Специализированные типы данных для цифровых библиотек♣

© О.С. Бартунов © Ф.Г. Сигаев
Государственный Астрономический Институт им. П.К.Штернберга
oleg@sai.msu.su teodor@sigaev.ru

Аннотация

При построении сложных информационных систем зачастую приходится решать разнообразные технологические задачи, связанные с хранением, доступом и поиском информации. Учитывая современные требования к производительности, надежности и шкалированию таких систем, такие задачи требуют использования достаточно сложных алгоритмов и специализированных структур данных. В статье рассматривается необходимость расширяемости базы данных и описываются несколько типов данных, разработанные авторами для свободно-распространяемой базы данных PostgreSQL, облегчающие работу со множествами, иерархическими данными, слабоструктурированными данными и полнотекстовым поиском.

1 Введение

Информатизация общества, развитие Web, не только увеличила количество информации, но и расширила круг задач перед разработчиками информационных систем, которые по большому счету предоставляют доступ к хранилищу документов, метаданных, обычно реализованных в виде баз данных. Например, появились мультимедийные базы данных для хранения изображений, звуковых файлов. Помимо стандартных запросов они позволяют, например, искать похожие объекты. Ведь неспроста в последнее время большое внимание стали уделять потоковым, темпоральным базам данных [1]. Появление языка XML [2] для описания слабо-структурированных данных и его повсеместное применение привело к появлению XML баз данных.

Кроме этого, развитие "онлайн" служб привело к появлению ограничения на скорость выполнения запроса в пределах нескольких секунд. Помимо качества предоставляемых услуг очень важна скорость их предоставления. Современный пользователь привык к таким стандартам и не будет тратить время на ожидание запроса и просто воспользуется конкурентной службой. Более того, развитие семантического веба предполагает наличие стандартизованного программного доступа к базам данных специальных программ-агентов, которые разыскивают необходимую информацию и здесь временные рамки становятся еще более актуальными. С другой стороны, количество единиц информации сильно возросло и продолжает увеличиваться, причем здесь наблюдаются две тенденции - оцифровывание старых материалов и появление новых. Сейчас нередки службы, которые предоставляют доступ ко многим миллионам единиц информации, астрономические базы данных в рабочем режиме оперируют с миллиардами объектов. Когда говорят о размерах базы данных следует различать разные случаи - не очень большое количество очень больших объектов, очень большое количество не очень больших объектов и их комбинация - очень большое количество очень больших объектов. Разные случаи требуют своего подхода к организации хранилища, типам данных и методам работы с ними. Более того, массовые сервисы требует большой степени конкурентности, т.е., быстрого выполнения многих запросов одновременно, что накладывает дополнительные требования на базу данных. Очевидно, что производительность работы базы данных зависит от многих факторов, мы будем рассматривать только те, которые определяют эффективность нижнего слоя базы данных - это типы данных, алгоритмы и эффективный доступ к ним.

Еще одна причина, почему требуются новые типы, связана с парадигмой разработки приложений с использованием "тонкого" клиента, когда почти вся логика и функциональность реализована в СУБД, а клиентом только является стандартное приложение, наподобие веб браузера, которое принимает запрос от пользователя и показывает результаты. Для облегчение работы с типами данных, например, тестирование на их валидность, преобразование к стандартному виду, операции, разрабатывается новый тип, который берет на себя всю работу. Примером может служить поддержка международных стандартов на серийные номера такие как EAN13, UPC, ISBN (книги), ISMN (музыка), ISSN, реализованная в PostgreSQL как расширение contrib/isn.

Ограниченность классических реляционных баз данных привела к появлению объектно-ориентированных баз данных, которые были призваны для поддержки возникающих новых

Труды 9ой Всероссийской научной конференции «Электронные библиотеки: перспективные методы и технологии, электронные коллекции» - RCDL'2007, Переславль-Залесский, Россия, 2007.

потребностей. Тем не менее, они не получили такого широкого распространения среди разработчиков приложений баз данных, см. [3], в то время как реляционные базы данных приобрели возможность расширения стандартных типов данных, при сохранении проверенных временем надежных и эффективных алгоритмов работы [22]. Необходимость поддержки новых типов данных помимо строк и чисел в реляционных базах данных в глобальном аспекте не вызывает ни у кого сомнения. Этим занимаются специальные комитеты по стандартизации, которые разрабатывают стандарты SQL. Хорошим примером является появление в стандарте SQL (SQL99, SQL2003) нового типа данных - массивов, или появление поддержки XML.

