

Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

институт

Программная инженерия

кафедра

ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ

Программирование на стороне сервера в среде СУБД PostgreSQL

тема

Преподаватель

подпись, дата

А. Д. Вожжов

инициалы, фамилия

Студент КИ23-17/16, 032320521

номер группы, зачётной книжки

подпись, дата

А. С. Лысаковский

инициалы, фамилия

Красноярск 2025

1 ВВЕДЕНИЕ

1.1 Цель работы

Изучить теоретический материал по теме «Программирование на стороне сервера в среде СУБД PostgreSQL». Выполнить задания.

1.2 Задачи

В рамках данной практической работы необходимо выполнить следующие задачи:

- 1 изучить теоретический материал по предложенной теме;
- 2 выполнить задание;
- 3 предоставить отчёт преподавателю.

1.3 Задание

Задание данной практической работы состоит из следующих частей:

- 1 Выполнить задания из главы 4 из книги на е-курсах.

2 ХОД РАБОТЫ

2.1 Задание 1

Вызов функции можно использовать в подзапросе в предложении FROM. Напишите такой запрос с подзапросом на примере функции generate_students_data().

На рисунке 1 показан результат выполнения задания.

```
ais=# CREATE OR REPLACE FUNCTION generate_students_data()
ais-# RETURNS TABLE (mark_book numeric(5), name text, psp_ser numeric(4), psp_num numeric(6))
ais-# AS $$
ais$# BEGIN
ais$#     -- Создадим последовательности для полей mark_book, psp_ser, psp_num.
ais$#     -- Значения поля «Номер зачетной книжки» должны быть пятизначными.
ais$#     CREATE TEMP SEQUENCE New_mark_book START WITH 10000;
ais$#     -- Значения поля «Серия паспорта» должны быть четырехзначными.
ais$#     CREATE TEMP SEQUENCE New_psp_ser START WITH 1000;
ais$#     -- Значения поля «Номер паспорта» должны быть шестизначными.
ais$#     CREATE TEMP SEQUENCE New_psp_num START WITH 100000;
ais$#     RETURN QUERY
ais$#     SELECT NEXTVAL('New_mark_book')::numeric(5),
ais$#            lname_base || lname_suffix || ' ' || fname || ' ' || pname,
ais$#            NEXTVAL('New_psp_ser')::numeric(4),
ais$#            NEXTVAL('New_psp_num')::numeric(6)
ais$#     FROM (VALUES ('Логин'), ('Перл'), ('Програм'), ('Лерам')) AS lname_base (lname_base),
ais$#            (VALUES ('ов'), ('ова')) AS lname_suffix (lname_suffix),
ais$#            (VALUES ('Иван'), ('Петр'), ('Антон')) AS fname (fname),
ais$#            (VALUES ('Иванович'), ('Петрович'), ('Антонович')) AS pname (pname);
ais$# END;
ais$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
ais=#
ais=# SELECT * FROM generate_students_data() LIMIT 5;
 mark_book |          name          | psp_ser | psp_num
-----+-----+-----+-----
10000 | Логинов Иван Иванович |    1000 | 100000
10001 | Перлов Иван Иванович  |    1001 | 100001
10002 | Програмов Иван Иванович |    1002 | 100002
10003 | Лерамов Иван Иванович  |    1003 | 100003
10004 | Логинов Иван Петрович |    1004 | 100004
(5 строк)
```

Рисунок 1 – Создание функции, применение её в запросе

2.2 Задание 2

Какая команда используется для удаления функции? Удалите какую-нибудь вашу функцию.

На рисунке 2 показан результат выполнения задания.

```
ais=# DROP FUNCTION IF EXISTS generate_students_data;
DROP FUNCTION
```

Рисунок 2 – Удаление функции

2.3 Задание 3

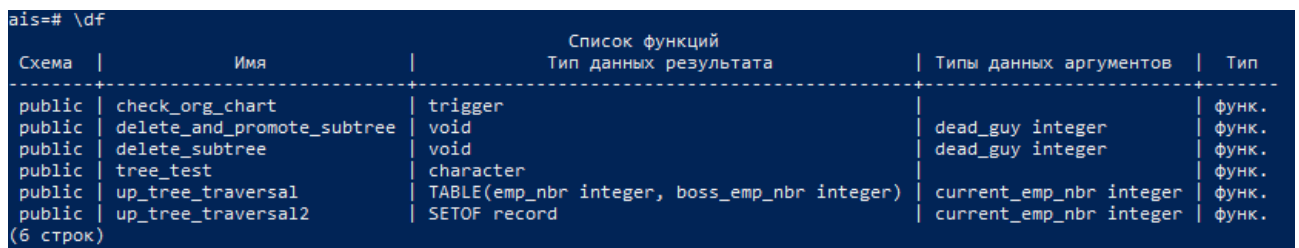
Для чего нужны модификаторы (ключевые слова) IN и OUT перед именами параметров функций?

Ответ. Ключевые слова IN и OUT служат для указания входных и выходных параметров для функции.

2.4 Задание 4

С помощью интерактивного терминала `psql` посмотрите список всех функций, созданных в вашей базе данных.

