

# Lab\_3\_Nemkov\_KMBO\_05\_23

## М.1. Установка

Демонстрационная база данных доступна на [edu.postgrespro.ru](http://edu.postgrespro.ru) в трёх версиях, которые отличаются только объёмом данных:

- [demo-small.zip](#) (21 МБ) — данные по полётам за один месяц (размер БД около 300 МБ)
- [demo-medium.zip](#) (62 МБ) — данные по полётам за три месяца (размер БД около 700 МБ)
- [demo-big.zip](#) (232 МБ) — данные по полётам за год (размер БД около 2,5 ГБ)

*image-173.png*

С прошлого семестра у меня стоит demo-big, думаю, ничего страшного, если я продолжу использовать её. Только результаты моих запросов, вероятно, будут по количеству строк больше, чем на demo-medium.

## Task 1.

Напишите функцию, возвращающую все данные, относящиеся к одному бронированию, номер которого задается в аргументах.

```
CREATE OR REPLACE FUNCTION first_task(input_book_ref text)
RETURNS TABLE(
    book_ref text,
    book_date timestamp with time zone,
    total_amount numeric(10,2),
    ticket_no text,
    passenger_id varchar(20),
    passenger_name text,
    contact_data text,
    flight_id int,
    fare_conditions varchar(10),
    flight_amount numeric(10,2),
    boarding_no int,
    seat_no varchar(4),
    flight_no text,
    scheduled_departure timestamp with time zone,
    scheduled_arrival timestamp with time zone,
    departure_airport text,
    arrival_airport text,
```

```

        status varchar(20),
        aircraft_code text,
        actual_departure timestamp with time zone,
        actual_arrival timestamp with time zone
    )
AS
$$
BEGIN
    RETURN QUERY
    SELECT
        -- bookings
        b.book_ref::text AS book_ref,
        b.book_date,
        b.total_amount,
        -- tickets
        t.ticket_no::text AS ticket_no,
        t.passenger_id,
        t.passenger_name,
        t.contact_data::text AS contact_data,
        -- ticket_flights
        tf.flight_id,
        tf.fare_conditions,
        tf.amount AS flight_amount,
        -- boarding_passes
        bp.boarding_no,
        bp.seat_no,
        -- flights
        f.flight_no::text AS flight_no,
        f.scheduled_departure,
        f.scheduled_arrival,
        f.departure_airport::text AS departure_airport,
        f.arrival_airport::text AS arrival_airport,
        f.status,
        f.aircraft_code::text AS aircraft_code,
        f.actual_departure,
        f.actual_arrival
    FROM bookings b
    JOIN tickets t ON b.book_ref = t.book_ref
    JOIN ticket_flights tf ON t.ticket_no = tf.ticket_no
    JOIN flights f ON tf.flight_id = f.flight_id
    -- (могут быть пустые значения, поэтому лучше left join наверное
    -- чтобы не потерялись записи без посадочных талонов)
    LEFT JOIN boarding_passes bp ON tf.ticket_no = bp.ticket_no

```

```

AND tf.flight_id = bp.flight_id

WHERE b.book_ref::text = p_book_ref;
END;
$$
LANGUAGE plpgsql;

```

## Result:

Возьму несколько первых бронирований, чтобы посмотреть, как оно работает

```
SELECT * FROM bookings LIMIT 5
```

	book_ref [PK] character (6)	book_date timestamp with time zone	total_amount numeric (10,2)
1	000004	2016-08-13 15:40:00+03	55800.00
2	00000F	2017-07-05 03:12:00+03	265700.00
3	000010	2017-01-08 19:45:00+03	50900.00
4	000012	2017-07-14 09:02:00+03	37900.00
5	000026	2016-08-30 11:08:00+03	95600.00

image-174.png

```
SELECT * FROM get_booking_data('000004');
```

На выходе получаю вот такой вот результат, вроде выглядит адекватно: Петр Макаров в одной брони указал билеты туда-обратно

