $BODY$
    DECLARE flightId INT;
    DECLARE planeId INT;
    DECLARE seatNo INT;
    DECLARE passportSeries INT;
    DECLARE passportNo INT;
    DECLARE flightTime TIMESTAMP;
    DECLARE isClosedByRequest BOOLEAN DEFAULT FALSE;
    DECLARE nowPassportSeries INT DEFAULT NULL;
    DECLARE nowPassportNo INT DEFAULT NULL;
```

```
  DECLARE transType INT DEFAULT NULL;
BEGIN
  flightId := NEW.FlightId;
  planeId := NEW.PlaneId;
  seatNo := NEW.SeatNo;
  passportSeries := NEW.PassportSeries;
  passportNo := NEW.PassportNo;
  SELECT (F.FlightTime, F.isClosedByRequest) INTO
    flightTime, isClosedByRequest
  FROM
    Flights as F
  WHERE
    F.FlightId = flightId AND
    F.PlaneId = planeId;

  IF (NOT F.isClosedByRequest AND NOW() <=
      flightTime - INTERVAL '24 hours') THEN
    SELECT (T.PassportSeries, T.PassportNo, T.TransType) INTO
      nowPassportSeries, nowPassportNo, transType
    FROM
      Transaction as T
    WHERE
      T.FlightId = flightId AND
      T.PlaneId = planeId AND
      T.SeatNo = seatNo;
    IF (nowPassportSeries IS NULL AND
        nowPassportNo IS NULL) OR
       (transType = 0 AND
        isBookingExpired(flightId, planeId, seatNo)) THEN
      DELETE FROM
        Transaction as T
      WHERE
        T.FlightId = flightId AND
        T.PlaneId = planeId AND
        T.SeatNo = seatNo;

      INSERT INTO Transaction
        (FlightId, PlaneId, SeatNo, PassportSeries, PassportNo, TransType)
      VALUES
        (flightId, planeId, seatNo, passportSeries, passportNo, 0);
      RETURN NEW;
    ELSE
      //seat it already booked
      //(maybe by yourself, but it doesn't matter), can't buy it
      RETURN OLD;
    END IF;
  ELSE
    //booking is closed or it is too late
    RETURN OLD;
  END IF;
```

```
END;
$BODY$ LANGUAGE plpgsql;
```

Теперь напишем триггеры для обновления базы данных:

```
CREATE TRIGGER BuySeatTrigger
AFTER INSERT OR UPDATE ON Transaction
FOR EACH ROW EXECUTE PROCEDURE buySeat();

CREATE TRIGGER BookSeatTrigger
AFTER INSERT OR UPDATE ON Transaction
FOR EACH ROW EXECUTE PROCEDURE bookSeat();

CREATE TRIGGER UpdateBookingTrigger
AFTER UPDATE ON Transaction
FOR EACH ROW EXECUTE PROCEDURE updateBooking();
```

3. **Определите, какие индексы требуется добавить к таблицам базы данных Airline.**
Добавление этих индексов представлено в пункте 5.

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

Хеш-индекс по $PlaneId$ в таблице $Seats$ для быстрого получения количества мест в самолете.

Хеш-индекс по $FlightId$ в таблице $Transactions$ для быстрого получения количества человек, летящих данным рейсом.

5. **Запишите добавление индексов на языке SQL.**
Индексы из пункта 4:

- CREATE INDEX SeatsIndex ON Seats USING HASH (PlaneId);
- CREATE INDEX FlightsIndex ON Flights USING HASH (FlightId);

Индексы из пункта 3:

- CREATE INDEX TransactionsIndex ON Transactions USING HASH (FlightId);
- CREATE INDEX TransactionsSeatIndex ON Transactions USING BTREE (FlightId, SeatNo);
- CREATE UNIQUE INDEX FlightsBTIndex ON Flights USING BTREE (FlightId, FlightTime);