Курс
PostgreSQL разработка

5 дней

Введение в PostgreSQL

- Последняя версия 15
- Наш стенд на версии 13

Конфигурация стенда вм с Postgresql

- ОС – Debian 11
- XFCE
- Postgresql 13.8
- psql – командная строка
- PGAdmin 4 – графический клиент

Общая конфигурация пользователей

- Пользователь student - обычный пользователь в ОС с паролем и правом входа и суперпользователь в субд.
- пароль student - и в ОС в СУБД одинаковый. Для простоты подключения, т.к. мы сейчас не занимаемся вопросами безопасности
- Суперпользователь СУБД postgres – изначально без пароля ОС и входа в систему, задали ему пароль в ОС, он может войти в систему, но домашнего каталога у него нет
  - postgres
    - пароль в ОС – postgres, пароль в субд - postgres
    - Пароль в субд сделан для подключения pgadmin. Работать от пользователя postgres в ОС не требуется.

Подключение пользователем postgres

- Входим в ОС как student
- Открываем терминал
  - student$ sudo su – postgres
    - Или
  - sudo -i -u postgres
    - Будет запрошен пароль для student
- Эти два варианта подключения переводят нас в пользователя postgres (меняется приглашение командной строки ОС)
- psql – мы вошли пользователем postgres

Подключение пользователем postgres

- sudo -u postgres psql
  - без переключения на пользователя postgres (остаемся в командной строке student, но подключаемся к субд как postgres)
- su - postgres
  - Переключение командной строки на пользователя Postgres, требует ввода пароля для postgres (если пароль не задан, то так войти нельзя)

Подключение PGAdmin 4

- Оконное приложение
- При запуске
  - Пароль на связку ключей (keyring) - Pa$$w0rd12
  - Master пароль password
  - Пароль для postgres одноименный: postgres
- Мы в основном будем пользоваться командной строкой psql

Демо 1

- Подключение psql
- Подключение PGAdmin 4
- Основные действия в VirtualBox

Система типов PostgreSQL

- Целочисленные типы
- Числа с плавающей запятой
- Числа с плавающей запятой заданной точности
- Монетарный тип
- Символьные типы
- Бинарный тип
- Типы даты/времени
- Булевский тип
- Тип строки битов
- Тип UUID
- Перечисления
- Составной тип (структура)
- Массивы
- Другие типы
- Приведение типов
- Последовательности

Целочисленные типы

Name	Storage Size	Min	Max
SMALLINT	2 bytes	-32,768	+32,767
INTEGER	4 bytes	-2,147,483,648	+2,147,483,647
BIGINT	8 bytes	-9,223,372,036,854,775,808	+9,223,372,036,854,775,807

Целочисленные типы

```
CREATE TABLE cities (
city_id serial PRIMARY KEY,
city_name VARCHAR (255) NOT NULL,
population INT NOT NULL CHECK (population >= 0));
```

Numeric и decimal

Числа фиксированной точности представлены двумя типами — numeric и decimal. Они одинаковы по своим возможностям.

Для задания значения этого типа используются два базовых понятия: масштаб (scale) и точность (precision).

Масштаб показывает число значащих цифр, стоящих справа от десятичной точки (запятой). Точность указывает общее число цифр как до десятичной точки, так и после нее.

Например - 10.3321

точность составляет 6 цифр, а масштаб — 4 цифры.

Параметры этого типа данных указываются в круглых скобках после имени типа:

numeric(точность, масштаб). Например, numeric(6, 4).

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.

Numeric и decimal

- Округления производятся стандартно по правилам математики
- Ввод значения с превышающей точностью вызовет ошибку

Real и double precisiom

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.

Поддерживают спецзначения Infinity -Infinity и NaN

Infinity -Infinity и NaN

Они представляют особые значения, описанные в IEEE 754, соответственно «бесконечность», «минус бесконечность» и «не число».

Записывая эти значения в виде констант в команде SQL, их нужно заключать в апострофы, например так: UPDATE table SET x = '-Infinity'. Регистр символов в этих строках не важен. В качестве альтернативы значения бесконечности могут быть записаны как inf и -inf.

Значения бесконечности соответствуют ожиданиям с точки зрения математики. Например, Infinity плюс любое конечное значение равно Infinity, как и Infinity плюс Infinity; но Infinity минус Infinity даёт NaN (не число), потому что в результате получается неопределённость. Обратите внимание, что бесконечность может быть сохранена только в столбце типа «неограниченный numeric», потому что она теоретически превышает любой конечный предел точности.

В большинстве реализаций «не число» (NaN) считается не равным любому другому значению (в том числе и самому NaN).

Чтобы значения numeric можно было сортировать и использовать в древовидных индексах, PostgreSQL считает, что значения NaN равны друг другу и при этом больше любых числовых значений (не NaN).

Float – стандарт ANSI SQL

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.

Денежные типы

Имя	Размер	Описание	Диапазон
money	8 байт	денежная сумма	-922337203685477 58.08 .. +922337203685477 58.07

Money

Тип money хранит денежную сумму с фиксированной дробной частью; определяется на уровне базы данных параметром lc_monetary.

Входные данные могут быть записаны по-разному, в том числе в виде целых и дробных чисел, а также в виде строки в денежном формате, например '$1,000.00’.

Выводятся эти значения обычно в денежном формате, зависящем от региональных стандартов.

