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 №2**

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

тема

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Преподаватель | |  |  |  | А. Д. Вожжов |
|  | |  | подпись, дата |  | инициалы, фамилия |
| Студент | КИ23-17/2б, 032320981 |  |  |  | Р. А. Троицкий |
|  | номер группы, зачетной книжки |  | подпись, дата |  | инициалы, фамилия |

Красноярск 2025

Содержание

[1 Цели 3](#_Toc190097585)

[2 Задачи 3](#_Toc190097586)

[3 Ход выполнения 12](#_Toc190097587)

[4 Выводы 25](#_Toc190097588)

1. Цели

Ознакомиться с типами данных в SQL.

1. Задачи

Для выполнения практической работы необходимо выполнить следующие задачи.

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' ); Продемонстрируйте генерирование ошибки при попытке ввода числа, количество цифр в котором слева от десятичной точки (запятой) превышает допустимое.

2. 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. В качестве примера приведем команду: SELECT 'NaN'::numeric > 10000; ?column? ---------- t (1 строка)

3. 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): SELECT 'Inf'::double precision > 1E+308; ?column? ---------- t (1 строка) Выполните аналогичный запрос для наименьшего возможного значения типа double precision.

4. 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. 76 Контрольные вопросы и задания 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.

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

6. 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. 78 Контрольные вопросы и задания SELECT current\_time; timetz -------------------- 19:46:14.584641+03 (1 строка) SELECT current\_time::time( 0 ); time ---------- 19:39:45 (1 строка) SELECT current\_time::time( 3 ); time -------------- 19:39:54.085 (1 строка) Выполните подобные команды для типов timestamp и interval. Тип date такой возможности — задавать точность — не имеет. Как вы думаете, почему?

7. 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.

8. 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. SELECT to\_char( current\_timestamp, 'mi:ss' ); to\_char --------- 47:43 (1 строка) 82 Контрольные вопросы и задания SELECT to\_char( current\_timestamp, 'dd' ); to\_char --------- 12 (1 строка) SELECT to\_char( current\_timestamp, 'yyyy-mm-dd' ); to\_char ------------ 2017-03-12 (1 строка) Поэкспериментируйте с этой функцией, извлекая из значения типа timestamp различные поля и располагая их в нужном вам порядке.

9. 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; ОШИБКА: неверный синтаксис для типа time: "21:15:16:22" СТРОКА 1: select '21:15:16:22'::time; ^

10. 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. Подробное описание всех допустимых арифметических операций с датами и временем приведено в документации в разделе 9.9 «Операторы и функции даты/времени».

11. 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;

12. 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 );

13. 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' ) ); date\_trunc --------------------- 1999-11-27 12:34:56 (1 строка) 85 Глава 4. Типы данных СУБД PostgreSQL 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. Обратите внимание, что результирующее значение получается не путем округления исходного значения, а именно путем отбрасывания более мелких единиц. При этом поля времени (часы, минуты и секунды) заменяются нулями, а поля даты (годы, месяцы и дни) — заменяются цифрами 01. Однако при использовании параметра week картина получается более интересная.

14. 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. date\_part ----------- 56123459 (1 строка) 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. 86 Контрольные вопросы и задания Обратите внимание, что в ряде случаев выводится не просто конкретное поле (фрагмент) из временной отметки, а некоторый продукт переработки этого ´ поля. Например, если в качестве первого параметра функции extract в вышеприведенной команде указать cent (век), то мы получим в ответ не 19 (что и было бы буквальным значением поля «век»), а 20, поскольку 1999 год принадлежит двадцатому веку

15. В тексте главы мы создавали таблицу с помощью команды CREATE TABLE databases ( is\_open\_source boolean, dbms\_name text ); и заполняли ее данными. INSERT INTO databases VALUES ( TRUE, 'PostgreSQL' ); INSERT INTO databases VALUES ( FALSE, 'Oracle' ); INSERT INTO databases VALUES ( TRUE, 'MySQL' ); INSERT INTO databases VALUES ( FALSE, 'MS SQL Server' ); 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;

