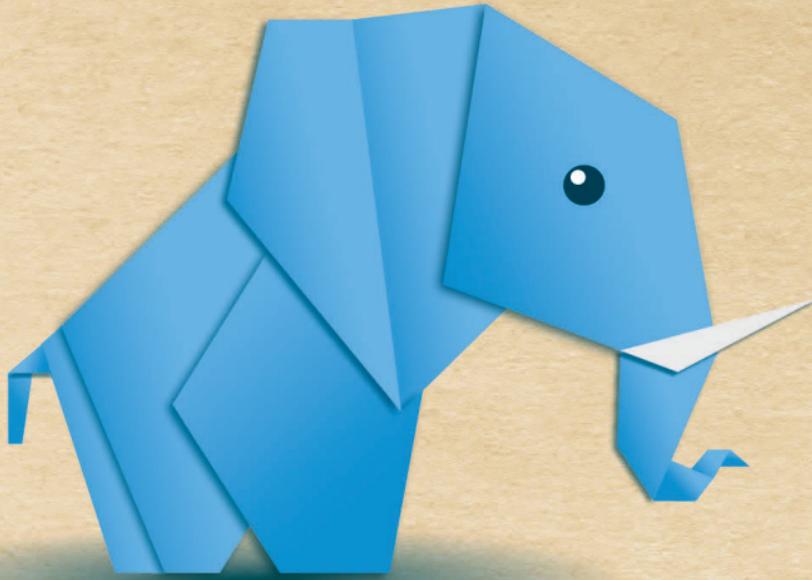


# PostgreSQL



для начинающих

v.2

# Предисловие

Эту небольшую книгу мы написали для тех, кто только начинает знакомиться с миром PostgreSQL. В бумажном виде ее можно взять с собой и полистать по дороге на работу, а из электронной версии (доступной по адресу [postgrespro.ru/education/introbook](http://postgrespro.ru/education/introbook)) удобно копировать примеры кода и переходить по ссылкам.

Из книги вы узнаете:

- Что вообще такое, этот PostgreSQL (и как произнести это название), откуда он появился и что умеет ..... c. 2
- Как установить PostgreSQL на Linux и Windows, как управлять службой СУБД..... c. 13
- Как подключиться к серверу и начать с ним работать. Мы познакомим вас с основными возможностями языка SQL, научим создавать таблицы, писать небольшие запросы и пользоваться транзакциями ..... c. 24
- Что нужно сделать, чтобы использовать PostgreSQL в качестве базы данных для вашего приложения ..... c. 50
- Про полезную программу pgAdmin..... c. 58
- Где найти документацию и пройти обучение ..... c. 63
- Как быть в курсе происходящего и не пропустить интересные материалы и события ..... c. 70
- И немного про нашу компанию Postgres Professional ..... c. 73

Мы надеемся, что наша книга сделает ваш первый опыт работы с PostgreSQL приятным и поможет влиться в сообщество пользователей этой СУБД.

Желаем удачи!



# O PostgreSQL

PostgreSQL — наиболее полнофункциональная, свободно распространяемая СУБД с открытым кодом. Разработанная в академической среде, за долгую историю сплотившая вокруг себя широкое сообщество разработчиков, эта СУБД обладает всеми возможностями, необходимыми большинству заказчиков. PostgreSQL активно применяется по всему миру для создания критичных бизнес-систем, работающих под большой нагрузкой.

## Немного истории

Современный PostgreSQL ведет происхождение от проекта POSTGRES, который разрабатывался под руководством Майкла Стоунбрейкера (Michael Stonebraker), профессора Калифорнийского университета в Беркли. До этого Майкл Стоунбрейкер уже возглавлял разработку INGRES — одной из первых реляционных СУБД, — и POSTGRES возник как результат осмысления предыдущей работы и желания преодолеть ограниченность жесткой системы типов.

Работа над проектом началась в 1985 году и до 1988 года был опубликован ряд научных статей, описывающих модель данных, язык запросов POSTQUEL (в то время SQL еще не был общепризнанным стандартом) и устройство хранилища данных.

POSTGRES иногда еще относят к так называемым постреляционным СУБД. Ограниченность реляционной модели всегда была предметом критики, хотя и являлась обратной стороной ее простоты и строгости. Однако проникновение компьютерных технологий во все сферы жизни требовало новых приложений, а от баз данных — поддержки новых типов данных и таких возможностей, как наследование, создание сложных объектов и управление ими.

Первая версия СУБД была выпущена в 1989 году. База данных совершенствовалась на протяжении нескольких лет, а в 1993 году, когда вышла версия 4.2, проект был закрыт. Но, несмотря на официальное прекращение, открытый код и BSD-лицензия позволили выпускникам Беркли Эндрю Ю и Джоли Чену в 1994 году взяться за его дальнейшее развитие. Они заменили язык запросов POSTQUEL на ставший к тому времени общепринятым SQL, а проект получил название Postgres95.

К 1996 году стало ясно, что название «Postgres95» не выдержит испытание временем и было выбрано новое имя — PostgreSQL, которое отражает связь и с оригинальным проектом POSTGRES, и с переходом на SQL. Именно поэтому PostgreSQL произносится «постгрес-ку-эль» или просто «постгрес», но только не «постгрэс-ку-эль».

Новая версия стартовала как 6.0, продолжая исходную нумерацию. Проект вырос и управление им взяла на себя поначалу небольшая группа инициативных пользователей и разработчиков, которая получила название Глобальной группы разработки PostgreSQL (PostgreSQL Global Development Group).

## Развитие

Все основные решения о планах развития и выпусках новых версий принимаются Управляющим комитетом (Core team) проекта. В настоящий момент он состоит из шести человек.

Помимо обычных разработчиков, вносящих посильную лепту в развитие системы, выделяется группа основных разработчиков (major contributors), сделавших существенный вклад в развитие PostgreSQL, а также группа разработчиков, имеющих право записи в репозиторий исходного кода (committers). Состав группы разработчиков со временем меняется, появляются новые члены, кто-то отходит от проекта. Актуальный список разработчиков поддерживается на официальном сайте PostgreSQL [www.postgresql.org](http://www.postgresql.org).

Вклад российских разработчиков в PostgreSQL весьма значителен. Это, пожалуй, самый крупный глобальный проект с открытым исходным кодом с таким широким российским представительством.

Большую роль в становлении и развитии PostgreSQL сыграл программист из Красноярска Вадим Михеев, входивший в Управляющий комитет. Он является автором таких важнейших частей системы, как многоверсионное управление одновременным доступом (MVCC), система очистки (vacuum), журнал транзакций (WAL), вложенные запросы, триггеры. Сейчас Вадим уже не занимается проектом.

В настоящий момент в число основных разработчиков входят трое представителей России — Олег Бартунов, Федор Сигаев и Александр Коротков, — основавших в 2015 году компанию Postgres Professional. Среди крупных направлений выполненных ими работ можно выделить локализацию PostgreSQL (поддержка национальных кодировок и Unicode), систему полнотекстового поиска, работу с массивами и слабо-структурированными данными (hstore, json, jsonb), новые методы индексации (GiST, GIN, SP-GiST). Они являются авторами большого числа популярных расширений.

Цикл работы над очередной версией PostgreSQL обычно занимает около года. За это время от всех желающих принимаются на рассмотрение патчи с исправлениями, изменениями и новым функционалом. Для обсуждения патчей по традиции используется список рассылки pgsql-hackers. Если сообщество признает идею полезной, ее реализацию — правильной, а код проходит обязательную проверку другими разработчиками, то он включается в новый релиз.

В некоторый момент объявляется этап стабилизации кода — новый функционал откладывается до следующей версии, а продолжают приниматься только исправления или улучшения уже включенных в релиз патчей. Иногда в процессе работы

над новой версией вскрываются и исправляются старые ошибки, а наиболее критические из них исправляются и в предыдущих версиях. По мере накопления таких исправлений принимается решение о выпуске новой стабильной версии, совместимой со старой. Например, версия 9.5.2 содержит только исправления ошибок, найденных в 9.5.

Несколько раз в течении релизного цикла выпускаются бета-версии, ближе к концу цикла появляется релиз-кандидат, а вскоре выходит и новая основная версия PostgreSQL: в ее номере увеличивается второе число (например, 9.6 после 9.5). Когда Управляющий комитет решает, что количество изменений переросло в качество, принимается решение об увеличении и первого числа. Так, возможно, вместо версии 9.7 в свое время мы увидим уже 10.0.

## Поддержка

Глобальная группа разработки выполняет поддержку основных версий системы в течении пяти лет с момента выпуска. Эта поддержка, как и координация разработки, осуществляется через списки рассылки. Корректно оформленное сообщение об ошибке имеет все шансы на скорейшее решение: нередки случаи, когда исправления ошибок выпускаются в течении суток.

Помимо поддержки сообществом разработчиков, ряд компаний по всему миру осуществляет коммерческую поддержку PostgreSQL. В России такой компанией является Postgres Professional ([postgrespro.ru](http://postgrespro.ru)), предоставляя услуги по поддержке в режиме 24x7.



# **Современное состояние**

PostgreSQL является одной из самых популярных баз данных. За более чем 20-летнюю историю развития на прочном фундаменте, заложенном академической разработкой, PostgreSQL выросла в полноценную СУБД уровня предприятия и составляет реальную альтернативу коммерческим базам данных. Чтобы убедиться в этом, достаточно посмотреть на важнейшие характеристики новейшей на сегодняшний день версии PostgreSQL 9.5.3.

## **Надежность и устойчивость**

Вопросы обеспечения надежности особенно важны в приложениях уровня предприятия для работы с критически важными данными. С этой целью PostgreSQL позволяет настраивать горячее резервирование, восстановление на заданный момент времени в прошлом, различные виды репликации (синхронную, асинхронную, каскадную).

## **Безопасность**

PostgreSQL позволяет работать по защищенному SSL-соединению и предоставляет большое количество методов аутентификации, включая аутентификация по паролю, клиентским сертификатам, а также с помощью внешних сервисов (LDAP, RADIUS, PAM, Kerberos).

При управлении пользователями и доступом к объектам БД предоставляются следующие возможности:

- Создание и управление пользователями и групповыми ролями;
- Разграничение доступа к объектам БД на уровне как отдельных пользователей, так и групп;

- Детальное управление доступом на уровне отдельных столбцов и строк;
- Поддержка SELinux через встроенную функциональность SE-PostgreSQL (мандатное управление доступом).

В компании Postgres Professional ведется работа по государственной сертификации PostgreSQL для использования в системах обработки конфиденциальной информации и персональных данных.

## Соответствие стандартам

По мере развития стандарта ANSI SQL его поддержка постоянно добавлялась в PostgreSQL. Это относится ко всем версиям стандарта: SQL-92, SQL:1999, SQL:2003, SQL:2008, SQL:2011. В настоящий момент PostgreSQL 9.5 обеспечивает высокий уровень соответствия стандарту SQL:2011 и поддерживает 160 из 179 обязательных возможностей, а также большое количество необязательных.

## Поддержка транзакционности

PostgreSQL обеспечивает полную поддержку свойств ACID и обеспечивает эффективную изоляцию транзакций. Для этого в PostgreSQL используется механизм многоверсионного управления одновременным доступом (MVCC). Он позволяет обходиться без блокировок во всех случаях, кроме одновременного изменения одной и той же строки данных в нескольких процессах. При этом читающие транзакции никогда не блокируют пишущие транзакции, а пишущие — читающих. Это справедливо и для самого строгого уровня изоляции serializable, который, используя инновационную систему Serializable Snapshot Isolation, обеспечивает полное отсутствие аномалий сериализации и гарантирует, что при одновременном выполнении транзакций результат будет таким же, как и при последовательном выполнении.

## Для разработчиков приложений

Разработчики приложений получают в свое распоряжение богатый инструментарий, позволяющий реализовать приложения любого типа:

- Возможность использования различных языков для серверного программирования, в том числе встроенного PL/pgSQL (удобного тесной интеграцией с SQL), C для критичных по производительности задач, Perl, Python, Tcl, а также Javascript, Java и другие;
- Программные интерфейсы для обращения к СУБД из приложений на любом языке, включая стандартные интерфейсы ODBC и JDBC;
- Набор объектов баз данных, позволяющий эффективно реализовать логику любой сложности на стороне сервера: таблицы и индексы, ограничения целостности, представления и материализованные представления, последовательности, секционирование, подзапросы и with-запросы (в том числе рекурсивные), агрегатные и оконные функции, хранимые функции, триггеры и т. д.;
- Встроенная гибкая система полнотекстового поиска с поддержкой русского и всех европейских языков, поддержанная специально разработанным эффективным индексным доступом;
- Поддержка слабо-структурированных данных в духе NoSQL: хранилище пар «ключ-значение» hstore, xml, json (как в текстовом, так и в эффективном двоичном представлении jsonb);
- Подключение внешних источников данных, включая все основные СУБД, в качестве таблиц с возможностью их полноценного использования, в том числе записи и распределенного выполнения запросов (Foreign Data Wrappers).

