



SQL в свободно- распространяемых СУБД

НИЯУ МИФИ, ИФТЭБ, МНМЦ

КУРС ЛЕКЦИЙ

В.Ю. РАДЫГИН., Д.Ю. КУПРИЯНОВ ЛЕКЦИЯ 1

Структура лекции

Лекция состоит из трёх частей: вводной части, основной части и дополнительной части.

Вводная часть содержит базовые понятия теории баз данных. Её рекомендуется просто внимательно прочесть и осмыслить.

Основная часть посвящена знакомству с основами работы с СУБД PostgreSQL и языком SQL. Она также содержит описание базовых составляющих создания таблиц. С данной частью лекции рекомендуется не только ознакомиться, но и разобраться, применяя показанные примеры собственноручно на ЭВМ, так как в будущем на её основе будет предложена лабораторная работа.

Дополнительная часть лекции предложена для общего развития и не требует пристального изучения.

Часть 1. Введение в теорию баз данных

На сегодняшний день точного понятия базы данных не существует. В разных ситуациях под этим термином подразумевают разные вещи. Мы попробуем дать следующее абстрактное определение:

База данных – это набор порций информации, существующий в течение длительного времени.

Данное определение очень обширно, поэтому в нашем случае этот термин будет немного усложнен. Мы будем работать лишь с компьютерными базами данных, то есть с такими, взаимодействие с которыми осуществляется посредством специального управляющего программного обеспечения, называемого **системой управления базами данных** (СУБД).

Модели данных

Большинство объектов физического мира невероятно сложны по своей организации. Когда мы пытаемся описать какой-либо из таких объектов, мы на самом деле придумываем модель, соответствующую ему в нашем понимании. Если объекты можно поделить на некоторые группы, удовлетворяющие одинаковым моделям, то мы получаем ситуацию, когда внутри базы данных хранятся две группы сущностей:

- описания моделей объектов;
- записи, удовлетворяющие какой-либо из моделей и соответствующие различным представителям объектов.

Но бывают ситуации, когда объекты настолько различны, что их нельзя классифицировать. Тогда база данных представляет из себя набор из одних лишь моделей.

СВЯЗИ

Когда мы говорим о моделях данных, мы должны понимать, что нужно уметь хранить не только сами объекты, но и взаимосвязи между ними. Существует несколько видов взаимосвязей между объектами:

- один к одному – 1:1;
- один ко многим – 1:M;
- многие ко многим – M:M.

Определение модели

Модель данных – это абстрактное, самодостаточное, логическое определение объектов, связей между ними, применимых к ним действий (операторов) и прочих элементов, в совокупности составляющих абстрактную машину, с которой взаимодействует пользователь.

Разновидности моделей данных

За долгую историю развития баз данных было разработано много вариантов организации информации. Вот основные из них:

- иерархическая модель;
- сетевая модель;
- реляционная модель;
- объектная модель;
- другие модели.

На сегодняшний день актуальны, в основном, реляционная и объектная модели. Причем, хотя наибольшее распространение получила первая из них, обе модели имеют применение в современном программном обеспечении.

Реляционная модель

В **реляционной модели** каждый вид объектов описывается при помощи таблицы. Таблицы в реляционной модели принято называть отношениями (отсюда и название: отношение по-английски – relation).

Атрибуты отношения – это столбцы таблицы.

Каждая строка таблицы соответствует одному экземпляру объектов данного вида. Строки принято называть **кортежами** или **записями**.

Пример 1: таблица персоны

Персоны

Фамилия	Имя	Отчество	Пол	Дата рождения
Иванов	Иван	Иванович	Мужской	10.10.1990
Петров	Пётр	Петрович	Мужской	11.11.1988
Александрова	Александра	Александровна	Женский	03.03.1992
Сидоров	Сидор	Сидорович	Мужской	08.08.1991

Пример 2: связи

Персоны

Фамилия	Имя	Отчество	Пол	Идентификатор группы
Иванов	Иван	Иванович	Мужской	1
Петров	Пётр	Петрович	Мужской	2
Александрова	Александра	Александровна	Женский	2
Сидоров	Сидор	Сидорович	Мужской	1

Группы

Идентификатор	Название	Специальность
1	M16-356	38.04.01
2	M16-556	38.04.05

Ключ

Ключом называется набор полей, по значениям которых можно однозначно определить значения всех остальных полей строки. Если считать, что в таблице не бывает двух одинаковых строк, то можно сказать, что ключ обладает свойством уникальности. То есть значение ключа однозначно идентифицирует запись.

У одной таблицы может быть несколько ключей.

Первичный ключ

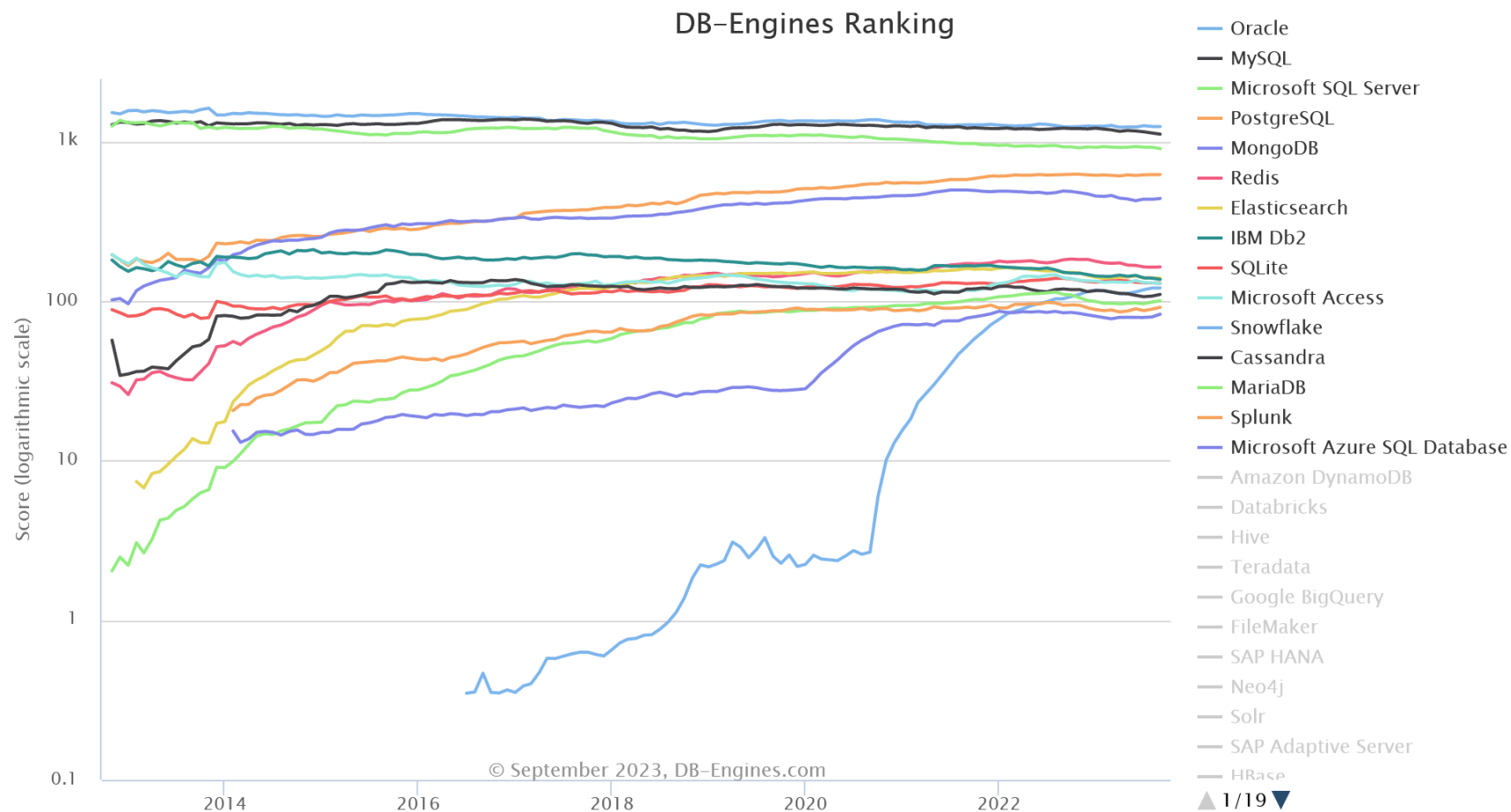
Первичный ключ — это наиболее часто используемый для поиска ключ таблицы. Каждая таблица может содержать только один первичный ключ.

Часть 2. Знакомство с СУБД PostgreSQL

СУБД PostgreSQL — это объектно-реляционная система управления базами данных с открытым исходным кодом, предусматривающая возможность легкого масштабирования и соответствующая стандартам ANSI/ISO.

По данным DB-Engines Ranking (<https://db-engines.com/en/ranking> [1]) PostgreSQL является 4-й по популярности в мире СУБД. Давайте взглянем на её конкурентов.

СУБД PostgreSQL в мире



СУБД PostgreSQL в мире

422 systems in ranking, September 2023

Rank			DBMS	Database Model	Score		
Sep 2023	Aug 2023	Sep 2022			Sep 2023	Aug 2023	Sep 2022
1.	1.	1.	Oracle +	Relational, Multi-model i	1240.88	-1.22	+2.62
2.	2.	2.	MySQL +	Relational, Multi-model i	1111.49	-18.97	-100.98
3.	3.	3.	Microsoft SQL Server +	Relational, Multi-model i	902.22	-18.60	-24.08
4.	4.	4.	PostgreSQL +	Relational, Multi-model i	620.75	+0.37	+0.29
5.	5.	5.	MongoDB +	Document, Multi-model i	439.42	+4.93	-50.21
6.	6.	6.	Redis +	Key-value, Multi-model i	163.68	+0.72	-17.79
7.	7.	7.	Elasticsearch	Search engine, Multi-model i	138.98	-0.94	-12.46
8.	8.	8.	IBM Db2	Relational, Multi-model i	136.72	-2.52	-14.67
9.	↑ 10.	↑ 10.	SQLite +	Relational	129.20	-0.72	-9.62
10.	↓ 9.	↓ 9.	Microsoft Access	Relational	128.56	-1.78	-11.47

Установка PostgreSQL

Так как PostgreSQL распространяется на основе свободной лицензии (PostgreSQL is released under the [PostgreSQL License](#), a liberal Open Source license, similar to the BSD or MIT licenses), то его можно свободно скачать с официального сайта:

<https://www.postgresql.org/download/> [4]

Сборки для windows представляются компанией EnterpriseDB (EDB):

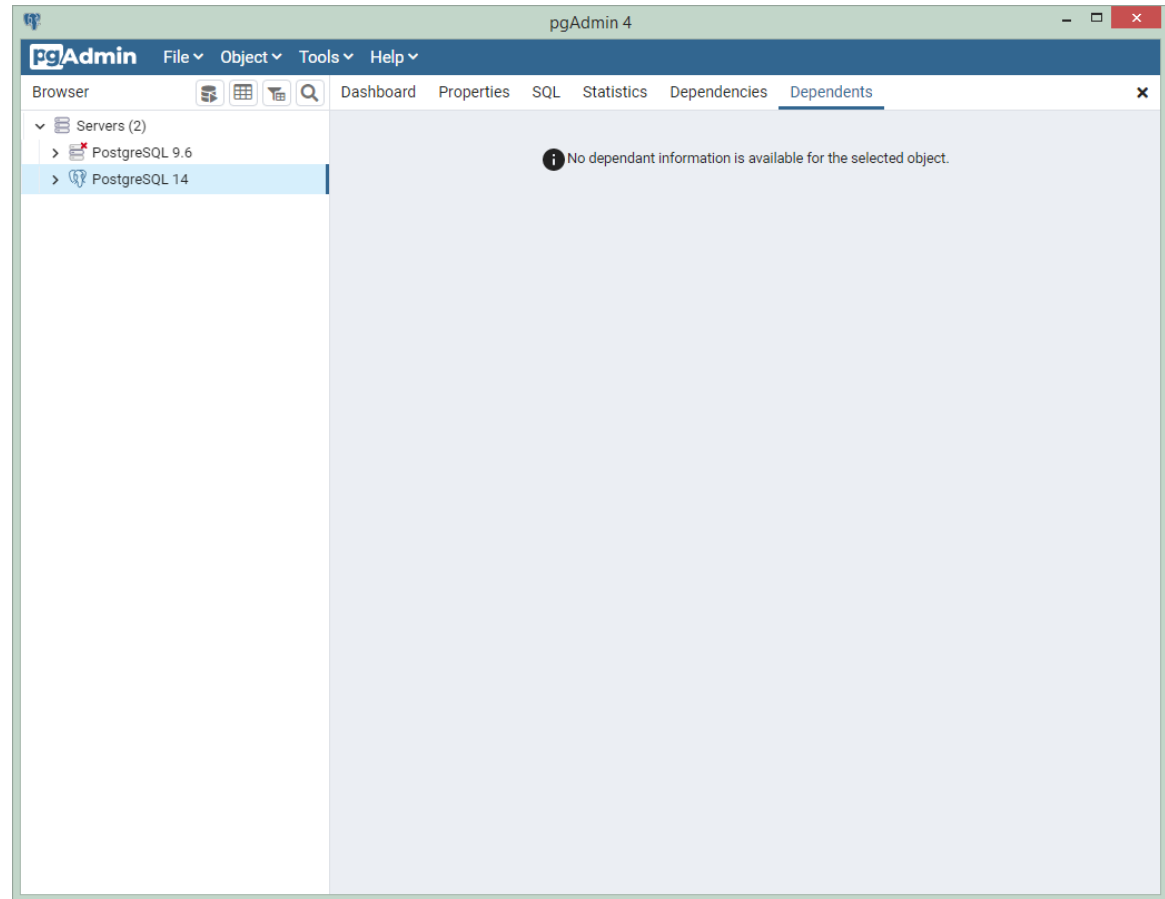
<https://www.enterprisedb.com/downloads/postgres-postgresql-downloads> [5]

На момент написания лекции текущая стабильная версия PostgreSQL – 16.

В лекциях мы будем стараться давать материалы для версий 14 и 15, так как на текущий момент именно они наиболее распространены в реальных проектах. Подробные материалы об установке PostgreSQL на Windows приведены в конце лекции.

Как мы будем работать с PostgreSQL?

