

ЛАБОРАТОРНАЯ РАБОТА №5

Цель работы: овладеть практическими создания и использования процедур, функций и триггеров в базе данных PostgreSQL.

Оборудование: компьютерный класс.

Програмное обеспечение: СУБД PostgreSQL, SQL Shell (psql).

Практическое задание:

Вариант 2:

1. Создать процедуры/функции согласно индивидуальному заданию
2. Модифицировать триггер (триггерную функцию) на проверку корректности входа и выхода сотрудника
3. Создать авторский триггер по варианту индивидуального задания

Задание 1. Создание хранимых процедур/функций.

1. Для увеличения цены всех номеров на 5 %, если в отеле нет свободных номеров:

```
1 lab_5=# CREATE OR REPLACE PROCEDURE increase_room_price()
2 lab_5-# LANGUAGE plpgsql AS $$
3 lab_5$# BEGIN
4 lab_5$# IF NOT EXISTS (SELECT 1 FROM room WHERE is_free = true) THEN
5 lab_5$# UPDATE cost_room_class SET cost_per_day = cost_per_day * 1.05;
6 lab_5$# END IF;
7 lab_5$# END;
8 lab_5$# $$;
9 CREATE PROCEDURE
```

Тестирование:

В отеле с id_hotel 2 все номера заняты:

```
1 SELECT CASE
2     WHEN NOT EXISTS (SELECT 1 FROM room WHERE is_free = true AND id_
3         hotel = 2) THEN 1
4     ELSE 0
5 END;
```

Ответ:

	case integer
1	1

однако, эта хранимая процедура подразумевает, что каждому номеру соответствует конкретная стоимость. В моей бд, стоимость определяется для типа номера, а не для комнаты. данный запрос изменил бы стоимость всех номеров во всех отелях, поскольку каждый отель содержит все классы комнат

2. Для получения информации о свободных одноместных номерах отеля на завтрашний день:

```

1 lab_5=# CREATE OR REPLACE FUNCTION get_available_single_rooms()
2 lab_5-# RETURNS TABLE(id_hotel INT, id_room INT, room_num INT, class
   VARCHAR, is_free BOOLEAN) AS $$
3 lab_5## BEGIN
4 lab_5$$      RETURN QUERY SELECT r.id_hotel, r.id_room, r.room_num,
   rc.class, r.is_free
5 lab_5$$      FROM room r
6 lab_5$$      JOIN room_class rc ON r.id_class = rc.id_class
7 lab_5$$      WHERE r.is_free = true AND rc.person_amount = 1
8 lab_5$$      AND r.id_room NOT IN (
9 lab_5$$          SELECT reg.id_room FROM registration reg
10 lab_5$$          WHERE reg.arrival_date <= CURRENT_DATE + 1 AND
   reg.departure_date > CURRENT_DATE + 1
11 lab_5$$      );
12 lab_5## END;
13 lab_5## $$ LANGUAGE plpgsql;
14 CREATE FUNCTION

```

Тестирование:

```

1 SELECT * FROM get_available_single_rooms();

```

Ответ:

	id_hotel integer	id_room integer	room_num integer	class character varying	is_free boolean
1	44	43	43	Class 4	true
2	48	47	47	Class 8	true

3. Процедура для бронирования двухместного номера:

```

1 lab_5=# CREATE OR REPLACE PROCEDURE book_double_room(
2 lab_5(#      p_id_client INTEGER,
3 lab_5(#      p_id_executor INTEGER,