## **Масштабируемость и производительность**

PostgreSQL эффективно использует современную архитектуру многоядерных процессоров — его производительность растет практически линейно с увеличением количества ядер.

В версии 9.6, которая должна появиться в 2016 году, PostgreSQL получит возможность параллельного чтения данных и выполнения запросов, что еще больше повысит возможности использования аппаратных средств для ускорения операций.

## **Планировщик запросов**

В PostgreSQL используется планировщик запросов, основанный на стоимости. Использую собираемую статистику и учитывая в своих математических моделях как дисковые операции, так и время работы процессора, планировщик позволяет оптимизировать самые сложные запросы. В его распоряжении находятся все методы доступа к данным и способы выполнения соединений, характерные для передовых коммерческих СУБД.

## **Возможности индексирования**

В PostgreSQL реализованы различные методы индексирования. Помимо традиционных B-деревьев, также доступны:

- GiST — обобщенное сбалансированное дерево поиска, которое может применяться для данных, не допускающих упорядочения. Примерами могут служить R-деревья для индексирования точек на плоскости с возможностью поиска ближайших соседей (k-NN search) и индексирование операции пересечения для интервалов;
- SP-GiST — обобщенное несбалансированное дерево поиска, основанное на разбиении области значений на непересекающиеся вложенные области. Примером может служить дерево квадрантов или префиксное дерево;

- GIN — обобщенный инвертированный индекс. Основной областью применения является полнотекстовый поиск, где требуется находить документы, в которых встречается указанное в поисковом запросе слово. Другим примером является поиск значений в массивах данных;
- BRIN — индекс небольшого размера, позволяющий найти компромисс между размером индекса и скоростью поиска. Эффективен на больших кластеризованных таблицах;
- Bloom — индекс, основанный на фильтре Блума (появился в версии 9.6). Такой индекс, имея очень компактное представление, позволяет быстро отсечь ненужные строки, однако требует перепроверки оставшихся.

За счет расширяемости набор доступных методов индексного доступа постоянно увеличивается.

Независимо от типа, индексы могут строиться как по одному, так и по нескольким столбцам таблицы, а также по выражениям. Можно создать частичный индекс только для определенных строк таблицы. Покрывающие индексы позволяют ускорить запросы за счет того, что все необходимые данные извлекаются из самого индекса без обращения к таблице.

Несколько индексов могут автоматически объединяться с помощью битовой карты для ускорения доступа к таблице.

## Кроссплатформенность

PostgreSQL работает на операционных системах семейства Unix, включая серверные и клиентские разновидности Linux, FreeBSD, Solaris, Mac OS X, а также на Windows.

За счет открытого и переносимого кода на языке C PostgreSQL можно собрать на самых разных платформах, даже если для нее отсутствует поддерживаемая сообществом сборка.

## Расширяемость

Расширяемость — одно из фундаментальных преимуществ системы, лежащее в основе архитектуры PostgreSQL.

Пользователи могут самостоятельно, не меняя базовый код системы, добавлять:

- Типы данных;
- Функции и операторы для работы с новыми типами;
- Индексные методы доступа;
- Языки серверного программирования;
- Подключения к внешним источникам (Foreign Data Wrappers);
- Загружаемые расширения.

Полноценная поддержка расширений позволяет реализовать функционал любой сложности, не внося изменений в ядро PostgreSQL, и допуская подключение по мере необходимости. Например, именно в виде расширений построены такие сложные системы, как:

- CitusDB — возможность распределения данных по разным экземплярам PostgreSQL (шардинг) и массивно-параллельного выполнения запросов;
- PostGIS — система обработки геоинформационных данных.

Только стандартный комплект, входящий в сборку PostgreSQL 9.5, содержит больше полусотни расширений, доказавших свою надежность и полезность.

## Доступность

Лицензия PostgreSQL разрешает неограниченное использование СУБД, модификацию кода, а также включение в состав других продуктов, в том числе закрытых и коммерческих.

## **Независимость**

PostgreSQL не принадлежит ни одной компании и развивается международным сообществом, в том числе и российскими разработчиками. Это означает, что системы, использующие PostgreSQL, не зависят от конкретного вендора, тем самым в любой ситуации сохраняя вложенные в них инвестиции.

# Установка и начало работы

Как же начать работать с PostgreSQL? Если вы располагаете получасом свободного времени и готовы попробовать, следуйте за нами. В этой главе мы объясним, как установить службу PostgreSQL и управлять ей, а потом создадим простую базу данных и покажем, как создать в ней таблицы. Мы расскажем и про основы языка SQL, на котором формулируются запросы. Будет неплохо, если вы сразу начнете пробовать команды по мере чтения.

Мы будем использовать дистрибутив Postgres Pro, разработанный в нашей компании Postgres Professional. Этот дистрибутив полностью совместим с обычной СУБД PostgreSQL и дополнительно включает в себя некоторые разработки, выполненные в нашей компании, а также ряд возможностей, которые будут доступны только в следующей версии PostgreSQL.

Итак, приступим. Установка и запуск сервера PostgreSQL зависит от того, какую операционную систему вы используете. Если у вас Windows, читайте дальше; если Linux семейства Debian или Ubuntu — переходите сразу к с. 20.

Инструкцию по установке для других операционных систем вы найдете на сайте нашей компании: [postgrespro.ru/products/postgrespro/download/latest](http://postgrespro.ru/products/postgrespro/download/latest).

Если нужной вам версии там не оказалось — воспользуйтесь обычным дистрибутивом PostgreSQL: инструкции находятся по адресу [www.postgresql.org/download](http://www.postgresql.org/download).



# Windows

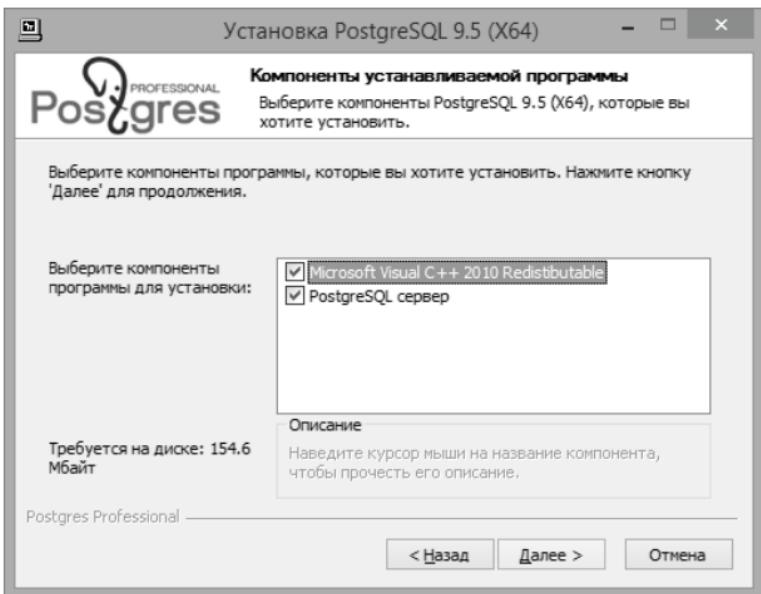
## Установка

Скачайте установщик с нашего сайта:  
[www.postgrespro.ru/windows](http://www.postgrespro.ru/windows). Выберите  
32- или 64-разрядную версию в зависимости  
от того, какая у вас версия Windows. Запустите  
скачанный файл и выберите язык установки.



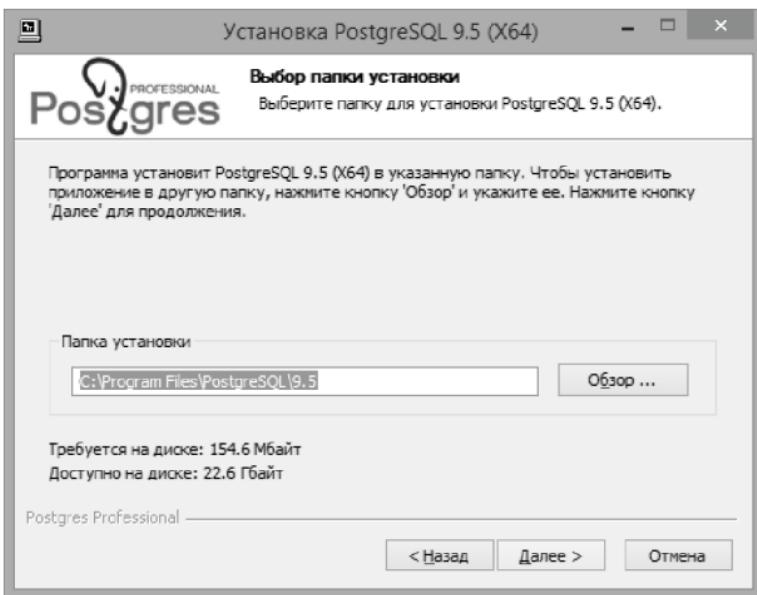
Установщик построен в традиционном стиле «мастера»:  
вы можете просто нажимать на кнопку «Далее», если вас  
устраивают предложенные варианты. Остановимся подробнее  
на основных шагах.

Компоненты устанавливаемой программы:

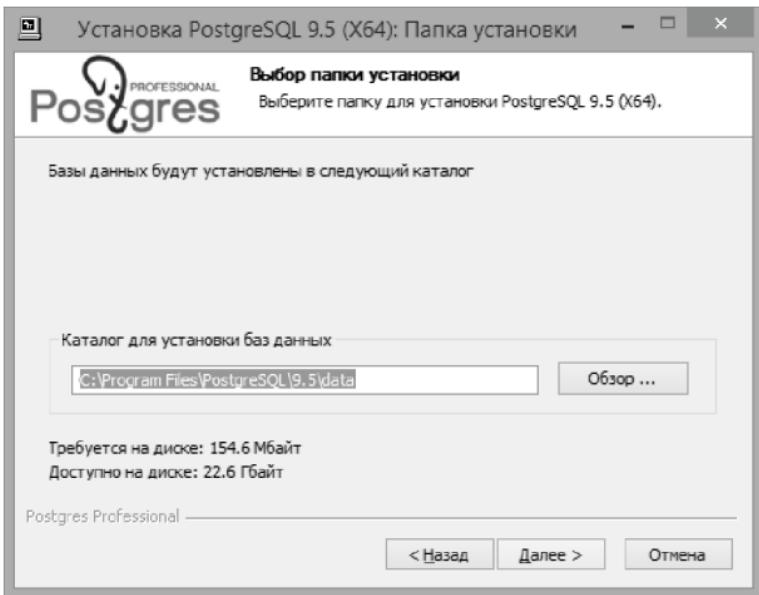


Оставьте оба флажка, если не уверены, какие выбрать.

## Выбор папки установки:

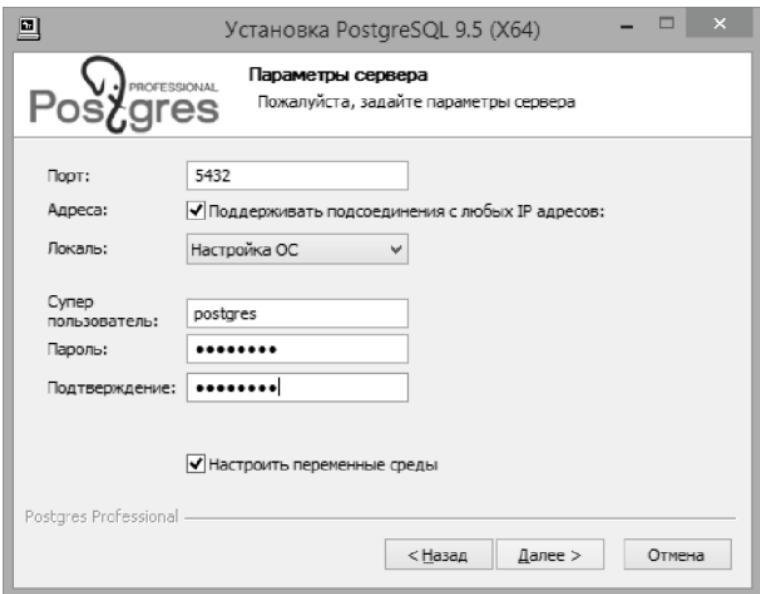


По умолчанию PostgreSQL устанавливается в каталог C:\Program Files\PostgreSQL\9.5 (или C:\Program Files (x86)\PostgreSQL\9.5 для 32-разрядной версии на 64-разрядной системе).