Для многих проектов, нацеленных на долгое использование, массовый сервис и большое количество информации, требование расширяемости базы данных сейчас является важнейшим при выборе какую базу данных использовать.

1.1 Что такое расширяемость баз данных

Под расширяемостью понимается не только сама возможность добавления новых типов данных, новых запросов, но и возможность делать это специалистами в определенной области знания, а не разработчиками ядра базы данных. При этом, эти типы данных должны обеспечивать такой же уровень надежности, конкурентности, предоставляемых СУБД для стандартных, встроенных типов данных, что обычно требует очень квалифицированных программистов, знакомых с ядром СУБД, а также, тщательного и продолжительного тестирования. Немаловажным фактором для практического использования является поддержка индексных методов доступа (access methods, AM) к пользовательским типам данных. Одним из способом расширения, удовлетворяющий вышеперечисленным требованиям, является наличие шаблонной (template) структуры в ядре СУБД, для которой реализованы все основные операции с данными - вставка, модификация и удаления, и индексная поддержка. Интегрированность с ядром СУБД означает, что новые типы данных, реализованные с помощью такого шаблона, унаследуют все основные свойства СУБД, такие как поддержка транзакций, конкурентность и восстановление. Одной из СУБД, в которой реализованы все эти возможности, и которая оказала большое влияние на развитие всех других СУБД, является свободно-распространяемая СУБД PostgreSQL [4, 5].

Рассматриваемые расширения стандартных типов данных являются результатом нашей многолетней работы (при частичной поддержке РФФИ) над расширяемостью PostgreSQL и

реального использования при разработке больших информационных систем. Все модули входят в состав дистрибутива PostgreSQL и широко используются в различных приложениях.

Статья организована следующим образом: сначала приводится общая информация о модулях расширения PostgreSQL и как их использовать, затем мы кратко приведем описание типов данных и примеры использования. В приложении описывается расширяемость PostgreSQL и приводятся результаты сравнения двух видов индексов, используемых в расширениях.

1.2 Модули расширения PostgreSQL

Расширения PostgreSQL - это программы, написанные с использование API, загружаемые в пользовательскую базу данных. Так как описание PostgreSQL содержится в самой базе данных (в системном каталоге), то добавление новых типов данных может производиться без ее остановки. Официальные расширения содержатся в дистрибутиве PostgreSQL и находятся в директории contrib. Существуют несколько возможных причин почему эти расширения являясь официальными не входят в ядро PostgreSQL - добавленная функциональность не так широко востребована, недостаточное тестирование, нестабильный внешний интерфейс, недостаток возможностей для включения в ядро PostgreSQL, заявленная функциональность уже существует в ядре и ее развитие планируется в будущем, расширение использует внешние библиотеки с лицензией отличной от BSD. Тем не менее, большинство официальных расширений проверено годами использования во многих проектах и часто рекомендуется для решения тех или иных сложных проблем. Это одна из специфик развития свободного программного обеспечения. Типичная процедура использования расширения состоит их нескольких шагов (требуются права суперюзера), на примере модуля ltree:

1. Войти в директорию расширения
   cd contrib/ltree
2. установка расширения
   make && make install && make installcheck
3. загрузка расширения в конкретную базу данных

```
psql DBNAME < /path/to/pgsql/share/contrib/ltree.sql
```

После этого в базе данных DBNAME можно использовать все типы данных, операторы и функции, реализованные в расширении.

2 Модули расширения

Многие наши расширения ориентированы на работу со множествами, например целочисленные массивы, полнотекстовый поиск.

