ДЗ 8

Содержание

Весь sql-код написан и протестирован на `PostgreSQL 9.5.4`.

1 Проектирование базы Airlines

Для начала определимся с таблицами, которые уже предоставлены (`Flights(FlightId, FligtTime, PlaneId), Seats(PlaneId, SeatNo)`) и создадим их. Добавим поле `SoldOut` в `Flights`, что будет семантически соответствовать требованию "продажи автоматически закрываются не позднее двух часов до вылета рейса, либо при распродаже всех мест либо по запросу администратора".

```
DROP TABLE IF EXISTS Seats;
CREATE TABLE Seats(
    PlaneId int NOT NULL PRIMARY KEY,
    SeatNum int NOT NULL
);

DROP TABLE IF EXISTS Flights;
CREATE TABLE Flights(
  FlightId int NOT NULL PRIMARY KEY,
  FlightTime timestamp NOT NULL,
  PlaneId int NOT NULL,
  SoldOut boolean NOT NULL,

  FOREIGN KEY (PlaneId) REFERENCES Seats(PlaneId) ON DELETE CASCADE
);
```

Для бронирования и покупки мест создадим следующую физическую модель: хранить билеты будем в таблице `Tickets`, представляя бронирования как неоплаченные билеты. Это поможет предоставить гарантию того, что место не может быть купленым и забронированным одновременно.

```
DROP TABLE IF EXISTS Tickets;
CREATE TABLE Tickets(
    UserId int NOT NULL,
    FlightId int NOT NULL,
    SeatId int NOT NULL,
    Expiration timestamp,
    IsPaid boolean NOT NULL DEFAULT False,

    PRIMARY KEY (UserId, FlightId, SeatId),
    FOREIGN KEY (FlightId) REFERENCES Flights(FlightId) ON DELETE CASCADE,
    UNIQUE(FlightId, SeatId) -- место не может быть продано/забронировано дважды
);
```

Заполним таблицу тестовыми данными:

```
DELETE FROM Seats WHERE True;
INSERT INTO Seats VALUES
```

```
  (111, 10),
  (222, 15),
  (333, 20);

DELETE FROM Flights WHERE True;
INSERT INTO Flights VALUES
  (0, '2016-11-29 12:00:00', 111, False),
  (1, '2016-11-29 22:00:00', 222, False),
  (2, '2016-11-30 06:30:00', 333, False),
  (3, '2016-11-28 06:30:00', 222, True);

DELETE FROM Tickets WHERE True;
INSERT INTO Tickets VALUES
  (0,0,2,null,True),
  (1,0,3,'2016-11-22 08:00:00',False),
  (2,2,5,'2016-11-29 12:00:00',False);
```

Объявим список представлений, которые помогут в составлении запросов в дальнейшем:

```
-- Flights we can buy tickets on.
DROP VIEW IF EXISTS ValidFlights;
CREATE VIEW ValidFlights AS
SELECT FlightId, PlaneId FROM Flights
WHERE NOT SoldOut AND
      FlightTime - now() > INTERVAL '2 hour';

-- (planeId, seatId) for every accessible seat in the plane
-- let's suppose there can't be more than 100 seats in plane
-- (we can use 1000 instead)
DROP VIEW IF EXISTS AllSeats;
CREATE VIEW AllSeats AS
SELECT S.PlaneId, n as SeatId
FROM generate_series(1,100) AS a(n), Seats AS S
WHERE n <= S.SeatNum
ORDER BY S.PlaneId;

-- Seats for valid flights that are bought or booked. Intuitively
-- for all flights that are accessible, seats for them that
-- can't be booked right now.
DROP VIEW IF EXISTS NonFreeSeats;
CREATE VIEW NonFreeSeats AS
SELECT FlightId, SeatId
FROM ValidFlights NATURAL JOIN Flights NATURAL JOIN Tickets
WHERE IsPaid OR -- payed tickets
      (Expiration is not null AND
       now() < Expiration AND
       FlightTime - Expiration > INTERVAL '1 day');

-- Seats that we can book or buy!
DROP VIEW IF EXISTS FreeSeats;
CREATE VIEW FreeSeats AS
SELECT FlightId, SeatId
FROM ValidFlights NATURAL JOIN AllSeats
WHERE (FlightId,SeatId) NOT IN (SELECT * from NonFreeSeats);
```

2 Запросы к базе

1. Покупка билета.

```
INSERT INTO Tickets (UserId, FlightId, SeatId, IsPaid)
SELECT 5,2,8,True -- some user data
ON CONFLICT (FlightId,SeatId) DO UPDATE SET IsPaid = True
```

```
WHERE EXISTS (
  SELECT * FROM FreeSeats
  WHERE FlightId = 2 AND SeatId = 8);
```

При ошибке будет вставлено 0 рядов, что как бы намекает. Аналогично и дальше.

2. Бронирование/продление бронирования

```
INSERT INTO Tickets (UserId, FlightId, SeatId, Expiration, IsPaid)
SELECT 6,3,4,now() + INTERVAL '1 day',False
ON CONFLICT (FlightId,SeatId) DO
  UPDATE SET Expiration = now() + INTERVAL '1 day'
WHERE EXISTS (
  SELECT * FROM FreeSeats
  WHERE FlightId = 2 AND SeatId = 8);
```

3. Оплата брони.

```
UPDATE Tickets
SET IsPaid = True
WHERE UserId = 5 AND
      FlightId = 0 AND
      SeatId = 3 AND
      EXISTS (SELECT * FROM FreeSeats
              WHERE FlightId = 0 AND SeatId = 3);
```

4. Реквест администратора заморозить продажи.

```
UPDATE Flights SET SoldOut = True WHERE FlightId = 123;
```

5. Запросы на проверки мест тривиально выводятся из представлений в первой части.

3 Индексы

Определим потенциально частые запросы:

1. `Flights`: поиск рейсов по времени – поиск по временным промежуткам, упорядоченный индекс на (`FlightTime`). Индексы по `PRIMARY KEY` автоматически есть.
2. `Tickets`:
   - Список заказов пользователя по его ID: хэш индекс по `UserId`.
   - По id рейса и месту билеты/брони на него. Упорядоченный индекс – ускорятся как запросы "все места по этому рейсу" так и "конкретное место конкретного рейса".
3. `Planes`: не вижу необходимости оптимизировать. Обычно самолетный парк не то, чтобы очень большой, поэтому запросы должны быть и так быстрыми. Но если очень нужно, то хэш индекс по номеру самолета или id самолета (номера/модели самолета нету в моем определении, но мог бы быть).

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

Во-первых нужно в таблицу `Flights` добавить `FlightNum`, который будет семантически соответствовать именно рейсу, а не id. Дальше на нем построить хэш индекс. Также помогут уже упомянутые индексы на `Tickets`, и, возможно, `Planes`, поскольку придется искать все свободные места и их количество, а также сортировать по занятости через `Tickets`. Формально данный запрос будет опираться на использование

представления `FreeSeats`, которое, в свою очередь, опирается на select'ы из, преимущественно, `Tickets`. В реальной имплементации я напишу хэш индекс на `FlightId`, существенной разницы в смысле целей обучения нет.

```
CREATE INDEX ON Flights USING btree (FlightTime);
CREATE INDEX ON Tickets USING hash (UserId);
CREATE INDEX ON Tickets USING btree (FlightId,SeatId);
```

Автор: Михаил Волхов M3438

Created: 2016-11-29 Tue 13:15

Validate