book_ref	book_date	total_ammou	ticket_no	passenger_passenger_name	contact_data	flight_id	fare_conditi	flight_ammou	boarding_nc	seat_no	flight_no	scheduled_s	scheduled_s	departure_s	arrival_a	airpc	status	aircraft_cod	actual_deps	actual_arrival
4	2016-08-13	55800.00	5434683522	4711 08409 PETR MAKAROV	("email": "p_makarov")	63787	Economy	27900.00		127 29F	PG0223	2016-09-07	2016-09-07	OVB	DME	Arrived		773	2016-09-07	2016-09-07 15:16:00+03
4	2016-08-13	55800.00	5434683522	4711 08409 PETR MAKAROV	("email": "p_makarov")	1865	Economy	27900.00		373 50E	PG0222	2016-08-29	2016-08-29	DME	OVB	Arrived		773	2016-08-29	2016-08-29 14:31:00+03

image-175.png

```
SELECT * FROM get_booking_data('000010');
```

Вот пример еще одного вызова функции, где мы видим в одном бронировании двух человек, которые судя по всему полетели в командировку "Уфа - Ханты-Мансийск"))

book_ref	book_date	total_ammou	ticket_no	passenger_passenger_name	contact_data	flight_id	fare_conditi	flight_ammou	boarding_nc	seat_no	flight_no	scheduled_s	scheduled_s	departure_s	arrival_a	airpc	status	aircraft_cod	actual_deps	actual_arrival
10	2017-01-08	50900.00	5432295360	0564 04430 LYUDMILA ("email": "bc		11943	Economy	16200.00		2 14A	PG0541	2017-01-22	2017-01-22	DME	OVS	Arrived	SU9	2017-01-22	2017-01-22 13:28:00+03	
10	2017-01-08	50900.00	5432295359	5722 83725 ALEKSAND ("email": "sc		11943	Economy	16200.00		5 15C	PG0541	2017-01-22	2017-01-22	DME	OVS	Arrived	SU9	2017-01-22	2017-01-22 13:28:00+03	
10	2017-01-08	50900.00	5432295360	0564 04430 LYUDMILA ("email": "bc		147968	Economy	8800.00		1 22B	PG0568	2017-01-22	2017-01-22	OVS	UFA	Arrived	CR2	2017-01-22	2017-01-22 17:37:00+03	
10	2017-01-08	50900.00	5432295359	5722 83725 ALEKSAND ("email": "sc		147968	Economy	9700.00		2 1A	PG0568	2017-01-22	2017-01-22	OVS	UFA	Arrived	CR2	2017-01-22	2017-01-22 17:37:00+03	

image-176.png

## Task 2

Напишите функцию find\_flights (departure CHAR(3), arrival CHAR(3)), которая возвращает список рейсов между двумя аэропортами.

```
CREATE OR REPLACE FUNCTION find_flights(  
    departure CHAR(3),  
    arrival CHAR(3)  
)  
RETURNS TABLE(  
    flight_id int,  
    flight_no char(6),  
    scheduled_departure timestamp with time zone,  
    scheduled_arrival timestamp with time zone,  
    departure_airport char(3),  
    arrival_airport char(3),  
    status character varying(20),  
    aircraft_code char(3),  
    actual_departure timestamp with time zone,  
    actual_arrival timestamp with time zone  
)  
AS $$  
SELECT  
    flight_id,  
    flight_no,  
    scheduled_departure,  
    scheduled_arrival,  
    departure_airport,  
    arrival_airport,  
    status,  
    aircraft_code,  
    actual_departure,  
    actual_arrival  
FROM flights  
WHERE departure_airport = departure  
AND arrival_airport = arrival;  
$$ LANGUAGE sql;
```

