

В. Б. Пикулев

Базы данных

2. Язык SQL на примере СУБД PostgreSQL

scilink.ru, 2021

Историческая справка

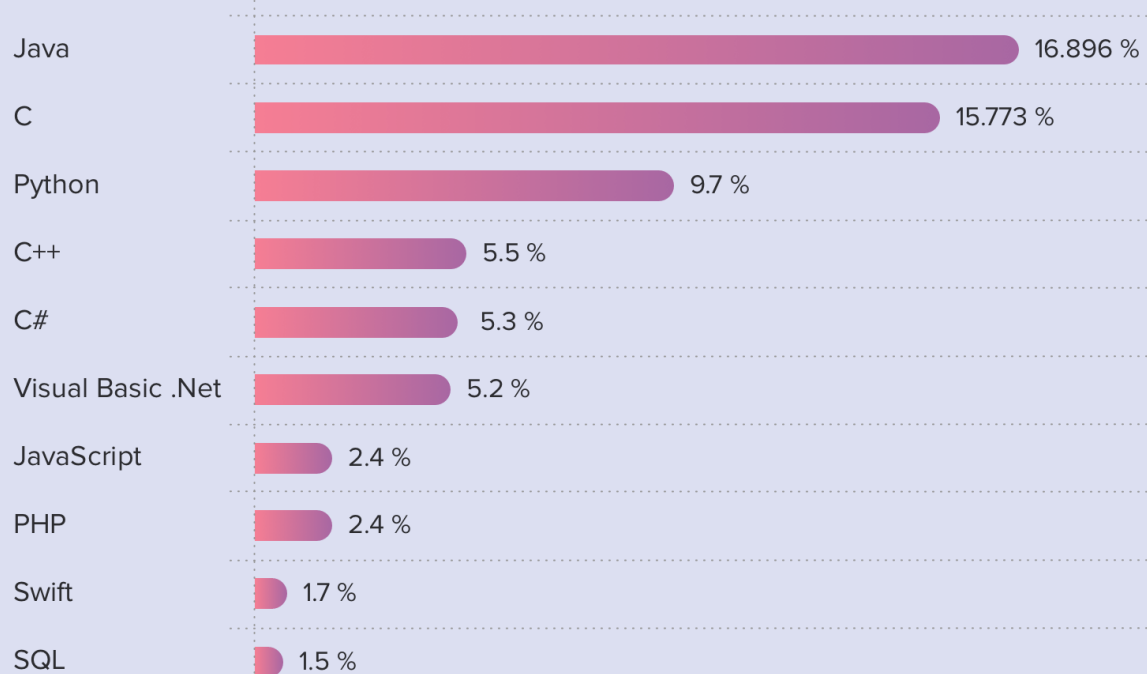


SQL (*Structured Query Language*, структурированный язык запросов) – стандарт языка для работы с данными в реляционных базах данных.

- ❖ Прототип языка – сначала QBE, затем SEQUEL (Structured English Query Language) – был разработан в начале 70-х годов в IBM Research и реализован в СУБД System R.
- ❖ 1989 – первый ANSI/ISO стандарт языка SQL (вторая редакция, первая была в 1987 г.). Однако развитие технологий БД потребовали его доработки и расширения.
- ❖ 1992 – стандарт SQL-92 или **SQL 2**. Практически все современные реляционные (и постреляционные) СУБД поддерживают этот стандарт полностью.
- ❖ 1999 – стандарт **SQL 3**. В стандарт введены структурированные типы данных и другие особенности, позволяющие сочетать реляционную и объектную модель данных.
- ❖ В настоящее время известно девять вариантов стандартов языка SQL ANSI/ISO, вплоть до SQL-2016 (введён JSON), причём каждый последующий включает в себя возможности предыдущего.

<https://www.cleveroad.com/blog/programming-languages-ranking>

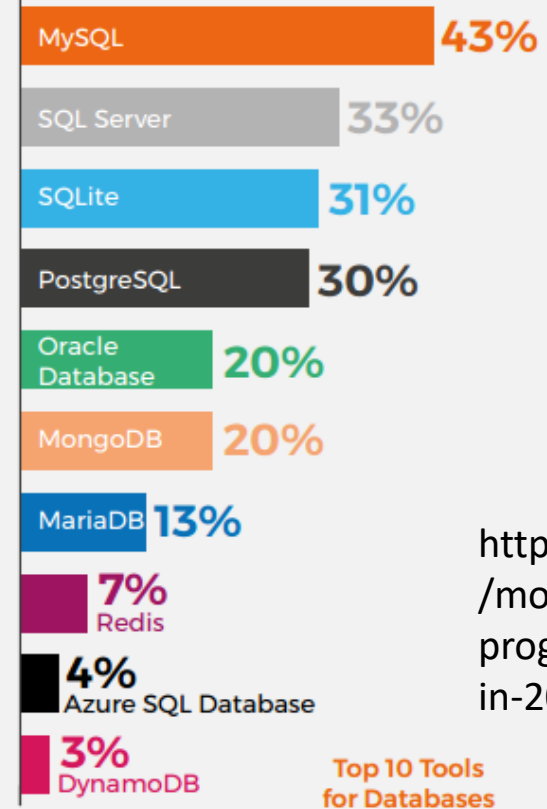
Top programming languages, TIOBE



Перспективы

> Top Databases

Database use is dominated by **SQL**-based choices, with **MySQL** still the most commonly utilized database solution.



<https://hub.packtpub.com/most-popular-programming-languages-in-2018/>

Особенности языка

SQL не является традиционным языком программирования: он не содержит операторы, позволяющие осуществлять пошаговые действия, а ориентирован на работу со множествами. Другими словами, на SQL пишется, *ЧТО* должно получиться в результате выполнения запроса, но не пишется, *КАК* это будет реализовано.

1. Data Definition Language, DDL:

- ❖ **CREATE** создаёт какую-либо структуру базы данных либо саму базу
- ❖ **ALTER** изменяет свойства этой структуры
- ❖ **DROP** удаляет структуру

2. Data Manipulation Language, DML:

- ❖ **SELECT** осуществляет выборку данных
- ❖ **INSERT** добавляет новые данные
- ❖ **UPDATE** изменяет существующие данные
- ❖ **DELETE** удаляет данные

1. Data Control Language, DCL:

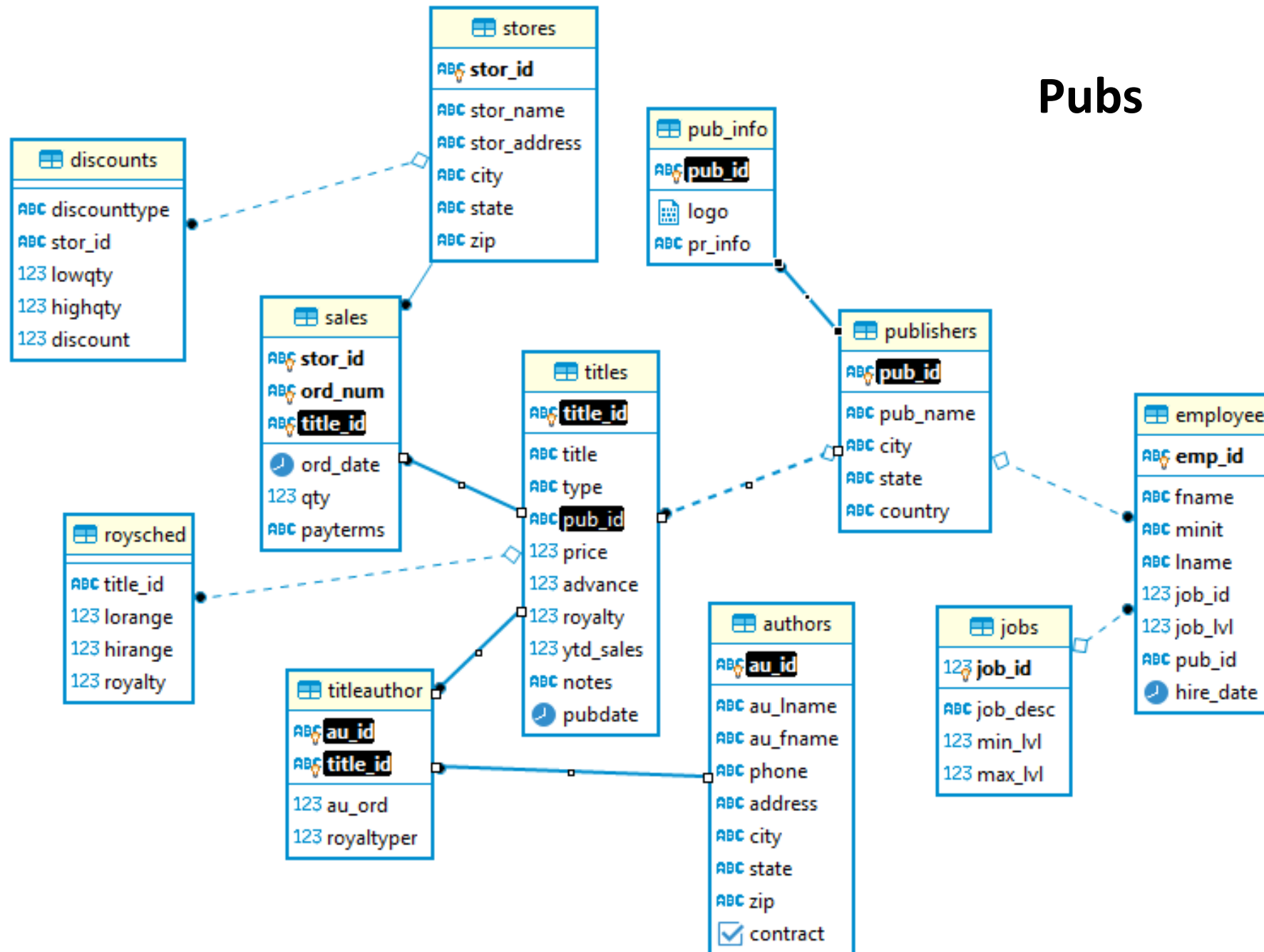
- ❖ **GRANT** предоставляет разрешения на определённые операции со структурами баз данных
- ❖ **REVOKE** задаёт "мягкий" запрет на работу со структурами баз данных
- ❖ **DENY** задаёт "жёсткий" запрет, имеющий приоритет над разрешением

