



Что такое PostgreSQL ?

PostgreSQL - это свободно распространяемая объектно-реляционная система управления базами данных (ORDBMS), наиболее развитая из открытых СУБД в мире и являющаяся реальной альтернативой коммерческим базам данных.

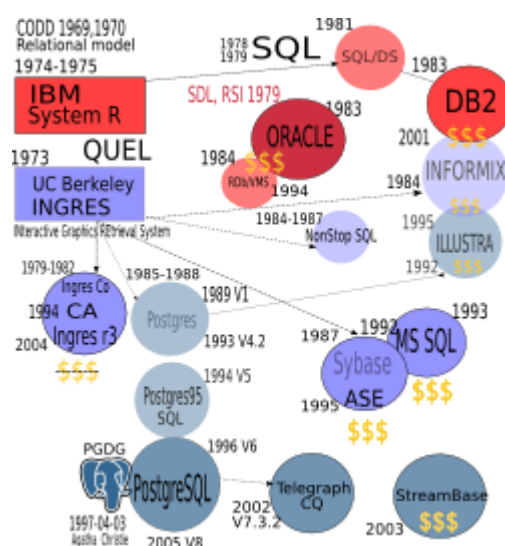
PostgreSQL произносится как **post-gress-Q-L** (можно скачать mp3 файл [postgresql.mp3](#)), в разговоре часто употребляется **postgres** (**пост-гресс**). Также, употребляется сокращение **pgsql** (**пэ-жэ-эс-ку-эль**).

Адрес этой статьи: http://www.sai.msu.su/~megera/postgres/talks/what_is_postgresql.html

История развития PostgreSQL

Краткую историю PostgreSQL можно прочитать в документации, распространяемой с дистрибутивом или на [сайте](#). Также, есть [перевод](#) на русский язык. Из нее следует, что современный проект PostgreSQL ведет происхождение из проекта POSTGRES, который разрабатывался под руководством Майкла Стоунбрейкера (Michael Stonebraker), профессора Калифорнийского университета в Беркли (UCB). Мне захотелось несколько подробнее показать взаимосвязи родословных баз данных, чтобы лучше понять место PostgreSQL среди основных игроков современного рынка баз данных.

Я попытался графически ([большая версия картинки](#) откроется в новом окне) отобразить все наиболее заметные RDBMS и связи между ними и приблизительно привел даты их создания и конца. Пересечение объектов означает поглощение, при этом поглощаемый объект более бледен и не окантован. Знак доллара означает, что база данных является коммерческой. При этом, я основывался на информации, доступной в интернете, в частности в [Wikipedia](#), в научных статьях, которые я читал и комментариях непосредственных пользователей БД, которые я получил после публикации этой картинки в интернете.



Надо сказать, что несмотря на то, что вся история реляционных баз данных насчитывает менее 4 десятков лет, многие факты из истории создания трактуются по-разному, даты не согласуются, а сами участники событий зачастую просто вольно трактуют прошлое. Здесь надо принимать во внимание тот факт, что базы данных - это большой бизнес, в котором развитие одних БД часто связано с концом других. Кроме того, БД в то время были предметом научных исследований, поэтому приоритетность работ является не последним аргументом при написании воспоминаний и интервью. Наверное, учитывая такую запутанность, премия ACM Software System Award #6 была присуждена одновременно двум соперничающим группам исследователей из IBM за работу над "System R" и Беркли - за INGRES, хотя Стоунбрейкер получил награду от ACM SIGMOD (сейчас это премия названа в честь Теда Кодда - автора реляционной теории баз данных) #1 в 1992 г., а Грей (James Gray, Microsoft) - #2 в 1993 году.

Итак, как следует из рисунка, видно две ветви развития баз данных - одна следует из "System R", которая разрабатывалась в IBM в начале 70-х, и другая из проекта "INGRES", которым руководил Стоунбрейкер приблизительно в тоже время. Эти два проекта начались как необходимость практического использования реляционной модели баз данных, разработанной Тедом Коддом (Ted Codd) из IBM в 1969, 1970 годах. Надо помнить, что в то время имелось две альтернативные модели баз данных - сетевая и иерархическая, причем за ними стояли мощные силы - CODASYL Data Base Task Group (сетевая) и сама IBM с ее базой IMS (Information Management System с иерархической моделью данных). Немного в стороне стоит "Oracle", взлет которой во многом связан с

коммерческим талантом Эллисона быть в нужном месте и в нужное время, как сказал Стоунбрейкер в своем [интервью](#), хотя она вместе с IBM сыграла большую роль в создании и продвижении SQL.

"System R" сыграла большую роль в развитии реляционных баз данных, создании языка SQL (изначально SEQUEL, но из-за проблем с уже существующей торговой маркой пришлось выкинуть все гласные буквы). Из "System R" развилась SQL/DS и DB2. На самом деле, в IBM было еще несколько проектов, но они были чисто внутренними. Подробнее об этой ветви можно прочитать в весьма поучительном документе ["The 1995 SQL Reunion: People, Projects, and Politics"](#), также доступен [русский перевод](#).

INGRES (или Ingres89), в отличие от "System R", вполне в духе Беркли развивалась как открытая база данных, коды которой распространялись на лентах практически бесплатно (оплачивались почтовые расходы и стоимость ленты). К 1980 году было распространено порядка 1000 копий. Название расшифровывается как **"I**nteractive **G**raphics **R**etrieval **S**ystem" и совершенно **случайно** связано с французским художником [Jean Auguste Dominique Ingres](#). Отличительной особенностью этой системы являлось то, что она разрабатывалась для операционной системы UNIX, которая работала на распространенных тогда PDP 11, что и предопределило ее популярность, в то время как "System R" работала только на больших и дорогих mainframe. Был разработан язык запросов QUEL, который, как писал Стоунбрейкер, похож на SEQUEL в том отношении, что программист свободен от знания о структуре данных и алгоритмах, что способствует значительной степени независимости от данных. Доступность INGRES и очень либеральная лицензия BSD, а также творческая деятельность, способствовали появлению большого количества реляционных баз данных, как показано на рисунке.

Стоунбрейкер лично способствовал их появлению, так он конце 70-х он организовал компанию Ingres Corporation (как он сам объясняет, ему пришлось на это пойти, так как Аризонский университет, потребовал поддержки), которая выпустила коммерческую версию Ingres, в 1994 году она была куплена CA (Computer Associates) и которая в 2004 году стала открытой как Ingres r3.

"NonStop SQL" компании Tandem Computers являлась модифицированной версией Ingres, которая эффективно работала на параллельных компьютерах и с распределенными данными. Она умела выполнять запросы параллельно и масштабировалась почти линейно с количеством процессоров. Ее авторами были выпускники из Беркли. Впоследствии, Tandem Computers была куплена компанией Compaq (2000 г.), а затем компанией HP.

Компания Sybase тоже была организована человеком из Беркли (Роберт Эпстейн) и на основе Ingres. Известно, что база данных компании Майкрософт "SQL Server" - это не что иное как база данных Sybase, которая была лицензирована для Windows NT. С 1993 года пути Sybase и Microsoft разошлись и уже в 1995 году Sybase переименовывает свою базу данных в ASE (Adaptive Server Enterprise), а Microsoft стала продолжать развивать MS SQL.

