

**Министерство науки и высшего образования Российской Федерации**  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«Национальный исследовательский университет ИТМО»**

**(Университет ИТМО)**

**Факультет прикладной информатики**

**Образовательная программа Мобильные и сетевые технологии**

**Направление подготовки 09.03.03 Мобильные и сетевые технологии**

**ОТЧЕТ по Лабораторной работе № 5**

**по дисциплине «Проектирование и реализация баз данных»**

**Обучающийся:** Майстренко Анастасия Николаевна К3241

**Преподаватель:** Говорова Марина Михайловна

Санкт-Петербург,

2025

## Цель работы

Приобрести практические навыки создания и использования процедур, функций и триггеров в PostgreSQL

## Практическое задание:

По варианту индивидуальной БД (ЛР 2, часть IV) необходимо:

1. Создать три хранимые процедуры.
2. Разработать семь оригинальных триггеров.
3. Проверить работу объектов в консоли psql, сделать скриншоты.

## Схема базы данных

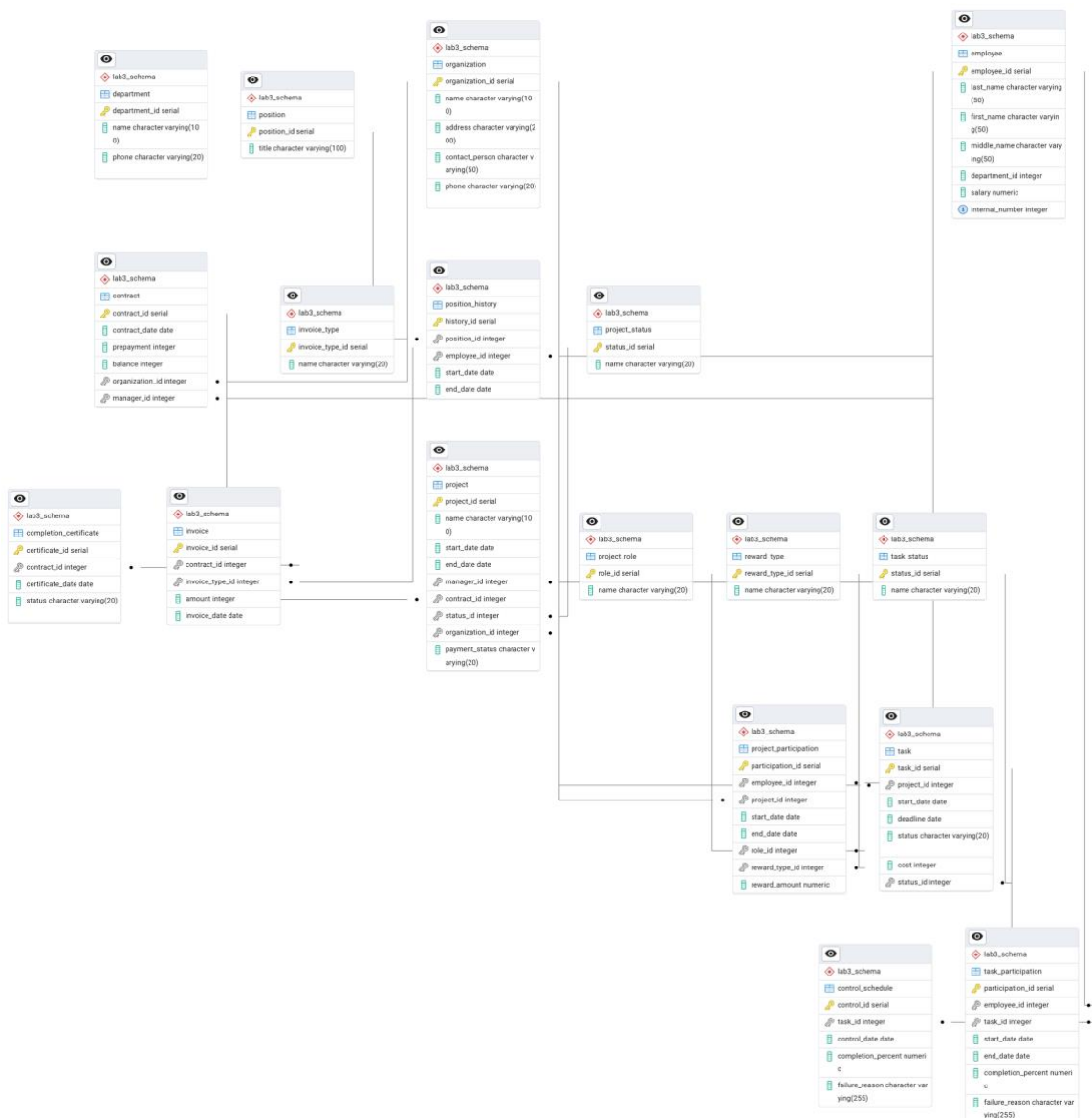


Рис.1: БД «Учет выполнения заданий»

## Разработка хранимых процедур

1. Для повышения оклада сотрудников, выполнивших задания с трехдневным опережением графика на заданный процент.

```
lab3_db=# CREATE OR REPLACE PROCEDURE increase_salary_for_early_completions(p_percent FLOAT)
LANGUAGE plpgsql
AS $$
BEGIN
    -- Обновляем оклад сотрудников, которые выполнили задание с опережением на 3 дня
    UPDATE lab3_schema.employee
    SET salary = salary + (salary * p_percent / 100)
    WHERE employee_id IN (
        SELECT e.employee_id
        FROM lab3_schema.employee e
        JOIN lab3_schema.task_participation tp ON e.employee_id = tp.employee_id
        JOIN lab3_schema.task t ON tp.task_id = t.task_id
        WHERE tp.end_date - tp.actual_date >= 3 -- Исправлено на сравнение разницы в днях
    );
END;
[$$;
CREATE PROCEDURE
```

```
lab3_db=# CALL increase_salary_for_early_completions(10);
CALL
lab3_db=# SELECT employee_id, last_name, first_name, salary
