MySQL

Урок 8

Альтернативные СУБД и функциональные надстройки

Знакомство с PostgreSQL. Модули Sphinx, Elastic search. Хранилище Redis.

Что такое PostgreSQL

Мы уже познакомились с СУБД MySQL и умеем работать с ней на базовом уровне. Но знание только одного инструмента для решения задач – это не очень хорошо. Поэтому стоит рассмотреть в качестве решения СУБД PostgreSQL. Почему именно PostgreSQL? Что может дать эта СУБД в отличие от аналогов (MySQL, Firebird)? Сами создатели PostgreSQL заявляют, что их система «самая продвинутая база данных с открытым исходным кодом в мире». Рассмотрим причины, по которым это утверждение может быть истинным.

Установка на Ubuntu производится следующим образом:

```
sudo apt-get install postgresql postgresql-contrib
pg_createcluster 9.3 main --start
```

Теперь можно переключиться на пользователя postgres, из-под которого можно настраивать БД по умолчанию.

```
sudo -i -u postgres
psql
```

Мы видим, что база доступна. Выйдем из интерфейса postgres и создадим пользователя для работы:

```
createuser --interactive
```

Создадим пользователя pgadmin со всеми правами. Также надо отредактировать файл /etc/postgresql/<version>/main/pg_hba.conf, добавив туда строчку:

```
host    all             pgadmin         192.168.230.67/32       md5
```

Теперь можно подключаться к самой БД. Для работы с PostgreSQL вам понадобится соответствующий клиент, например, pgAdmin (https://www.postgresql.org/).

Хранение данных

MySQL, с которой мы уже знакомы, является реляционной БД. В свою очередь, PostgreSQL не просто реляционная, а объектно-реляционная БД. Это означает, что она поддерживает пользовательские объекты, их поведение, типы данных, функции и многое другое.

Таким образом, уже на уровне БД мы можем определять сущности более конкретно, чем таблицы и поля в них. Т. е. это что-то вроде ООП, но на уровне данных со всеми вытекающими – инкапсуляция, полиморфизм, наследование, переопределение методов и т. п.

Есть очень много типов данных, поддерживаемых в Postgres. Здесь есть не только множество вариаций стандартных типов, но и денежный тип, перечисляемый, геометрический, бинарный; сетевые адреса, битовые строки, текстовый поиск, xml, json, массивы, композитные типы и диапазоны. В различных вариациях и MySQL, и Firebird также знают некоторые из перечисленных типов данных, но лишь Postgres поддерживает их все.

Очень интересным типом хранения в Postgres является массив. Postgres – это объектно-реляционная БД, массивы значений здесь хранятся практически для всех типов данных. Создать их можно, добавляя квадратные скобки к спецификации типа данных или при помощи выражения ARRAY. Указание размерности массива необязательно. Рассмотрим пример применения данного типа через меню пикника:

```
-- создаем таблицу, у которой значения являются массивами
CREATE TABLE holiday_picnic (
     holiday varchar(50), -- строковое значение
     sandwich text[], -- массив
     side text[] [], -- многомерный массив
     dessert text ARRAY, -- массив
     beverage text ARRAY[4] -- массив из 4-х элементов
);
 -- вставляем значения массивов в таблицу
INSERT INTO holiday_picnic VALUES
     ('Labor Day',
     '{"roast beef","veggie","turkey"}',
     '{
        {"potato salad","green salad"},
        {"chips","crackers"}
     }',
     '{"fruit cocktail","berry pie","ice cream"}',
     '{"soda","juice","beer","water"}'
     );
```

Например, MySQL не умеют совершать такое (JSON type). Чтобы создать нечто подобное, придется придумывать обходной путь и строить отдельную таблицу для каждого из значений массива.

Ещё один интересный тип – геометрические данные. Они требуются в реализациях задач, связанных с геолокацией. Геоданные в настоящее время часто становятся неотъемлемой частью многих приложений. PostgreSQL поддерживает большой набор геометрических типов данных, таких как точки, линии, круги и многоугольники. Например, тип PATH (путь), который состоит из последовательности точек. Путь может быть открытым (начальная и конечная точки не связаны) или закрытым (начальная и конечная точки связаны). Рассмотрим в качестве примера пешеходный маршрут, идущий по петле – т. е. начальная и конечная точки связаны. Круглые скобки вокруг набора координат указывают на закрытый путь, а квадратные – на открытый.

```
-- создаем таблицу для хранения троп
CREATE TABLE trails (
     trail_name varchar(250),
     trail_path path
);
 -- вставляем тропу в таблицу,
 -- для которой маршрут определяется координатами в формате широта-долгота
INSERT INTO trails VALUES
     ('Dool Trail - Creeping Forest Trail Loop',
    '((37.172,-122.22261666667),
     (37.171616666667,-122.22385),
     (37.1735,-122.2236),
```

```
     (37.175416666667,-122.223),
     (37.1758,-122.22378333333),
     (37.179466666667,-122.22866666667),
     (37.18395,-122.22675),
     (37.180783333333,-122.22466666667),
     (37.176116666667,-122.2222),
     (37.1753,-122.22293333333),
     (37.173116666667,-122.22281666667))');
```

Расширение PostGIS позволяет расширить свойства геометрических данных дополнительными пространственными типами, функциями, операторами и индексами. Это расширение позволяет приложениям поддерживать местоположения, растровые и векторные данные, а также даёт совместимость с многими сторонними гео-API.

В MySQL 5.7.8 добавлены подобные расширения для соответствия стандарту OpenGIS. Предлагается хранение, аналогичное Postgres. Но в MySQL значения данных сначала конвертируются в геометрический формат простыми командами.

