

Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение высшего
образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчет

по практической работе «Триггеры и функции»

по дисциплине «Проектирование и реализация баз данных»

Автор: Будунов Б. С.

Факультет: ИКТ

Группа: К3241

Преподаватель: Говорова М.М.



Санкт-Петербург 2024

Практическое задание:

Пример 1. Таймер для учета работы сотрудников

Задание 1:

1. Создайте БД `emp_time`. Используйте консоль `psql`.
2. Проверьте список активных БД.
3. Подключитесь к БД `emp_time`.
4. Создайте таблицы БД со следующей структурой:

Таблица `employee`

`id` (тип `serial`, первичный ключ)

`username` (тип `varchar`)

Таблица `time_punch`

`id` (тип `serial`, первичный ключ)

`employee_id` (тип `int`, обязательное, внешний ключ: соответствует ключу `employee.id`)

`is_out_punch` (тип `boolean`, обязательное, значение по умолчанию `false`)

`punch_time` (тип `timestamp`, обязательное, значение по умолчанию `now()`).

```
SQL Shell (psql)
postgres=# \l chcp 1251
Текущая кодовая страница: 1251
postgres=# \list

```

Имя	Владелец	Кодировка	LC_COLLATE	LC_CTYPE	Права доступа
postgres	postgres	UTF8	Russian_Russia.1251	Russian_Russia.1251	
template0	postgres	UTF8	Russian_Russia.1251	Russian_Russia.1251	=c/postgres
template1	postgres	UTF8	Russian_Russia.1251	Russian_Russia.1251	=c/postgres

```
(3 строки)

postgres=# create database
postgres=# create database emp_time
postgres=# \c emp_time
ВАЖНО: база данных "emp_time" не существует
Сохранено предыдущее подключение
postgres=# create database emp_time
postgres=# \c emp_time
ВАЖНО: база данных "emp_time" не существует
Сохранено предыдущее подключение
postgres=# CREATE DATABASE emp_time;
ОШИБКА: ошибка синтаксиса (примерное положение: "!")
СТРОКА 1: 1:chcp 1251
^
postgres=# create database emp_time;
CREATE DATABASE
postgres=# \c emp_time
Вы подключены к базе данных "emp_time" как пользователь "postgres".
emp_time=# \dt
отношения не найдены.
emp_time=# CREATE TABLE employee (
emp_time=#   id serial PRIMARY KEY,
emp_time=#   username varchar
emp_time=# );
CREATE TABLE
emp_time=#
emp_time=# CREATE TABLE employee (
emp_time=#   id serial PRIMARY KEY,
emp_time=#   username varchar
emp_time=# );
```

```
SQL Shell (psql)
CREATE DATABASE
postgres=# \c emp_time
Вы подключены к базе данных "emp_time" как пользователь "postgres".
emp_time=# \dt
Отношения не найдены.
emp_time=# CREATE TABLE employee (
emp_time=#   id serial PRIMARY KEY,
emp_time=#   username varchar
emp_time=# );
emp_time=# CREATE TABLE
emp_time=# CREATE TABLE employee (
emp_time=#   id serial PRIMARY KEY,
emp_time=#   username varchar
emp_time=# );
ОШИБКА: отношение "employee" уже существует
emp_time=# \dt
Список отношений
Схема | Имя | Тип | Владелец
-----+-----+-----+-----
public | employee | таблица | postgres
(1 строка)

emp_time=# CREATE TABLE time_punch (
emp_time=#   id serial PRIMARY KEY,
emp_time=#   employee_id int REFERENCES employee(id),
emp_time=#   is_out_punch boolean DEFAULT false,
emp_time=#   punch_time timestamp DEFAULT now()
emp_time=# );
emp_time=# CREATE TABLE
emp_time=# CREATE TABLE time_punch (
emp_time=#   id serial PRIMARY KEY,
emp_time=#   employee_id int REFERENCES employee(id),
emp_time=#   is_out_punch boolean DEFAULT false,
emp_time=#   punch_time timestamp DEFAULT now()
emp_time=# );
ОШИБКА: отношение "time_punch" уже существует
emp_time=# \dt
Список отношений
Схема | Имя | Тип | Владелец
-----+-----+-----+-----
public | employee | таблица | postgres
public | time_punch | таблица | postgres
(2 строки)
```