FROM lab3_schema.employee;
```

employee_id	last_name	first_name	salary
52	Иванов	Иван	50000
54	Иванов	Иван	50000
1	Иванов	Иван	106293.66
2	Петров	Пётр	55000
33	Иванов	Игорь	55000
35	Кузнецова	Елена	47000
36	Лебедев	Андрей	53000
38	Захарова	Светлана	49000
39	Григорьев	Юрий	62000
41	Морозова	Татьяна	53000
32	Петров	Александр	72600.00
34	Сидоров	Михаил	54450.00
37	Новикова	Мария	61710.00
40	Васильева	Наталья	72600.00
42	Петров	Василий	84700.00
45	Петров	Василий	84700.00
46	Петров	Василий	84700.00
47	Иванов	Алексей	60500.00
49	Иванов	Алексей	55000.0

(19 rows)

```
lab3_db=# SELECT employee_id, last_name, first_name, salary
FROM lab3_schema.employee;
```

employee_id	last_name	first_name	salary
52	Иванов	Иван	50000
1	Иванов	Иван	96630.6
54	Иванов	Иван	50000
2	Петров	Пётр	55000
33	Иванов	Игорь	55000
35	Кузнецова	Елена	47000
36	Лебедев	Андрей	53000
38	Захарова	Светлана	49000
39	Григорьев	Юрий	62000
41	Морозова	Татьяна	53000
32	Петров	Александр	72600.00
34	Сидоров	Михаил	54450.00
37	Новикова	Мария	61710.00
40	Васильева	Наталья	72600.00
42	Петров	Василий	84700.00
45	Петров	Василий	84700.00
46	Петров	Василий	84700.00
47	Иванов	Алексей	60500.00
49	Иванов	Алексей	55000.0

(19 rows)

```
CREATE OR REPLACE PROCEDURE
increase_salary_for_early_completions(p_percent FLOAT)
LANGUAGE plpgsql
AS $$
BEGIN
    -- Обновляем оклад сотрудников, которые выполнили задание с опережением
    на 3 дня
    UPDATE lab3_schema.employee
    SET salary = salary + (salary * p_percent / 100)
    WHERE employee_id IN (
        SELECT e.employee_id
        FROM lab3_schema.employee e
```

```

JOIN lab3_schema.task_participation tp ON e.employee_id = tp.employee_id
JOIN lab3_schema.task t ON tp.task_id = t.task_id
WHERE tp.end_date - tp.actual_date >= 3 -- Исправлено на сравнение
разницы в днях
);
END;
$$;

```

**CREATE OR REPLACE PROCEDURE** — создаём (или заменяем, если такая уже есть) процедуру с именем **increase\_salary\_for\_early\_completions**.