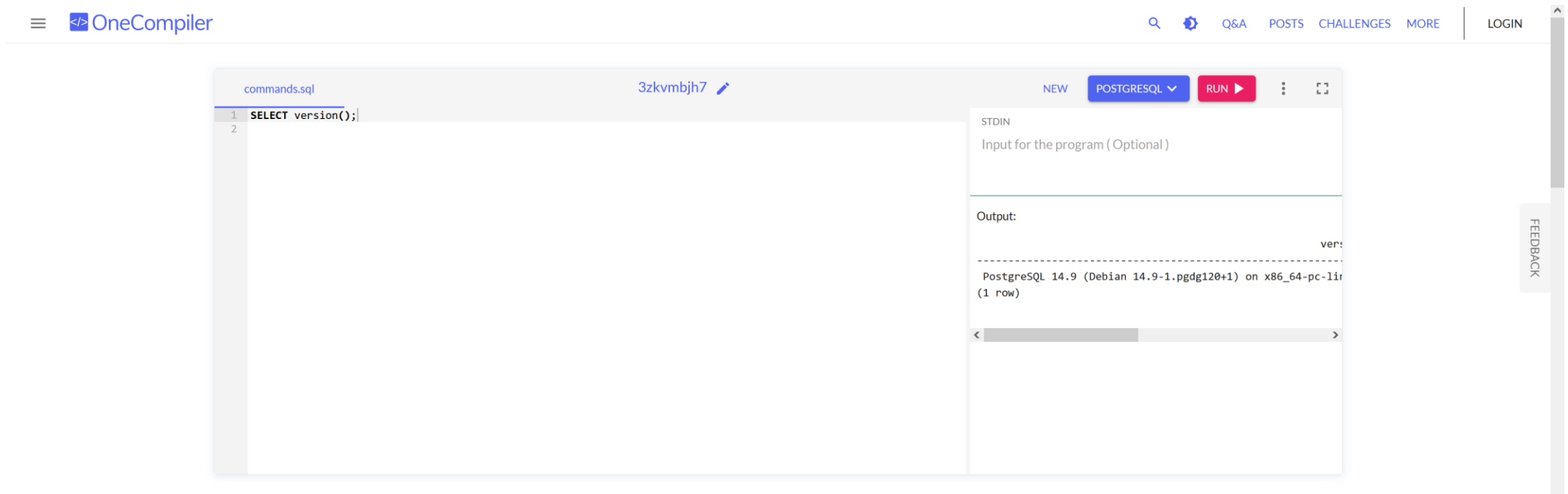
Для работы с PostgreSQL мы будем использовать входящее в его релиз программное приложение pgAdmin4, как основной вариант.



Как мы будем работать с PostgreSQL?

Если ставить PostgreSQL не хочется, то можно использовать online-ресурсы:

<https://onecompiler.com/postgresql/> [6]. На данный момент данный ресурс предоставляет PostgreSQL версии 14.9:



The screenshot shows the OneCompiler web interface for PostgreSQL. The top navigation bar includes the OneCompiler logo, a search icon, a settings icon, and links for Q&A, POSTS, CHALLENGES, MORE, and LOGIN. The main editor area has a tab labeled 'commands.sql' with the code 'SELECT version();'. To the right of the editor are buttons for 'NEW', 'POSTGRESQL' (with a dropdown arrow), and 'RUN' (with a play icon). Below these buttons is a 'STDIN' section for optional input. The 'Output' section displays the result of the query: 'PostgreSQL 14.9 (Debian 14.9-1.pgdg120+1) on x86_64-pc-lin (1 row)'. A vertical 'FEEDBACK' button is located on the far right.

Часть 3. Язык SQL

Язык SQL (Structured Query Language¹) – это основной язык взаимодействия с информацией, размещённой в реляционной базе данных. В той или иной модификации язык SQL есть во всех основных СУБД, но для каждой из них характерен свой диалект языка. Мы будем изучать язык SQL для СУБД PostgreSQL².

Язык SQL позволяет создавать и манипулировать объектами реляционной БД. Например, SQL позволяет создавать таблицы, модифицировать их, наполнять строками, модифицировать строки, удалять строки и т.д.

¹ – переводится как язык структурированных запросов.

² – кроме типа СУБД важна также его версия. В нашем курсе рекомендуется к использованию версия языка для СУБД PostgreSQL не менее 14. На большинстве занятий будет достаточно бесплатного облачного решения OneCompiler.

Составляющие языка SQL

Название	Назначение	Примеры команд
DDL (Data Definition Language)	Набор действий языка SQL, отвечающих за определение данных. То есть команды создания/удаления/модификации объектов. Например, таблиц.	CREATE TABLE DROP TABLE ALTER TABLE
DML (Data Manipulation Language)	Набор действий языка SQL, отвечающих за манипуляцию с данными. То есть команды, отвечающие за наполнение объектов или выборку из них. Например, добавление, удаление, выборка или модификация строк таблиц.	SELECT INSERT UPDATE DELETE
DCL (Data Control Language)	Набор действий языка SQL, отвечающих за разграничение прав доступ. Например, разрешение на выборку строк из таблицы.	GRANT REVOKE

Синтаксис языка

Допустимо размещение любого числа пробельных символов (пробел, табуляция, новая строка) между двумя любыми лексическими конструкциями языка.

Следующие два примера эквивалентны:

```
CREATE TABLE Test(id integer PRIMARY KEY, test varchar(16));
```

```
CREATE TABLE Test
(
    id            integer PRIMARY KEY,
    test         varchar(
                        16
                    ));
```

Синтаксис языка

Каждое выражение должно чем-то завершаться. В разных программных окружениях могут быть разные соглашения по этому вопросу. В pgAdmin4 (а также в терминале PostgreSQL psql) и в OneCompiler каждое выражение должно оканчиваться символом точки с запятой «;».

Однострочные комментарии обычно указываются при помощи двух символов минус: «--». Многострочные комментарии можно указывать при помощи комбинации «/*» «*/». Комментарии можно размещать в любом месте:

```
-- Создание таблицы
CREATE TABLE Test(
    -- Поле 1
    id integer PRIMARY KEY,
    -- Поле 2
    test varchar(16));
/* Удаление
   таблицы */
DrOp TABLE "test";
```

Синтаксис языка

Обычно язык SQL нечувствителен к регистру букв¹. Основные команды можно писать как угодно. Но хорошим стилем, считается их написание заглавными буквами. Имена объектов (например, таблиц и полей), если они указаны без кавычек, автоматически преобразуются к нижнему регистру букв. Например, следующие два примера эквивалентны:

```
CREATE TABLE test(  
    id integer PRIMARY KEY,  
    info char(16));
```

```
CrEaTe table TEST(  
    Id integer PRIMARY KEY,  
    iNfo char(16));
```

¹ — это может быть изменено системными настройками.

Синтаксис языка

Если название объекта написано в двойных кавычках, то оно рассматривается именно в том виде, в котором написано. Например, следующие три примера создают три разных таблицы:

```
CREATE TABLE "TEST1" (  
    id integer PRIMARY KEY,  
    info char(16));
```

```
CREATE TABLE "Test1" (  
    id integer PRIMARY KEY,  
    info char(16));
```

```
CREATE TABLE test1 (  
    id integer PRIMARY KEY,  
    info char(16));
```


Правила задания имён объектов

При работе с базой данных очень часто необходимо подбирать имена для различных создаваемых в ней объектов (например, для таблиц или колонок в них). При этом необходимо придерживаться правила задания имён. Эти правила очень просты.

1. Имя может содержать только символы, являющиеся заглавными или строчными буквами английского алфавита (A-Z и a-z), символы, являющиеся цифрами (0-9) или следующие специальные символы: «_», «\$».
2. Первый символ имени должен быть буквой или «_».
3. Длина имени не должна превышать 63 байта по умолчанию (можно изменить в настройках).
4. Имя не может совпадать с именами других объектов, принадлежащих тому же пользователю.
5. Имя не может совпадать с зарезервированными словами.

Создание таблиц

Для создания таблиц в языке SQL используется команда CREATE TABLE. Кратко её синтаксис можно описать следующим образом:

```
CREATE TABLE [схема.]имя_таблицы (  
    имя_колонки    тип_данных    [DEFAULT    выражение]    [ограничение  
целостности]  
    [, ...]  
);
```

Здесь и далее квадратные скобки обозначают необязательность присутствия их содержимого. Фигурные скобки, содержащие несколько значений, разделённых вертикальной чертой, означают выбор одного из нескольких вариантов. Многоточие означает возможное повторение 0 или более раз предыдущей конструкции.

Пример 3

```
CREATE TABLE myuser.test(  
    id integer PRIMARY KEY,  
    info varchar(16) DEFAULT 'No info' NOT NULL  
);
```

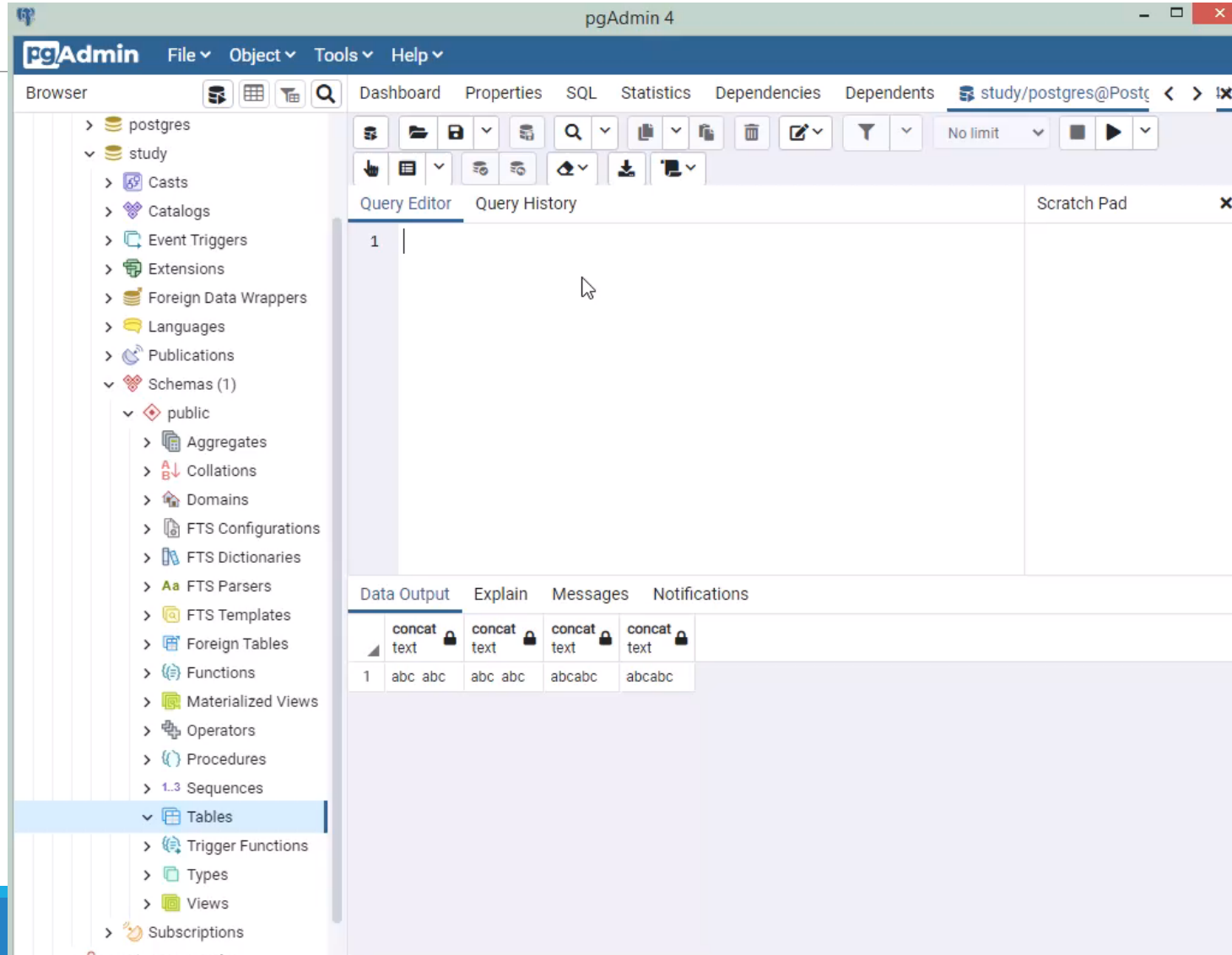
Типы данных

Тип данных - это множество значений, которые может принимать соответствующий ему объект. Например, если мы говорим, что колонка `digit` имеет тип данных целое число из одной цифры (в PostgreSQL это тип данных `numeric(1)` или `numeric(1, 0)`), то это означает, что данная колонка может хранить только одно значение из следующих 19: -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. При этом данная колонка не может хранить никакие другие значения (например, 'd', 11, -55). Исключение составляет специальная конструкция `NULL` – отсутствие значения.