Вставьте данные в таблицы:

1. Вставьте в таблицу сотрудников строку для сотрудника с именем Михаил.
2. Вставьте в таблицу учета времени две строки: вход Михаил в 10.00 2021-01-01 и выход в 11.30 2021-01-01.
3. Проверьте содержимое таблиц.

```
SQL Shell (psql)
Список отношений
Схема | Имя | Тип | Владелец
-----+-----+-----+-----
public | employee | таблица | postgres
public | time_punch | таблица | postgres
(2 строки)

emp_time=# INSERT INTO employee (username)
emp_time=# VALUES ('Михаил');
INSERT 0 1
emp_time=# \dt
Список отношений
Схема | Имя | Тип | Владелец
-----+-----+-----+-----
public | employee | таблица | postgres
public | time_punch | таблица | postgres
(2 строки)

emp_time=# SELECT * FROM employee;
 id | username
-----+-----
  1 | Михаил
(1 строка)

emp_time=# INSERT INTO time_punch (employee_id, is_out_punch, punch_time)
emp_time=# VALUES ((SELECT id FROM employee WHERE username = 'Михаил'), false, '2021-01-01 10:00:00'),
emp_time=# ((SELECT id FROM employee WHERE username = 'Михаил'), true, '2021-01-01 11:30:00');
INSERT 0 2
emp_time=# SELECT * FROM time_punch;
 id | employee_id | is_out_punch | punch_time
-----+-----+-----+-----
  1 |          1 | f            | 2021-01-01 10:00:00
  2 |          1 | t            | 2021-01-01 11:30:00
(2 строки)

emp_time=# select tp1.punch_time - tp2.punch_time as time_worked
emp_time=# from time_punch tp1 join time_punch tp2
emp_time=#   on tp2.id = (select tps.id from time_punch tps
emp_time=#   where tps.id < tp1.id and
emp_time=#   tps.employee_id = tp1.employee_id and
emp_time=#   not tps.is_out_punch
```

Вычисление рабочего времени

Запрос позволяет на каждый выход найти соответствующий вход:

```

select tp1.punch_time - tp2.punch_time as time_worked
  from time_punch tp1 join time_punch tp2
    on tp2.id = (select tps.id from time_punch tps
                  where tps.id < tp1.id and
                      tps.employee_id = tp1.employee_id and
                      not tps.is_out_punch
                  order by tps.id desc limit 1
                )
 where tp1.employee_id = 1 and tp1.is_out_punch;

```

Задание 2:

Проверьте работу запроса.

Примечание.