На рисунке 3 показан результат выполнения задания.



```
ais=# \df
```

Схема	Имя	Список функций Тип данных результата	Типы данных аргументов	Тип
public	check_org_chart	trigger		функ.
public	delete_and_promote_subtree	void	dead_guy integer	функ.
public	delete_subtree	void	dead_guy integer	функ.
public	tree_test	character		функ.
public	up_tree_traversal	TABLE(emp_nbr integer, boss_emp_nbr integer)	current_emp_nbr integer	функ.
public	up_tree_traversal2	SETOF record	current_emp_nbr integer	функ.

(6 строк)

Рисунок 3 – Список всех функций

2.5 Задание 5

Каким образом можно сделать так, чтобы функция возвратила табличное значение?

Чтобы функция возвратила табличное значение, необходимо в RETURNS указать TABLE (...), где в скобках перечислить возвращаемые столбцы.

2.6 Задание 6

Модифицируйте функцию `count_letters()`, подсчитывающую количество фамилий в таблице `students` («Студенты»), начинающихся на каждую букву.

Сделайте так, чтобы в случае отсутствия в таблице фамилий, начинающихся с каких-то букв, в выводе функции эти буквы были представлены нулевыми (пустыми) значениями.

На рисунках 4, 5 показан результат выполнения задания.

```

ais=# CREATE OR REPLACE FUNCTION count_letters()
ais=# RETURNS TABLE (letter char(1), num bigint) AS $$
ais=# BEGIN
ais=#     -- Возвращаем результат с использованием RIGHT JOIN для включения всех букв
ais=#     RETURN QUERY
ais=#     WITH alphabet AS (
ais=#         -- Создаём временную таблицу со всеми буквами русского алфавита
ais=#         SELECT unnest(ARRAY[
ais=#             'А', 'Б', 'В', 'Г', 'Д', 'Е', 'Ё', 'Ж', 'З', 'И', 'Й',
ais=#             'К', 'Л', 'М', 'Н', 'О', 'П', 'Р', 'С', 'Т', 'У', 'Ф',
ais=#             'Х', 'Ц', 'Ч', 'Ш', 'Щ', 'Ъ', 'Ы', 'Ь', 'Э', 'Ю', 'Я'
ais=#         ]::char[]) AS letter
ais=#     ),
ais=#     letter_counts AS (
ais=#         -- Подсчитываем количество фамилий по первой букве
ais=#         SELECT substr(name, 1, 1) AS letter, count(*) AS cnt
ais=#         FROM students
ais=#         GROUP BY substr(name, 1, 1)
ais=#     )
ais=#     -- Соединяем алфавит с подсчётами, заменяя NULL на 0
ais=#     SELECT a.letter, COALESCE(lc.cnt, 0) AS num
ais=#     FROM alphabet a
ais=#     LEFT JOIN letter_counts lc ON a.letter = lc.letter
ais=#     ORDER BY a.letter;
ais=# END;
ais=# $$ LANGUAGE plpgsql;
CREATE FUNCTION

```

Рисунок 4 – Функция

```

ais=# CREATE TABLE students (
ais(#      mark_book numeric(5),
ais(#      name text,
ais(#      psp_ser numeric(4),
ais(#      psp_num numeric(6)
ais(# );
CREATE TABLE
ais=# INSERT INTO students (mark_book, name, psp_ser, psp_num)
ais=# VALUES
ais-#      (10001, 'Иванов Иван Иванович', 0402, 123456),
ais-#      (10002, 'Петров Пётр Петрович', 0403, 123457),
ais-#      (10003, 'Сидоров Сидор Сидорович', 0404, 123458);
INSERT 0 3
ais=# SELECT * FROM count_letters();
 letter | num
-----+----
 А      |  0
 Б      |  0
 В      |  0
 Г      |  0
 Д      |  0
 Е      |  0
 Ё      |  0
 Ж      |  0
 З      |  0
 И      |  1
 Й      |  0
 К      |  0
 Л      |  0
 М      |  0
 Н      |  0
 О      |  0
 П      |  1
 Р      |  0
 С      |  1
 Т      |  0
 У      |  0
 Ф      |  0
 Х      |  0
 Ц      |  0
 Ч      |  0
 Ш      |  0
 Щ      |  0
 Ъ      |  0
 Ы      |  0
 Ь      |  0
 Э      |  0
 Ю      |  0
 Я      |  0
(33 строки)

```

Рисунок 5 – Результат

2.7 Задание 7

Что такое триггер?

Триггер – это набор действий, выполняемый в случае возникновения определённых событий, привязанный к таблицам или представлениям. Событиями могут быть выполняемые операции над таблицей: SELECT, INSERT, UPDATE, DELETE.

2.8 Задание 8

Какие особенности имеет триггерная функция?

Триггер-функция может выполняться до операции и после операции. В представлениях заменяет собой операцию по умолчанию. На уровне затрагивания триггер-функция может выполняться для каждой строки операции или единожды для всей операции в целом, даже если строк не было получено. Триггерная функция лишь вызывает другую функцию, которая выполняет всю работу. В триггерной функции доступны новые (NEW) и старые (OLD) представления строк для взаимодействия.

