Федеральное государственное бюджетное образовательное учреждение высшего образования
"Уфимский государственный авиационный технический университет"

Кафедра Высокопроизводительных вычислительных технологий и систем

Дисциплина: Базы данных

Отчет по лабораторной работе № 1

Тема: «Введение в SQL. Типы данных PostgreSQL.»

Группа ПМ-353	Фамилия И.О.	Подпись	Дата	Оценка
Студент	Шамаев И.Р.			
Принял	Ямилева А.М.			

Уфа 2022

Цель: изучить типы данных PostgreSQL и основные операции с таблицами.

Теоретический материал

Язык SQL — очень многообразный, он включает в себя целый ряд команд, которые, в свою очередь, иной раз имеют множество параметров и ключевых слов.

Для создания таблиц в языке SQL служит команда CREATE TABLE. Ее полный синтаксис представлен в документации на PostgreSQL, а упрощенный синтаксис таков:

```
CREATE TABLE имя-таблицы
(
имя-поля тип-данных [ограничения-целостности],
имя-поля тип-данных [ограничения-целостности],
...
имя-поля тип-данных [ограничения-целостности],
[ограничение-целостности],
[первичный-ключ],
[внешний-ключ]
);
```

Команда для удаления таблицы:

```
DROP TABLE имя-таблицы;
```

Для ввода данных служит команда INSERT. Ее упрощенный формат таков:

```
INSERT INTO имя-таблицы [( имя-атрибута, имя-атрибута, ... )]
VALUES ( значение-атрибута, значение-атрибута, ... );
```

Для выборки информации из таблиц базы данных служит команда SELECT. Ее синтаксис, упрощенный до предела, таков:

```
SELECT имя-атрибута, имя-атрибута, ...
FROM имя-таблицы;
```

Часто бывает так, что требуется вывести значения из всех столбцов таблицы. В таком случае можно не перечислять имена атрибутов, а просто ввести символ «*».

Если же вы хотите каким-то образом упорядочить расположение выводимых строк, то необходимо предпринять дополнительные меры, а именно: использовать предложение ORDER BY команды SELECT.

Теперь мы ознакомимся с командой UPDATE, предназначенной для обновления данных в таблицах. Ее упрощенный синтаксис таков:

```
UPDATE имя-таблицы
```

SET имя-атрибута1 = значение-атрибута1,
имя-атрибута2 = значение-атрибута2, ...
WHERE условие;

Итак, мы добрались до операции удаления строк из таблиц. Для этого используется команда DELETE, которая похожа на команду SELECT:
DELETE FROM имя-таблицы WHERE условие;

Типы данных — это одно из базовых понятий любого языка программирования, и язык SQL в этом плане не является исключением.

Группа числовых типов данных включает в себя целый ряд разновидностей: целочисленные типы, числа фиксированной точности, типы данных с плавающей точкой, последовательные типы (serial). В составе целочисленных типов находятся следующие представители: smallint, integer, bigint.

Числа фиксированной точности представлены двумя типами — numeric и decimal. Однако они являются идентичными по своим возможностям. Поэтому мы будем проводить изложение на примере типа numeric.

Параметры этого типа данных указываются в круглых скобках после имени типа:
numeric(точность, масштаб).

Его главное достоинство — это обеспечение точных результатов при выполнении вычислений, когда это, конечно, возможно в принципе. Это оказывается возможным при выполнении сложения, вычитания и умножения.

Данный тип следует выбирать для хранения денежных сумм, а также в других случаях, когда требуется гарантировать точность вычислений.

Представителями типов данных с плавающей точкой являются типы real и double precision. Тип данных real может представить числа в диапазоне, как минимум, от 1E−37 до 1E+37 с точностью не меньше 6 десятичных цифр. Тип double precision имеет диапазон значений примерно от 1E−307 до 1E+308 с точностью не меньше 15 десятичных цифр.