Одна из проблем в этой схеме состоит в том, что возможно вставить несколько "входов" или "выходов" подряд. С созданным запросом это приведет к неоднозначности, которая может привести к неточным расчетам и зарплате сотрудников – больше или меньше, чем они должны были бы получить.

```

emp_time=# \INSERT INTO time_punch (employee_id, is_out_punch, punch_time)
Неверная команда \INSERT. Справка по командам: \?
emp_time=# VALUES ((SELECT id FROM employee WHERE username = 'Михаил'), false, '2021-01-01 10:00:00'),
emp_time=# ((SELECT id FROM employee WHERE username = 'Михаил'), true, '2021-01-01 11:30:00');
 column1 | column2 | column3
-----+-----+-----
 1 | f | 2021-01-01 10:00:00
 1 | t | 2021-01-01 11:30:00
(2 строки)

emp_time=# INSERT INTO time_punch (employee_id, is_out_punch, punch_time)
emp_time=# VALUES ((SELECT id FROM employee WHERE username = 'Михаил'), false, '2021-01-01 10:00:00'),
emp_time=# ((SELECT id FROM employee WHERE username = 'Михаил'), true, '2021-01-01 11:30:00');
INSERT 0 2
emp_time=# ((SELECT id FROM employee WHERE username = 'Михаил'), true, '2021-01-01 11:30:00');
ОШИБКА: ошибка синтаксиса (примерное положение: ",")
СТРОКА 1: ...SELECT id FROM employee WHERE username = 'Михаил'), true, '2...
^

emp_time=# SELECT * FROM time_punch;
 id | employee_id | is_out_punch | punch_time
----+-----+-----+-----
 1 | 1 | f | 2021-01-01 10:00:00
 2 | 1 | t | 2021-01-01 11:30:00
 3 | 1 | f | 2021-01-01 10:00:00
 4 | 1 | t | 2021-01-01 11:30:00
(4 строки)

emp_time=# ROLLBACK
emp_time=# SELECT * FROM time_punch;
ОШИБКА: ошибка синтаксиса (примерное положение: "SELECT")
СТРОКА 2: SELECT * FROM time_punch;
^

emp_time=# SELECT * FROM time_punch
emp_time=# SELECT * FROM time_punch;
ОШИБКА: ошибка синтаксиса (примерное положение: "SELECT")
СТРОКА 2: SELECT * FROM time_punch;
^

emp_time=# \SELECT * FROM time_punch;
Неверная команда \SELECT. Справка по командам: \?
emp_time=# SELECT * FROM time_punch;
 id | employee_id | is_out_punch | punch_time
----+-----+-----+-----
 1 | 1 | f | 2021-01-01 10:00:00

```

Требуется то, что не позволит нарушить шаблон вход/выход. К сожалению, ограничения check только отслеживают вставляемую или обновляемую строку и не могут учитывать данные из других строк.

Создадим триггер для предотвращения события INSERT, которое нарушает шаблон. Сначала создадим "триггерную функцию". Эта функция есть то, что будет выполнять триггер при наступлении события.

Триггерная функция создается как обычная функция PostgreSQL за тем исключением, что возвращает *trigger*.

```

create or replace function fn_check_time_punch() returns
trigger as $psql$

```

```

begin
    if new.is_out_punch = (
        select tps.is_out_punch
        from time_punch tps
        where tps.employee_id = new.employee_id
        order by tps.id desc limit 1
    ) then
        return null;
    end if;
    return new;
end;
$psql$ language plpgsql;

```

```

emp_time=# CREATE TABLE logs (
emp_time=#     text TEXT,
emp_time=#     added TIMESTAMP WITHOUT TIME ZONE
emp_time=# );
emp_time=# CREATE TABLE
emp_time=# \dt
emp_time=#
Список отношений
-----
Схема | Имя | Тип | Владелец
-----
public | employee | таблица | postgres
public | logs | таблица | postgres
public | time_punch | таблица | postgres
(3 строки)

emp_time=# CREATE TRIGGER t_employee AFTER INSERT OR UPDATE OR DELETE ON employee FOR EACH ROW EXECUTE PROCEDURE add_to_log ();
ОШИБКА: функция add_to_log() не существует
emp_time=# CREATE OR REPLACE FUNCTION add_to_log() RETURNS TRIGGER AS $$
emp_time=# DECLARE
emp_time=#     mstr varchar(30);
emp_time=#     astr varchar(100);
emp_time=#     retstr varchar(254);
emp_time=# BEGIN
emp_time=#     IF TG_OP = 'INSERT' THEN
emp_time=#         astr = NEW.id;
emp_time=#         mstr := 'Add new user ';
emp_time=#         retstr := mstr||astr;
emp_time=#         INSERT INTO logs(text,added) values (retstr,NOW());
emp_time=#         RETURN NEW;
emp_time=#     ELSEIF TG_OP = 'UPDATE' THEN
emp_time=#         astr = NEW.id;
emp_time=#         mstr := 'Update user ';
emp_time=#         retstr := mstr||astr;
emp_time=#         INSERT INTO logs(text,added) values (retstr,NOW());
emp_time=#         RETURN NEW;
emp_time=#     ELSEIF TG_OP = 'DELETE' THEN
emp_time=#         astr = OLD.id;
emp_time=#         mstr := 'Remove user ';
emp_time=#         retstr := mstr || astr;
emp_time=#         INSERT INTO logs(text,added) values (retstr,NOW());
emp_time=#         RETURN OLD;
emp_time=#     END IF;
emp_time=# END;
emp_time=# $$ LANGUAGE plpgsql;
emp_time=#
CREATE FUNCTION