Строковые типы данных

Тип данных	Описание	Примеры
char(размер)	Набор символов постоянной длины	code char(3), sex char
varchar(размер)	Набор символов переменной длины	last_name varchar(64)
text (или varchar без размера или character varying)	Набор символов произвольной длины	description text, info varchar, info2 character varying

Разница между CHAR и VARCHAR



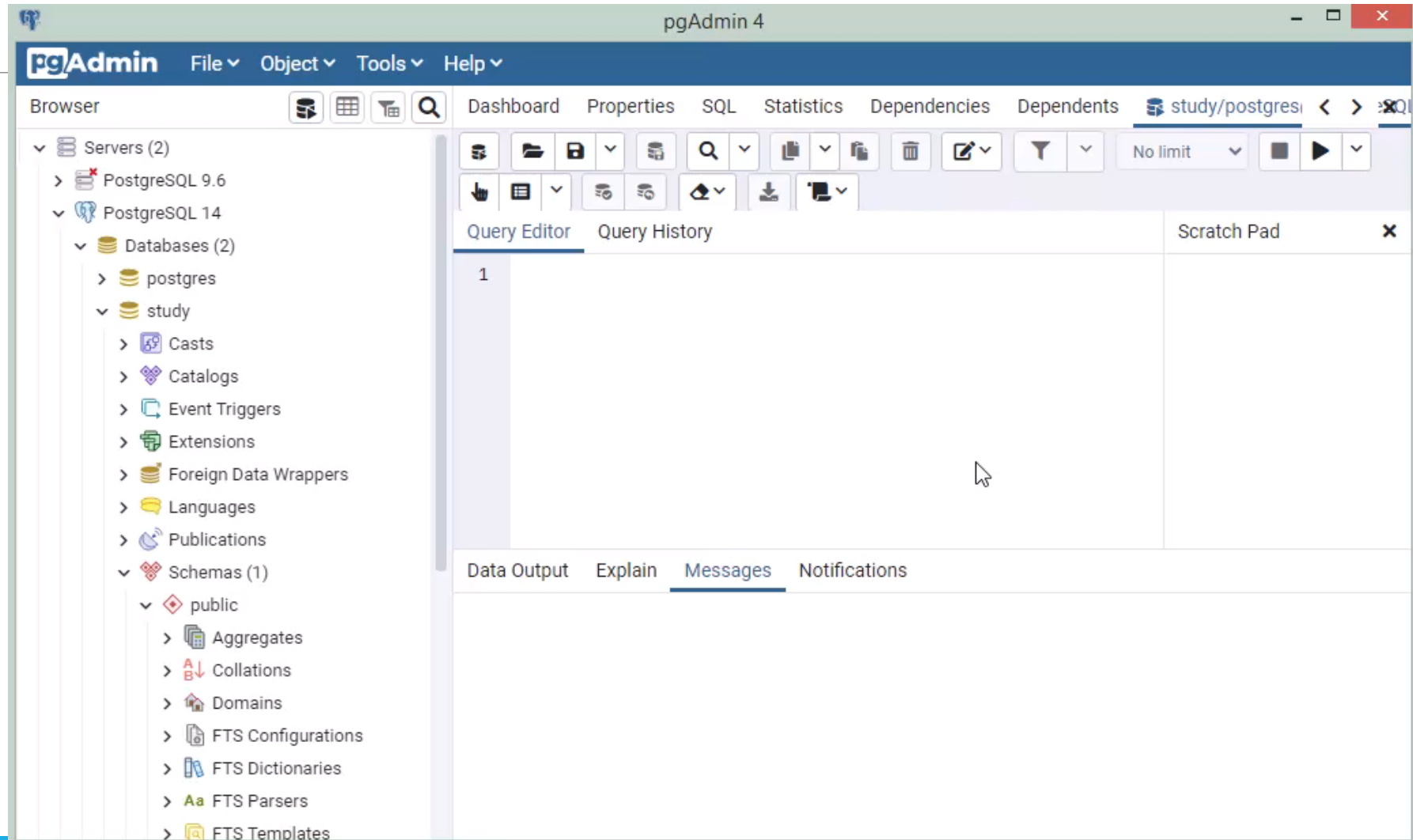
The screenshot shows the pgAdmin 4 web interface. The left sidebar displays a tree view of the database structure, with the 'public' schema selected. The main area is divided into a 'Query Editor' and a 'Data Output' section. The 'Query Editor' contains a single line of SQL code: `1 |`. The 'Data Output' section shows the results of the query, which is a single row with four columns, all of type 'concat text'. The results are: 'abc abc', 'abc abc', 'abcbc', and 'abcbc'.

	concat text	concat text	concat text	concat text
1	abc abc	abc abc	abcbc	abcbc

Числовые типы данных

Тип данных	Описание	Примеры
numeric(p) numeric(p, s) decimal(p) decimal(p, s)	Число, точностью в p цифр всего и масштабом в s после запятой (до 131072 до запятой и до 16383 после запятой). Точность – это число всех цифр. Масштаб – это число цифр после десятичной запятой (точки).	f1 numeric(5), f2 numeric(5, 3), f3 decimal(2, 2), f4 numeric
smallint	Целое число от -32768 до 32767	f1 smallint
integer	Целое число от -2147483648 до 2147483647	f1 integer
bigint	Целое число от -9223372036854775808 до 9223372036854775807	f1 bigint
real	4-х байтовое вещественное машинное число	f1 real
double precision (или float на 64-битных машинах)	8-й байтовое вещественное машинное число	f1 double precision

Точность и масштаб



Дата и время

Тип данных	Описание	Примеры
date	Дата от 4713 года до нашей эры до 294276 года нашей эры	created_at date
time, time without time zone	Время без указания часового пояса в микросекундах	arrival time
time with time zone	Время с указанием часового пояса	departure time with time zone
timestamp, timestamp without time zone	Дата и время без указания часового пояса	created_at timestamp
timestamp with time zone	Дата и время с указанием часового пояса	updated_at timestamp with time zone
interval	Разница между двумя timestamp	period interval

Пример 4

Напишем SQL-код для создания таблицы, описанной в примере 1:

```
CREATE TABLE people(  
    last_name varchar(64) ,  
    first_name varchar(64) ,  
    second_name varchar(64) ,  
    sex varchar(8) , -- очень плохое решение  
    birthday date  
);
```

Немного оптимизируем

В примере 4 таблица из примера 1 реализована таким образом, чтобы она могла содержать те же самые значения. Но такая реализация поля для хранения пола человека очень неэффективна.

Физический пол человека — это величина, которая может принимать только два значения: мужской и женский. Из базового курса информатики мы знаем, что в этом случае для представления данной информации достаточно одного бита. Ближайший к биту тип данных — это один символ (один байт). Поэтому гораздо правильнее будет закодировать пол одним символом (либо числом из одной цифры).

Пример 5

SQL-код для оптимизированной таблицы из примера 4:

```
CREATE TABLE people(  
    last_name varchar(64) ,  
    first_name varchar(64) ,  
    second_name varchar(64) ,  
    sex char ,  
    birthday date  
);
```

Пример заполнения таблицы

	Data Output	Explain	Messages	Notifications	
	last_name character varying (64) 🔒	first_name character varying (64) 🔒	second_name character varying (64) 🔒	sex character (1) 🔒	birthday date 🔒
1	Иванов	Иван	Иванович	m	1990-10-10
2	Петров	Пётр	Петрович	m	1998-11-11
3	Александрова	Александра	Александровна	f	1992-03-03
4	Сидоров	Сидор	Сидорович	m	1991-08-08

Ограничения целостности

Очень часто бывает, что человек, создавший таблицу, предполагал, что её будут заполнять одними значениями, в то время как злоумышленник или просто неаккуратный пользователь может заполнить её совершенно по-другому. Чтобы избежать таких ситуаций, в таблицах предусмотрен механизм проверки целостности данных. Или, говоря другими словами, механизм проверки «правильности» вносимых значений.

Чтобы механизм проверки целостности заработал, необходимо при создании таблицы описать выполняющиеся для неё ограничения целостности данных (constraints). Для этого есть набор специальных конструкций.

Базовые ограничения целостности

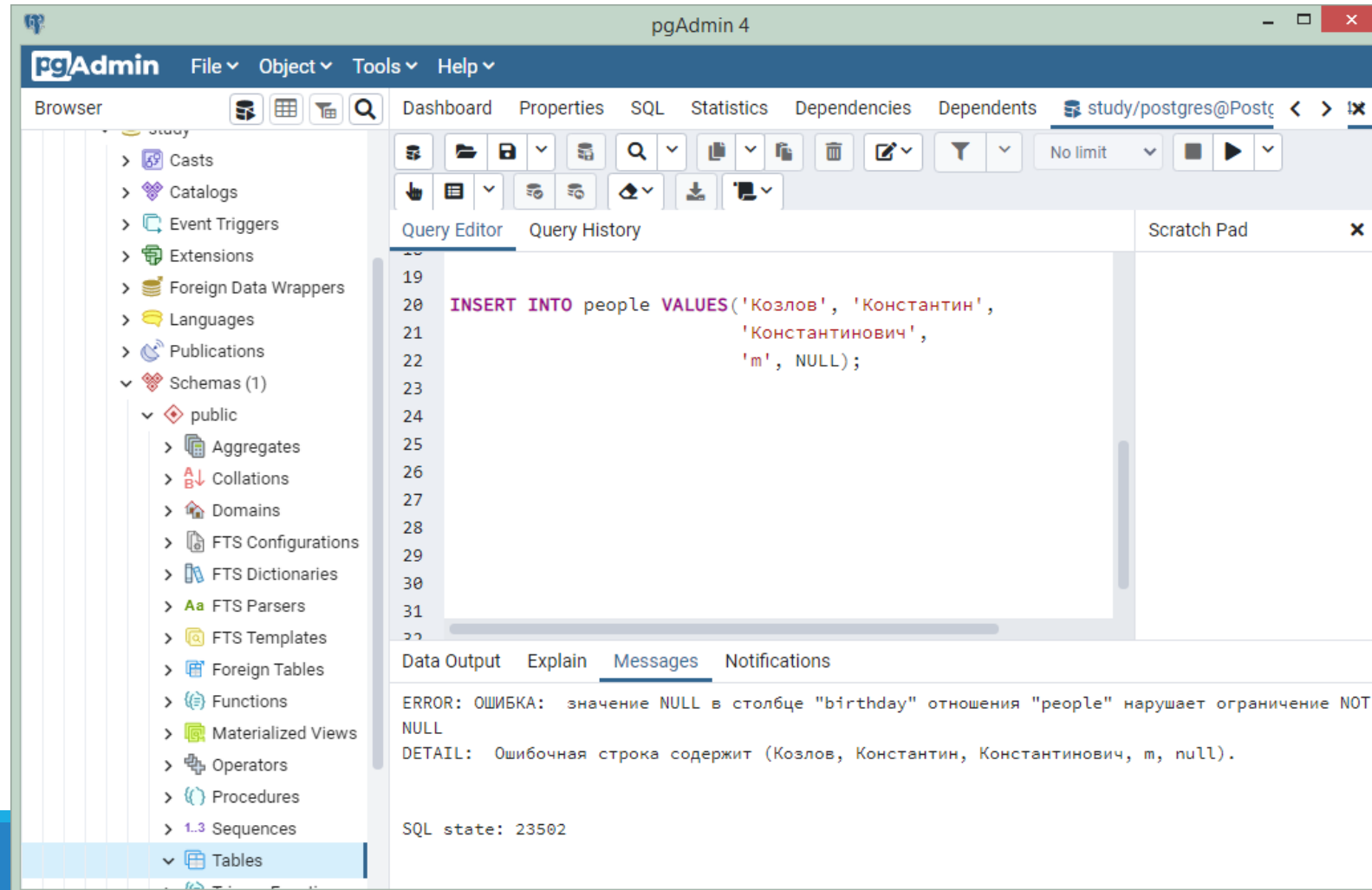
Тип данных	Описание
NOT NULL	Запрет на запись пустого значения (NULL).
UNIQUE	Запрет на запись повторяющихся значений. Фактически, это задание альтернативного (вторичного) ключа таблицы.
PRIMARY KEY	Задание первичного ключа таблицы. Фактически – это комбинация конструкций NOT NULL и UNIQUE.
CHECK	Запрет на добавление значений, не удовлетворяющих определённому условию. Условие задаётся в виде функции над множеством значений полей строки таблицы, действующей на пространство {true false}. Или, говоря простым языком, условие задаётся как какое-то выражение, состоящее из значений полей добавляемой/изменяемой строки, операторов и функций. Значение выражения должно быть истина или ложь.
FOREIGN KEY	Внешний ключ, контролирующий, чтобы значение данного поля (комбинации полей) строки всегда соответствовало одному из значений ключа (первичного или вторичного) какой-то (указывается) другой таблицы.

NOT NULL

Разберём все ограничения целостности на примерах. Пусть в нашем примере 5 часть полей обязательно должна быть заполнена. То есть эти поля не могут быть пустыми. Тогда мы можем ввести данные ограничения при помощи NOT NULL. К примеру, пусть не могут быть пустыми поля last_name, sex и birthday:

```
CREATE TABLE people(  
    last_name varchar(64) NOT NULL,  
    first_name varchar(64),  
    second_name varchar(64),  
    sex char NOT NULL,  
    birthday date NOT NULL  
);
```


Что произойдёт, если нарушим?



UNIQUE

Пусть в нашем примере 5 не может быть людей с одинаковой датой рождения. Тогда мы можем ввести данное ограничение при помощи UNIQUE.

```
CREATE TABLE people(  
    last_name varchar(64) NOT NULL,  
    first_name varchar(64) ,  
    second_name varchar(64) ,  
    sex char NOT NULL,  
    birthday date NOT NULL UNIQUE  
);
```

Лучше задавать имена!

Каждому ограничению целостности PostgreSQL автоматически даёт имена. Зачем это нужно?

Однажды заданное ограничение целостности может оказаться через некоторое время ненужным или недостаточным. Чтобы его удалить или модифицировать, потребуется выделить его из множества других ограничений целостности. Для этого потребуется использовать имя ограничения.

Конечно, кто-то может сказать, что проще удалить таблицу и создать её заново, уже с правильным ограничением. Но что делать, если таблица уже заполнена?

Системные имена не всегда удобны для использования. PostgreSQL пытается их сделать осмысленными, например, `people_birthday_key`. Но часто они похожи, и непонятно, с чем работать. Поэтому ограничениям целостности лучше давать собственные имена при помощи конструкции `CONSTRAINT`.

С именем

```
CREATE TABLE people(  
    last_name varchar(64) NOT NULL,  
    first_name varchar(64) ,  
    second_name varchar(64) ,  
    sex char NOT NULL,  
    birthday date CONSTRAINT birthday_u UNIQUE  
);
```

Другая форма UNIQUE

Пусть в нашем примере 5 вместо уникальности даты рождения введено требование отсутствия полных тезок. То есть не должно быть записей, у которых одновременно попарно совпадают поля `last_name`, `first_name`, `second_name`. Это тоже можно решить при помощи `UNIQUE`:

```
CREATE TABLE people(  
    last_name varchar(64) NOT NULL,  
    first_name varchar(64),  
    second_name varchar(64),  
    sex char NOT NULL,  
    birthday date NOT NULL,  
    UNIQUE(last_name, first_name, second_name)  
);
```

PRIMARY KEY

Первичный ключ — это комбинация конструкций NOT NULL и UNIQUE. Представим, что у нас в примере 5 поля last_name, first_name и birthday всегда должны быть заполнены и не могут повторяться в совокупности. Мы можем это описать как при помощи UNIQUE и NOT NULL, так и при помощи PRIMARY KEY:

```
CREATE TABLE people(  
    last_name varchar(64),  
    first_name varchar(64),  
    second_name varchar(64),  
    sex char NOT NULL,  
    birthday date,  
    PRIMARY KEY(last_name, first_name, birthday)  
);
```

Индексы

При использовании ограничений целостности UNIQUE или PRIMARY KEY для охватываемого ими набора полей автоматически создаётся индексная структура (индекс). Индекс – это специальный механизм ускорения поиска.

```
CREATE TABLE countries(  
    code varchar(3) PRIMARY KEY,  
    full_name varchar(64) NOT NULL UNIQUE  
);
```

CHECK

Ограничим в примере 5 множество значений поля пол (sex) только двумя вариантами: «f» (от английского слова female) и «m» (от английского слова male). Для этого используем check:

```
CREATE TABLE people(  
    last_name varchar(64),  
    first_name varchar(64),  
    second_name varchar(64),  
    sex char NOT NULL CHECK(sex = 'f' OR sex = 'm'),  
    birthday date,  
    PRIMARY KEY(last_name, first_name, birthday)  
);
```


Усложним пример 5

Пусть у каждого человека есть гражданство (citizenship), задаваемое кодом страны. Для того, чтобы оператор не мог ошибиться, вводя коды стран, создадим новую таблицу страны (countries) с двумя полями: код (code) и полное название (full_name):

```
CREATE TABLE countries(  
    code varchar(3) PRIMARY KEY,  
    full_name varchar(64) NOT NULL UNIQUE  
);
```

FOREIGN KEY

Теперь, чтобы избежать ошибки оператора, мы можем запретить записывать в поле citizenship таблицы people значения, которым нет соответствия среди значений поля code таблицы countries:

```
CREATE TABLE people(  
    last_name varchar(64),  
    first_name varchar(64),  
    second_name varchar(64),  
    sex char NOT NULL CHECK(sex = 'f' OR sex = 'm'),  
    birthday date,  
    PRIMARY KEY(last_name, first_name, birthday),  
    citizenship varchar(3) REFERENCES countries(code)  
);
```

FOREIGN KEY

Фактически, в примере 6 мы создали смысловую связь между таблицами страны и люди. Одной стране соответствует много граждан. Это связь вида один ко многим (1:M). Различают также связи один к одному (1:1) и многие ко многим (M:M).