**p\_percent FLOAT** — это **входной параметр** для процедуры, который будет указывать процент повышения оклада. Его тип — **FLOAT** (вещественное число).

**LANGUAGE plpgsql** — мы используем язык **PL/pgSQL**, это встроенный язык для работы с процедурами и функциями в PostgreSQL.

**BEGIN ... END;** — это блок, внутри которого будет происходить выполнение всех действий процедуры.

**UPDATE lab3\_schema.employee** — обновляем таблицу **employee**, которая хранит информацию о сотрудниках.

**SET salary = salary + (salary \* p\_percent / 100)** — увеличиваем оклад сотрудника.

**salary \* p\_percent / 100** — вычисляем **прибавку к окладу**: например, если процент равен 10, то прибавляется **10%** от текущего оклада.

**salary + (...)** — прибавляем эту прибавку к текущему окладу.

**WHERE employee\_id IN (...)** — обновление будет выполнено **только для тех сотрудников**, чьи **employee\_id** попадают в список, возвращаемый подзапросом.

Подзапрос находит всех сотрудников, которые:

Выполнили задания с **опережением на 3 дня или больше**.

Для этого мы соединяем три таблицы:

1. **employee e** — таблица сотрудников.
2. **task\_participation tp** — таблица участия сотрудников в заданиях.

### 3. **task t** — таблица задач.

Мы выбираем сотрудников, для которых разница между датой **end\_date** (дедлайн задания) и **actual\_date**(фактическая дата выполнения задания) **больше или равна 3 дням**.

2. Для вычисления количества проектов, в выполнении которых участвует сотрудник.

```
lab3_db=# CREATE OR REPLACE PROCEDURE count_projects_for_employee(p_employee_id INT, OUT p_project_count INT)
LANGUAGE plpgsql
AS $$
BEGIN
    SELECT COUNT(DISTINCT pp.project_id) INTO p_project_count
    FROM lab3_schema.project_participation pp
    WHERE pp.employee_id = p_employee_id;
END;
$$;
CREATE PROCEDURE
lab3_db=# CALL count_projects_for_employee(2, p_project_count);
ERROR:  column "p_project_count" does not exist
LINE 1: CALL count_projects_for_employee(2, p_project_count);
                                                ^

lab3_db=# DO $$
DECLARE
    project_count INT;
BEGIN
    -- Вызов процедуры
    CALL count_projects_for_employee(2, project_count);

    -- Выводим результат
    RAISE NOTICE 'Количество проектов: %', project_count;
END;
$$;
NOTICE:  Количество проектов: 1
DO
```

```
CREATE OR REPLACE PROCEDURE count_projects_for_employee(p_employee_id
INT, OUT p_project_count INT)
LANGUAGE plpgsql
AS $$
BEGIN
    -- Считаем количество проектов, в которых участвует сотрудник
    SELECT COUNT(DISTINCT pp.project_id) INTO p_project_count
    FROM lab3_schema.project_participation pp
    WHERE pp.employee_id = p_employee_id;
END;
$$;
```

**CREATE OR REPLACE PROCEDURE** — эта строка говорит о том, что мы создаём процедуру или заменяем её, если она уже существует.

**p\_employee\_id INT** — это **входной параметр**, который мы передаем в процедуру. Это **employee\_id** сотрудника, для которого нужно посчитать количество проектов.

**OUT p\_project\_count INT** — это **выходной параметр**, который будет возвращать **количество проектов** (целое число **INT**), в которых участвует сотрудник.

**BEGIN ... END;** — это начало и конец блока кода процедуры, внутри которого выполняются все действия.

**SELECT COUNT(DISTINCT pp.project\_id) INTO p\_project\_count:**

**COUNT(DISTINCT pp.project\_id)** — считает **уникальные project\_id**, то есть количество **разных проектов**, в которых участвует сотрудник. Использование **DISTINCT** гарантирует, что один и тот же проект не будет посчитан несколько раз, если сотрудник участвует в одном проекте несколько раз.