Строковые (символьные) типы

Character Types	Description
CHARACTER VARYING(n), VARCHAR(n)	variable-length with length limit
CHARACTER(n), CHAR(n)	fixed-length, blank padded
TEXT, VARCHAR	variable unlimited length

Особенности строковых типов

- char(n) и varchar(n) дают ошибку при вводе строки длиной больше n
- Исключение – если в конце строки только пробелы, строка урезается до размера n, отбрасывая лишние пробелы
- Если строка явно преобразуется в char(n) и varchar(n), сервер также отбросит лишние символы в конце строки перед добавлением записи в таблицу
- Text и varchar без указания n – одно и то же, обслуживается как text без ограничения размера строки

Особенности задания строкового типа

```
-- одинарные кавычки
SELECT 'PostgreSQL';

-- удвоение кавычки для спецсимвола

SELECT 'PGDAY''17';

-- $$

SELECT $$PGDAY'17$$;

-- стиль с - символ E

SELECT E'PGDAY\'17';
```

Бинарный тип

Имя	Размер	Описание
bytea	1 или 4 байта плюс сама двоичная строка	двоичная строка переменной длины

Бинарный тип - особенности

- Двоичные строки представляют собой последовательность октетов (байт) и имеют два отличия от текстовых строк. Во-первых, в двоичных строках можно хранить байты с кодом 0 и другими «непечатаемыми» значениями (обычно это значения вне десятичного диапазона 32..126). В текстовых строках нельзя сохранять нулевые байты, а также значения и последовательности значений, не соответствующие выбранной кодировке базы данных. Во-вторых, в операциях с двоичными строками обрабатываются байты в чистом виде, тогда как текстовые строки обрабатываются в зависимости от языковых стандартов. То есть, двоичные строки больше подходят для данных, которые программист видит как «просто байты», а символьные строки — для хранения текста.

- Тип bytea поддерживает два формата ввода и вывода: «шестнадцатеричный» и традиционный для PostgreSQL формат «спецпоследовательностей». Входные данные принимаются в обоих форматах, а формат выходных данных зависит от параметра конфигурации bytea_output; по умолчанию выбран шестнадцатеричный. (Заметьте, что шестнадцатеричный формат был введён в PostgreSQL 9.0; в ранних версиях и некоторых программах он не будет работать.)
- https://postgrespro.ru/docs/postgrespro/10/datatype-binary

Типы даты-времени

Имя	Размер	Описание	Наименьшее значение	Наибольшее значение	Точность
timestamp [(*p*)] [without time zone]	8 байт	дата и время (без часового пояса)	4713 до н. э.	294276 н. э.	1 микросекунда
timestamp [(*p*)] with time zone	8 байт	дата и время (с часовым поясом)	4713 до н. э.	294276 н. э.	1 микросекунда
date	4 байта	дата (без времени суток)	4713 до н. э.	5874897 н. э.	1 день
time [(*p*)] [without time zone]	8 байт	время суток (без даты)	00:00:00	24:00:00	1 микросекунда
time [(*p*)] with time zone	12 байт	время дня (без даты), с часовым поясом	00:00:00+1559	24:00:00-1559	1 микросекунда
interval [*поля*] [(*p*)]	16 байт	временной интервал	-178000000 лет	178000000 лет	1 микросекунда

Date

Занимает 4 байта
Диапазон дат 4713 до н.э. - 5874897 н.э.

Для хранения используется формат
yyyy-mm-dd 2000-12-31

Для ввода данных можно использовать как его, так и формат локали

Приведение к типу date

```
SELECT '2016-09-12'::date;

SELECT 'Sep 12, 2016'::date;
```

Функции для типа date

```
--получение типа date
SELECT NOW()::date;
SELECT CURRENT_DATE;

--Форматирование вывода даты, возвращает тип text
SELECT TO_CHAR(NOW() :: DATE, 'dd/mm/yyyy’);
SELECT TO_CHAR(NOW() :: DATE, 'Mon dd, yyyy');
```

Вычисление интервала дат

minus (-) operator

```
SELECT    first_name,    last_name,     now() - hire_date as diff
FROM      employees;
```

AGE() function

```
SELECT    employee_id,first_name,    last_name,     AGE(birth_date)
FROM      employees;
```

EXTRACT() function

```
SELECT     employee_id,       first_name,    last_name,     EXTRACT (YEAR FROM birth_date) AS YEAR,
EXTRACT (MONTH FROM birth_date) AS MONTH, EXTRACT (DAY FROM birth_date) AS DAY
FROM      employees;
```

Функции преобразования типов

to_char (timestamp, text) → text
to_char (timestamp with time zone, text) → text
Converts time stamp to string according to the given format.
to_char(timestamp '2002-04-20 17:31:12.66', 'HH12:MI:SS') → 05:31:12

to_char (interval, text) → text
Converts interval to string according to the given format.
to_char(interval '15h 2m 12s', 'HH24:MI:SS') → 15:02:12

to_char (*numeric_type*, text) → text
Converts number to string according to the given format; available for integer, bigint, numeric, real, double precision.
to_char(125, '999') → 125
to_char(125.8::real, '999D9') → 125.8
to_char(-125.8, '999D99S') → 125.80-