Внешний ключ может ссылаться только на ключ (первичный или вторичный).

Ссылаться можно только на уже созданную таблицу или на саму создаваемую таблицу (на саму себя).

Другая форма

```
CREATE TABLE people(  
    last_name varchar(64),  
    first_name varchar(64),  
    second_name varchar(64),  
    sex char NOT NULL CHECK(sex = 'f' OR sex = 'm'),  
    birthday date,  
    citizenship varchar(3),  
    PRIMARY KEY(last_name, first_name, birthday),  
    CONSTRAINT code_fk FOREIGN KEY(citizenship) REFERENCES countries(code)  
);
```

Несколько полей

Для демонстрации внешнего ключа с несколькими полями модифицируем таблицу groups и добавим в таблицу people связующий ключ:

```
CREATE TABLE groups (  
    name varchar(16) NOT NULL,  
    study_year varchar(16) NOT NULL,  
    speciality varchar(8) NOT NULL,  
    PRIMARY KEY(name, study_year)  
);
```

Несколько полей

```
CREATE TABLE people(  
    last_name varchar(64),  
    first_name varchar(64),  
    second_name varchar(64),  
    sex char NOT NULL CHECK(sex = 'f' OR sex = 'm'),  
    birthday date,  
    citizenship varchar(3) REFERENCES countries(code),  
    group_name varchar(16) NOT NULL,  
    study_year varchar(16) NOT NULL,  
    PRIMARY KEY(last_name, first_name, birthday),  
    FOREIGN KEY(group_name, study_year) REFERENCES groups(name, study_year)  
);
```

Значения по умолчанию

Если, добавляя в таблицу строку, мы оставим какие-либо поля незаполненными, то в данных колонках будет размещено отсутствие значения – NULL. Это не всегда удобно. Очень часто встречаются ситуации, когда колонка заполняется преимущественно одним, отличным от NULL значением. Другие варианты заполнения данного поля бывают, но их частота возникновения достаточно мала. В этом случае было бы удобно, чтобы при отсутствии указания значения данного поля оно заполнялось наиболее часто встречающимся вариантом. Для решения данной задачи в таблицах для колонок предусмотрены значения по умолчанию, устанавливаемые с помощью конструкции DEFAULT.

Пример

Установим для каждого человека код страны по умолчанию равным RUS:

```
CREATE TABLE people(  
    last_name varchar(64),  
    first_name varchar(64),  
    second_name varchar(64),  
    sex char NOT NULL CHECK(sex = 'f' OR sex = 'm'),  
    birthday date,  
    PRIMARY KEY(last_name, first_name, birthday),  
    citizenship varchar(3) DEFAULT 'RUS' REFERENCES countries(code)  
);
```


Удаление таблиц

Для удаления таблиц предусмотрена команда DROP TABLE. Её общий синтаксис следующий:

```
DROP TABLE [схема.] имя_таблицы;
```

Пример:

```
DROP TABLE people;
```

Модификация таблиц

Для модификации таблиц «на лету», когда нет возможности их удаления и пересоздания, используется команда `ALTER TABLE`. Она позволяет изменять название таблицы, атрибутов и ограничений целостности, а также добавлять/удалять/изменять столбцы и ограничения целостности.

Переименование таблиц

Для переименования таблиц, атрибутов и ограничений целостности предусмотрена команда **ALTER TABLE** с опцией **RENAME TO**. Синтаксис для переименования таблиц следующий:

```
ALTER TABLE [IF EXISTS] старое_имя_таблицы RENAME TO  
новое_имя_таблицы;
```

Пример:

```
ALTER TABLE people RENAME TO staff;
```

Переименование атрибутов

Синтаксис для переименования атрибутов таблицы следующий:

```
ALTER TABLE [IF EXISTS] имя_таблицы RENAME [COLUMN]  
строе_имя_столбца TO новое_имя_столбца;
```

Пример:

```
ALTER TABLE countries RENAME COLUMN code TO code3;
```

Переименование ограничений целостности

Синтаксис для переименования ограничений целостности таблицы следующий:

```
ALTER TABLE [IF EXISTS] имя_таблицы RENAME CONSTRAINT  
старое_имя_ограничения TO новое_имя_ограничения;
```

Пример:

```
ALTER TABLE people RENAME CONSTRAINT birthday_u TO birthday_uniq;
```

Модификация атрибутов

Для модификации атрибутов таблицы используется команда ALTER TABLE с различными опциями:

```
ALTER TABLE [ IF EXISTS ] имя_таблицы действие [, ... ]
```

действия для атрибутов:

- **ADD [COLUMN] [IF NOT EXISTS]** имя_столбца тип_данных [ограничение_столбца [...]]
- **DROP [COLUMN] [IF EXISTS]** имя_столбца [**RESTRICT** | **CASCADE**]
- **ALTER [COLUMN]** имя_столбца [**SET DATA**] **TYPE** тип_данных
- **ALTER [COLUMN]** имя_столбца **SET DEFAULT** выражение
- **ALTER [COLUMN]** имя_столбца **DROP DEFAULT**
- **ALTER [COLUMN]** имя_столбца { **SET** | **DROP** } **NOT NULL**

Модификация ограничений целостности

Для модификации ограничений целостности таблицы используется команда ALTER TABLE с различными опциями:

```
ALTER TABLE [ IF EXISTS ] имя_таблицы действие [ , ... ]
```

Действия для ограничений целостности:

- **ADD** ограничение_таблицы [**NOT VALID**]
- **ADD** ограничение_таблицы_по_индексу /для PRIMARY KEY и UNIQUE/
- **VALIDATE CONSTRAINT** имя_ограничения
- **DROP CONSTRAINT [IF EXISTS]** имя_ограничения [**RESTRICT** | **CASCADE**]

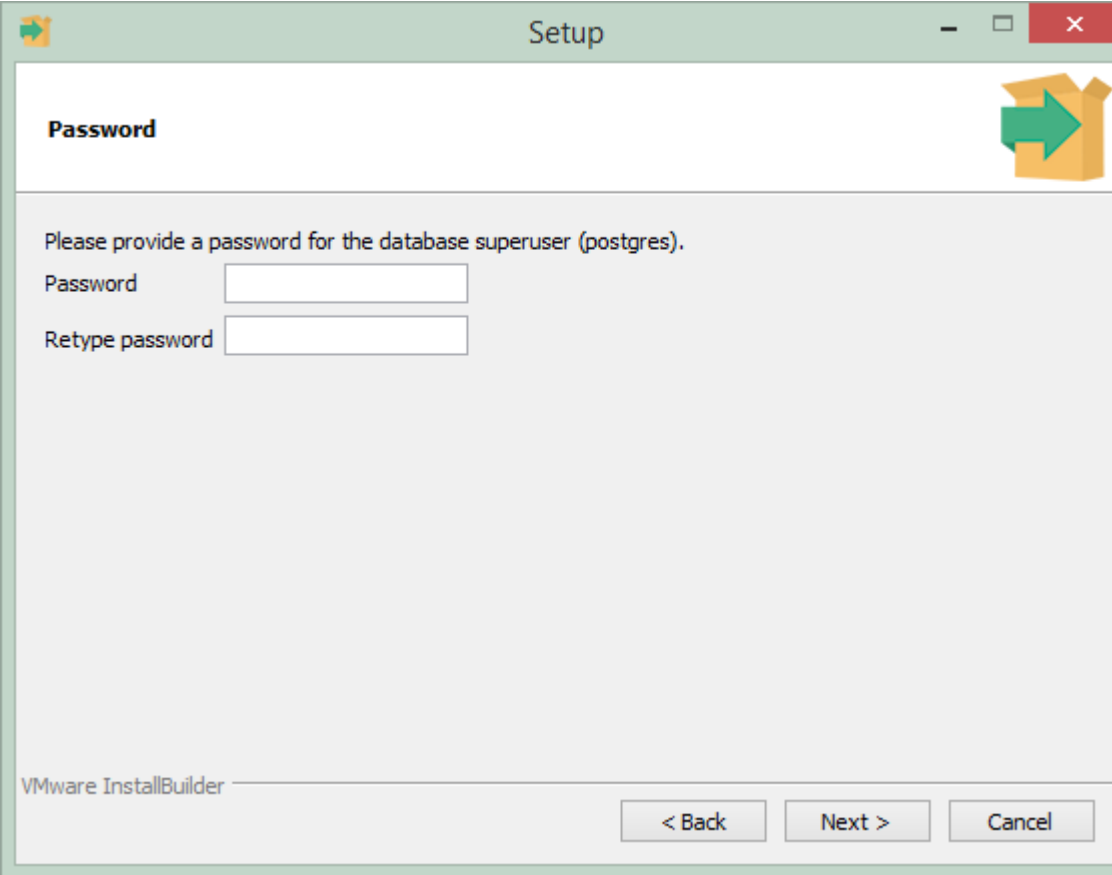
Примеры

```
ALTER TABLE people ADD passport varchar(10) UNIQUE;  
ALTER TABLE people ALTER COLUMN sex TYPE char;  
ALTER TABLE people ALTER COLUMN sex SET DEFAULT 'm';  
ALTER TABLE people DROP passport;  
ALTER TABLE people ALTER sex SET NOT NULL;  
ALTER TABLE people ADD CHECK(sex = '0' OR sex = '1') NOT VALID;  
ALTER TABLE people VALIDATE CONSTRAINT people_sex_check;  
ALTER TABLE people DROP CONSTRAINT IF EXISTS some_constraint;  
    ◦ /* ЗАМЕЧАНИЕ: ограничение "some_constraint" в таблице "people" не существует,  
    ◦ пропускается ALTER TABLE */
```


Часть 4. Установка PostgreSQL

Важные моменты установки

При первоначальной установке Вы вводите пароль суперпользователя. Не забудьте его и убедитесь, что используете правильную раскладку клавиатуры при его вводе.



The screenshot shows a window titled "Setup" with a green header bar. In the top right corner of the window is a yellow box icon with a green arrow pointing right. The main content area has a title "Password" in bold. Below it, the text "Please provide a password for the database superuser (postgres)." is displayed. There are two input fields: "Password" and "Retype password". At the bottom left, the text "VMware InstallBuilder" is visible. At the bottom right, there are three buttons: "< Back", "Next >", and "Cancel".

Setup

Password

Please provide a password for the database superuser (postgres).