2.9 Задание 9

Чем триггеры уровня строки (row-level) отличаются от триггеров уровня команды (statement-level)?

Row-level говорит выполняться триггер-функции каждую строчку. Statement-level говорит выполняться триггер-функции единожды за операцию.

2.10 Задание 10

Напишите триггер уровня строки (row-level) для таблицы «Студенты» или таблицы «Успеваемость».

На рисунках с 57 по 60 показан результат работы.

```
ais=# CREATE TABLE students (  
ais(#   mark_book numeric(5) NOT NULL, -- Номер зачётной книжки  
ais(#   name text NOT NULL,           -- Ф.И.О.  
ais(#   psp_ser numeric(4),            -- Серия паспорта  
ais(#   psp_num numeric(6),            -- Номер паспорта  
ais(#   PRIMARY KEY (mark_book)  
ais(# );  
CREATE TABLE  
ais=#  
ais=# CREATE TABLE students_audit (  
ais(#   audit_id SERIAL PRIMARY KEY,  
ais(#   mark_book numeric(5),          -- Номер зачётной книжки  
ais(#   old_name text,                 -- Старое Ф.И.О.  
ais(#   new_name text,                 -- Новое Ф.И.О.  
ais(#   old_psp_ser numeric(4),        -- Старая серия паспорта  
ais(#   new_psp_ser numeric(4),        -- Новая серия паспорта  
ais(#   old_psp_num numeric(6),        -- Старый номер паспорта  
ais(#   new_psp_num numeric(6),        -- Новый номер паспорта  
ais(#   operation_type text,           -- Тип операции (INSERT/UPDATE)  
ais(#   change_timestamp timestamp     -- Время изменения  
ais(# );  
CREATE TABLE
```

Рисунок 6 – Создание таблиц

```

ais=# CREATE OR REPLACE FUNCTION check_and_log_students()
ais-# RETURNS trigger AS $$
ais$# BEGIN
ais$#     -- Проверка корректности серии паспорта (для INSERT и UPDATE)
ais$#     IF (NEW.psp_ser IS NOT NULL) THEN
ais$#         IF (NEW.psp_ser < 1000 OR NEW.psp_ser > 9999) THEN
ais$#             RAISE EXCEPTION 'Серия паспорта должна быть в диапазоне от 1000 до 9999, получено: %', NEW.psp_ser;
ais$#         END IF;
ais$#     END IF;
ais$#
ais$#     -- Логирование в зависимости от операции
ais$#     IF (TG_OP = 'INSERT') THEN
ais$#         INSERT INTO students_audit (
ais$#             mark_book, new_name, new_psp_ser, new_psp_num, operation_type, change_timestamp
ais$#         ) VALUES (
ais$#             NEW.mark_book, NEW.name, NEW.psp_ser, NEW.psp_num, 'INSERT', CURRENT_TIMESTAMP
ais$#         );
ais$#         RETURN NEW; -- Разрешаем вставку
ais$#
ais$#     ELSIF (TG_OP = 'UPDATE') THEN
ais$#         -- Логируем старые и новые значения
ais$#         INSERT INTO students_audit (
ais$#             mark_book, old_name, new_name, old_psp_ser, new_psp_ser,
ais$#             old_psp_num, new_psp_num, operation_type, change_timestamp
ais$#         ) VALUES (
ais$#             OLD.mark_book, OLD.name, NEW.name, OLD.psp_ser, NEW.psp_ser,
ais$#             OLD.psp_num, NEW.psp_num, 'UPDATE', CURRENT_TIMESTAMP
ais$#         );
ais$#         RETURN NEW; -- Разрешаем обновление
ais$#
ais$#     END IF;
ais$#
ais$#     RETURN NULL; -- На случай, если TG_OP не INSERT и не UPDATE (хотя здесь это не произойдёт)
ais$# END;
ais$# $$ LANGUAGE plpgsql;
CREATE FUNCTION
ais=#
ais=# CREATE TRIGGER students_check_and_log
ais-# BEFORE INSERT OR UPDATE ON students
ais-# FOR EACH ROW EXECUTE FUNCTION check_and_log_students();
CREATE TRIGGER

```

Рисунок 7 – Создание и привязка триггер-функции

```

ais=# INSERT INTO students (mark_book, name, psp_ser, psp_num)
ais-# VALUES (10001, 'Иванов Иван Иванович', 1234, 567890);
INSERT 0 1
ais=#
ais=# INSERT INTO students (mark_book, name, psp_ser, psp_num)
ais-# VALUES (10002, 'Петров Пётр Петрович', 999, 123456);
ОШИБКА: Серия паспорта должна быть в диапазоне от 1000 до 9999, получено: 999
КОНТЕКСТ: функция PL/pgSQL check_and_log_students(), строка 6, оператор RAISE
ais=#
ais=# UPDATE students
ais-# SET name = 'Иванов Иван Сергеевич', psp_ser = 4321
ais-# WHERE mark_book = 10001;
UPDATE 1
ais=#
ais=# SELECT * FROM students_audit;
 audit_id | mark_book | old_name | new_name | old_psp_ser | new_psp_ser | old_psp_num | new_psp_num | operation_type | change_timestamp
-----
1 | 10001 |  | Иванов Иван Иванович |  | 1234 |  | 567890 | INSERT | 2025-03-05 21:01:40.115166
2 | 10001 | Иванов Иван Иванович | Иванов Иван Сергеевич | 1234 | 4321 | 567890 | 567890 | UPDATE | 2025-03-05 21:01:58.898971
(2 строки)

```

Рисунок 8 – Тестирование функции

2.11 Задание 11

В базе данных ais создайте таблицы, функции и триггеры, необходимые для изучения метода хранения иерархий в реляционных базах данных.