16. Пусть в таблице birthdays хранятся даты рождения какой-то группы людей. Создайте эту таблицу с помощью команды CREATE TABLE birthdays ( person text NOT NULL, birthday date NOT NULL ); Добавьте в нее несколько строк, например: INSERT INTO birthdays VALUES ( 'Ken Thompson', '1955-03-23' ); INSERT INTO birthdays VALUES ( 'Ben Johnson', '1971-03-19' ); INSERT INTO birthdays VALUES ( 'Andy Gibson', '1987-08-12' ); Давайте выберем из таблицы birthdays строки для всех людей, родившихся в каком-то конкретном месяце, например, в марте: SELECT \* FROM birthdays WHERE extract( 'mon' from birthday ) = 3; В этой команде в вызове функции extract имеет место неявное приведение типов, т. к. ее вторым параметром должно быть значение типа timestamp. Полагаться на неявное приведение типов можно не всегда. person | birthday --------------+------------ Ken Thompson | 1955-03-23 Ben Johnson | 1971-03-19 (2 строки) 88 Контрольные вопросы и задания Если нам потребуется выяснить, кто из этих людей достиг возраста, скажем, 40 лет на момент выполнения запроса, то команда может быть такой (в последнем столбце показана дата достижения возраста 40 лет): SELECT \*, birthday + '40 years'::interval FROM birthdays WHERE birthday + '40 years'::interval < current\_timestamp; person | birthday | ?column? --------------+------------+--------------------- Ken Thompson | 1955-03-23 | 1995-03-23 00:00:00 Ben Johnson | 1971-03-19 | 2011-03-19 00:00:00 (2 строки) Можно заменить current\_timestamp на current\_date: SELECT \*, birthday + '40 years'::interval FROM birthdays WHERE birthday + '40 years'::interval < current\_date; 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; person | birthday | interval --------------+------------+------------ Ken Thompson | 1955-03-23 | 22477 days Ben Johnson | 1971-03-19 | 16637 days Andy Gibson | 1987-08-12 | 10647 days (3 строки) 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.

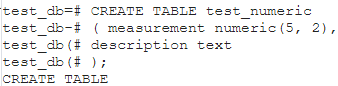
17. В разделе документации 8.15 «Массивы» сказано, что массивы могут быть многомерными и в них могут содержаться значения любых типов. Давайте сначала рассмотрим одномерные массивы текстовых значений. Предположим, что пилоты авиакомпании имеют возможность высказывать свои пожелания насчет конкретных блюд, из которых должен состоять их обед во время полета. Для учета пожеланий пилотов необходимо модифицировать таблицу pilots, с которой мы работали в разделе 4.5. CREATE TABLE pilots ( pilot\_name text, schedule integer[], meal text[] ); 90 Контрольные вопросы и задания Добавим строки в таблицу: INSERT INTO pilots VALUES ( 'Ivan', '{ 1, 3, 5, 6, 7 }'::integer[], '{ "сосиска", "макароны", "кофе" }'::text[] ), ( 'Petr', '{ 1, 2, 5, 7 }'::integer [], '{ "котлета", "каша", "кофе" }'::text[] ), ( 'Pavel', '{ 2, 5 }'::integer[], '{ "сосиска", "каша", "кофе" }'::text[] ), ( 'Boris', '{ 3, 5, 6 }'::integer[], '{ "котлета", "каша", "чай" }'::text[] ); INSERT 0 4 Обратите внимание, что каждое из текстовых значений, включаемых в литерал массива, заключается в двойные кавычки, а в качестве типа данных указывается text[]. Вот что получилось: SELECT \* FROM pilots; pilot\_name | schedule | meal ------------+-------------+------------------------- Ivan | {1,3,5,6,7} | {сосиска,макароны,кофе} Petr | {1,2,5,7} | {котлета,каша,кофе} Pavel | {2,5} | {сосиска,каша,кофе} Boris | {3,5,6} | {котлета,каша,чай} (4 строки) Давайте получим список пилотов, предпочитающих на обед сосиски: SELECT \* FROM pilots WHERE meal[ 1 ] = 'сосиска'; pilot\_name | schedule | meal ------------+-------------+------------------------- Ivan | {1,3,5,6,7} | {сосиска,макароны,кофе} Pavel | {2,5} | {сосиска,каша,кофе} (2 строки) 91 Глава 4. Типы данных СУБД PostgreSQL Предположим, что руководство авиакомпании решило, что пища пилотов должна быть разнообразной. Оно позволило им выбрать свой рацион на каждый из четырех дней недели, в которые пилоты совершают полеты. Для нас это решение руководства выливается в необходимость модифицировать таблицу, а именно: столбец meal теперь будет содержать двумерные массивы. Определение этого столбца станет таким: meal text[][]. Задание. Создайте новую версию таблицы 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.

