

Министерство науки и высшего образования Российской Федерации  
федеральное государственное автономное образовательное учреждение высшего образования  
«Национальный исследовательский университет ИТМО»

Факультет инфокоммуникационных технологий

**ЛАБОРАТОРНАЯ РАБОТА №3**

**ПРОЦЕДУРЫ, ФУНКЦИИ, ТРИГГЕРЫ В PostgreSQL**

**по дисциплине:**  
**«Проектирование и Реализация Баз Данных»**

**Выполнил:**  
студент II курса ИКТ  
группы К3241  
Селезнев Илья

Санкт-Петербург  
2022

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

**Задачи:**

1. Создать процедуры/функции согласно индивидуальному заданию и (согласно индивидуальному заданию, часть 4).
2. Создать триггер для логирования событий вставки, удаления, редактирования данных в базе данных PostgreSQL (согласно индивидуальному заданию, часть 5). Допустимо создать универсальный триггер или отдельные триггеры на логирование действий.

3.       Вариант 3. БД «Библиотека»

Описание предметной области: Каждая книга может храниться в нескольких экземплярах. Для каждого экземпляра известно место его хранения (комната, стеллаж, полка). Читателю не может быть выдано более 3-х книг одновременно. Книги выдаются читателям на срок не более 10 дней. БД должна содержать следующий минимальный набор сведений:

- Название (заглавие) издания;
- Автор (фамилия и имя (инициалы) или псевдоним автора издания);
- Язык, с которого выполнен перевод издания;
- Номер тома (части, книги, выпуска);
- Область знания;
- Вид издания (сборник, справочник, монография ...);
- Переводчик (фамилия и инициалы переводчика);
- Место издания (город);
- Год выпуска издания;
- Издательство (название издательства);
- Библиотечный шифр (например, ББК 32.973);
- Номер стеллажа в комнате;
- Номер (инвентарный номер) экземпляра;

- Номер комнаты (помещения для хранения экземпляров);
- Номер полки на стеллаже;
- Дата изъятия экземпляра с установленного места;
- Цена конкретного экземпляра;
- Номер читательского билета (формуляра);
- Фамилия читателя;
- Имя читателя;
- Отчество читателя;
- Адрес читателя;
- Телефон читателя.

Дополнить исходные данные информацией о читательском абонементе (выдаче книг).

#### **Задание 4. Создать хранимые процедуры:**

- Вывести список свободных аудиторий для проведения практических занятий заданной группы в заданное время.
- Вывести расписание занятий для заданного преподавателя.
- Вывести список аудиторий, в которых может разместиться заданная группа.

#### **Задание 5. Создать необходимые триггеры.**

## **Выполнение**

### **Хранимые процедуры**

#### **Задание 4. Создать хранимые процедуры:**

- Для проверки наличия экземпляров заданной книги в библиотеке (процедура должна возвращать количество экземпляров книги).

Создание процедуры

```
CREATE OR REPLACE FUNCTION GetCnt_Book (p_idBook int)
  RETURNS TABLE (
    p_name_book text,
    p_cnt_ex INT
  )
AS $$
BEGIN
  RETURN QUERY SELECT
```

```

        name_book,
        cast(count(id_example) as integer)
from books b
    join example e on e.id_book = b.id_book
    where b.id_book = p_idBook
    group by name_book;
END; $$
LANGUAGE 'plpgsql';

```

Вызов функции

```
select * from GetCnt_Book(1)
```

- Для ввода в базу данных новой книги.

```

CREATE OR REPLACE procedure AddBook (
    p_name_book text,
    p_author text,
    p_psevd_author text,
    p_part int,
    p_lang text,
    p_type_book text,
    p_field_know text,
    p_interpretor text,
    p_year_book int,
    p_id_izd int
)
AS $$
BEGIN
    insert into books (name_book, author, psevd_author, part, lang, type_book, field_know,
    interpreter, year_book, id_izd) values
        (p_name_book,
        p_author,
        p_psevd_author,
        p_part,
        p_lang,
        p_type_book,
        p_field_know,
        p_interpretor,
        p_year_book,
        p_id_izd);
END; $$
LANGUAGE 'plpgsql';