Informix тоже возник из Ingres, но на это раз людьми не из Беркли, хотя Стоунбрейкер все-таки поработал в ней CEO после того, как Informix купила в 1995 году компанию Ilustra, чтобы прибавить себе объектно-реляционности и расширяемости (DataBlade), которую организовал все тот же Майкл Стоунбрейкер как результат коммерциализации Postgres в 1992 году. В 2001 году она была куплена IBM, которая приобретала немалое количество пользователей Informix и технологию. Таким образом, DB2 также приобрела немного объектно-реляционности.

Проект Postgres возник как результат осмысления ошибок Ingres и желания преодолеть ограниченность типов данных, за счет возможности определения новых типов данных. Работа над проектом началась в 1985 и в период 1985-1988 было опубликовано несколько статей, описывающих модель данных, язык запросов POSTQUEL, и хранилище Postgres. POSTGRES иногда еще относят к так называемым **постреляционным СУБД**. Ограниченность реляционной модели всегда являлась предметом критики, хотя все понимали, что это является следствием ее простоты и ее заслугой. Однако, проникновение компьютерных технологий во все сферы жизни требовали новых приложений, а от баз данных - поддержки новых типов данных и возможностей, например, поддержка наследования, создание и управление сложными объектами.

Еще при проектировании оригинальной версии **POSTGRES** основное внимание было уделено расширяемости и объектно-ориентированным возможностям. Уже тогда было ясно необходимость расширения функциональности DMBS от управления данными (**data management**) в сторону управления объектами (**object management**) и знаниями (**knowledge management**). При этом объектная функциональность позволит эффективно хранить и манипулировать нетрадиционными типами данных, а управление знаниями позволяет хранить и обеспечивать выполнения коллекции правил (**rules**), которые несут семантику приложения. Стоунбрейкер так и определил основную задачу POSTGRES как **"обеспечить поддержку приложений, которые требуют службы управления данными, объектами и знаниями"**.

Одним из фундаментальным понятием POSTGRES является *class*. Class есть именованная коллекция экземпляров (*instances*) объектов. Каждый экземпляр имеет коллекцию именованных **атрибутов** и каждый атрибут имеет определенный **тип**. Классы могут быть трех типов - это основной класс, чьи экземпляры хранятся в базе данных, виртуальный (*view*), чьи экземпляры материализуются только при запросе (они поддерживаются системой управления правилами), и может быть версией другого (*parent*) класса.

Первая версия была выпущена в 1989 году, затем последовало еще несколько переписываний системы правил (*rule system*). Отметим, что коды Ingres и Postgres не имели ничего общего ! В POSTGRES была реализована поддержка таких типов как многомерные массивы, что уже шло в противоречие с реляционной моделью, timetravel - хранение версии объектов (впоследствии, в версии 6.3 этот тип был удален, так как его поддержка требовала больших усилий, а версия могла быть реализована на стороне приложения с помощью триггеров). В 1992 году была образована компания Illustra, а сам проект был закрыт в 1993 году выпуском версии 4.2. Однако, несмотря на официальное закрытие проекта, открытый код и BSD лицензия сподвигли выпускников Беркли Andrew Yu и Jolly Chen в 1994 году взяться за его дальнейшее развитие. В 1995 году они заменили язык запросов POSTQUEL на общепринятый SQL, проект получил название **Postgres95**, изменилась нумерация версий, был создан веб сайт проекта и появились много новых пользователей (среди которых был и автор).

К 1996 году стало ясно, что название "Postgres95" не выдержит испытанием временем и было выбрано новое имя - "**PostgreSQL**", которое отражает связь с оригинальным проектом POSTGRES и приобретением SQL. Также, вернули старую нумерацию версий, таким образом новая версия стартовала как 6.0. В 1997 был предложен слон в качестве логотипа, сохранилось письмо в архивах рассылки -hackers за 3 марта 1997 года и последующая дискуссия. Слон был предложен Дэвидом Янгом в честь романа Агаты Кристи "Elephants can remember" (Слоны могут вспоминать). До этого, логотипом был бегущий леопард (ягуар). Проект стал большой и управление на себя взяла небольшая вначале группа инициативных пользователей и разработчиков, которая и получила название PGDG (PostgreSQL Global Development Group). Дальнейшее развитие проекта полностью документировано в документации и отражено в архивах списка рассылки -hackers.

Что есть PostgreSQL сегодня ?

На сегодняшний день выпущена версия PostgreSQL v8 (19 января 2005 года), которая является значительным событием в мире баз данных, так как количество новых возможностей добавленных в этой версии, позволяет говорить о возникновении интереса крупного бизнеса как в использовании, так и его продвижении. Так, крупнейшая компания в мире, Fujitsu поддержала работы над версией 8, выпустила коммерческий модуль Extended Storage Management. Либеральная BSD-лицензия позволяет коммерческим компаниям выпускать свои версии PostgreSQL под своим именем и осуществлять коммерческую поддержку. Например, компания Pervasive объявила о выпуске Pervasive Postgres.

PostgreSQL поддерживается на всех современных Unix системах (34 платформы), включая наиболее распространенные, такие как Linux, FreeBSD, NetBSD, OpenBSD, SunOS, Solaris, DUX, а также под Mac OS X. Начиная с версии 8.X PostgreSQL работает в "native" режиме под MS Windows NT, Win2000, WinXP, Win2003. Известно, что есть успешные попытки работать с PostgreSQL под Novell Netware 6 и OS2.

PostgreSQL неоднократно признавалась базой года, например, Linux New Media AWARD 2004, 2003 Editors' Choice Awards, 2004 Editors' Choice Awards.

PostgreSQL используется как полигон для исследований нового типа баз данных, ориентированных на работу с **потоками данных** - это проект TelegraphCQ, стартовавший в 2002 году в Беркли после успешного проекта **Telegraph** (название главной улицы в Беркли). Интересно, что компания Streambase, которая была основана Майком Стоунбрейкером в 2003 году (изначально "Grassy Brook") для коммерческого продвижения этого нового поколения баз данных, никаким образом не ассоциируется с проектом Беркли.

Основные возможности и функциональность

Полный список всех возможностей предоставляемых PostgreSQL и подробное описание можно найти в объемной документации (1300 страниц).