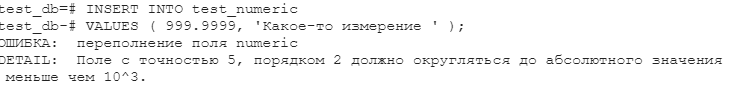
18. Изучая приемы работы с типами JSON, можно, как и в случае с массивами, пользоваться способностью команды SELECT обходиться без создания таблиц. Покажем лишь один пример. Добавить новый ключ и соответствующее ему значения в уже существующий объект можно оператором ||: SELECT '{ "sports": "хоккей" }'::jsonb || '{ "trips": 5 }'::jsonb; ?column? ---------------------------------- {"trips": 5, "sports": "хоккей"} (1 строка) 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.

19. 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 -.

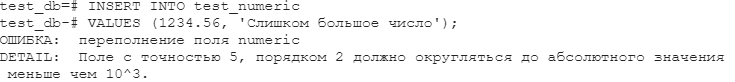
1. Ход выполнения

Выполнил пункт 1 задач, получил сообщение об ошибке, которое появилось из-за того, что 999.9999 округляется до 1000.00, что превышает допустимое количество цифр (5), остальные команды выполнятся успешно так как округление не превысит допустимое количество цифр. При попытке ввести слишком большое число тоже получил ошибку так как заданный формат предусматривает 5- 2 = 3 цифры слева от запятой.