Для этого можно поступить следующим образом:

Сначала создайте текстовый файл с именем, например, adj_list.sql, содержащий все команды и определения функций, приведенные в тексте пособия.

Затем выполните команду в среде операционной системы:

```
psql -d ais -f adj_list.sql
```

Возьмём структуру из главы 4. На рисунке 9 представлен прогресс работы.


```

PS C:\Users\Alex> psql -U postgres -d ais -f "C:\Users\Alex\YandexDisk\СМК\4 01-Г000\Г_0\09\adj_list.sql"
Пароль пользователя postgres:
psql:C:\Users\Alex\YandexDisk\ИКИТ\4 семестр\ТБД\ПР9\adj_list.sql:9: ЗАМЕЧАНИЕ:  таблица "personnel" не существует, пропускается
DROP TABLE
CREATE TABLE
INSERT 0 9
psql:C:\Users\Alex\YandexDisk\ИКИТ\4 семестр\ТБД\ПР9\adj_list.sql:31: ЗАМЕЧАНИЕ:  таблица "org_chart" не существует, пропускается
DROP TABLE
CREATE TABLE
INSERT 0 8
CREATE FUNCTION
psql:C:\Users\Alex\YandexDisk\ИКИТ\4 семестр\ТБД\ПР9\adj_list.sql:84: ЗАМЕЧАНИЕ:  триггер "check_org_chart" для отношения "org_chart" не существует, пропускается
DROP TRIGGER
CREATE TRIGGER
CREATE FUNCTION
CREATE FUNCTION
CREATE FUNCTION
CREATE FUNCTION
psql:C:\Users\Alex\YandexDisk\ИКИТ\4 семестр\ТБД\ПР9\adj_list.sql:181: ЗАМЕЧАНИЕ:  представление "personnel_org_chart" не существует, пропускается
DROP VIEW
CREATE VIEW
psql:C:\Users\Alex\YandexDisk\ИКИТ\4 семестр\ТБД\ПР9\adj_list.sql:189: ЗАМЕЧАНИЕ:  представление "create_paths" не существует, пропускается
DROP VIEW
CREATE VIEW
CREATE FUNCTION

```

Рисунок 9 – Прогресс работы

2.12 Задание 12

Сделайте выборки данных из таблиц «Персонал» и «Организационная структура», а также реконструируйте организационную структуру с помощью двух представлений (view).

Команды можно выполнять не только в среде интерактивного терминала psql, но также и из командной строки операционной системы:

Выполните эти команды в командной строке:

```
psql -d ais -c "SELECT * FROM Personnel"
```

```
psql -d ais -c "SELECT * FROM Org_chart"
```

```
psql -d ais -c "SELECT * FROM Personnel_org_chart"
```

```
psql -d ais -c "SELECT * FROM Create_paths"
```

Примечание:

Если не указан параметр -U, утилита psql подключается к базе данных от имени пользователя ОС. Возможно, потребуется использовать параметр -U, если в БД нет соответствующей учетной записи.

На рисунках 10, 11 представлен прогресс работы.

```
ais=# SELECT * FROM Personnel;
```

emp_nbr	emp_name	emp_addr	birth_date
0	вакансия		2014-05-19
1	Иван	ул. Любителей языка С	1962-12-01
2	Петр	ул. UNIX гуру	1965-10-21
3	Антон	ул. Ассемблерная	1964-04-17
4	Захар	ул. им. СУБД PostgreSQL	1963-09-27
5	Ирина	просп. Программистов	1968-05-12
6	Анна	пер. Перловый	1969-03-20
7	Андрей	пл. Баз данных	1945-11-07
8	Николай	наб. ОС Linux	1944-12-01
9	Мария	ул. SQL	1970-01-15
10	Олег	пр. Алгоритмов	1980-03-22
11	Елена	пер. Базовый	1985-07-10

(12 строк)

```
ais=# SELECT * FROM Org_chart;
```

job_title	emp_nbr	boss_emp_nbr	salary
Президент	1		1000.0000
Вице-президент 1	2	1	900.0000
Программист С	6	4	500.0000
Тестировщик	10	7	420.0000
Стажёр	11	10	300.0000
Программист Perl	7	6	450.0000
Оператор	8	6	400.0000
Архитектор	4	6	700.0000
Аналитик	9	6	550.0000

(9 строк)

Рисунок 10 – Запросы, часть 1

```
ais=# SELECT * FROM Personnel_org_chart;
```

emp_nbr	emp	boss_emp_nbr	boss
1	Иван		
2	Петр	1	Иван
6	Анна	4	Захар
10	Олег	7	Андрей
11	Елена	10	Олег
7	Андрей	6	Анна
8	Николай	6	Анна
4	Захар	6	Анна
9	Мария	6	Анна

(9 строк)

```
ais=# SELECT * FROM Create_paths;
```

level1	level2	level3	level4	level5
Иван	Петр			

(1 строка)

Рисунок 11 – Запросы, часть 2

2.13 Задание 13

Выполните проверку структуры дерева на отсутствие циклов командой:

```
SELECT * FROM tree_test();
```

При первоначальной проверке без изменений в таблице "Организационная структура" функция покажет корректную древовидную структуру.