2. Transaction Control Language, TCL:

- ❖ **COMMIT** применяет транзакцию
- ❖ **ROLLBACK** откатывает все изменения, сделанные в контексте текущей транзакции

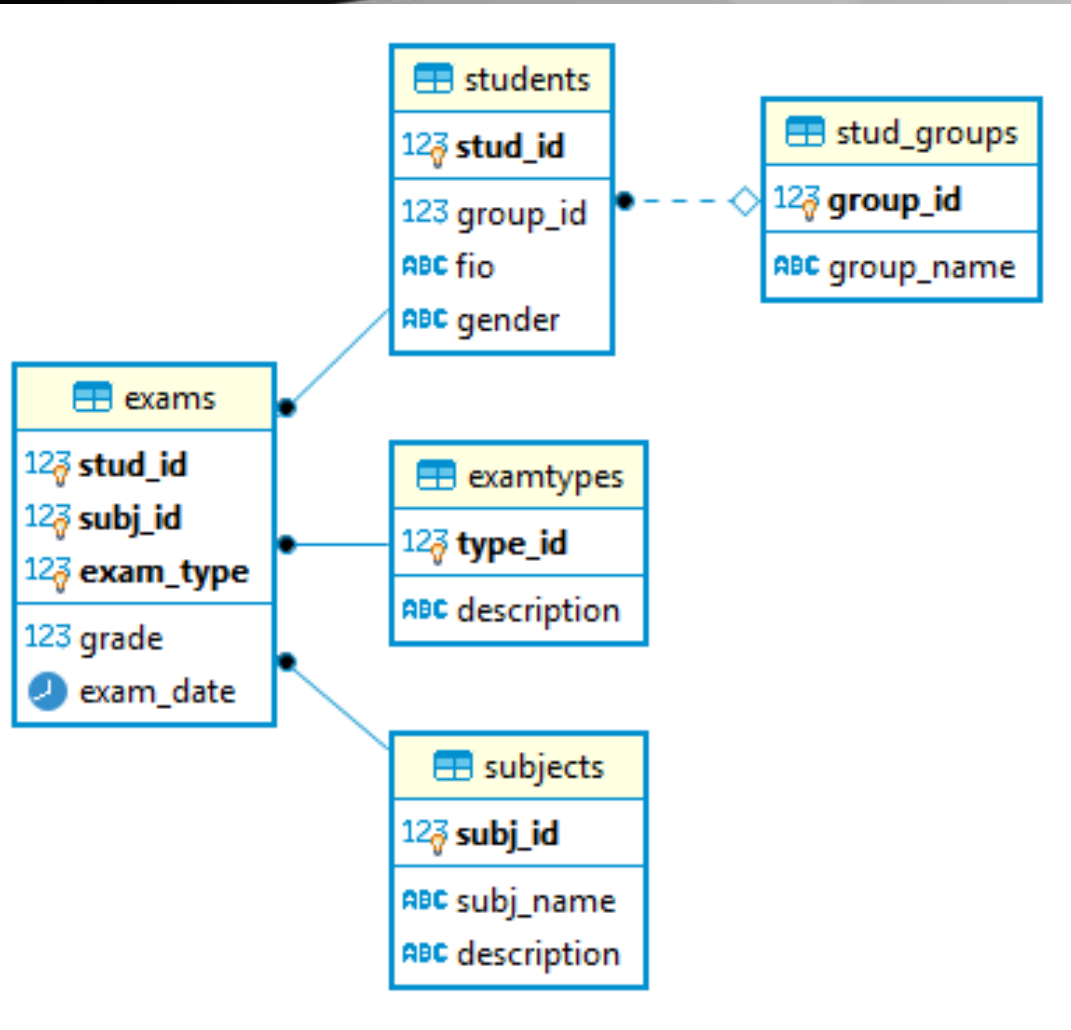
Тестовая база данных - 1

Pubs



Тестовая база данных - 2

Session



```
SELECT s.fio AS "студент", s.group_id AS "группа", j.subj_name AS "дисциплина",  
t.description AS "вид тестирования", e.grade AS "оценка", e.exam_date AS "дата"  
FROM exams e INNER JOIN students s ON e.stud_id = s.stud_id  
INNER JOIN subjects j ON e.subj_id = j.subj_id  
INNER JOIN examtypes t ON e.exam_type = t.type_id  
ORDER BY 2, 1, 3, 4
```

ents(+)

CT s.fio AS "студент", s.group_id AS "гр

Введите SQL выражение чтобы отфильтровать результаты

студент	группа	дисциплина	вид тестирования	оценка	дата
Стравинский Фёдор Михайлович	21 211	Базы данных	Курсовой проект	5	2020-06-24
Стравинский Фёдор Михайлович	21 211	Базы данных	Экзамен	4	2020-06-20
Стравинский Фёдор Михайлович	21 211	Физика	Зачёт	1	2020-05-28
Стравинский Фёдор Михайлович	21 211	Физика	Экзамен	3	2020-06-14
Шариков Полиграф Полиграфович	21 211	Базы данных	Экзамен	3	2020-06-20
Шариков Полиграф Полиграфович	21 211	Физика	Зачёт	0	2020-05-28
Милославский Жорж Вячеславович	21 216	Базы данных	Экзамен	2	2020-06-20
Фанг Анна Фенг Хуа	21 216	Базы данных	Зачёт	5	2020-06-20
Фанг Анна Фенг Хуа	21 216	Базы данных	Курсовой проект	5	2020-06-30
Фанг Анна Фенг Хуа	21 216	Физика	Зачёт	5	2020-06-14
Тимофеев Александр Сергеевич	21 219	Базы данных	Экзамен	4	2020-06-20

Оператор SELECT

The screenshot shows the DBeaver 7.2.1 interface. The main window displays a SQL script with the query `select * from authors;`. Below the script, the results of the query are shown in a table format. The table has columns for author ID, name, phone, address, city, state, zip, and a contract checkbox. The second row is highlighted, showing the address '309 63rd St. #411'.

SQL Query:

```
select * from authors;
```

Table Results:

Table	ABC au_id	ABC au_lname	ABC au_fname	ABC phone	ABC address	ABC city	ABC state	ABC zip	contract
1	409-56-7008	Bennet	Abraham	415 658-9932	6223 Bateman St.	Berkeley	CA	94705	<input checked="" type="checkbox"/>
2	213-46-8915	Green	Marjorie	415 986-7020	309 63rd St. #411	Oakland	CA	94618	<input checked="" type="checkbox"/>
3	238-95-7766	Carson	Cheryl	415 548-7723	589 Darwin Ln.	Berkeley	CA	94705	<input checked="" type="checkbox"/>
4	998-72-3567	Ringer	Albert	801 826-0752	67 Seventh Av.	Salt Lake City	UT	84152	<input checked="" type="checkbox"/>
5	899-46-2035	Ringer	Anne	801 826-0752	67 Seventh Av.	Salt Lake City	UT	84152	<input checked="" type="checkbox"/>
6	722-51-5454	DeFrance	Michel	219 547-9982	3 Balding Pl.	Gary	IN	46403	<input checked="" type="checkbox"/>
7	807-91-6654	Panteley	Sylvia	301 946-8853	1956 Arlington Pl.	Rockville	MD	20853	<input checked="" type="checkbox"/>
8	893-72-1158	McBadden	Heather	707 448-4982	301 Putnam	Vacaville	CA	95688	<input type="checkbox"/>
9	724-08-9931	Stringer	Dirk	415 843-2991	5420 Telegraph Av.	Oakland	CA	94609	<input type="checkbox"/>
10	274-80-9391	Straight	Dean	415 834-2919	5420 College Av.	Oakland	CA	94609	<input checked="" type="checkbox"/>
11	756-30-7391	Karsen	Livia	415 534-9219	5720 McAuley St.	Oakland	CA	94609	<input checked="" type="checkbox"/>
12	724-80-9391	MacFeather	Stearns	415 354-7128	44 Upland Hts.	Oakland	CA	94612	<input checked="" type="checkbox"/>
13	427-17-2319	Dull	Ann	415 836-7128	3410 Blonde St.	Palo Alto	CA	94301	<input checked="" type="checkbox"/>
14	672-71-3249	Yokomoto	Akiko	415 935-4228	3 Silver Ct.	Walnut Creek	CA	94595	<input checked="" type="checkbox"/>
15	267-41-2394	O'Leary	Michael	408 286-2428	22 Cleveland Av. #14	San Jose	CA	95128	<input checked="" type="checkbox"/>
16	472-27-2349	Gringlesby	Burt	707 938-6445	PO Box 792	Covelo	CA	95428	<input type="checkbox"/>
17	527-72-3246	Greene	Morningstar	615 297-2723	22 Graybar House Rd.	Nashville	TN	37215	<input type="checkbox"/>
18	172-32-1176	White	Johnson	408 496-7223	10932 Bigge Rd.	Menlo Park	CA	94025	<input checked="" type="checkbox"/>

Оператор SELECT

ОСНОВЫ

```
SELECT ALL * FROM publishers;
```

```
SELECT TOP 5 p.country  
FROM pubs.public.publishers p LIMIT 5;
```

```
SELECT DISTINCT state FROM authors;
```

```
SELECT 'Название книги: ', title, pubdate  
FROM titles
```

```
select 2+2 as "итого";
```

```
SELECT * INTO NewAuthors FROM authors;
```

- 1 SELECT [ALL | DISTINCT] <список вывода>
- 2 [INTO <имя новой таблицы>]
- 3 [FROM <список таблиц и условий соединения>]
- 4 [WHERE <условие отбора или соединения>]
- 5 [GROUP BY <список полей группировки>]
- 6 [HAVING <условия, накладываемые на группу>]
- 7 [ORDER BY <список полей для сортировки вывода>]
- 8 [UNION <запрос на выборку для объединения>]
- 9 ...

порядок следования разделов нарушать нельзя!

