Санкт-Петербургский Национальный Исследовательский Университет

Информационных Технологий, Механики и Оптики

Факультет инфокоммуникационных технологий

Лабораторная работа №5

Выполнил:

Конопля А. К.

Проверила:

Говорова М. М.

Санкт-Петербург

2023

Введение

Данная лабораторная работа посвящена изучению и написанию триггеров и процедур на языке PostgreSQL.

Цель

Целью данной лабораторной работы является написание процедур и триггеров логирования изменения данных в базе.

Задачи

1. Создать процедуры/функции согласно индивидуальному заданию (часть 4).
    1) Снижение цен на продукт на указанный процент, в случае отсутствия продаж во введённый период;
    2) Процедура для возврата товара
    3) Для определения выручки магазина за отпрядённый период времени, по адресу
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.

Выполнение

Задание 1

Процедура 1

Снижение цен на продукт на указанный процент, в случае отсутствия продаж во введённый период.

```
CREATE OR REPLACE FUNCTION decrease_price_if_outdated(selected_item_id INT, period_of_time_in_days INT, deacrease_in_percent INT)
RETURNS VOID AS $$
DECLARE
    total_sales INT;
BEGIN
    SELECT COUNT(*) INTO total_sales
    FROM check_list
    JOIN checks ON check_list.check_id = checks.check_id
    WHERE checks.check_date BETWEEN CURRENT_DATE - INTERVAL '1 day' * period_of_time_in_days AND CURRENT_DATE
        AND check_list.item_id = selected_item_id;

    IF total_sales = 0 THEN
        UPDATE items
        SET price = price * (1 - deacrease_in_percent / 100.0)
        WHERE items.item_id = selected_item_id;

        RAISE NOTICE 'Цена продукта % была изменена. Нынешняя цена продукта: %', selected_item_id, (SELECT price FROM items WHERE items.item_id = selected_item_id);
    ELSE
        RAISE NOTICE 'Продукт % был продан в последние % дней', selected_item_id, period_of_time_in_days;
    END IF;
END;
$$
LANGUAGE plpgsql;
```

Код для запроса 1

```
NOTICE:  Цена продукта 1 была изменена. Нынешняя цена продукта: 794.7
```

Результат выполнения запроса

Процедура 2

Процедура для возврата товара.

```
CREATE OR REPLACE FUNCTION return_item(current_check_id INT, returned_item_id INT, current_returned_amount INT)
RETURNS INT AS
$$
DECLARE
    amount_in_check_list INT;
    returned_money_amount FLOAT;
BEGIN
    SELECT item_amount - returned_amount INTO amount_in_check_list
    FROM check_list
    JOIN checks ON check_list.check_id = checks.check_id
    WHERE check_list.item_id = returned_item_id AND checks.check_id = current_check_id;

    IF amount_in_check_list >= current_returned_amount THEN
        UPDATE check_list
        SET returned_amount = returned_amount + current_returned_amount
        WHERE check_list.check_id = current_check_id AND check_list.item_id = returned_item_id;

        SELECT current_returned_amount * price INTO returned_money_amount
        FROM items
        WHERE items.item_id = returned_item_id;

        RAISE NOTICE 'Было возвращено % рублей.', returned_money_amount;
        RETURN current_returned_amount;
    ELSE
        RAISE NOTICE 'Количество купленного товара не соответствует количеству товара в чеке.';
        RETURN 0;
    END IF;
END;
$$
LANGUAGE plpgsql;
```

Код для создания процедуры

```
SELECT return_item(1, 1, 1)
```

Data Output | Сообщения | Notifications

```
NOTICE:  Было возвращено 794.7 рублей.
```