Password

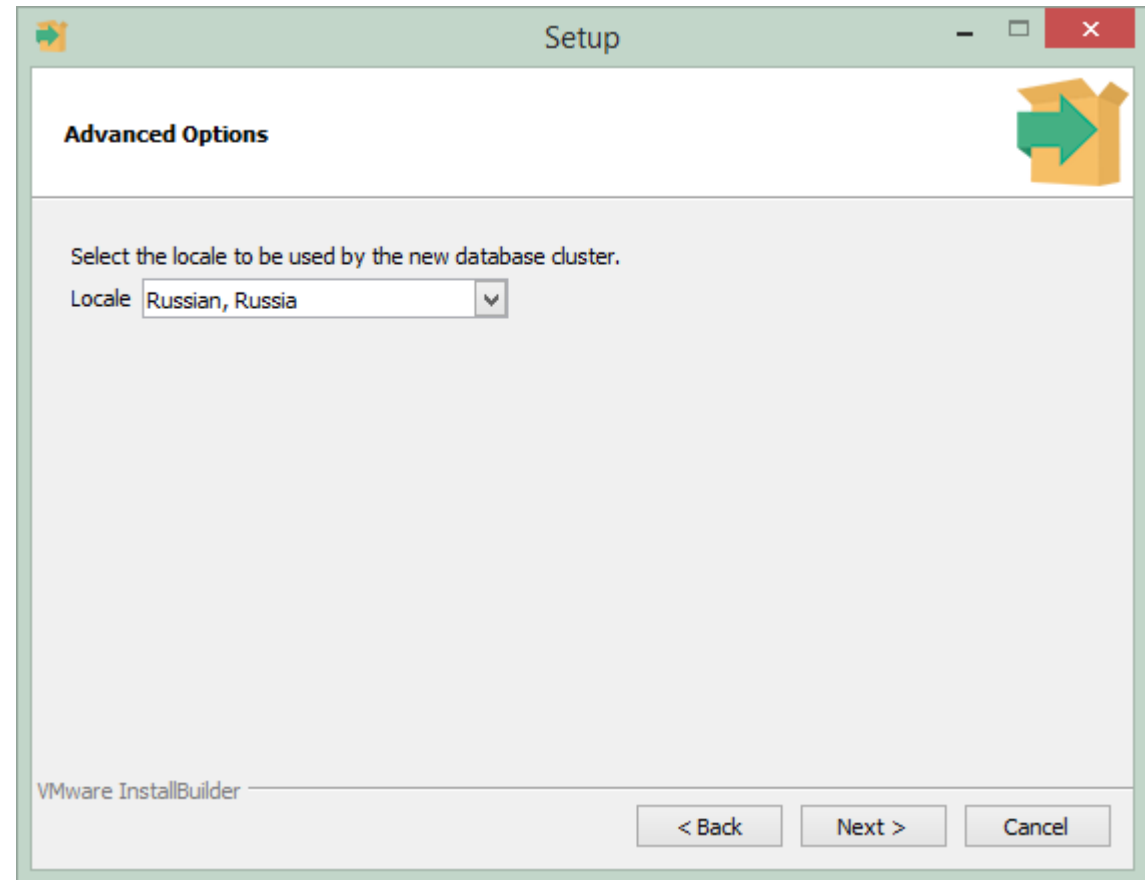
Retype password

VMware InstallBuilder

< Back Next > Cancel

Правильная локализация

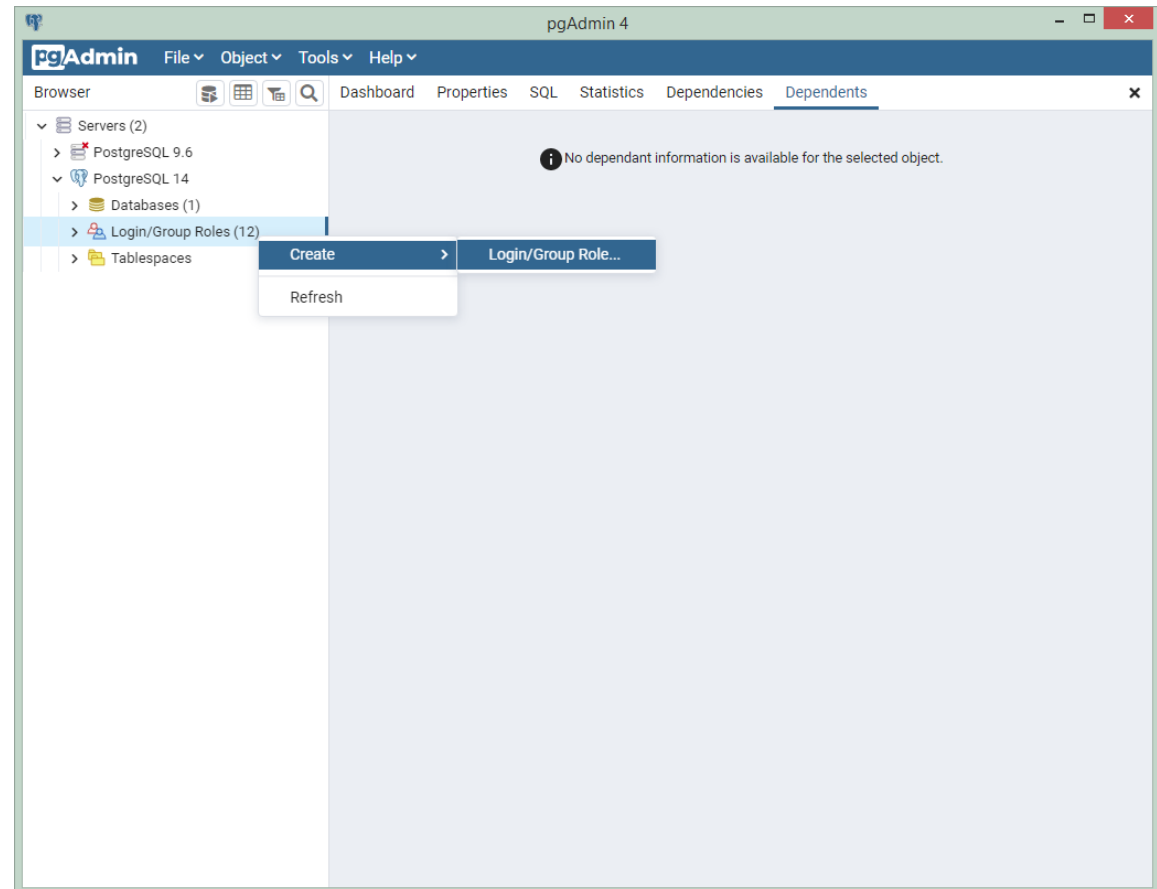
Не забудьте выбрать
Российскую локализацию
(кодировка, время, запись чисел и
т.д.).



Подготовка к работе на домашнем компьютере

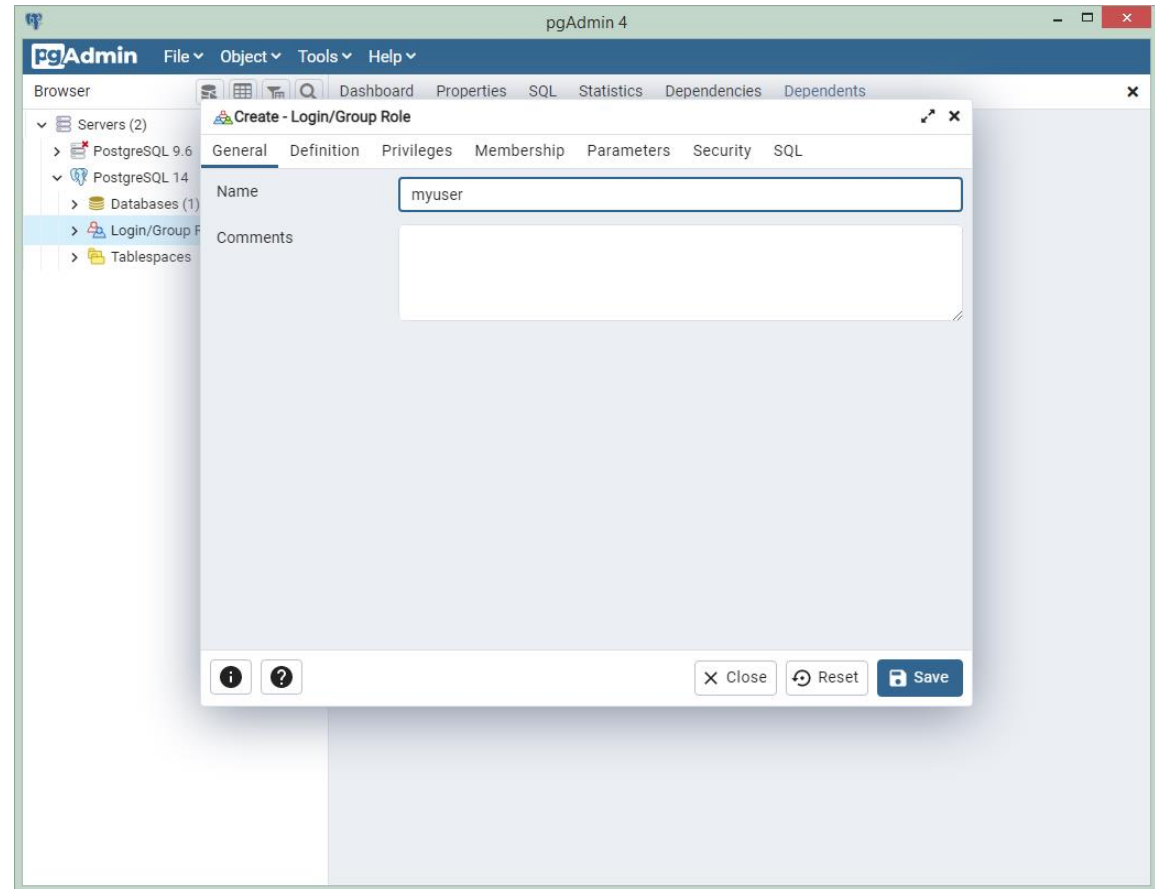
Создадим пользователя, отличного от postgres.

пользователя, суперпользователя



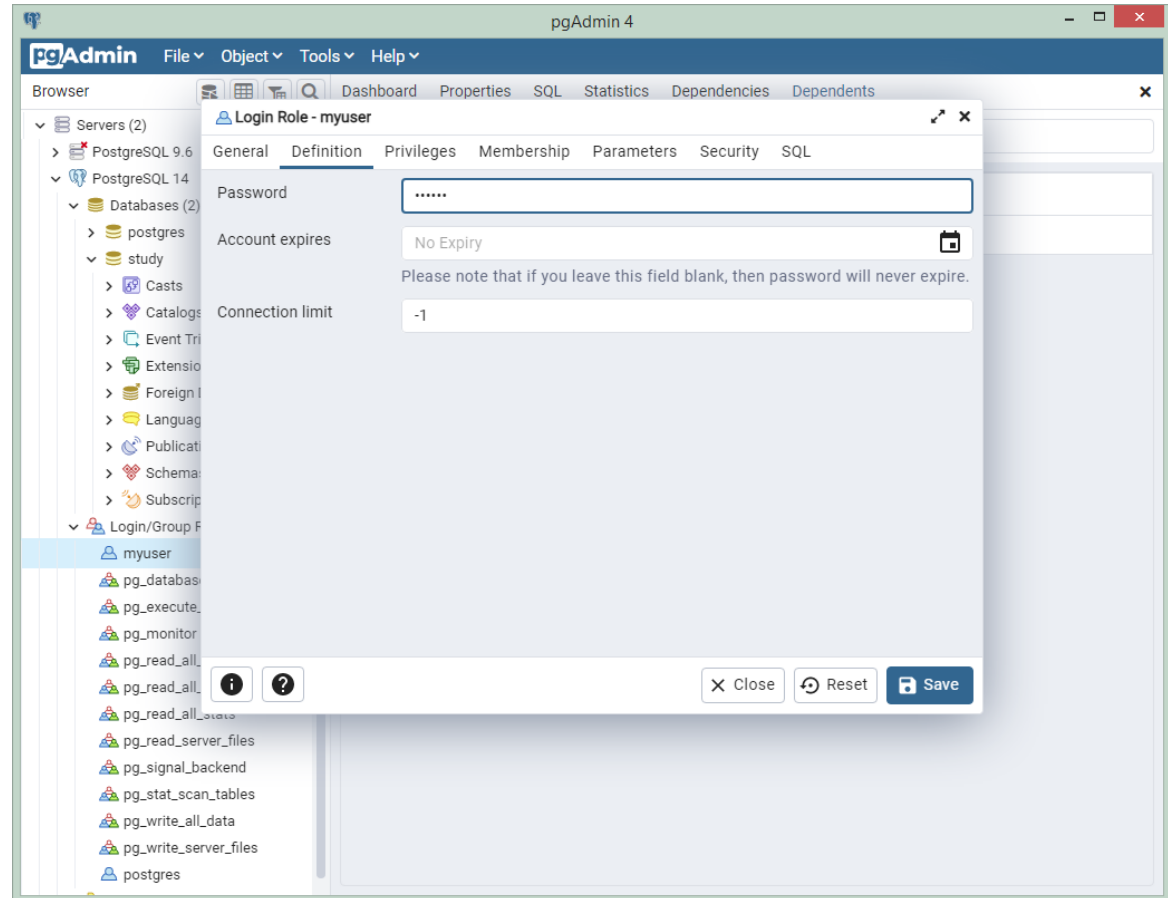
Подготовка к работе на домашнем компьютере

Вводим логин нового пользователя (на слайде myuser).



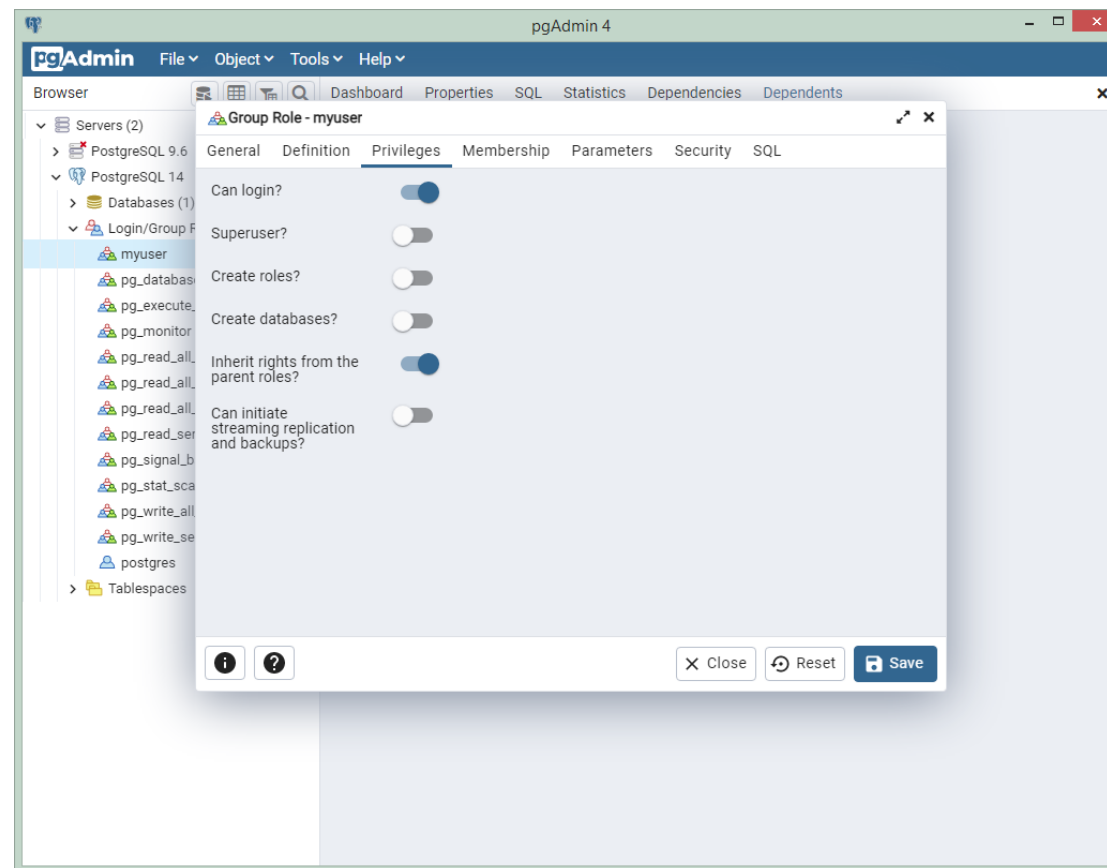
Подготовка к работе на домашнем компьютере

Устанавливаем пароль (поле Password).