## Result

```
SELECT * FROM find_flights('SVO', 'CSY');
```

## Посмотрим рейсы "Шереметьево - Чебоксары"

	flight_id integer	flight_no character	scheduled_departure timestamp with time zone	scheduled_arrival timestamp with time zone	departure_airport character	arrival_airport character	status character varying	aircraft_code character
1	47691	PG0241	2017-04-09 09:40:00+03	2017-04-09 10:30:00+03	SVO	CSY	Arrived	SU9
2	47692	PG0241	2017-04-08 09:40:00+03	2017-04-08 10:30:00+03	SVO	CSY	Arrived	SU9
3	47693	PG0241	2016-08-16 09:40:00+03	2016-08-16 10:30:00+03	SVO	CSY	Arrived	SU9
4	47694	PG0241	2017-09-14 09:40:00+03	2017-09-14 10:30:00+03	SVO	CSY	Scheduled	SU9
5	47695	PG0241	2016-08-17 09:40:00+03	2016-08-17 10:30:00+03	SVO	CSY	Arrived	SU9

Total rows: 396 of 396    Query complete 00:00:00.188    Ln 35, Col 1

*image-177.png*

```
SELECT * FROM find_flights('SVO', 'VKO');
```

Рейсов "Шереметьево - Внуково" нет, что вполне логично - функция вернула 0 строк

## Task 3

Создайте функцию `count_tickets_by_class(flight_id INT, fare_class VARCHAR)`, которая считает, сколько билетов продано в данном классе (Economy, Business, First).

```
CREATE OR REPLACE FUNCTION count_tickets_by_class(  
    p_flight_id INT,  
    p_fare_class VARCHAR  
)  
RETURNS INT  
AS $$  
DECLARE  
    our_counter INT;  
BEGIN  
    SELECT COUNT(*)  
    INTO our_counter  
    FROM ticket_flights  
    WHERE flight_id = p_flight_id  
    AND fare_conditions = p_fare_class;  
  
    RETURN our_counter;  
END;  
$$ LANGUAGE plpgsql;
```

## Result

```
SELECT count_tickets_by_class(3, 'Economy') AS economy_count,  
       count_tickets_by_class(3, 'Comfort') AS comfort_count,  
       count_tickets_by_class(3, 'Business') AS business_count;
```

Выполнив такой запрос, получим:

	economy_count integer	comfort_count integer	business_count integer
1	77	0	20

*image-178.png*

Проверим нашу функцию, посчитаем ручками количество проданных билетов в рейсе 3

```
SELECT COUNT(*) FROM ticket_flights WHERE flight_id = 3 AND  
fare_conditions='Economy';  
SELECT COUNT(*) FROM ticket_flights WHERE flight_id = 3 AND  
fare_conditions='Comfort';  
SELECT COUNT(*) FROM ticket_flights WHERE flight_id = 3 AND  
fare_conditions='Business';
```

И получим те же числа.

---

## Task 4

Создайте функцию `get_delayed_flights()`, которая возвращает список рейсов, задержанных на более чем 30 минут и суммарную потерю по выручке для каждого рейса (при условии, что средства возвращаются только пассажирам бизнес-класса в размере 50% от цены билета)

```
CREATE OR REPLACE FUNCTION get_delayed_flights()  
RETURNS TABLE (  
    flight_id integer,  
    flight_no character(6),  
    delay interval,  
    total_refund numeric(10,2)  
) AS $$  
BEGIN  
    RETURN QUERY
```

```

SELECT
    f.flight_id,
    f.flight_no,
    (f.actual_departure - f.scheduled_departure) AS delay,
    COALESCE(SUM(tf.amount * 0.5), 0) AS total_refund
FROM bookings.flights f
LEFT JOIN bookings.ticket_flights tf
    ON f.flight_id = tf.flight_id
    AND tf.fare_conditions = 'Business'
WHERE (f.actual_departure - f.scheduled_departure) > interval '30
minutes'
GROUP BY
    f.flight_id,
    f.flight_no,
    f.actual_departure,
    f.scheduled_departure;

END;
$$ LANGUAGE plpgsql;

```

## Result

Вызовем нашу функцию:

```
SELECT * FROM get_delayed_flights();
```

И получим:

	flight_id integer	flight_no character	delay interval	total_refund numeric	
1	16	PG0403	03:04:00	0	
2	23	PG0402	03:23:00	0	
3	30	PG0404	02:56:00	120000.000	
4	44	PG0403	03:23:00	0	
5	48	PG0403	03:12:00	0	
6	57	PG0403	03:57:00	0	
7	66	PG0405	03:09:00	170000.000	
8	82	PG0405	03:36:00	190000.000	
9	109	PG0402	03:03:00	0	
10	111	PG0404	03:17:00	150000.000	
Total rows: 1000 of 9901			Query complete 00:00:00.626		Ln 28, Col 1