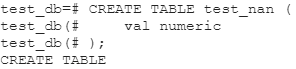


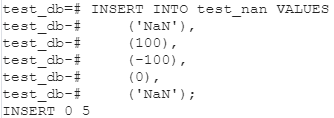


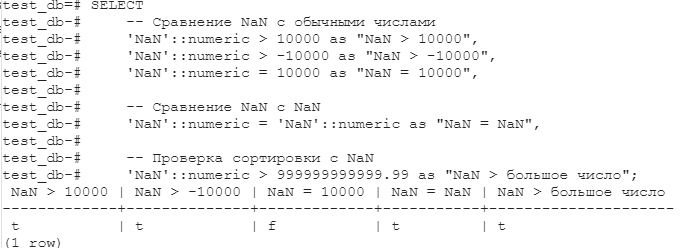




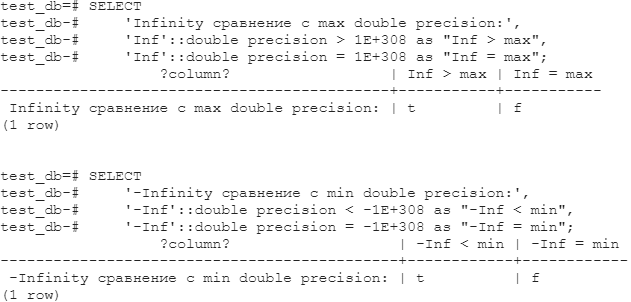
Выполнил пункт 2 задач



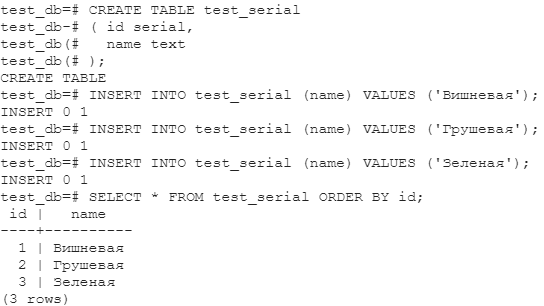


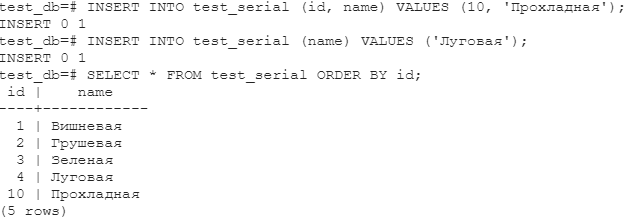


Выполнил пункт 3 задач



Выполнил пункт 4 задач

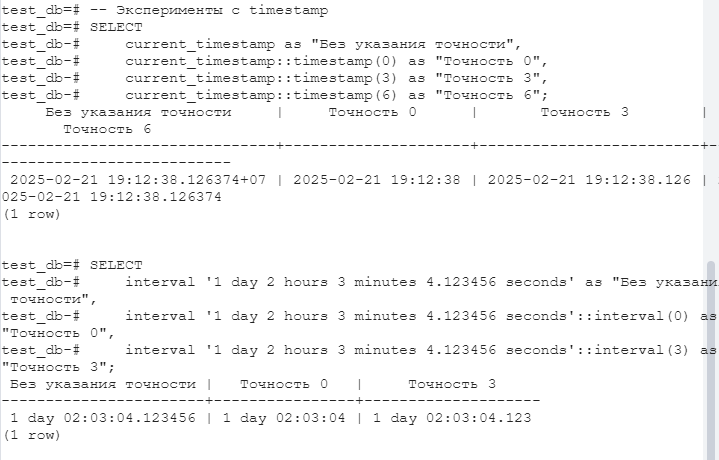




Выполнил пункт 5 задач

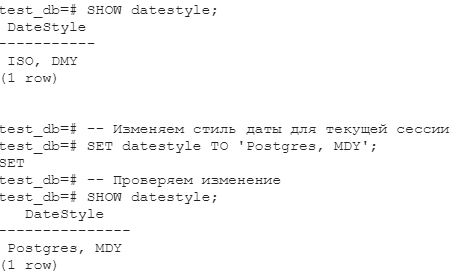
PostgreSQL использует григорианский календарь для работы с датами

Выполнил пункт 6 задач

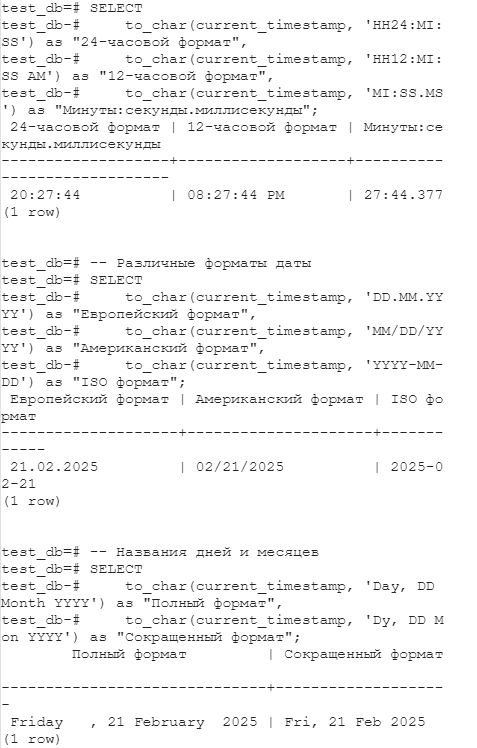


date представляет только дату (год, месяц, день), минимальная еденица измерения здесь день, поэтому нет необходимости указывать его точнее

