Федеральное государственное автономное

образовательное учреждение

Высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

|  |
| --- |
| Институт космических и информационных технологий |
| институт |
| Информатика |
| кафедра |

**ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ**

|  |
| --- |
| Типы данных СУБД PostgreSQL |
| тема |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Преподаватель | |  |  |  | Е. П. Моргунов |
|  | |  | подпись, дата |  | инициалы, фамилия |
| Студент | КИ21-17/2Б, 032156940 |  |  |  | Н. А. Самарин |
|  | номер группы, зачётной книжки |  | подпись, дата |  | инициалы, фамилия |

Красноярск 2023

**СОДЕРЖАНИЕ**

1 Задание............................................................................................................... 3

2 Выполнение заданий........................................................................................ 3

3 Вывод................................................................................................................. 16

**1 Задание**

- Изучить материал главы 4 учебного пособия;

- Выполнить задания, приведенные в разделе "Контрольные вопросы и  
задания";

- Подготовить отчет. Включить в него тексты заданий из пособия. Для  
каждого задания сделать снимки экрана с введенными командами языка SQL и  
результатами их выполнения в среде утилиты psql.

**2 Выполнение заданий**

1. Создайте таблицу, содержащую атрибут типа numeric(precision, scale).  
Пусть это будет таблица, содержащая результаты каких-то измерений. Команда  
может быть, например, такой: CREATE TABLE test\_numeric ( measurement  
numeric(5, 2), description text ); Попробуйте с помощью команды INSERT  
продемонстрировать округление вводимого числа до той точности, которая  
задана при создании таблицы. Подумайте, какая из следующих команд вызовет  
ошибку и почему? Проверьте свои предположения, выполнив эти команды.  
INSERT INTO test\_numeric VALUES ( 999.9999, 'Какое-то измерение ' );  
INSERT INTO test\_numeric VALUES ( 999.9009, 'Еще одно измерение' );  
INSERT INTO test\_numeric VALUES ( 999.1111, 'И еще измерение' ); INSERT  
INTO test\_numeric VALUES ( 998.9999, 'И еще одно' );

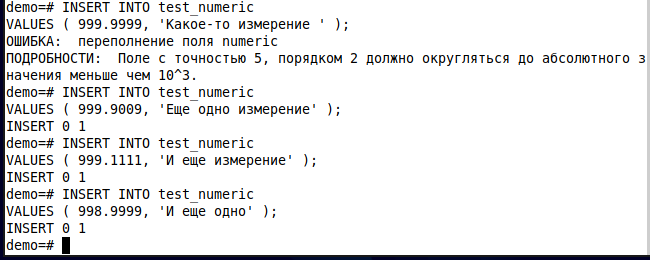


Рисунок 1 – Выполнение задания 1

3. Тип данных numeric поддерживает специальное значение NaN, которое  
означает «не число» (not a number). В документации утверждается, что  
значение NaN считается равным другому значению NaN, а также что значение  
NaN считается большим любого другого «нормального» значения, т. е. не-NaN.  
Проверьте эти ´ утверждения с помощью SQL-команды SELECT.

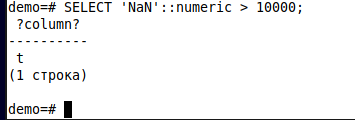


Рисунок 2 – Выполнение задания 3

5. Типы данных real и double precision поддерживают специальные  
значения Infinity (бесконечность) и −Infinity (отрицательная бесконечность).  
Проверьте с помощью SQL-команды SELECT ожидаемые свойства этих  
значений. Например, сравните Infinity с наибольшим значением, которое  
допускается для типа double precision (можно использовать сокращенное  
написание Inf):

