МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»**

**Факультет безопасности информационных технологий**

**Направление подготовки: 10.03.01 Информационная безопасность**

**Образовательная программа: "Информационная безопасность / Information security"**

**Дисциплина:**

**«*Информационная безопасность баз данных*»**

**ОТЧЕТ ПО ЛАБОРАТОРНОЙРАБОТЕ № 3**

***«Функции и триггеры в БД»***

**Выполнил студент:**

группа/поток 1.3

Бардышев Артём Антонович/\_\_\_\_\_\_\_\_\_\_

*Подпись*

**Проверил:**

Карманова Наталья Андреевна/\_\_\_\_\_\_\_\_\_\_

*Подпись*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Отметка о выполнении (один из вариантов:*

*отлично, хорошо, удовлетворительно, зачтено)*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Дата*

Санкт-Петербург

2025г.

1. **Цель работы: Получение навыков написание процедур, функций и триггеров в БД.**
2. **Теоретическая информация:**

**При наличии необходимости реализации сложного сценария работы с данными в БД применяются процедуры и триггеры. Они представляют собой код, написанный на одном из расширений SQL (для PostgreSQL PL/pgSQL), который позволяет реализовать более сложную обработку данных и имеет в своем арсенале операторы циклов, ветвлений и т.п. Процедура является определена функцией, которую может запускать пользователь, использую клиента БД. Триггер представляет собой операцию, которую нужно выполнить при возникновении события в БД, например, вставка записи в таблицу.**

**В качестве СУБД, используемой в лабораторной работе, предполагается PostgreSQL. В ней триггеры создаются на основе уже определенных ранее функций.**

**Оператор для создания процедуры**

**CREATE OR REPLACE FUNCTION функция() RETURNS тип AS**

**$$**

**BEGIN**

**команды;**

**END**

**$$ LANGUAGE plpgsql;**

**Для создания хранимой процедуры нужно определить ее в соответствии с представленным синтаксисом описать команды и возвращаемый результат, который может быть простым типом данных, набором данных, сложным набором данных, пользовательским набором данных.**

**Для последующего вызова хранимой процедуры можно воспользоваться синтаксисом, приведенным ниже:**

**Вызов функции**

**SELECT \* FROM функция();**

**Пример:**

**CREATE OR REPLACE FUNCTION max\_value() RETURNS real AS**

**$$**

**DECLARE**

**maxVal real;**

**BEGIN**

**maxVal := (select max(sensor\_value) from svalues);**

**RETURN maxVal;**

**END**

**$$ LANGUAGE plpgsql;**

**Для последующего создания триггера необходимо определить триггерную функцию в соответствии с синтаксисом ниже.**

**Оператор для создания функции для триггера**

**CREATE FUNCTION *функция* () RETURNS *trigger* AS $$   
DECLARE   
        *объявления;*  
BEGIN   
*команды;*  
END; $$   
LANGUAGE  plpgsql;**

**Пример:**

**CREATE OR REPLACE FUNCTION trigger\_update\_maxvalue() RETURNS trigger AS**

**$$**

**DECLARE**

**maxVal real;**

**BEGIN**

**IF NEW.sensor\_value>(select current\_val from max\_value)**

**THEN update max\_value set current\_val=NEW.sensor\_value;**

**END IF;**

**RETURN NEW;**

**END;**

**$$ LANGUAGE plpgsql;**

**Оператор определения триггера**

**CREATE TRIGGER *триггер*BEFORE | AFTER } { *событие* [ OR *событие* ] } ON *таблица*  
FOR EACH { ROW |  STATEMENT }   
EXECUTE PROCEDURE *функция* ( *аргументы* )**

**Добавление новых записей в таблицу производится оператором INSERT. Необходимо в явном виде указать таблицу, в которую добавляются строки, а также список атрибутов добавляемой записи и ее значения.**

**Пример:**

**CREATE TRIGGER trigger\_keep\_maxvalue   
AFTER INSERT OR UPDATE ON svalues FOR EACH ROW   
EXECUTE PROCEDURE trigger\_update\_maxvalue();**