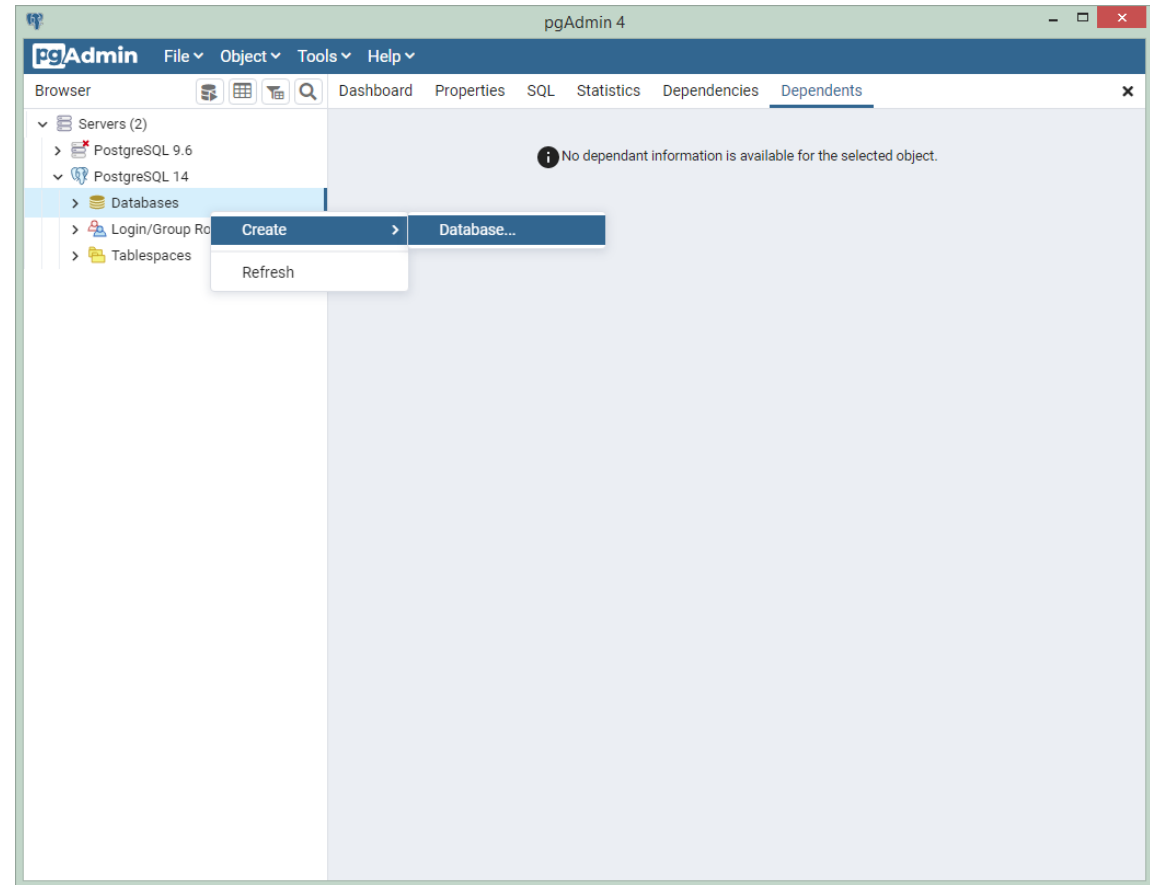
Подготовка к работе на домашнем компьютере

Разрешаем вход (переключатель Can login?) и сохраняем.



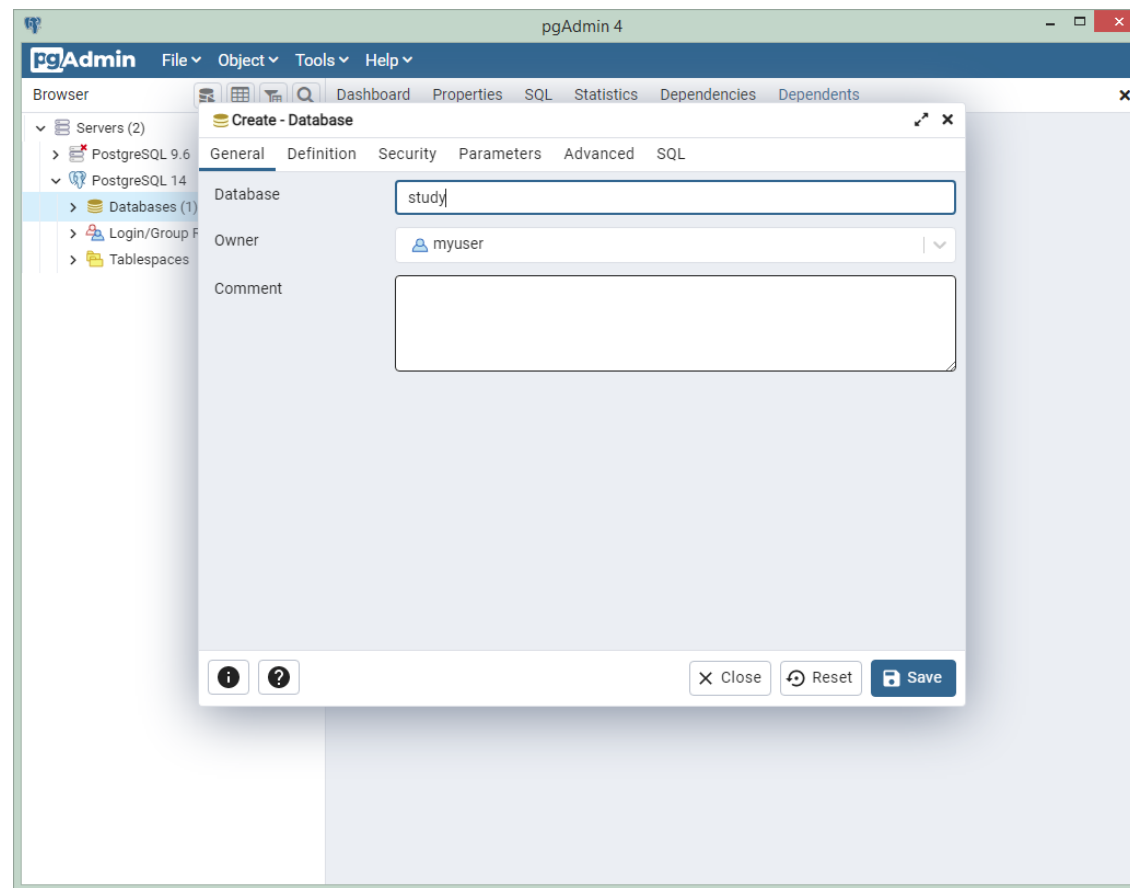
Подготовка к работе на домашнем компьютере

Создадим базу данных, отличную от БД суперпользователя.



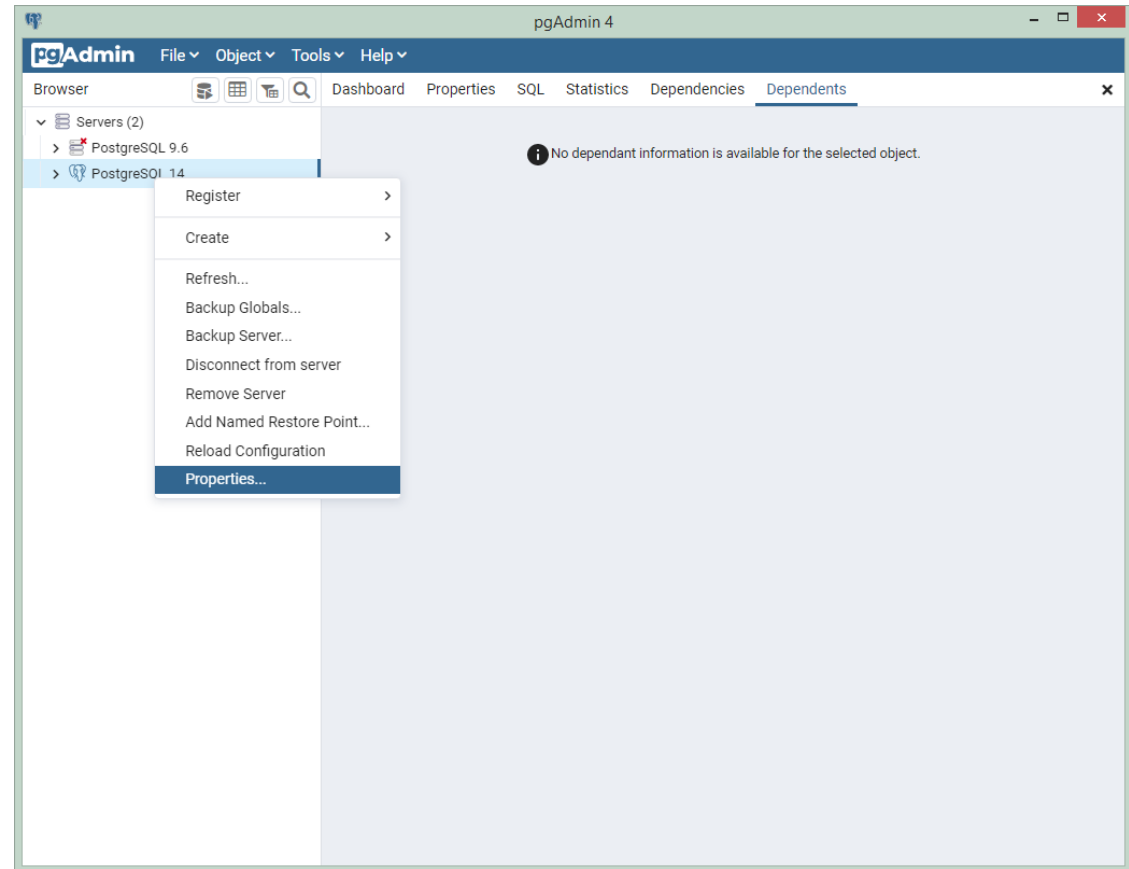
Подготовка к работе на домашнем компьютере

Введем название (на слайде study) и укажем пользователя, которого мы создали ранее (на слайде myuser).



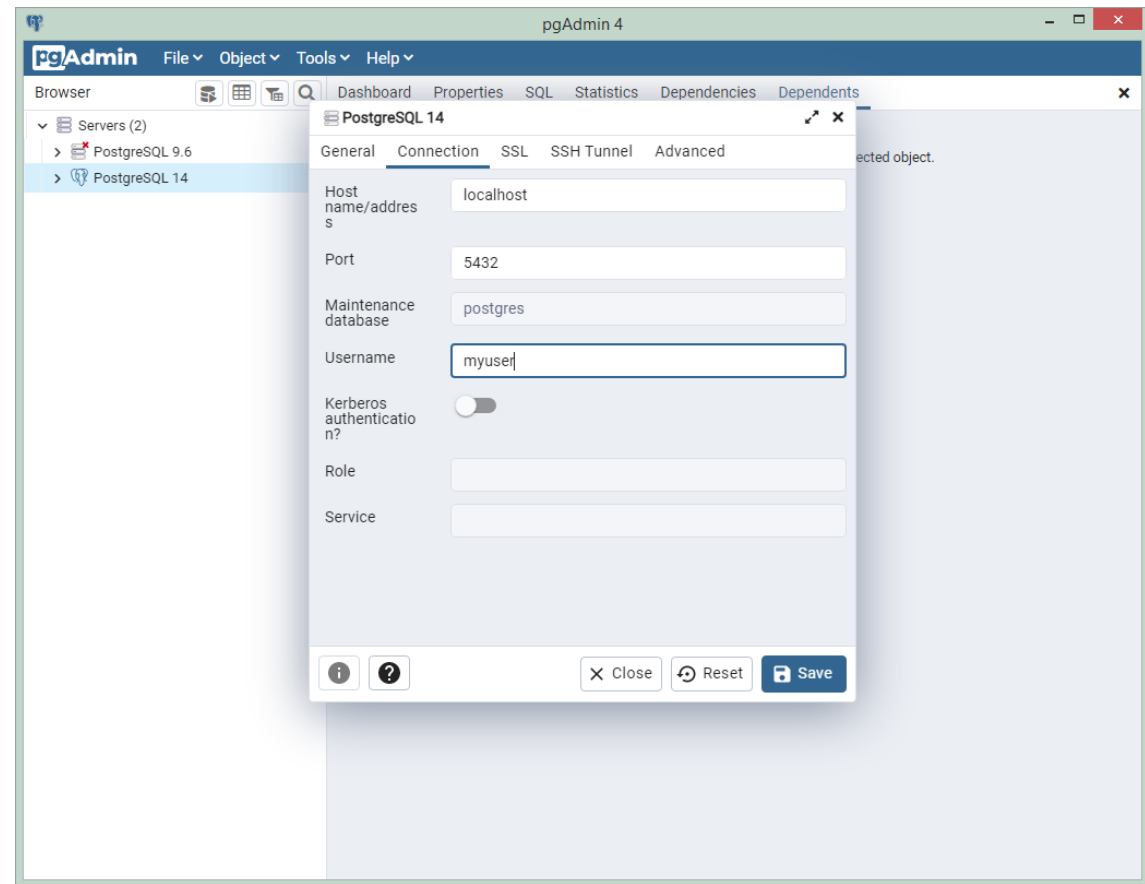
Подготовка к работе на домашнем компьютере

Теперь нужно заменить соединение, чтобы работа шла через нового пользователя.