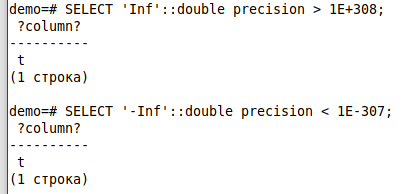


Рисунок 3 – Выполнение задания 5

7. Тип serial может применяться для столбцов, содержащих числовые  
значения, которые должны быть уникальными в пределах таблицы, например,  
идентификаторы каких-то объектов. В качестве иллюстрации применения типа  
serial предложим таблицу, содержащую наименования улиц и площадей:  
CREATE TABLE test\_serial ( id serial, name text ); Введите несколько строк.  
Обратите внимание, что значение для столбца id указывать не обязательно (и  
даже не нужно). Но поскольку мы задаем значения не для всех столбцов,  
имеющихся в таблице, мы должны указать в команде INSERT не только список  
значений, но и список столбцов. Конечно, в данном простом случае эти списки  
состоят лишь из одного элемента. INSERT INTO test\_serial ( name ) VALUES (  
'Вишневая' ); INSERT INTO test\_serial ( name ) VALUES ( 'Грушевая' ); INSERT  
INTO test\_serial ( name ) VALUES ( 'Зеленая' ); Сделайте выборку данных из  
таблицы, вы увидите, что значения столбца id имеют последовательные  
значения, начиная с 1. Давайте проведем эксперимент со столбцом id.  
Выполните команду INSERT, в которой укажите явное значение столбца id:  
INSERT INTO test\_serial ( id, name ) VALUES ( 10, 'Прохладная' ); А теперь

добавьте еще одну строку, но уже не указывая явно значение для столбца id  
(как мы поступали в предыдущих командах): INSERT INTO test\_serial ( name )  
VALUES ( 'Луговая' ); Вы увидите, что явное задание значения для столбца id  
не влияет на автоматическое генерирование значений этого столбца.

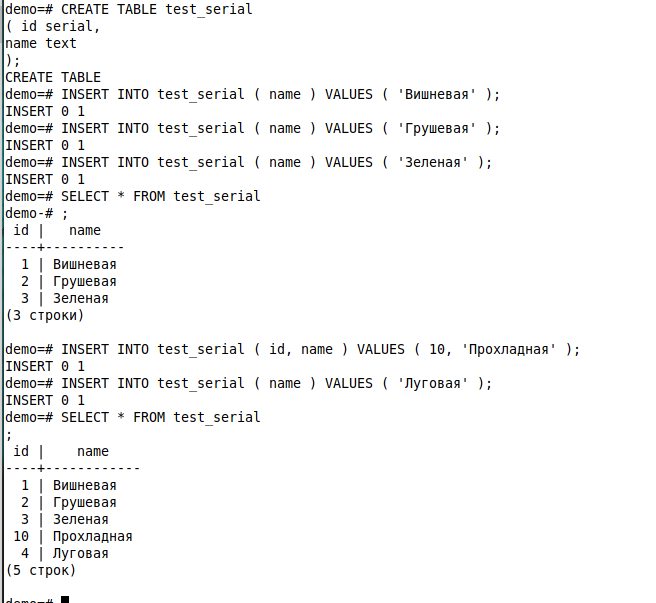


Рисунок 4 – Выполнение задания 7

9. Какой календарь используется в PostgreSQL для работы с датами:  
юлианский или григорианский?

Ответ: Григорианский

11. Типы timestamp, time и interval позволяют задать точность ввода и  
вывода значений. Точность предписывает количество десятичных цифр в поле  
секунд. Проиллюстрируем эту возможность на примере типа time, выполнив  
три запроса: в первом запросе вообще не используем параметр точности, во  
втором назначим его равным 0, в третьем запросе сделаем его равным 3.

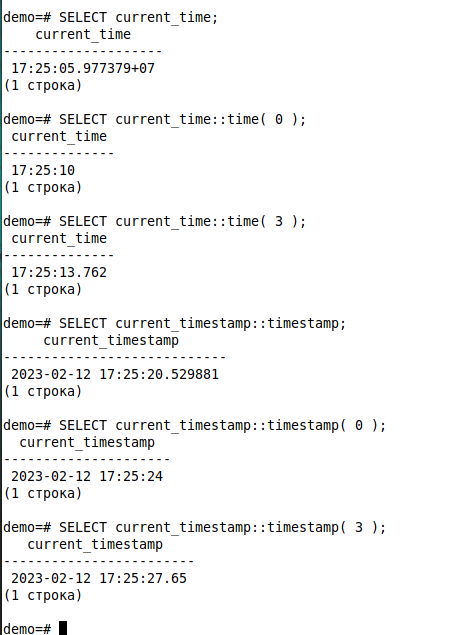


Рисунок 5 – Выполнение задания 11

13. Установить новое значение параметра datestyle можно с помощью  
создания переменной системного окружения PGDATESTYLE. Назначить эту  
переменную можно в конфигурационных файлах операционной системы. Но  
если нам нужно сделать это только на время текущего сеанса работы  
клиентской программы, например утилиты psql, то можно ввести значение этой  
переменной непосредственно в командной строке: PGDATESTYLE="Postgres"  
psql -d test -U имя-пользователя Проделайте эти действия, а затем уже из  
командной строки утилиты psql проверьте текущее значение параметра  
datestyle с помощью команды SHOW.