Для ускорения работы с этими типами данных используются индексная поддержка. Множество представляется тем или иным способом битовой строкой (сигнатурой) фиксированной длины, с которой удобно и быстро работать. Подробно о сигнатурах смотри статью "Bloom Filters" в Википедии [6].Сигнатуры хранятся в индексной структуре RD-Tree [7], реализованная с помощью

механизма разработки расширений GiST [8], которая требует от разработчика расширения создания семи интерфейсных функций. Так как GiST поддерживает конкурентность и восстанавливаемость после сбоя, то все расширения автоматически их наследуют. Построенный подобным образом индекс, не может не быть точным и может найти некоторое количество ошибочных множеств, так называемых ложных попаданий (false hits). Поэтому требуется дополнительная проверка с использованием оригинального значения, что требует доступа к таблице, а следовательно, дополнительных операций с диском и может существенно замедлять поиск. PostgreSQL предоставляет автоматическую поддержку таких неточных (lossy) индексов. Вероятность таких ложных попаданий зависит от размера сигнатуры, количества уникальных элементов во множестве и количества множеств (записей в таблице). Такое ограничение не является существенным для небольших баз данных, но существенно ограничивает применение в больших. Дальнейшая работа по улучшению индексной поддержки работы со множествами привела к разработке нового индекса на базе GIN [9,10] - обобщенного обратного индекса, который являясь точным, имеет высокую скорость поиска, очень хорошо шкалируется и слабо зависит от размера множества. Обратный индекс имеет свою проблему - он плохо обновляется, что связано с его организацией. Несмотря на то, что мы используем специальную оптимизацию по ускорению обновления, он сильно проигрывает GiST в скорости обновления. Таким образом, GIN индекс оптимально использовать для очень больших и слабо меняющихся хранилищ, в то время как GiST оптимален для не очень больших, но часто меняющихся данных. PostgreSQL предоставляет возможность совместного использования лучших свойств этих индексов. Сравнение GIN и GiST на примере полнотекстового поиска приведено в [20]. В приложении сравнивается скорость обновления GIN И GiST индексов на примере целочисленных массивов.

2.1 Поддержка целочисленных массивов - contrib/intarray

Этот модуль часто используется в тех случаях, когда требуется денормализовать БД для повышения производительности. Например, типичная задача построения списка документов из нескольких разделов рубрикатора. Классическая нормализованная схема предусматривает использование трех таблиц - документов, рубрик и связей между документами и рубриками. Заметим, что документ может принадлежать нескольким рубрикам. Построение такого списка требует соединения таблиц, что влияет на производительность и в некоторых случаях просто неприемлемо. Денормализация приводит к тому, что в таблицу документов добавляется атрибут, который является массивом целых чисел - идентификаторов рубрик, к которым принадлежит данный документ. Однако, поиск будет все равно медленным из-за того, что операция "поиск в массиве" не использует индекс. Наш модуль обеспечивает индексную поддержку для операций над целочисленными массивами - overlap, contains, contained и ряд дополнительных операторов и функций.
Пример типичного запроса - найти все документы из секций с идентификаторами
2,15,23

```
select * from documents where sections @> '{2,15,23}';
```

Здесь @> - обозначение оператора contains, предоставленный расширением. Начиная с версии 8.2 можно использовать GIN индекс для ускорения операций над массивами overlap, contains, contained.

2.2 Поддержка иерархических данных - contrib/ltree

Добавляет тип, поддерживающий хранение древообразных данных и их индексную поддержку с помощью GiST и B-Tree. Такие данные типичны для работы с каталогами, рубрикаторами. Стандартный способ работы с иерархическими данными, например, с каталогами, заключается в использовании таблиц связей (parent_id,child_id), что приводит, так или иначе, к необходимости использования рекурсивных запросов, см., например, классическую книгу Joe Celko "Trees and Hierarchies in SQL for Smarties". Идея модуля ltree состоит в том, чтобы хранить иерархию связей в специальном типе данных ltree, например, 'Top.Countries.Europe.Russia', и предоставлять индексную поддержку для основных операций - поиск всех родителей, поиск всех детей и много других полезных операторов и функций [12]. Удобство использования этого модуля и большое количество полезных функций делает ltree очень полезным для решения типичных задач цифровых библиотек. Поддерживаются также операции над массивами ltree.
Примеры запросов:
a) Найти всех подрубрики 'Top.Science':

```
ltreetest=# select path from test where path <@ 'Top.Science';
path
-----------------------------------
Top.Science
Top.Science.Astronomy
Top.Science.Astronomy.Astrophysics
Top.Science.Astronomy.Cosmology
```

b) Найти все рубрики с рубрикой 'Astronomy':

```
ltreetest=# select path from test where path ~ '*.Astronomy.*';
                path
-----------------------------------------
Top.Science.Astronomy
Top.Science.Astronomy.Astrophysics
Top.Science.Astronomy.Cosmology
Top.Collections.Pictures.Astronomy
Top.Collections.Pictures.Astronomy.Stars
Top.Collections.Pictures.Astronomy.Gala..
Top.Collections.Pictures.Astronomy.Astr..
```