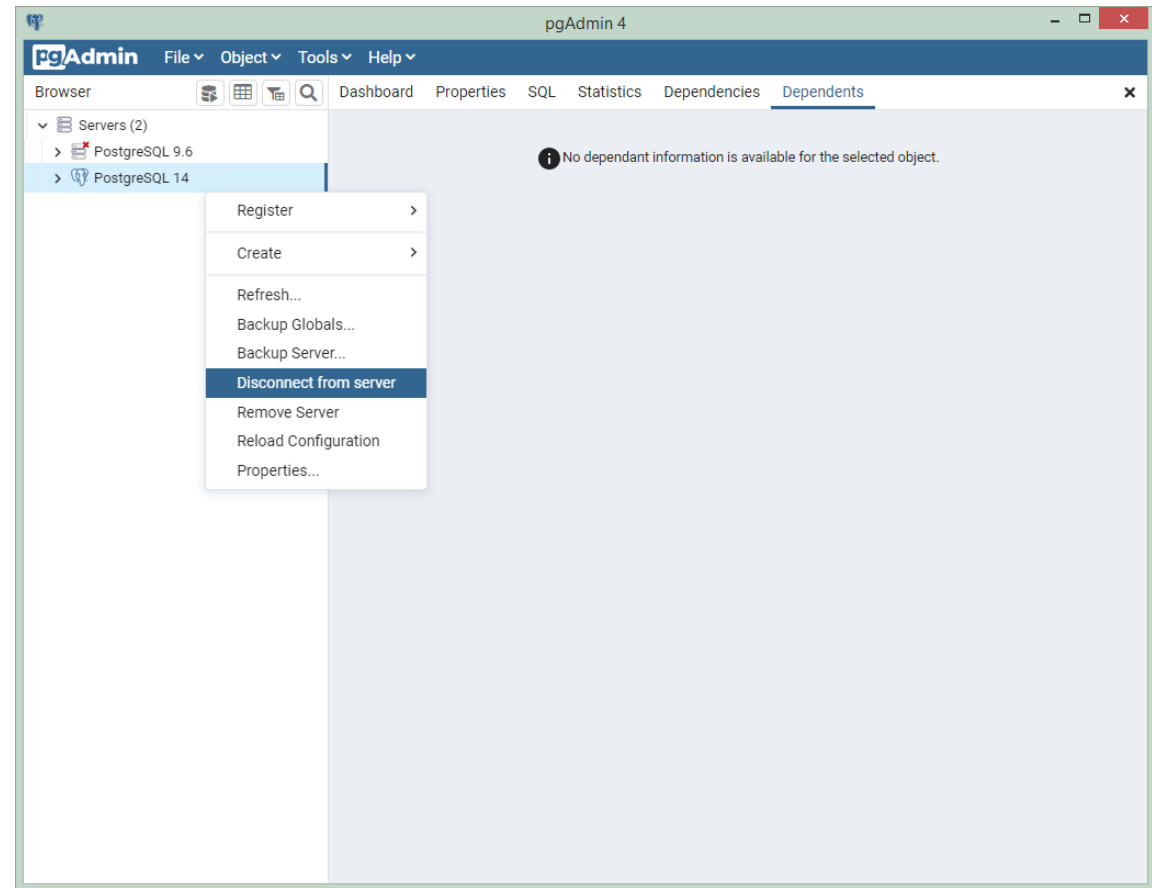
Подготовка к работе на домашнем компьютере

В поле Username меняем postgres на созданного нами пользователя (на слайде myuser).



Подготовка к работе на домашнем компьютере

Разрываем соединение (см. рисунок на слайде) и двойным кликом мыши по нужной версии PostgreSQL (на слайде PostgreSQL 14) устанавливаем соединение заново.



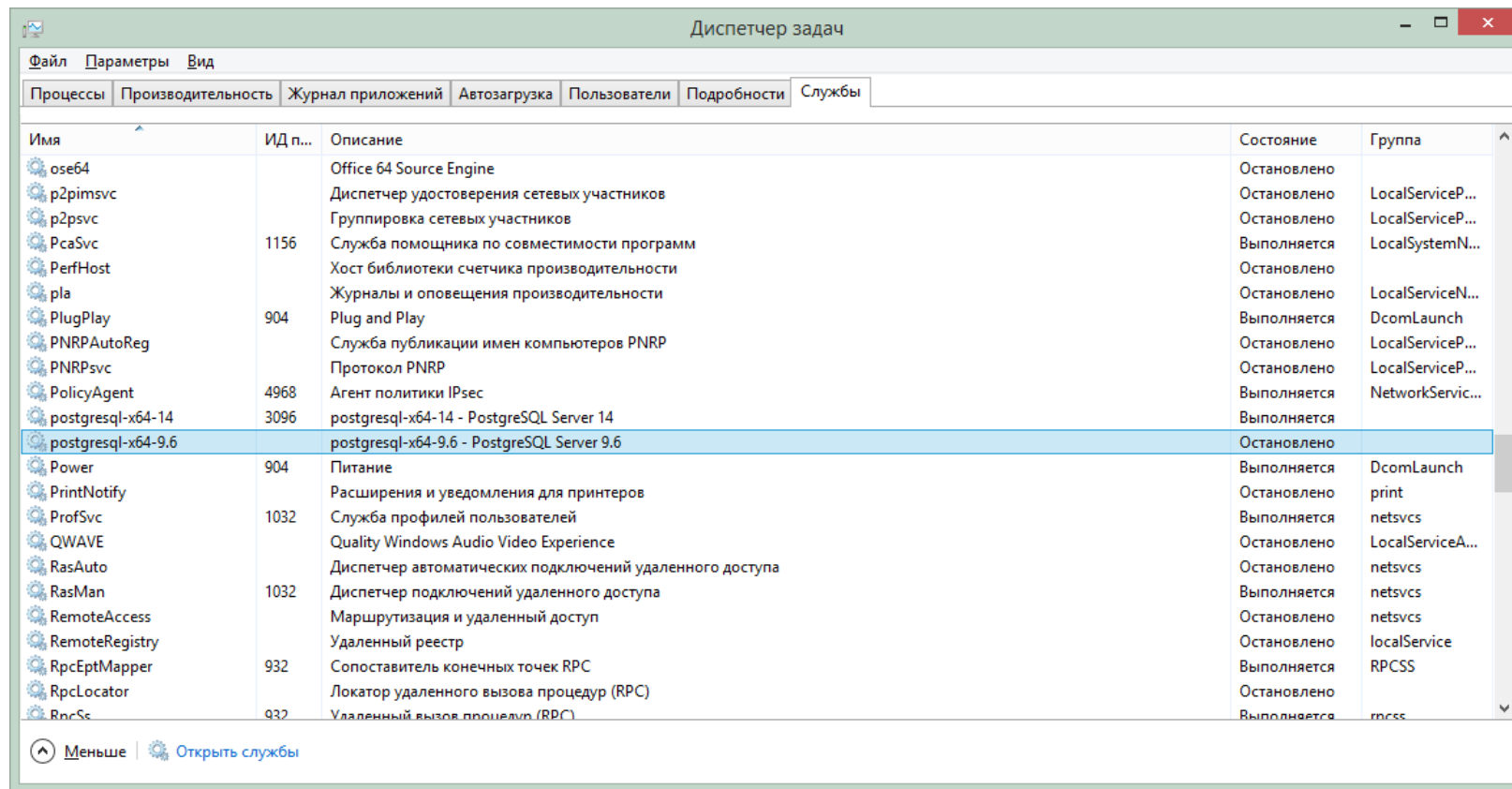
Часть 5. Как временно отключить PostgreSQL

Шаг 1

Инструкция написана для Windows.

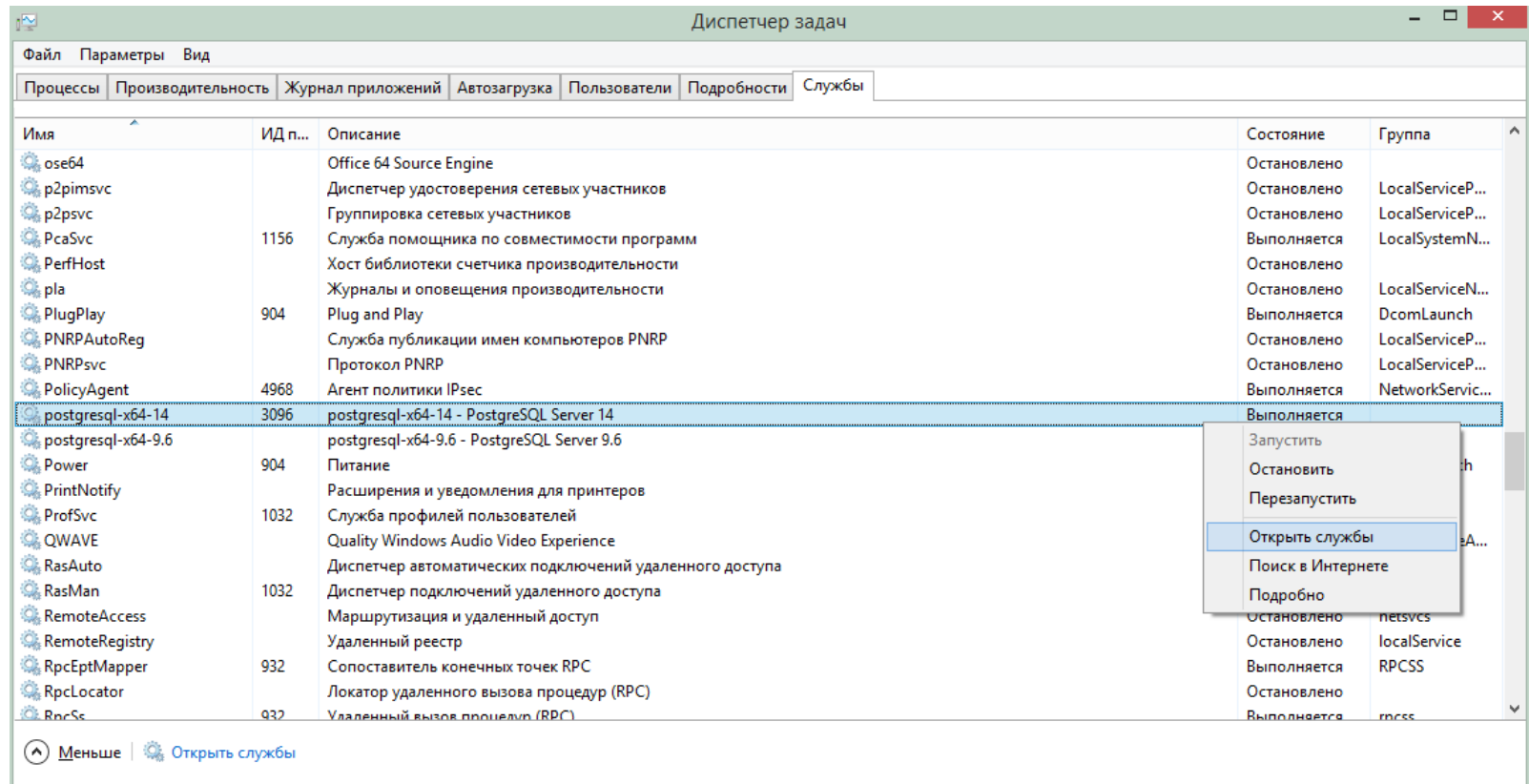
Нажимаем комбинацию клавиш **ctrl+alt+del** и открываем окно диспетчера задач.

Переходим на вкладку «Службы». Ищем В первом столбике название «postgresql-x64-14» (или 15, 16).



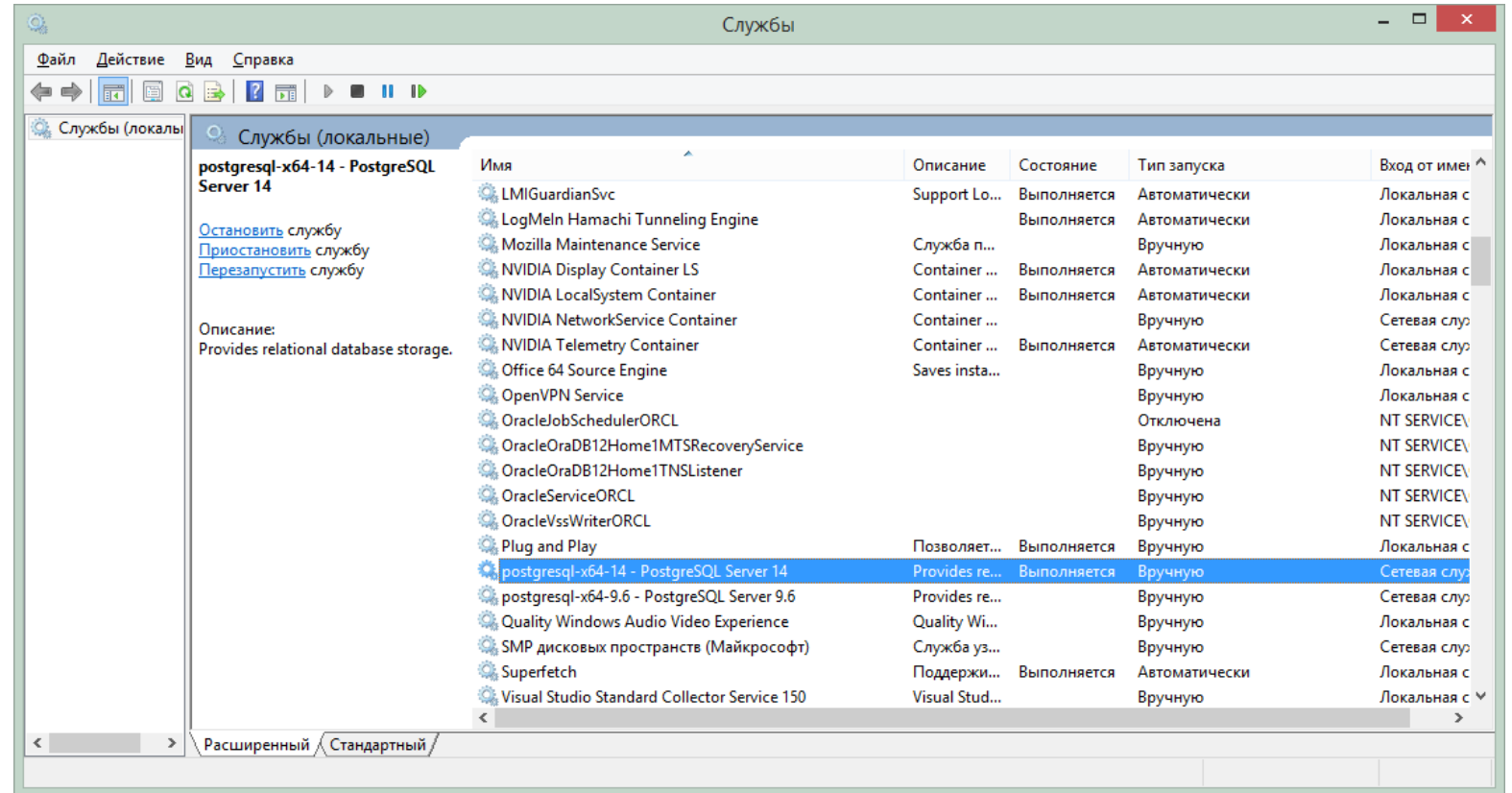
Шаг 2

Используя правую кнопку мыши переходим в раздел службы.