Отдельно можно выбрать расположение каталога для баз данных. Именно здесь будет находиться хранимая в СУБД информация, так что убедитесь, что на диске достаточно места, если вы планируете хранить много данных.

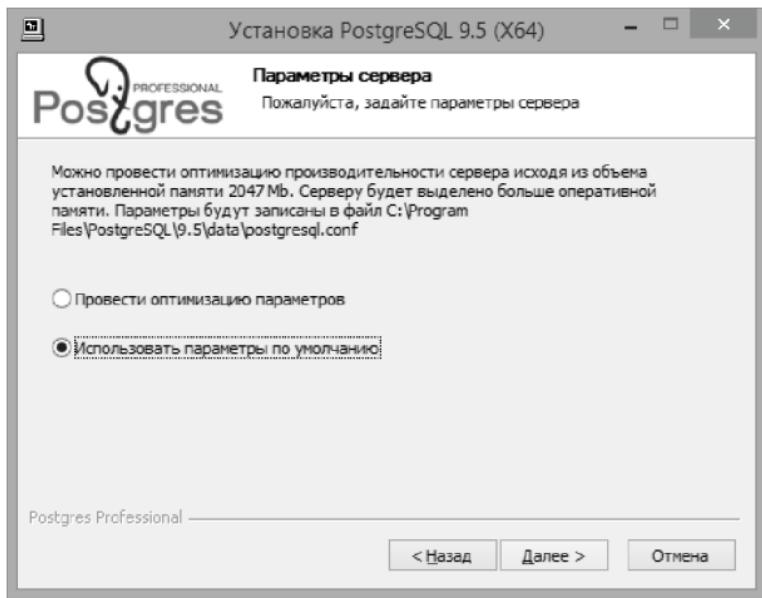
Параметры сервера:



Если вы планируете хранить данные на русском языке, выберите локаль «Russian, Russia» (или оставьте вариант «Настройка ОС», если в Windows используется русская локаль).

Введите (и подтвердите повторным вводом) пароль пользователя СУБД postgres. Также отметьте флашок «Настроить переменные среды», чтобы подключаться к серверу PostgreSQL под текущим пользователем ОС.

Остальные поля можно оставить со значениями по умолчанию.

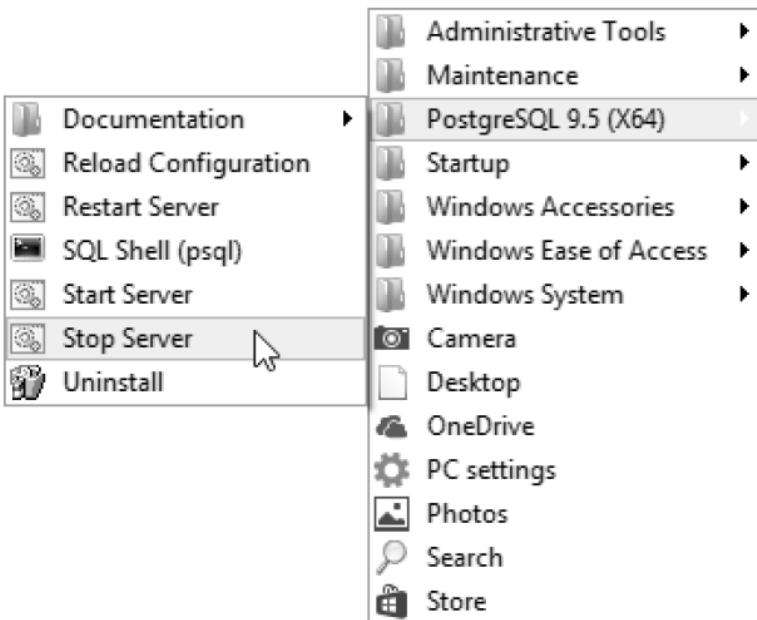


Если вы планируете установить PostgreSQL только для ознакомительных целей, можно отметить вариант «Использовать параметры по умолчанию», чтобы СУБД не занимала много оперативной памяти.

## Управление службой и основные файлы

При установке PostgreSQL в вашей системе регистрируется служба «postgresql-X64-9.5» (или «postgresql-X86-9.5» для 32-разрядной версии). Она запускается автоматически при старте компьютера под учетной записью Network Service. При необходимости вы можете изменить параметры службы с помощью стандартных средств Windows.

Чтобы временно остановить службу сервера баз данных, выполните программу «Stop Server» из папки в меню «Пуск», которую вы указали при установке:



Для запуска службы там же есть программа «Start Server».

Если при запуске службы произошла ошибка, для поиска причины следует заглянуть в журнал сообщений сервера. Он находится в подкаталоге pg\_log того каталога, который был выбран при установке для баз данных (обычно это будет C:\Program Files\PostgreSQL\9.5\pg\_log). Журнал настроен таким образом, чтобы запись периодически переключалась в новый файл. Найдите актуальный файл либо по дате последнего изменения, либо по имени, которое содержит дату и время переключения.

Есть несколько важных конфигурационных файлов, которые определяют настройки сервера. Они располагаются в каталоге баз данных. Вам не нужно их изменять, чтобы начать знакомство с PostgreSQL, но в реальной работе они непременно потребуются:

- `postgresql.conf` — основной конфигурационный файл, содержащий значения параметров сервера;
- `pg_hba.conf` — файл, определяющий настройки доступа. В целях безопасности по умолчанию доступ должен быть подтвержден паролем и допускается только с локального компьютера.

Загляните в эти файлы — они прекрасно документированы.

Теперь мы готовы подключиться к базе данных и попробовать некоторые команды и запросы. Переходите к разделу «Пробуем SQL» на с. 24.

# Linux

## Установка

Если вы используете Linux, то для установки необходимо подключить пакетный репозиторий нашей компании.

Для ОС Debian (в настоящее время поддерживаются версии 6 «Squeeze», 7 «Wheezy» и 8 «Jessie») выполните в терминале следующие команды:

```
$ sudo apt-get install lsb-release  
$ sudo sh -c 'echo "deb \http://repo.postgrespro.ru/pgpro-9.5/debian \  
\$\(lsb\_release -cs\) main" > \  
/etc/apt/sources.list.d/postgrespro.list'
```

Для ОС Ubuntu (в настоящее время поддерживаются версии 12.04 «Precise», 14.04 «Trusty», 15.04 «Vivid» и 15.10 «Wily») команды немного отличаются:

```
$ sudo sh -c 'echo "deb \
http://repo.postgrespro.ru/pgpro-9.5/ubuntu \
$(lsb_release -cs) main" > \
/etc/apt/sources.list.d/postgrespro.list'
```

Дальше все одинаково для обеих систем:

```
$ wget --quiet -O - http://repo.postgrespro.ru/pgpro\
-9.5/keys/GPG-KEY-POSTGRES PRO-95 | sudo apt-key add -
$ sudo apt-get update
```

Перед установкой проверьте настройки локализации:

```
$ locale
```

Если вы планируете хранить данные на русском языке, значение переменных LC\_CTYPE и LC\_COLLATE должно быть равно «ru\_RU.UTF8» (значение «en\_US.UTF8» тоже подходит, но менее предпочтительно). При необходимости установите эти переменные:

```
$ export LC_CTYPE=ru_RU.UTF8
$ export LC_COLLATE=ru_RU.UTF8
```

Также убедитесь, что в операционной системе установлена соответствующая локаль:

```
$ locale -a | grep ru_RU
ru_RU.utf8
```

Если это не так, сгенерируйте ее:

```
$ sudo locale-gen ru_RU.utf8
```

Теперь можно приступить к установке. Дистрибутив включает много разных пакетов, но для того, чтобы начать работу, достаточно одного основного:

```
$ sudo apt-get install postgrespro-9.5
```

Как только эта команда выполнится, СУБД PostgreSQL будет установлена, запущена и готова к работе. Чтобы проверить это, выполните команду:

```
$ sudo -u postgres psql -c 'select now();'
```

Если все проделано успешно, в ответ вы должны получить текущее время.

## Управление службой и основные файлы

При установке PostgreSQL на вашей системе автоматически был создан специальный пользователь `postgres`, под именем которого работают процессы, обслуживающие сервер, и которому принадлежат все файлы, относящиеся к СУБД. PostgreSQL будет автоматически запускаться при перезагрузке операционной системы. С настройками по умолчанию это не проблема: если вы не работаете с сервером базы данных, он потребляет совсем немного ресурсов вашей системы. Если вы все-таки захотите отключить автозапуск, вам придется отредактировать файл `/etc/postgresql/9.5/main/start.conf`.

Чтобы временно остановить службу сервера баз данных, выполните команду:

```
$ service postgresql stop 9.5
```

Запустить службу сервера можно командой:

```
$ service postgresql start 9.5
```

Если при запуске службы произошла ошибка, для поиска причины следует просмотреть журнал сообщений сервера, который находится в файле  
`/var/log/postgresql/postgresql-9.5-main.log`.

Полный список команд управления службой можно получить, выполнив:

```
$ service postgresql
```

Вся информация, которая попадает в базу данных, располагается в файловой системе в каталоге  
`/var/lib/pgsql/9.5/main/` — убедитесь, что у вас достаточно свободного места, если собираетесь хранить много данных.

Есть несколько важных конфигурационных файлов, которые определяют настройки сервера. Для начала работы вам не придется их изменять, но в них имеет смысл заглянуть (они прекрасно документированы), потому что в дальнейшем они вам непременно понадобятся:

- `/etc/postgresql/9.5/main/postgresql.conf` — основной конфигурационный файл, содержащий значения параметров сервера;
- `/etc/postgresql/9.5/main/pg_hba.conf` — файл, определяющий настройки доступа. В целях безопасности по умолчанию доступ разрешен только с локального компьютера и только для пользователя ОС `postgres`.

Самое время подключиться к базе данных и попробовать SQL в деле.

# Пробуем SQL

## Подключение с помощью psql

Чтобы подключиться к серверу СУБД и выполнить какие-либо команды, требуется программа-клиент. В главе «PostgreSQL для приложения» мы будем говорить о том, как посыпать запросы из разных языков программирования, а сейчас речь пойдет о терминальном клиенте `psql`, работа с которых происходит интерактивно в режиме командной строки.

К сожалению, многие недолюбливают командную строку. Почему же имеет смысл научиться с ней работать?

Во-первых, `psql` — стандартный клиент, входящий в любую сборку PostgreSQL, и поэтому он всегда под рукой. Безусловно, хорошо иметь настроенную под себя среду, но нет никакого резона оказаться беспомощным в незнакомом окружении.

Во-вторых, `psql` действительно удобен для повседневных задач по администрированию баз данных, для написания небольших запросов и автоматизации процессов, например, для периодической установки изменений программного кода на сервер СУБД. Он имеет собственные команды, позволяющие сориентироваться в объектах, хранящихся в базе данных и удобно представить информацию из таблиц.

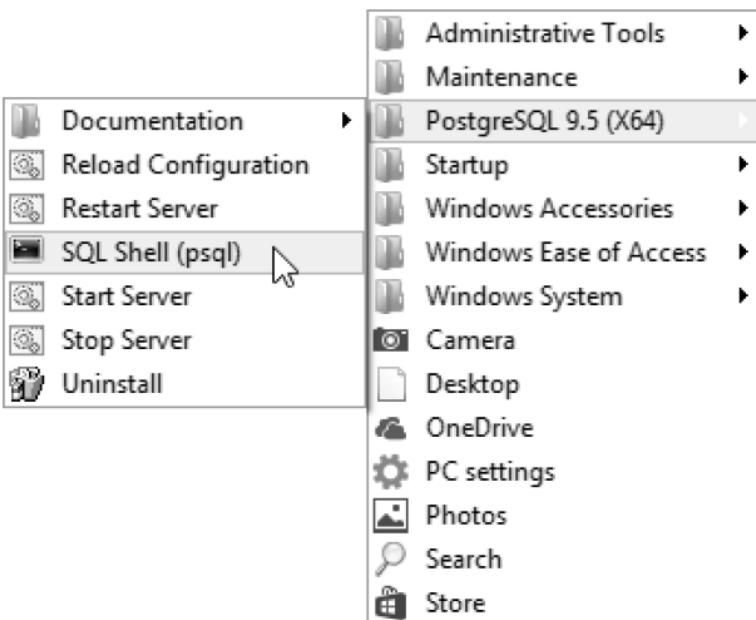
Но если вы привыкли работать с графическими пользовательскими интерфейсами, попробуйте `pgAdmin3` — мы еще упомянем эту программу ниже — или другие аналогичные продукты:  
[wiki.postgresql.org/wiki/Community\\_Guide\\_to\\_PostgreSQL\\_GUI\\_Tools](http://wiki.postgresql.org/wiki/Community_Guide_to_PostgreSQL_GUI_Tools)

Чтобы запустить `psql`, в системе Linux выполните команду:

```
$ sudo -u postgres psql
```



В Windows запустите программу «SQL Shell (psql)» из папки меню «Пуск», которую вы указали при установке:



В ответ на запрос введите пароль пользователя `postgres`, который вы указали при установке PostgreSQL.