2.3 btree_gist - contrib/btree_gist

Вспомогательный модуль для создания композитных индексов, поддерживает практические все основные типы данных используемые в PostgreSQL. Типичным примером использования является создание индекса по полнотекстовому полю и времени. Такой индекс можно использовать не только для обычного полнотекстового поиска, но и для его ускорения поиска в определенном временном интервале.

2.4 Хранилище для слабо-структурированных данных - contrib/hstore

На пользовательском уровне новый тип очень похож на хеш (ключ => значение), причем синтаксис операторов и функций подобен интерфейсу к хешам в языке Perl. Модуль hstore очень полезен для хранения атрибутов, по которым не производится поиска, но которые нужны для показа. Например, требуется хранить несколько разнородные объекты, которые имеют только несколько общих атрибутов, что неминуемо приводит к большому количеству атрибутов, большое количество их которых имеет значение NULL. При этом, поиск производится только по некоторым из них. Модуль позволяет "упаковать" неиспользуемые атрибуты в один и обеспечивает доступ в случае показа.
Например, для хранения статей и книг требуется разные атрибуты, но поиск требуется организовать по всем документам. В частном случае, можно иметь две таблицы для статей и документов и проводить два отдельных поиска, объединяя результаты на уровне приложения. А что будет, если потребуется хранить короткие новости, рефераты и другие виды документов. Все они имеют общие параметры, по которым ведется поиск, например, автор(ы), название, год и время издания, краткий абстракт, но в тоже время они могут свои уникальные атрибуты. На практике часто встречается простой способ решения проблемы - все виды документов собираются в одну большую таблицу с очень большим количеством полей. Однако, этот способ неэффективен и нерасширяем. Хорошим решением могло быть использование XML, однако его поддержка в PostgreSQL еще находится на начальном уровне и не предоставляет индексной поддержки. Модуль hstore позволяет решить эту проблему, сохранив в себе все неиспользуемые для поиска атрибуты. Пример возможной структуры таблицы для описанного примера:

```
CREATE TABLE documents (
  id integer PRIMARY KEY,
  author text,
  title text,
  pubdate timestamp,
  misc hstore
);
```

Пример - получить ISBN номер, который хранится в поле misc:

```
select misc->'isbn' as isbn from documents;
     isbn
------------------
'5-229-000000-0'
```

2.5 Полнотекстовый поиск - contrib/tsearch2

Этот модуль предназначен для организации полнотекстового поиска в БД и является де-факто стандартным поисковым механизмом PostgreSQL. Его отличительной особенностью является online-индекс и полная интеграция с БД, что дает возможность проводить полнотекстовый поиск с ограничениями по атрибутам. Например, искать по документам, в зависимости от уровня доступа пользователя и дате публикации. Tsearch2 [13] поддерживает использование словарей, предоставляет API для их создания. Поддержка словарей популярных форматов ispell (для приведения слов к их нормальной форме) и стемминга на основе Портеровского алгоритма [14] позволяет использовать tsearch2 со многими языками. Гибкость настройки tsearch2, конфигурация которого хранится в базе данных и доступна с помощью стандартных команд SQL, позволяет разрабатывать различные поиски ориентированные на разные задачи. Модуль предоставляет два вида ранжирующих функций, использующие координатную информацию и разные веса слов, которые можно использовать для сортировки результатов поиска по их релевантности запросу.
Характерные свойства:

- Подключаемые словари.
- Полная поддержка UTF-8.
- Поддержка специализированных словарей - тезаурус, синонимы.
- Подключаемые парсеры текста.
- Выбор индексной поддержки через GiST или GIN. Индексная поддержка через GiST позволяет строить online индексы с малым временем вставки, в то время как использование GIN дает возможность индексировать гораздо большее количество документов.
- Настраиваемая конфигурация взаимодействия парсера и словарей
- Несколько функций ранжирования результатов поиска
- Генерация текстовых фрагментов и "подсветка" найденных слов

Отметим, что полнотекстовый поиск в PostgreSQL полностью интегрирован с ядром БД, т.е. поддерживает транзакции, конкурентность и восстановление после сбоев. В предстоящей версии PostgreSQL (8.3) полнотекстовый поиск будет уже встроенным и доступен без установки дополнительных модулей [15].