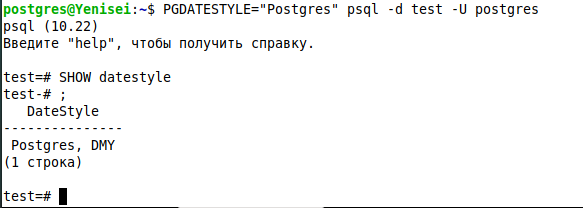


Рисунок 6 – Выполнение задания 13

15. В документации в разделе 9.8 «Функции форматирования данных»  
представлены описания множества полезных функций, позволяющих  
преобразовать в строку данные других типов, например, timestamp. Одна из  
таких функций — to\_char. Приведем несколько команд, иллюстрирующих  
использование этой функции. Ее первым параметром является форматируемое  
значение, а вторым — шаблон, описывающий формат, в котором это значение  
будет представлено при вводе или выводе. Сначала попробуйте разобраться, не  
обращаясь к документации, в том, что означает второй параметр этой функции  
в каждой из приведенных команд, а затем проверьте свои предположения по  
документации.

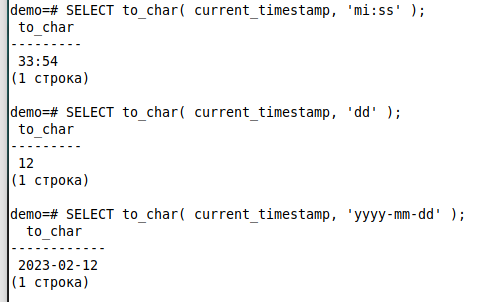


Рисунок 7 – Выполнение задания 15

mi - минуты, ss - секунды, dd - дни, yyyy - года, mm - месяцы

17. При выполнении приведения типа данных производится проверка  
значения на допустимость. Попробуйте ввести недопустимое значение  
времени, например, с нарушением формата. SELECT '21:15:16:22'::time;

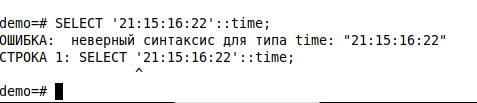


Рисунок 8 – Выполнение задания 17

19. С типами даты и времени можно выполнять различные  
арифметические операции. Как правило, их применение является интуитивно  
понятным. Выполните следующую команду и проанализируйте результат.  
SELECT ( '20:34:35'::time - '19:44:45'::time ); А теперь попробуйте  
предположить, какой результат будет получен, если в этой команде знак  
«минус» заменить на знак «плюс»? Проверьте ваши предположения с помощью  
утилиты psql.

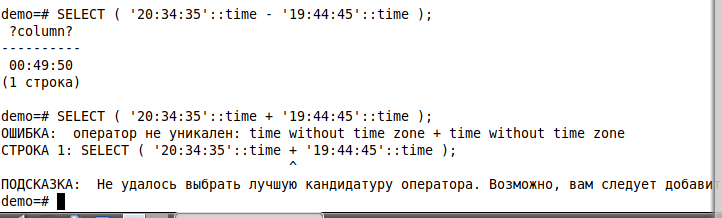


Рисунок 9 – Выполнение задания 19

21. Можно с высокой степенью уверенности предположить, что при  
прибавлении интервалов к датам и временным отметкам PostgreSQL учитывает  
тот факт, ´ что различные месяцы имеют различное число дней. Но как это  
реализуется на практике? Например, что получится при прибавлении интервала  
в 1 месяц к последнему дню января и к последнему дню февраля? Сначала  
сделайте обоснованные предположения о результатах следующих двух команд,  
а затем проверьте предположения на практике и проанализируйте полученные  
результаты: SELECT ( '2016-01-31'::date + '1 mon'::interval ) AS new\_date;  
SELECT ( '2016-02-29'::date + '1 mon'::interval ) AS new\_date;

Предположительно, при прибавлении месяца в результате получается  
дата следующего месяца с тем же значение дня что и в изначальной дате или с  
значение последнего дня нового месяца, если в нём меньше дней.