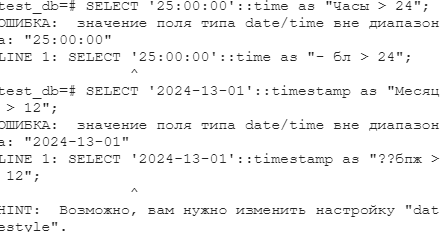
Выполнил пункт 7 задач



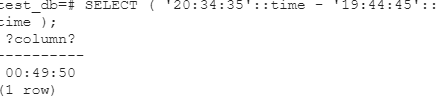
Выполнил пункт 8 задач

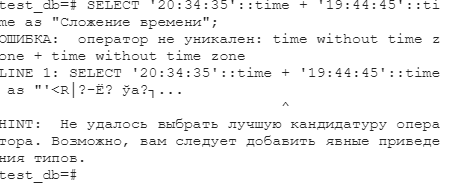


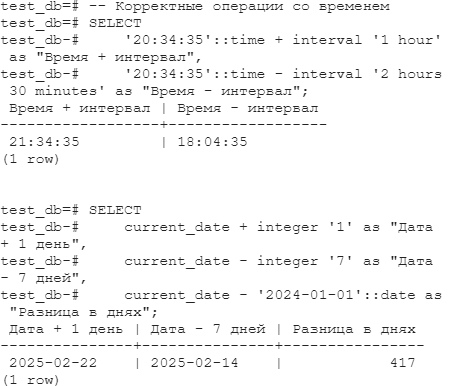
Выполнил пункт 9 задач



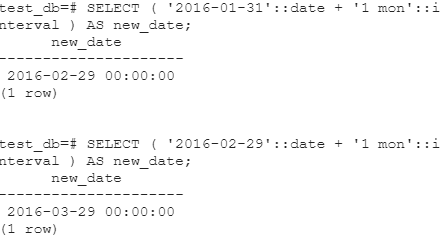
Выполнил пункт 10 задания



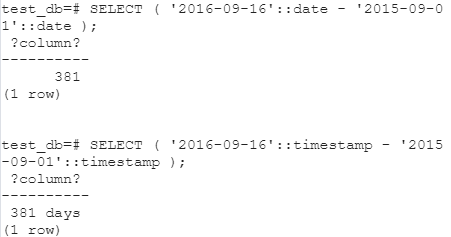




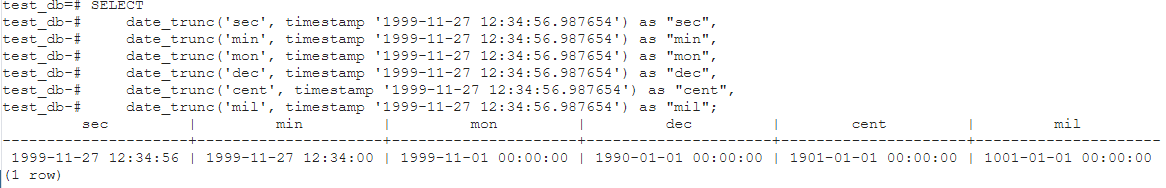
Выполнил пункт 11 задания

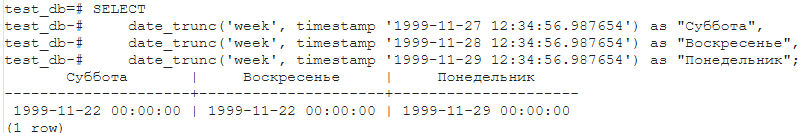


Выполнил пункт 12 задания, в обоих случаях получил одинаковое число дней, но в случае с вычитанием дат результат имеет тип integer, а в случае вычитания временных меток результат имеет тип interval.

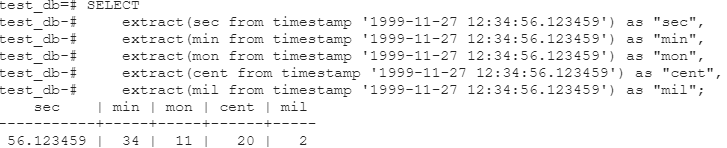


Выполнил пункт 13 задания, при использовании week дата приводится к началу недели