**Все перечисленные операторы имеют возможности по использованию дополнительных параметров, с которыми необходимо ознакомиться в документации.**

1. **Задание**
2. **Написать процедуру, которая выполняет агрегацию значений в таблице и обновляет значение в другой таблице. Таким образом, чтобы при запуске пользователем информация в таблице обновлялась и содержала агрегированные значения из другой таблицы.**
3. **Написать триггер, который будет выполнять действие из 1 пункта автоматически при вставке записи в исходную таблицу. Таким образом, чтобы агрегированная информация всегда была актуальна.**
4. **Написать триггер, который на основании даты из вставляемой записи, вставлял ее в соответствующую таблицу.**
5. **Написать триггер, который при вставке в таблицу, производил подмену вставляемого значения в соответствии с уже существующим словарем.**
6. **Реализуйте триггер, который использует по крайней мере 2-3 специальных переменных (NEW, OLD, TG\_OP и др). Список специальных переменных для postgresql https://postgrespro.ru/docs/postgresql/9.6/plpgsql-trigger**

1. **Требования к оформлению отчета и защите**

**Отчет должен содержать код процедур и триггерных функций, выполненные операторы по каждому пункту задания и вывод, полученный при их выполнении.**

1. **Источники и информация для подготовки**

[**http://www.postgresql.org/docs/9.5/static/index.html**](http://www.postgresql.org/docs/9.5/static/index.html)

ХОД РАБОТЫ:

1) **Подготовка базы данных**

Для выполнения заданий создадим тестовые таблицы:

-- Таблица с исходными значениями

CREATE TABLE sensor\_data (

id SERIAL PRIMARY KEY,

sensor\_id INTEGER,

sensor\_value REAL,

record\_date TIMESTAMP

);

-- Таблица для агрегированных данных

CREATE TABLE aggregated\_data (

sensor\_id INTEGER PRIMARY KEY,

max\_value REAL,

min\_value REAL,

avg\_value REAL,

last\_update TIMESTAMP

);

-- Таблица для разделения по датам

CREATE TABLE sensor\_data\_2023 (

CHECK (record\_date >= '2023-01-01' AND record\_date < '2024-01-01')

) INHERITS (sensor\_data);

CREATE TABLE sensor\_data\_2024 (

CHECK (record\_date >= '2024-01-01' AND record\_date < '2025-01-01')

) INHERITS (sensor\_data);

-- Таблица словаря для подмены значений

CREATE TABLE value\_mapping (

original\_value REAL,

mapped\_value REAL

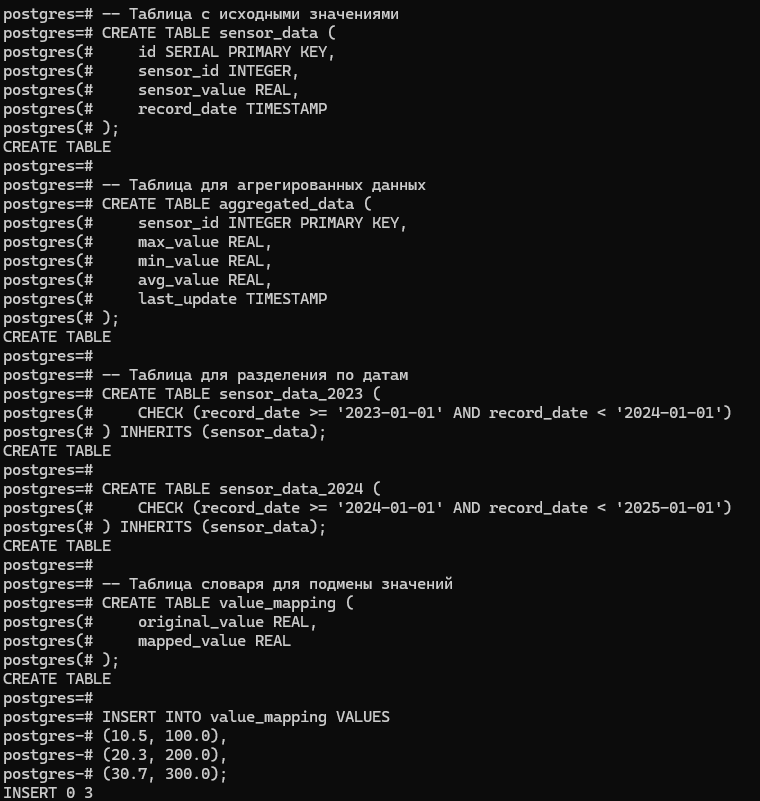
);