Пользователи Windows могут столкнуться с неправильным отображением русских букв в терминале. Если вместо букв вы видите закорючки, сделайте две вещи:

1. Закройте окно терминала и отредактируйте файл «SQL Shell (psql)» — добавьте в него первой строкой команду:  
`chcp 1251`
2. Снова запустите psql и в свойствах окна измените шрифт с растрового на TrueType (обычно «Lucida Console»).

В итоге и в одной, и в другой операционной системе вы увидите одинаковое приглашение `postgres=#`. «Postgres» здесь — имя базы данных, к которой вы сейчас подключены. Один сервер PostgreSQL может одновременно обслуживать несколько баз данных, но одновременно вы работаете только с одной из них.

Дальше мы будем приводить некоторые команды. Вводите только то, что выделено жирным шрифтом; приглашение и ответ системы на команду приведены исключительно для удобства.

## База данных

Давайте создадим новую базу данных с именем `test`. Выполните:

```
postgres=# CREATE DATABASE test;
CREATE DATABASE
```

Не забудьте про точку с запятой в конце команды — пока PostgreSQL не увидит этот этот символ, он будет считать, что вы продолжаете ввод (так что команду можно разбить на несколько строк).

Теперь переключимся на созданную базу:

```
postgres=# \c test
You are now connected to database "test" as user
"postgres".
test=#
```

Как вы видите, приглашение сменилось на `test=#`.

Команда, которую мы только что ввели, не похожа на SQL — она начинается с обратной косой черты. Так выглядят специальные команды, которые понимает только psql (поэтому, если у вас открыт pgAdmin или другое графическое средство, пропускайте все, что начинается на косую черту, или попытайтесь найти аналог).

Команд `psql` довольно много и с некоторыми из них мы познакомимся чуть позже, а полный список с кратким описанием можно получить прямо сейчас:

```
test=# \?
```

Поскольку справочная информация довольно объемна, она будет показана с помощью настроенной в операционной системе команды-пейджера; обычно это `more` или `less`.

## Таблицы

В реляционных СУБД данные представляются в виде *таблиц*. Заголовок таблицы определяет *столбцы*; собственно данные располагаются в *строках*. Данные не упорядочены (в частности, нельзя полагаться на то, что строки хранятся в том порядке, в котором они добавлялись в таблицу).

Для каждого столбца устанавливается *тип* данных, а значения соответствующих полей строк должны удовлетворять этим типам. PostgreSQL располагает большим числом встроенных типов данных (полный список: [postgrespro.ru/doc/datatype.html](http://postgrespro.ru/doc/datatype.html)) и возможностями для создания новых, но мы ограничимся всего несколькими:

- `integer` — целые числа;
- `text` — текстовые строки;
- `boolean` — логический тип, принимающий значения `true` (истина) или `false` (ложь).

Помимо обычных значений, определяемых типом данных, поле может иметь *неопределенное значение* `null` — его можно рассматривать как «значение неизвестно» или «значение не задано».



Давайте создадим таблицу дисциплин, читаемых в ВУЗе:

```
test=# CREATE TABLE courses(  
test(#   c_no text PRIMARY KEY,  
test(#   title text,  
test(#   hours integer  
test(# );  
CREATE TABLE
```

Обратите внимание, как меняется приглашение psql: это подсказка, что ввод команды продолжается на новой строке. (В дальнейшем для удобства мы не будем дублировать приглашение на каждой строке.)

В этой команде мы определили, что таблица с именем `courses` будет состоять из трех столбцов: `c_no` — текстовый номер курса, `title` — название курса, и `hours` — целое число лекционных часов.

Кроме столбцов и типов данных, мы можем определить ограничения целостности, которые будут автоматически проверяться — СУБД не допустит появление в базе некорректных данных. В нашем примере мы добавили ограничение `primary key` для столбца `c_no`, которое говорит о том, что значения в этом столбце должны быть уникальными, а неопределенные значения не допускаются. Такой столбец можно использовать для того, чтобы отличить одну строку в таблице от других.

Полный список ограничений целостности:  
[postgrespro.ru/doc/ddl-constraints.html](http://postgrespro.ru/doc/ddl-constraints.html).



Точный синтаксис команды `create table` можно посмотреть в документации, либо попросить справку прямо в psql:

```
test=# \help create table
```

Такая справка есть по каждой команде SQL, а полный список команд легко получить с помощью `\help` без параметров.

# Наполнение таблиц

Добавим в созданную таблицу несколько строк:

```
test=# INSERT INTO courses(c_no, title, hours)
VALUES ('CS301', 'Базы данных', 30),
       ('CS305', 'Сети ЭВМ', 60);
INSERT 0 2
```

Если вам требуется массовая загрузка данных из внешнего источника, команда `insert` — не лучший выбор; посмотрите на специально предназначенную для этого команду `copy`:  
[postgrespro.ru/doc/sql-copy.html](http://postgrespro.ru/doc/sql-copy.html).



Для дальнейших примеров нам потребуется еще две таблицы: студенты и экзамены. Для каждого студента мы будем хранить его имя и год поступления; идентифицироваться он будет числовым номером студенческого билета.

```
test=# CREATE TABLE students(
  s_id integer PRIMARY KEY,
  name text,
  start_year integer
);
CREATE TABLE
test=# INSERT INTO students(s_id, name, start_year)
VALUES (1451, 'Анна', 2014),
       (1432, 'Виктор', 2014),
       (1556, 'Нина', 2015);
INSERT 0 3
```

Экзамен содержит оценку, полученную студентом по некоторой дисциплине. Таким образом, студенты и дисциплины связаны друг с другом отношением «многие ко многим»: один студент может сдавать экзамены по многим дисциплинам, а экзамен по одной дисциплине могут сдавать много студентов.

Запись в таблице экзаменов идентифицируется совокупностью имени студента и номера курса. Такое ограничение целостности, относящее сразу к нескольким столбцам, определяется с помощью фразы constraint:

```
test=# CREATE TABLE exams(
    s_id integer REFERENCES students(s_id),
    c_no text REFERENCES courses(c_no),
    score integer,
    CONSTRAINT pk PRIMARY KEY(s_id, c_no)
);
CREATE TABLE
```

Кроме того, с помощью фразы references мы определили два ограничения ссылочной целостности, называемые *внешними ключами*. Такие ключи показывают, что значения в одной таблице *ссылаются* на строки в другой таблице. Теперь при любых действиях СУБД будет проверять, что все идентификаторы s\_id, указанные в таблице экзаменов, соответствуют реальным студентам (то есть записям в таблице студентов), а номера c\_no — реальным курсам. Таким образом, будет исключена возможность оценить несуществующего студента или поставить оценку по несуществующей дисциплине — независимо от действий пользователя или возможных ошибок в приложении.

Поставим нашим студентам несколько оценок:

```
test=# INSERT INTO exams(s_id, c_no, score)
VALUES (1451, 'CS301', 5),
        (1556, 'CS301', 5),
        (1451, 'CS305', 5),
        (1432, 'CS305', 4);
INSERT 0 4
```

# Выборка данных

## Простые запросы

Чтение данных из таблиц выполняется оператором `select`.  
Например, выведем два столбца из таблицы `courses`:

```
test=# SELECT title AS course_title, hours
FROM courses;
course_title | hours
-----+-----
Базы данных |    30
Сети ЭВМ     |    60
(2 rows)
```

Конструкция `as` позволяет переименовать столбец, если это необходимо. Чтобы вывести все столбцы, достаточно указать символ звездочки:

```
test=# SELECT * FROM courses;
c_no | title | hours
-----+-----+
CS301 | Базы данных |    30
CS305 | Сети ЭВМ     |    60
(2 rows)
```

В результирующей выборке мы можем получить несколько одинаковых строк. Даже если все строки были различны в исходной таблице, дубликаты могут появиться, если выводятся не все столбцы:

```
test=# SELECT start_year FROM students;
start_year
-----
2014
2014
2015
(3 rows)
```

Чтобы выбрать все *различные* года поступления, после `select` надо добавить слово `distinct`:

```
test=# SELECT DISTINCT start_year FROM students;
          start_year
-----
          2014
          2015
(2 rows)
```



Подробнее смотрите в документации:

[postgrespro.ru/doc/sql-select.html#SQL-DISTINCT](http://postgrespro.ru/doc/sql-select.html#SQL-DISTINCT)

Вообще после слова `select` можно указывать не только столбцы, но и любые выражения. А если не указать фразу `from`, то результирующая таблица будет содержать одну строку.

Например:

```
test=# SELECT 2+2 AS result;
      result
-----
          4
(1 row)
```

Обычно при выборке данных требуется получить не все строки, а только удовлетворяющие какому-либо условию. Такое условие фильтрации записывается во фразе `where`:

```
test=# SELECT * FROM courses WHERE hours > 45;
   c_no  |  title   | hours
-----+-----+-----+
  CS305 | Сети ЭВМ |      60
(1 row)
```

Условие должно иметь логический тип. Например, оно может содержать отношения `=`, `<>` (или `!=`), `>`, `>=`, `<`, `<=`; может объединять более простые условия с помощью логических

операций `and`, `or`, `not` и круглых скобок — как в обычных языках программирования.

Тонкий момент представляет собой неопределенное значение `null`. В результирующую таблицу попадают только те строки, для которых условие фильтрации истинно; если же значение ложно или не определено, строка отбрасывается.

Учтите:

- результат сравнения чего-либо с неопределенным значением не определен;
- результат логических операций с неопределенным значением как правило не определен (исключения: `true or null = true`, `false and null = false`);
- для проверки определенности значения используются специальные отношения `is null` (`is not null`) и `is distinct from` (`is not distinct from`), а также бывает удобно воспользоваться функцией `coalesce`.

Подробнее смотрите в документации:  
[postgrespro.ru/doc/functions-comparison.html](http://postgrespro.ru/doc/functions-comparison.html)



## Соединения

Грамотно спроектированная база данных не содержит избыточных данных. Например, таблица экзаменов не должна содержать имя студента, потому что его можно найти в другой таблице по номеру студенческого билета.

Поэтому для получения всех необходимых значений в запросе часто приходится *соединять* данные из нескольких таблиц, перечисляя их имена во фразе `from`:

```
test=# SELECT * FROM courses, exams;
 c_no | title      | hours | s_id | c_no | score
-----+-----+-----+-----+-----+
 CS301 | Базы данных |    30 | 1451 | CS301 |    5
 CS305 | Сети ЭВМ   |    60 | 1451 | CS301 |    5
 CS301 | Базы данных |    30 | 1556 | CS301 |    5
 CS305 | Сети ЭВМ   |    60 | 1556 | CS301 |    5
 CS301 | Базы данных |    30 | 1451 | CS305 |    5
 CS305 | Сети ЭВМ   |    60 | 1451 | CS305 |    5
 CS301 | Базы данных |    30 | 1432 | CS305 |    4
 CS305 | Сети ЭВМ   |    60 | 1432 | CS305 |    4
(8 rows)
```

То, что у нас получилось, называется прямым или декартовым произведением таблиц — к каждой строке одной таблицы добавляется каждая строка другой.