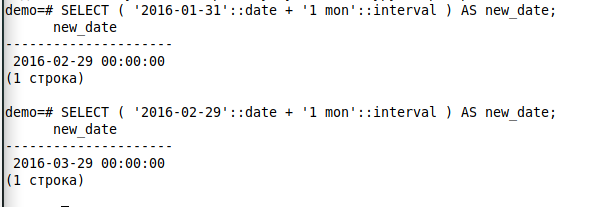


Рисунок 10 – Выполнение задания 21

23. Выполните следующие две команды и объясните различия в  
выведенных результатах: SELECT ( '2016-09-16'::date - '2015-09-01'::date );  
SELECT ( '2016-09-16'::timestamp - '2015-09-01'::timestamp );

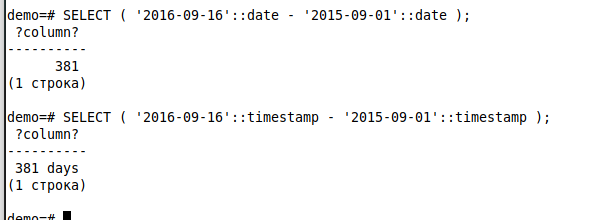


Рисунок 11 – Выполнение задания 23

В типе данных date дни являются "наименьшей величиной", в timestamp -  
нет.

25. Значения временных отметок можно усекать с той или иной  
точностью с помо- ´ щью функции date\_trunc. Например, с помощью  
следующей команды можно «отрезать» дробную часть секунды: SELECT (  
date\_trunc( 'sec', timestamp '1999-11-27 12:34:56.987654' ) ); Напомним, что в  
данной команде используется операция приведения типа. Выполните эту  
команду, последовательно указывая в качестве первого параметра значения  
microsecond, millisecond, second, minute, hour, day, week, month, year, decade,  
century, millennium (которые обозначают соответственно микросекунды,  
миллисекунды, секунды, минуты, часы, дни, недели, месяцы, годы,  
десятилетия, века и тысячелетия). Допустимы сокращения sec, min, mon, dec,  
cent, mil.

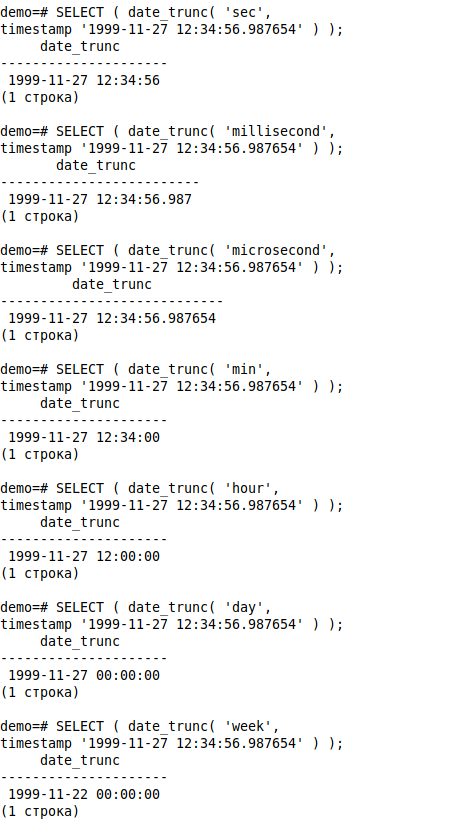


Рисунок 12 – Выполнение задания 25 часть 1

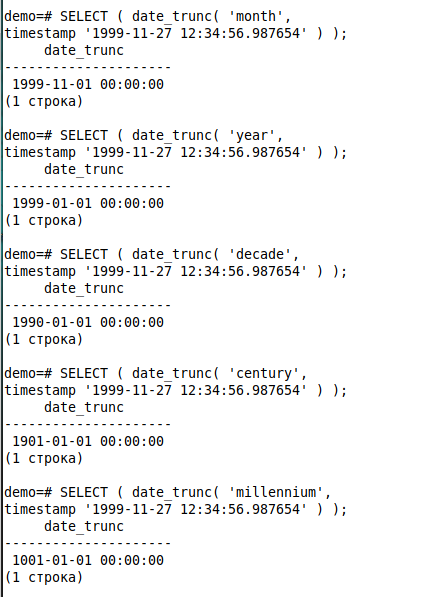


Рисунок 13 – Выполнение задания 25 часть 2

27. Весьма полезной является функция extract. С ее помощью можно  
извлечь значение отдельного поля из временной отметки ´ timestamp.  
Наименование поля задается в первом параметре. Эти наименования такие же,  
что и для функции date\_trunc. Выполните следующую команду SELECT extract(  
'microsecond' from timestamp '1999-11-27 12:34:56.123459' ); Она выводит не  
просто значение поля микросекунд, т. е. 123459, а дополнительно преобразует  
число секунд в микросекунды и добавляет значение поля микросекунд.  
Выполните эту команду, последовательно указывая в качестве первого  
параметра значения microsecond, millisecond, second, minute, hour, day, week,  
month, year, decade, century, millennium. Можно использовать сокращения этих  
наименований, которые приведены в предыдущем задании.