```

- Для ввода нового читателя (необходимо проверить наличие читателя в картотеке, чтобы не назначить ему номер вторично).

```

CREATE OR REPLACE procedure AddReaders (
    p_fio text,
    p_edu text,
    p_phone int,
    p_address text
)
AS $$

```

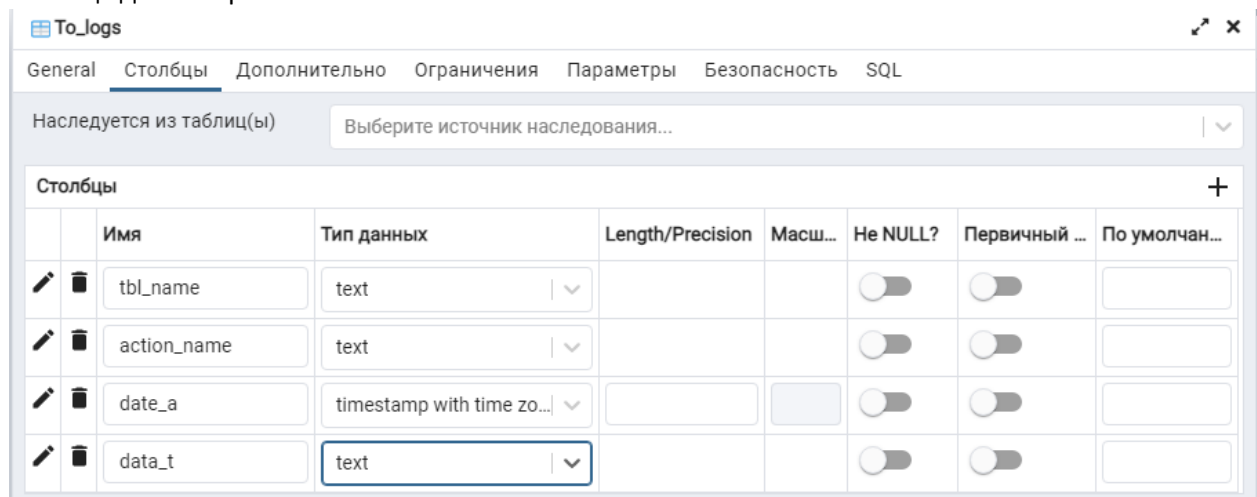
```

declare
    l_fio text;
BEGIN
    select into l_fio fio
    from readers
    where fio = p_fio;
    if l_fio <> '' then
        RAISE EXCEPTION 'Такой читатель уже существует!';
    else
        insert into readers(
            fio, edu, phone, address) values (p_fio,p_edu,p_phone,p_address);
    end if;
END; $$
LANGUAGE 'plpgsql';

```

**Задание 5.** Создать необходимые триггеры.

Таблица для логирования



Имя	Тип данных	Length/Precision	Масш...	Не NULL?	Первичный ...	По умолчан...
tbl_name	text			<input type="checkbox"/>	<input type="checkbox"/>	
action_name	text			<input type="checkbox"/>	<input type="checkbox"/>	
date_a	timestamp with time zo...			<input type="checkbox"/>	<input type="checkbox"/>	
data_t	text			<input type="checkbox"/>	<input type="checkbox"/>	

Процедура для триггера Persons

```

DECLARE
    l_action varchar(30);
    l_data text;
    l_tbl text;
BEGIN
    l_tbl := 'persons';
    IF TG_OP = 'INSERT' THEN
        l_action := 'Insert';
        l_data := NEW.fio;
        INSERT INTO To_logs(tbl_name, action_name, date_a, data_t) values (l_tbl, l_action, NOW(), l_data);
        RETURN NEW;
    ELSIF TG_OP = 'UPDATE' THEN
        l_action := 'Update';
        l_data := NEW.fio;
        INSERT INTO To_logs(tbl_name, action_name, date_a, data_t) values (l_tbl, l_action, NOW(), l_data);
        RETURN NEW;
    ELSIF TG_OP = 'DELETE' THEN
        l_action := 'Delete';
        l_data := NEW.fio;
        INSERT INTO To_logs(tbl_name, action_name, date_a, data_t) values (l_tbl, l_action, NOW(), l_data);