Для тестирования:

Создайте короткий цикл в таблице

Создайте длинный цикл в таблице

После каждого изменения проверяйте структуру:

SELECT * FROM tree_test();

На рисунках с 12 по 14 показан прогресс работы.

```
ais=# SELECT * FROM org_chart;
```

job_title	emp_nbr	boss_emp_nbr	salary
Президент	1		1000.0000
Вице-президент 1	2	1	900.0000
Вице-президент 2	3	1	800.0000
Архитектор	4	3	700.0000
Ведущий программист	5	3	600.0000
Программист С	6	3	500.0000
Программист Perl	7	5	450.0000
Оператор	8	5	400.0000

(8 строк)

Рисунок 12 – Исходная таблица

```
ais=# UPDATE Org_chart
ais=# SET boss_emp_nbr = 5
ais=# WHERE emp_nbr = 4;
UPDATE 1
ais=#
ais=# UPDATE Org_chart
ais=# SET boss_emp_nbr = 4
ais=# WHERE emp_nbr = 5;
UPDATE 1
ais=#
ais=# SELECT tree_test();
tree_test
-----
Cycles
(1 строка)
```

```
ais=# SELECT * FROM org_chart;
```

job_title	emp_nbr	boss_emp_nbr	salary
Президент	1		1000.0000
Вице-президент 1	2	1	900.0000
Программист С	6	3	500.0000
Программист Perl	7	5	450.0000
Оператор	8	5	400.0000
Вице-президент 2	3	2	800.0000
Архитектор	4	5	700.0000
Ведущий программист	5	4	600.0000

(8 строк)

Рисунок 13 – Создание короткого цикла

```

ais=# UPDATE Org_chart
ais=# SET boss_emp_nbr = 5
ais=# WHERE emp_nbr = 4;
UPDATE 1
ais=# UPDATE Org_chart
ais=# SET boss_emp_nbr = 6
ais=# WHERE emp_nbr = 5;
UPDATE 1
ais=# UPDATE Org_chart
ais=# SET boss_emp_nbr = 4
ais=# WHERE emp_nbr = 6;
UPDATE 1
ais=# SELECT tree_test();
tree_test
-----
Cycles
(1 строка)

ais=# SELECT * FROM org_chart;
   job_title   | emp_nbr | boss_emp_nbr | salary
-----+-----+-----+-----
Президент      |      1 |              | 1000.0000
Вице-президент 1 |      2 |              | 900.0000
Программист Perl |      7 |              | 450.0000
Оператор        |      8 |              | 400.0000
Вице-президент 2 |      3 |              | 800.0000
Архитектор      |      4 |              | 700.0000
Ведущий программист |      5 |              | 600.0000
Программист С   |      6 |              | 500.0000
(8 строк)

```

Рисунок 14 – Создание длинного цикла

2.14 Задание 14

Выполните обход дерева организационной структуры снизу вверх, начиная с конкретного узла, можно с помощью функции `up_tree_traversal()` либо функции `up_tree_traversal2()`.

Сначала сделайте это с помощью первой из функций:
`SELECT * FROM up_tree_traversal(6);`

Параметром этих функций является код работника. Измените код работника и повторите команду.

Теперь воспользуйтесь второй функцией. Учтите, что она возвращает `SETOF RECORD`, поэтому команда будет более сложной:

`SELECT * FROM up_tree_traversal2(6) AS (emp int, boss int);`

Очевидно, что для использования числового кода работника нужно знать этот код. Удобнее иметь дело с именем работника. Поэтому можно в качестве параметра этих функций использовать подзапрос, возвращающий код работника в качестве своего результата. Не забудьте, что текст подзапроса заключается в скобки, поэтому появляются двойные скобки:

```
SELECT * FROM up_tree_traversal((SELECT ... FROM Personnel WHERE ...));
```

Завершите эту команду и выполните ее с различными именами работников. На рисунке 15 показан прогресс работы.

```
ais=# SELECT * FROM up_tree_traversal( ( SELECT emp_nbr FROM Personnel WHERE emp_name = 'Антон' ));
 emp_nbr | boss_emp_nbr 
-----+-----
      3 |           2
      2 |           1
      1 |
(3 строки)
```

Рисунок 15 – Обход дерева по имени сотрудника

2.15 Задание 15

Выполните операцию удаления поддерева с помощью функции `delete_subtree()`. Параметром функции является код работника.

```
SELECT * FROM delete_subtree(6);
```

Аналогично работе с функцией `up_tree_traversal()` используйте подзапрос для получения кода работника по его имени:

```
SELECT * FROM delete_subtree((SELECT emp_id FROM Personnel WHERE emp_name = 'Иванов'));
```