Код и результат выполнения программы

Процедура 3

Для определения выручки магазина за отпрядённый период времени, по адресу.

```
CREATE OR REPLACE FUNCTION calculate_revenue_by_address(
    current_store_address VARCHAR,
    start_date DATE,
    end_date DATE
) RETURNS NUMERIC AS
$$
DECLARE
    total_revenue NUMERIC;
BEGIN
    SELECT SUM(items.price * check_list.item_amount)
    INTO total_revenue
    FROM stores
    JOIN employees ON stores.store_id = employees.store_id
    JOIN checks ON employees.employee_id = checks.employee_id
    JOIN check_list ON checks.check_id = check_list.check_id
    JOIN items ON check_list.item_id = items.item_id
    WHERE stores.store_address = current_store_address
    AND checks.check_date BETWEEN start_date AND end_date;

    IF total_revenue IS NULL THEN
        total_revenue := 0;
    END IF;
    RAISE NOTICE 'доход магазина в период с % по % составил %', start_date, end_date, total_revenue;
    RETURN total_revenue;
END;
$$
LANGUAGE plpgsql;

```

Код для создания запроса

```
SELECT calculate_revenue_by_address('Lenina street,2', '2020-01-01', '2022-12-31')
```

Data Output | Сообщения | Notifications

```
NOTICE:  доход магазина в период с 2020-01-01 по 2022-12-31 составил 33503.3
```

Вызов функции и результат вызова

Задание 2

Триггер логирования изменения данных.

```sql
CREATE OR REPLACE FUNCTION log_audit_changes()
RETURNS TRIGGER AS $$
DECLARE
    audit_row audit_log;
BEGIN
    audit_row.event_date = current_timestamp;
    audit_row.logged_table_name = TG_TABLE_NAME;
    IF TG_OP = 'INSERT' THEN
        audit_row.event_type = 'INSERT';
        audit_row.new_data = to_jsonb(NEW);
    ELSIF TG_OP = 'UPDATE' THEN
        audit_row.event_type = 'UPDATE';
        audit_row.new_data = to_jsonb(NEW);
        audit_row.old_data = to_jsonb(OLD);
    ELSIF TG_OP = 'DELETE' THEN
        audit_row.event_type = 'DELETE';
        audit_row.old_data = to_jsonb(OLD);
    END IF;

    INSERT INTO audit_log (event_type, event_date, logged_table_name, new_data, old_data)
    VALUES (audit_row.event_type, audit_row.event_date, audit_row.logged_table_name, audit_row.new_data, audit_row.old_data);

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

Код для создания универсального триггера логирования

```sql
CREATE TABLE audit_log (
    log_id SERIAL PRIMARY KEY,
    event_type VARCHAR(20) NOT NULL,
    event_date TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    logged_table_name VARCHAR(50) NOT NULL,
    new_data JSONB,
    old_data JSONB
);
```

Код для создания таблицы логирования

```sql
CREATE OR REPLACE FUNCTION create_audit_trigger()
RETURNS VOID AS $$
DECLARE
    logged_table_name RECORD;
BEGIN
    FOR logged_table_name IN SELECT table_name FROM information_schema.tables WHERE table_schema = 'public' AND table_type = 'BASE TABLE' LOOP
        EXECUTE 'CREATE TRIGGER audit_trigger_' || logged_table_name.table_name ||
                ' AFTER INSERT OR UPDATE OR DELETE ON ' || logged_table_name.table_name ||
                ' FOR EACH ROW EXECUTE FUNCTION log_audit_changes()';
    END LOOP;
END;
$$ LANGUAGE plpgsql;
SELECT create_audit_trigger();
```

Функция для создания индивидуальных триггеров для каждой таблицы в базе данных

Вывод

В ходе выполнения данной лабораторной работы были изучены и написаны процедуры и триггур для логирования изменения данных в базе данных.

Список источников:

1. Документация PostgreSQL [Электронный ресурс] // Официальный сайт PostgreSQL. 1996-2023. URL: https://www.postgresql.org/docs/13/index.html (дата обращения: 11.02.2023).
2. Документация pgAdmin 4 PostgreSQL [Электронный ресурс] // Официальный сайт pgAdmin. URL: https://www.pgadmin.org/docs/pgadmin4/latest/ (дата обращения: 11.02.2023)