Оператор SELECT

раздел WHERE

Раздел `WHERE` предназначен для наложения горизонтальных фильтров на данные, обрабатываемые запросом. Для этого указывается логическое условие, от результата вычисления которого зависит, будет ли строка включена в результат выборки или нет.

```
SELECT * FROM titles WHERE ytd_sales > 5000;
```

```
SELECT * FROM authors WHERE 1 = 1;
```

```
SELECT title FROM titles  
WHERE extract('year' FROM pubdate) > 1991 AND extract('year' FROM pubdate) <= 1995;
```

```
SELECT * FROM titles  
WHERE title LIKE '%Database%' OR "type" = 'popular_comp';
```

```
SELECT pub_name, 'не Америка!' AS "Особенность" FROM publishers WHERE state IS NULL;
```

Оператор SELECT

раздел WHERE

Раздел `WHERE` **не предназначен** для создания связей между таблицами (по внешним ключам), хотя такая возможность есть. Данный способ является устаревшим и его категорически не рекомендуется использовать при профессиональной работе с СУБД.

```
SELECT title, pub_name FROM titles, publishers  
WHERE titles.pub_id = publishers.pub_id AND country = 'USA';
```

```
SELECT * FROM authors  
WHERE au_fname IN ('Sylvia', 'Anne', 'Livia');
```

```
SELECT * FROM titles  
WHERE price NOT BETWEEN 10::money AND 20::money;
```

```
SELECT Count(*) FROM authors WHERE contract = FALSE;
```

Оператор SELECT

раздел FROM

С помощью раздела FROM определяются источники данных, с которыми будет работать запрос. Можно создать одну или несколько связей между отношениями, явно указывая те поля, которые должны играть роль внешних ключей.

```
SELECT authors.au_lname, authors.au_fname, titleauthor.royaltyper
FROM authors JOIN titleauthor ON authors.au_id = titleauthor.au_id
WHERE authors.state='CA';
```

```
SELECT t.title, a.au_lname, a.au_fname, ta.au_ord
FROM titles t INNER JOIN titleauthor ta ON t.title_id = ta.title_id
INNER JOIN authors a ON ta.au_id = a.au_id;
```

```
SELECT t.title, p.pub_name
FROM titles t RIGHT JOIN publishers p ON t.pub_id = p.pub_id;
```

```
SELECT * FROM titles t
JOIN publishers p ON t.pub_id = p.pub_id AND t.pub_id2 = p.pub_id2;
```

JOIN:

INNER
LEFT
RIGHT
FULL

Оператор SELECT

раздел ORDER BY

Раздел ORDER BY предназначен для упорядочения набора данных, возвращаемых после выполнения запроса.

```
SELECT * FROM titles WHERE price IS NULL  
ORDER BY "type";
```

```
SELECT * FROM students s ORDER BY group_id DESC, fio ASC;
```

```
SELECT s.fio, sg.group_name, sj.subj_name, et.description, e.grade FROM exams e  
JOIN examtypes et ON e.exam_type = et.type_id  
JOIN students s ON e.stud_id = s.stud_id  
JOIN stud_groups sg ON s.group_id = sg.group_id  
JOIN subjects sj ON e.subj_id = sj.subj_id  
ORDER BY 3 DESC, 2, 4 DESC, 5 DESC;
```

Раздел **GROUP BY** позволяет выполнять группировку строк таблиц по определённым критериям. **GROUP BY** почти всегда используется вместе с функциями агрегирования.

```
SELECT p.pub_name, Count(t.title)
FROM titles t JOIN publishers p
ON t.pub_id = p.pub_id
GROUP BY p.pub_name
ORDER BY Count(t.title) DESC;
```

```
SELECT p.pub_name, Count(t.title) AS
"Количество"
FROM titles t JOIN publishers p
ON t.pub_id = p.pub_id
GROUP BY 1 ORDER BY 2 DESC;
```

```
SELECT Max(p.pub_name), Count(t.title)
FROM titles t JOIN publishers p
ON t.pub_id = p.pub_id
GROUP BY p.pub_id ORDER BY 2 DESC;
```

Оператор SELECT

раздел GROUP BY

Функции агрегирования:

AVG(<поле>)	Среднее значение для указанного столбца или выражения
COUNT(<поле>)	Количество строк, исключая NULL-строки в указанном столбце
COUNT(*)	Общее количество строк, включая NULL-строки
MAX(<поле>)	Максимальное значение в указанном столбце
MIN(<поле>)	Минимальное значение в указанном столбце
SUM(<поле>)	Сумма всех значений в указанном столбце
STDEV(<поле>)	Статистическое стандартное отклонение для значений столбца
VAR(<поле>)	Несмещенная оценка дисперсии величин указанного столбца

Важно!

1. Функции агрегирования не работают со значениями NULL.
2. Раздел WHERE не допускает использования функций агрегирования.

Оператор SELECT

раздел GROUP BY

```
SELECT Count(*) AS "Число книг", a.au_fname AS "Имя", a.au_lname AS "Фамилия"  
FROM titles t JOIN titleauthor ta ON t.title_id = ta.title_id  
JOIN authors a ON a.au_id = ta.au_id  
GROUP BY a.au_id  
ORDER BY 2;
```

```
SELECT Cast(sg.group_id AS varchar) || ' (' || sg.group_name || ')' AS "Группа",  
sj.subj_name AS "Дисциплина", Avg(e.grade) AS "Средний балл"  
FROM exams e INNER JOIN students s ON e.stud_id = s.stud_id  
INNER JOIN examtypes et ON e.exam_type = et.type_id  
INNER JOIN stud_groups sg ON s.group_id = sg.group_id  
INNER JOIN subjects sj ON e.subj_id = sj.subj_id  
WHERE et.description LIKE 'Экзамен'  
GROUP BY sg.group_id, sj.subj_id  
ORDER BY 1;
```

Оператор SELECT

раздел HAVING

Раздел HAVING практически аналогичен по назначению разделу WHERE и выполняет функцию горизонтальной фильтрации, однако используется при задании групповой фильтрации. В этом разделе допускается использование функций агрегирования.

```
SELECT max(t.title), Count(*) as "Число авторов"  
FROM titles t JOIN titleauthor ta ON t.title_id = ta.title_id  
GROUP BY ta.title_id  
HAVING count(*)>1
```

```
SELECT s.group_id, Count(*) AS "Число студентов"  
FROM students s GROUP BY 1 HAVING Count(*) > 20;
```

```
SELECT sj.subj_name  
FROM exams e INNER JOIN subjects sj ON e.subj_id = sj.subj_id  
GROUP BY 1  
HAVING Avg(e.grade) > 3.5;
```


Оператор SELECT

прочие разделы

Раздел **UNION** служит для объединения результатов выборки, возвращаемых двумя и более запросами. Это может быть выборка из одной таблицы или слияние данных из множества таблиц. Иными словами, раздел **UNION** вставляется между двумя запросами, возвращающими одинаковый набор столбцов. В результат будут включены строки как первого, так и второго запроса. По умолчанию дублирующие строки в результат не включаются.

```
SELECT pub_name, city FROM publishers  
UNION  
SELECT publisher, location FROM MyAdditionalTable;
```

Пример SELECT

```
SELECT s.fio AS "студент", s.group_id AS "группа",  
j.subj_name AS "дисциплина", t.description AS "вид тестирования",  
CASE WHEN e.grade = -1 THEN 'не аттестован'  
  WHEN e.grade = 0 THEN 'незачтено'  
  WHEN e.grade = 1 THEN 'зачтено'  
  WHEN e.grade = 2 THEN 'неуд.'  
  WHEN e.grade = 3 THEN 'удовлетворительно'  
  WHEN e.grade = 4 THEN 'хорошо'  
  WHEN e.grade = 5 THEN 'отлично'  
  ELSE e.grade::varchar(20)  
END AS "оценка",  
e.exam_date AS "дата"  
FROM exams e INNER JOIN students s ON e.stud_id = s.stud_id  
  INNER JOIN subjects j ON e.subj_id = j.subj_id  
  INNER JOIN examtypes t ON e.exam_type = t.type_id  
ORDER BY 2, 1, 3, 4
```

Оператор SELECT

вложенные запросы

```
SELECT title FROM titles WHERE pub_id IN  
(SELECT pub_id FROM publishers WHERE pub_name LIKE 'New Moon Books');
```

```
SELECT ti.titleinfo  
FROM (SELECT t.title || ' - ' || t.notes AS "titleinfo", t.price FROM titles t) ti  
WHERE ti.price::NUMERIC > 20
```

```
SELECT * FROM publishers p  
WHERE EXISTS (  
    SELECT 1 FROM titles t  
    WHERE p.pub_id = t.pub_id AND t.advance::NUMERIC > 10000  
);
```

```
SELECT p.pub_name FROM publishers p  
WHERE NOT EXISTS (  
    SELECT 1 FROM titles t  
    WHERE p.pub_id = t.pub_id AND t.price IS NULL  
);
```

Типы данных PostgreSQL

числовые типы данных

- ❖ **integer**: целые числа от -2147483648 до +2147483647 (4 байта)
- ❖ **smallint**: целые числа от -32768 до +32767 (2 байта)
- ❖ **bigint**: целые числа от -9223372036854775808 до +9223372036854775807 (8 байт)

- ❖ **serial**: целочисленный счётчик (автоинкремент) от 1 до 2147483647 (4 байта). Отсутствует в стандарте SQL
- ❖ **smallserial**: целочисленный счётчик (автоинкремент) от 1 до 32767 (2 байта)
- ❖ **bigserial**: целочисленный счётчик (автоинкремент) от 1 до 9223372036854775807 (8 байт)

- ❖ **numeric**: числа с фиксированной точностью - до 131072 знаков в целой части и до 16383 знаков после запятой. Описание: `numeric(precision, scale)`. Параметр *precision* - максимальное количество цифр в числе. Параметр *scale* - максимальное количество цифр после запятой
- ❖ **decimal**: синоним `numeric`

- ❖ **real**: числа с плавающей точкой от $1E-37$ до $1E+37$ (4 байта)
- ❖ **double precision**: числа с плавающей точкой от $1E-307$ до $1E+308$ (8 байт)