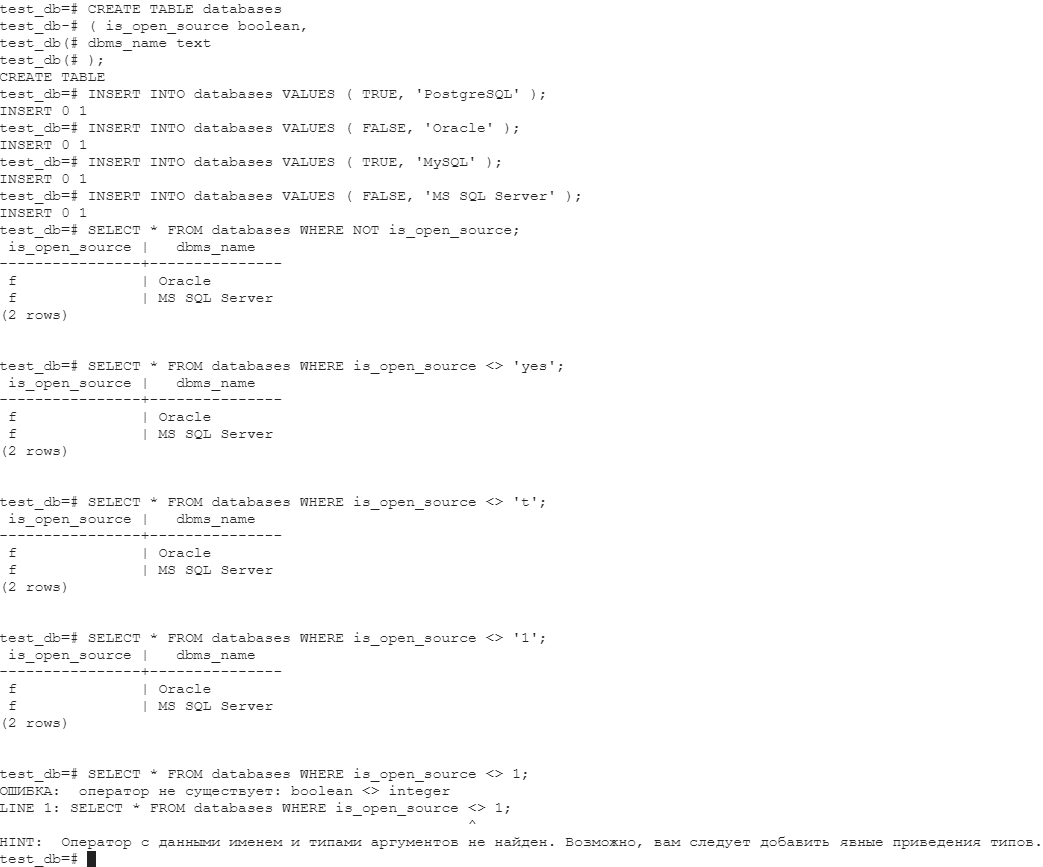




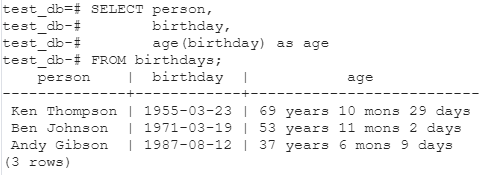
Выполнил пункт 14 задания



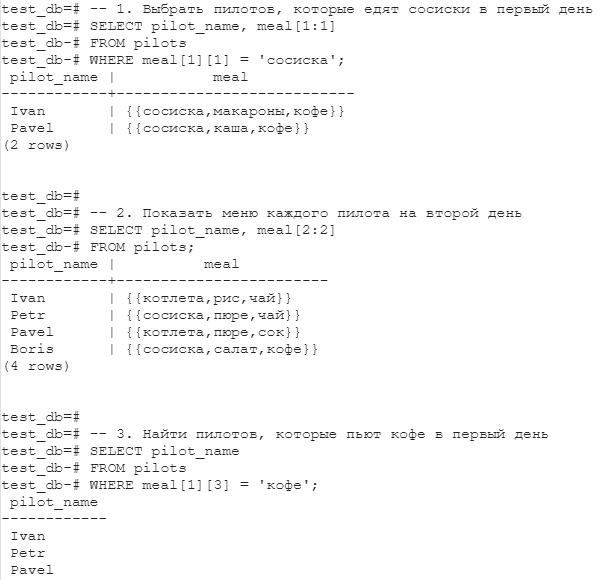