- **Надежность** PostgreSQL является проверенным и доказанным фактом и обеспечивается следующими возможностями:

- полное соответствие принципам **ACID** - атомарность, непротиворечивость, изолированность, сохранность данных.
 - Atomicity - транзакция рассматривается как единая логическая единица, все ее изменения или сохраняются целиком, или полностью откатываются.
 - Consistency - транзакция переводит базу данных из одного непротиворечивого состояния (на момент старта транзакции) в другое непротиворечивое состояние (на момент завершения транзакции). Непротиворечивым считается состояние базы, когда выполняются все ограничения физической и логической целостности базы данных, при этом допускается нарушение ограничений целостности в течение транзакции, но на момент завершения все ограничения целостности, как физические, так и логические, должны быть соблюдены.
 - Isolation - изменения данных при конкурентных транзакциях изолированы друг от друга на основе системы версионности
 - Durability - PostgreSQL заботится о том, что результаты успешных транзакций **гарантировано** сохраняются на жесткий диск вне зависимости от сбоев аппаратуры.
- **многоверсионность** (Multiversion Concurrency Control, MVCC) используется для поддержания согласованности данных в конкурентных условиях, в то время как в традиционных базах данных используются блокировки. MVCC означает, что каждая транзакция видит копию данных (версию базы данных) на время начала транзакции, несмотря на то, что состояние базы могло уже измениться. Это защищает транзакцию от несогласованных изменений данных, которые могли быть вызваны (другой) конкурентной транзакцией, и обеспечивает изоляцию транзакций. Основным выигрышем от использования MVCC по сравнению с блокировкой заключается в том, что блокировка, которую ставит MVCC для чтения не конфликтует с блокировкой на запись, и поэтому чтение никогда не блокирует запись и наоборот. Конкурентные операции записи "мешают" друг другу только при работе с одной и той же записью.
- наличие **Write Ahead Logging** (WAL) - общепринятый механизм протоколирования всех транзакций, что позволяет восстановить систему после возможных сбоев. Основная идея WAL состоит в том, что все изменения должны записываться в файлы на диск только после того, как эти записи журнала, описывающие эти изменения будут и **гарантировано** записаны на диск. Это позволяет не сбрасывать страницы данных на диск после фиксации каждой транзакции, так как мы знаем и уверены, что сможем всегда восстановить базу данных используя журнал транзакций.
- **Point in Time Recovery** (PITR) - возможность восстановления базы данных (используя WAL) на любой момент в прошлом, что позволяет осуществлять непрерывное резервное копирование кластера PostgreSQL.
- **Репликация** также повышает надежность PostgreSQL. Существует несколько систем репликации, например, [Slony](#) (тестируется версия 1.1), который является свободным и самым используемым решением, поддерживает master-slaves репликацию. Ожидается, что Slony-II будет поддерживать multi-master режим.
- **Целостность данных** является сердцем PostgreSQL. Помимо MVCC, PostgreSQL поддерживает целостность данных на уровне схемы - это внешние ключи (foreign keys), ограничения (constraints).
- **Модель развития PostgreSQL**, которая абсолютно прозрачна для любого, так как все планы, проблемы и приоритеты открыто обсуждаются. Пользователи и разработчики находятся в постоянном диалоге через мэйлинг листы. Все предложения, патчи проходят тщательное тестирование до принятия их в программное дерево. Большое количество бета-тестеров способствует тестированию версии до релиза и вычищению мелких ошибок.
- **Открытость кодов** PostgreSQL означает их абсолютную доступность для любого, а либеральная BSD лицензия не накладывает никаких ограничений на использование кода.
- **Производительность** PostgreSQL основывается на использовании индексов, интеллектуальном планировщике запросов, тонкой системе блокировок, системе управления буферами памяти и кэширования, превосходной масштабируемости при конкурентной работе.
 - **Поддержка индексов**
 - Стандартные индексы - B-tree, hash, R-tree, GiST (обобщенное поисковое дерево)
 - Частичные индексы (**partial indices**) - можно создавать индекс по

ограниченному подмножеству значений, например,

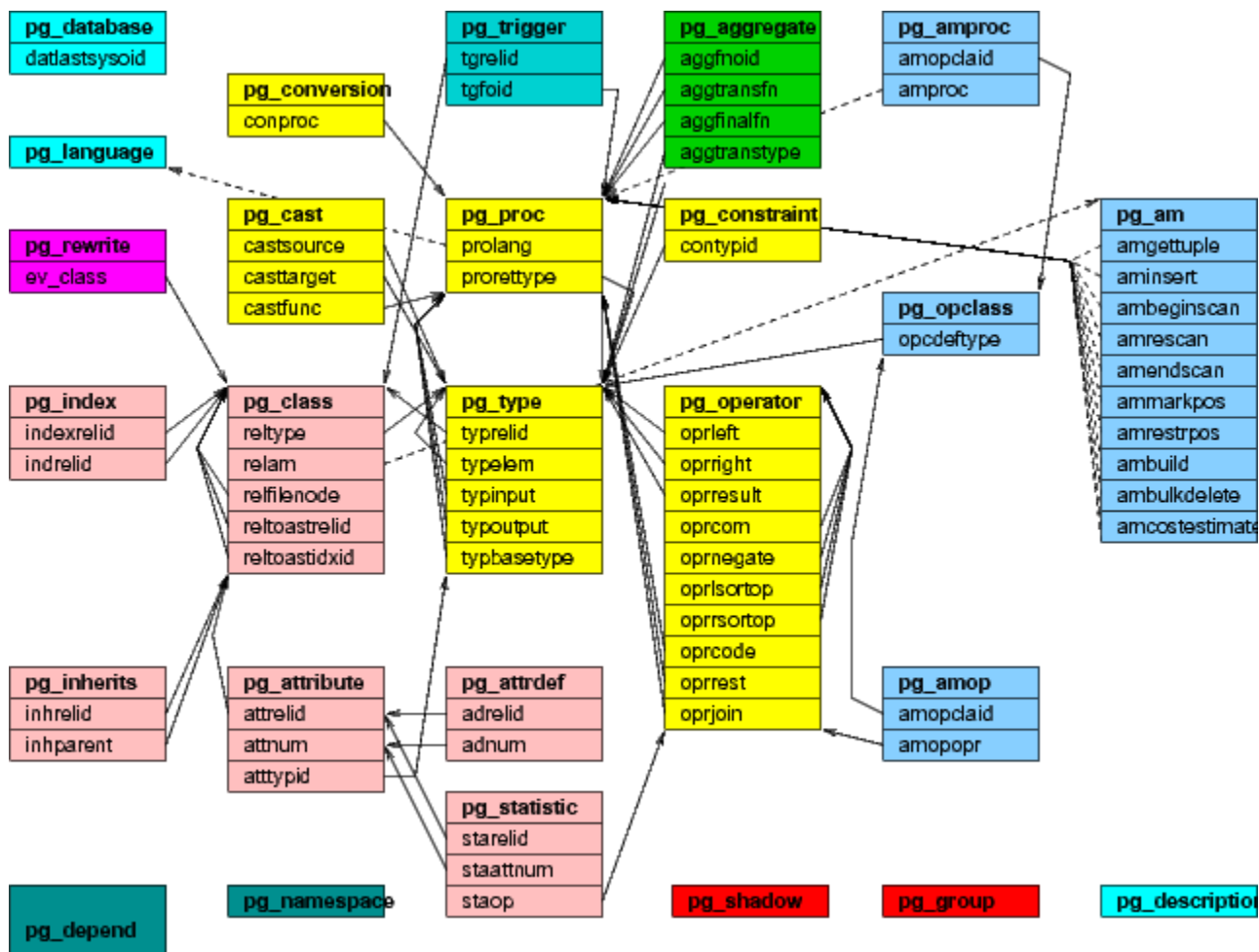
```
create index idx_partial on foo (x) where x > 0;
```

- **Функциональные индексы (**expressional indices**)** позволяют создавать индексы используя значения функции от параметра, например,

```
create index idx_functional on foo ( length(x) );
```