Функции преобразования типов

to_date (text, text) → date
Converts string to date according to the given format.
to_date('05 Dec 2000', 'DD Mon YYYY') → 2000-12-05

to_number (text, text) → numeric
Converts string to numeric according to the given format.
to_number('12,454.8-', '99G999D9S') → -12454.8

to_timestamp (text, text) → timestamp with time zone
Converts string to time stamp according to the given format. (See also to_timestamp(double precision) in Table 9.33.)
to_timestamp('05 Dec 2000', 'DD Mon YYYY') → 2000-12-05 00:00:00-05

Age()

AGE(timestamp,timestamp);

Вычитает второй аргумент из первого, возвращает тип interval

AGE(timestamp);

Первый аргумент current_date

Time

column_name TIME(precision); -- задание типа в таблице

Precision до 6 цифр

Точность времени

HH:MI HH:MI:SS HHMISS

01:02 01:02:03 010203

MI:SS.pppppp HH:MI:SS.pppppp HHMISS.pppppp
04:59.999999 04:05:06.777777 040506.777777

TIME with time zone type

column TIME with time zone – определение типа

TIME with time zone использует 12 байт, диапазон от 00:00:00+1459 до 24:00:00-1459

Пример типа
04:05:06 PST 04:05:06.789-8

PST -08:00 Pacific Standard Time
EET +02:00 Eastern Europe, USSR Zone 1
GMT +00:00 Greenwich Mean Time
UT +00:00 Universal Time
UTC +00:00 Universal Time, Coordinated

Функции для time

```
SELECT CURRENT_TIME;  --возвращает тип time with time zone
SELECT CURRENT_TIME(5);  -- задаем точность

SELECT LOCALTIME; --возвращает тип time
 SELECT LOCALTIME(0);

SELECT LOCALTIME AT TIME ZONE 'UTC-7’; -- возвращает тип time with time zone

SELECT
    LOCALTIME,
    EXTRACT (HOUR FROM LOCALTIME) as hour;
```

Как посмотреть возвращаемый тип

- SELECT LOCALTIME;
- SELECT pg_typeof(LOCALTIME);

- https://www.postgresql.org/docs/15/functions-datetime.html

Тип interval

interval data type - периоды years, months, days, hours, minutes, seconds

16 bytes storage size диапазон -178,000,000 years до 178,000,000 years

```
SELECT    now(),     now() - INTERVAL '1 year 3 hours 20 minutes'
AS "3 hours 20 minutes ago of last year";
```

Timestamp

Временная метка – дата + время +(часовой пояс)

timestamp: a timestamp without timezone one.

timestamptz: timestamp with a timezone

timestamptz хранится в формате UTC

Внутренний формат хранения – в микросекундах от начала 2000 года

При вставке timestamptz, PostgreSQL конвертирует timestamptz в UTC и UTC хранит в таблице.

При запросе к timestamptz из базы, PostgreSQL конвертирует UTC в time value of the timezone используя настройки сервера, юзера, или текущего коннекта

Используют 8 байт для хранения каждого из этих типов

Функции для timestamp

```
--timestamp with time zone
SELECT NOW();
SELECT CURRENT_TIMESTAMP;

SELECT CURRENT_TIME;  --time with time zone

SELECT TIMEOFDAY(); --возвращает text
SHOW TIMEZONE;
```

At time zone

```
SET TIME ZONE 'UTC';
SELECT '2020-01-01 00:00:00+00'::timestamptz AT TIME ZONE
‘Europe/Moscow’;
```

```
SET TIME ZONE 'UTC';
SELECT '2020-01-01 00:00:00+00'::timestamp AT TIME ZONE
‘Europe/Moscow’;
```

Сравним результаты. Почему они такие?

Фокусы стандарта POSIX

```
SELECT
  '2020-01-01 00:00:00+00' AT TIME ZONE 'Etc/GMT+3’,
'2020-01-01 00:00:00+00' AT TIME ZONE 'Etc/GMT-3',
  '2020-01-01 00:00:00+00' AT TIME ZONE ‘Europe/Moscow’;
```

https://kaiwern.com/posts/2021/07/20/what-you-need-to-know-about-postgresql-timezone/

Летнее время и «так исторически сложилось…»

```
SET TimeZone = 'UTC';

SELECT TIMESTAMPTZ '2022-04-01 12:00:00 Europe/Vienna';

      timestamptz
═══════════════════════
 2022-04-01 10:00:00+00
(1 row)

SELECT TIMESTAMPTZ '2022-03-01 12:00:00 Europe/Vienna';

      timestamptz
═══════════════════════
 2022-03-01 11:00:00+00
(1 row)

SELECT TIMESTAMPTZ '1850-02-01 12:00:00 Europe/Vienna';

      timestamptz
═══════════════════════
 1850-02-01 10:54:39+00
(1 row)
```

Булевский тип

- Тройная, а не двойная логика в языках стандарта ANSI SQL
- TRUE, FALSE, NULL

Синтаксис

True	False
true	false
‘t’	‘f ‘
‘true’	‘false’
‘y’	‘n’
‘yes’	‘no’
‘1’	‘0’

Тип UUID

Тип данных uuid сохраняет универсальные уникальные идентификаторы (Universally Unique Identifiers, UUID), определённые в RFC 4122, ISO/IEC 9834-8:2005 и связанных стандартах. (В некоторых системах это называется GUID, глобальным уникальным идентификатором.) Этот идентификатор представляет собой 128-битное значение, генерируемое специальным алгоритмом, практически гарантирующим, что этим же алгоритмом оно не будет получено больше нигде в мире. Таким образом, эти идентификаторы будут уникальными и в распределённых системах, а не только в единственной базе данных, как значения генераторов последовательностей.