**INTO p\_project\_count** — результат этого подсчета сохраняется в выходной параметр **p\_project\_count**.

**FROM lab3\_schema.project\_participation pp** — выбираем данные из таблицы **project\_participation**, которая связывает сотрудников с проектами.

**WHERE pp.employee\_id = p\_employee\_id** — фильтруем только те записи, где **employee\_id** соответствует переданному параметру **p\_employee\_id**, т.е. ищем проекты именно для указанного сотрудника.

Как работает процедура:

1. Мы передаем **employee\_id** как входной параметр.
2. Процедура подсчитывает количество **уникальных проектов** для этого сотрудника в таблице **project\_participation**.
3. Результат (количество проектов) сохраняется в **p\_project\_count**.
4. Когда процедура выполнена, **p\_project\_count** будет содержать количество проектов, в которых участвует сотрудник.

### 3. Для поиска номера телефона сотрудника (телефон установлен в каждом отделе)

```
lab3_db=# CREATE OR REPLACE PROCEDURE find_employee_phone(p_employee_id INT, OUT p_phone_number TEXT)
LANGUAGE plpgsql
AS $$
BEGIN
    -- Ищем номер телефона сотрудника, основываясь на его отделе
    SELECT d.phone INTO p_phone_number
    FROM lab3_schema.employee e
    JOIN lab3_schema.department d ON e.department_id = d.department_id
    WHERE e.employee_id = p_employee_id;
END;
$$;
CREATE PROCEDURE
lab3_db=# DO $$
DECLARE
    phone_number TEXT;
BEGIN
    -- Вызов процедуры
    CALL find_employee_phone(2, phone_number);

    -- Выводим результат
    RAISE NOTICE 'Номер телефона сотрудника: %', phone_number;
END;
$$;
NOTICE:  Номер телефона сотрудника: 222-222
DO
```

```
CREATE OR REPLACE PROCEDURE find_employee_phone(p_employee_id INT,
OUT p_phone_number TEXT)
LANGUAGE plpgsql
AS $$
BEGIN
    -- Ищем номер телефона сотрудника, основываясь на его отделе
    SELECT d.phone INTO p_phone_number
    FROM lab3_schema.employee e
    JOIN lab3_schema.department d ON e.department_id = d.department_id
    WHERE e.employee_id = p_employee_id;
END;
$$;
```

**p\_employee\_id** — входной параметр, который мы передаем в процедуру. Это **employee\_id** сотрудника, для которого ищем номер телефона.

**p\_phone\_number** — выходной параметр, который будет содержать номер телефона сотрудника.

Мы **соединяем** таблицы **employee** и **department** по полю **department\_id**, чтобы найти номер телефона сотрудника.

**SELECT d.phone INTO p\_phone\_number** — выбираем номер телефона из таблицы **department** и сохраняем его в переменную **p\_phone\_number**.

1. Мы объявляем переменную **phone\_number** для хранения номера телефона.
2. Выполняем вызов процедуры с **employee\_id = 2** и получаем номер телефона сотрудника.
3. С помощью **RAISE NOTICE** выводим номер телефона в консоль.

## Разработка триггеров

### Триггер 1: Логирование изменений в таблице сотрудников

```
lab3_db=# CREATE OR REPLACE FUNCTION log_employee_changes()
RETURNS trigger AS $$
BEGIN
    INSERT INTO lab3_schema.employee_log (action, employee_id, changed_at, old_data, new_data)
    VALUES (TG_OP, NEW.employee_id, NOW(), row_to_json(OLD), row_to_json(NEW));
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER employee_log_trigger
AFTER INSERT OR UPDATE OR DELETE ON lab3_schema.employee
FOR EACH ROW
EXECUTE FUNCTION log_employee_changes();
CREATE FUNCTION
CREATE TRIGGER
```

CREATE OR REPLACE FUNCTION log\_employee\_changes()

RETURNS trigger AS \$\$

BEGIN

INSERT INTO lab3\_schema.employee\_log (action, employee\_id, changed\_at,  
old\_data, new\_data)