```

```

4 lab_5(#      p_residence_address TEXT,
5 lab_5(#      p_booking_date DATE,
6 lab_5(#      p_arrival_date DATE,
7 lab_5(#      p_departure_date DATE,
8 lab_5(#      p_payment_method TEXT,
9 lab_5(#      p_reg_status TEXT,
10 lab_5(#      p_payment_status BOOLEAN,
11 lab_5(#      p_is_archived BOOLEAN)
12 lab_5-# LANGUAGE plpgsql AS $$
13 lab_5$$# DECLARE
14 lab_5$$#      v_id_room INTEGER;
15 lab_5$$# BEGIN
16 lab_5$$#      SELECT id_room INTO v_id_room FROM room
17 lab_5$$#      JOIN room_class ON room.id_class = room_class.id_class
18 lab_5$$#      WHERE room.is_free = true AND room_class.person_amount = 2
19 lab_5$$#      LIMIT 1;
20 lab_5$$#
21 lab_5$$#      IF FOUND THEN
22 lab_5$$#          INSERT INTO registration
23 lab_5$$#              (id_client, id_room, id_executor,
24 lab_5$$#                  residence_address, booking_date, arrival_date, departure_date,
25 lab_5$$#                      payment_method, reg_status, payment_status, is_archived)
26 lab_5$$#              VALUES
27 lab_5$$#                  (p_id_client, v_id_room, p_id_executor,
28 lab_5$$#                      p_residence_address, p_booking_date, p_arrival_date,
29 lab_5$$#                          p_departure_date, p_payment_method, p_reg_status, p_payment_status,
30 lab_5$$#                              p_is_archived);
31 lab_5$$#      ELSE
32 lab_5$$#          RAISE EXCEPTION 'Нет доступных двухместных номеров';
33 lab_5$$#      END IF;
34 lab_5$$# END;

```

```

30 lab_5$# $$;
31 CREATE PROCEDURE

```

Тест:

```

1 CALL book_double_room(
2     1,
3     10,
4     '123 Main St',
5     CURRENT_DATE,
6     CURRENT_DATE + 1,
7     CURRENT_DATE + 4,
8     'card',
9     'забронирован',
10    false,
11    false
12 );

```

проверка:

```

1 SELECT * FROM registration
2 WHERE id_client = 1
3 ORDER BY booking_date DESC;

```

Ответ:

	id_reg [PK] integer	id_client integer	id_room integer	id_executor integer	is_archived boolean	residence_address character varying (255)	booking_date date	arrival_date date	departure_date date	payment_method character varying (10)	reg_status character varying (18)
1	1005	1	10	100	false	Адрес какой-нибудь	2024-01-27	2024-01-28	2024-01-31	card	забронирован
2	1006	1	100	10	false	Адрес какой-нибудь	2024-01-27	2024-01-28	2024-01-31	card	забронирован
3	1009	1	20	10	false	123 Main St	2024-01-27	2024-01-28	2024-01-31	card	забронирован
4	500	1	501	101	true	Address 500	2024-01-21	2024-01-22	2024-01-23	card	забронирован
5	1000	1	1	1	true	Address 1000	2024-01-11	2024-01-12	2024-01-13	card	забронирован

Задание 2. Таймер для учета работы сотрудников

1. Триггер для проверки корректности входа и выхода сотрудника

1) Создание БД и таблиц:

```
1 lab_5=# CREATE DATABASE emp_time;
2 CREATE DATABASE
3 lab_5=#
4 lab_5=# \c emp_time
5 Вы подключены к базе данных "emp_time" как пользователь "postgres".
6 emp_time=#
7 emp_time=#
8 emp_time=# CREATE TABLE employee (
9 emp_time(#      id serial PRIMARY KEY,
10 emp_time(#      username varchar
11 emp_time(# );
12 CREATE TABLE
13 emp_time=#
14 emp_time=# CREATE TABLE time_punch (
15 emp_time(#      id serial PRIMARY KEY,
16 emp_time(#      employee_id int NOT NULL REFERENCES employee(id),
17 emp_time(#      is_out_punch boolean NOT NULL DEFAULT false,
18 emp_time(#      punch_time timestamp NOT NULL DEFAULT now()
19 emp_time(# );
20 CREATE TABLE
```

2) Вставка данных:

```
1 emp_time=# INSERT INTO employee (username) VALUES ('Михаил');
2 INSERT 0 1
3 emp_time=#
4 emp_time=# INSERT INTO time_punch (employee_id, is_out_punch,
5 emp_time-# (1, false, '2021-01-01 10:00:00'),
```

```

6 emp_time=# (1, true, '2021-01-01 11:30:00');
7 INSERT 0 2