- ❖ **money**: специальный тип для работы с денежными единицами, от -92233720368547758.08 до +92233720368547758.07, производный от `numeric`, с указанием денежной единицы (8 байт)

Типы данных PostgreSQL

прочие типы данных

Наиболее удобный формат задания:
даты: yyyy-mm-dd – 2020-01-08
времени: hh:mi:ss – 1:21:34

Символьные типы:

- ❖ **char(n)**: строка из фиксированного количества символов, *n* - количество символов в строке
- ❖ **varchar(n)**: строка из произвольного количества символов, *n* – максимальное количество символов в строке. Максимально возможный размер строки составляет около 1 ГБ
- ❖ **text**: текст произвольной длины. Не входит в стандарт SQL

Двоичные типы:

- ❖ **bytea**: двоичная строка переменной длины. Входные данные по умолчанию принимаются в шестнадцатиричном формате: `SELECT '\xDEADBEE01'`; отличается от стандарта SQL

Типы даты/времени:

- ❖ **timestamp**: дата и время без часового пояса от 4713 г. до н. э. до 294276 г. н. э., точность 1 мкс, 8 байт
- ❖ **timestamp with time zone**: дата и время с часовым поясом, всё остальное как для предыдущего
- ❖ **date**: дата (без времени суток), от 4713 г. до н. э. до 5874897 г. н. э., точность 1 день, 4 байта
- ❖ **time**: время суток (без даты), точность 1 мкс, 8 байт
- ❖ **time with time zone**: время суток (без даты) с часовым поясом (+/- 1459), точность 1 мкс, 12 байт
- ❖ **interval**: временной интервал от -178000000 лет до 178000000 лет, точность 1 мкс, 16 байт

некоторые типы данных
ещё не рассмотрены...

Типы данных PostgreSQL

прочие типы данных

Логический тип:

- ❖ **boolean**: значения TRUE, FALSE, NULL::boolean, размер 1 байт.

Типы для представления Интернет-адресов:

- ❖ **cidr**: интернет-адрес в формате IPv4 и IPv6. Например, 192.168.0.1. От 7 до 19 байт
- ❖ **inet**: интернет-адрес в формате cidr/y. Например, 192.168.0.1/24. От 7 до 19 байт
- ❖ **macaddr**: хранит MAC-адрес. Пример ввода: '08-00-2b-01-02-03'. 6 байт
- ❖ **macaddr8**: хранит MAC-адрес в формате EUI-64. Пример ввода: '08-00-2b-01-02-03-04-05'. 8 байт

Геометрические типы:

- ❖ **point, line, lseg, box, path, polygon, circle**

Прочие типы:

- ❖ **json**: хранит данные json в текстовом виде
- ❖ **jsonb**: хранит данные json в бинарном формате (ускоряет обработку, поддерживает индексацию)
- ❖ **uuid**: хранит (но не генерирует) универсальный уникальный идентификатор (UUID), например, a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11. 32 байта
- ❖ **xml**: хранит данные в формате XML

Используется для ввода данных: добавляет одну или несколько строк в одну таблицу.

Команда INSERT

Синтаксис (упрощённый):

```
INSERT INTO <имя_таблицы> [  
<имя_столбца>,... ] {  
  [VALUES (<значение>,..) ]  
  |[ <SELECT-запрос> ]  
  |[ DEFAULT VALUES ]  
}
```

Все столбцы, не представленные в явном или неявном списке столбцов, получают значения по умолчанию, если для них заданы эти значения, либо NULL в противном случае.

```
INSERT INTO exams VALUES  
(301667, 1, 2, 2, '20.06.2020'::date);
```

```
INSERT INTO subjects(subj_id, subj_name, description)  
VALUES  
(1, 'Базы данных', 'основы работы с PostgreSQL'),  
(2, 'Физика', 'Часть 3. Оптика'),  
(3, 'Английский язык', '');
```

```
INSERT INTO authors DEFAULT VALUES;
```

```
INSERT INTO films  
SELECT * FROM tmp_films WHERE date_prod < '2004-05-07';
```


Используется для
изменения данных:
модифицирует одну или несколько
строк в таблице.

Команда UPDATE

```
UPDATE students SET fio = 'Нетсуорти Эстер'  
WHERE fio = 'Шоу Эстер';
```

```
UPDATE subjects SET subj_name = 'Иностранный язык',  
description = 'группы английского и французского языков'  
WHERE subj_id = 3;
```

```
UPDATE exams e SET grade = 5  
FROM students s, subjects j  
WHERE e.stud_id = s.stud_id AND e.subj_id = j.subj_id  
AND s.fio = 'Стравинский Фёдор Михайлович'  
AND j.subj_name = 'Базы данных'  
AND e.exam_date = '20.06.2020'::date;
```

Синтаксис (упрощённый):

```
UPDATE <имя_таблицы>  
SET <имя_столбца> = <значение>, ...  
[FROM {<имя_таблицы>  
|<SELECT-запрос>},...]  
[WHERE <условие>]
```

Если в UPDATE будет пропущен раздел WHERE, то заданные в разделе SET изменения будут сделаны в каждой строке таблицы.

В PostgreSQL надёжнее ссылаться на другие таблицы в подзапросах, хотя такие запросы часто работают медленнее, чем FROM-соединение.

Команда UPDATE

ещё несколько примеров

```
UPDATE weather SET (temp_lo, temp_hi, prcp) = (temp_lo+1, temp_lo+15, DEFAULT)
WHERE city = 'San Francisco' AND "date" = '2003-07-03';
```

```
UPDATE titles SET price = price + 30
WHERE extract('year' FROM pubdate) = 2020;
```

```
UPDATE employees SET sales_count = sales_count + 1 WHERE id =
(SELECT sales_person FROM accounts WHERE name = 'Acme Corporation');
```

```
UPDATE summary s SET (sum_x, sum_y, avg_x, avg_y) =
(SELECT sum(x), sum(y), avg(x), avg(y) FROM data d
WHERE d.group_id = s.group_id);
```

Используется для
удаления данных:
удаляет одну или несколько
строк из таблицы.

Команда DELETE

Синтаксис (упрощённый):

```
DELETE FROM <имя_таблицы>  
[ WHERE <условие> ]}
```

Без WHERE будут удалены все
строки таблицы. В разделе WHERE
также можно использовать
вложенные подзапросы (т.е.
используя данные других таблиц).

Быстрый вариант удаления всех
данных в одной или нескольких
таблицах:

```
TRUNCATE table1, table2
```

```
DELETE FROM titleauthor;
```

```
DELETE FROM publishers p  
WHERE p.pub_name = 'Microsoft Press';
```

```
DELETE FROM students s  
WHERE s.stud_id IN (  
  SELECT e.stud_id FROM exams e  
  WHERE e.grade = 2  
  GROUP BY e.stud_id  
  HAVING count(*) > 3  
);
```

```
DELETE FROM tasks  
WHERE status = 'DONE' RETURNING *;
```

Используется для
создания новой таблицы.

Команда CREATE TABLE

Синтаксис (упрощённый):

```
CREATE TABLE [ IF NOT EXISTS ] имя_таблицы ( [  
  { имя_столбца тип_данных [ ограничение_столбца [ ... ] ]  
  | ограничение_таблицы  
  | LIKE исходная_таблица [ вариант_копирования ... ] }  
  [, ... ]  
  ] )  
[ TABLESPACE табл_пространство ]
```

ограничение_столбца:

```
[ CONSTRAINT имя_ограничения ]  
{ NOT NULL | NULL |  
CHECK ( выражение ) [ NO INHERIT ] |  
DEFAULT выражение_по_умолчанию |  
UNIQUE параметры_индекса |  
PRIMARY KEY параметры_индекса |  
REFERENCES целевая_таблица [ (  
целевой_столбец ) ]
```

ограничение_таблицы:

```
[ CONSTRAINT имя_ограничения ]  
{ CHECK ( выражение ) [ NO INHERIT ] |  
UNIQUE ( имя_столбца [, ... ] ) параметры_индекса |  
PRIMARY KEY ( имя_столбца [, ... ] ) параметры_индекса |  
EXCLUDE [ USING индексный_метод ] ( элемент_исключения WITH  
оператор [, ... ] ) параметры_индекса [ WHERE ( предикат ) ] |  
FOREIGN KEY ( имя_столбца [, ... ] ) REFERENCES целевая_таблица [ (  
целевой_столбец [, ... ] ) ]
```

Примеры создания таблиц

база данных "Сессия"

```
CREATE TABLE stud_groups (  
  group_id integer PRIMARY KEY,  
  group_name TEXT NOT NULL  
);
```

```
CREATE TABLE subjects (  
  subj_id serial PRIMARY KEY,  
  subj_name TEXT NOT NULL,  
  description TEXT  
);
```

```
CREATE TABLE examtypes (  
  type_id integer PRIMARY KEY,  
  description TEXT NOT NULL  
);
```

```
CREATE TABLE students(  
  stud_id integer PRIMARY KEY,  
  group_id integer REFERENCES stud_groups( group_id ),  
  fio TEXT NOT NULL,  
  gender char(1), CHECK (gender IN ('F', 'M'))  
);
```

```
CREATE TABLE exams(  
  stud_id integer REFERENCES students( stud_id ),  
  subj_id integer REFERENCES subjects( subj_id ),  
  exam_type integer REFERENCES examtypes ( type_id ),  
  grade integer, CHECK (grade IN (0, 1, 2, 3, 4, 5)),  
  exam_date date,  
  PRIMARY KEY (stud_id, subj_id, exam_type)  
);
```

Примеры создания таблиц

из документации

```
CREATE TABLE films (  
    code      char(5) CONSTRAINT firstkey PRIMARY KEY,  
    title     varchar(40) NOT NULL,  
    did       integer NOT NULL,  
    date_prod date,  
    kind      varchar(10),  
    len       interval hour to minute  
);  
  
CREATE TABLE distributors (  
    did      integer PRIMARY KEY DEFAULT nextval('serial'),  
    name     varchar(40) NOT NULL CHECK (name <> '')  
);  
  
CREATE TABLE users (  
    id serial PRIMARY KEY,  
    description text  
);
```

ALTER TABLE модифицирует структуру таблицы,
DROP TABLE удаляет таблицу целиком.