VALUES (TG\_OP, NEW.employee\_id, NOW(), row\_to\_json(OLD),  
row\_to\_json(NEW));

RETURN NEW;

END;

\$\$ LANGUAGE plpgsql;

CREATE TRIGGER employee\_log\_trigger

AFTER INSERT OR UPDATE OR DELETE ON lab3\_schema.employee

FOR EACH ROW



```
EXECUTE FUNCTION log_employee_changes();
```

The screenshot shows a database management interface. On the left, a tree view displays 'Triggers (1)' expanded, showing 'employee\_log\_trigger' and 'public.log\_employee'. The main area shows a SQL query: `SELECT * FROM lab3_schema.employee_log;`. Below the query, the 'Data Output' tab is active, showing a table with 7 columns: `log_id` (integer), `action` (text), `employee_id` (integer), `changed_at` (timestamp without time zone), `old_data` (jsonb), and `new_data` (jsonb). The table contains one row with the following values: `log_id` is 1, `action` is 'INSERT', `employee_id` is 56, `changed_at` is '2025-06-22 23:16:14.464082', `old_data` is '[null]', and `new_data` is '({\"salary\": 50000, \"last\_name\": \"Иванов\", \"first\_name\": \"Иван\", \"employee\_id\": 56, \"middle\_name\": null, \"departm.

Логирует все изменения в таблице **employee**: добавление, обновление или удаление сотрудников.

Записывает информацию в таблицу **employee\_log**:

**action**: тип операции (INSERT, UPDATE, DELETE)

**employee\_id**: идентификатор сотрудника

**changed\_at**: временная метка, когда операция была выполнена

**old\_data**: старые данные до изменений (в формате JSON)

**new\_data**: новые данные после изменений (в формате JSON)

## Триггер 2: Запрещаем удаление сотрудника, если у него есть активные проекты

```
lab3_db=# CREATE OR REPLACE FUNCTION prevent_delete_employee_if_in_project()
RETURNS trigger AS $$
BEGIN
    IF EXISTS (SELECT 1 FROM lab3_schema.project_participation WHERE employee_id = OLD.employee_id) THEN
        RAISE EXCEPTION 'Невозможно удалить сотрудника с активными проектами';
    END IF;
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER prevent_delete_employee_trigger
BEFORE DELETE ON lab3_schema.employee
FOR EACH ROW
EXECUTE FUNCTION prevent_delete_employee_if_in_project();
CREATE FUNCTION
CREATE TRIGGER
```

CREATE OR REPLACE FUNCTION prevent\_delete\_employee\_if\_in\_project()

RETURNS trigger AS \$\$

BEGIN

IF EXISTS (SELECT 1 FROM lab3\_schema.project\_participation WHERE  
employee\_id = OLD.employee\_id) THEN

RAISE EXCEPTION 'Невозможно удалить сотрудника с активными  
проектами';

END IF;

RETURN OLD;

END;

\$\$ LANGUAGE plpgsql;

CREATE TRIGGER prevent\_delete\_employee\_trigger

BEFORE DELETE ON lab3\_schema.employee

FOR EACH ROW

EXECUTE FUNCTION prevent\_delete\_employee\_if\_in\_project();

Query

Query History

1

DELETE FROM lab3\_schema.employee WHERE employee\_id = 1;

2

Data Output

Messages

Notifications

ERROR: Невозможно удалить сотрудника с активными проектами

CONTEXT: PL/pgSQL function prevent\_delete\_employee\_if\_in\_project() line 4 at RAISE

SQL state: P0001

Этот триггер предотвращает удаление сотрудника, если он участвует в активных проектах.

При попытке удалить сотрудника из таблицы **employee**, триггер проверяет, есть ли у него записи в таблице **project\_participation**.

Если у сотрудника есть активные проекты, триггер возбуждает исключение с сообщением "Невозможно удалить сотрудника с активными проектами".