После удаления поддерева проверьте изменения в организационной структуре с помощью представлений:

```
SELECT * FROM Personnel_org_chart;
```

```
SELECT * FROM Create_paths;
```

На рисунках 16, 17 показан прогресс работы.

```
ais=# SELECT * FROM Personnel_org_chart;
emp_nbr | emp      | boss_emp_nbr | boss
-----+-----+-----+-----
      1 | Иван    |              |
      2 | Петр    |              |
      7 | Андрей  |              |
      8 | Николай |              |
      3 | Антон    |              |
      4 | Захар   |              |
      5 | Ирина   |              |
      6 | Анна    |              |
(8 строк)

ais=# SELECT * FROM Create_paths;
level1 | level2 | level3 | level4
-----+-----+-----+-----
Иван   | Петр   | Антон   |
(1 строка)

ais=# SELECT * FROM delete_subtree( (SELECT emp_nbr FROM Personnel WHERE name='Антон') );
ОШИБКА: столбец "name" не существует
СТРОКА 1: ...ete_subtree( (SELECT emp_nbr FROM Personnel WHERE name='Анто...
                                         ^
ais=# SELECT * FROM delete_subtree( (SELECT emp_nbr FROM Personnel WHERE emp_name='Антон') );
delete_subtree
-----
(1 строка)
```

Рисунок 16 – Удаление сотрудника по имени

```
ais=# SELECT * FROM Personnel_org_chart;
emp_nbr | emp      | boss_emp_nbr | boss
-----+-----+-----+-----
      1 | Иван    |              |
      2 | Петр    |              |
      7 | Андрей  |              |
      8 | Николай |              |
      4 | Захар   |              |
      5 | Ирина   |              |
      6 | Анна    |              |
(7 строк)

ais=# SELECT * FROM Create_paths;
level1 | level2 | level3 | level4
-----+-----+-----+-----
Иван   | Петр   |         |
(1 строка)
```

Рисунок 17 – Проверка

2.16 Задание 16

Если в таблице «Организационная структура» осталось мало данных, выполните следующие действия:

Дополните таблицу необходимыми данными

Выполните удаление элемента иерархии с продвижением дочерних элементов:

```
SELECT * FROM delete_and_promote_subtree(5);
```

Для удобства можно использовать подзапрос для получения кода работника по имени (аналогично функции `up_tree_traversal()`):

```
SELECT * FROM delete_and_promote_subtree(  
(SELECT emp_id FROM Personnel WHERE emp_name = 'Фамилия')  
);
```

После выполнения операции проверьте изменения в организационной структуре:

```
SELECT * FROM Personnel_org_chart;  
SELECT * FROM Create_paths;
```

На рисунках с 18 по 20 представлен прогресс работы.

```
ais=# INSERT INTO Personnel (emp_nbr, emp_name, emp_addr, birth_date) VALUES  
ais-#      (9, 'Мария', 'ул. SQL', '1970-01-15'),  
ais-#      (10, 'Олег', 'пр. Алгоритмов', '1980-03-22'),  
ais-#      (11, 'Елена', 'пер. Базовый', '1985-07-10');  
INSERT 0 3  
ais=#  
ais=# INSERT INTO Org_chart (job_title, emp_nbr, boss_emp_nbr, salary) VALUES  
ais-#      ('Аналитик', 9, 5, 550.00),      -- Подчиняется Ведущему программисту (5)  
ais-#      ('Тестировщик', 10, 7, 420.00),  -- Подчиняется Программисту Perl (7)  
ais-#      ('Стажёр', 11, 10, 300.00);      -- Подчиняется Тестировщику (10)  
INSERT 0 3  
ais=#  
ais=# SELECT * FROM org_chart;  
      job_title      | emp_nbr | boss_emp_nbr | salary  
-----+-----+-----+-----  
Президент           |      1 |             | 1000.0000  
Вице-президент 1    |      2 |             | 900.0000  
Программист Perl     |      7 |             | 450.0000  
Оператор             |      8 |             | 400.0000  
Архитектор           |      4 |             | 700.0000  
Ведущий программист |      5 |             | 600.0000  
Программист C        |      6 |             | 500.0000  
Аналитик             |      9 |             | 550.0000  
Тестировщик          |     10 |             | 420.0000  
Стажёр               |     11 |             | 300.0000  
(10 строк)
```

Рисунок 18 – Добавление сотрудников

```

ais=# SELECT * FROM delete_and_promote_subtree(5);
delete_and_promote_subtree
-----
(1 строка)

ais=# SELECT * FROM orgchart;
ОШИБКА: отношение "orgchart" не существует
СТРОКА 1: SELECT * FROM orgchart;
      ^
ais=# SELECT * FROM org_chart;

```

job_title	emp_nbr	boss_emp_nbr	salary
Президент	1		1000.0000
Вице-президент 1	2	1	900.0000
Программист С	6	4	500.0000
Тестировщик	10	7	420.0000
Стажёр	11	10	300.0000
Программист Perl	7	6	450.0000
Оператор	8	6	400.0000
Архитектор	4	6	700.0000
Аналитик	9	6	550.0000

```

(9 строк)

