ГУАП

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Старший преподаватель |  |  |  | Н.В Путилова |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №8  Обеспечение активной целостности данных базы данных |
| **по дисциплине: Проектирование баз данных** |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

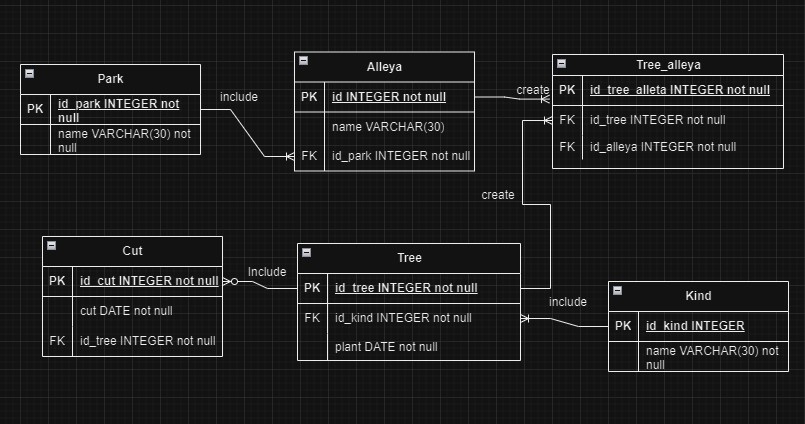
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. | 4134к |  |  |  | Столяров Н.С. |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург

2023

# Задание

Реализовать для своей базы данных триггеры для всех событий (insert,delete, update) до и после.(6 триггеров) Часть из которых будет обеспечивать ссылочную целостность, остальные могут иметь другое назначение из других предложенных , но не менее 2 различных  
(- Вычисление/поддержание в актуальном состоянии вычислимых (производных) атрибутов (полей);  
- логирование (запись) изменений;  
- обеспечения безопасности данных;  
- логическое (мягкое) удаление данных  
- проверка корректности проводимых действий.).  
Вычислимые поля можно добавить при необходимости.



## Before insert

----лимит даты tree before insert

    CREATE OR REPLACE FUNCTION check\_plant\_date()

    RETURNS TRIGGER AS $$

    BEGIN

        IF NEW.plant > CURRENT\_DATE THEN

            RAISE EXCEPTION 'Дата посадки не может быть в будущем';

        END IF;

        RETURN NEW;

    END;

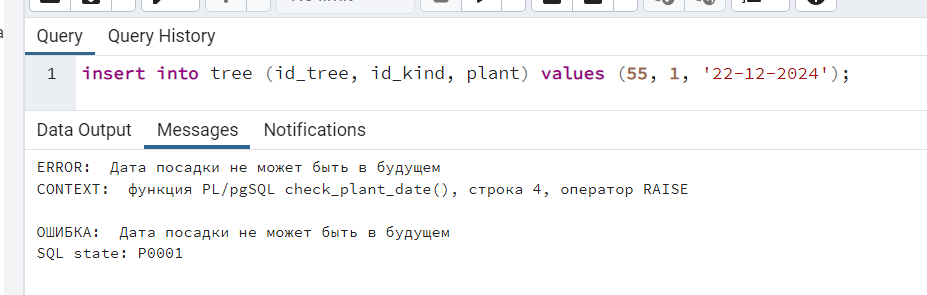
    $$ LANGUAGE plpgsql;

    CREATE TRIGGER before\_insert\_check\_plant\_date

    BEFORE INSERT ON TREE

    FOR EACH ROW

    EXECUTE FUNCTION check\_plant\_date();



## After delete

--удаление парков если в нем нет аллей

CREATE OR REPLACE FUNCTION after\_delete\_alley\_delete\_park()

RETURNS TRIGGER AS $$

BEGIN

    -- Если в парке больше нет аллей, удаляем парк

    IF NOT EXISTS (SELECT 1 FROM ALLEYA a WHERE a.ID\_PARK = OLD.ID\_PARK) THEN

        DELETE FROM PARK WHERE ID\_PARK = OLD.ID\_PARK;