### Триггер 3: Автоматическое обновление времени последнего входа

```
lab3_db=# CREATE OR REPLACE FUNCTION update_last_login()
RETURNS trigger AS $$
BEGIN
    NEW.last_login = NOW();
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER update_last_login_trigger
BEFORE UPDATE ON lab3_schema.employee
FOR EACH ROW
EXECUTE FUNCTION update_last_login();
CREATE FUNCTION
CREATE TRIGGER
```

```
CREATE OR REPLACE FUNCTION update_last_login()
RETURNS trigger AS $$
BEGIN
    NEW.last_login = NOW();
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER update_last_login_trigger
BEFORE UPDATE ON lab3_schema.employee
FOR EACH ROW
EXECUTE FUNCTION update_last_login();
```

Этот триггер обновляет поле **last\_login** в таблице **employee** на текущее время каждый раз, когда сотрудник обновляет свои данные в системе.

При обновлении записи сотрудника триггер устанавливает текущую дату и время в поле **last\_login**.

Query Query History

1 SELECT last\_login FROM lab3\_schema.employee WHERE employee\_id = 1;  
2

Data Output Messages Notifications

+

📄

▼

📋

▼

🗑️

🗑️

📥

⬇️

📈

SQL

Showing results

	last_login timestamp without time zone
1	2025-06-22 23:49:29.811319

#### Триггер 4: Логирование изменений в таблице времени работы сотрудников

```

CREATE TRIGGER
lab3_db=# CREATE OR REPLACE FUNCTION log_time_punch_changes()
RETURNS trigger AS $$
BEGIN
    INSERT INTO lab3_schema.time_punch_log (employee_id, action, punch_time, old_data, new_data)
    VALUES (NEW.employee_id, TG_OP, NOW(), row_to_json(OLD), row_to_json(NEW));
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER time_punch_log_trigger
AFTER INSERT OR UPDATE OR DELETE ON lab3_schema.time_punch
FOR EACH ROW
EXECUTE FUNCTION log_time_punch_changes();
CREATE FUNCTION

```

```

CREATE OR REPLACE FUNCTION log_time_punch_changes()
RETURNS trigger AS $$
BEGIN
    INSERT INTO lab3_schema.time_punch_log (employee_id, action, punch_time,
old_data, new_data)
    VALUES (NEW.employee_id, TG_OP, NOW(), row_to_json(OLD),
row_to_json(NEW));

```

```
RETURN NEW;  
END;  
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER time_punch_log_trigger  
AFTER INSERT OR UPDATE OR DELETE ON lab3_schema.time_punch  
FOR EACH ROW  
EXECUTE FUNCTION log_time_punch_changes();
```

QueryQuery History

12

```
SELECT * FROM lab3_schema.time_punch_log;
```

Scratch Pad X

Data OutputMessagesNotifications

Showing rows: 1 to 1Page No: 1of 1

	log_id [PK] integer	employee_id integer	action text	punch_time timestamp without time zone	old_data jsonb	new_data jsonb
1	1	1	INSERT	2025-06-22 23:50:27.858242	[null]	{ "punch_id": 3, "punch_time": "2025-06-21T08:00:00", "employee_id": 1, "is_out_punch": false }

После обновления записи

QueryQuery History

12

```
SELECT * FROM lab3_schema.time_punch_log;
```

Scratch Pad X

Data OutputMessagesNotifications

Showing rows: 1 to 2Page No: 1of 1

	log_id [PK] integer	employee_id integer	action text	punch_time timestamp without time zone	old_data jsonb	new_data jsonb
1	1	1	INSERT	2025-06-22 23:50:27.858242	[null]	{ "punch_id": 3, "punch_time": "2025-06-21T08:00:00", "employee_id": 1, "is_out_punch": false }
2	2	1	UPDATE	2025-06-22 23:51:02.977898	{ "punch_id": 3, "punch_time": "2025-06-21T08:00:00", "employee_id": 1, "is_out_punch": false }	{ "punch_id": 3, "punch_time": "2025-06-21T08:00:00", "employee_id": 1, "is_out_punch": false }

Этот триггер логирует изменения в таблице **time\_punch**, где отслеживается время входа/выхода сотрудников.

Логируются все изменения: добавление, обновление или удаление записей о времени сотрудников.