Как правило, более полезный и содержательный результат можно получить, указав во фразе `where` условие соединения. Получим оценки по всем дисциплинам, сопоставляя курсы с теми экзаменами, которые проводились именно по данному курсу:

```
test=# SELECT courses.title, exams.s_id, exams.score
FROM courses, exams
WHERE courses.c_no = exams.c_no;
      title | s_id | score
-----+-----+-----+
 Базы данных | 1451 |    5
 Базы данных | 1556 |    5
 Сети ЭВМ     | 1451 |    5
 Сети ЭВМ     | 1432 |    4
(4 rows)
```

Запросы можно формулировать и в другом виде, указывая соединения с помощью ключевого слова `join`. Выведем студентов и их оценки по курсу «Сети ЭВМ»:

```
test=# SELECT students.name, exams.score
FROM students
JOIN exams
  ON students.s_id = exams.s_id
  AND exams.c_no = 'CS305';
name | score
-----+
Анна |      5
Виктор |      4
(2 rows)
```

С точки зрения СУБД обе формы эквивалентны, так что можно использовать тот способ, который представляется более наглядным.

Этот пример показывает, что в результат не включаются строки исходной таблицы, для которых не нашлось пары в другой таблице: хотя условие наложено на дисциплины, но при этом исключаются и студенты, которые не сдавали экзамен по данной дисциплине. Чтобы в выборку попали все студенты, независимо от того, сдавали они экзамен или нет, надо использовать операцию внешнего соединения:

```
test=# SELECT students.name, exams.score
FROM students
LEFT JOIN exams
  ON students.s_id = exams.s_id
  AND exams.c_no = 'CS305';
name | score
-----+
Анна |      5
Виктор |      4
Нина |
(3 rows)
```

В этом примере в результат добавляются строки из левой таблицы (поэтому операция называется *left join*), для которых не нашлось пары в правой. При этом для столбцов правой таблицы возвращаются неопределенные значения.

Условия во фразе `where` применяются к уже готовому результату соединений, поэтому, если вынести ограничение на дисциплины из условия соединения, Нина не попадет в выборку — ведь для нее `exams.c_no` не определен:

```
test=# SELECT students.name, exams.score
  FROM students
LEFT JOIN exams ON students.s_id = exams.s_id
 WHERE exams.c_no = 'CS305';
   name | score
-----+-----
 Анна   |     5
 Виктор |     4
(2 rows)
```

Не стоит опасаться соединений. Это обычная и естественная для реляционных СУБД операция, и у PostgreSQL имеется целый арсенал эффективных механизмов для ее выполнения. Не соединяйте данные в приложении, доверьте эту работу серверу баз данных — он прекрасно с ней справляется.

Подробнее смотрите в документации:  
[postgrespro.ru/doc/sql-select.html#SQL-FROM](http://postgrespro.ru/doc/sql-select.html#SQL-FROM)



## Подзапросы

Оператор `select` формирует таблицу, которая (как мы уже видели) может быть выведена в качестве результата, а может быть использована в другой конструкции языка SQL в любом месте, где по смыслу может находиться таблица. Такая вложенная команда `select`, заключенная в круглые скобки, называется *подзапросом*.

Если подзапрос возвращает одну строку и один столбец, его можно использовать как обычное скалярное выражение:

```
test=# SELECT name,
  (SELECT score
   FROM exams
   WHERE exams.s_id = students.s_id
   AND exams.c_no = 'CS305')
FROM students;
      name    | score
-----+-----
Анна     |      5
Виктор   |      4
Нина     |
(3 rows)
```

Если подзапрос, использованный в списке выражений select, не содержит ни одной строки, возвращается неопределенное значение (как в последней строке результата примера).

Такие скалярные подзапросы можно использовать и в условиях фильтрации. Получим все экзамены, которые сдавали студенты, поступившие после 2014 года:

```
test=# SELECT *
FROM exams
WHERE (SELECT start_year
       FROM students
       WHERE students.s_id = exams.s_id) > 2014;
      s_id | c_no    | score
-----+-----+
  1556 | CS301   |      5
(1 row)
```

В SQL можно формулировать условия и на подзапросы, возвращающие произвольное количество строк. Для этого существует несколько конструкций, одна из которых — отношение `in` — проверяет, содержится ли значение в таблице, возвращаемой подзапросом.

Выведем студентов, получивших оценки по указанному курсу:

```
test=# SELECT name, start_year
FROM students
WHERE s_id IN (SELECT s_id
                FROM exams
                WHERE c_no = 'CS305');
      name | start_year
-----+-----
    Анна |      2014
   Виктор |      2014
(2 rows)
```

Вариантом является отношение `not in`, возвращающее противоположный результат. Например, список студентов, получивших только отличные оценки (то есть не получивших более низкие оценки):

```
test=# SELECT name, start_year
FROM students
WHERE s_id NOT IN (SELECT s_id
                     FROM exams
                     WHERE score < 5);
      name | start_year
-----+-----
    Анна |      2014
    Нина |      2015
(2 rows)
```

Другая возможность — предикат `exists`, проверяющий, что подзапрос возвратил хотя бы одну строку. С его помощью можно записать предыдущий запрос в другом виде:

```
test=# SELECT name, start_year
FROM students
WHERE NOT EXISTS (SELECT s_id
                     FROM exams
                     WHERE exams.s_id = students.s_id
                     AND score < 5);
```

Подробнее смотрите в документации:  
[postgrespro.ru/doc/functions-subquery.html](http://postgrespro.ru/doc/functions-subquery.html)

В примерах выше мы уточняли имена столбцов названиями таблиц, чтобы избежать неоднозначности. Иногда этого недостаточно. Например, в запросе одна и та же таблица может участвовать два раза, или вместо таблицы в предложении `from` мы можем использовать безымянный подзапрос. В этих случаях после подзапроса можно указать произвольное имя, которое называется псевдонимом (alias). Псевдонимы можно использовать и для обычных таблиц.



Имена студентов и их оценки по предмету «Базы данных»:

```
test=# SELECT s.name, ce.score
FROM students s
JOIN (SELECT exams.* 
      FROM courses, exams
      WHERE courses.c_no = exams.c_no
      AND courses.title = 'Базы данных') ce
    ON s.s_id = ce.s_id;
name | score
-----+
Анна |      5
Нина |      5
(2 rows)
```

Здесь `s` — псевдоним таблицы, а `ce` — псевдоним подзапроса. Псевдонимы обычно выбирают так, чтобы они были короткими, но оставались понятными.

Тот же запрос можно записать и без подзапросов, например, так:

```
test=# SELECT s.name, e.score
FROM students s, courses c, exams e
WHERE c.c_no = e.c_no
AND c.title = 'Базы данных'
AND s.s_id = e.s_id;
```

## Сортировка

Как уже говорилось, данные в таблицах не упорядочены, но часто бывает важно получить строки результата в строго определенном порядке. Для этого используется предложение `order by` со списком выражений, по которым надо выполнить сортировку. После каждого выражения (ключа сортировки) можно указать направление: `asc` — по возрастанию (этот порядок используется по умолчанию) или `desc` — по убыванию.

```
test=# SELECT * FROM exams
ORDER BY score, s_id, c_no DESC;
s_id | c_no | score
-----+-----+
1432 | CS305 |      4
1451 | CS305 |      5
1451 | CS301 |      5
1556 | CS301 |      5
(4 rows)
```

Здесь строки упорядочены сначала по возрастанию оценки, для совпадающих оценок — по возрастанию номера студенческого билета, а при совпадении первых двух ключей — по убыванию номера курса.

Операцию сортировки имеет смысл выполнять в конце запроса непосредственно перед получением результата; в подзапросах она обычно бессмысленна.



Подробнеесмотрите в документации:  
[postgrespro.ru/doc/sql-select.html#SQL-ORDERBY](http://postgrespro.ru/doc/sql-select.html#SQL-ORDERBY)

## Группировка

При группировке в одной строке результата размещается значение, вычисленное на основании данных нескольких строк исходных таблиц. Вместе с группировкой используют

*агрегатные функции.* Например, выведем общее количество проведенных экзаменов, количество сдававших их студентов и средний балл:

```
test=# SELECT count(*), count(distinct s_id),
avg(score)
FROM exams;
 count | count |      avg
-----+-----+
 4 |     3 | 4.750000000000000000000000000000
(1 row)
```

Аналогичную информацию можно получить в разбивке по номерам курсов с помощью предложения group by, в котором указываются ключи группировки:

```
test=# SELECT c_no, count(*), count(DISTINCT s_id),
avg(score)
FROM exams
GROUP BY c_no;
 c_no | count | count |      avg
-----+-----+-----+
CS301 |     2 |     2 | 5.000000000000000000000000000000
CS305 |     2 |     2 | 4.500000000000000000000000000000
(2 rows)
```

Полный список агрегатных функций:

[postgrespro.ru/doc/functions-aggregate.html](http://postgrespro.ru/doc/functions-aggregate.html).

В запросах, использующих группировку, может возникнуть необходимость отфильтровать строки на основании результатов агрегирования. Такие условия можно задать в предложении having. Отличие от where состоит в том, что условия where применяются до группировки (в них можно использовать столбцы исходных таблиц), а условия having — после группировки (и в них можно также использовать столбцы таблицы-результата).



Выберем имена студентов, получивших более одной пятерки по любому предмету:

```
test=# SELECT students.name
  FROM students, exams
 WHERE students.s_id = exams.s_id AND exams.score = 5
 GROUP BY students.name
 HAVING count(*) > 1;
      name
 -----
    Анна
(1 row)
```



Подробнее смотрите в документации:

[postgrespro.ru/doc/sql-select.html#SQL-GROUPBY](http://postgrespro.ru/doc/sql-select.html#SQL-GROUPBY)

## Изменение и удаление данных

Изменение данных в таблице выполняет оператор update, в котором указываются новые значения полей для строк, определяемых предложением where (таким же, как в операторе select).

Например, увеличим число лекционных часов для курса «Базы данных» в два раза:

```
test=# UPDATE courses
SET hours = hours*2
WHERE c_no = 'CS301';
UPDATE 1
```



Подробнее смотрите в документации:

[postgrespro.ru/doc/sql-update.html](http://postgrespro.ru/doc/sql-update.html)

Оператор `delete` удаляет из указанной таблицы строки, определяемые все тем же предложением `where`:

```
test=# DELETE FROM exams WHERE score < 5;  
DELETE 1
```

Подробнее смотрите в документации:  
[postgrespro.ru/doc/sql-delete.html](http://postgrespro.ru/doc/sql-delete.html)



## Транзакции

Давайте немного расширим нашу схему данных и распределим студентов по группам. При этом потребуем, чтобы у каждой группы в обязательном порядке был староста. Для этого создадим таблицу групп:

```
test=# CREATE TABLE groups(  
    g_no text PRIMARY KEY,  
    headman integer NOT NULL REFERENCES students(s_id)  
)  
CREATE TABLE
```

Здесь мы использовали ограничение целостности `not null`, которое запрещает неопределенные значения.

Теперь в таблице студентов нам необходимо еще одно поле — номер группы, — о которым мы не подумали при создании. К счастью, в уже существующую таблицу можно добавить новый столбец:

```
test=# ALTER TABLE students ADD g_no text REFERENCES  
groups(g_no);  
ALTER TABLE
```

С помощью команды `psql` всегда можно посмотреть, какие поля определены в таблице:

```
test=# \d students
      Table "public.students"
 Column | Type | Modifiers
-----+-----+-----
 s_id   | integer | not null
 name   | text |
 start_year | integer |
 g_no   | text |
 ...

```

Также можно вспомнить, какие вообще у нас есть таблицы:

```
test=# \d
      List of relations
 Schema | Name | Type | Owner
-----+-----+-----+
 public | courses | table | postgres
 public | exams | table | postgres
 public | groups | table | postgres
 public | students | table | postgres
(4 rows)
```

Создадим теперь группу «А-101» и поместим в нее всех студентов, а старостой сделаем Анну.

Тут возникает затруднение. С одной стороны, мы не можем создать группу, не указав старосту. А с другой, как мы можем назначить Анну старостой, если она еще не входит в группу? Это привело бы к тому, что в базе данных некоторое время (пусть и небольшое) находились бы логически некорректные, несогласованные данные.

Мы столкнулись с тем, что две операции надо совершить одновременно, потому что ни одна из них не имеет смысла без другой. Такие операции, составляющие неделимую единицу работы, называются *транзакцией*.

Начнем транзакцию:

```
test=# BEGIN;  
BEGIN
```

Затем добавим группу вместе со старостой, использовав запрос в команде добавления строк:

```
test=# INSERT INTO groups(g_no, headman)  
SELECT 'A-101', s_id  
FROM students  
WHERE name = 'Анна';  
INSERT 0 1
```

Откройте теперь новое окно терминала и запустите еще один процесс psql: это будет сеанс, работающий параллельно с первым.

