Лабораторная работа №12 Триггеры в PostgreSQL

Цель работы:

Освоить создание и использование триггеров в PostgreSQL. Научиться автоматизировать задачи аудита, валидации данных и поддержания согласованности данных.

Программное обеспечение: PostgreSQL, DBeaver
Время: 2 академических часа

Подготовка базы данных

Создание таблиц

```
SET search_path TO ваш_логин;

-- Таблица пользователей
CREATE TABLE users (
    user_id SERIAL PRIMARY KEY,
    username VARCHAR(50) UNIQUE NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    full_name VARCHAR(100),
    date_of_birth DATE,
    registration_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    last_login TIMESTAMP,
    is_active BOOLEAN DEFAULT TRUE,
    login_attempts INTEGER DEFAULT 0,
    account_balance DECIMAL(10,2) DEFAULT 0.00,
    user_level VARCHAR(20) DEFAULT 'regular'
);

-- Таблица для аудита изменений
CREATE TABLE audit_log (
    audit_id SERIAL PRIMARY KEY,
    table_name VARCHAR(50) NOT NULL,
    operation VARCHAR(10) NOT NULL,
```

```
    user_id INTEGER,
    old_data JSONB,
    new_data JSONB,
    changed_by VARCHAR(100) DEFAULT current_user,
    change_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    ip_address INET
);

-- Таблица истории изменений пользователей
CREATE TABLE user_history (
    history_id SERIAL PRIMARY KEY,
    user_id INTEGER NOT NULL,
    field_name VARCHAR(50) NOT NULL,
    old_value TEXT,
    new_value TEXT,
    change_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    changed_by VARCHAR(100) DEFAULT current_user
);

-- Таблица заблокированных пользователей
CREATE TABLE blocked_users (
    block_id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(user_id),
    reason TEXT,
    blocked_by VARCHAR(100),
    block_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    unblock_time TIMESTAMP
);
```

Заполнение таблиц тестовыми данными

```
INSERT INTO users (username, email, full_name, date_of_birth,
account_balance, user_level) VALUES
('ivanov', 'ivanov@example.com', 'Иван Иванов', '1990-05-15', 1000.00,
'regular'),
('petrova', 'petrova@example.com', 'Мария Петрова', '1995-08-20',
2500.50, 'premium'),
('sidorov', 'sidorov@example.com', 'Алексей Сидоров', '1985-03-10',
500.00, 'regular'),
('kozlov', 'kozlov@example.com', 'Дмитрий Козлов', '2000-11-30', 0.00,
'regular'),
```

```
('smirnova', 'smirnova@example.com', 'Анна Смирнова', '1992-07-25',
15000.75, 'vip');
```

Задания с примерами выполнения

Задание 1: Триггер для автоматического аудита

Задача: Создать триггер, который автоматически записывает в таблицу `audit_log` информацию о всех изменениях в таблице `users`.

Пример выполнения:

```
-- 1. Создаем триггерную функцию
CREATE OR REPLACE FUNCTION audit_trigger_function()
RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = 'INSERT' THEN
        INSERT INTO audit_log (table_name, operation, user_id, new_data)
        VALUES (TG_TABLE_NAME, TG_OP, NEW.user_id, row_to_json(NEW));
    ELSIF TG_OP = 'UPDATE' THEN
        INSERT INTO audit_log (table_name, operation, user_id, old_data,
new_data)
        VALUES (TG_TABLE_NAME, TG_OP, NEW.user_id, row_to_json(OLD),
row_to_json(NEW));
    ELSIF TG_OP = 'DELETE' THEN
        INSERT INTO audit_log (table_name, operation, user_id, old_data)
        VALUES (TG_TABLE_NAME, TG_OP, OLD.user_id, row_to_json(OLD));
    END IF;

    RETURN NULL;
END;
$$ LANGUAGE plpgsql;

-- 2. Создаем триггер
CREATE TRIGGER users_audit_trigger
AFTER INSERT OR UPDATE OR DELETE ON users
FOR EACH ROW
EXECUTE FUNCTION audit_trigger_function();

-- 3. Тестируем триггер
```

```sql
-- Тест INSERT
INSERT INTO users (username, email, full_name, account_balance)
VALUES ('test_user', 'test@example.com', 'Тестовый Пользователь',
100.00);

-- Тест UPDATE
UPDATE users
SET account_balance = account_balance + 50
WHERE username = 'test_user';

-- Тест DELETE
DELETE FROM users WHERE username = 'test_user';

-- 4. Проверяем результат
SELECT * FROM audit_log ORDER BY change_time DESC;
```