2.6 Нечеткий поиск - contrib/pg_trgm

Часто бывает необходимо предложить пользователю возможность исправить запрос, например, из-за опечатки. Модуль pg_trgm позволяет быстро найти подходящие слова-кандидаты из заранее проиндексированного словаря, которые разбиваются

на триграммы и индексируются с помощью GIN и GiST. Слово из запроса также разбивается на триграммы и среди всех слов в словаре находятся слова-кандидаты, которые имеют наибольшее количество совпадающих триграмм. Чаще всего этот модуль используется в сочетании с полнотекстовым поиском.
Пример: Найти ближайшее похожее слово:

```
=# select word, similarity(word, 'supyrnova') AS sml
from apod_words where word % 'supyrnova' order by
sml desc, word;
   word  | sml
-----------+----------
supernova  | 0.538462
```

3 Заключение

Вопросы выбора технологической платформы является весьма важным при планировании проекта и не последним фактором является возможность расширяемости хранилища, способного удовлетворить не только количественные факторы, такие как способность работать с возрастающим количеством данных, но и добавление новых типов запросов, требующих новых типов данных, при сохранении производительности и надежности работы.
Родившись как академический проект в Беркли (идеи PostgreSQL оказали большое влияние на почти все базы данных), PostgreSQL в настоящее время развивается независимой международной группой разработчиков и распространяется под лицензией BSD. Недавнее тестирование SPEC [16] показало, что PostgreSQL является реальной альтернативой коммерческим базам данных, в то время как богатый набор возможностей позволяет реализовать любые сложные приложения.
Опыт использования свободно-распространяемой СУБД PostgreSQL многих проектах, уникальные свойства расширяемости, признанной надежности позволяет нам рекомендовать ее для использования в проектах, разного типа и сложности. Многолетнее использование описанных типов доказало их функциональность и надежность. В частности, они используются в таких крупных российских проектах как портал Рамблер, "Мой Круг", "Работа для Вас", федеральные порталы министерства образования, Астронет, "Все о Геологии" [17].

4. Приложение

4.1 Расширяемость в PostgreSQL

Несмотря на большие ожидания от объектно-ориентированных баз данных, они так и не завоевали широкого распространения среди разработчиков приложений баз данных [3], в то время как реляционные базы данных тем или иным способом научились предоставлять такие возможности. Так, наиболее продвинутая среди свободно-распространяемых баз данных PostgreSQL, еще на этапе проектирования рассматривалась как расширение стандартной реляционной модели от управления данными в сторону управления объектами и знаниями [18].
Первоначальный подход [19] к расширяемости СУБД в PostgreSQL, основанный на использовании хорошо изученных структур, таких как B+-tree и R-tree, и абстрагирования процедур доступа и обновления записей, позволял добавлять поддержку новых типов данных, но был ограничен теми запросами и методами доступа, которые предоставляли эти базовые структуры.

4.2 GiST (Generalized Search Tree)

Для того, чтобы преодолеть это ограничение, Хеллерстейн и др. [8] предложили структуру индекса, называемую GiST (Generalized Search Tree, Обобщенное поисковое дерево), которое является обобщенной разновидностью R-tree и предоставляет стандартные методы навигации по дереву и его обновления (разделение и удаление узлов). Было отмечено, что очень многие AM можно представить как иерархию предикатов, в которой каждый предикат выполняется для всех ключей, содержащихся в подузлах этой иерархии. Таким образом, такая структура данных может служить шаблоном для реализации многих AM, не накладывая существенных ограничений. Например, в B+-tree записи во внутренних узлах представляют диапазоны, которые задают ограничения на ключи в концевых узлах соответствующего поддерева. GiST предоставляет индексный доступ к новым типам данным и поддерживает расширяемый набор запросов.
Современная реализация GiST в PostgreSQL поддерживает ключи переменной длины, композитные (multi-key) ключи, конкурентность и восстановление после краха (concurrency and recovery), что позволяет разрабатывать расширения экспертам в области данных, не будучи экспертами-разработчиками ядра СУБД. Практически все наши расширения имеют индексную поддержку с помощью GiST. Программный интерфейс GiST и практические примеры рассмотрены в [21].