INSERT INTO value\_mapping VALUES

(10.5, 100.0),

(20.3, 200.0),

(30.7, 300.0);



2) **Процедура для агрегации данных**

**CREATE OR REPLACE FUNCTION update\_aggregated\_data()**

**RETURNS VOID AS $$**

**BEGIN**

**-- Удаляем старые агрегированные данные**

**DELETE FROM aggregated\_data;**

**-- Вставляем новые агрегированные данные**

**INSERT INTO aggregated\_data**

**SELECT**

**sensor\_id,**

**MAX(sensor\_value) AS max\_value,**

**MIN(sensor\_value) AS min\_value,**

**AVG(sensor\_value) AS avg\_value,**

**NOW() AS last\_update**

**FROM**

**sensor\_data**

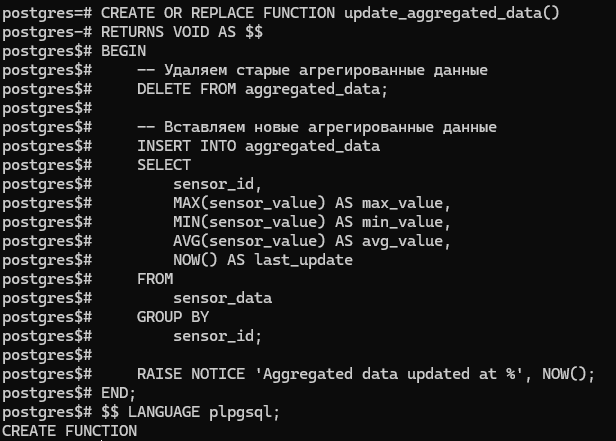
**GROUP BY**

**sensor\_id;**

**RAISE NOTICE 'Aggregated data updated at %', NOW();**

**END;**

**$$ LANGUAGE plpgsql;**



3) **Триггер для автоматического обновления агрегированных данных**

-- Сначала создаем триггерную функцию

CREATE OR REPLACE FUNCTION trigger\_update\_aggregated()

RETURNS TRIGGER AS $$

BEGIN

PERFORM update\_aggregated\_data();

RETURN NULL;

END;

$$ LANGUAGE plpgsql;

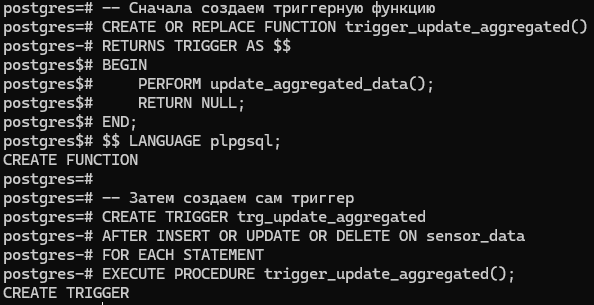
-- Затем создаем сам триггер

CREATE TRIGGER trg\_update\_aggregated

AFTER INSERT OR UPDATE OR DELETE ON sensor\_data

FOR EACH STATEMENT

EXECUTE PROCEDURE trigger\_update\_aggregated();



4) **Триггер для вставки в соответствующую таблицу по дате**

CREATE OR REPLACE FUNCTION trigger\_insert\_by\_date()