UUID записывается в виде последовательности шестнадцатеричных цифр в нижнем регистре, разделённых знаками минуса на несколько групп, в таком порядке: группа из 8 цифр, за ней три группы из 4 цифр и, наконец, группа из 12 цифр, что в сумме составляет 32 цифры и представляет 128 бит. Пример UUID в этом стандартном виде:

a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11

Сгенерировать UUID можно с помощью дополнительного модуля uuid-ossp, в котором реализованы несколько стандартных алгоритмов, а можно воспользоваться модулем pgcrypto, где тоже есть функция генерирования случайных UUID

Тип перечисления

Типы перечислений (enum) определяют статический упорядоченный набор значений, так же как и типы enum, существующие в ряде языков программирования. В качестве перечисления можно привести дни недели или набор состояний.

```
CREATE TYPE mood AS ENUM ('sad', 'ok', 'happy');
```

Составной тип - структура

Составной тип представляет структуру табличной строки или записи; по сути это просто список имён полей и соответствующих типов данных.

```
CREATE TYPE inventory_item AS (
    name            text,
    supplier_id     integer,
    price           numeric
);

CREATE TABLE on_hand (
    item      inventory_item,
    count     integer
);

INSERT INTO on_hand VALUES (ROW('fuzzy dice', 42, 1.99), 1000);
```

Массивы - array

PostgreSQL позволяет создавать в таблицах 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 пользовательских типов

integer[] – объявление одномерного целочисленного массива

Число элементов массива задавать необязательно

Псевдотип serial - счетчик

CREATE TABLE table_name(id SERIAL);

В фоне создается sequence и default для колонки, использующий этот объект sequence для генерации значений счетчика

Добавляется NOT NULL

Владельцем sequence становится сама колонка. Поэтому, когда колонка удалится, sequence тоже автоматически удалится

Генератор счетчика позволяет разрывы в нумерации, т.к. при отмене транзакции выделенный номер теряется

Закадровые действия для serial

```
CREATE SEQUENCE table_name_id_seq;

CREATE TABLE table_name (
    id integer NOT NULL DEFAULT nextval('table_name_id_seq')
);

ALTER SEQUENCE table_name_id_seq
OWNED BY table_name.id;
```

Три псевдотипа для serial

Name	Storage Size	Range
SMALLSERIAL	2 bytes	1 to 32,767
SERIAL	4 bytes	1 to 2,147,483,647
BIGSERIAL	8 bytes	1 to 9,223,372,036,854,775,807

Serial – особенности реализации

Start Value	Нет	Always 1	ALTER SEQUENCE RESTART WITH to change
Increment	Нет	Always 1	ALTER SEQUENCE INCREMENT BY to change
How to Generate IDs	Omit the SERIAL column in INSERT, or specify DEFAULT keyword		
Explicit ID Insert	Да		
Multiple SERIAL per Table	Да		
Constraints	NOT NULL	Added automatically	
	Primary or unique key	*Not* required and not added automatically	

Serial – особенности реализации

Retrieve Last ID	LASTVAL()	Returns the last ID inserted in the *current* session		
	CURRVAL('seq_name')	Returns the current ID for the specified sequence		
	INSERT … RETURNING *serialcol* returns ID immediately after INSERT statement			
Gaps	If a value is *explicitly* inserted, this has *no* effect on sequence generator			
Restart (Reset)	ALTER SEQUENCE *tablename_serialcol*_seq RESTART WITH *new_current_id*;			
Alternatives	BIGSERIAL	64-bit ID numbers		
	Using a sequence and *DEFAULT NEXTVAL('seq_name')*			
	OID system column			
Synonym	SERIAL4			

Как изменить seed и increment?

```
CREATE TABLE teams2
 (
   id SERIAL UNIQUE,
   name VARCHAR(90)
 );

 -- Modify initial value and increment
 ALTER SEQUENCE teams2_id_seq RESTART WITH 3 INCREMENT BY 3;

 -- Insert data
 INSERT INTO teams2 (name) VALUES ('Crystal Palace');
 INSERT INTO teams2 (name) VALUES ('Leeds United');
```

Тип json

- Поддерживается с версии 9.2

Операторы

- operator -> возращает JSON object field как ключ
- operator ->> возвращает JSON object field как текст

Функции

- json_each и json_each_text
- json_object_keys
- json_typeof

- https://www.postgresql.org/docs/current/functions-json.html

Тип xml

```
CREATE TABLE test (
...,
data xml,
...);

INSERT INTO test VALUES (... , '<foo>...</foo>', ...);
SELECT data FROM test;
```

- Поддержка типа
- Функции

Создаем xml

- Выражение xmlelement создаёт XML-элемент с заданным именем, атрибутами и содержимым.

```
SELECT xmlelement(name foo);

xmlelement
------------
 <foo/>

SELECT xmlelement(name foo, xmlattributes('xyz' as bar));

    xmlelement
------------------
 <foo bar="xyz"/>
```

Экспорт xml

Data export:

- `table_to_xml`
- `query_to_xml`
- `cursor_to_xml`

Schema export:

- `table_to_xmlschema`
- `query_to_xmlschema`
- `cursor_to_xmlschema`

XPath

- Всегда возвращает XML массив! Даже в случае одного элемента
- Для получения скаляра надо взять нужный элемент массива, преобразовать в требуемый тип
- *Специфичные для PostgreSQL функции xpath() и xpath_exists() выполняют запросы к XML-документам на языке XPath. В PostgreSQL также имеются поддерживающие только XPath стандартные функции XMLEXISTS и XMLTABLE, хотя согласно стандарту они должны поддерживать XQuery. Все эти функции в PostgreSQL реализованы с использованием библиотеки libxml2, которая поддерживает только XPath 1.0.*

XMLTABLE

- PASSING – определяет xml колонку в таблице откуда берем данные для преобразования в реляционную таблицу
- '/books/book’ перед PASSING стоит генератор строки, опреляющий что будет строкой
- COLUMNS определяет колонки через PATH

XMLTABLE

```
SELECT xmltable.*
  FROM hoteldata,
       XMLTABLE ('/hotels/hotel/rooms/room' PASSING hotels
                 COLUMNS
                   id FOR ORDINALITY,
                   hotel_name text PATH '../../name' NOT NULL,
                   room_id int PATH '@id' NOT NULL,
                   capacity int,
                   comment text PATH 'comment' DEFAULT 'A regular room'
                );
```

Другие типы данных

- Геометрия
- Сетевые адреса
- Диапазоны
- Pg_lsn
- Псевдотипы

https://postgrespro.ru/docs/postgrespro/14/datatype
https://postgrespro.ru/docs/postgresql/15/datatype

Приведение типов

Приведение типа определяет преобразование данных из одного типа в другой. PostgreSQL воспринимает две равносильные записи приведения типов:

CAST (выражение AS тип)

выражение::тип

Запись с CAST соответствует стандарту SQL, тогда как вариант с :: — историческое наследие PostgreSQL.

Приведение типов при вызове функции

Также можно записать приведение типа как вызов функции:

имя_типа (выражение)

Однако это будет работать только для типов, имена которых являются также допустимыми именами функций. Например, double precision так использовать нельзя, а float8 (альтернативное название того же типа) — можно. Кроме того, имена типов interval, time и timestamp из-за синтаксического конфликта можно использовать в такой записи только в кавычках. Таким образом, запись приведения типа в виде вызова функции провоцирует несоответствия и, возможно, лучше будет её не применять.

Написание запросов на языке SQL

Язык SQL

- История названия и произношение
- Эскуэль или Сиквел?
- Как SEQL стал SQL? – промышленный шпионаж эпохи первичной разработки движков БД))
- SQL и стандарт ANSI

ANSI SQL

Дата	Версия	Описание (комментарии)
1986	SQL-86(SQL-87)	Первая версия, стандартизованная ANSI.
1989	SQL-89	Эта версия включает незначительные изменения, которые добавляют ограничения целостности.
1992	SQL-92(SQL2)	Эта версия включает в себя основные изменения в языке SQL. Многие системы баз данных используют этот стандарт для своего языка спецификации.
1999	SQL:1999(SQL3)	В этой версии представлено много новых функций, таких как сопоставление регулярных выражений, триггеры, некоторые объектно-ориентированные функции и возможности OLAP. SQL:1999 также добавляет BOOLEAN тип данных, но многие коммерческие серверы баз данных не поддерживают его как тип столбца. Между тем SQL:1999 считается устаревшим.
2003	SQL:2003	•Эта версия включает незначительные изменения ко всем частям SQL:1999. Эта версия также добавляет новые функции, такие как: Оконные функции, которые очень полезные для анализа данных. •Столбцы с автоматической генерацией значений и идентификацией.
2006	SQL:2006	Эта версия определяет способы импорта, хранения и управления данными XML. Кроме того, он позволяет приложениям использовать XQuery для запроса данных в формате XML.
2008	SQL:2008	Эта версия добавляет такие функции, как оператор TRUNCATE TABLE, предложение FETCH и триггеры INSTEAD OF.
2011	SQL:2011	Эта версия добавляет улучшения для оконных функций и предложения FETCH.
2016	SQL:2016	Эта версия добавляет различные функции для работы с данными JSON.
2019	SQL:2019	Эта версия указывает тип данных многомерных массивов (MDarray).[1]

Общепринятая терминология языка SQL

- DQL
- DML
- DDL
- DCL

DQL и DML

- Команды на работу с данными в таблицах
  - DQL – SELECT
  - DML – INSERT, UPDATE, DELETE
- Часто DML подразумевает все четыре вида команд, включая SELECT (зависит от терминологии конкретного движка)

DCL

- Административные команды для раздачи прав доступа к объектам БД

  - GRANT
  - DENY
  - REVOKE

DDL

- Команды на создание, модификацию и удаление объектов БД

  - CREATE
  - ALTER
  - DROP

Кластер

- При инициализации кластера создается 3 базы данных
- postgres
- template0
- template1