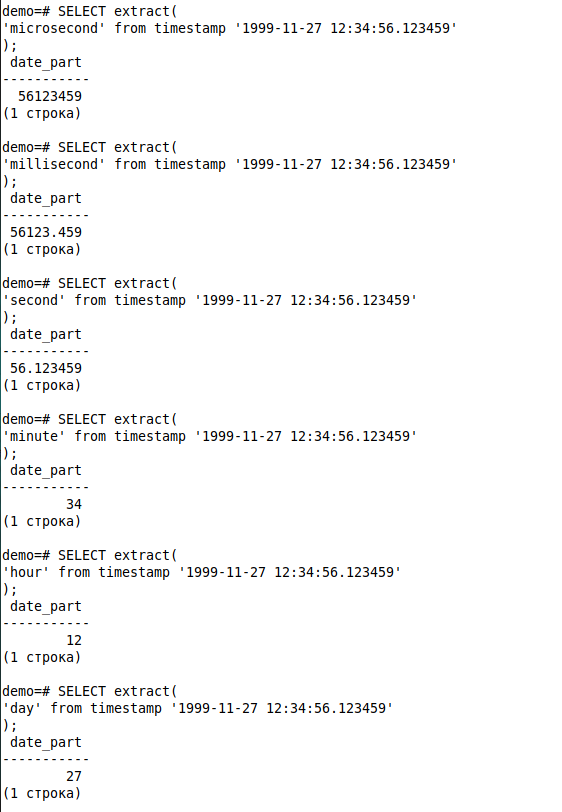


Рисунок 14 – Выполнение задания 27 часть 1

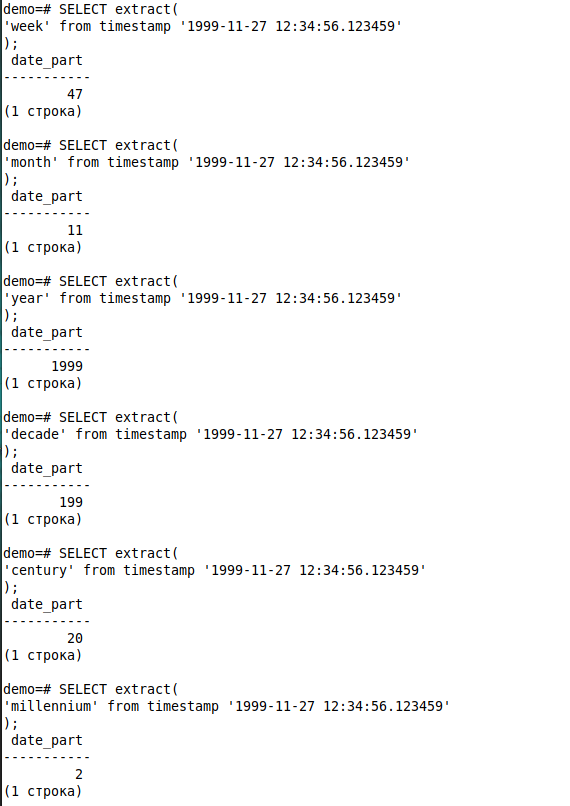


Рисунок 15 – Выполнение задания 27 часть 2

29. Как вы думаете, являются ли все приведенные ниже команды  
равнозначными в смысле результатов, получаемых с их помощью? SELECT \*  
FROM databases WHERE NOT is\_open\_source; SELECT \* FROM databases  
WHERE is\_open\_source <> 'yes'; SELECT \* FROM databases WHERE  
is\_open\_source <> 't'; SELECT \* FROM databases WHERE is\_open\_source <> '1';  
SELECT \* FROM databases WHERE is\_open\_source <> 1;

Нет, последняя команда ошибочна так как логический тип не может быть  
сравнён с целочисленным значением.