Задание 2: Триггер для валидации данных

Задача: Создать триггер, который проверяет, что пользователю не меньше 18 лет при добавлении или изменении.

Пример выполнения:

```sql
-- 1. Создаем триггерную функцию
CREATE OR REPLACE FUNCTION validate_user_age()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.date_of_birth IS NOT NULL THEN
        IF AGE(NEW.date_of_birth) < INTERVAL '18 years' THEN
            RAISE EXCEPTION 'Пользователь должен быть старше 18 лет.
Текущий возраст: %',
                            AGE(NEW.date_of_birth);
        END IF;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- 2. Создаем триггер
CREATE TRIGGER validate_user_age_trigger
BEFORE INSERT OR UPDATE ON users
```

```
FOR EACH ROW
EXECUTE FUNCTION validate_user_age();

-- 3. Тестируем триггер
-- Попытка добавить несовершеннолетнего пользователя (ожидается ошибка)
INSERT INTO users (username, email, full_name, date_of_birth)
VALUES ('young_user', 'young@example.com', 'Молодой Пользователь',
'2010-01-01');

-- Попытка обновить дату рождения на недопустимую (ожидается ошибка)
UPDATE users
SET date_of_birth = '2015-05-05'
WHERE username = 'ivanov';
```

Задание 3: Триггер для ведения истории изменений

Задача: Создать триггер, который отслеживает изменения в полях `email` и `user_level` и записывает их в таблицу `user_history`.

Пример выполнения:

```
-- 1. Создаем триггерную функцию
CREATE OR REPLACE FUNCTION track_user_changes()
RETURNS TRIGGER AS $$
BEGIN
    -- Если изменился email
    IF OLD.email IS DISTINCT FROM NEW.email THEN
        INSERT INTO user_history (user_id, field_name, old_value,
new_value)
        VALUES (NEW.user_id, 'email', OLD.email, NEW.email);
    END IF;

    -- Если изменился user_level
    IF OLD.user_level IS DISTINCT FROM NEW.user_level THEN
        INSERT INTO user_history (user_id, field_name, old_value,
new_value)
        VALUES (NEW.user_id, 'user_level', OLD.user_level,
NEW.user_level);
    END IF;

    RETURN NEW;
END;
```

```
$$ LANGUAGE plpgsql;

-- 2. Создаем триггер
CREATE TRIGGER track_user_changes_trigger
AFTER UPDATE ON users
FOR EACH ROW
WHEN (OLD.email IS DISTINCT FROM NEW.email OR OLD.user_level IS DISTINCT
FROM NEW.user_level)
EXECUTE FUNCTION track_user_changes();

-- 3. Тестируем триггер
UPDATE users
SET email = 'new_email@example.com',
    user_level = 'premium'
WHERE username = 'ivanov';

-- 4. Проверяем результат
SELECT * FROM user_history ORDER BY change_time DESC;
```

Задания для самостоятельного выполнения

Задание 4: Триггер для автоматической блокировки пользователя

Задача: Создать триггер, который автоматически блокирует пользователя (устанавливает `is_active = FALSE`) при достижении 5 неудачных попыток входа (`login_attempts >= 5`). При блокировке необходимо записать информацию в таблицу `blocked_users`.