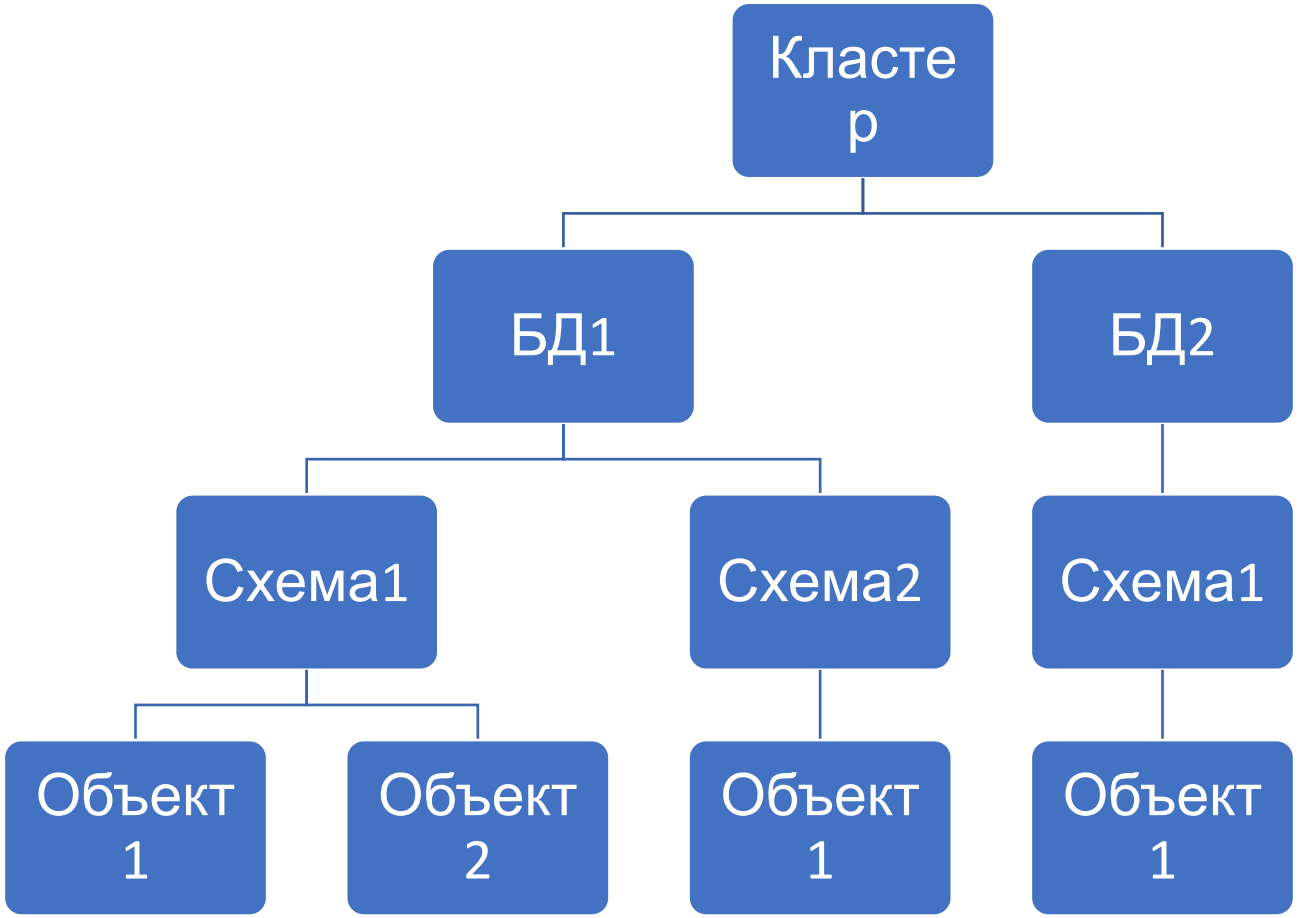
Схемы

- Пространство имен для объектов базы данных
  - tables, views, indexes, data types, functions, stored procedures и operators
- Каждый объект обязательно находится в какой-то одной схеме
- Находиться одновременно в нескольких схемах невозможно
- В БД можно работать с объектами любой схемы, при наличии прав доступа
- Есть системные (стандартные) схемы и пользовательские

lowercase для имен

- Имена таблиц и колонок хранятся в нижнем регистре!
- Для различия регистров case-sensitive нужно использовать двойные кавычки
- DESCRIPTION, description и “description” одно и то же для Postgres, а вот “Description” это другое имя
- Рекомендация – в именах использовать только нижний регистр, можно разделять слова подчеркиваниями, например my_table

Логическая структура кластера



Путь поиска

- Объекты с одинаковыми именами могут находиться в разных схемах
- Обращение к объекту по полному имени схема.объект точно указывает на объект
- Обращение только по имени заставляет сервер выполнять поиск схемы
- Для этого используется параметр search_path

search_path

```
SELECT current_schema();
SELECT current_schemas(true);
SHOW search_path;

CREATE SCHEMA sales;
```

Специальные схемы

public
по умолчанию входит в путь поиска
если ничего не менять, все объекты будут в этой схеме

$user
Схема, совпадающая по имени с пользователем
по умолчанию входит в путь поиска, но не существует
если создать, объекты пользователя будут в этой схеме

pg_catalog
схема для объектов системного каталога
если pg_catalog нет в пути, она неявно подразумевается первой

Локализация

В операционной системе сервера должны быть установлены локали с поддержкой русского языка:

```
student$ locale -a|grep ru
```

Важные переменные локалей

LC_ALL - если установлена, то используется для всех категорий локалей, даже если они заданы;

LANG - используется для тех категорий локалей, для которых значение не задано;

LANGUAGE - если установлена, то используется вместо LC_MESSAGES.