```

Рисунок 19 – Удаление по номеру

```

ais=# SELECT * FROM delete_and_promote_subtree( (SELECT emp_nbr FROM personnel WHERE emp_name = 'Ирина') );
delete_and_promote_subtree
-----
(1 строка)

ais=# SELECT * FROM org_chart;

```

job_title	emp_nbr	boss_emp_nbr	salary
Президент	1		1000.0000
Вице-президент 1	2	1	900.0000
Программист С	6	4	500.0000
Тестировщик	10	7	420.0000
Стажёр	11	10	300.0000
Программист Perl	7	6	450.0000
Оператор	8	6	400.0000
Архитектор	4	6	700.0000
Аналитик	9	6	550.0000

```

(9 строк)

```

Рисунок 20 – Удаление по имени

2.17 Задание 17

Представление `Create_paths` позволяет отобразить только четыре уровня иерархии. Модифицируйте его так, чтобы оно могло работать с пятью уровнями иерархии.

На рисунке 19 показан прогресс работы.


```

ais=# DROP VIEW IF EXISTS Create_paths;
DROP VIEW
ais=#
ais=# CREATE VIEW Create_paths (level1, level2, level3, level4, level5) AS
ais=# SELECT
ais=#     01.emp AS e1,
ais=#     02.emp AS e2,
ais=#     03.emp AS e3,
ais=#     04.emp AS e4,
ais=#     05.emp AS e5
ais=# FROM Personnel_org_chart AS 01
ais=# LEFT OUTER JOIN Personnel_org_chart AS 02 ON 01.emp = 02.boss
ais=# LEFT OUTER JOIN Personnel_org_chart AS 03 ON 02.emp = 03.boss
ais=# LEFT OUTER JOIN Personnel_org_chart AS 04 ON 03.emp = 04.boss
ais=# LEFT OUTER JOIN Personnel_org_chart AS 05 ON 04.emp = 05.boss
ais=# WHERE 01.emp = 'Иван';
CREATE VIEW
ais=# SELECT create_paths;
ОШИБКА: столбец "create_paths" не существует
СТРОКА 1: SELECT create_paths;
          ^
ais=# SELECT * FROM create_paths;
 level1 | level2 | level3 | level4 | level5
-----+-----+-----+-----+-----
 Иван  | Петр  |        |        |
(1 строка)

```

Рисунок 21 – Результат

2.18 Задание 18

Самостоятельно ознакомьтесь с курсорами (cursors) - средством работы с таблицами базы данных. Используйте техническую документацию PostgreSQL, главу «PL/pgSQL – SQL Procedural Language».

На рисунках 22, 23 показан прогресс работы.

```

ais=# CREATE OR REPLACE FUNCTION get_subordinates_by_name(start_emp_name text)
ais=# RETURNS TABLE (subordinate_name text, job_title text) AS $$
ais$# DECLARE
ais$#     cur_emp_nbr INTEGER;           -- Код текущего сотрудника
ais$#     sub_emp_nbr INTEGER;          -- Код подчинённого
ais$#     sub_name TEXT;                -- Имя подчинённого
ais$#     sub_job_title TEXT;           -- Должность подчинённого
ais$#     emp_cursor CURSOR FOR         -- Курсор для получения подчинённых
ais$#         SELECT O.emp_nbr, P.emp_name, O.job_title
ais$#         FROM Org_chart O
ais$#         JOIN Personnel P ON O.emp_nbr = P.emp_nbr
ais$#         WHERE O.boss_emp_nbr = cur_emp_nbr;
ais$# BEGIN
ais$#     -- Находим emp_nbr по имени сотрудника
ais$#     SELECT P.emp_nbr INTO cur_emp_nbr
ais$#     FROM Personnel P
ais$#     WHERE P.emp_name = start_emp_name;
ais$#
ais$#     IF NOT FOUND THEN
ais$#         RAISE NOTICE 'Сотрудник с именем % не найден', start_emp_name;
ais$#         RETURN;
ais$#     END IF;
ais$#
ais$#     -- Создаём временную таблицу для хранения уже обработанных сотрудников (избегаем циклов)
ais$#     CREATE TEMP TABLE processed_emps (emp_nbr INTEGER) ON COMMIT DROP;
ais$#
ais$#     -- Добавляем стартового сотрудника в обработанные
ais$#     INSERT INTO processed_emps VALUES (cur_emp_nbr);
ais$#
ais$#     -- Открываем курсор для первого уровня подчинённых
ais$#     OPEN emp_cursor;
ais$#
ais$#     LOOP
ais$#         -- Извлекаем следующего подчинённого
ais$#         FETCH emp_cursor INTO sub_emp_nbr, sub_name, sub_job_title;
ais$#         EXIT WHEN NOT FOUND; -- Выход, если больше нет строк
ais$#
ais$#         -- Проверяем, не обработан ли уже этот сотрудник
ais$#         IF NOT EXISTS (SELECT 1 FROM processed_emps WHERE emp_nbr = sub_emp_nbr) THEN
ais$#             -- Добавляем в результат
ais$#             subordinate_name := sub_name;
ais$#             job_title := sub_job_title;
ais$#             RETURN NEXT;
ais$#
ais$#             -- Добавляем в обработанные
ais$#             INSERT INTO processed_emps VALUES (sub_emp_nbr);
ais$#
ais$#             -- Рекурсивно открываем курсор для подчинённых текущего сотрудника
ais$#             cur_emp_nbr := sub_emp_nbr;
ais$#             CLOSE emp_cursor;
ais$#             OPEN emp_cursor;
ais$#         END IF;
ais$#     END LOOP;
ais$#
ais$#     -- Закрываем курсор
ais$#     CLOSE emp_cursor;
ais$#
ais$#     -- Удаляем временную таблицу (автоматически при ON COMMIT DROP, но для ясности)
ais$#     DROP TABLE IF EXISTS processed_emps;
ais$# END;
ais$# $$ LANGUAGE plpgsql;
CREATE FUNCTION