RETURNS TRIGGER AS $$

BEGIN

IF NEW.record\_date >= '2023-01-01' AND NEW.record\_date < '2024-01-01' THEN

INSERT INTO sensor\_data\_2023 VALUES (NEW.\*);

ELSIF NEW.record\_date >= '2024-01-01' AND NEW.record\_date < '2025-01-01' THEN

INSERT INTO sensor\_data\_2024 VALUES (NEW.\*);

ELSE

RAISE EXCEPTION 'Date out of range. No partition for year %',

EXTRACT(YEAR FROM NEW.record\_date);

END IF;

RETURN NULL; -- Отменяем вставку в основную таблицу

END;

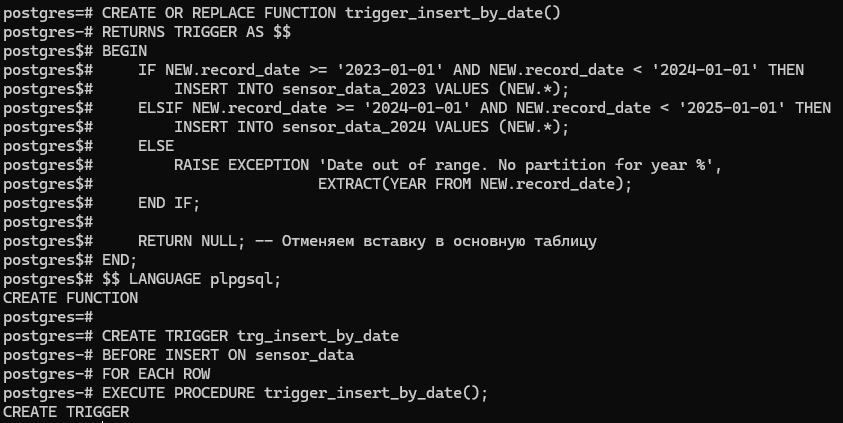
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_insert\_by\_date

BEFORE INSERT ON sensor\_data

FOR EACH ROW

EXECUTE PROCEDURE trigger\_insert\_by\_date();



5) **Триггер для подмены значений по словарю**

CREATE OR REPLACE FUNCTION trigger\_map\_values()

RETURNS TRIGGER AS $$

DECLARE

mapped\_val REAL;

BEGIN

SELECT mapped\_value INTO mapped\_val

FROM value\_mapping

WHERE original\_value = NEW.sensor\_value;

IF FOUND THEN

NEW.sensor\_value := mapped\_val;

RAISE NOTICE 'Mapped value from % to %',

OLD.sensor\_value, NEW.sensor\_value;

END IF;

RETURN NEW;

END;

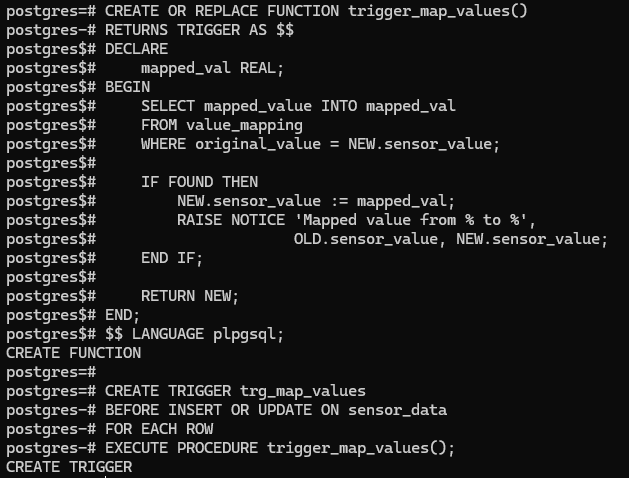
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_map\_values

BEFORE INSERT OR UPDATE ON sensor\_data

FOR EACH ROW

EXECUTE PROCEDURE trigger\_map\_values();



6) **Триггер с использованием специальных переменных**

CREATE OR REPLACE FUNCTION trigger\_log\_operations()