Локали Postgresql

```
SELECT name, setting, context FROM pg_settings WHERE name LIKE 'lc%';
    name     |  setting   |  context
-------------+------------+-----------
 lc_collate  | ru_RU.utf8 | internal
 lc_ctype    | ru_RU.utf8 | internal
 lc_messages | en_US.utf8 | superuser
 lc_monetary | ru_RU.utf8 | user
 lc_numeric  | ru_RU.utf8 | user
 lc_time     | ru_RU.utf8 | user
(6 rows)
```

параметры lc_monetary, lc_numeric и lc_time может изменить пользователь, а lc_messages только суперпользователь. lc_ctype и lc_collate изменить невозможно.

Локализация на клиенте

```
SET client_encoding = 'UTF8’;

SET lc_time = 'ru_RU.UTF8’;
SELECT to_char(current_date, 'TMDay, DD TMMonth YYYY’); --префикс TM

SET lc_numeric = 'ru_RU.UTF8’;
SELECT to_char('12345'::numeric, '999G999D00’);

SET lc_monetary = 'ru_RU.UTF8’;
SELECT '12345'::money;
```

Из чего состоит команда SELECT?

Clause	Expression
SELECT	<select list>
FROM	<table or view>
WHERE	<search condition>
GROUP BY	<group by list>
ORDER BY	<order by list>

Select list

```
SELECT first_name FROM customer;
SELECT   first_name,   last_name,   email FROM   customer;
SELECT * FROM customer;
```

FROM

- Указывает на источник данных (таблицу, любой объект возвращающий таблицу – вью, функция, вложенный запрос)
- Источником данных может быть комбинация таблиц (join)

JOIN

- Соединение двух и более таблиц с целью получения новой структуры выборки и отбора записей в нее по критериям, заданным в операторе join
- UNION соединяет выборки «вертикально», JOIN – «горизонтально»
- Необходимость JOIN есть следствие принципов нормализации реляционных баз
- Нормализация приводит к тому, что данные об одном объекте могут лежать в разных таблицах, и JOIN дает возможность выполнить обратную операцию – собрать их из разных таблиц в одну выборку

Виды JOIN

Вид JOIN	Описание
Cross	Произведение множеств (все со всеми – Картезианский продукт)
Inner	В результат берутся только те записи, которые удовлетворяют условию соединения с обеих сторон
Outer	В результат берутся все записи с одной стороны и только удовлетворяющие условию соединения с другой

Cross Join

```
SELECT * FROM T1 CROSS JOIN  T2;
```

Картезианское произведение множест – все со всеми

Inner Join

```
SELECT c.customer_id, first_name, last_name, email, amount,
payment_date
FROM customer c
INNER JOIN payment p
ON p.customer_id = c.customer_id
WHERE c.customer_id = 2;
```

Inner join 3 таблицы

```
SELECT    c.customer_id,  c.first_name customer_first_name,
c.last_name customer_last_name, s.first_name staff_first_name,
s.last_name staff_last_name,    amount,  payment_date
FROM customer c
INNER JOIN payment p
ON p.customer_id = c.customer_id
INNER JOIN staff s
ON p.staff_id = s.staff_id
ORDER BY payment_date;
```

Предложение USING в join

Если обе таблицы имеют одноименную колонку

```
SELECT customer_id, first_name, last_name, amount, payment_date
FROM customer
INNER JOIN payment
 USING(customer_id)
ORDER BY payment_date;
```

Варианты запросов

```
SELECT    film.film_id,   title,       inventory_id
FROM      film
LEFT JOIN inventory
ON inventory.film_id = film.film_id
ORDER BY title;

-- записи с отсутствующими значениями полей
SELECT    film.film_id,   film.title,inventory_id
FROM      film
LEFT JOIN inventory
ON inventory.film_id = film.film_id
WHERE inventory.film_id IS NULL
ORDER BY title;
```

Outer join – left и right

```
SELECT    review,    title
FROM    films
RIGHT JOIN film_reviews
ON film_reviews.film_id = films.film_id;
```

```
SELECT review, title
FROM films
RIGHT JOIN film_reviews
USING (film_id);
```

Natural Join

Имплицитно соединяет таблицы по одноименным колонкам

Может давать неожиданные результаты

Лучше не использовать

Замена - USING

Natural Join

```
--natural
SELECT * FROM products NATURAL JOIN categories;


--using
SELECT * FROM products
INNER JOIN categories
USING (category_id);
```

Псевдонимы для таблиц

Для удобства написания join, но необязательны

Для ряда других вычислительных операций могут быть обязательны, например self-join

```
SELECT    e.first_name employee,    m .first_name manager
FROM    employee e
INNER JOIN employee m
ON m.employee_id = e.manager_id
ORDER BY manager;
```

Псевдонимы для колонок

```
SELECT column_name AS alias_name
FROM table_name;
```

```
SELECT column_name alias_name
FROM table_name;
```

Необходимы для создания заголовка вычисляемых колонок (может быть требованием различных клиентских API)

Подзапросы

- Запрос внутри запроса
- Вложенные
- Lateral

Вложенный подзапрос - скаляр

```
SELECT  film_id,  title,  rental_rate
FROM film
WHERE  rental_rate >
(      SELECT          AVG (rental_rate)      FROM        film   );
```