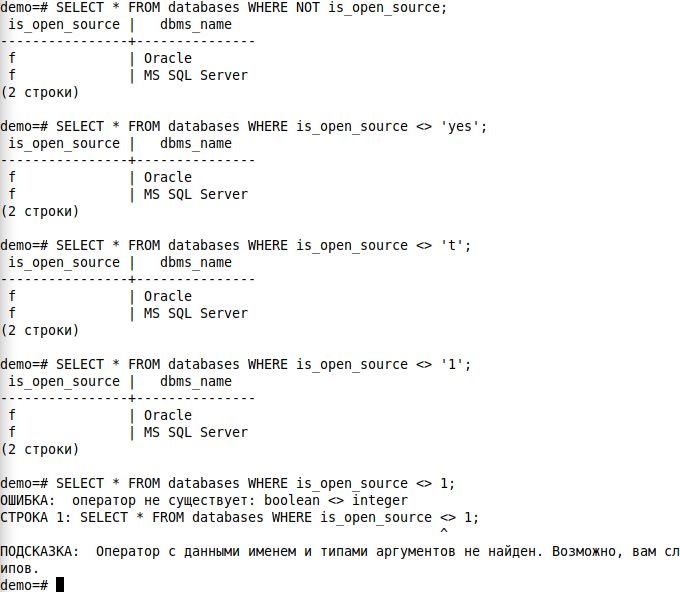


Рисунок 16 – Выполнение задания 29

31. ...А вот если мы захотим определить точный возраст каждого  
человека на текущий момент времени, то как получить этот результат? Первый  
вариант таков: SELECT \*, ( current\_date::timestamp - birthday::timestamp  
)::interval FROM birthdays; Этот вариант не дает результата, представленного в  
удобной форме: он показывает возраст в днях, а для пересчета числа дней в  
число лет нужны дополнительные действия. Хотя, наверное, возможны  
ситуации, когда требуется определить возраст именно в днях. В PostgreSQL  
предусмотрена специальная функция, позволяющая решить нашу задачу  
простым способом. Самостоятельно найдите ее описание в документации (см.  
раздел 9.9 «Операторы и функции даты/времени») и напишите команду с ее  
использованием.

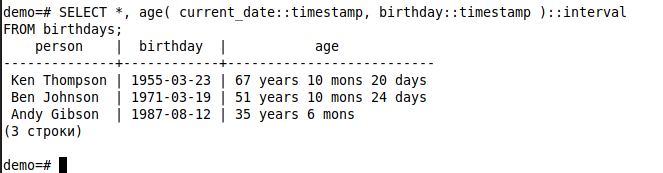


Рисунок 17 – Выполнение задания 31

33. ...Создайте новую версию таблицы pilots и соответственно измените  
команду INSERT, чтобы в ней содержались литералы двумерных массивов.  
Они будут выглядеть примерно так: '{ { "сосиска", "макароны", "кофе" }, {  
"котлета", "каша", "кофе" }, { "сосиска", "каша", "кофе" }, { "котлета", "каша",  
"чай" } }'::text[][] Сделайте ряд выборок и обновлений строк в этой таблице.

Для обращения к элементам двумерного массива нужно использовать два  
индекса. Не забывайте, что по умолчанию номера индексов начинаются с  
единицы.