Реконструкция и удаление таблиц

Команда **ALTER TABLE** берёт на себя все действия по копированию данных во временную таблицу, удалению старой таблицы, созданию вместо неё новой таблицы с нужной структурой и последующим переписыванием в неё данных. Все эти действия происходят без участия пользователя, при этом установленные права доступа к таблице сохраняются.

DROP TABLE удаляет все индексы, правила, триггеры и ограничения, существующие для таблицы. Таблица удаляется вместе с содержащимися в ней данными. Однако, чтобы удалить таблицу, на которую ссылается представление или ограничение внешнего ключа в другой таблице, необходимо дополнительно указать опцию **CASCADE**.

```
ALTER TABLE distributors  
  ADD COLUMN address varchar(30);
```

```
ALTER TABLE distributors  
  DROP COLUMN address RESTRICT;
```

```
ALTER TABLE distributors  
  ALTER COLUMN address TYPE varchar(80),  
  ALTER COLUMN name TYPE varchar(100);
```

```
DROP TABLE users;
```

```
DROP TABLE films, distributors;
```


Временные таблицы

Временные таблицы автоматически удаляются в конце сеанса или в конце текущей транзакции, если они были созданы внутри транзакции.

```
BEGIN;  
  CREATE TEMP TABLE myTable (  
    id      serial PRIMARY KEY,  
    fio     varchar(300),  
    dt      date DEFAULT now(),  
    remark  text  
  ) ON COMMIT DROP;  
  
  INSERT INTO myTable (fio, remark)  
    VALUES ('Босой Никанор Иванович', 'председатель жилтоварищества');  
  
  SELECT * FROM myTable;  
END;  
  
SELECT * FROM myTable; -- выдаст сообщение об ошибке
```

В случае, если возникла ошибка внутри транзакции, которая привела к её остановке, все дальнейшие команды будут игнорироваться, пока текущая транзакция не будет отменена с помощью команды **ROLLBACK TRANSACTION**

```
DROP VIEW IF EXISTS publications;  
  
CREATE VIEW publications AS  
  SELECT a.au_lname AS "Фамилия",  
         a.au_fname AS "Имя",  
         t.title AS "Название книги"  
FROM authors a JOIN titleauthor ta  
  ON a.au_id = ta.au_id  
  JOIN titles t  
  ON ta.title_id = t.title_id;  
  
SELECT * FROM publications  
ORDER BY Фамилия, Имя;  
  
ALTER VIEW publications  
RENAME TO summary1;
```

Представление для пользователей базы данных выглядит как таблица, однако на самом деле его содержимое формируется запросом. Физически данные, виртуально принадлежащие представлению, находятся в таблицах, к которым обращается этот запрос.

Имя представления должно отличаться от имён других представлений, таблиц, последовательностей, индексов или сторонних таблиц в той схеме данных, где оно создаётся. При анализе запроса нет абсолютно никакой разницы между таблицами и представлениями.

Столбец представления будет изменяемым, если это простая ссылка на изменяемый столбец нижележащего базового отношения; в противном случае этот столбец будет доступен только для чтения. Представления, выбирающие данные не из одной таблицы, недоступны для добавления или изменения данных.

Курсоры

Жизненный цикл курсора:

- ❖ Объявление (**DECLARE**)
- ❖ Открытие курсора (**OPEN**)
- ❖ Работа с данными: считывание **FETCH**, перемещение курсора **MOVE**, изменение (**UPDATE**) и удаление (**DELETE**) данных
- ❖ Закрытие курсора (**CLOSE**)

Курсоры удобно использовать внутри пользовательских функций, где можно реализовать достаточно сложный алгоритм работы с данными. Курсоры выполняются только внутри блоков транзакций (**BEGIN ... END**), а для этого следует писать программу на расширении языка PL/pgSQL.

Курсор – это указатель на записи внутри временной выборки данных, полученной в результате выполнения некоего SQL-запроса. Курсор может двигаться по соответствующему набору данных, позволяя возвращать клиенту отдельные записи. Использование курсоров удобно тогда, когда выборка данных очень большая и, кроме того, над данными необходимо проводить какие-то сложные преобразования.

Курсоры позволяют работать со строками таблицы посредством указания их порядкового номера в наборе данных. Курсор может быть использован не только для выборки данных, но также для изменения и добавления данных в отдельные строки таблицы.

```
DO $$ --переход в режим программирования на языке PL/pgSQL
DECLARE
    myCurs CURSOR FOR SELECT title FROM titles;
    varTitle varchar(200);
BEGIN -- начало транзакции
    OPEN myCurs;
    LOOP
        FETCH myCurs INTO varTitle;
        varTitle = '<td class="myColumn">' || varTitle || '</td>';
        RAISE NOTICE '%', varTitle; -- вывод в консоль для отладки
        IF NOT FOUND
            THEN EXIT;
        END IF;
    END LOOP;
CLOSE myCurs;
END $$;
```

Варианты FETCH и MOVE:

```
FETCH NEXT FROM cur1
INTO v1, v2, v3;
```

```
FETCH LAST FROM cur2
INTO x, y;
```

```
FETCH RELATIVE -2
FROM cur3 INTO x;
```

```
M MOVE cur1;
```

```
M MOVE FORWARD 2
FROM cur2;
```

Хранимые процедуры

```
-- создаём хранимую процедуру
CREATE PROCEDURE insert_info
(description varchar(50),
smin int, smax int)
LANGUAGE SQL
AS $$
    INSERT INTO jobs SELECT 1 + MAX(job_id),
description, smin, smax FROM jobs;
$$;

-- выполняем хранимую процедуру
CALL insert_info('Разработчик баз данных',
200, 250);

SELECT * FROM jobs;
```

Хранимая процедура – это именованный набор SQL-команд либо языковых инструкций, сохраняемая непосредственно на сервере баз данных и оптимизированная для наиболее эффективной работы. Хранимая процедура не возвращает каких-либо значений.

Хранимые процедуры существуют независимо от таблиц или каких-либо других объектов баз данных. Команда **CREATE PROCEDURE** (создать хранимую процедуру) определена в стандарте SQL, но лишь в последних версиях PostgreSQL была принята к использованию. В этой СУБД язык программирования для написания хранимой процедуры может быть различным.

Для изменения содержания кода хранимой процедуры используется команда **ALTER PROCEDURE**, для удаления - **DROP PROCEDURE**.

```
CREATE FUNCTION clean_emp(v int) RETURNS int AS $$
DELETE FROM employee
WHERE job_lvl < v; -- уровень значимости
SELECT count(*) FROM employee
$$ LANGUAGE SQL;
```

```
SELECT clean_emp(40); -- неинформативный вывод
```

```
DROP FUNCTION clean_emp; -- исправим:
```

```
CREATE FUNCTION clean_emp(v int) RETURNS int AS $$
DECLARE
    t1 int;
    t2 int;
BEGIN
    SELECT count(*) INTO t1 FROM employee;
    DELETE FROM employee
    WHERE job_lvl < v;
    SELECT count(*) INTO t2 FROM employee;
RETURN t1-t2;
END;
$$ LANGUAGE plpgsql;
```

```
SELECT clean_emp(40); -- теперь лучше
```

Пользовательские функции

Пользовательская функция – это именованный набор SQL-команд либо языковых инструкций, сохраняемая непосредственно на сервере баз данных и оптимизированная для наиболее эффективной работы. Функции имеют возможность их вызова непосредственно из выражений и способны возвращать результат (в том числе как множество записей). Функции могут принимать в качестве аргументов (параметров) базовые типы, составные типы или их сочетания.

Функции можно писать на языках SQL, PL/pgSQL, на процедурных языках (C, Python и др.)

Пользовательские функции

Ещё несколько примеров создания функций:

```
CREATE FUNCTION bigJobs(integer) RETURNS SETOF character AS $$  
-- Функция возвращает описание должностей с указанного уровня значимости  
SELECT job_desc FROM jobs WHERE job_lvl > $1;  
$$ LANGUAGE sql;
```

```
SELECT bigJobs(100);
```

```
CREATE FUNCTION f_add(x integer, y integer) RETURNS integer AS $$  
SELECT x + y; -- Просто сумма двух чисел  
$$ LANGUAGE SQL;
```

```
CREATE FUNCTION work_with_tab (x int)  
RETURNS SETOF record  
AS $$  
SELECT $1 + tab.y, $1 * tab.y FROM tab;  
$$ LANGUAGE SQL; -- Функция возвращает таблицу из двух столбцов
```


Пример триггера: ведение лога для списка пользователей

```
-- создаём необходимые таблицы
CREATE TABLE users (
  id SERIAL PRIMARY KEY,
  "name" TEXT);

CREATE TABLE log (
  id SERIAL PRIMARY KEY,
  info TEXT NOT NULL,
  ts timestamp WITHOUT TIME ZONE);

-- создаём пользовательскую функцию
CREATE FUNCTION addLog() RETURNS TRIGGER AS
$$
BEGIN
  IF TG_OP='INSERT' THEN
    INSERT INTO log (info, ts)
    VALUES ('Добавлен пользователь', now());
    RETURN NEW;
  END IF;
END;
$$ LANGUAGE plpgsql;
```

Триггеры

Триггер определяет операцию (функцию), которая должна выполняться при *наступлении некоторого события* в базе данных. Чаще всего функция-триггер привязывается к некоторой таблице и, в случае применения к этой таблице какой-либо команды (INSERT, UPDATE, DELETE) триггер *срабатывает* и осуществляет какие-либо действия с этой же таблицей либо с другими объектами базы данных.

В PostgreSQL триггер создаётся на основе существующей функции, т.е. сначала командой **CREATE FUNCTION** определяется пользовательская функция, затем на её основе командой **CREATE TRIGGER** определяется сам триггер.