В таблице **time\_punch\_log** сохраняются данные:

**employee\_id**: идентификатор сотрудника

**action**: тип операции (INSERT, UPDATE, DELETE)

**punch\_time**: временная метка, когда операция была выполнена

**old\_data**: старые данные (в формате JSON)

**new\_data**: новые данные (в формате JSON)

#### Триггер 5: Проверка минимальной зарплаты сотрудника

```
lab3_db=# CREATE OR REPLACE FUNCTION check_min_salary()
RETURNS trigger AS $$
BEGIN
    IF NEW.salary < 20000 THEN
        RAISE EXCEPTION 'Зарплата не может быть меньше 20,000';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER check_min_salary_trigger
BEFORE INSERT OR UPDATE ON lab3_schema.employee
FOR EACH ROW
EXECUTE FUNCTION check_min_salary();
CREATE FUNCTION
CREATE TRIGGER
```

```
CREATE OR REPLACE FUNCTION check_min_salary()
RETURNS trigger AS $$
BEGIN
    IF NEW.salary < 20000 THEN
        RAISE EXCEPTION 'Зарплата не может быть меньше 20,000';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER check_min_salary_trigger
BEFORE INSERT OR UPDATE ON lab3_schema.employee
```

FOR EACH ROW  
EXECUTE FUNCTION check\_min\_salary();



The screenshot shows a database query editor with two tabs: 'Query' and 'Query History'. The 'Query' tab is active, displaying an SQL INSERT statement. Below the query, there are tabs for 'Data Output', 'Messages', and 'Notifications'. The 'Messages' tab is selected, showing an error message. The SQL statement is as follows:

```
1  INSERT INTO lab3_schema.employee (last_name, first_name, salary)
2  VALUES ('Сидоров', 'Сидор', 15000);
3  |
```

The error message in the 'Messages' tab is:

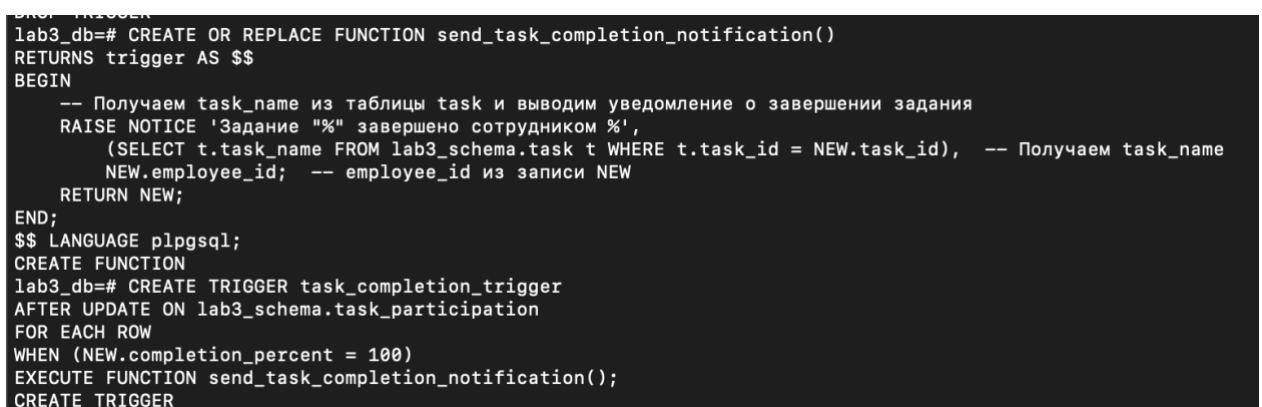
```
ERROR: Зарплата не может быть меньше 20,000
CONTEXT: PL/pgSQL function check_min_salary() line 4 at RAISE
SQL state: P0001
```

Этот триггер проверяет, чтобы зарплата сотрудника не была ниже установленного минимального значения (20,000).

При добавлении или обновлении записи о сотруднике, триггер проверяет, если его зарплата меньше 20,000.

Если зарплата ниже 20,000, триггер возбуждает исключение с сообщением "Зарплата не может быть меньше 20,000".