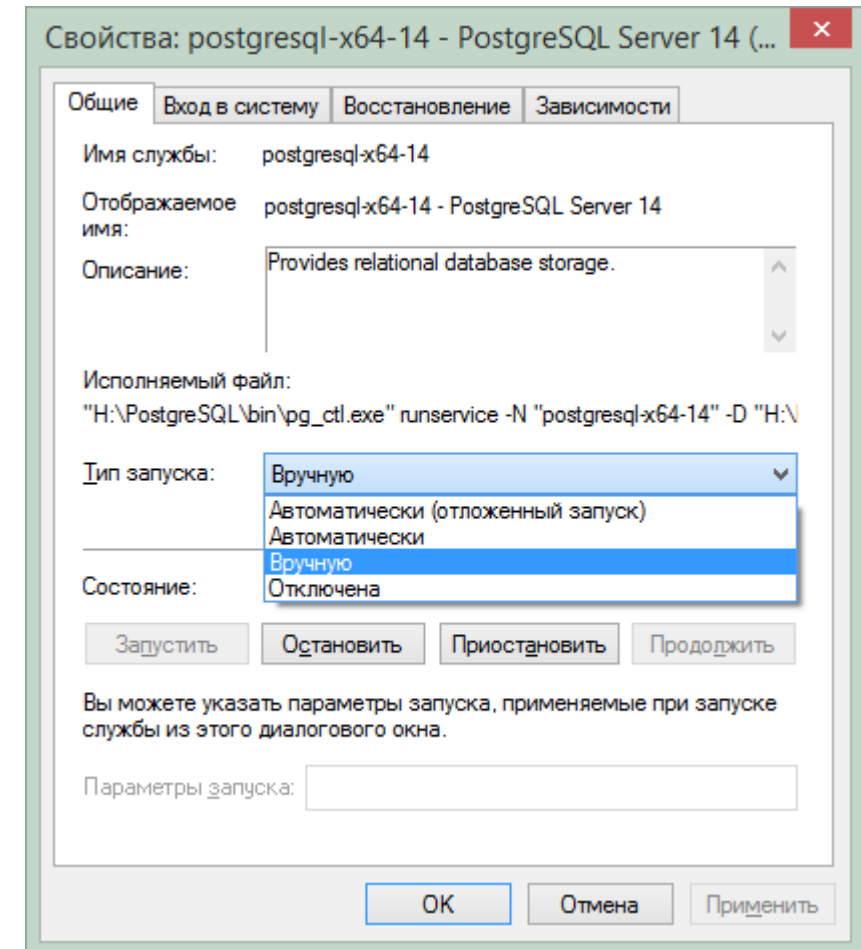
Шаг 3

Находим postgresql-x64-14 – PostgreSQL Server 14 (15 или 16) и щелкаем по нему двойным кликом.



Шаг 4

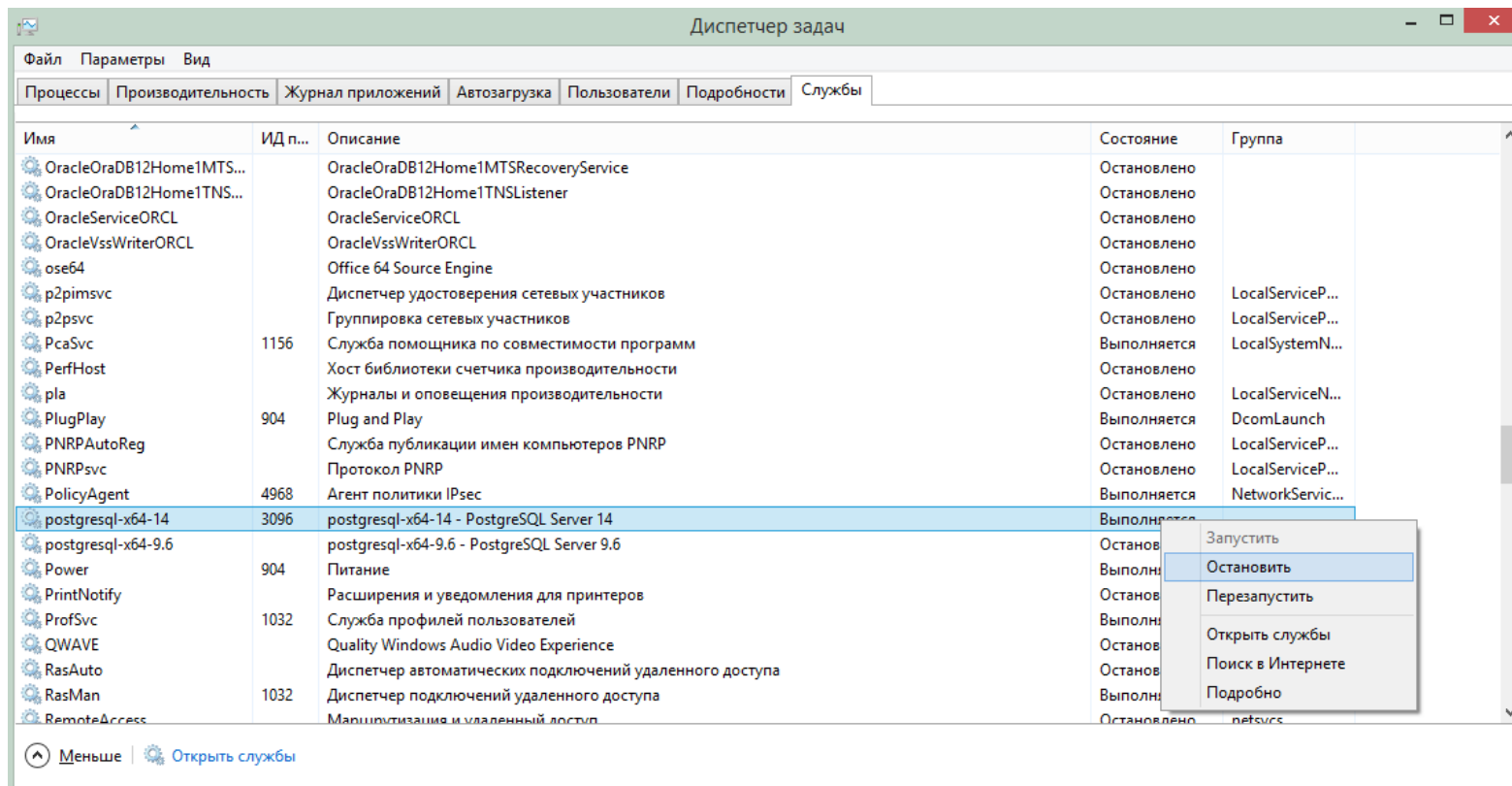
В открывшемся окне меняем тип запуска на «вручную» и нажимаем «ОК». Теперь PostgreSQL не будет запускаться сам при старте компьютера. Но он все еще работает в этом сеансе. Вернемся в диспетчер задач, чтобы его выключить.



Шаг 5

При помощи правой кнопки выбираем пункт «Остановить» и выключаем PostgreSQL. Теперь он полностью остановлен!

В случае необходимости его включения в этом же меню используем «Запустить».



Часть 6. История баз данных

0-ой период – файловые системы. В 60-х годах XX века в составе операционных систем появляются первые файловые системы, которые становятся прообразом первых баз данных.

I-й этап – иерархические и сетевые СУБД

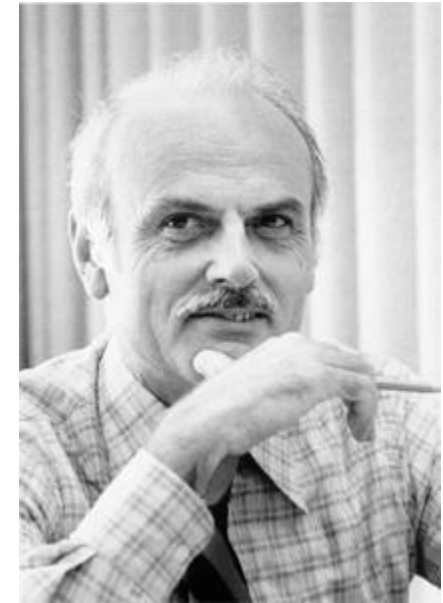
История развития баз данных также начинается в 60-х годах прошлого века. Первая промышленная СУБД была разработана фирмой IBM в 1968 году и носила название IMS. Первые СУБД во многом унаследовали особенности файловых систем и в своей основе содержали сетевую или иерархическую модель представления данных.



70-е годы прошлого века стали временем бурного развития теоретических аспектов баз данных. В 1975 году был сформулирован стандарт сетевых баз данных, получивший название CODASYL в честь образованного в 1959 году одноименного консорциума. Стандарт включал в себя спецификации двух основных языков: языка описания данных (DDL) и языка манипуляции данными (DML). На основе данной спецификации было разработано большое количество сетевых СУБД, например, IDS/2, IDMS, DBMS32, db_Vista и другие.

II-й этап – реляционные СУБД

Все СУБД, базирующиеся на сетевых и иерархических моделях данных, имели большие недостатки в плане удобства построения запросов к ним. По этой причине в начале 80-х годов прошлого века в мире баз данных власть захватили реляционные системы. Их историю можно начать с опубликованной в 1970 году статьи Э.Ф. Кодда, предлагающей описывать данные при помощи таблиц, называемых отношениями (relations). Можно даже вернуться еще на год назад в 1969, когда тот же Эдгар Кодд во внутреннем документе компании IBM сформулировал основные идеи реляционной модели и своих будущих 12 правил реляционных СУБД. Официально же 12 правил Кодда предстали свету в 1985 году, как публикация в журнале «ComputerWorld», направленная на борьбу с «неправильными» реляционными СУБД.



Эдгар Франк Кодд

Появление языка SQL и СУБД Oracle

В 1979 году в свет вышла первая версия знаменитой СУБД Oracle (правда, уже носившая номер 2). Она еще не поддерживала транзакций и многих других механизмов современных баз данных, но зато стала первой коммерческой реляционной СУБД.



Производитель Oracle компания RSI (раньше называющаяся SDL) смогла опередить в этом вопросе даже фирму IBM, в которой зародилась теория реляционных баз данных и у которой уже был разработан прототип реляционной СУБД System R.

Зато в своей СУБД фирма IBM заложила основы будущего общепринятого языка запросов к реляционным базам данных – SQL (Structured Query Language). Правда, изначально он назывался SEQUEL (Structured English QUery Language), но позднее был переименован по юридическим соображениям.

Стандарты языка SQL

В 1986 году институтом ANSI был принят первый стандарт языка SQL. В 1987 году он был одобрен ISO.

В 1989 году доработанный вариант SQL/86 был принят как полноценный первый международный стандарт языка запросов к реляционным базам данных SQL. Он получил название SQL/89.

К сожалению, у него был ряд недостатков. Важнейшими из которых были следующие два: во-первых, многие аспекты были описаны в языке как зависящие от реализации и, во-вторых, многие практически важные вещи просто отсутствовали. Поэтому в конце 1992 года был принят новый международный стандарт языка SQL – SQL/92.

В 1999 году стандарт языка SQL/92 был дополнен поддержкой регулярных выражений, рекурсивных запросов, триггеров, базовых процедурных расширений, нескалярных типов данных и некоторыми другими объектно-ориентированными возможностями. Тем самым был получен стандарт SQL:1999.

Позднее были приняты еще три стандарта: SQL:2003, SQL:2006, SQL:2008, расширяющие язык новыми возможностями.



Появление СУБД PostgreSQL

Разработка PostgreSQL (тогда просто Postgres) началась в 1986 году студентами департамента Беркли, Калифорнийского Университета под руководством Майкла Стоунбрекера. Первая «демоверсия» заработала в 1987 и была показана в 1988 на конференции ACM-SIGMOD.

В 1994 Эндри Ю и Джолли Чен добавили в POSTGRES интерпретатор языка SQL. Позднее в 1995 он разделился на две ветки, одна из которых получила название Postgres95.

В 1996 году Postgres95 сменил имя на PostgreSQL. В этом же году была опубликована его версия 6.0, которую можно считать первым промышленным релизом PostgreSQL.



Майкл Стоунбрейкер

III-й этап – объектные СУБД

Параллельно с реляционными СУБД в середине 80-х годов начали развиваться объектные (объектно-ориентированные) СУБД. Их появление связано с развитием объектно-ориентированного программирования (ООП) и соответствующих языков. Как известно, первым объектно-ориентированным языком программирования был разработанный в 1967 году язык Симула. Первоначально объектные СУБД разрабатывались только как поддержка различных систем САПР.

В 90-х годах разработчики объектных баз данных стали обращать внимание и на другие области применения. В 1989 году был опубликован манифест систем объектно-ориентированных баз данных, в котором была предпринята попытка дать определение системы объектно-ориентированных баз данных.

В 1993 году в сотрудничестве с OMG, ANSI, ISO и другими организациями был создан стандарт ODMG-93. Этот стандарт включает в себя средства для построения законченного приложения, которое будет работать (после перекомпиляции) в любой совместимой с этой спецификацией объектной СУБД. В стандарт ODMG-93 вошли следующие разделы:

- язык определения объектов (ODL);
- язык объектных запросов (OQL);
- связывание с языком C++;
- связывание с языком Smalltalk.

IV-й этап – NoSQL СУБД

Современные высоконагруженные системы (например поисковые сервера), обслуживающие в параллельном режиме миллионы клиентов, сталкиваются с серьёзными проблемами при работе с физическими устройствами хранения. Самая медленная часть современных ЭВМ – это жёсткий диск. Реляционные СУБД с точки зрения работы с жёстким диском не являются оптимальным решением. По этой причине в XXI веке на смену реляционным СУБД в высоконагруженных системах пришли так называемые NoSQL СУБД. К ним относятся:

- колоночные СУБД;
- документно-ориентированные СУБД;
- хранилища «ключ-значение»;
- другие виды СУБД.

Полезные ссылки

1. <https://db-engines.com/en/ranking> – рейтинг СУБД в мире.
2. <https://www.fivetran.com/blog/postgresql-vs-mysql> – PostgreSQL vs MySQL. За рубежом сравнение.
3. <https://sbercloud.ru/ru/warp/blog/mysql-vs-postgresql> – PostgreSQL vs MySQL. Сравнение от Сбербанка.
4. <https://www.postgresql.org/download/> – страница загрузки PostgreSQL.
5. <https://www.enterprisedb.com/downloads/postgres-postgresql-downloads> – страница загрузки PostgreSQL под Windows.
6. <https://onecompiler.com/postgresql/> – один из онлайн-интерпретаторов PostgreSQL.
7. <https://www.postgresql.org/docs/14/index.html> – документация PostgreSQL 14.9.
8. <https://www.postgresql.org/docs/15/index.html> – документация PostgreSQL 15.4.
9. <https://www.postgresql.org/docs/current/index.html> - документация последней версии PostgreSQL (на момент лекции – 16).