```

3) Создание триггерной функции и триггера:

```

1 emp_time=# CREATE OR REPLACE FUNCTION fn_check_time_punch() RETURNS
    TRIGGER AS $$
2 emp_time$$ BEGIN
3 emp_time$$     IF NEW.is_out_punch = (
4 emp_time$$     SELECT tps.is_out_punch
5 emp_time$$     FROM time_punch tps
6 emp_time$$     WHERE tps.employee_id = NEW.employee_id
7 emp_time$$     ORDER BY tps.id DESC LIMIT 1
8 emp_time$$     ) THEN
9 emp_time$$     RETURN NULL;
10 emp_time$$     END IF;
11 emp_time$$     RETURN NEW;
12 emp_time$$ END;
13 emp_time$$ $$ LANGUAGE plpgsql;
14 CREATE FUNCTION
15 emp_time=#
16 emp_time=# CREATE TRIGGER check_time_punch BEFORE INSERT ON time_punch
17 emp_time=# FOR EACH ROW EXECUTE PROCEDURE fn_check_time_punch();
18 CREATE TRIGGER

```

4) Модификация триггера:

Модифицируем триггер таким образом, чтобы он проверял, что каждый "выход" сотрудника происходит после его последнего "входа" и гарантировал, что не записывались подряд два события одного типа (два "входа" или два "выхода").

Обновленный код:

```

1 CREATE OR REPLACE FUNCTION fn_check_time_punch() RETURNS TRIGGER AS $$
2 DECLARE

```

```

3     last_punch_record RECORD;
4 BEGIN
5     -- Получаем последнюю запись о пробитии времени для данного сотрудн
    ика
6     SELECT is_out_punch, punch_time INTO last_punch_record
7     FROM time_punch
8     WHERE employee_id = NEW.employee_id
9     ORDER BY id DESC LIMIT 1;
10
11     IF last_punch_record.is_out_punch IS NOT NULL AND
12        NEW.is_out_punch = last_punch_record.is_out_punch THEN
13        RAISE EXCEPTION 'Неправильная последовательность входа
    /выхода.';
14     END IF;
15
16     IF NEW.is_out_punch AND last_punch_record.punch_time IS NOT NULL
    AND
17        NEW.punch_time <= last_punch_record.punch_time THEN
18        RAISE EXCEPTION 'Время выхода должно быть после последнего врем
    ени входа.';
19     END IF;
20
21     IF NEW.punch_time > CURRENT_TIMESTAMP THEN
22        RAISE EXCEPTION 'Время входа не может быть после текущей даты
    .';
23     END IF;
24
25     IF NEW.is_out_punch > CURRENT_TIMESTAMP THEN
26        RAISE EXCEPTION 'Время выхода не может быть после текущей даты
    .';
27     END IF;

```



```

28
29     RETURN NEW;
30 END;
31 $$ LANGUAGE plpgsql;

```

Теперь:

- Сначала извлекается последняя запись для сотрудника. Эта запись содержит информацию о типе события (вход/выход) и времени события.
- Проверяется, чтобы новая запись не совпадала по типу с последней записью (например, чтобы не было двух "выходов" подряд).
- Если новая запись — это "выход", проверяется, что его время позже времени последнего "входа".