- **Планировщик запросов** основывается на стоимости различных планов, учитывая множество факторов. Он предоставляет возможность пользователю отлаживать запросы и настраивать систему.
- **Система блокировок** поддерживает блокировки на нижнем уровне, что позволяет сохранять высокий уровень конкурентности при защите целостности данных. Блокировка поддерживается на уровне таблиц и записей. На нижнем уровне, блокировка для общих ресурсов оптимизирована под конкретную ОС и архитектуру.
- **Управление буферами и кэширование** используют сложные алгоритмы для поддержания эффективности использования выделенных ресурсов памяти.
- **Tablespaces** (табличные пространства) для управления хранения данных на уровне **объектов**, таких как базы данных, схемы, таблицы и индексы. Это позволяет гибко использовать дисковое пространство и повышает надежность, производительность, а также способствует масштабируемости системы.
- **Масштабируемость** основывается на описанных выше возможностях. Низкая требовательность PostgreSQL к ресурсам и гибкая система блокировок обеспечивают его шкалирование, в то время как индексы и управление буферами обеспечивают хорошую управляемость системы даже при высоких нагрузках.
- **Расширяемость** PostgreSQL означает, что пользователь может настраивать систему путем определения новых функций, агрегатов, типов, языков, индексов и операторов. Объектно-ориентированность PostgreSQL позволяет перенести логику приложения на уровень базы данных, что сильно упрощает разработку клиентов, так как вся бизнес логика находится в базе данных. Функции в PostgreSQL однозначно определяются названием, количеством и типами аргументов.

На рисунке приведена ER диаграмма **системного каталога** PostgreSQL, в котором заложены все сведения об объектах системы, операторах и методах доступа к ним. При инициализации PostgreSQL кластера (команда `initdb`) создаются две базы данных - `template0` и `template1`, которые содержат предопределенный по умолчанию набор функциональностей. Любая другая база данных наследует `template1`, таким образом, часто используемые объекты и методы можно добавить в системный каталог `template1`.



PostgreSQL предоставляет командный интерфейс для работы с системным каталогом, с помощью которого можно не только получать информацию об объектах системы, но и создавать новые. Например, создавать базы данных с помощью **CREATE DATABASE**, новый домен - **CREATE DOMAIN**, оператор - **CREATE OPERATOR**, тип данных - **CREATE TYPE**.

Для создания нового типа данных и индексных методов доступа достаточно:

- Написать функции ввода/вывода и зарегистрировать их в системном каталоге с помощью **CREATE FUNCTION**
- Определить тип в системном каталоге с помощью **CREATE TYPE**
- Создать операторы для этого типа данных с помощью **CREATE OPERATOR**
- Написать функции сравнения и зарегистрировать их в системном каталоге с помощью **CREATE FUNCTION**
- Создать оператор по умолчанию, который будет использоваться для создания индекса по *primary key* - **CREATE OPERATOR CLASS**

Описанный сценарий использует существующих вид индекса. Для создания новых индексов надо использовать GiST.

Одной из примечательных особенностей PostgreSQL является обобщенное поисковое дерево или GiST ([домашняя страница проекта](#)), которое дает возможность специалистам в конкретной области знаний создавать специализированные типы данных и обеспечивает индексный доступ к ним не будучи экспертами в области баз данных. Аналогом GiST является технология DataBlade, которой сейчас владеет IBM (см. историческую справку выше). Идея GiST была придумана профессором Беркли Джозефом Хеллерстейном (Joseph M. Hellerstein) и опубликована в статье [Generalized Search Trees for Database Systems](#). Оригинальная версия GiST была разработана в Беркли как патч к POSTGRES и позднее была инкорпорирована в PostgreSQL. Позже, в 2001 году код был сильно модифицирован для поддержки ключей переменной длины, много-атрибутных индексов и безопасной работы с NULL, также были исправлено несколько ошибок. К настоящему времени написано довольно много интересных расширений на основе GiST, в том числе:

- модуль полнотекстового поиска [tsearch2](#)
- модуль для работы с иерархическими данными (tree-like) [ltree](#)

- модуль для работы с массивами целых чисел **intarray**

Дистрибутив PostgreSQL в поддиректории *contrib/* содержит большое количество (около 80) так называемых контриб-модулей, реализующих разнообразную дополнительную функциональность, такую как, полнотекстовый поиск, работа с xml, функции математической статистики, поиск с ошибками, криптографические модули и т.д. Также, есть утилиты, облегчающие миграцию с mysql, oracle, для административных работ.

- **Поддержка SQL**, кроме основных возможностей, присущих любой SQL базе данных, PostgreSQL поддерживает:
 - Очень высокий уровень соответствия ANSI SQL 92, ANSI SQL 99 и ANSI SQL 2003. Подробнее можно прочитать в [документации](#).
 - **Схемы**, которые обеспечивают пространство имен на уровне SQL. Схемы содержат таблицы, в них можно определять типы данных, функции и операторы. Используя полное имя объекта можно одновременно работать с несколькими схемами. Схемы позволяют организовать базы данных совокупность нескольких логических частей, каждая из которых имеет свою политику доступа, типы данных. Для приложений, которые создают новые объекты в базе данных удобно и безопасно создавать отдельную схему (и включать ее в SEARCH_PATH) с тем, чтобы избежать возможной коллизии с именами объектов и удобством обновления приложения.
 - **Subqueries** - подзапросы (subselects), полная поддержка SQL92. Подзапросы делают язык SQL более гибким и зачастую более эффективным.
 - **Outer Joins** - внешние связи (LEFT, RIGHT, FULL)
 - **Rules** - правила, согласно которым модифицируется исходный запрос. Главное отличие от триггеров состоит в том, что rule работает на уровне запроса и **перед** исполнением запроса, а триггер - это реакция системы на изменение данных, т.е. триггер запускается **в процессе** исполнения запроса для каждой измененной записи (PER ROW). Правила используются для указания системе, какие действия надо произвести при попытке обновления *view*.
 - **Views** - представления, виртуальные таблицы. Реальных экземпляров этих таблиц не существуют, они материализуются только при запросе. Одним из основных предназначений 'view' является разделение прав доступа к родительским таблицам и к 'view', а также обеспечение постоянства пользовательского интерфейса при изменении родительских таблиц. Обновление 'view' (материализация) [ВОЗМОЖНО](#) в PostgreSQL с помощью pl/pgsql.
 - **Cursors** - курсоры, позволяют уменьшить трафик между клиентом и сервером, а также память на клиенте, если требуется получить не весь результат запроса, а только его часть.
 - **Table Inheritance** - наследование таблиц, позволяющее создавать объекты, которые наследуют структуру родительского объекта и добавлять свои специфические атрибуты. При этом наследуются значения атрибутов по умолчанию (**DEFAULTS**) и ограничение целостности (**CONSTRAINTS**). Поиск по родительской таблице автоматически включает поиск по дочерним объектам, при этом сохраняется возможность поиска только по ней (**only**). Наследование можно использовать для работы с очень большими таблицами для эмуляции **partitioning**.
 - Prepared Statements (преподготовленные запросы) - это объекты, живущие на стороне сервера, которые представляют собой оригинальный запрос после команды **PREPARE**, который уже прошел стадии разбора запроса (parser), модификации запроса (rewriting rules) и создания плана выполнения запроса (planner), в результате чего, можно использовать команду **EXECUTE**, которая уже не требует прохождения этих стадий. Для сложных запросов это может быть большим выигрышем.
 - **Stored Procedures** - серверные (хранимые) процедуры позволяют реализовывать бизнес логику приложения на стороне сервера. Кроме того, они позволяют сильно уменьшить трафик между клиентом и сервером.
 - **Savepoints**(nested transactions) - в отличие от "плоских транзакций", которые не имеют промежуточных точек фиксации, использование **savepoints** позволяет отменять работу части транзакции, например вследствие ошибочно введенной команды, без влияния на оставшуюся часть транзакции. Это бывает очень полезно для транзакций, которые работают с большими объемами данных.
 - Права доступа к объектам системы на основе системы привилегий. Владелец объекта или суперюзер может как разрешать доступ (**GRANT**), так и отменять (**REVOKE**).
 - Система обмена сообщениями между процессами - **LISTEN** и **NOTIFY** позволяют

организовывать событийную модель взаимодействия между клиентом и сервером (клиенту передается название события, назначенное командой *notify* и PID процесса).