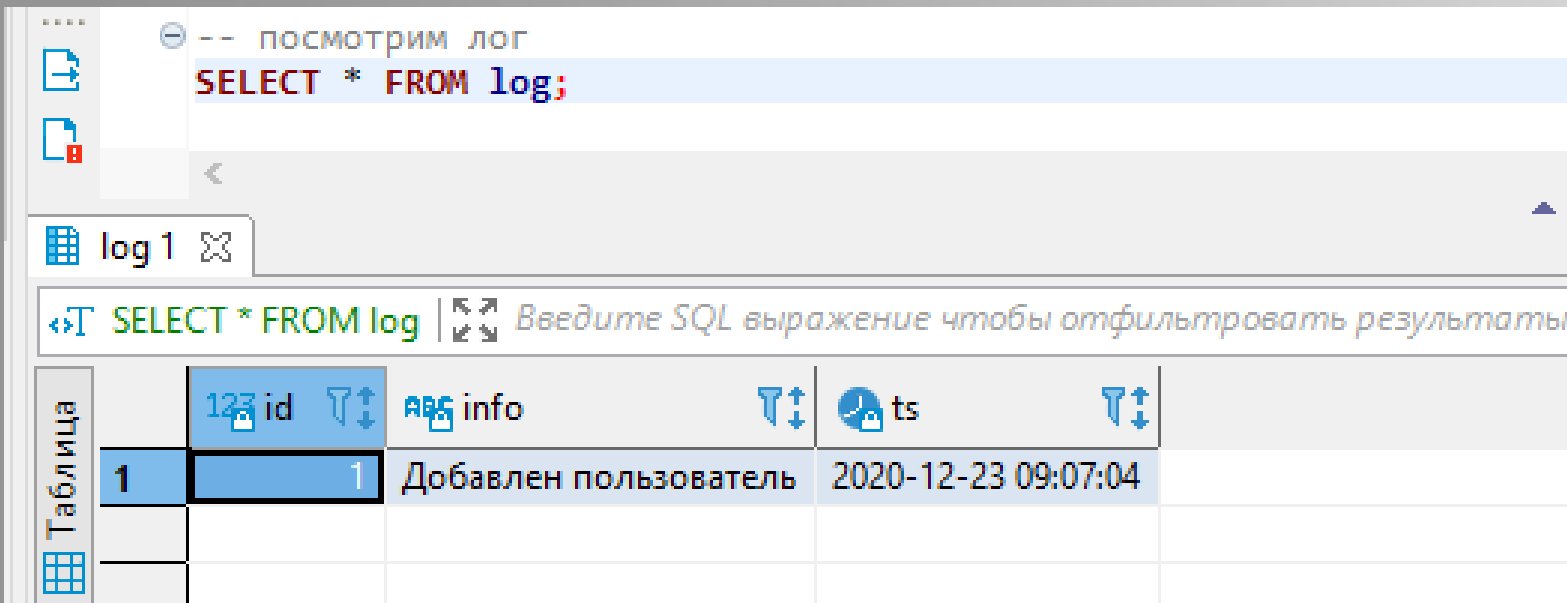
В триггерных функциях используются специальные переменные (*NEW*, *OLD*, *TG_NAME* и др.), содержащие информацию о сработавшем триггере. С помощью этих переменных триггерная функция может работать с данными.

```
-- создаём триггер
CREATE TRIGGER TrUser AFTER INSERT -- OR UPDATE OR DELETE
ON users FOR EACH ROW
EXECUTE PROCEDURE addLog();

-- добавляем запись в таблицу users
INSERT INTO users(name) VALUES ('Виталий Пикулев');
```

В PostgreSQL можно создавать триггеры двух типов:

- ❖ **триггер на изменение данных** - объявляется как функция без аргументов и с типом результата trigger. Пример такого триггера приведен выше.
- ❖ **триггер на событие** - объявляется как функция без аргументов и с типом результата event_trigger.



The screenshot shows a PostgreSQL client interface. At the top, a SQL query is entered: `-- посмотрим лог` followed by `SELECT * FROM log;`. Below the query editor, a table titled "log 1" is displayed. The table has three columns: "id", "info", and "ts". The first row of data shows the value "1" in the "id" column, "Добавлен пользователь" in the "info" column, and "2020-12-23 09:07:04" in the "ts" column.

id	info	ts
1	Добавлен пользователь	2020-12-23 09:07:04



Администрирование PostgreSQL

общие принципы

PostgreSQL использует концепцию **ролей** (roles) для управления разрешениями на доступ к базе данных. Роль можно рассматривать как **пользователя** базы данных или как **группу** пользователей, в зависимости от того, как эта роль настроена. До версии 8.1 в PostgreSQL *пользователи* и *группы* были отдельными сущностями, но теперь есть только роли. Роли базы данных являются глобальными для всего *кластера* базы данных. Роль суперпользователя по умолчанию называется *postgres*. Для создания других ролей вначале нужно подключиться с ролью суперпользователя.

У каждой роли есть **набор прав**, позволяющий ей совершать с использованием СУБД определённые манипуляции над базой данных, вплоть до создания и удаления базы данных. Когда в базе данных создаётся *объект*, ему назначается *владелец*. Владелцем обычно является роль, с которой был создан этот объект. Для большинства типов объектов в исходном состоянии только владелец (или суперпользователь) может делать с объектом всё, что угодно. Чтобы разрешить использовать его другим ролям, нужно дать им права.

Каждое подключение к серверу базы данных выполняется под именем конкретной роли. Пароль имеет значение, если того требует **метод аутентификации клиентов**. База данных и операционная система используют отдельные пароли.

Администрирование PostgreSQL

общие принципы

Табличные пространства – средство для организации физического хранения данных. Табличные пространства позволяют администраторам организовать логику размещения файлов объектов базы данных в файловой системе и могут использоваться несколькими базами данных кластера. Табличные пространства позволяют оптимизировать производительность согласно бизнес-процессам, связанным с объектами баз данных.

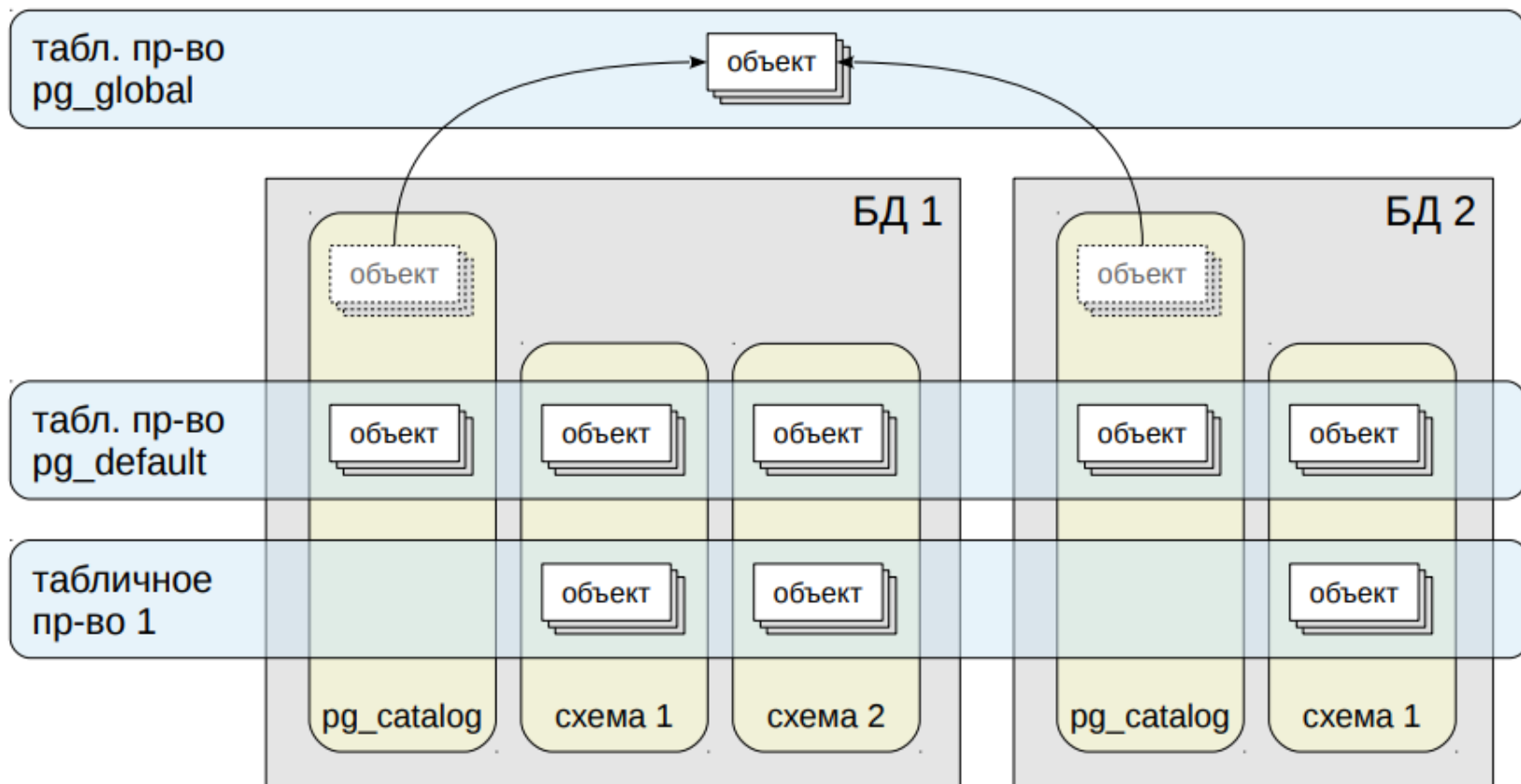
База данных содержит одну или несколько именованных **схем**, которые, в свою очередь, содержат таблицы и другие объекты базы данных (функции, процедуры, представления и др.). Необходимость схемы данных определяется возможностью разделения объектов на логические группы (между различными пользователями и приложениями). Схема *public* содержится во всех создаваемых базах данных и создаётся по умолчанию. Каждая схема имеет своего владельца. По умолчанию роль не может обращаться к объектам в чужих схемах.

PostgreSQL управляет доступом к базе данных с помощью **системы привилегий**. Право выдачи и отзыва привилегий имеют владелец и суперпользователь (это право не может быть ограничено). Если роли были выданы привилегии с правом перевыдачи (*with grant option*), то такая роль может выдать аналогичную привилегию другой роли.