5) Тестирование модифицированного триггера:

Ввод корректных данных:

```

1 emp_time=# -- Запись входа
2 emp_time=# INSERT INTO time_punch (employee_id, is_out_punch,
3     punch_time) VALUES (1, false, '2021-01-02 09:00:00');
4 INSERT 0 1
5 emp_time=# -- Запись выхода
6 emp_time=# INSERT INTO time_punch (employee_id, is_out_punch,
7     punch_time) VALUES (1, true, '2021-01-02 17:00:00');
8 INSERT 0 1

```

Ввод некорректных данных. Повторный выход:

```

1 emp_time=# INSERT INTO time_punch (employee_id, is_out_punch,
2     punch_time) VALUES (1, true, '2021-01-02 18:00:00');
3 ERROR:  Неправильная последовательность входа/выхода.
4 КОНТЕКСТ:  PL/pgSQL function fn_check_time_punch() line 14 at RAISE

```

Ввод некорректных данных. Два входа подряд в разное время:

```

1 emp_time=# INSERT INTO time_punch (employee_id, is_out_punch,
    punch_time) VALUES (1, false, '2021-01-03 09:00:00');
2 INSERT 0 1
3 emp_time=# INSERT INTO time_punch (employee_id, is_out_punch,
    punch_time) VALUES (1, false, '2021-01-03 19:00:00');
4 ERROR:  Неправильная последовательность входа/выхода.
5 КОНТЕКСТ:  PL/pgSQL function fn_check_time_punch() line 14 at RAISE

```

6) Данные учета времени:

```

1 emp_time=# SELECT * FROM time_punch ORDER BY employee_id, punch_time;
2 id | employee_id | is_out_punch |    punch_time
3 ----+-----+-----+-----
4  1 |           1 | f           | 2021-01-01 10:00:00
5  2 |           1 | t           | 2021-01-01 11:30:00
6  4 |           1 | f           | 2021-01-02 09:00:00
7  7 |           1 | t           | 2021-01-02 17:00:00
8  9 |           1 | f           | 2021-01-03 09:00:00
9 (5 строк)

```

Задание 3. Создание авторского триггера по варианту индивидуального задания
(БД "Отель")

Триггер для автоматического обновления статуса номера при бронировании или освобождении:

```
1 emp_time=# \c lab_5
2 Вы подключены к базе данных "lab_5" как пользователь "postgres".
3 lab_5=# CREATE OR REPLACE FUNCTION update_room_status()
4 lab_5=# RETURNS TRIGGER AS $$
5 lab_5$$ BEGIN
6 lab_5$$      -- При бронировании номера
7 lab_5$$      IF NEW.reg_status = 'забронирован' THEN
8 lab_5$$      UPDATE room SET is_free = false WHERE id_room =
      NEW.id_room;
9 lab_5$$      ELSIF NEW.reg_status = 'выселен' THEN
10 lab_5$$      -- При выписке из номера
11 lab_5$$      UPDATE room SET is_free = true WHERE id_room =
      NEW.id_room;
12 lab_5$$      END IF;
13 lab_5$$
14 lab_5$$      RETURN NEW;
15 lab_5$$ END;
16 lab_5$$ $$ LANGUAGE plpgsql;
17 CREATE FUNCTION
18 lab_5=# CREATE TRIGGER trigger_update_room_status
19 lab_5=# AFTER INSERT OR UPDATE ON registration
20 lab_5=# FOR EACH ROW EXECUTE FUNCTION update_room_status();
21 CREATE TRIGGER
```

Теперь протестируем триггер. Для начала найдем какой-нибудь свободный номер в отеле с id_hotel = 1:

```
1 SELECT id_room FROM room WHERE is_free = true AND id_hotel = 1 LIMIT 1;
```

Ответ:

	id_room [PK] integer
1	100

Теперь заселим клиента в комнату №100 в отель с id_hotel = 1:

```
1 INSERT INTO registration
2   (id_client, id_room, id_executor, residence_address, booking_date,
3    arrival_date, departure_date, payment_method, reg_status,
4    payment_status, is_archived)
5 VALUES
6   (1, 100, 10, 'Адрес какой-нибудь', CURRENT_DATE, CURRENT_DATE +
7    INTERVAL '1 day', CURRENT_DATE + INTERVAL '4 days', 'card', 'заброни
8    рован', true, false);
```

Проверка статуса комнаты:

```
1 SELECT id_room, is_free FROM room WHERE id_room = 100 AND id_hotel = 1;
```

Ответ:

	id_room [PK] integer	is_free boolean
1	100	false