- **Триггеры** позволяют управлять реакцией системы на изменение данных (INSERT, UPDATE, DELETE), как перед самой операцией (**BEFORE**), так и после (**AFTER**). Во время выполнения триггера доступны специальные переменные NEW (запись, которая будет вставлена или обновлена) и OLD (запись перед обновлением).
- **Cluster table** - упорядочивание записей таблицы на диске согласно индексу, что иногда за счет уменьшения доступа к диску ускоряет выполнение запроса.
- **Богатый набор типов данных** PostgreSQL включает:
 - Символьные типы (**character(n)**) как определено в стандарте SQL и тип **text** с практически неограниченной длиной.
 - **Numeric** тип поддерживает произвольную точность, очень востребованную в научных и финансовых приложениях.
 - **Массивы** согласно стандарту SQL:2003
 - Большие объекты (**Large Objects**) позволяют хранить в базе данных бинарные данные размером до 2Gb
 - **Геометрические типы** (point, line, circle, polygon, box,...) позволяют работать с пространственными данными на плоскости.
 - ГИС (**GIS**) типы в PostgreSQL являются доказательством расширяемости PostgreSQL и позволяют эффективно работать с трехмерными данными. Подробности можно найти на сайте проекта [PostGis](#).
 - Сетевые типы (**Network types**) поддерживают типы данных **inet** для IPV4, IPV6, а также **cidr** (Classless Internet Domain Routing) блоки и **macaddr**
 - Композитные типы (**composite types**) объединяют один или несколько элементарных типов и позволяют пользователям манипулировать со сложными объектами.
 - Временные типы (timestamp, interval, date, time) реализованы с очень большой точностью
 - Псевдотипы **serial** и **bigserial** позволяют организовать упорядоченную последовательность целых чисел (AUTO_INCREMENT у некоторых СУБД).
- PostgreSQL имеет очень богатый набор **встроенных функций** и **операторов** для работы с данными, полный список которых можно посмотреть в [документации](#).
- Поддержка 25 различных наборов символов (**charsets**), включая ASCII, LATIN, WIN, KOI8 и UNICODE, а также поддержка **locale**, что позволяет корректно работать с данными на разных языках.
- Поддержка **NLS**(Native Language Support) - документация, сообщения об ошибках доступны на различных языках, включая японский, немецкий, итальянский, французский, русский, испанский, португальский, словенский, словацкий и несколько диалектов китайского языков.
- **Интерфейсы** в PostgreSQL реализованы для доступа к базе данных из ряда языков (C, C++, C#, python, perl, ruby, php, Lisp и другие) и методов доступа к данным (JDBC, ODBC).
- **Процедурные языки** позволяют пользователям разрабатывать свои функции на стороне сервера, тем самым переносить логику приложения на сторону базы данных, используя языки программирования, отличные от встроенных SQL и C. К настоящему времени поддерживаются (включены в стандартный дистрибутив) PL/pgSQL, pl/Tcl, PL/Perl и pl/Python. Кроме них, существует поддержка PHP, Java, Ruby, R, shell.
- **Простота использования** всегда являлась важным фактором для разработчиков.

Утилита **psql** (входит в дистрибутив) предоставляет удобный интерфейс для работы с базой данных, содержит краткий справочник по SQL, облегчает ввод команд (используя стрелки для повтора и табулятор для расширения), поддерживает историю и буфер запросов, а также позволяет работать как в интерактивном режиме, так и потоковом режиме.

[phpPgAdmin](#) (лицензия GPL) представляет возможность с помощью веб браузера администрировать PostgreSQL кластер.

[pgAdmin III](#) (GNU Artistic license) предоставляет удобный интерфейс для работы с базами данных PostgreSQL и работает под Linux, FreeBSD и Windows 2000/XP.

PgEdit - программная среда для разработки приложений и SQL-редактор, доступна для Windows и Mac.

- **Безопасность данных** также является важнейшим аспектом любой СУБД. В PostgreSQL она обеспечивается 4-мя уровнями безопасности:
 - PostgreSQL нельзя запустить под привилегированным пользователем - системный контекст
 - SSL,SSH шифрование трафика между клиентом и сервером - сетевой контекст
 - сложная система аутентификации на уровне хоста или IP адреса/подсети. Система аутентификации поддерживает пароли, шифрованные пароли, Kerberos, IDENT и прочие системы, которые могут подключаться используя механизм подключаемых аутентификационных модулей.
 - Детализированная система прав доступа ко всем объектам базы данных, которая совместно со схемой, обеспечивающая изоляцию названий объектов для каждого пользователя, PostgreSQL предоставляет богатую и гибкую инфраструктуру.

Некоторые пределы PostgreSQL

Название	Значение
Максимальный размер БД	Unlimited
Максимальный размер таблицы	32 TB
Максимальная длина записи	1.6 TB
Максимальный длина атрибута	1 GB
Максимальное количество записей в таблице	Unlimited
Максимальное количество атрибутов в таблице	250 - 1600 в зависимости от типа атрибута
Максимальное количество индексов на таблицу	Unlimited

Сводная таблица основных реляционных баз данных

За основу взяты данные из [Wikipedia](https://en.wikipedia.org/wiki/List_of_database_management_systems)

Название	ASE	DB2	FireBird	InterBase	MS SQL	MySQL	Oracle	PostgreSQL
Лицензия	\$\$\$	\$\$\$	IPL ²	\$\$\$	\$\$\$	GPL/\$\$\$	\$\$\$	BSD
ACID	Yes	Yes	Yes	Yes	Yes	Depends ¹	Yes	Yes
Referential integrity	Yes	Yes	Yes	Yes	Yes	Depends ¹	Yes	Yes
Transaction	Yes	Yes	Yes	Yes	Yes	Depends ¹	Yes	Yes
Unicode	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Schema	Yes	Yes	Yes	Yes	No ⁵	No	Yes	Yes
Temporary table	No	Yes	No	Yes	Yes	Yes	Yes	Yes
View	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes
Materialized view	No	Yes	No	No	No	No	Yes	No ³
Expression index	No	No	No	No	No	No	Yes	Yes
Partial index	No	No	No	No	No	No	Yes	Yes
Inverted index	No	No	No	No	No	Yes	Yes	No
Bitmap index	No	Yes	No	No	No	No	Yes	No
Domain	No	No	Yes	Yes	No	No	Yes	Yes
Cursor	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes
User Defined Functions	Yes	Yes	Yes	Yes	Yes	No ⁴	Yes	Yes
Trigger	Yes	Yes	Yes	Yes	Yes	No ⁴	Yes	Yes
Stored procedure	Yes	Yes	Yes	Yes	Yes	No ⁴	Yes	Yes