Увидит ли он наши изменения (чтобы не запутаться, команды второго сеанса мы будем показывать с отступом)?

```
postgres=# \c test  
You are now connected to database "test" as  
user "postgres".  
test=# SELECT * FROM groups;  
 g_no | headman  
-----+-----  
(0 rows)
```

Нет, не увидит, ведь транзакция еще не завершена.

Теперь переведем всех студентов в созданную группу:

```
test=# UPDATE students  
SET g_no = 'A-101';  
UPDATE 3
```

И снова второй сеанс видит согласованные данные, актуальные на начало еще не оконченной транзакции:

```
test=# SELECT * FROM students;
 s_id | name    | start_year | g_no
-----+-----+-----+-----+
 1451 | Анна   |      2014 | 
 1432 | Виктор |      2014 | 
 1556 | Нина   |      2015 | 
(3 rows)
```

А теперь завершим транзакцию, зафиксировав изменения:

```
test=# COMMIT;
COMMIT
```

И только в этот момент второму сеансу становятся доступны все изменения, сделанные в транзакции, как будто они появились одномоментно:

```
test=# SELECT * FROM groups;
 g_no | headman
-----+-----
 A-101 |     1451
(1 row)

test=# SELECT * FROM students;
 s_id | name    | start_year | g_no
-----+-----+-----+-----+
 1451 | Анна   |      2014 | A-101
 1432 | Виктор |      2014 | A-101
 1556 | Нина   |      2015 | A-101
(3 rows)
```

СУБД гарантирует выполнение нескольких важных свойств.

Во-первых, транзакция либо выполняется целиком (как в нашем примере), либо не выполняется совсем. Если бы в одной из команд произошла ошибка, или мы сами прервали бы

транзакцию командой `rollback`, то база данных осталась бы в том состоянии, в котором она была до команды `begin`. Это свойство называется *атомарностью*.

Во-вторых, когда фиксируются изменения транзакции, все ограничения целостности должны быть выполнены, иначе транзакция прерывается. Таким образом, в начале транзакции данные находятся в согласованном состоянии, и в конце своей работы транзакция оставляет их согласованными; это свойство так и называется — *согласованность*.

В-третьих, как мы убедились на примере, другие пользователи никогда не увидят несогласованные данные, которые транзакция еще не зафиксировала. Это свойство называется *изоляцией*; за счет его соблюдения СУБД способна параллельно обслуживать много сеансов. Особенностью PostgreSQL является очень эффективная реализация изоляции: несколько сеансов могут одновременно читать и изменять данные, не блокируя друг друга. Блокировка возникает только при одновременном изменении одной и той же строки двумя разными процессами.

И в-четвертых, гарантируется *долговечность*: зафиксированные данные не пропадут даже в случае сбоя (конечно, при правильных настройках и регулярном выполнении резервного копирования).

Это крайне полезные свойства, без которых невозможно представить себе реляционную систему управления базами данных.



Подробнее о транзакциях:

[postgrespro.ru/doc/tutorial-transactions.html](http://postgrespro.ru/doc/tutorial-transactions.html)  
(и еще более подробно: [postgrespro.ru/doc/mvcc.html](http://postgrespro.ru/doc/mvcc.html)).

# Полезные команды psql

\?	Справка по командам psql.
\h	Справка по SQL: список доступных команд или синтаксис конкретной команды.
\x	Переключает обычный табличный вывод (столбцы и строки) на расширенный (каждый столбец на отдельной строке) и обратно. Удобно для просмотра нескольких «широких» строк.
\l	Список баз данных.
\du	Список пользователей.
\dt	Список таблиц.
\di	Список индексов.
\dv	Список представлений.
\df	Список функций.
\dn	Список схем.
\dx	Список установленных расширений.
\dp	Список привилегий.
\d имя	Подробная информация по конкретному объекту.
\d+ имя	Еще более подробная информация по конкретному объекту.
\timing on	Показывать время выполнения операторов.

# Заключение

Конечно, мы успели осветить только малую толику того, что необходимо знать о СУБД, но надеемся, что вы убедились: начать использовать PostgreSQL совсем нетрудно. Язык SQL позволяет формулировать запросы самой разной сложности, а PostgreSQL предоставляет качественную поддержку стандарта и эффективную реализацию. Пробуйте, экспериментируйте!

И еще одна важная команда `psql`: для того, чтобы завершить сеанс работы, наберите

```
test=# \q
```

# PostgreSQL для приложения

## Отдельный пользователь

В предыдущей главе мы подключались к серверу баз данных под пользователем `postgres`, единственным существующим сразу после установки СУБД. Но `postgres` обладает правами суперпользователя, поэтому приложению не следует использовать его для подключения к базе данных. Лучше создать нового пользователя и сделать его владельцем отдельной базы данных — тогда его права будут ограничены этой базой.

```
postgres=# CREATE USER app password 'p@ssw0rd';
CREATE ROLE
postgres=# CREATE DATABASE appdb owner app;
CREATE DATABASE
```

Подробнее про пользователей и привилегии:  
[postgrespro.ru/doc/user-manag.html](http://postgrespro.ru/doc/user-manag.html)  
[и postgrespro.ru/doc/ddl-priv.html](http://postgrespro.ru/doc/ddl-priv.html).



Чтобы подключиться к новой базе данных и работать с ней от имени созданного пользователя, выполните команду:

```
postgres=# \c appdb app localhost 5432
Password for user app: ***
You are now connected to database "appdb" as user
"app" on host "127.0.0.1" at port "5432".
appdb=>
```

В команде указываются последовательно имя базы данных (`appdb`), имя пользователя (`app`), узел (`localhost`

или 127.0.0.1) и номер порта (5432). Обратите внимание, что подсказка изменилась: вместо «решетки» (#) теперь отображается символ «больше» (>) — решетка указывает на суперпользователя по аналогии с пользователем root в юниксе.

Со своей базой данных пользователь app может работать без ограничений. Например, можно создать в ней таблицу:

```
appdb=> CREATE TABLE greeting(s text);
CREATE TABLE
appdb=> INSERT INTO greeting VALUES ('Привет, мир!');
INSERT 0 1
```

## Удаленное подключение

В нашем примере клиент и СУБД находятся на одном и том же компьютере. Разумеется, вы можете установить PostgreSQL на выделенный сервер, а подключаться к нему с другой машины (например, с сервера приложений). В этом случае вместо localhost надо указать адрес вашего сервера СУБД. Но этого недостаточно: по умолчанию из соображений безопасности PostgreSQL допускает только локальные подключения.

Чтобы подключиться к базе данных снаружи, необходимо отредактировать два файла.

Во-первых, файл основных настроек `postgresql.conf` (расположение конфигурационных файлов мы обсуждали при установке). Найдите строку, определяющую, какие сетевые интерфейсы слушает PostgreSQL:

```
#listen_addresses = 'localhost'
```

и замените ее на:

```
listen_addresses = '*'
```

Во-вторых, файл настроек аутентификации pg\_hba.conf.

Допишите в конец файла следующую строку:

```
host appdb app all md5
```

Это разрешит доступ к базе данных appdb пользователю app с любого адреса при указании верного пароля.

Подробнее о настройках аутентификации:  
[postgrespro.ru/doc/client-authentication.html](http://postgrespro.ru/doc/client-authentication.html)



## Проверка связи

Для того, чтобы обратиться к PostgreSQL из программы на каком-либо языке программирования, необходимо использовать подходящую библиотеку и установить драйвер СУБД.

Ниже приведены простые примеры для нескольких популярных языков, которые помогут быстро проверить соединение с базой данных. Приведенные программы намеренно содержат только минимально необходимый код для запроса к базе данных; в частности, не предусмотрена никакая обработка ошибок. Не стоит рассматривать эти фрагменты как пример для подражания.

Если вы используете ОС Windows, не забудьте в окне Command Prompt сменить шрифт с растрового на True Type (например, на «Lucida Console») и выполнить команды:

```
C:\> chcp 2151  
Active code page: 1251  
C:\> set PGCLIENTENCODING=WIN1251
```

# PHP

В языке PHP работа с PostgreSQL организована с помощью специального расширения. В Linux кроме самого PHP нам потребуется пакет с этим расширением:

```
$ sudo apt-get install php5-cli  
$ sudo apt-get install php5pgsql
```

PHP для Windows можно установить с сайта [windows.php.net/download](http://windows.php.net/download). Расширение для PostgreSQL уже входит в комплект, но в файле php.ini необходимо найти и раскомментировать строку (убрать точку с запятой):

```
;extension=php_pgsql.dll
```

Пример программы (test.php):

```
<?php  
$conn = pg_connect('host=localhost port=5432 ' .  
                   'dbname=appdb user=app ' .  
                   'password=p@ssw0rd') or die;  
$query = pg_query('select * from greeting') or die;  
while ($row = pg_fetch_array($query)) {  
    echo $row[0].PHP_EOL;  
}  
pg_free_result($query);  
pg_close($conn);  
?>
```

Выполняем:

```
$ php test.php  
Привет, мир!
```



Расширение для PostgreSQL описано в документации: [php.net/manual/ru/book.pgsql.php](http://php.net/manual/ru/book.pgsql.php)

## Perl

В языке Perl работа с базами данных организована с помощью интерфейса DBI.

Сам Perl предустановлен в Debian и Ubuntu, так что дополнительно нужен только драйвер:

```
$ sudo apt-get install libdbd-pg-perl
```

Существует несколько сборок Perl для Windows, которые перечислены на сайте [www.perl.org/get.html](http://www.perl.org/get.html).

ActiveState Perl и Strawberry Perl уже включают необходимый для PostgreSQL драйвер.

Пример программы (test.pl):

```
use DBI;
my $conn = DBI->connect(
    'dbiPg:dbname=appdb;host=localhost;port=5432',
    'app', 'p@ssw0rd') or die;
my $query = $conn->prepare('select * from greeting');
$query->execute() or die;
while (my @row = $query->fetchrow_array()) {
    print @row[0]."\n";
}
$query->finish();
$conn->disconnect();
```

Выполняем:

```
$ perl test.pl
Привет, мир!
```

Интерфейс описан в документации:  
[metacpan.org/pod/DBD::Pg](http://metacpan.org/pod/DBD::Pg)



# Python

В языке Python для работы с PostgreSQL обычно используется библиотека `psycopg` (название обычно произносится как «сайко-пи-джи»). В Debian и Ubuntu язык Python версии 2 предустановлен, так что нужен только драйвер:

```
$ sudo apt-get install python-psycopg2
```

(Если вы используете Python 3, установите пакет `python3-psycopg2`.)

Python для Windows можно скачать с сайта [www.python.org](http://www.python.org).

Библиотека `psycopg` доступна на сайте проекта [initd.org/psycopg](http://initd.org/psycopg) (выберите версию, соответствующую установленной версии Python), там же находится необходимая документация.



Пример программы (`test.py`):

```
import psycopg2
conn = psycopg2.connect(
    host='localhost', port='5432', database='appdb',
    user='app', password='p@ssw0rd')
cur = conn.cursor()
cur.execute('select * from greeting')
rows = cur.fetchall()
for row in rows:
    print row[0]
conn.close()
```

Выполняем:

```
$ python test.py
```

Привет, мир!

## Java

В языке Java работа с базой данных организована через интерфейс JDBC. Предполагая, что установлен JDK 1.7:

```
$ sudo apt-get install openjdk-7-jdk
```

нам дополнительно потребуется пакет с драйвером JDBC:

```
$ sudo apt-get install libpostgresql-jdbc-java
```

JDK для Windows можно скачать с сайта  
[www.oracle.com/technetwork/java/javase/downloads](http://www.oracle.com/technetwork/java/javase/downloads).

Драйвер JDBC доступен на сайте  
[jdbc.postgresql.org](http://jdbc.postgresql.org) (выберите версию,  
соответствующую установленной версии JDK),  
там же находится вся необходимая документация.



Пример программы (Test.java):

```
import java.sql.*;
public class Test {
    public static void main(String[] args)
        throws SQLException {
        Connection conn = DriverManager.getConnection(
            "jdbc:postgresql://localhost:5432/appdb",
            "app", "p@ssw0rd");
        Statement st = conn.createStatement();
        ResultSet rs = st.executeQuery(
            "select * from greeting");
        while (rs.next()) {
            System.out.println(rs.getString(1));
        }
        rs.close();
        st.close();
        conn.close();
    }
}
```

Компилируем и выполняем, указывая путь к классу-драйверу JDBC (в Windows пути разделяются не двоеточием, а точкой с запятой):

```
$ javac Test.java  
$ java -cp .:/usr/share/java/postgresql-jdbc4.jar \  
Test  
Привет, мир!
```

# pgAdmin3

pgAdmin — графическое средство для администрирования PostgreSQL, ставшее стандартом де-факто. Программа упрощает основные задачи администрирования, отображает объекты баз данных, позволяет выполнять запросы SQL.