Выполнил пункт 15 задания



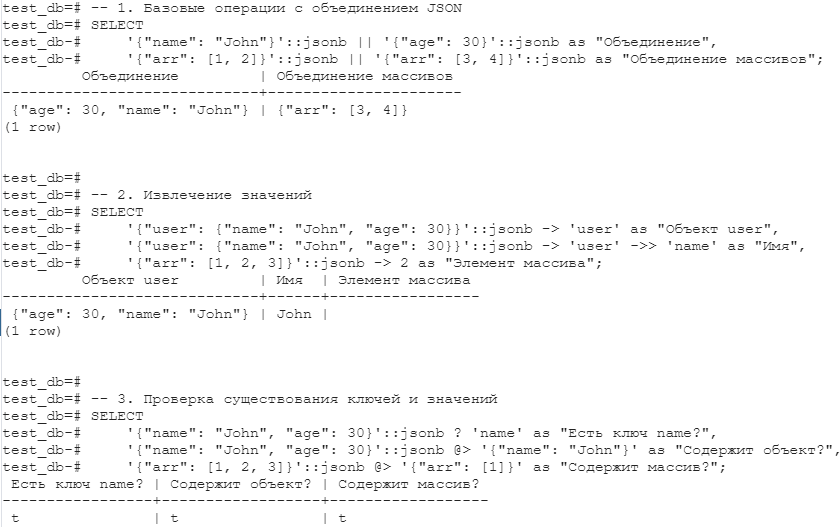
Выполнил пункт 16 задания

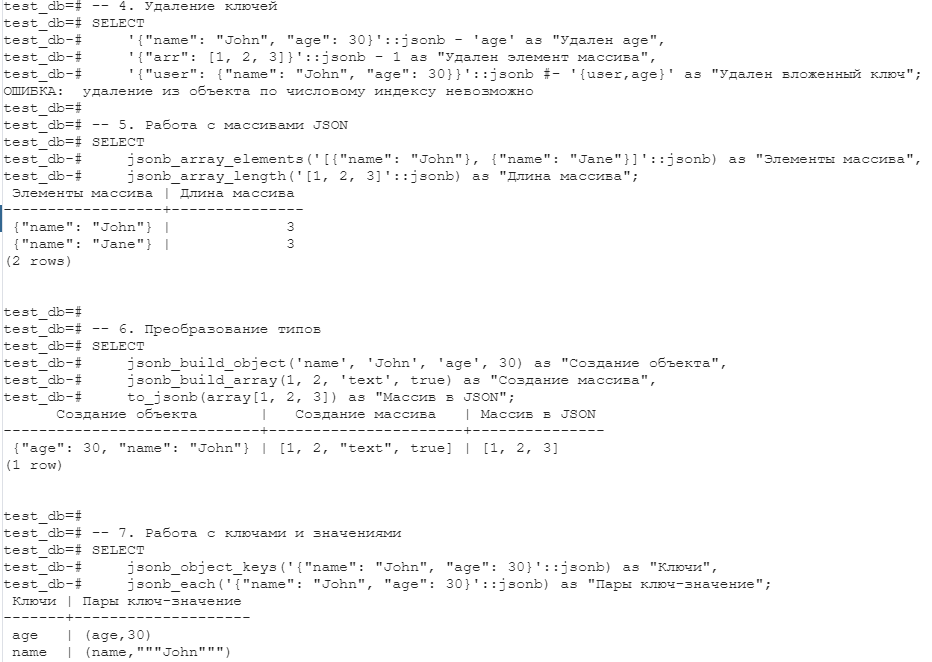
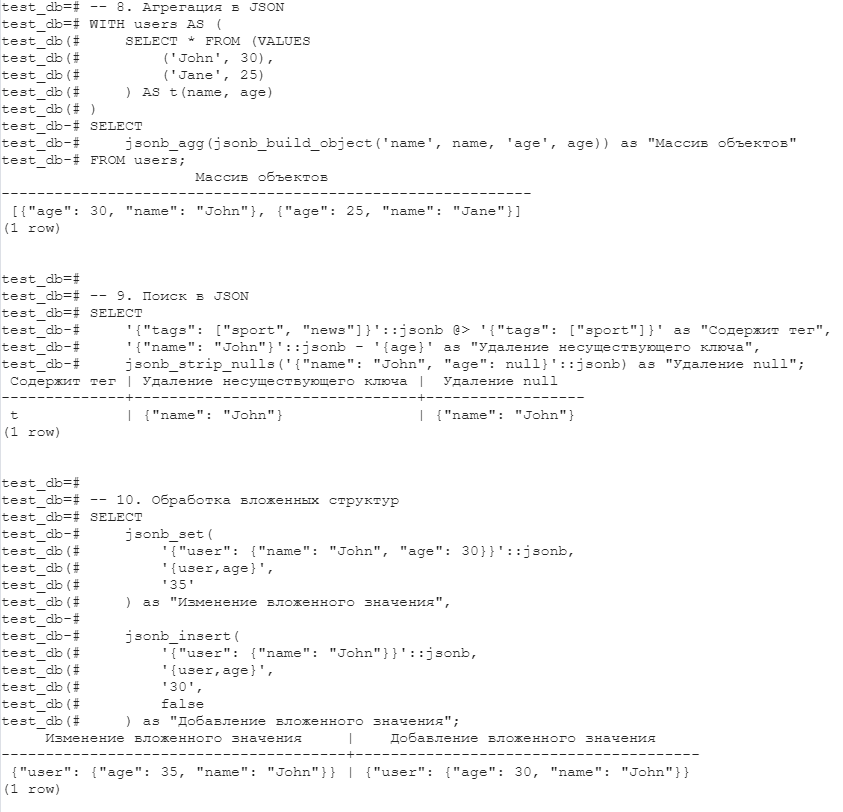