Tablespace	Yes	Yes	No	?	No ⁵	No ¹	Yes	Yes
Название	ASE	DB2	FireBird	InterBase	MS SQL	MySQL	Oracle	PostgreSQL

Замечания:

- **1** - для поддержки транзакций и ссылочной целостности требуется InnoDB (не является типом таблицы по умолчанию)
- **2** - Interbase Public License
- **3** - Materialized view (обновляемые представления) могут быть эмулированы на PL/pgSQL
- **4** - только в MySQL 5.0, которая является экспериментальной версией
- **5** - только в MS SQL Server 2005 (Yukon)

Что ожидается в будущих версиях

Полный список новых возможностей приведен в большом списке TODO, который уже много лет поддерживает Брюс Момжан (Bruce Momjian), однако приоритеты для версии 8.1 еще не определены, более того, пока не определена продолжительность цикла разработки. Пока можно достаточно уверенно утверждать, что в 8.1 версии, помимо исправлений ошибок и улучшения существующей функциональности или приведение синтаксиса к стандарту SQL, будут:

- **bitmap** индексы
- интегрирование **autovacuum** в серверный процесс
- Two phase commit JDBC driver
- поддержка IN,OUT,INOUT параметров для pl/pgsql (**CVS**)
- увеличение предела максимального количества аргументов у функции (100 по умолчанию) (**CVS**)
- Оптимизация MIN,MAX за счет использования индексов (**CVS**)

Также, недавно проходило обсуждение о возможных планах о поддержке **table partitioning**, что сильно увеличивает производительность базы данных при работе с большими таблицами.

Новость:

Вышла версия 8.0.2, в которой, помимо исправления ошибок и изменения версии библиотеки **libpq** (ВНИМАНИЕ ! Все клиентские приложения, которые используют libpq, требуется пересобрать, например DBD::Pg), алгоритм кэширования страниц "ARC", которым владеет IBM, был заменен на другой, "патентно-чистый" алгоритм "2Q".

Поскольку история с заменой алгоритма "ARC" в PostgreSQL вызвала большой интерес и обсуждение в сети (а она связана с очень "горячей" темой выдачи и использования патентов на программное обеспечение), я остановлюсь подробнее на описании механизма кэширования (buffer management) в PostgreSQL. Я использовал архив обсуждений, оригинальные работы и статью Элейн Мустэйн (A. Elein Mustain) The Saga of the ARC Algorithm and Patent.

Управление буферами в PostgreSQL

Кэширование страниц, или сохранение прочитанных с диска страниц в памяти, очень важно для эффективной работы любой СУБД, так как времена доступа к диску и памяти отличаются на многие порядки. В идеале, мы хотим, чтобы все страницы, к которым происходит обращение, попадали в память, с тем, чтобы последующее ее использование не требовало обращения к диску. Однако, так как количество доступной памяти ограничено, то возникает ситуация, когда требуется принимать решение, какую страницу надо освободить (заместить) для того, чтобы поместить в кэш новую страницу. Практически все коммерческие системы используют ту или иную вариацию **LRU** (Least Recently Used) алгоритма, в котором высвобождается та страница, к которой дольше всего не обращались. В чистом виде этот алгоритм не очень хорош для использования в СУБД в силу большой разнообразности последовательности запросов, например, не учитывает частоту обращения к странице, не защищен от "cache flooding", когда всего одно единичное последовательное чтение большого количества страниц (sequential scan) может заполнить кэш страницами, к которым может не быть больше обращения, т.е., к полной потере эффективности кэширования. Иногда, используют термин "scan-resistant", когда говорят, что хороший алгоритм должен быть устойчив по отношению к "cache flooding".

PostgreSQL использовал разновидность этого алгоритма, известную как **LRU/K**, реализованную Томом Лайном (Tom Lane). В этом алгоритме используется история **K-последних** обращений к странице (именно **последних**, что позволяет этому алгоритму адаптироваться к изменениям шаблона запросов, в отличие от **LFU** алгоритма), которая позволяет отличить популярные страницы от давно не используемых. Для этого строится упорядоченная очередь (priority queue) указателей на страницы в кэше на основе времени обращения к странице по правилу: если у страницы P1 K-тое обращение (**предпоследнее**, для наиболее важного случая K=2, LRU/2) является более свежим чем у P2, то P1 будет замещено после P2. Классический LRU алгоритм можно рассматривать как **LRU/1**, так как он использовал информацию только об одном (последнем) обращении к странице. Важным является не то, что произошло единичное обращение к странице, а то, насколько эта страница была популярна в течение некоторого времени. Однако, этот алгоритм требовал нетривиальной настройки и время на построение очереди растет логарифмически в зависимости от размера буфера.

ARC (Adaptive Replacement Cache) алгоритм был привлекателен тем, что он учитывал не только как **часто** страница была использована, но и насколько **недавно** это происходило и не сильно "нагружал" процессор, как это происходило с LRU/K алгоритмом. Он динамически поддерживает баланс между часто используемыми и недавно используемыми страницами. Этот алгоритм был реализован Яном Виеком (Jan Wieck) для версии 7.5 (впоследствии 8.0), который впоследствии был несколько улучшен после **статьи**, описывающей **CAR** (Clock with Adaptive Replacement) алгоритм. Однако, незадолго (за два дня) до выхода PostgreSQL 8.0 было обнаружено (см. **постинг** Нейла Конвея (Neil Conway) и последующее обсуждение), что IBM подала **заявку на алгоритм ARC** еще в 2002 году. Так как было уже поздно что-либо менять было решено выпустить 8.0 версию как есть, а потом заняться решением проблемы. Несмотря на то, что IBM еще не получила патент на ARC алгоритм и то, что IBM имеет хорошую практику **поддержки OSS проектов**, и можно было надеяться на получения официального разрешения на его использование в PostgreSQL, как предлагали многие, было решено исследовать вопрос о действительном нарушении патента и выяснить возможность замены ARC алгоритма на "патентно-чистый" алгоритм.

Основным аргументом в пользу замены алгоритма было желание сохранить PostgreSQL доступным для **"любого использования"** согласно BSD лицензии, которая позволяет коммерческое использование PostgreSQL без каких-либо лицензионных отчислений. В начале февраля 2005 года Том Лэйн предложил измененную версию ARC алгоритма, близкую к **2Q** и опубликованную в 1994 году задолго до ARC, и которая решала проблему "cache flooding" ("scan resistant") и не требовала больших изменений в коде (в основном удаление кода), которая и была реализована в версии 8.0.2. 2Q алгоритм (Two Queue) почти также эффективен как LRU/K, но проще, не требует настройки и быстрее. Он добивается этого тем, что хранит в основном буфере только "горячие" страницы, а не занимается очищением "холодных" страниц в основном буфере как LRU/2. Упрощенно алгоритм выглядит так: при первом обращении указатель на страницу помещается в очередь A1 (FIFO), и если во время второго обращения страница еще находилась в A1, то страница называется горячей и помещается в основной буфер, который уже контролируется как LRU очередь. Если к странице не обращались пока она была в A1, то страница, вероятно, "холодная" и 2Q алгоритм удаляет ее из буфера.