IN – список значений

```
SELECT film_id, title
FROM film WHERE film_id IN
( SELECT inventory.film_id FROM rental
INNER JOIN inventory ON inventory.inventory_id = rental.inventory_id
WHERE return_date BETWEEN '2005-05-29' AND
'2005-05-30');
```

EXISTS – выборка не пуста

```
SELECT    first_name,  last_name
FROM customer WHERE  EXISTS
(       SELECT       1       FROM       payment       WHERE
payment.customer_id = customer.customer_id  );
```

Lateral

При использовании Lateral подзапрос может ссылаться на поля внешнего запроса

```
SELECT * FROM foo, LATERAL (SELECT * FROM bar WHERE bar.id =
foo.bar_id) ss;
```

where – фильтрация данных в запросе

- Предложение (фильтр) WHERE использует ПРЕДИКАТЫ
- Предикат – логическое выражение
- Логическое выражение в языках SQL может принимать ТРИ значения: TRUE, FALSE, NULL (UNKNOWN)
- WHERE отбирает в результат выборки записи со значением TRUE
- Записи с результатами FALSE или NULL(UNKNOWN) отбрасываются
- Фильтрация происходит на сервере, клиент получает только те записи, которые удовлетворяют предикату WHERE

where

```
SELECT    last_name,   first_name
FROM customer
WHERE    first_name = 'Jamie';
```

GROUP BY

- Создает группы в выборке на основании уникальных комбинаций значений в GROUP BY
- Вычисляет агрегат для каждой группы
- Детали после этого удаляются из выборки

Правила запроса с группировкой

- HAVING, SELECT и ORDER BY должны возвращать скалярное значение для каждой группы
- Все колонки в SELECT, HAVING и ORDER BY или должны быть перечислены в GROUP BY или быть входными значениями для агрегатных функций
- Эти правила запоминать необязательно – если вы неверно напишете запрос с группировкой, вы получите СИНТАКСИЧЕСКУЮ ОШИБКУ с конкретным указанием, что вы написали не так!

Примеры GROUP BY

```
SELECT    customer_id,    SUM (amount)
FROM payment
GROUP BY    customer_id;
```

```
SELECT    customer_id,    staff_id,  SUM(amount)
 FROM    payment
GROUP BY    staff_id,  customer_id
ORDER BY    customer_id;
```

Фильтрация результатов группировки

- HAVING
- Обрабатывается после GROUP BY
- Отбирает значения предикатов с результатом TRUE (аналогично WHERE
- Если в запросе есть и WHERE и HAVING, сначала идет фильтрация детальных записей по условию WHERE, затем происходит вычисление агрегатов для групп, а затем прикладывается фильтр HAVING к результатам групп

Пример HAVING

```
SELECT    customer_id,    SUM (amount)
FROM      payment
GROUP BY    customer_id
HAVING   SUM (amount) > 200;
```

Сортировка выборки

- Изначально выборка, таблица не сортирована
- Результат гарантированно отсортирован в нужном порядке только при применении предложения ORDER BY

Особенности применения ORDER BY

- Обрабатывается последним
- Все NULL считает одинаковыми
- Можно сортировать по любой колонке, даже если она не отображается и не входит в select list
- Порядок сортировки ASC (восходящий, по умолчанию, можно не указывать), DESC (нисходящий, должен быть указан явно)
- ASC и DESC относятся только к одной колонке рядом с которой они написаны (нельзя применить эти опции на несколько колонок сразу)

Примеры ORDER BY

```
SELECT    first_name,  last_name
FROM customer
ORDER BY    first_name;
```

```
SELECT    first_name,  last_name
FROM customer
ORDER BY    first_name DESC;
```

Order by и NULL

ORDER BY sort_expresssion [ASC | DESC] [NULLS FIRST | NULLS LAST]

Limit и Offset

Взять первые N записей выборки

```
SELECT film_id, title, release_year FROM film ORDER BY film_id
LIMIT 5;
```

Сколько пропустить записей

```
SELECT film_id, title, release_year FROM film ORDER BY film_id
LIMIT 4 OFFSET 3;
```

Union и UNION ALL

```
-- исключит дубли
SELECT * FROM top_rated_films
UNION
SELECT * FROM most_popular_films;

-вернет все записи
SELECT * FROM top_rated_films
UNION ALL
SELECT * FROM most_popular_films;
```

CASE выражение

- Разбор по веткам
- Это выражение, а не оператор

Тройная логика: NULL

- Что такое NULL и откуда он берется
- Сравнение двух NULL
- Проблема: неоднозначность результатов на одних и тех же данных
- IS NULL / IS NOT NULL

vs

= NULL / <> NULL

Пример запросов

```
SELECT custid, city, region, country
FROM customers
WHERE region IS NULL;
```

```
SELECT custid, city, region, country
FROM customers
WHERE region = NULL;
```

Представления View

- Объект базы данных
- Определяется одним селектом в теле объекта
- Не принимает параметры
- Может использоваться в запросах в предложении from вместо таблицы

View

CREATE VIEW view_name AS query;

Select * from view_name;

Материализованные представления

CREATE MATERIALIZED VIEW view_name AS query WITH [NO] DATA;

--копирует данные запроса как таблицу, по обслуживанию почти как таблица

REFRESH MATERIALIZED VIEW view_name;

--обновление только командой, автоматического нет

REFRESH MATERIALIZED VIEW CONCURRENTLY view_name;

--требует уникального индекса

DROP MATERIALIZED VIEW view_name;