Выполнил пункт 17 задания

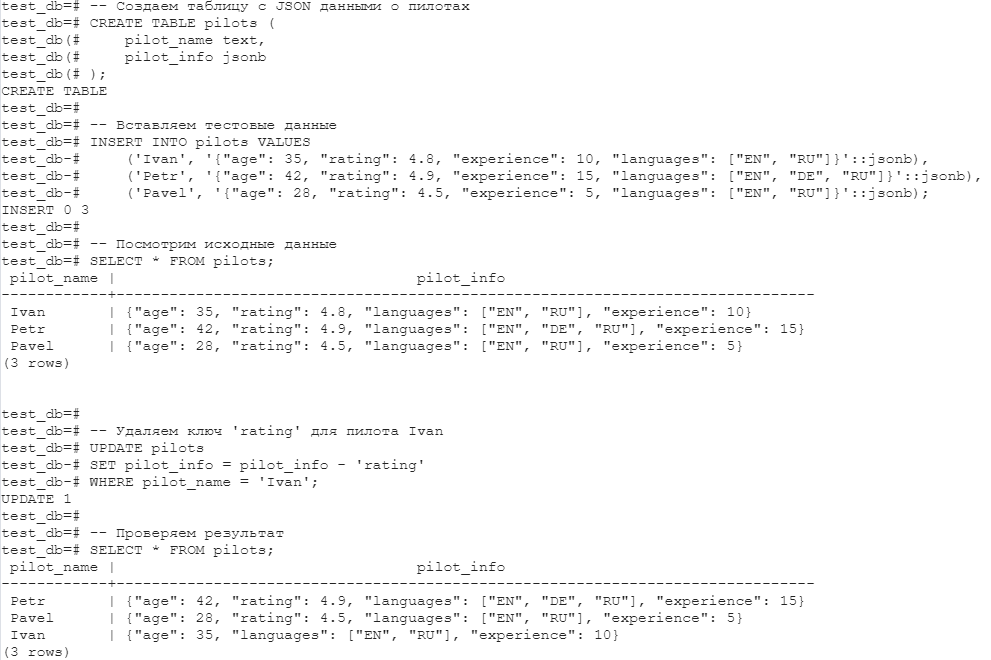


Выполнил пункт 18 задач



Выполнил пункт 19 задач



4 Выводы

Выполнив задание, были изучены типы данных SQL.