```

```
        RETURN OLD;
    END IF;
END;
```

Код триггера Persons

```
CREATE TRIGGER trg_pers
AFTER INSERT OR UPDATE OR DELETE
ON persons FOR EACH ROW
EXECUTE PROCEDURE add_to_log_pers();
```

Процедура для триггера Books

```
DECLARE
    l_action varchar(30);
    l_data text;
    l_tbl text;
BEGIN
    l_tbl := 'books';
    IF TG_OP = 'INSERT' THEN
        l_action := 'Insert';
        l_data := NEW.name_book;
        INSERT INTO To_logs(tbl_name, action_name, date_a, data_t) values (l_tbl, l_action, NOW(), l_data);
        RETURN NEW;
    ELSIF TG_OP = 'UPDATE' THEN
        l_action := 'Update';
        l_data := NEW.name_book;
        INSERT INTO To_logs(tbl_name, action_name, date_a, data_t) values (l_tbl, l_action, NOW(), l_data);
        RETURN NEW;
    ELSIF TG_OP = 'DELETE' THEN
        l_action := 'Delete';
        l_data := NEW.name_book;
        INSERT INTO To_logs(tbl_name, action_name, date_a, data_t) values (l_tbl, l_action, NOW(), l_data);
        RETURN OLD;
    END IF;
END;
```

Код триггера Books

```
CREATE TRIGGER trg_books
AFTER INSERT OR UPDATE OR DELETE
ON books FOR EACH ROW
EXECUTE PROCEDURE add_to_log_books();
```

Процедура для триггера Readers

```
DECLARE
    l_action varchar(30);
    l_data text;
    l_tbl text;
BEGIN
    l_tbl := 'readers';
    IF TG_OP = 'INSERT' THEN
        l_action := 'Insert';
        l_data := NEW.fio;
```

```

        INSERT INTO To_logs(tbl_name, action_name, date_a, data_t) values (l_tbl, l_action, NOW(), l_data);
        RETURN NEW;
    ELSIF TG_OP = 'UPDATE' THEN
        l_action := 'Update';
        l_data := NEW.fio;
        INSERT INTO To_logs(tbl_name, action_name, date_a, data_t) values (l_tbl, l_action, NOW(), l_data);
        RETURN NEW;
    ELSIF TG_OP = 'DELETE' THEN
        l_action := 'Delete';
        l_data := NEW.fio;
        INSERT INTO To_logs(tbl_name, action_name, date_a, data_t) values (l_tbl, l_action, NOW(), l_data);
        RETURN OLD;
    END IF;
END;

```

Код триггера Readers

```

CREATE TRIGGER trg_readers
AFTER INSERT OR UPDATE OR DELETE
ON readers FOR EACH ROW
EXECUTE PROCEDURE add_to_log_readers();

```

### Задание 1:

1. Создайте БД emp\_time. Используйте консоль psql.

```

postgres=# create database emp_time;
CREATE DATABASE

```

2. Проверьте список активных БД.

```

postgres=# \list

```

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

(5 строк)

3. Подключитесь к БД emp\_time.

```

postgres=# \c emp_time
Вы подключены к базе данных "emp_time" как пользователь "postgres".
emp_time=#

```

4. Создайте таблицы БД со следующей структурой:

Таблица employee

id (тип serial, первичный ключ)

username (тип varchar)

```

emp_time=# create table employee(id serial primary key, username varchar(40));
CREATE TABLE

```

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()).

```
emp_time=# create table time_punch (
emp_time(# id serial primary key,
emp_time(# employee_id int not null,
emp_time(# is_out_punch boolean not null default false,
emp_time(# punch_time timestamp not null default now(),
emp_time(# constraint employee_id_time_punch_employee_id_foreign foreign key (employee_id)
emp_time(# references employee (id));
CREATE TABLE
```