    END IF;

    RETURN NULL;

END;

$$ LANGUAGE plpgsql;

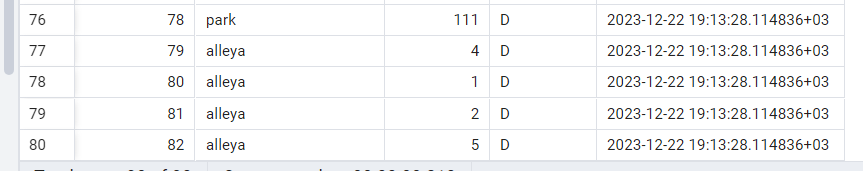
CREATE TRIGGER after\_delete\_alley\_delete\_park

AFTER DELETE ON ALLEYA

FOR EACH ROW

EXECUTE FUNCTION after\_delete\_alley\_delete\_park();

Для того, чтобы показать, как парк удаляется приведу логи из change\_log



## Before delete

--не даст удалить информацию из cut, если дерево еще существует

CREATE OR REPLACE FUNCTION before\_delete\_cut\_check\_tree\_existence()

RETURNS TRIGGER AS $$

BEGIN

    -- Проверяем существование дерева

    IF EXISTS (SELECT 1 FROM TREE WHERE ID\_TREE = OLD.ID\_TREE) THEN

        RAISE EXCEPTION 'Нельзя удалить информацию об обрезке, так как дерево с ID % еще существует.', OLD.ID\_TREE;

    END IF;

    RETURN NULL;

END;

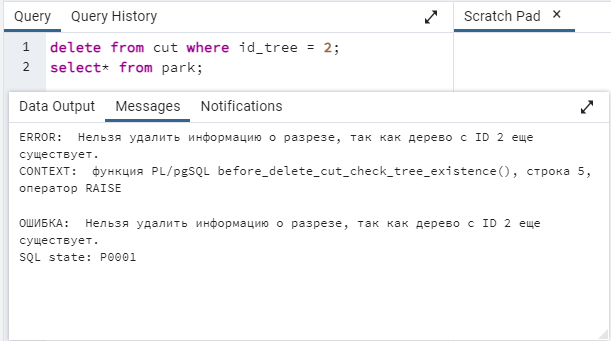
$$ LANGUAGE plpgsql;

CREATE TRIGGER before\_delete\_cut\_check\_tree\_existence

BEFORE DELETE ON CUT

FOR EACH ROW

EXECUTE FUNCTION before\_delete\_cut\_check\_tree\_existence();



## Before update

-- существует ли новый парк, и если нет, отменит операцию обновления.

CREATE OR REPLACE FUNCTION before\_update\_alleya\_check\_park\_existence()

RETURNS TRIGGER AS $$

BEGIN

    -- Проверяем существование нового парка

    IF NOT EXISTS (SELECT 1 FROM PARK WHERE ID\_PARK = NEW.ID\_PARK) THEN

        RAISE EXCEPTION 'Нельзя обновить аллею. Парк с ID % не существует.', NEW.ID\_PARK;

    END IF;

    RETURN NEW;

END;

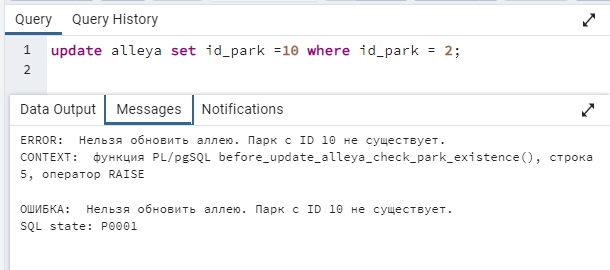
$$ LANGUAGE plpgsql;

CREATE TRIGGER before\_update\_alleya\_check\_park\_existence

BEFORE UPDATE ON ALLEYA

FOR EACH ROW

EXECUTE FUNCTION before\_update\_alleya\_check\_park\_existence();