## Установка

Чтобы установить pgAdmin на Windows, воспользуйтесь нашей сборкой, доступной по адресу [www.postgrespro.ru/windows](http://www.postgrespro.ru/windows). Установщик задает несколько вопросов; можно оставить значения, предлагаемые по умолчанию.

В системах Debian или Ubuntu просто установите пакет pgadmin3:

```
$ sudo apt-get install pgadmin3
```



или возьмите самую свежую версию с сайта [www.pgadmin.org](http://www.pgadmin.org).

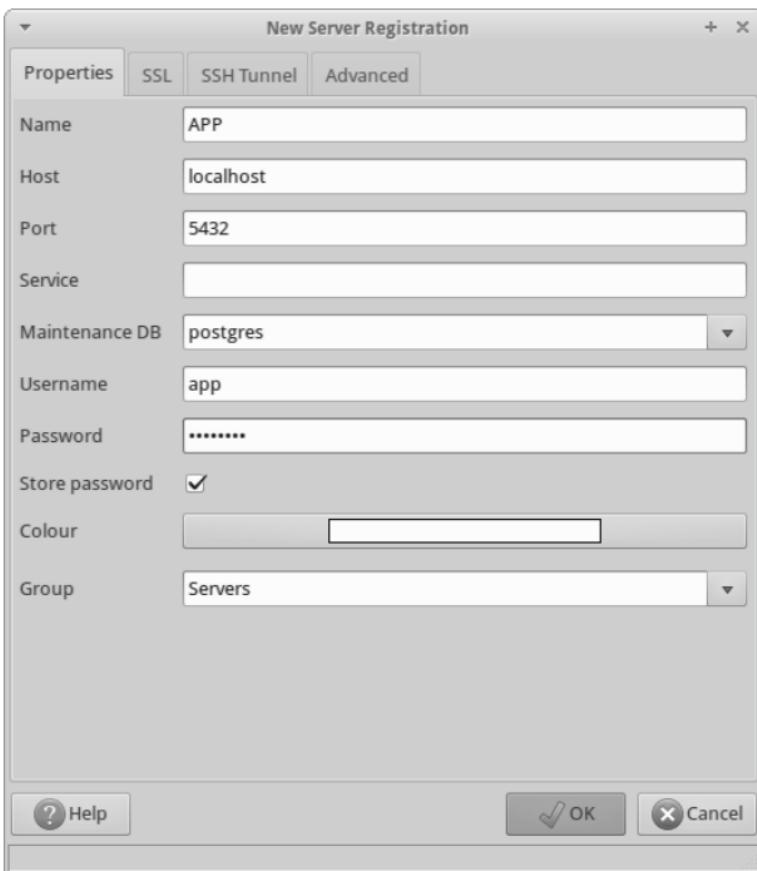
## Возможности

### Язык интерфейса

Вы можете в любой момент поменять язык интерфейса программы, выбрав в меню **File — Options...** (Файл — Параметры...) и изменив настройку **Miscellaneous — User Interface** (Разное — Интерфейс). После этого требуется перезапуск программы.

## Подключение к серверу

В первую очередь настроим подключение к серверу. Выберите в меню **File — Add Server...** (Файл — Добавить сервер...) и в появившемся окне введите произвольное имя для соединения (**Name**), имя узла (**Host**), порт (**Port**), имя пользователя (**Username**) и пароль (**Password**). Если не хотите вводить пароль каждый раз вручную, отметьте флажок **Store password** (Сохранять пароль).

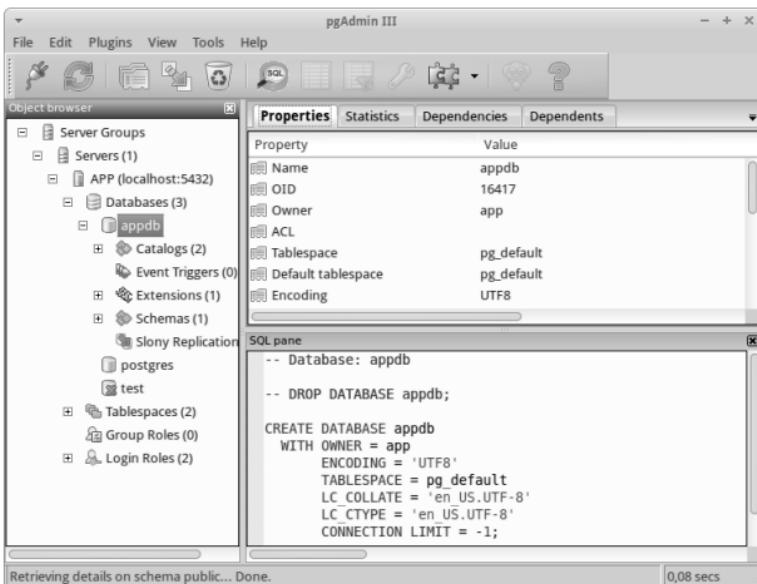


При нажатии на кнопку **OK** программа проверит доступность сервера с указанными параметрами и запомнит новое подключение.

Может оказаться, что ваша версия pgAdmin не поддерживает версию установленной базы данных. В таком случае при создании подключения будет выведено предупреждение, которые вы в большинстве случаев можете проигнорировать.

## Навигатор

В левой части основного окна находится навигатор объектов. Разворачивая пункты списка, вы можете спуститься до сервера, который мы назвали APP. Ниже будут перечислены созданные в нем базы данных: appdb мы создали для проверки подключения к PostgreSQL из разных языков программирования, test — когда знакомились с SQL, а база postgres создается автоматически при установке СУБД.



Развернув пункт **Schemas (Схемы)** для базы данных appdb, можно обнаружить и созданную нами ранее таблицу `greetings`, посмотреть ее столбцы, ограничения целостности, индексы, триггеры и т. п.

Не всегда бывает удобно разворачивать многочисленные пункты в навигаторе, чтобы добраться до интересующего нас объекта. Если известно имя или хотя бы часть имени объекта, его можно найти, выбрав в меню **Edit — Search objects... (Правка — Поиск объектов...)**; при этом в навигаторе должна быть выделена нужная база данных.

В появившемся окне введите, например, «`greet%`», при необходимости ограничьте поиск конкретным типом (таблицы, колонки и т. п.) и нажмите кнопку **Find (Найти)**. Если теперь выбрать в списке результатов нужный объект, навигатор автоматически переключится на него.

В правой части окна выводится справочная информация о выбранном объекте:

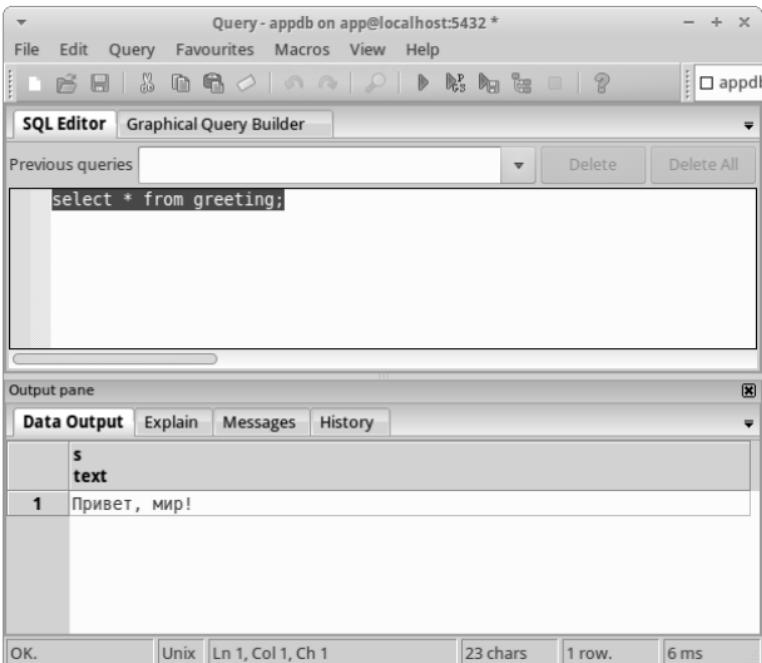
- Свойства — зависят от типа объекта (например, для столбца будет показан тип его данных);
- Статистика — эта информация используется системой для построения планов выполнения запросов и может рассматриваться администратором СУБД для анализа ситуации;
- Зависимости между объектами.

Также внизу правой части отображается команда SQL, с помощью которой можно создать данный объект.

## Выполнение запросов

Чтобы выполнить запрос, откройте окно SQL, выбрав в меню **Tools — Query tool (Инструменты — Инструмент запросов)**.

Введите запрос в верхней части окна на вкладке **SQL Editor** (Редактор SQL) и нажмите F5. Внизу окна на вкладке **Data Output** (Вывод данных) появится результат.



Вы можете вводить следующий запрос на новой строке, не стирая предыдущий; просто выделите нужный фрагмент кода перед тем, как нажимать F5. Таким образом история ваших действий всегда будет у вас перед глазами — обычно это удобнее, чем искать нужный запрос в списке **Previous queries** (Предыдущие запросы).

## Другое

С другими возможностями pgAdmin вы можете познакомиться на сайте продукта [www.pgadmin.org](http://www.pgadmin.org) или в справочной системе самой программы.

# Обучение и документация

## Документация

Для серьезной работы с PostgreSQL не обойтись без чтения документации. Это не только описание всех возможностей СУБД, но и исчерпывающее справочное руководство, которое всегда должно быть под рукой. Читая документацию, вы получаете емкую и точную информацию из первых рук — она написана самими разработчиками и всегда аккуратно поддерживается в актуальном состоянии.

Мы в Postgres Professional перевели на русский язык весь комплект документации — он доступен на нашем сайте: [www.postgrespro.ru/doc](http://www.postgrespro.ru/doc).



Предпочитающие оригинальную документацию на английском языке найдут ее по адресу [www.postgresql.org/docs](http://www.postgresql.org/docs).

## Учебные курсы

Помимо документации, мы занимаемся и разработкой учебных курсов для администраторов баз данных, разработчиков приложений и тех, кто хочет присоединиться к разработке самой СУБД:

- DBA1: Администрирование PostgreSQL. Базовый курс
- DBA2: Администрирование PostgreSQL. Расширенный курс
- DEV1: Базовый курс для разработчиков PostgreSQL
- DEV2: Расширенный курс для разработчиков PostgreSQL
- Hacking PostgreSQL

Деление курсов на базовые и расширенные вызвано большим объемом информации, который невозможно изложить и усвоить за несколько дней. Не стоит считать, что базовый курс предназначен только для новичков, а расширенный — только для опытных администраторов или разработчиков. Хотя между курсами есть пересечения по темам, но их не очень много.

Например, базовый трехдневный курс DBA1 знакомит с PostgreSQL и подробно объясняет базовые понятия администрирования, включая резервное копирование, а в пятидневный DBA2 охватывает подробности внутреннего устройства СУБД, репликацию, оптимизацию запросов и ряд других тем. Расширенный курс предполагает, что к темам базового курса возвращаться не нужно. Те же принципы лежат в основе деления на две части курса для разработчиков.

Каждый курс представляет собой связанный набор тем, последовательно раскрывающих его содержание (в отличие от документации, где необходимые сведения могут быть разбросаны по разным главам). Каждая тема состоит из теоретической части и практики. Теория — это не только презентация, но в большинстве случаев еще и демонстрация работы на «живой» системе. На практике слушателям предлагается выполнить ряд заданий для закрепления пройденного материала.

Темы поделены таким образом, чтобы теоретическая часть не превышала часа, так как большее время значительно усложняет восприятие материала. Практика, как правило, не превышает 30 минут.

В качестве материалов слушателям выдаются презентации с подробными комментариями к каждому слайду, результат работы демонстрационных скриптов и решения практических заданий.

Для некоммерческого использования материалы курсов доступны на нашем сайте всем желающим.

# **Курсы для администраторов**

## **DBA1. Администрирование PostgreSQL. Базовый курс**

Продолжительность:

3 дня

Предварительные знания:

Владение Unix.

Минимальные сведения о базах данных и SQL.

Какие навыки будут получены:

Общее представление о PostgreSQL и его архитектуре.

Установка базовая настройка, запуск СУБД.

Управление пользователями, данными, доступом.