RETURNS TRIGGER AS $$

BEGIN

IF TG\_OP = 'INSERT' THEN

RAISE NOTICE 'Inserting new record with ID %, value %, date %',

NEW.id, NEW.sensor\_value, NEW.record\_date;

ELSIF TG\_OP = 'UPDATE' THEN

RAISE NOTICE 'Updating record ID %. Old value: %, new value: %',

NEW.id, OLD.sensor\_value, NEW.sensor\_value;

ELSIF TG\_OP = 'DELETE' THEN

RAISE NOTICE 'Deleting record ID % with value %',

OLD.id, OLD.sensor\_value;

END IF;

IF TG\_WHEN = 'BEFORE' THEN

RAISE NOTICE 'This is a BEFORE trigger';

ELSE

RAISE NOTICE 'This is an AFTER trigger';

END IF;

RAISE NOTICE 'Trigger name: %, Table name: %',

TG\_NAME, TG\_TABLE\_NAME;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER trg\_log\_insert

BEFORE INSERT ON sensor\_data

FOR EACH ROW

EXECUTE PROCEDURE trigger\_log\_operations();

CREATE TRIGGER trg\_log\_update

BEFORE UPDATE ON sensor\_data

FOR EACH ROW

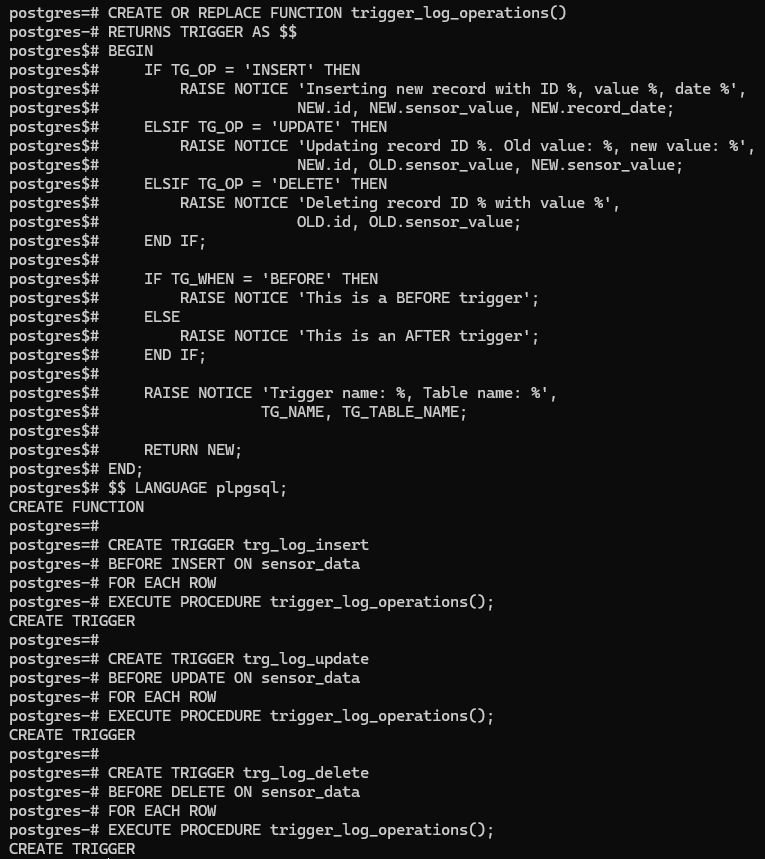
EXECUTE PROCEDURE trigger\_log\_operations();

CREATE TRIGGER trg\_log\_delete

BEFORE DELETE ON sensor\_data

FOR EACH ROW

EXECUTE PROCEDURE trigger\_log\_operations();



7) **Тестирование работы**

-- Вставка тестовых данных

INSERT INTO sensor\_data (sensor\_id, sensor\_value, record\_date) VALUES

(1, 10.5, '2023-05-10'),

(1, 20.3, '2023-06-15'),

(2, 30.7, '2024-02-20'),

(2, 15.0, '2024-03-25');