## After update

 --существует ли дерево с новым  id\_tree

CREATE OR REPLACE FUNCTION after\_update\_cut\_check\_tree\_existence()

RETURNS TRIGGER AS $$

BEGIN

    -- Проверяем наличие дерева с новым ID\_TREE

    IF NOT EXISTS (SELECT 1 FROM TREE WHERE ID\_TREE = NEW.ID\_TREE) THEN

        RAISE EXCEPTION 'Дерева с ID\_TREE = % не существует', NEW.ID\_TREE;

    ELSE

        RAISE NOTICE 'Данные об обрезке изменены для дерева с ID\_TREE = %', NEW.ID\_TREE;

    END IF;

    RETURN NEW;

END;

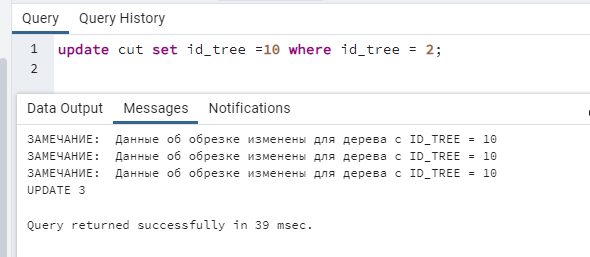
$$ LANGUAGE plpgsql;

CREATE TRIGGER after\_update\_cut\_check\_tree\_existence

AFTER UPDATE ON CUT

FOR EACH ROW

EXECUTE FUNCTION after\_update\_cut\_check\_tree\_existence();



## After insert и логирование

--логирование

CREATE TABLE CHANGE\_LOG (

    LOG\_ID SERIAL PRIMARY KEY,

    TABLE\_NAME VARCHAR(50) NOT NULL,

    RECORD\_ID INTEGER NOT NULL,

    OPERATION CHAR(1) NOT NULL, -- 'I' (insert), 'U' (update), 'D' (delete)

    LOGGED\_AT TIMESTAMPTZ DEFAULT CURRENT\_TIMESTAMP

);

CREATE OR REPLACE FUNCTION log\_change\_park()

RETURNS TRIGGER AS $$

BEGIN

    IF TG\_OP = 'INSERT' THEN

        INSERT INTO CHANGE\_LOG (TABLE\_NAME, RECORD\_ID, OPERATION)

        VALUES (TG\_TABLE\_NAME, NEW.ID\_PARK, 'I');

    ELSIF TG\_OP = 'UPDATE' THEN

        INSERT INTO CHANGE\_LOG (TABLE\_NAME, RECORD\_ID, OPERATION)

        VALUES (TG\_TABLE\_NAME, NEW.ID\_PARK, 'U');

    ELSIF TG\_OP = 'DELETE' THEN

        INSERT INTO CHANGE\_LOG (TABLE\_NAME, RECORD\_ID, OPERATION)

        VALUES (TG\_TABLE\_NAME, OLD.ID\_PARK, 'D');

    END IF;

    RETURN NULL;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER log\_park\_changes

AFTER INSERT OR UPDATE OR DELETE ON PARK

FOR EACH ROW

EXECUTE FUNCTION log\_change\_park();

CREATE OR REPLACE FUNCTION log\_change\_alleya()

RETURNS TRIGGER AS $$

BEGIN

    IF TG\_OP = 'INSERT' THEN

        INSERT INTO CHANGE\_LOG (TABLE\_NAME, RECORD\_ID, OPERATION)

        VALUES (TG\_TABLE\_NAME, NEW.ID\_ALLEYA, 'I');

    ELSIF TG\_OP = 'UPDATE' THEN

        INSERT INTO CHANGE\_LOG (TABLE\_NAME, RECORD\_ID, OPERATION)

        VALUES (TG\_TABLE\_NAME, NEW.ID\_ALLEYA, 'U');