*image-179.png*

Используя функцию из предыдущего задания, можно проверить наличие проданных мест из бизнес-класса. И действительно, там мы видим, что в рейсе 16 было продано 0 мест из бизнес-класса, что подтверждает наш нолик в результате вызова этой функции для этого рейса:

```
SELECT count_tickets_by_class(16, 'Business') AS business_count;
```

	business_count integer	
1	0	

*image-180.png*

## Task 5

Напишите процедуру, которая удаляет все записи о пассажирах, которые купили билеты на рейс с определённым `flight_id`, если этот рейс был отменён.

```
CREATE OR REPLACE PROCEDURE delete_passengers_on_cancelled_flight(  
    p_flight_id integer  
)  
LANGUAGE plpgsql  
AS $$  
DECLARE  
    flight_status text;  
BEGIN  
    -- устанавливаем текущий статус нашего рейса  
    SELECT status INTO flight_status  
    FROM bookings.flights  
    WHERE flight_id = p_flight_id;  
    -- проверяем, что этот рейс правда есть  
    IF flight_status IS NULL THEN  
        RAISE NOTICE 'Рейс с flight_id = % не найден.', p_flight_id;  
    -- проверяем, отменён ли этот рейс  
    ELSIF flight_status <> 'Cancelled' THEN  
        RAISE NOTICE 'Рейс с flight_id = % не отменён. Удаление не  
выполнено.', p_flight_id;  
    ELSE
```



```

-- удаляем данные из таблицы ticket_flights (сначала, чтобы не
нарушить FK)
DELETE FROM bookings.ticket_flights
WHERE flight_id = p_flight_id;
-- удаляем билеты пассажиров этого рейса
DELETE FROM bookings.tickets
WHERE ticket_no IN (
    SELECT ticket_no
    FROM bookings.ticket_flights
    WHERE flight_id = p_flight_id
);
RAISE NOTICE 'Записи о пассажирах и связанные данные для рейса с
flight_id = % удалены.', p_flight_id;
END IF;
END;
$$;

```

## Result

Проверим на рейсе, **который не был отменён**:

```
CALL delete_passengers_on_cancelled_flight(1);
```

ЗАМЕЧАНИЕ: Рейс с flight\_id = 1 не отменён. Удаление не выполнено.  
CALL

Запрос завершён успешно, время выполнения: 98 msec.

*image-181.png*

Проверим на **рейсе, которого нет**:

```
CALL delete_passengers_on_cancelled_flight(11111111);
```

ЗАМЕЧАНИЕ: Рейс с flight\_id = 11111111 не найден.  
CALL

Запрос завершён успешно, время выполнения: 103 msec.

*image-182.png*

Проверим на рейсе, **который отменён**:

Нашли первый попавшийся отменённый рейс - 130013

```
SELECT * FROM flights WHERE status = 'Cancelled';
```

Вызываем нашу процедуру:

```
CALL delete_passengers_on_cancelled_flight(130013);
```

```
ЗАМЕЧАНИЕ: Записи о пассажирах и связанные данные для рейса с flight_id = 130013 удалены.  
CALL
```

```
Запрос завершён успешно, время выполнения: 2 secs 802 msec.
```

*image-183.png*

Проверяем: пробуем найти записи о пассажирах рейса 130013:

```
SELECT * FROM tickets WHERE ticket_no IN (  
    SELECT ticket_no  
    FROM ticket_flights  
    WHERE flight_id = 130013  
);
```

И в результате выполнения запроса получаем 0 строк.