Администрирование PostgreSQL

общие принципы

кластер



Типичные схемы:

public — по умолчанию в ней создаются все объекты

pg_catalog — системные таблицы

Управление схемами данных

```
-- создание
CREATE SCHEMA mySchema;

-- изменение
ALTER SCHEMA mySchema RENAME TO newSchema;

-- удаление
DROP SCHEMA newSchema cascade

-- список схем
select * from pg_namespace;

-- перемещение объектов между схемами
ALTER TABLE myTable SET SCHEMA mySchema;
```

Для временных таблиц создаются схемы *pg_temp_N*. По окончании сеанса все объекты временной схемы удаляются, сама схема остаётся для повторного использования.

Схема *pg_catalog* используется для объектов системного каталога. Альтернативный вариант (в соответствии со стандартом SQL) - *information_schema*.

Виды привилегий

Для таблиц:

- ❖ **SELECT** - чтение данных
- ❖ **INSERT** - вставка строк
- ❖ **UPDATE** - изменение строк
- ❖ **DELETE** - удаление строк
- ❖ **TRUNCATE** - очистка таблицы
- ❖ **REFERENCES** - ссылка на таблицу во внешнем ключе
- ❖ **TRIGGER** - создание триггеров для таблицы

Для представлений:

- ❖ **SELECT** - чтение данных
- ❖ **TRIGGER** - создание триггеров для представления

Для функций:

- ❖ **EXECUTE** – выполнение

Привилегии для конкретных ролей выдаются командой **GRANT**, имеющей много различных вариантов для разных типов объектов. Отзыв привилегий выполняется командой **REVOKE** и по синтаксису похож на выдачу разрешений.

Для схем:

- ❖ **CREATE** - создание объектов в схеме
- ❖ **USAGE** - доступ к объектам в схеме

Для баз данных:

- ❖ **CREATE** - создание схем в базе данных
- ❖ **CONNECT** – подключение
- ❖ **TEMPORARY** - создание временных таблиц

Выдача привилегий

```
-- разрешить чтение и изменение таблицы для роли workRole
GRANT SELECT, UPDATE ON TABLE authors TO workRole;

-- разрешить выполнение всех функций в схеме для роли workRole
GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA mySchema TO workRole;

-- выдача привилегии с правом перевыдачи
GRANT SELECT ON authors TO role2 WITH GRANT OPTION;

-- выдача привилегии на ввод данных в таблицу films
-- всем пользователям (ролям)
GRANT INSERT ON films TO PUBLIC;

-- ввести роль pikulev в группу (роль) admins
GRANT admins TO pikulev;

-- дать указанной роли все разрешения на указанную таблицу
GRANT ALL ON products TO pikulev;

-- разрешаем полный доступ ко всем таблицам для роли demo
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA bookings TO demo;
```

команда GRANT

Для назначения права сразу всем ролям в системе можно использовать специальное имя *PUBLIC*.

Владелец объекта может лишиться себя прав по умолчанию если, например, разрешит всем (*PUBLIC*) только чтение таблицы.

GRANT может включать роль в члены одной или нескольких ролей. Права, выданные для роли, распространяются на всех её членов.

Роль может отозвать привилегию только у той роли, которой она привилегию выдала.

Отмена привилегий

команда REVOKE

```
-- лишение группы public права добавлять данные в таблицу authors
REVOKE INSERT ON authors FROM PUBLIC;

-- лишение пользователя pikulev всех прав для представления titleview
REVOKE ALL PRIVILEGES ON titleview FROM pikulev;

-- исключение из членов роли admins пользователя ivanov
REVOKE admins FROM ivanov;

-- запретить доступ ко всем таблицам заданной схемы для пользователя pikulev
REVOKE ALL PRIVILEGES ON ALL TABLES IN SCHEMA public FROM pikulev;

-- каскадное удаление прав
REVOKE SELECT ON titles FROM ivanov CASCADE;

-- можно лишить прав на выдачу прав
REVOKE GRANT OPTION FOR SELECT ON titles FROM Ivanov CASCADE;
```

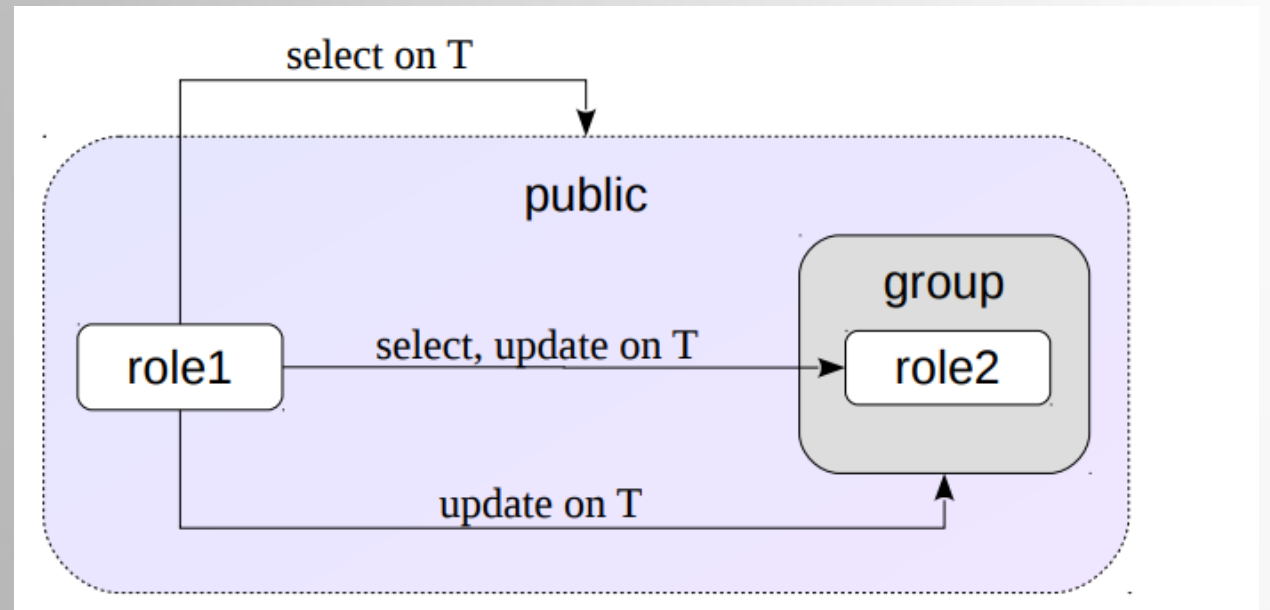
Правила привилегий

ALTER OBJECT OWNER TO new_role

Любая конкретная роль получает права, данные непосредственно ей, плюс права, данные любой роли, в которую она включена, плюс права, данные группе PUBLIC. Поэтому, например, лишение PUBLIC права SELECT не обязательно будет означать, что все роли лишатся права SELECT для данного объекта: оно сохранится у тех ролей, которым оно дано непосредственно или косвенно, через другую роль.

Следует обратить внимание, что лишение права SELECT какого-либо пользователя может не повлиять на его возможность пользоваться правом SELECT, если это право дано группе PUBLIC или другой роли, в которую он включён.

Если команду GRANT или REVOKE выполняет суперпользователь, эта команда выполняется так, как будто её выполняет владелец затрагиваемого объекта. Так как все права всегда исходят от владельца объекта (возможно, косвенно по цепочке или через право распоряжения правом), суперпользователь может отозвать все права, но это может потребовать применения режима CASCADE.



Если *role1* выполнит REVOKE SELECT, UPDATE ON t FROM role2, какие привилегии останутся у *role2*?

Посмотреть текущие привилегии роли можно с помощью запроса

```
select * from  
INFORMATION_SCHEMA.table_privileges
```

```
-- заходим в БД с ролью суперпользователя и  
-- создаём новый пользовательский аккаунт (роль)  
CREATE USER arti WITH PASSWORD 'somepassword';  
-- либо альтернативный (современный) вариант  
CREATE ROLE arti LOGIN PASSWORD 'somepassword';
```

```
-- даём новой роли права на доступ к БД  
GRANT CONNECT ON DATABASE demo TO arti;
```

```
-- разрешаем новой роли доступ к схеме данных  
GRANT USAGE ON SCHEMA bookings TO arti;
```

```
-- но мягко ограничиваем роль в правах  
GRANT SELECT ON ALL TABLES IN SCHEMA bookings TO arti;  
GRANT SELECT ON ALL SEQUENCES IN SCHEMA bookings TO arti;  
GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA bookings TO arti;
```

Управление пользователями

Посмотреть все активные роли БД можно с помощью запроса:

```
select * from pg_user;
```

Перечитать и **применить** новые настройки администрирования БД следует таким образом:

```
select pg_reload_conf();
```

Не следует забывать и про команды управления базой данных:

```
ALTER DATABASE demo  
OWNER TO arti;
```

Управление пользователями

```
-- создание роли, которая может создавать БД
-- и управлять ролями
CREATE ROLE admin WITH CREATEDB CREATEROLE;

-- создание роли с паролем, ограниченным во времени
CREATE ROLE miriam
WITH LOGIN ENCRYPTED PASSWORD 'somepassword'
VALID UNTIL '2022-01-01';

-- разрешаем роли создавать базы данных
ALTER USER arti WITH createdb;

-- создаём базу данных
CREATE DATABASE myNewDB;

CREATE DATABASE sales
OWNER salesapp TABLESPACE salespace;
```

Команда **CREATE ROLE** добавляет новую роль в набор баз данных СУБД PostgreSQL. Чтобы выполнить эту команду, необходимо быть суперпользователем или иметь право **CREATEROLE**. Для изменения атрибутов роли применяется **ALTER ROLE**, а для удаления роли — **DROP ROLE**.

Пароль передаётся на сервер открытым текстом и может попасть в историю команд клиента или в протокол работы сервера.