PostgreSQL поддерживает также тип данных float, определенный в стандарте SQL. В объявлении типа может использоваться параметр: float(p). Если его значение лежит в диапазоне от 1 до 24, то это будет равносильно использованию типа real, а если же значение лежит в диапазоне от 25 до 53, то это будет равносильно использованию типа double precision. Если же при объявлении типа параметр не используется, то это также будет равносильно использованию типа double precision.

Последним из числовых типов является тип serial. Однако он фактически реализован не как настоящий тип, а просто как удобная замена целой группы SQL-команд. Тип serial удобен в тех случаях, когда требуется в какой-либо

столбец вставлять уникальные целые значения, например, значения суррогатного первичного ключа.

Стандартные представители строковых типов — это типы character varying(n) и character(n), где параметр указывает максимальное число символов в строке, которую можно сохранить в столбце такого типа. При работе с многобайтовыми кодировками символов, например UTF-8, нужно учитывать, что речь идет о символах, а не о байтах. Если сохраняемая строка символов будет короче, чем указано в определении типа, то значение типа character будет дополнено пробелами до требуемой длины, а значение типа character varying будет сохранено так, как есть.

Типы character varying(n) и character(n) имеют псевдонимы varchar(n) и char(n) соответственно. На практике, как правило, используют именно эти краткие псевдонимы.

PostgreSQL дополнительно предлагает еще один символьный тип — text. В столбец этого типа можно ввести сколь угодно большое значение, конечно, в пределах, установленных при компиляции исходных текстов СУБД.

PostgreSQL поддерживает все типы данных, предусмотренные стандартом SQL для даты и времени. Даты обрабатываются в соответствии с григорианским календарем, причем это делается даже в тех случаях, когда дата относится к тому моменту времени, когда этот календарь в данной стране еще не был принят. Для этих типов данных предусмотрены определенные форматы для ввода значений и для вывода. Причем эти форматы могут не совпадать. Важно помнить, что при вводе значений их нужно заключать в одинарные кавычки, как и текстовые строки.

Начнем рассмотрение с типа date. Рекомендуемый стандартом ISO 8601 формат ввода дат таков: «yyyy-mm-dd», где символы «y», «m» и «d» обозначают цифру года, месяца и дня соответственно.

Для хранения времени суток служат два типа данных: time и time with time zone. Первый из них хранит только время суток, а второй — дополнительно — еще и часовой пояс.

В результате объединения типов даты и времени получается интегральный тип — временная отметка. Этот тип существует в двух вариантах: с учетом часового пояса — timestamp with time zone, либо без учета часового пояса — timestamp. Для
первого варианта существует сокращенное наименование — timestamptz, которое является расширением PostgreSQL.

Последним типом является interval, который представляет продолжительность отрезка времени между двумя моментами времени. Его формат ввода таков:

quantity unit [quantity unit ...] direction
Здесь unit означает единицу измерения, а quantity — количество таких единиц. В качестве единиц измерения можно использовать следующие: microsecond, millisecond, second, minute, hour, day, week, month, year, decade, century, millennium. Параметр direction может принимать значение ago (т. е. «тому назад») либо быть пустым.

Логический (boolean) тип может принимать три состояния: истина и ложь, а также неопределенное состояние, которое можно представить значением NULL. Таким образом, тип boolean реализует трехзначную логику.

В качестве истинного состояния могут служить следующие значения: TRUE, 't', 'true', 'y', 'yes', 'on', '1'.

В качестве ложного состояния могут служить следующие значения: FALSE, 'f', 'false', 'n', 'no', 'off', '0'.

PostgreSQL позволяет создавать в таблицах такие столбцы, в которых будут содержаться не скалярные значения, а массивы переменной длины. Эти массивы могут быть многомерными и могут содержать значения любого из встроенных типов, а также типов данных, определенных пользователем.

Для указания на то, что это массив, нужно добавить квадратные скобки к наименованию типа данных. При этом задавать число элементов не обязательно.