## Task 6

Создайте триггер, который автоматически уменьшает цену билета на 10%, если класс обслуживания — "Business" и цена выше 4000. (то есть при вставке в таблицу билета со стоимостью 4500, в базу запишется 4050)

```
CREATE OR REPLACE FUNCTION reduce_business_price()  
RETURNS trigger  
LANGUAGE plpgsql  
AS $$  
BEGIN  
    -- проверяем условия для применения скидки  
    IF NEW.fare_conditions = 'Business' AND NEW.amount > 4000 THEN  
        NEW.amount := NEW.amount * 0.9; -- применяем 10% скидку
```

```

    END IF;

    RETURN NEW;
END;
$$;

-- триггер, который связывает функцию с таблицей
CREATE TRIGGER trg_reduce_business_price
BEFORE INSERT ON bookings.ticket_flights -- активируется перед вставкой
FOR EACH ROW                             -- для каждой новой строки
EXECUTE FUNCTION reduce_business_price(); -- выполняет нашу функцию

```

## Result

Проверим, как работает наш триггер. Создадим новое бронирование, в которое поместим билетик:

```

INSERT INTO bookings.bookings (book_ref, book_date, total_amount)
VALUES ('B77778', now(), 4500);

INSERT INTO bookings.tickets(ticket_no, book_ref, passenger_id,
passenger_name, contact_data)
VALUES (
    'T1234567899',
    'B77778',
    'P12345',
    'Даниил Немков',
    '{"email": "nemkov@example.com", "phone": "+79991234567"}'
);

```

Далее вставляем этот билетик в ticket\_flights и проверяем, как он вставился:

```




INSERT INTO bookings.ticket_flights(ticket_no, flight_id,
fare_conditions, amount)
VALUES (
    'T1234567899',
    1311,
    'Business',
    4500
);

SELECT ticket_no, fare_conditions, amount

```

```
FROM bookings.ticket_flights
WHERE ticket_no = 'T1234567899';
```

В результате наш триггер сработал при вставке и изменил цену билета, назначив скидку:

	ticket_no character (13) 	fare_conditions character varying (10) 	amount numeric (10,2) 
1	T1234567899	Business	4050.00

*image-184.png*

## Task 7

Создайте триггер `update_flight_status_trigger`, который:

- При обновлении `actual_departure` обновляет статус на "Departed".
- При обновлении `actual_arrival` обновляет статус на "Arrived".

```
CREATE OR REPLACE FUNCTION update_flight_status_func()
RETURNS trigger
LANGUAGE plpgsql
AS $$
BEGIN
    -- проверяем изменение времени прибытия
    IF NEW.actual_arrival IS NOT NULL AND
       (OLD.actual_arrival IS DISTINCT FROM NEW.actual_arrival) THEN
        NEW.status := 'Arrived';
    -- проверяем изменение времени отправления
    ELSIF NEW.actual_departure IS NOT NULL AND
          (OLD.actual_departure IS DISTINCT FROM NEW.actual_departure)
    THEN
        NEW.status := 'Departed';
    END IF;
    RETURN NEW;
END;
$$;

-- триггер связывает функцию с таблицей flights
CREATE TRIGGER update_flight_status_trigger
BEFORE UPDATE ON bookings.flights -- активируется перед обновлением
```

```
FOR EACH ROW                                -- для каждой изменяемой строки
EXECUTE FUNCTION update_flight_status_func();
```

## Result

Проверим, как работает наш триггер:

- изменим время прибытия для рейса 1:

```
UPDATE bookings.flights
SET actual_arrival = now()
WHERE flight_id = 1;

SELECT flight_id, status, actual_arrival
FROM bookings.flights
WHERE flight_id = 1;
```

И в результате увидим, что у нас изменилось время прибытия, статус изменён в соответствии с логикой нашего триггера:

	flight_id [PK] integer	status character varying (20)	actual_arrival timestamp with time zone
1	1	Arrived	2025-03-28 15:16:48.877417+03

*image-185.png*

- теперь проверим работу триггера при изменении времени отправления рейса:

```
UPDATE bookings.flights
SET actual_departure = '2006-01-31 18:30:00+03'
WHERE flight_id = 1;

SELECT flight_id, status, actual_departure
FROM bookings.flights
WHERE flight_id = 1;
```

установить `actual_departure = now()` у меня не получается, видимо существует проверка, которая не позволяет установить `actual_departure > actual_arrival`.