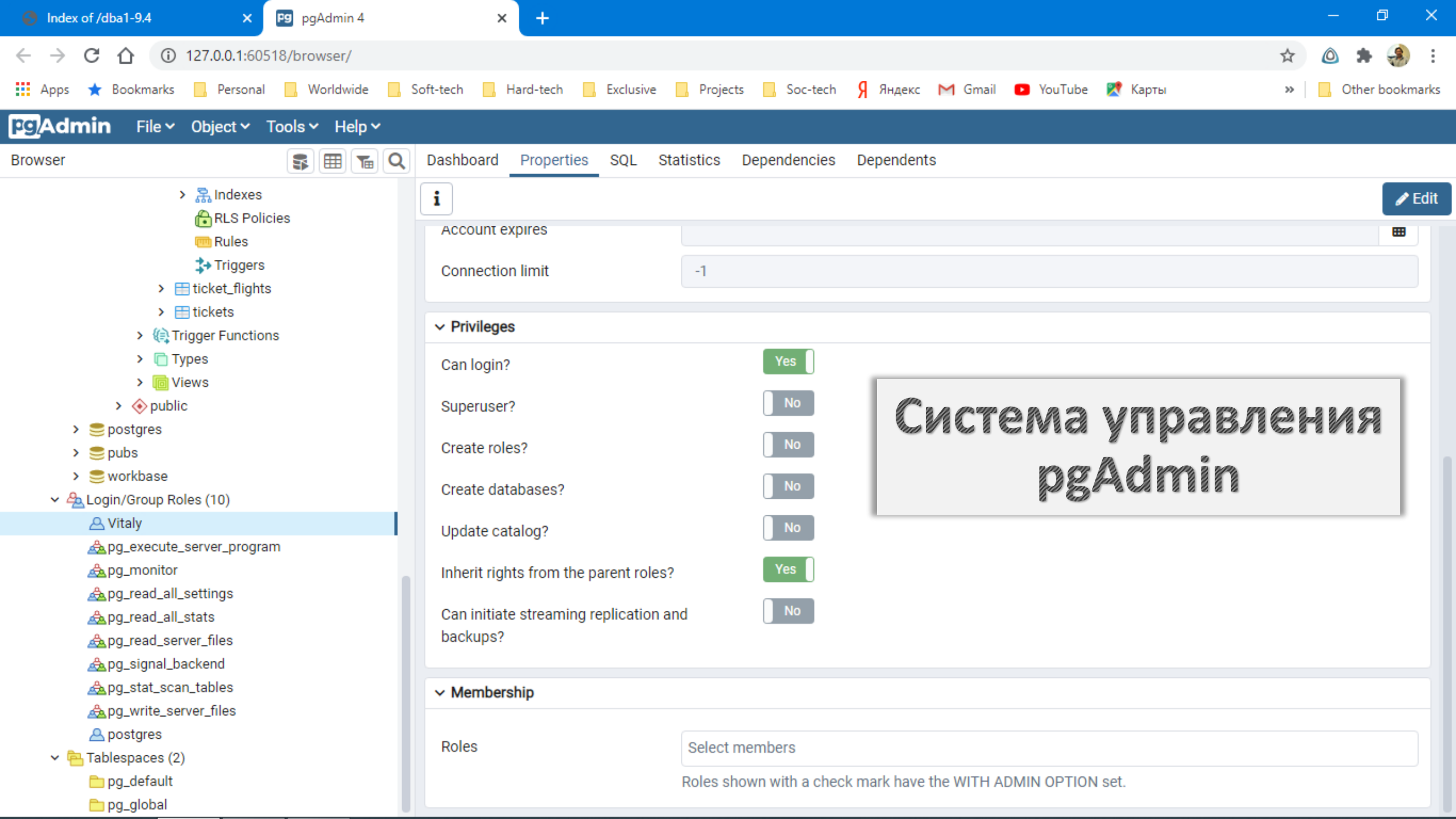
Система управления pgAdmin

The screenshot displays the pgAdmin 4 web interface in a browser window. The browser's address bar shows the URL `127.0.0.1:60518/browser/`. The interface includes a navigation menu with options like 'File', 'Object', 'Tools', and 'Help'. A left sidebar shows a tree view of the database structure, including 'Servers (1)', 'PostgreSQL 13', 'Databases (4)', and 'demo'. The main content area features several monitoring dashboards:

- Database sessions:** A line chart showing 'Total', 'Active', and 'Idle' sessions over time. The y-axis ranges from 0 to 1.
- Transactions per second:** A line chart showing 'Transactions', 'Commits', and 'Rollbacks' over time. The y-axis ranges from 0 to 25.
- Tuples in:** A line chart showing 'Inserts', 'Updates', and 'Delete' operations over time. The y-axis ranges from 0 to 1.
- Tuples out:** A line chart showing 'Fetched' and 'Returned' tuples over time. The y-axis ranges from 0 to 20000.
- Block I/O:** A line chart showing 'Reads' and 'Hits' over time. The y-axis ranges from 0 to 1800.

At the bottom, the 'Server activity' section is visible, with tabs for 'Sessions', 'Locks', and 'Prepared Transactions'. The 'Sessions' tab is active, displaying a table with the following data:

		PID	User	Application	Client	Backend start	State	Wait event	Blocking PIDs
⊗	■	▶	6212	postgres	pgAdmin 4 - DB:demo	::1	2020-12-24 12:07:40 MSK	active	



- > Indexes
- > RLS Policies
- > Rules
- > Triggers
- > ticket_flights
- > tickets
- > Trigger Functions
- > Types
- > Views
- > public
- > postgres
- > pubs
- > workbase
- > Login/Group Roles (10)
 - Vitaly
 - pg_execute_server_program
 - pg_monitor
 - pg_read_all_settings
 - pg_read_all_stats
 - pg_read_server_files
 - pg_signal_backend
 - pg_stat_scan_tables
 - pg_write_server_files
 - postgres
- > Tablespaces (2)
 - pg_default
 - pg_global



Account expires

Connection limit

Privileges

- Can login? Yes
- Superuser? No
- Create roles? No
- Create databases? No
- Update catalog? No
- Inherit rights from the parent roles? Yes
- Can initiate streaming replication and backups? No

Система управления pgAdmin

Membership

Roles

Roles shown with a check mark have the WITH ADMIN OPTION set.

psql

psql – терминальный клиент, который используется администраторами и разработчиками для интерактивной работы с PostgreSQL. Запросы (входной поток) могут быть получены из файла или интерактивно из командной строки.

Строка подключения:

```
psql -d database -h host -p port -U username
```

Для подключения к базе данных нужно знать имя базы данных, имя сервера, номер порта сервера и имя пользователя, под которым следует подключиться.

```
Командная строка - psql -d pubs
Пароль пользователя Vitaly:
psql (13.0)
ПРЕДУПРЕЖДЕНИЕ: Кодовая страница консоли (866) отличается от основной
                  страницы Windows (1251).
                  8-битовые (русские) символы могут отображаться некорректно.
                  Подробнее об этом смотрите документацию psql, раздел
                  "Notes for Windows users".
Введите "help", чтобы получить справку.

pubs=> \! chcp 1251
Текущая кодовая страница: 1251
pubs=> \l
```

Имя	Владелец	Кодировка	Список баз данных LC_COLLATE	LC_CTYPE	Права доступа
demo	postgres	UTF8	Russian_Russia.1251	Russian_Russia.1251	
postgres	postgres	UTF8	Russian_Russia.1251	Russian_Russia.1251	
pubs	postgres	UTF8	Russian_Russia.1251	Russian_Russia.1251	
template0	postgres	UTF8	Russian_Russia.1251	Russian_Russia.1251	=c/postgres + postgres=CtC/postgres
template1	postgres	UTF8	Russian_Russia.1251	Russian_Russia.1251	=c/postgres + postgres=CtC/postgres
workbase	postgres	UTF8	Russian_Russia.1251	Russian_Russia.1251	

```
(6 строк)

pubs=>
```

- ⇒ \? - список управляющих команд psql
- ⇒ \h - список команд языка sql
- ⇒ \l - вывести список баз данных
- ⇒ \c demo - подключиться к базе данных с указанным именем
- ⇒ \d - вывести список таблиц в базе данных
- ⇒ \q - выйти из программы psql

- ⇒ \dt - список всех таблиц в БД
- ⇒ \dn - список всех схем в БД
- ⇒ \df - список всех функций в БД
- ⇒ \du - список ролей БД

psql

```
testdb=> CREATE TABLE my_table (
testdb(> first integer not null default 0,
testdb(> second text)
testdb-> ;
CREATE TABLE
```

```
testdb=> \d my_table
           Таблица "my_table"
Атрибут | Тип      | Модификаторы
-----+-----+-----
first   | integer  | NOT NULL DEFAULT 0
second  | text     |
```

```
testdb=> SELECT * FROM my_table;
```

В командной строке пользователь может вводить команды SQL. Обычно введённые строки отправляются на сервер, когда встречается **точка с запятой**, завершающая команду. Конец строки не завершает команду. Это позволяет разбивать команды на несколько строк для лучшего понимания. Если команда была отправлена и выполнена без ошибок, то результат команды выводится на экран. Всё, что вводится в psql не взятое в кавычки и начинающееся с обратной косой черты, является метакомандой psql и обрабатывается самим psql. В Windows консоль по умолчанию поддерживает кодировку CP866, в связи с чем необходимо принудительно переключить кодовую страницу с помощью команды **\! chcp 1251** и установкой шрифтов *Lucida Console*.

- ❖ Механизм безусловной поддержки транзакций во всех режимах доступа к данным.
- ❖ Механизм контрольных точек (checkpoints), устанавливаемых через определённые интервалы времени или по определённым условиям.
- ❖ Журнализация событий в базе данных.
- ❖ Встроенные и внешние механизмы проверки целостности базы данных (запускаются периодически или вручную).
- ❖ Физическое дублирование файлов баз данных средствами операционной системы (RAID-массивы, специализированные дисковые контроллеры).
- ❖ Периодическое резервирование баз данных и журналов транзакций путём записи дампа базы данных на устройство резервирования (магнитную ленту или внешний диск).
- ❖ Репликация – возможность дублирования информации путём её периодической (в некоторых случаях – синхронной) передачи с одного сервера баз данных на другой.
- ❖ Эксклюзивный доступ со стороны СУБД к рабочим файлам баз данных.
- ❖ Шифрование трафика между клиентом и сервером, а также шифрование кодов, использованных для работы с объектами базы данных.

Спасибо за внимание!