Типы JSON предназначены для сохранения в столбцах таблиц базы данных таких значений, которые представлены в формате JSON (JavaScript Object Notation). Существует два типа: json и jsonb. Основное различие между ними заключается в быстродействии. Если столбец имеет тип json, тогда сохранение значений происходит быстрее, потому что они записываются в том виде, в котором были введены. Но при последующем использовании этих значений в качестве операндов или параметров функций будет каждый раз выполняться их разбор, что замедляет работу. При использовании типа jsonb разбор производится однократно, при записи значения в таблицу. Это несколько замедляет операции вставки строк, в которых содержатся значения данного типа. Но все последующие обращения к сохраненным значениям выполняются быстрее, т. к. выполнять их разбор уже не требуется.

Есть еще ряд отличий, в частности, тип json сохраняет порядок следования ключей в объектах и повторяющиеся значения ключей, а тип jsonb этого не делает. Рекомендуется в приложениях использовать тип jsonb, если только нет каких-то особых аргументов в пользу выбора типа json.

Практическая часть

Задание 1

```
demo=# INSERT INTO aircrafts VALUES ( 'SU9', 'Sukhoi SuperJet-100', 3000 );
ОШИБКА:  повторяющееся значение ключа нарушает ограничение уникальности "aircrafts_pkey"
ПОДРОБНОСТИ:  Ключ "(aircraft_code)=(SU9)" уже существует.
```

Задание 2

```
demo=# SELECT * FROM aircrafts ORDER BY range DESC;
 aircraft_code |        model         | range
---------------+----------------------+-------
 773           | Boeing 777-300       | 11100
 763           | Boeing 767-300       |  7900
 319           | Airbus A319-100      |  6700
 320           | Airbus A320-200      |  5700
 321           | Airbus A321-200      |  5600
 733           | Boeing 737-300       |  4200
 SU9           | Sukhoi SuperJet-100  |  3000
 CR2           | Bombardier CRJ-200   |  2700
 CN1           | Cessna 208 Caravan   |  1200
(9 строк)
```

Задание 3

```
demo=# UPDATE aircrafts SET range=range*2 WHERE model='Sukhoi SuperJet-100';
UPDATE 1
demo=# SELECT * FROM aircrafts ORDER BY range DESC;
 aircraft_code |        model         | range
---------------+----------------------+-------
 773           | Boeing 777-300       | 11100
 763           | Boeing 767-300       |  7900
 319           | Airbus A319-100      |  6700
 SU9           | Sukhoi SuperJet-100  |  6000
 320           | Airbus A320-200      |  5700
 321           | Airbus A321-200      |  5600
 733           | Boeing 737-300       |  4200
 CR2           | Bombardier CRJ-200   |  2700
 CN1           | Cessna 208 Caravan   |  1200
(9 строк)
```

Задание 1

```
demo=# CREATE TABLE test_numeric (measurement numeric(5, 2), description text);
CREATE TABLE
demo=# INSERT INTO test_numeric VALUES ( 999.9999, 'Какое-то измерение ' );
ОШИБКА:  переполнение поля numeric
ПОДРОБНОСТИ:  Поле с точностью 5, порядком 2 должно округляться до абсолютного значения меньше чем 10^3.
demo=# INSERT INTO test_numeric VALUES ( 999.9009, 'Еще одно измерение' );
INSERT 0 1
demo=# INSERT INTO test_numeric VALUES ( 999.1111, 'И еще измерение' );
INSERT 0 1
demo=# INSERT INTO test_numeric VALUES ( 998.9999, 'И еще одно' );
INSERT 0 1
demo=# INSERT INTO test_numeric VALUES ( 9989.9999, 'И еще одно' );
ОШИБКА:  переполнение поля numeric
ПОДРОБНОСТИ:  Поле с точностью 5, порядком 2 должно округляться до абсолютного значения меньше чем 10^3.
demo=#
```