```

Ключевое слово `new` представляет значения вставляемой строки. Это также объект, который можно вернуть, чтобы позволить продолжиться вставке. Напротив, возвращение `null` остановит вставку.

Этот запрос сначала находит в `time_punch` предыдущее значение и гарантирует, что это значение входа/выхода не совпадает с вставляемым значением. Если значения совпадают, то триггер возвращает `null`, и `time_punch` не записывается. В противном случае, триггер возвращает `new` и оператор `insert` продолжается.

Теперь привяжем функцию в качестве триггера к таблице `time_punch`. BEFORE здесь ключевой момент. Если выполним этот триггер как триггер AFTER, он будет выполнен слишком поздно, чтобы остановить вставку.

```

create trigger check_time_punch before insert on time_punch
for each row execute procedure fn_check_time_punch();

```

Задание 3:

1. Проверьте работу триггера на корректных и некорректных данных по входу и выходу сотрудника.

```
SQL Shell (psql)
CREATE FUNCTION
emp_time=#
emp_time=# CREATE TRIGGER t_employee AFTER INSERT OR UPDATE OR DELETE ON employee FOR EACH ROW EXECUTE PROCEDURE add_to_log ();
CREATE TRIGGER
emp_time=#
emp_time=# INSERT INTO employee (username) VALUES ('Ульяна');
INSERT 0 1
emp_time=# SELECT * FROM employee;
 id | username
-----+-----
  1 | Михаил
  2 | Ульяна
(2 строки)

emp_time=# SELECT * FROM logs;
 text | added
-----+-----
Add new user 2 | 2023-11-23 16:32:44.085872
(1 строка)

emp_time=# UPDATE employee SET username = 'Не Ульяна' WHERE id = 1;
UPDATE 1
emp_time=# SELECT * FROM employee;
 id | username
-----+-----
  2 | Ульяна
  1 | Не Ульяна
(2 строки)

emp_time=# SELECT * FROM logs;
 text | added
-----+-----
Add new user 2 | 2023-11-23 16:32:44.085872
Update user 1 | 2023-11-23 16:35:37.341799
(2 строки)

emp_time=# DELETE FROM employee WHERE id = 2;
DELETE 1
emp_time=# SELECT * FROM employee;
 id | username
-----+-----
(0 строки)
```

```
SQL Shell (psql)
emp_time=# DELETE FROM employee WHERE id = 2;
DELETE 1
emp_time=# SELECT * FROM employee;
 id | username
-----+-----
  1 | Не Ульяна
(1 строка)

emp_time=# SELECT * FROM logs;
 text | added
-----+-----
Add new user 2 | 2023-11-23 16:32:44.085872
Update user 1 | 2023-11-23 16:35:37.341799
Remove user 2 | 2023-11-23 16:37:45.396171
(3 строки)

emp_time=# UPDATE employee SET username = 'Михаил' WHERE id = 1;
UPDATE 1
emp_time=# SELECT * FROM employee;
 id | username
-----+-----
  1 | Михаил
(1 строка)

emp_time=# SELECT * FROM logs;
 text | added
-----+-----
Add new user 2 | 2023-11-23 16:32:44.085872
Update user 1 | 2023-11-23 16:35:37.341799
Remove user 2 | 2023-11-23 16:37:45.396171
Update user 1 | 2023-11-23 16:38:37.239725
(4 строки)

emp_time=# create table time_punch_audit (
emp_time(# id serial primary key,
emp_time(# change_time timestamp not null default now(),
emp_time(# change_employee_id int not null references employee(id),
emp_time(# time_punch_id int not null references time_punch(id),
emp_time(# punch_time timestamp not null
emp_time(# );
CREATE TABLE
```