4.3 GIN (Generalized Inverted Index)

В дополнение к GiST, нами был разработан обобщенный обратный индекс (GIN) - хранилище для множества пар (ключ, постинг лист), где постинг лист - это упорядоченный массив идентификаторов записей, в которых встречается ключ. Например, для полнотекстового поиска ключ - это слово, а постинг лист содержит идентификаторы документов, в которых встречается это слово. Другой пример - это массивы, где ключом является элемент массива, а сам массив - постинг лист. При этом, постинг лист состоит из ссылок на массивы, документы, записи. GIN поддерживает типичные операции со множествами - contains, overlap и contained.
Обобщенность означает то, что GIN не знает какую именно операцию он ускоряет и все определяется

интерфейсными функциями, определенные для конкретного типа. В этом смысле GIN схож с GiST и отличается от B-tree индексов, которые используют фиксированный набор операций сравнения. При добавлении документа, который состоит из N уникальных слов, требуется N обновлений обратного индекса, поэтому обратный индекс лучше всего использовать для статических данных. Реализация GIN состоит из B-tree, построенного по ключам, что дает быстрый поиск, и B-tree для хранения длинных постинг листов для оптимизации скорости обновления индекса. GIN имеет хорошую производительность на больших объемах, которая слабо зависит от количества ключей.

4.4 Сравнение производительности GiST и GIN на примере целочисленных массивов

Для тестирования скорости обновления индексов использовались искусственно сгенеренные данные со следующими параметрами - максимальное количество элементов в массиве (MAXLEN), количество записей в таблице (MAXNUM) и количество уникальных элементов MAXVAL. Измерялось время создания индекса, времена обновления для 10,100,1000,1000 записей.
Тесты проводились на машине со следующими параметрами: Linux 2.6.19.1, Slackware, x86, Intel(R) Pentium(R) 4 CPU 2.40GHz, 2Gb RAM.
PostgreSQL версия 8.2.4 был сконфигурен со следующими параметрами:

shared_buffers = 256MB
work_mem = 8MB
maintenance_work_mem = 64MB
effective_cache_size = 256MB

Все времена приводятся в милисекундах (ms).
Ниже приводятся данные для следующих параметров (см.полное описание и результаты тестов [11]): MAXLEN=500, MAXNUM=100000, MAXVAL=100000;

Скорость создания индекса:
GIN - 146765.140
GiST - 12721.421

Скорость обновления индекса:

#Updates	GIN	GiST
10	160.569	72.390
100	543.285	83.026
1000	3702.067	284.039
10000	32571.510	2969.296

Скорость поиска:
Использовался запрос:
explain analyze select * from tt where a @> '{2}';

GiST:

Bitmap Heap Scan on tt (cost=29.27..405.87 rows=100 width=979) (actual time=68
.714..6311.757 rows=678 loops=1)
Filter: (a @> '{2}'::integer[])
-> Bitmap Index Scan on gist_idx (cost=0.00..29.24 rows=100 width=0) (actual time=43.430..43.430 rows=10774 loops=1)
Index Cond: (a @> '{2}'::integer[])
Total runtime: 6312.807 ms

GIN:

Bitmap Heap Scan on tt (cost=150.50..527.11 rows=100 width=982) (actual time=0
.610..11.277 rows=678 loops=1)
Recheck Cond: (a @> '{2}'::integer[])
-> Bitmap Index Scan on gin_idx (cost=0.00..150.48 rows=100 width=0) (actual time=0.433..0.433 rows=678 loops=1)
Index Cond: (a @> '{2}'::integer[])
Total runtime: 11.892 ms

Выводы:
1. GiST индекс существенно быстрее обновляется чем GIN, более чем в 10 раз при больших обновлениях
2. GIN индекс вполне обновляем при небольшом количестве обновлений
3. Скорость поиска GIN индекса во много раз превышает GiST индекс. Это объясняется тем, что GiST индекс неточный, особенно для такого большого количества уникальных элементов, и требуется проверка оригинального документа. Протокол команды explain analyze показывает, что GiST индекс нашел 10774 записей, в то время как правильное количество записей - 678 и потребовалось проверить все 10774 записей, чтобы отсеять ложные. GIN индекс нашел правильное количество записей.