Задание 2

```
demo=# DROP TABLE test_numeric;
DROP TABLE
demo=# CREATE TABLE test_numeric ( measurement numeric, description text);
CREATE TABLE
demo=# INSERT INTO test_numeric VALUES ( 1234567890.0987654321, 'Точность 20 знаков, масштаб 10 знаков' );
INSERT 0 1
demo=# INSERT INTO test_numeric VALUES ( 1.5, 'Точность 2 знака, масштаб 1 знак' );
INSERT 0 1
demo=# INSERT INTO test_numeric VALUES ( 0.12345678901234567890, 'Точность 21 знак, масштаб 20 знаков' );
INSERT 0 1
demo=# INSERT INTO test_numeric VALUES ( 1234567890, 'Точность 10 знаков, масштаб 0 знаков (целое число)' );
INSERT 0 1
demo=# SELECT * FROM test_numeric;
      measurement       |                   description
------------------------+-------------------------------------------------
  1234567890.0987654321 | Точность 20 знаков, масштаб 10 знаков
                   1.5 | Точность 2 знака, масштаб 1 знак
 0.12345678901234567890 | Точность 21 знак, масштаб 20 знаков
             1234567890 | Точность 10 знаков, масштаб 0 знаков (целое число)
(4 строки)
```

Задание 3

```
demo=# SELECT 'NaN'::numeric > 10000;
 ?column?
----------
 t
(1 строка)


demo=# SELECT 'NaN'::numeric = 'NaN'::numeric;
 ?column?
----------
 t
(1 строка)
```

Задание 4

```
demo=# SELECT 1.79e+308::double precision > 1.78e+308::double precision;
 ?column?
----------
 t
(1 строка)


demo=# SELECT 1.79e+308::double precision;
  float8
-----------
 1.79e+308
(1 строка)


demo=# SELECT 1.78e+308::double precision;
  float8
-----------
 1.78e+308
(1 строка)


demo=# SELECT 1.8e+308::double precision > 1.78e+308::double precision;
ОШИБКА:  "18000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000" вне диапазона для типа double precision
```

```
demo=# SELECT 3.40e+38::real > 3.39e+38::real;
 ?column?
----------
 t
(1 строка)


demo=# SELECT 3.40e+38::real;
 float4
---------
 3.4e+38
(1 строка)


demo=# SELECT 3.39e+38::real;
  float4
----------
 3.39e+38
(1 строка)


demo=# SELECT 3.41e+38::real > 3.39e+38::real;
ОШИБКА:  "341000000000000000000000000000000000000" вне диапазона для типа real
```

Задание 5

```
demo=# SELECT '-Inf'::double precision < 5E-324;
 ?column?
----------
 t
(1 строка)


demo=# SELECT 'Inf'::double precision > 1E+308;
 ?column?
----------
 t
(1 строка)
```

Задание 15

```
demo=# SELECT current_timestamp;
       current_timestamp
-------------------------------
 2022-05-02 21:08:51.426975+05
(1 строка)


demo=# SELECT to_char( current_timestamp, 'mi:ss' );
 to_char
---------
 08:53
(1 строка)


demo=# SELECT to_char( current_timestamp, 'dd' );
 to_char
---------
 02
(1 строка)
```

```
demo=# SELECT to_char( current_timestamp, 'yyyy-mm-dd' );
  to_char
------------
 2022-05-02
(1 строка)


demo=# SELECT to_char( current_timestamp, 'yyyy-mm-dd hh24:mi:ss' );
       to_char
---------------------
 2022-05-02 21:09:24
(1 строка)
```

Задание 16

```
demo=# SELECT 'Feb 29, 2015'::date;
ОШИБКА:  значение поля типа date/time вне диапазона: "Feb 29, 2015"
СТРОКА 1: SELECT 'Feb 29, 2015'::date;
                 ^
```

Задание 17

```
demo=# SELECT '21:15:16:22'::time;
ОШИБКА:  неверный синтаксис для типа time: "21:15:16:22"
СТРОКА 1: SELECT '21:15:16:22'::time;
                 ^
```

Задание 18

```
demo=# SELECT ( '2016-09-16'::date - '2016-09-01'::date );
 ?column?
----------
       15
(1 строка)
```