Пример 2. Аудит изменения данных

Вариант 2

Пусть компания хочет воссоздавать в хронологии состояние таблицы на случай обнаружения нарушений.

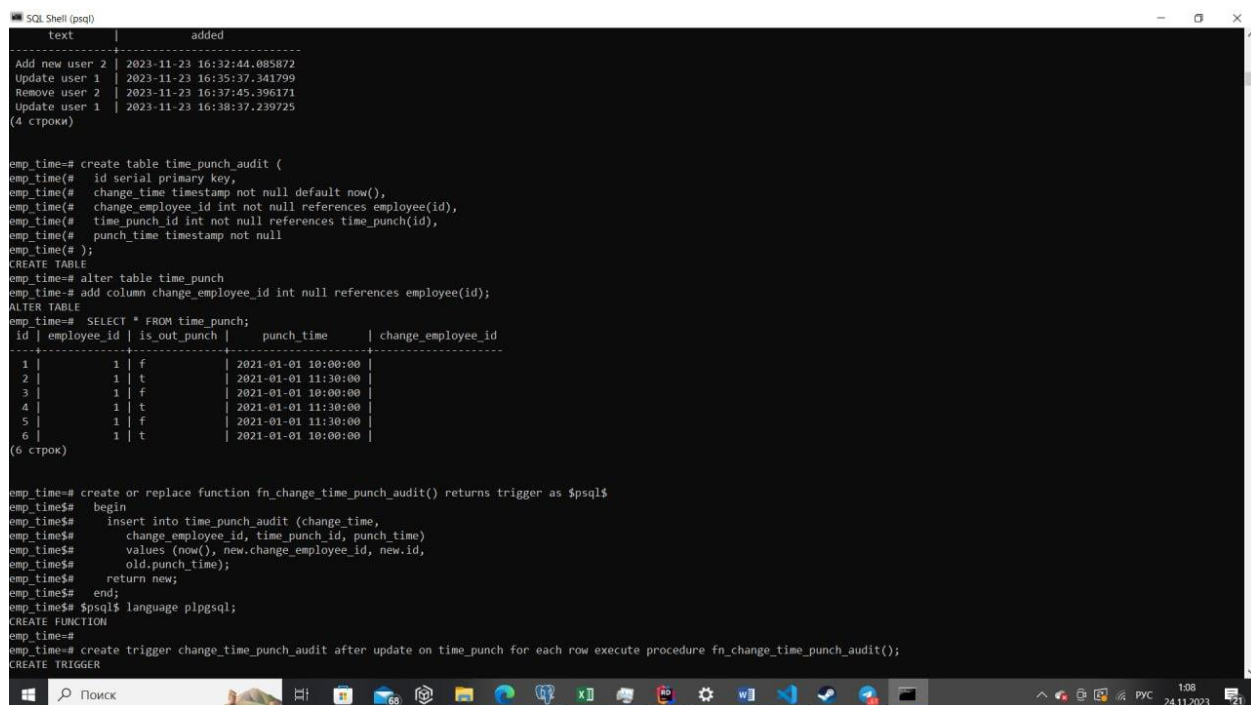
Таблица аудита выполняет эту роль, отслеживая каждое изменение основной таблицы. Когда в главной таблице обновляется строка, в таблицу аудита вставляется строка в ее предыдущем состоянии.