    ELSIF TG\_OP = 'DELETE' THEN

        INSERT INTO CHANGE\_LOG (TABLE\_NAME, RECORD\_ID, OPERATION)

        VALUES (TG\_TABLE\_NAME, OLD.ID\_ALLEYA, 'D');

    END IF;

    RETURN NULL;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER log\_alleya\_changes

AFTER INSERT OR UPDATE OR DELETE ON ALLEYA

FOR EACH ROW

EXECUTE FUNCTION log\_change\_alleya();

CREATE OR REPLACE FUNCTION log\_change\_tree()

RETURNS TRIGGER AS $$

BEGIN

    IF TG\_OP = 'INSERT' THEN

        INSERT INTO CHANGE\_LOG (TABLE\_NAME, RECORD\_ID, OPERATION)

        VALUES (TG\_TABLE\_NAME, NEW.ID\_TREE, 'I');

    ELSIF TG\_OP = 'UPDATE' THEN

        INSERT INTO CHANGE\_LOG (TABLE\_NAME, RECORD\_ID, OPERATION)

        VALUES (TG\_TABLE\_NAME, NEW.ID\_TREE, 'U');

    ELSIF TG\_OP = 'DELETE' THEN

        INSERT INTO CHANGE\_LOG (TABLE\_NAME, RECORD\_ID, OPERATION)

        VALUES (TG\_TABLE\_NAME, OLD.ID\_TREE, 'D');

    END IF;

    RETURN NULL;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER log\_tree\_changes

AFTER INSERT OR UPDATE OR DELETE ON TREE

FOR EACH ROW

EXECUTE FUNCTION log\_change\_tree();

CREATE OR REPLACE FUNCTION log\_change\_kind()

RETURNS TRIGGER AS $$

BEGIN

    IF TG\_OP = 'INSERT' THEN

        INSERT INTO CHANGE\_LOG (TABLE\_NAME, RECORD\_ID, OPERATION)

        VALUES (TG\_TABLE\_NAME, NEW.ID\_KIND, 'I');

    ELSIF TG\_OP = 'UPDATE' THEN

        INSERT INTO CHANGE\_LOG (TABLE\_NAME, RECORD\_ID, OPERATION)

        VALUES (TG\_TABLE\_NAME, NEW.ID\_KIND, 'U');

    ELSIF TG\_OP = 'DELETE' THEN

        INSERT INTO CHANGE\_LOG (TABLE\_NAME, RECORD\_ID, OPERATION)

        VALUES (TG\_TABLE\_NAME, OLD.ID\_KIND, 'D');

    END IF;

    RETURN NULL;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER log\_kind\_changes

AFTER INSERT OR UPDATE OR DELETE ON KIND

FOR EACH ROW

EXECUTE FUNCTION log\_change\_kind();

CREATE OR REPLACE FUNCTION log\_change\_cut()

RETURNS TRIGGER AS $$

BEGIN

    IF TG\_OP = 'INSERT' THEN

        INSERT INTO CHANGE\_LOG (TABLE\_NAME, RECORD\_ID, OPERATION)

        VALUES (TG\_TABLE\_NAME, NEW.ID\_CUT, 'I');

    ELSIF TG\_OP = 'UPDATE' THEN

        INSERT INTO CHANGE\_LOG (TABLE\_NAME, RECORD\_ID, OPERATION)

        VALUES (TG\_TABLE\_NAME, NEW.ID\_CUT, 'U');

    ELSIF TG\_OP = 'DELETE' THEN

        INSERT INTO CHANGE\_LOG (TABLE\_NAME, RECORD\_ID, OPERATION)

        VALUES (TG\_TABLE\_NAME, OLD.ID\_CUT, 'D');

    END IF;

    RETURN NULL;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER log\_cut\_changes

AFTER INSERT OR UPDATE OR DELETE ON CUT

FOR EACH ROW

EXECUTE FUNCTION log\_change\_cut();