Вставьте данные в таблицы:

1. Вставьте в таблицу сотрудников строку для сотрудника с именем Михаил.

```
emp_time=# insert into employee (id, username) values (1, 'Михаил');
INSERT 0 1
```

2. Вставьте в таблицу учета времени две строки: вход Михаил в 10.00 2021-01-01 и выход в 11.30 2021-01-01.

```
emp_time=# insert into time_punch (id, employee_id, is_out_punch, punch_time) values (
emp_time(# 1, 1, true, '2021-01-01 10:00:00');
INSERT 0 1
emp_time=# insert into time_punch (id, employee_id, is_out_punch, punch_time) values (
emp_time(# 2, 1, false, '2021-01-01 11:30:00');
INSERT 0 1
```

3. Проверьте содержимое таблиц.

```
emp_time=# select * from employee;
 id | username
----+-----
  1 | Михаил
(1 row)

emp_time=# select * from time_punch;
 id | employee_id | is_out_punch |      punch_time
----+-----+-----+-----
  1 |          1 | t             | 2021-01-01 10:00:00
  2 |          1 | f             | 2021-01-01 11:30:00
```

### *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.



```
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
emp_time=# order by tps.id desc limit 1)
emp_time=# where tp1.employee_id = 1 and tp1.is_out_punch;
time_worked
-----
01:30:00
```

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.

### ***Триггер INSERT BEFORE: сохранение целостности данных***

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 *триггер*.

```
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 or replace function fn_check_time_punch() returns trigger
emp_time=# as $psql$
emp_time$# begin
emp_time$# if new.is_out_punch = (
emp_time$# select tps.is_out_punch
emp_time$# from time_punch tps
emp_time$# where tps.employee_id = new.employee_id
emp_time$# order by tps.id desc limit 1) then
emp_time$# return null;
emp_time$# end if;
emp_time$# return new;
emp_time$# end;
emp_time$# $psql$ language plpgsql;
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();
emp_time=# create trigger check_time_punch before insert on time_punch
emp_time=# for each row execute procedure fn_check_time_punch();
CREATE TRIGGER
```

### Задание 3:

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

```
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            | 2022-06-06 03:00:00
  4 |           1 | t            | 2022-06-06 13:46:47.607707
(4 rows)
```

2. Найдите “узкие” места в работе триггера.

3. Выполните запрос на просмотр данных в таблице учета времени.

```
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
emp_time=# order by tps.id desc limit 1)
emp_time=# where tp1.employee_id = 1 and tp1.is_out_punch;
      time_worked
-----
 01:30:00
10:46:47.607707
```

## Пример 2. Аудит изменения данных

### Вариант 1

Задача: реализовать самую простую систему логирования сотрудников. Она будет следить за изменениями в таблице сотрудников и при изменениях добавлять текстовые экшны в таблицу логов.

### Задание 4:

Создайте таблицу для хранения логов `logs` следующей структурой:

`text` (тип `text`),

`added` (тип `timestamp without time zone`)

```
emp_time=# create table logs (text text, added timestamp without time zone);  
CREATE TABLE
```

Триггер будет обрабатывать операции Update, Insert и Delete для таблицы employee, будет выполняться после (after) для каждой строки (for each row) вызывая функцию add\_to\_log():

```
CREATE TRIGGER t_employee AFTER INSERT OR UPDATE OR DELETE ON employee  
FOR EACH ROW EXECUTE PROCEDURE add_to_log ();
```

```
emp_time=# CREATE TRIGGER t_employee AFTER INSERT OR UPDATE OR DELETE ON employee FOR EACH ROW EXECUTE PROCEDURE add_to_  
log ();  
CREATE TRIGGER
```