Требования:

1. Триггер должен срабатывать при `UPDATE` операциях
2. Проверять, достигло ли поле `login_attempts` значения 5
3. Если да и пользователь еще активен, то:
   - Установить `is_active = FALSE`
   - Добавить запись в таблицу `blocked_users` с причиной блокировки
4. Выводить уведомление о блокировке

Подсказки:

- Используйте триггер `BEFORE UPDATE`

- Проверяйте условие `NEW.login_attempts >= 5 AND NEW.is_active = TRUE`
- Для добавления записи используйте `INSERT INTO blocked_users`
- Для вывода сообщения используйте `RAISE NOTICE`

Задание 5: Триггер для обновления времени последнего входа

Задача: Создать триггер, который автоматически обновляет поле `last_login` при успешном входе пользователя. Успешным входом считается сброс счетчика неудачных попыток (`login_attempts = 0`).

Требования:

1. Триггер должен срабатывать при `UPDATE` операциях
2. Если пользователь активен и счетчик попыток сброшен (стал 0), то:
   - Установить `last_login = CURRENT_TIMESTAMP`
   - Вывести уведомление о обновлении времени входа
3. Триггер не должен срабатывать при других изменениях

Подсказки:

- Используйте триггер `BEFORE UPDATE`
- Сравнивайте `OLD.login_attempts` и `NEW.login_attempts`
- Условие: `NEW.is_active = TRUE AND NEW.login_attempts = 0 AND OLD.login_attempts > 0`
- Используйте `NEW.last_login := CURRENT_TIMESTAMP` для установки времени

Задание 6: Триггер для предотвращения удаления активных пользователей

Задача: Создать триггер, который предотвращает удаление активных пользователей. При попытке удалить активного пользователя должна возникать ошибка с сообщением.

Требования:

1. Триггер должен срабатывать при `DELETE` операциях
2. Если удаляемый пользователь активен (`is_active = TRUE`), то:
   - Отменить операцию удаления
   - Вызвать исключение с сообщением о необходимости деактивации
3. Если пользователь неактивен, разрешить удаление

Подсказки:

- Используйте триггер `BEFORE DELETE`
- Проверяйте `OLD.is_active`
- Используйте `RAISE EXCEPTION` для отмены операции
- Возвращайте `NULL` или `OLD` в зависимости от условия

Тестовые сценарии для проверки

После создания триггеров из заданий 4-6, проверьте их работу:

Для задания 4:

```
-- Имитируем 5 неудачных попыток входа
UPDATE users
SET login_attempts = 5
WHERE username = 'petrova';

-- Проверяем результат
SELECT username, is_active, login_attempts FROM users WHERE username =
'petrova';
SELECT * FROM blocked_users;
```

Для задания 5:

```
-- Имитируем успешный вход (сброс счетчика после блокировки)
UPDATE users
SET login_attempts = 0,
    is_active = TRUE
WHERE username = 'petrova';

-- Проверяем обновление last_login
SELECT username, last_login, login_attempts FROM users WHERE username =
'petrova';
```

Для задания 6:

```
-- Пытаемся удалить активного пользователя (должна быть ошибка)
DELETE FROM users WHERE username = 'sidorov';

-- Деактивируем пользователя
```

```sql
UPDATE users SET is_active = FALSE WHERE username = 'sidorov';

-- Пытаемся удалить снова (должно получиться)
DELETE FROM users WHERE username = 'sidorov';
```

Контрольные вопросы

1. В чем разница между триггерами `BEFORE` и `AFTER`?
2. Какие операции (`INSERT`, `UPDATE`, `DELETE`) можно использовать в триггерах?
3. Как получить значение поля до изменения в триггерной функции?
4. Что такое `TG_OP` и для чего она используется?
5. Как отменить операцию в триггере `BEFORE`?
6. Можно ли создать несколько триггеров для одной таблицы и одного события?
7. Как удалить или временно отключить триггер?

Требования к отчету

1. **SQL-скрипты** всех созданных триггеров (6 заданий)
2. **Результаты тестирования** каждого триггера
3. **Ответы на контрольные вопросы**
4. **Описание проблем**, возникших при выполнении