PGDG - PostgreSQL Global Development Group

PostgreSQL развивается силами международной группы разработчиков (PGDG), в которую входят как непосредственно программисты, так и те, кто отвечают за продвижение PostgreSQL (Public Relation), за поддержание серверов и сервисов, написание и перевод документации, всего на 2005 год насчитывается около 200 человек. Другими словами, PGDG - это сложившийся коллектив, который полностью самодостаточен и устойчив. Проект развивается по общепринятой среди открытых проектов схеме, когда приоритеты определяются реальными нуждами и возможностями. При этом, практикуется публичное обсуждение всех вопросов в списке рассылке, что практически исключает возможность неправильных и несогласованных решений.

Это относится и к тем предложениям, которые уже имеют или рассчитывают на финансовую поддержку коммерческих компаний.

Цикл разработки

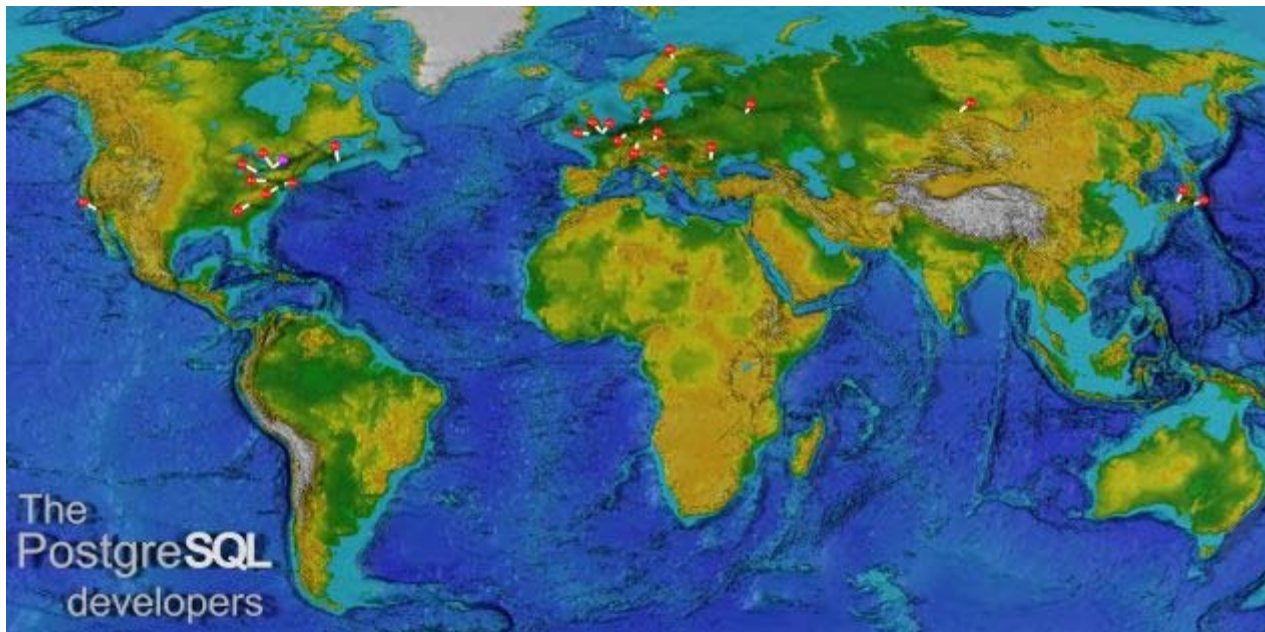
Цикл работой над новой версией обычно длится 10-12 месяцев (сейчас ведется дискуссия о более коротком цикле 2-3 месяца) и состоит из нескольких этапов (упрощенная версия):

- **Обсуждение** предложений в списке **-hackers**. На собственном опыте могу заверить, что это

очень непростой процесс и плохо подготовленный proposal не пройдет. Учитываются много факторов - алгоритмы, структуры данных, совместимость с существующей архитектурой, совместимость с SQL и так далее.

- После принятия решения о работе над новой версией в CVS открывается новая ветка и с этого момента все изменения, касающиеся новых возможностей, вносятся туда. Также, анализируются патчи, которые присылаются в список [-patches](#). Все изменения протоколируются и доступны любому для рассмотрения ([anonymous CVS](#), [-committers](#) лист рассылки или через [веб-интерфейс](#) к CVS). Иногда, в процессе работы над новой версией вскрываются или исправляются старые ошибки, в этом случае, наиболее критические исправляются и в предыдущих версиях (**backporting**). По мере накопления таких исправлений принимается решение о выпуске новой стабильной версии, которая совместима со старой и не требует обновления хранилища. Например, 7.4.7 - является **bugfix**-ом стабильной версии 7.4.
- В некоторый момент объявляется этап **code freeze**(замораживания кода), после которого в CVS не допускается новая функциональность, а только исправление или улучшение кода. Граница между новой функциональностью и улучшением кода не описана и иногда возникают разногласия на этот счет, к документации и расширениям (**contribution modules** в поддиректории contrib/) обычно относятся более либерально. Замечу, что все это время все CVS версия проходит **непрерывное тестирование** на большом количестве машин, под разными архитектурами, операционными системами и компиляторами. Все это стало возможно благодаря проекту [pgbuildfarm](#), который является распределенной системой тестирования, объединяющая добровольцев, которые предоставляют свои машины для тестирования. Проверяется не только корректность сборки, но и, благодаря обширному набору тестов (**regression test**), и правильность работы. Текущий статус (всех поддерживаемых версий, не только CVS) можно посмотреть на [этой](#) странице. Исходные тексты CVS версии PostgreSQL проходят тестирование в [OSDL](#) что помогает в обнаружении систематических изменений производительности (в обе стороны), иногда такие обнаружения приводят к необходимости "размораживания кода". Начиная с версии 8 такие тестирования будут регулярно проводиться в OSDL STP и PLM (STP - Scalable Test Platform и PLM - Patch Lifecycle Manager).
- После внутреннего тестирования "собирается" дистрибутив и объявляется выход **бета** версии, на тестирование и исправление ошибок отводится 1-3 месяца. Бета версия не рекомендуется для использования в продакшн проектах (**production**), но практика показала хорошее качество таких версий и многие начинают ее использовать ради апробирования новой функциональности. Как правило, окончательная версия совместима с бета-версией и не требует обновления хранилища. По мере исправления замеченных ошибок выпускаются новые бета-версии.
- После исправления **всех** замеченных ошибок, выпускается **релиз-кандидат**, который уже практически ничем не отличается от окончательной версии, разве что не хватает документации и списка изменений.
- В течении месяца выходит окончательная версия, которая анонсируется на главном [веб-сайте](#) проекта и его зеркалах, мэйлинг листах. Также, PR группа, которая к этому моменту подготовила анонсы на разных языках, распространяет их по всем ведущим сайтам и СМИ. Они принимают участие в конференциях, семинарах и прочих общественных мероприятиях.