```

Рисунок 22 – Функция

```
ais=# SELECT * FROM get_subordinates_by_name('Захар');
subordinate_name | job_title
-----+-----
Анна             | Программист С
Андрей           | Программист Perl
Олег             | Тестировщик
Елена            | Стажёр
(4 строки)
```

Рисунок 23 – Результат

2.19 Задание 19

Самостоятельно ознакомьтесь с правилами (rules) - средством работы с таблицами базы данных. Используйте техническую документацию PostgreSQL, главу «The Rule System».

На рисунках с 24 по 26 показан прогресс работы.

```
ais=# CREATE TABLE students (
ais(#   mark_book numeric(5) NOT NULL PRIMARY KEY, -- Номер зачётной книжки
ais(#   name text NOT NULL,                        -- Ф.И.О.
ais(#   psp_ser numeric(4),                        -- Серия паспорта
ais(#   psp_num numeric(6)                        -- Номер паспорта
ais(# );
CREATE TABLE
ais=#
ais=# CREATE TABLE students_log (
ais(#   log_id SERIAL PRIMARY KEY,                -- Уникальный идентификатор записи
ais(#   operation_type VARCHAR(10),              -- Тип операции: INSERT, UPDATE, DELETE
ais(#   mark_book numeric(5),                    -- Номер зачётной книжки
ais(#   old_name text,                          -- Старое Ф.И.О. (NULL для INSERT)
ais(#   new_name text,                          -- Новое Ф.И.О. (NULL для DELETE)
ais(#   old_psp_ser numeric(4),                  -- Старая серия паспорта
ais(#   new_psp_ser numeric(4),                  -- Новая серия паспорта
ais(#   old_psp_num numeric(6),                  -- Старый номер паспорта
ais(#   new_psp_num numeric(6),                  -- Новый номер паспорта
ais(#   change_timestamp TIMESTAMP              -- Время изменения
ais(# );
CREATE TABLE
```

Рисунок 24 – Создание таблиц

```

ais=# CREATE RULE students_insert_log AS ON INSERT TO students
ais=# DO ALSO
ais=#     INSERT INTO students_log (
ais(#         operation_type, mark_book, new_name, new_psp_ser, new_psp_num, change_timestamp
ais(#     ) VALUES (
ais(#         'INSERT', NEW.mark_book, NEW.name, NEW.psp_ser, NEW.psp_num, CURRENT_TIMESTAMP
ais(#     );
CREATE RULE
ais=#
ais=# CREATE RULE students_update_log AS ON UPDATE TO students
ais=# DO ALSO
ais=#     INSERT INTO students_log (
ais(#         operation_type, mark_book, old_name, new_name, old_psp_ser, new_psp_ser,
ais(#         old_psp_num, new_psp_num, change_timestamp
ais(#     ) VALUES (
ais(#         'UPDATE', OLD.mark_book, OLD.name, NEW.name, OLD.psp_ser, NEW.psp_ser,
ais(#         OLD.psp_num, NEW.psp_num, CURRENT_TIMESTAMP
ais(#     );
CREATE RULE
ais=#
ais=# CREATE RULE students_delete_log AS ON DELETE TO students
ais=# DO ALSO
ais=#     INSERT INTO students_log (
ais(#         operation_type, mark_book, old_name, old_psp_ser, old_psp_num, change_timestamp
ais(#     ) VALUES (
ais(#         'DELETE', OLD.mark_book, OLD.name, OLD.psp_ser, OLD.psp_num, CURRENT_TIMESTAMP
ais(#     );
CREATE RULE

```

Рисунок 25 – Создание правил

```

ais=# INSERT INTO students (mark_book, name, psp_ser, psp_num)
ais=# VALUES (10001, 'Иванов Иван Иванович', 1234, 567890);
INSERT 0 1
ais=# SELECT * FROM students_log;

```

log_id	operation_type	mark_book	old_name	new_name	old_psp_ser	new_psp_ser	old_psp_num	new_psp_num	change_timestamp
1	INSERT	10001		Иванов Иван Иванович		1234		567890	2025-03-05 23:01:18.355835

(1 строка)

Рисунок 26 – Проверка

3 ЗАКЛЮЧЕНИЕ

По результатам работы был изучен теоретический материал по теме «Программирование на стороне сервера в среде СУБД PostgreSQL». Все поставленные цели и задачи были выполнены.