Используем таблицу `time_punch` для демонстрации создания и автоматического обновления таблицы аудита с помощью триггеров.

Задание 6:

Создайте таблицу аудита следующей командой:

```
create table time_punch_audit (  
    id serial primary key,  
    change_time timestamp not null default now(),  
    change_employee_id int not null references employee(id),  
    time_punch_id int not null references time_punch(id),  
    punch_time timestamp not null  
);
```



```
SQL Shell (psql)  
text | added  
-----  
Add new user 2 | 2023-11-23 16:32:44.085872  
Update user 1 | 2023-11-23 16:35:37.341799  
Remove user 2 | 2023-11-23 16:37:45.396171  
Update user 1 | 2023-11-23 16:38:37.239725  
(4 строки)  
  
emp_time=# create table time_punch_audit (  
emp_time# id serial primary key,  
emp_time# change_time timestamp not null default now(),  
emp_time# change_employee_id int not null references employee(id),  
emp_time# time_punch_id int not null references time_punch(id),  
emp_time# punch_time timestamp not null  
emp_time# );  
emp_time# CREATE TABLE  
emp_time# alter table time_punch  
emp_time# add column change_employee_id int null references employee(id);  
emp_time# ALTER TABLE  
emp_time# SELECT * FROM time_punch;  
id | employee_id | is_out_punch | punch_time | change_employee_id  
-----  
1 | 1 | f | 2021-01-01 10:00:00 |  
2 | 1 | t | 2021-01-01 11:30:00 |  
3 | 1 | f | 2021-01-01 10:00:00 |  
4 | 1 | t | 2021-01-01 11:30:00 |  
5 | 1 | f | 2021-01-01 11:30:00 |  
6 | 1 | t | 2021-01-01 10:00:00 |  
(6 строк)  
  
emp_time=# create or replace function fn_change_time_punch_audit() returns trigger as $psql$  
emp_time# begin  
emp_time# insert into time_punch_audit (change_time,  
emp_time# change_employee_id, time_punch_id, punch_time)  
emp_time# values (now(), new.change_employee_id, new.id,  
emp_time# old.punch_time);  
emp_time# return new;  
emp_time# end;  
emp_time# $psql$ language plpgsql;  
emp_time# CREATE FUNCTION  
emp_time# create trigger change_time_punch_audit after update on time_punch for each row execute procedure fn_change_time_punch_audit();  
emp_time# CREATE TRIGGER
```

В эту таблицу записывается:

- время обновления прохождения,
- сотрудник, который выполнил обновление,
- ID прохода, который был изменен,
- время прохода до того, как было сделано обновление.

Прежде чем создавать триггер, сначала нужно добавить столбец `change_employee_id` в таблицу `time_punch`. Тогда триггер будет знать, какой сотрудник сделал каждое изменение в таблице `time_punch`.

```
alter table time_punch
```

```
add column change_employee_id int null references
employee(id);
```

Далее создадим функцию, которая и запишет OLD (старое) значение времени прохода в нашу таблицу аудита.

```
create or replace function fn_change_time_punch_audit()
returns trigger as $psql$
begin
    insert into time_punch_audit (change_time,
        change_employee_id, time_punch_id, punch_time)
        values (now(), new.change_employee_id, new.id,
            old.punch_time);
    return new;
end;
$psql$ language plpgsql;
```

Триггер для обеспечения аудита:

```
create trigger change_time_punch_audit after update on
time_punch for each row execute procedure
fn_change_time_punch_audit();
```

Функция NOW() возвращает текущую дату и время с точки зрения сервера SQL. Если бы это было привязано к настоящему приложению, можно передавать точное время, когда пользователь фактически сделал запрос, чтобы избежать расхождения из-за задержки.