Литература

[1] С.Д. Кузнецов, "Предвестники новых манифестов управления данными", http://citforum.ru/database/articles/premanifest/

[2] Extensible Markup Language (XML) 1.1. W3C Recommendation, REC-xml11-20060816./W3C, 2006. http://www.w3.org/TR/2006/REC-xml11-20060816/

[3] С.Д. Кузнецов, "Объектно-реляционные базы данных: прошедший этап или недооцененные возможности?",http://www.citforum.ru/database/articles/ordbms10/4.shtml

[4] СУБД PostgreSQL. http://www.postgresql.org

[5] О.С.Бартунов, "Что такое PostgreSQL?" ,2005. http://www.sai.msu.su/~megera/postgres/talks/what_is_postgresql.html

[6] Bloom Filter. http://en.wikipedia.org/wiki/Bloom_filter

[7] J.M. Hellerstein and A. Pfeifer, "The RD-tree: an Index Structure for Sets", Technical Report No. 1252, 1994

[8] Joseph M. Hellerstein, Jeffrey F. Naughton and Avi Pfeffer, "Generalized Search Trees for Database Systems", 1996, Proceedings of the 21th International Conference on Very Large Data Bases. http://db.cs.berkeley.edu/papers/vldb95-gist.pdf

[9] GIN Readme. http://www.sai.msu.su/~megera/wiki/Gin

[10] GIN Presentation. http://www.sigaev.ru/gin/Gin.pdf

[11] Тестирование скорости обновления GIN и GiST индексов. http://www.sai.msu.su/~megera/wiki/GinTestUpdate

[12] Ltree. http://www.sai.msu.su/~megera/postgres/gist/ltree/

[13] Tsearch2. http://www.sai.msu.su/~megera/postgres/gist/tsearch/V2

[14] Snowball stemmer. http://snowball.tartarus.org

[15] О.С.Бартунов, Ф.Г.Сигаев, "Введение в полнотекстовый поиск в PostgreSQL".http://www.sai.msu.su/~megera/postgres/talks/fts_pgsql_intro.html

[16] SPECjAppServer2004 Result, SPEC (2007-07-07). http://www.spec.org/jAppServer2004/results/res2007q3/jAppServer2004-20070703-00073.html

[17] Мой Круг, http://moikrug.ru/, Работа для Вас, http://www.rabota.ru/ Проект "Астронет", http://www.astronet.ru, Проект "Все о Геологии", http://geo.web.ru, Проект "Научная сеть", http://nature.web.ru.

[18] Lawrence A. Rowe, Michael Stonebraker, "The POSTGRES Data Model", Proceedings of the 13th International Conference on Very Large Data Bases, 1987. http://www.vldb.org/conf/1987/P083.PDF

[19] Michael Stonebraker, "Inclusion of new types in relational database systems", In Proceedings of the 4th IEEE International Conference on Data Engineering, pp. 262-269, Washington, D.C., 1986

[20] O. Bartunov, "Full-Text Search in PostgreSQL", presentation at PGDay conference, 2007, Prato, Italy. http://www.sai.msu.su/~megera/postgres/talks/fts-pgday-2007.pdf

[21] О.С.Бартунов, Ф.Г.Сигаев, "Написание расширений для PostgreSQL с использованием GiST", 2005. http://www.sai.msu.su/~megera/postgres/talks/gist_tutorial.html

[22] Michael Stonebraker and Greg Kemnitz, "The Postgres Next Generation Database Management System", 1991, Commun. ACM, v.34, 10, pp. 78-92.

Specialized data types for digital libraries

O.S. Bartunov, T.G.Sigaev

Complex modern informational systems require specialized data types, optimized for fast access and tasks of informational retrieval. Rapid changes in patterns of access to information require extensibility of database engine to allow experts in the data domain to develop custom data type, optimized for data domain. We describe several data types, developed for the open-source ORDBMS PostgreSQL, which facilitate operations with sets, hierarchical data, semistructured data and full-text search. Also, we describe PostgreSQL infrastructure for developing extensions.

♣Авторы выражают благодарность Российскому Фонду Фундаментальных Исследований за многолетнюю поддержку их исследований (гранты 05-07-90225-а, 06-07-89182-а).