в результате мы получаем новую дату отправления и обновленный статус рейса:

	flight_id [PK] integer	status character varying (20)	actual_departure timestamp with time zone
1	1	Departed	2006-01-31 18:30:00+03

## Task 8

Создайте триггер, который записывает изменения status в таблицу flight\_logs (flight\_id, old\_status, new\_status, changed\_at).

```
-- таблица для хранения логов изменений статусов рейсов
CREATE TABLE IF NOT EXISTS bookings.flight_logs (
    flight_id integer NOT NULL,
    old_status character varying(20),
    new_status character varying(20),
    changed_at timestamp with time zone DEFAULT CURRENT_TIMESTAMP
);

-- триггер для логирования изменений статуса
CREATE OR REPLACE FUNCTION log_status_change()
RETURNS trigger
AS $$
BEGIN
    -- логируем изменение статуса
    INSERT INTO bookings.flight_logs (flight_id, old_status, new_status)
    VALUES (OLD.flight_id, OLD.status, NEW.status);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- триггер для отслеживания изменений статуса
CREATE TRIGGER trg_log_status_change
AFTER UPDATE OF status ON bookings.flights -- срабатывает после
обновления поля status
FOR EACH ROW                               -- для каждой измененной
строки
WHEN (OLD.status IS DISTINCT FROM NEW.status) -- только если статус
действительно изменился
EXECUTE FUNCTION log_status_change();
```

## Result

Проверим работу нашего триггера. Изменим статус первого рейса:

```
UPDATE bookings.flights
SET status = 'Delayed'
WHERE flight_id = 1;

SELECT * FROM bookings.flight_logs;
```

В результате видим, что наш логгер работает, в таблице появилась запись о нашем изменении статуса рейса:

	flight_id integer	old_status character varying (20)	new_status character varying (20)	changed_at timestamp with time zone
1	1	Departed	Delayed	2025-03-28 15:28:00.795374+03

*image-187.png*

## Task 9

Если рейс задерживается более чем на 6 часов, создайте триггер, который автоматически меняет статус на "Cancelled".

```
CREATE OR REPLACE FUNCTION auto_cancel_delayed_flight()
RETURNS trigger AS $$
BEGIN
    -- проверяем, нужно ли отменять рейс
    IF NEW.actual_departure IS NOT NULL AND
        (NEW.actual_departure - NEW.scheduled_departure) > INTERVAL '6
hours' THEN
        NEW.status := 'Cancelled';
        -- для удобства добавим логирование
        RAISE NOTICE 'Рейс % отменен из-за задержки более 6 часов',
NEW.flight_no;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_auto_cancel_delayed_flight
BEFORE UPDATE ON bookings.flights -- срабатывает перед обновлением
FOR EACH ROW                      -- для каждой изменяемой строки
EXECUTE FUNCTION auto_cancel_delayed_flight();
```

## Result

Проверим, как работает наш триггер. Для этого изменим задержку у какого-нибудь рейса:

```
SELECT flight_id, flight_no, scheduled_departure, actual_departure,
status
FROM bookings.flights
WHERE flight_id = 1;

UPDATE bookings.flights
SET actual_departure = scheduled_departure + INTERVAL '7 hours'
WHERE flight_id = 1;
```

По логам видим, что наш триггер сработал:

```
ЗАМЕЧАНИЕ: Рейс PG0403 отменен из-за задержки более 6 часов
UPDATE 1

Запрос завершён успешно, время выполнения: 54 msec.
```

*image-188.png*

Проверяем данные рейса:

```
SELECT flight_id, scheduled_departure, actual_departure, status
FROM bookings.flights
WHERE flight_id = 1;
```

И видим, что статус нашего рейса был изменен.

	flight_id [PK] integer 	scheduled_departure timestamp with time zone 	actual_departure timestamp with time zone 	status character varying (20) 
1	1	2017-06-13 11:25:00+03	2017-06-13 18:25:00+03	Cancelled

*image-191.png*