Базовые задачи сопровождения и мониторинга.

Резервное копирование и восстановление.

Темы:

01. Введение в PostgreSQL
02. Архитектура PostgreSQL
03. Установка PostgreSQL
04. Использование psql
05. Базы данных
06. Табличные пространства
07. Системный каталог
08. Основные объекты БД
09. Пользователи и роли
10. Схемы
11. Привилегии
12. Конфигурирование сервера
13. Подключение и аутентификация
14. Мониторинг работы системы

15. Сопровождение PostgreSQL
16. Логическое резервирование
17. Физическое резервирование

Материалы учебного курса DBA1 (презентации, демонстрации, практические задания, видеозапись лекций) доступны для самостоятельного изучения по адресу [www.postgrespro.ru/education/courses/DBA1](http://www.postgrespro.ru/education/courses/DBA1).



## **DBA2. Администрирование PostgreSQL. Расширенный курс**

Продолжительность:

5 дней

Предварительные знания:

Владение Unix.

Базовые знания об архитектуре, установке, настройке, обслуживании СУБД.

Какие навыки будут получены:

Понимание архитектуры PostgreSQL.

Мониторинг и настройка базы, решение задач оптимизации производительности.

Выполнение задач сопровождения.

Резервирование, репликация, клонирование.

Темы:

Введение

01. Архитектура PostgreSQL

Изоляция и многоверсионность

02. Изоляция транзакций

03. Страницы и версии строк

04. Снимки и блокировки

05. Очистка
06. Автоочистка и заморозка

### Журналирование

07. Буферный кэш
08. Упреждающий журнал
09. Контрольная точка

### Репликация

10. Файловая репликация
11. Потоковая репликация
12. Переключение на реплику
13. Репликация: варианты

### Основы оптимизации

14. Обработка запроса
15. Методы доступа
16. Способы соединения
17. Статистика
18. Использование памяти
19. Оптимизация запросов

### Разные темы

20. Секционирование
21. Локализация
22. Обновление сервера
23. Управление расширениями
24. Внешние данные

Материалы учебного курса DBA2 (презентации, демонстрации, практические задания, видеозапись лекций) доступны для самостоятельного изучения по адресу:  
[www.postgrespro.ru/education/courses/DBA2](http://www.postgrespro.ru/education/courses/DBA2).



# Hacking PostgreSQL

Курс «Hacking PostgreSQL» собран из личного опыта разработчиков нашей компании, материалов конференций, статей и вдумчивого чтения документации и исходных кодов. В первую очередь он адресован начинающим разработчикам ядра PostgreSQL, но будет интересен и администраторам, которым иногда приходится обращаться к коду, и просто всем неравнодушным к архитектуре большой системы и желающим узнать, «как это работает на самом деле?».

Предварительные знания:

Знания основ языка SQL, функционала транзакций, индексов и т. п.

Знание языка C в объеме, достаточном как минимум для чтения исходных кодов (лучше иметь практические навыки).

Знакомство с базовыми структурами и алгоритмами.

Темы:

01. Обзор архитектуры
02. Сообщество PostgreSQL и инструменты разработчика
03. Расширяемость
04. Обзор исходного кода
05. Особенности кода
06. Системный каталог
07. Физическое представление данных
08. Работа с памятью
09. Разделяемая память и блокировки
10. Nodes & Trees
11. Патч ядра PostgreSQL
12. Отладка и тестирование
13. Транзакции. MVCC и VACUUM
14. WAL: и восстановление, и репликация

15. Индексы

16. Тенденции развития СУБД в целом и PostgreSQL  
в частности

Материалы курса Hacking PostgreSQL  
доступны для самостоятельного  
изучения по адресу  
[www.postgrespro.ru/education/courses/hacking](http://www.postgrespro.ru/education/courses/hacking).



## Курсы для разработчиков

Курсы для разработчиков находятся в процессе подготовки  
и должны появиться в самое ближайшее время.

## Где пройти обучение

Если вы хотите пройти обучение по перечисленным курсам  
в специализированном учебном центре, под руководством  
опытного преподавателя и с получением сертификата, то есть  
и такая возможность. Мы авторизовали несколько известных  
учебных центров, которые читают наши курсы. Их список можно  
посмотреть здесь: [www.postgrespro.ru/education](http://www.postgrespro.ru/education).



# Путеводитель по галактике

## Новости и обсуждения

Если вы собираетесь работать с PostgreSQL, вам захочется быть в курсе событий, узнавать о новых возможностях предстоящего выпуска, знакомиться с другими новостями. Много людей ведут свои блоги, публикуя интересные и полезные материалы. Удобный способ получить все англоязычные заметки в одном месте — читать сайт [planet.postgresql.org](http://planet.postgresql.org).

Не забывайте и про [wiki.postgresql.org](http://wiki.postgresql.org) — сборник статей, поддерживаемый и развивающий сообществом. Здесь вы найдете ответы на часто задаваемые вопросы, обучающие материалы, статьи про настройку и оптимизацию, про особенности миграции с разных СУБД и многое другое. Часть материалов этого сайта доступна и на русском языке: [wiki.postgresql.org/wiki/Russian](http://wiki.postgresql.org/wiki/Russian). Вы тоже можете помочь сообществу, переведя заинтересовавшую вас англоязычную статью.

Около двух тысяч русскоязычных пользователей PostgreSQL входят в группу «PostgreSQL в России» на фейсбуке ([www.facebook.com/groups/postgresql](https://www.facebook.com/groups/postgresql)).

Свой вопрос можно задать и на профильных сайтах. Например, на [stackoverflow.com](http://stackoverflow.com) на английском языке или [ru.stackoverflow.com](http://ru.stackoverflow.com) на русском (не забудьте поставить метку «postgresql»), или на форуме [www.sql.ru/forum/postgresql](http://www.sql.ru/forum/postgresql).

Новости нашей компании читайте по адресу [postgrespro.ru/blog](http://postgrespro.ru/blog).



# Списки рассылки

Если вы хотите узнавать обо всем первым, не дожидаясь, пока кто-нибудь напишет заметку в блоге, читайте списки рассылки. Разработчики PostgreSQL по старой традиции обсуждают между собой все вопросы исключительно по электронной почте. Для этого используется список рассылки `pgsql-hackers` (часто называемый просто «`hackers`»).

Полный перечень всех списков рассылки находится по адресу [www.postgresql.org/list](http://www.postgresql.org/list). Среди них — `pgsql-general` для обсуждения общих вопросов, `pgsql-bugs` для сообщений о найденных ошибках и многие другие. На любой список может подписаться каждый желающий, чтобы регулярно получать сообщения по электронной почте — каждое отдельно или в виде дайджеста.

Другой вариант — время от времени читать архив сообщений, который можно найти на [www.postgresql.org/list](http://www.postgresql.org/list), или, в несколько более удобном виде, на [postgresql.nabble.com](http://postgresql.nabble.com).

# Commitfest

Еще один способ быть в курсе событий, не тратя на это много времени — заглядывать на `commitfest.postgresql.org`. Через этот сайт проходят все патчи PostgreSQL.

Периодически открывается «окно», в котором разработчики регистрируют патчи для включения в определенную версию. Например, окно 01.03.2016—31.03.2016 относится к 9.6, а следующее за ним окно 01.09.2016—30.09.2016 — уже к следующей версии. Это делается для того, чтобы примерно за полгода до выхода новой версии PostgreSQL прекратить прием новых возможностей и успеть стабилизировать код.

Патчи проходят несколько этапов: рецензируются и исправляются по результатам рецензии, потом либо

принимаются, либо переносятся в следующее окно, либо — если совсем не повезло — отвергаются.

Таким образом вы можете быть в курсе возможностей, которые уже включены или предполагаются к включению в еще не вышедшую версию.

## Конференции

В России ежегодно проводятся две крупные международные конференции, собирающие сотни разработчиков и пользователей PostgreSQL:

- март — PGConf в Москве ([pgconf.ru](http://pgconf.ru))
- июль — PGDay в Санкт-Петербурге ([pgday.ru](http://pgday.ru))

Конференции по PostgreSQL проводятся и во всем мире, например:

- май — PGCon в Оттаве ([pgcon.org](http://pgcon.org))
- ноябрь — PGConf Europe проводится в разных городах, в 2016 году — в Таллине ([pgconf.eu](http://pgconf.eu))

Кроме того, в разных городах России проводятся конференции с более широкой тематикой, на которых представлено и направление баз данных и, в том числе, PostgreSQL.

К ним относятся:

- март — CodeFest в Новосибирске ([codefest.ru](http://codefest.ru))
- апрель — Dump в Екатеринбурге ([dump-conf.ru](http://dump-conf.ru))
- апрель — SECON в Пензе ([www.secon.ru](http://www.secon.ru))
- апрель — Стачка в Ульяновске ([nastachku.ru](http://nastachku.ru))
- декабрь — HappyDev в Омске ([happydev.ru](http://happydev.ru))

Помимо конференций проходят и менее официальные регулярные встречи; например, в Москве: [www.meetup.com/postgresqlrussia](http://www.meetup.com/postgresqlrussia).

# О компании

Компания Postgres Professional была основана в 2015 году и объединила всех ключевых российских разработчиков, вклад которых в развитие PostgreSQL существенен и признан мировым сообществом. Компания является российским вендором PostgreSQL и выполняет разработки на уровне ядра СУБД и расширений, оказывает услуги по проектированию и поддержке прикладных систем, миграции на PostgreSQL.

Компания уделяет большое внимание образовательной деятельности, организует крупнейшую международную конференцию PgConf.Russia в Москве и принимает участие в конференциях по всему миру.

Наш адрес:

117036, г. Москва, ул. Дмитрия Ульянова, д. 7А

Телефон:

+7 495 150-06-91

Сайт компании:

[postgrespro.ru](http://postgrespro.ru)

Электронная почта:

[info@postgrespro.ru](mailto:info@postgrespro.ru)



# **Услуги**

## **Промышленные решения на основе PostgreSQL**

- Проектирование и участие в создании критичных высоконагруженных систем с использованием СУБД PostgreSQL.
- Оптимизация конфигурации СУБД.
- Консультирование по вопросам использования СУБД в промышленных системах.
- Аудит систем заказчика по вопросам использования СУБД, проектирования баз данных, высокопроизводительных и отказоустойчивых архитектур.
- Внедрение СУБД PostgreSQL.

## **Вендорская техническая поддержка**

- Вторая и третья линии техподдержки СУБД PostgreSQL в режиме 24x7.
- Круглосуточная поддержка опытными администраторами: мониторинг, восстановление работоспособности, анализ непредвиденных обстоятельств, обеспечение производительности.
- Исправление ошибок, обнаруженных в СУБД и ее расширениях.

## **Миграция прикладных систем**

- Анализ имеющихся прикладных систем и определение сложности их миграции с других СУБД на PostgreSQL.

- Определение архитектуры нового решения и требований к доработкам прикладных систем.
- Миграция систем на СУБД PostgreSQL, в том числе действующих систем под нагрузкой.
- Поддержка прикладных разработчиков в процессе миграции.

## Разработка на уровне ядра и расширений

- Заказные разработки на уровне ядра СУБД и ее модулей расширения.
- Создание специальных модулей расширения для решения прикладных и системных задач заказчика.
- Создание кастомизированных версий СУБД в интересах заказчика.
- Публикация изменений в основную версию кода СУБД.

## Организация обучения

- Обучение администраторов баз данных работе с СУБД PostgreSQL.
- Обучение разработчиков и архитекторов прикладных систем особенностям СУБД PostgreSQL и эффективному использованию ее достоинств.
- Информирование о новой функциональности и важных изменениях в новых версиях.
- Проведение семинаров по разбору проектов заказчиков.

**Postgres Professional –  
российский вендор PostgreSQL.**

**Компания является членом  
международного сообщества  
PostgreSQL и приглашает  
разработчиков к участию  
в Open Source проектах.**

**В Postgres Professional работают  
ведущие российские разработчики  
PostgreSQL, чей вклад высоко  
оценивается международным  
сообществом.**

**Postgres Professional  
предоставляет услуги миграции,  
поддержки, разработки,  
консалтинга и обучения.**

**Компания работает с крупными  
российскими заказчиками  
различных отраслей.**

**«PostgreSQL для начинающих»**  
П.Лузанов, Е.Рогов, вёрстка А.Климковский © Postgres Professional 2016



[opensource@postgrespro.ru](mailto:opensource@postgrespro.ru)  
[www.postgrespro.ru](http://www.postgrespro.ru)  
+7 (495) 150-06-91