На карте обозначены точки, где живут и работают члены PGDG, оригинальная версия с большей функциональностью находится на официальном [сайте разработчиков](#).



Структура

- Управляющий комитет (6 человек).
Принимает решение о планах развития и выпусках новых версий.
- Заслуженные разработчики (2 человека).
Бывшие члены управляющего комитета, которые отошли от участия в проекте.
- Основные разработчики (23).
- Разработчики (22)

Кроме PGDG, значительное участие в развитии PostgreSQL принимает некоммерческая организация **"The PostgreSQL Foundation"**, созданная для продвижения и поддержки PostgreSQL. Сайт фонда находится по адресу www.thepostgresqlfoundation.org.

Спонсорская помощь на развитие PostgreSQL поступает как от частных лиц, так и от коммерческих компаний, которые:

- принимают на работу членов PGDG
- оплачивают разработку каких-либо новых возможностей
- предоставляют услуги в виде хостинга или оплаты трафика
- поддерживают публичные мероприятия PGDG
- выделяют своих программистов на участие в разработке

Кроме того, некоторые разработки поддерживаются государственными фондами, например, Российский Фонд Фундаментальных Исследований.

Где используется

Если изначально POSTGRES использовался в основном в академических проектах для исследования алгоритмов баз данных, в университетах как отличная база для обучения, то сейчас PostgreSQL применяется практически повсеместно. Например, зоны .org, .info полностью обслуживаются PostgreSQL, известны многотерабайтные хранилища астрономических данных, Lycos, BASF. Из российских проектов, использующих PostgreSQL, наиболее известными является портал [Рамблер](http://rambler.ru), в разработке которого я принимал участие в 2000-2002 годах, федеральные порталы Минобразования.

Сообщество

Сообщество PostgreSQL состоит из большого количества пользователей, объединенных разными

интересами, такими как участие в разработке, поиск советов, решений, возможность коммерческого использования.

Поддержка

- Основной источник актуальной информации о PostgreSQL является его **официальный сайт** www.postgresql.org, который имеет зеркала по всему миру. На нем публикуются сведения о всех событиях (анонсы релизов, семинаров, конференций), поддерживается список ресурсов, относящихся к PostgreSQL.
- Основная поддержка осуществляется через **почтовую рассылку**, архивы которой доступны через Web по адресам:
 - archives.postgresql.org
Архив pgsql-ru-general - русскоязычного списка рассылки, [как подписаться](#).
 - www.pgsql.ru/db/mw

Как показала многолетняя практика, списки рассылок являются наиболее эффективным и очень полезным источником знаний, обмена мнениями и помощи в самых различных ситуациях. На март 2005 года зарегистрировано 32812 пользователей, которые когда-либо писали в мэйлинг лист.

Небольшая статистика списков рассылок PostgreSQL по данным www.pgsql.ru на 1 апреля 2005 года.

Первая 20-ка мэйлинг листов по количеству постингов		Распределение постингов по годам	
name	count	#	Year
HACKERS	107696	19355	2005
GENERAL	93272	68403	2004
SQL	27574	71884	2003
COMMITTERS	21384	61604	2002
ADMIN	20397	58072	2001
PATCHES	17354	38793	2000
NOVICE	13772	25258	1999
BUGS	13700	16779	1998
MISC	13545	15315	1997
INTERFACES	13029	612	1996
JDBC	12705	7	1995
QUESTIONS	7865		
ADVOCACY	6676		
CYGWIN	6166		
WWW	5636		
PERFORMANCE	5359		
ODBC	5182		
PORTS	4769		
DOCS	3991		
PHP	3106		

- **Поисковая система PGsearch** (разработана при поддержке [РФФИ](#) и [Дельта-Софт](#)) предоставляет поиск по сайтам сообщества. На момент написания этой статьи проиндексировано 480000 страниц из 67 сайтов, индекс обновляется еженедельно.
- Много **полезной информации** по PostgreSQL можно найти на сайтах
 - techdocs.postgresql.org
 - [Varlena](#)
 - [Powerpostgresql](#)
 - [PGnotes](#)
- **Документация на русском** (переводы и оригинальные статьи) доступны на сайте русскоязычного сообщества <http://www.linuxshare.ru/postgresql/>.
- Ответы на ваши вопросы можно найти в "**PostgreSQL FAQ**" (часто задаваемые вопросы):
 - [Оригинальная версия](#)
 - на [русском языке](#)
- Дистрибутивы PostgreSQL доступны для **скачивания** с основного ftp-сервера проекта и его зеркал. Подробная информация доступна со страницы <http://www.postgresql.org/download/>. Кроме того, многие дистрибутивы Linux распространяются с бинарной версией PostgreSQL и обеспечивают поддержку обновлений. Для **ознакомления** с PostgreSQL можно скачать образ загрузочного CD (Live CD) - [bittorrent формат](#) или в [ISO формате](#). Информацию о том,

как установить PostgreSQL под Mac OS X можно найти [здесь](#). Инсталлятор для Win32 можно скачать с сайта проекта [Pginstaller](#).

- **Коммерческая поддержка** осуществляется рядом компаний, список которых доступен по адресу www.postgresql.org/support/. Также на российском сайте ведется [список российских компаний](#), которые заявили о поддержке PostgreSQL.

Разработка

Для проектов, имеющих отношение к PostgreSQL, предоставляется возможность размещать их на специальных сайтах, поддерживаемые PGDG и предоставляющие практически все, необходимые для разработчиков, сервисы:

- gborg.postgresql.org
- pgfoundry.org

Заключение

PostgreSQL является полнофункциональной объектно-реляционной СУБД, готовой для практического использования. Ее функциональность и надежность обусловлены богатой историей развития, профессионализмом разработчиков и технологией тестирования, а ее перспективы заложены в ее расширяемости и свободной лицензии.

Благодарности

Автор благодарит русскоязычное сообщество за критику и дополнения, Российский Фонд Фундаментальных Исследований ([РФФИ](#)) за поддержку гранта 05-07-90225-в.

Текст написан [Олегом Бартуновым](#) в 2005 году, поправки и комментарии приветствуются.

Краткая справка:



Олег Бартунов (основная специальность астроном, работает в ГАИШ МГУ) является членом PGDG (основной разработчик) с 1996 года, был в числе основателей компании "GreatBridge", является членом ["The PostgreSQL Foundation"](#). Помимо использования PostgreSQL в проектах (самые известные - это портал [Рамблер](#), [Научная Сеть](#), [Астронет](#)), он в 1996 году добавил поддержку **locale** в PostgreSQL, затем совместно с Федором Сигаевым (компания [Delta-Soft](#)) занимался поддержкой и разработкой [GiST в PostgreSQL](#), на основе которого были разработаны такие популярные модули как полнотекстовый поиск, работа с массивами, поиск с ошибками, поддержка иерархических данных. Соавтор свободного полнотекстового поиска для PostgreSQL [OpenFTS](#). Является автором и создателем (совместно с Федором Сигаевым) сайта [pgsql.ru](#). Занимается продвижением PostgreSQL для использования в астрономии, в частности, для работы с очень большими астрономическими каталогами, проект [pgSphere](#) - хранение данных со сферическими координатами и индексные методы доступа к ним.