Задание 19

```
demo=# SELECT ( '20:34:35'::time - '19:44:45'::time );
 ?column?
----------
 00:49:50
(1 строка)


demo=# SELECT ( '20:34:35'::time + '19:44:45'::time );
ОШИБКА:  оператор не уникален: time without time zone + time without time zone
СТРОКА 1: SELECT ( '20:34:35'::time + '19:44:45'::time );
                                    ^
ПОДСКАЗКА:  Не удалось выбрать лучшую кандидатуру оператора. Возможно, вам следует добавить явные приведения типов.
```

Задание 20

```
demo=# SELECT ( current_timestamp - '2016-01-01'::timestamp ) AS new_date;
         new_date
---------------------------
 2313 days 21:15:51.724819
(1 строка)


demo=# SELECT ( current_timestamp + '1 mon'::interval ) AS new_date;
           new_date
-------------------------------
 2022-06-02 21:16:22.613554+05
(1 строка)


demo=# SELECT ( current_timestamp + '1 mon'::interval );
           ?column?
-------------------------------
 2022-06-02 21:16:30.813525+05
(1 строка)
```

Задание 21

```
demo=# SELECT ( '2016-01-31'::date + '1 mon'::interval ) AS new_date;
      new_date
---------------------
 2016-02-29 00:00:00
(1 строка)


demo=# SELECT ( '2016-02-29'::date + '1 mon'::interval ) AS new_date;
      new_date
---------------------
 2016-03-29 00:00:00
(1 строка)


demo=# SELECT ( '2016-03-31'::date + '1 mon'::interval ) AS new_date;
      new_date
---------------------
 2016-04-30 00:00:00
(1 строка)
```

Задание 32

```
demo=# SELECT array_cat( ARRAY[ 1, 2, 3 ], ARRAY[ 3, 5 ] );
  array_cat
-------------
 {1,2,3,3,5}
(1 строка)


demo=# SELECT array_remove( ARRAY[ 1, 2, 3 ], 3 );
 array_remove
--------------
 {1,2}
(1 строка)


demo=# SELECT ARRAY[1.1,2.1,3.1]::int[] = ARRAY[1,2,3];
 ?column?
----------
 t
(1 строка)
```

Задание 34

```
demo=# CREATE TABLE pilot_hobbies(pilot_name text, hobbies jsonb);
CREATE TABLE
demo=# INSERT INTO pilot_hobbies VALUES ( 'Ivan', '{ "sports": [ "футбол", "плавание" ], "home_lib": true, "trips": 3}':
:jsonb),
demo-# ( 'Petr', '{ "sports": [ "теннис", "плавание" ], "home_lib": true, "trips": 2}'::jsonb),
demo-# ( 'Pavel', '{ "sports": [ "плавание" ], "home_lib": false, "trips": 4}'::jsonb),
demo-# ( 'Boris', '{ "sports": [ "футбол", "плавание", "теннис" ], "home_lib": true, "trips": 0}'::jsonb);
INSERT 0 4
demo=# UPDATE pilot_hobbies SET hobbies = jsonb_set( hobbies, '{ home_lib }', 'true' ) WHERE pilot_name = 'Pavel';
UPDATE 1
demo=# SELECT pilot_name, hobbies->'home_lib' AS home_lib FROM pilot_hobbies;
 pilot_name | home_lib
------------+----------
 Ivan       | true
 Petr       | true
 Boris      | true
 Pavel      | true
(4 строки)
```

Задание 35

```
demo=# SELECT '{ "sports": "хоккей" }'::jsonb || '{ "trips": 5 }'::jsonb;
             ?column?
-----------------------------------
 {"trips": 5, "sports": "хоккей"}
(1 строка)


demo=# SELECT '{"a": {"b":"foo"}}'::json->'a';
  ?column?
-------------
 {"b":"foo"}
(1 строка)
```

Вывод

В ходе выполнения лабораторной работы были изучены типы данных PostgreSQL и основные операции с таблицами.

Список использованной литературы

1. Моргунов, Е. П. PostgreSQL. Основы языка SQL: учеб. пособие [Текст] : учебное пособие / Е.П. Моргунов ; — СПб. : Издательство «БХВ-Петербург», 2018. — 336 с.