-- Проверка агрегированных данных

SELECT \* FROM aggregated\_data;

-- Обновление данных

UPDATE sensor\_data SET sensor\_value = 25.0 WHERE sensor\_value = 20.3;

-- Проверка обновленных агрегированных данных

SELECT \* FROM aggregated\_data;

-- Проверка разделения по датам

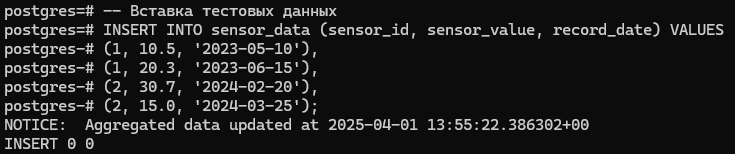
SELECT \* FROM sensor\_data\_2023;

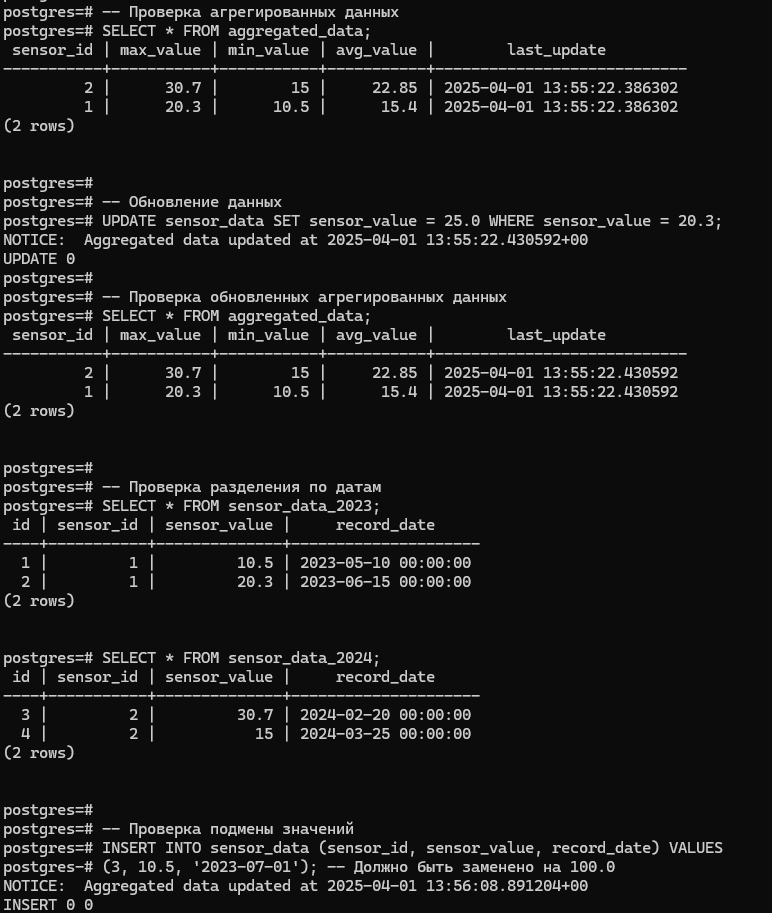
SELECT \* FROM sensor\_data\_2024;

-- Проверка подмены значений

INSERT INTO sensor\_data (sensor\_id, sensor\_value, record\_date) VALUES

(3, 10.5, '2023-07-01'); -- Должно быть заменено на 100.0





Вывод: в ходе лабораторной работы были реализованы:

1. Хранимая процедура для агрегации данных
2. Триггер для автоматического обновления агрегированных данных
3. Триггер для разделения данных по таблицам в зависимости от даты
4. Триггер для подмены значений по словарю
5. Триггер с использованием специальных переменных (NEW, OLD, TG\_OP, TG\_NAME, TG\_TABLE\_NAME, TG\_WHEN)

Все триггеры и процедуры работают корректно и выполняют поставленные задачи.