Но до создания триггера нужно сначала создать функцию триггера add\_to\_log():

```
CREATE OR REPLACE FUNCTION add_to_log() RETURNS TRIGGER AS $$  
DECLARE  
mstr varchar(30);  
astr varchar(100);  
retstr varchar(254);  
BEGIN  
IF TG_OP = 'INSERT' THEN  
astr = NEW.id;  
mstr := 'Add new user ';  
retstr := mstr||astr;  
INSERT INTO logs(text,added) values (retstr,NOW());  
RETURN NEW;  
ELSIF TG_OP = 'UPDATE' THEN  
astr = NEW.id;  
mstr := 'Update user ';  
retstr := mstr||astr;  
INSERT INTO logs(text,added) values (retstr,NOW());  
RETURN NEW;  
ELSIF TG_OP = 'DELETE' THEN  
astr = OLD.id;  
mstr := 'Remove user ';  
retstr := mstr || astr;  
INSERT INTO logs(text,added) values (retstr,NOW());  
RETURN OLD;  
END IF;  
END;  
$$ LANGUAGE plpgsql;
```

```

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$$ ELSIF TG_OP = 'UPDATE' THEN
emp_time$$ 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$$ ELSIF TG_OP = 'DELETE' THEN
emp_time$$ 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;
CREATE FUNCTION

```

Пояснение:

Определяется новая функция, без входящих параметров, возвращает специальный тип TRIGGER.

Для внутреннего использования определяются 3-и переменные в разделе DECLARE.

В самой процедуре внутренняя переменная триггера TG\_OP определяет, с какой операцией была вызвана процедура.

В зависимости от операции определяется переменная mstr, собирается строка retstr, которая будет записана в базу данных (обратите внимание, как производится конкатенация строк в pgsql, через ||), и делается запись в таблицу логов (INSERT INTO).

Переменные NEW и OLD: Это строки, которые обрабатывает триггер. В случае INSERT переменная NEW будет содержать новую строку, а OLD будет пустая, в случае UPDATE обе переменные будут определены (соответствующими данными), а в случае DELETE переменная NEW будет пустая, OLD содержать удаляемую строку.

Если все правильно сделано, то теперь при любой работе с таблицей employee без нашего участия будет записываться лог действий с таблицей.

### Задание 5:

1. Протестируйте работу триггера, выполнив операции вставки, удаления и редактирования данных в таблице сотрудников.

```

emp_time=# insert into employee (id, username) values (2, 'SSS');
INSERT 0 1
emp_time=# update employee set username = 'ss' where id = 2;
UPDATE 1
emp_time=# delete from employee where id = 2;
DELETE 1

```

2. Проверьте содержание таблицы логов после выполнения операций.

```
emp_time=# select * from logs;
```

text	added
Add new user 2	2022-06-06 14:13:41.253074
Update user 2	2022-06-06 14:15:17.218356
Remove user 2	2022-06-06 14:15:40.530388

## 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
);
```

```
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
```

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);
emp_time=# alter table time_punch
emp_time=# add column change_employee_id int null references employee(id);
ALTER TABLE
```

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;
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;
CREATE FUNCTION

```

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();

```

```

emp_time=# create trigger change_time_punch_audit after update on time_punch for each row execute procedure fn_change_time_punch_audit();
CREATE TRIGGER

```

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.

```

emp_time=# insert into employee (id, username) values (2,'Georg');
INSERT 0 1

```

2. Пусть он будет редактором времени прохода Михаила.

3. Пусть необходимо отредактировать время выхода в 11:30:

```

emp=# select punch_time
emp-#
emp-# from time_punch
emp-#
emp-# where id=2;
        punch_time
-----
2020-01-01 11:30:00
(1 строка)

```

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;

```

```
emp_time=# update time_punch set punch_time = punch_time + interval '5 minute', change_employee_id = 2 where id = 2;
UPDATE 1
emp_time=# update time_punch set punch_time = punch_time + interval '5 minute', change_employee_id = 2 where id = 2;
UPDATE 1
```

## 5. Проверьте содержимое таблицы аудита

```
emp_time=# select * from time_punch_audit;
```

id	change_time	change_employee_id	time_punch_id	punch_time
1	2022-06-06 14:23:56.928807	2	2	2021-01-01 11:30:00
2	2022-06-06 14:24:00.344081	2	2	2021-01-01 11:35:00

## Выводы

Созданы хранимые процедуры и триггеры в SQL Shell.