Рисунок 18 – Выполнение задания 33 часть 1

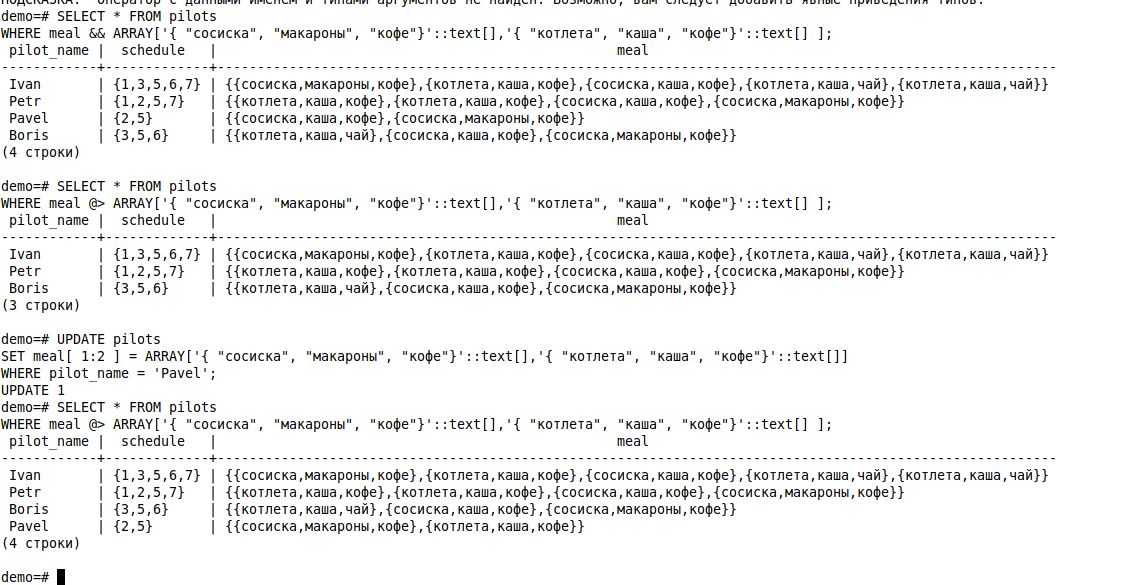


Рисунок 19 – Выполнение задания 33 часть 2

35. Изучая приемы работы с типами JSON, можно, как и в случае с  
массивами, пользоваться способностью команды SELECT обходиться без  
создания таблиц. Покажем лишь один пример. Добавить новый ключ и  
соответствующее ему значения в уже существующий объект можно оператором  
||: SELECT '{ "sports": "хоккей" }'::jsonb || '{ "trips": 5 }'::jsonb; Для работы с типами JSON предусмотрено много различных функций и операторов, представленных в разделе документации 9.15 «Функции и операторы JSON». Самостоятельно ознакомьтесь с ними, используя описанную технологию работы с командой SELECT.

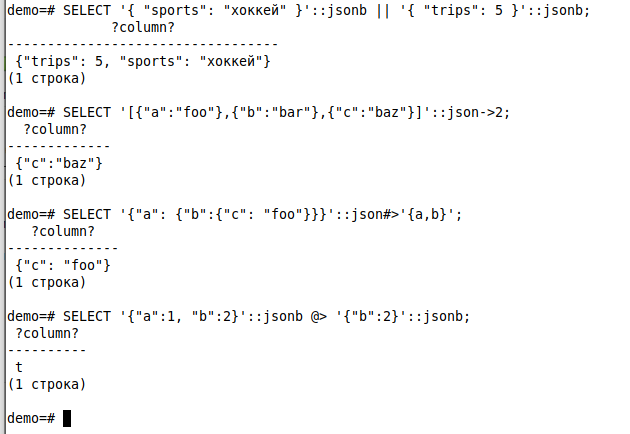


Рисунок 20 – Выполнение задания 35

37. Объекты JSON позволяют не только добавлять в них новые ключи, но  
также и удалять из них ключи существующие. Удалите один из ключей из  
JSON-объекта какой-нибудь строки таблицы pilots. Соответствующее ему  
значение будет также удалено, т. к. без ключа оно не может существовать. Воспользуйтесь оператором -.

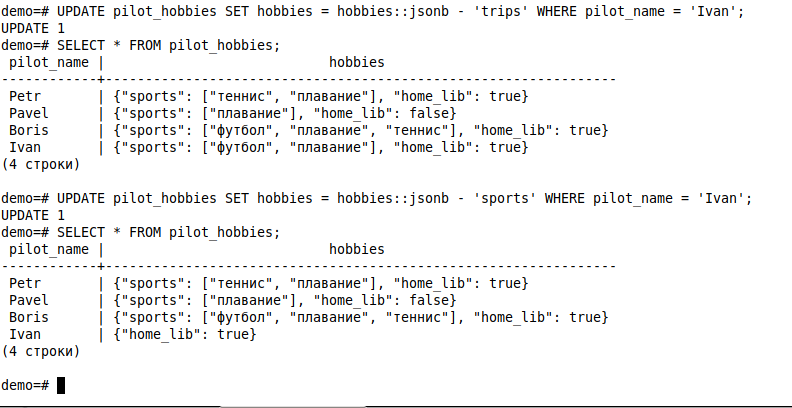


Рисунок 21 – Выполнение задания 37

**3 Вывод**

Были изучены типы данных СУБД PostgreSQL и выполнены задания к  
главе 3 учебного пособия.