Для триггера на обновление объект NEW представляет те значения, которые будут содержаться в строке при успешном обновлении. Можно использовать триггер для "перехвата" вставки или обновления простым присвоением своих собственных значений в объект NEW. Объект OLD содержит значения строки до обновления.

Задание 7:

1. Добавьте нового сотрудника в БД.
2. Пусть он будет редактором времени прохода Михаила.
3. Пусть необходимо отредактировать время выхода в 11:30
4. Выполните дважды запрос для имитации 2 редакций, которые увеличивают время на 5 минут.

```
update time_punch set punch_time = punch_time + interval '5
minute', change_employee_id = 2 where id = 2;
```

Проверьте содержимое таблицы аудита time_punch_audit.


```
SQL Shell (psql)
emp_time=# INSERT INTO time_punch (employee_id, is_out_punch, punch_time)
emp_time=# VALUES ((SELECT id FROM employee WHERE username = 'Редактор Михаила'), false, '2021-01-01 10:00:00'),
emp_time=# ((SELECT id FROM employee WHERE username = 'Редактор Михаила'), true, '2021-01-01 11:30:00');
INSERT 0 2
emp_time=# SELECT * FROM time_punch;
 id | employee_id | is_out_punch | punch_time | change_employee_id
-----+-----+-----+-----+-----
 1 |          1 | f           | 2021-01-01 10:00:00 |
 2 |          1 | t           | 2021-01-01 11:30:00 |
 3 |          1 | f           | 2021-01-01 10:00:00 |
 4 |          1 | t           | 2021-01-01 11:30:00 |
 5 |          1 | f           | 2021-01-01 11:30:00 |
 6 |          1 | t           | 2021-01-01 10:00:00 |
 7 |          3 | f           | 2021-01-01 10:00:00 |
 8 |          3 | t           | 2021-01-01 11:30:00 |
(8 строк)

emp_time=# SELECT * FROM logs;
 text | added
-----+-----
Add new user 2 | 2023-11-23 16:32:44.085872
Update user 1 | 2023-11-23 16:35:37.341799
Remove user 2 | 2023-11-23 16:37:45.396171
Update user 1 | 2023-11-23 16:38:37.239725
Add new user 3 | 2023-11-23 16:47:10.557287
(5 строк)

emp_time=# update time_punch set punch_time = punch_time + interval '5 minute', change_employee_id = 2 where id = 2;
ОШИБКА: INSERT или UPDATE в таблице "time_punch" нарушает ограничение внешнего ключа "time_punch_change_employee_id_fkey"
ПОДРОБНОСТИ: Ключ (change_employee_id)=(2) отсутствует в таблице "employee".
emp_time=# update time_punch set punch_time = punch_time + interval '5 minute', change_employee_id = 3 where id = 3;
UPDATE 1
emp_time=# SELECT * FROM logs;
 text | added
-----+-----
Add new user 2 | 2023-11-23 16:32:44.085872
Update user 1 | 2023-11-23 16:35:37.341799
Remove user 2 | 2023-11-23 16:37:45.396171
Update user 1 | 2023-11-23 16:38:37.239725
Add new user 3 | 2023-11-23 16:47:10.557287
(5 строк)
```

```
SQL Shell (psql)
СТРОКА 1: SELECT * FROM time_punchtime_punch_audit;
emp_time=# SELECT * FROM punchtime_audit;
ОШИБКА: отношение "punchtime_audit" не существует
СТРОКА 1: SELECT * FROM punchtime_audit;
emp_time=# SELECT * FROM time_punch_audit;
 id | change_time | change_employee_id | time_punch_id | punch_time
-----+-----+-----+-----+-----
 1 | 2023-11-23 16:55:41.34779 | 3 | 3 | 2021-01-01 10:00:00
 2 | 2023-11-23 16:58:05.336719 | 3 | 3 | 2021-01-01 10:05:00
(2 строки)
```