Триггер 6: Уведомление о завершении задания



The screenshot shows a database script for creating a function and a trigger. The script is as follows:

```
DROP TRIGGER
lab3_db=# CREATE OR REPLACE FUNCTION send_task_completion_notification()
RETURNS trigger AS $$
BEGIN
    -- Получаем task_name из таблицы task и выводим уведомление о завершении задания
    RAISE NOTICE 'Задание "%" завершено сотрудником %',
        (SELECT t.task_name FROM lab3_schema.task t WHERE t.task_id = NEW.task_id), -- Получаем task_name
        NEW.employee_id; -- employee_id из записи NEW
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE FUNCTION
lab3_db=# CREATE TRIGGER task_completion_trigger
AFTER UPDATE ON lab3_schema.task_participation
FOR EACH ROW
WHEN (NEW.completion_percent = 100)
EXECUTE FUNCTION send_task_completion_notification();
CREATE TRIGGER
```



```

CREATE OR REPLACE FUNCTION send_task_completion_notification()
RETURNS trigger AS $$
BEGIN
    -- Получаем task_name из таблицы task и выводим уведомление о
    -- завершении задания
    RAISE NOTICE 'Задание "%" завершено сотрудником %',
        (SELECT t.task_name FROM lab3_schema.task t WHERE t.task_id =
            NEW.task_id), -- Получаем task_name
        NEW.employee_id; -- employee_id из записи NEW
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER task_completion_trigger
AFTER UPDATE ON lab3_schema.task_participation
FOR EACH ROW
WHEN (NEW.completion_percent = 100)
EXECUTE FUNCTION send_task_completion_notification();

```

```

1  UPDATE lab3_schema.task_participation
2  SET completion_percent = 100
3  WHERE task_id = 1 AND employee_id = 2;
4

```

Data Output Messages Notifications

NOTICE: Задание "Анализ требований" завершено сотрудником 2  
 UPDATE 1

Query returned successfully in 45 msec.

Этот триггер отправляет уведомление (в виде сообщения) при завершении задания сотрудником.

Если **completion\_percent** задачи достигает 100%, триггер срабатывает и отправляет уведомление с информацией о том, что задание завершено.

Уведомление содержит имя задания и идентификатор сотрудника.

## Триггер 7: Автоматическое удаление устаревших записей

```
lab3_db=# CREATE OR REPLACE FUNCTION delete_old_time_punches()
RETURNS trigger AS $$
BEGIN
    DELETE FROM lab3_schema.time_punch
    WHERE punch_time < NOW() - INTERVAL '30 days';
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER delete_old_time_punches_trigger
AFTER INSERT ON lab3_schema.time_punch
FOR EACH ROW
EXECUTE FUNCTION delete_old_time_punches();
CREATE FUNCTION
CREATE TRIGGER
```

```
CREATE OR REPLACE FUNCTION delete_old_time_punches()
RETURNS trigger AS $$
BEGIN
    DELETE FROM lab3_schema.time_punch
    WHERE punch_time < NOW() - INTERVAL '30 days';
    RETURN NULL;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER delete_old_time_punches_trigger
AFTER INSERT ON lab3_schema.time_punch
FOR EACH ROW
EXECUTE FUNCTION delete_old_time_punches();
```

Этот триггер автоматически удаляет записи, которые старше 30 дней из таблицы **time\_punch**.

Когда новая запись добавляется в таблицу **time\_punch**, триггер проверяет старые записи и удаляет те, которые старше 30 дней.

QueryQuery history

1

2

SELECT \* FROM lab3\_schema.time\_punch;

Data OutputMessagesNotifications

Showing rows: 1 to 1Page No: 1

	punch_id [PK] integer	employee_id integer	punch_time timestamp without time zone	is_out_punch boolean
1	3	1	2025-06-21 08:00:00	true

QueryQuery history

1

2

SELECT \* FROM lab3\_schema.time\_punch;

Data OutputMessagesNotifications

	punch_id [PK] integer	employee_id integer	punch_time timestamp without time zone	is_out_punch boolean
--	--------------------------	------------------------	---	-------------------------

## Выводы

В ходе выполнения работы:

- Разработала три процедуры

- Создала семь триггеров.
- Проверила их работоспособность, результаты зафиксированы скриншотами.

Использование процедур и триггеров позволяет концентрировать логику на стороне СУБД, упрощая клиентские приложения и обеспечивая целостность данных.