Поддержка JSON в PostgreSQL позволяет отказаться от строгих схем хранения данных для некоторых сущностей в SQL БД. Это полезно, когда структура требует гибкости: к примеру, если в процессе разработки структура постоянно меняется или заранее неизвестно, какими полями будет обладать объект данных.

В MySQL 5.7.8 добавлена поддержка объектов JSON. Но они индексируются хуже, чем JSONB в PostgreSQL.

Создание нового типа

Теперь рассмотрим процесс создания нового типа данных, который вы хотели бы хранить в PostgreSQL. Для этого потребуется команда CREATE TYPE, которая создаёт новые типы данных: составной, перечисляемый, диапазон или базовый.

```
-- создаем новый составной тип "wine"
CREATE TYPE wine AS (
     wine_vineyard varchar(50),
     wine_type varchar(50),
     wine_year int
);
 -- создаем таблицу, которая использует составной тип "wine"
CREATE TABLE pairings (
     menu_entree varchar(50),
     wine_pairing wine
);
 -- вставляем данные в таблицу при помощи выражения ROW
INSERT INTO pairings VALUES
     ('Lobster Tail',ROW('Stag''s Leap','Chardonnay', 2012)),
     ('Elk Medallions',ROW('Rombauer','Cabernet Sauvignon',2012));
 /*
   выборка из таблицы с использованием имени колонки
   (используйте скобки, отделяемые точкой от имени поля
   в составном типе)
```

```
 */
SELECT (wine_pairing).wine_vineyard, (wine_pairing).wine_type
FROM pairings
WHERE menu_entree = 'Elk Medallions';
```

Объёмы данных

Разумеется, ничто в IT-мире не является неограниченным физически. И PostgreSQL не исключение. СУБД имеет следующие ограничения:

- Максимальный размер базы данных не ограничен;
- Максимальный размер таблицы: 32 TB;
- Максимальный размер строки: 1.6 TB;
- Максимальный размер поля: 1 GB;
- Максимальное количество строк в таблице не ограничено;
- Максимальное количество столбцов в таблице: 250–1600;
- Максимальное количество индексов в таблице не ограничено.

Естественно, это не означает, что нужно использовать максимальные настройки для всех возможных параметров. Как и с другими БД, следует с умом создавать таблицы и добавлять индексы.

В MySQL есть ограничение размера строк в 65 535 байт. Как правило, общий объём данных лимитируется максимальным размером файла для выбранной операционной системы. PostgreSQL может сохранять данные в наборе файлов, что позволяет обходить это ограничение. Разумеется, большое количество файлов негативно скажется на скорости работы.

Индексы

Postrges имеет широкие возможности индексирования данных. Например, это частичные индексы, которые создаются при необходимости индексации подмножества данных. Это могут быть строки, отфильтрованные по заданным условиям. Данный функционал позволяет сжимать размеры индексов, необходимых для выполнения узкоспециализированных задач.

```
-- создание индекса только для платящих пользователей
CREATE INDEX order_idx ON orders (order_id)
WHERE status_id > 5;
```

Также в Postgres есть функциональные индексы, которые создаются при помощи любой функции для предварительного вычисления данных для индексирования. К примеру, у вас есть список наименований, который содержит в себе некие ключевые слова в разных форматах. В такой таблице можно создать индекс, который приводит все ключевые слова к нижнему регистру для нормализации данных

```
-- создание индекса для URL в нижнем регистре
CREATE INDEX webhits_lower_urls_idx ON webhits (lower(url));
```

Выводы

СУБД PostgreSQL чрезвычайно богата функционалом, имеющим много встроенных приспособлений, и огромным количеством методов их кастомизации. С учётом признания СУБД со стороны сообщества разработчиков, можно говорить, что данное решение для БД устроит многие команды. Мы рассмотрели небольшое количество возможностей, которые предоставляет Postgres по сравнению с другими решениями. Естественно, их гораздо больше, чем мы перечислили выше.

Sphinx

Мы подробно рассмотрели СУБД MySQL и познакомились с СуБД PostgreSQL. Однако, базы данных, какими бы производительными они ни были, всё равно остаются одним из наиболее медленных мест в системе. И одной из задач, создающих максимальную нагрузку, является задача поиска по массивам данных. Сам по себе SQL ни в одной реализации из коробки не даёт возможности искать данные быстро. Полнотекстовый поиск – это поиск вхождений в БД на основании содержимого, а также совокупность подходов к оптимизации данного процесса.

Для решения подобной задачи существует поисковый механизм Sphinx. Это ПО для полнотекстового поиска, которое разработано Андреем Аксёновым и распространяется по лицензии GNU GPL. Преимуществом решения является быстрота индексации и процесса поиска. Механизм интегрируется с такими СУБД, как MySQL и PostgreSQL, а также имеет API для популярных языков программирования: PHP, Python, Java.

Одним из самых крупных примеров применения Sphinx является проект бесплатных объявлений Craiglist – ежесуточно он обрабатывает 300 миллионов запросов на поиск.

Для работы Sphinx изначально строит индекс, который описывается в файле конфигурации, затем проводит поиск по выстроенному индексу. Но ведь на многих сайтах данные обновляются ежесекундно. Выходит, чтобы не искать по старым данным, нужно будет каждую секунду перестраивать индекс?

Для решения данной проблемы существует симбиоз двух индексов. Главный (main-index) хранит основные данные, а дополнительный (delta-index) – это небольшой фрагмент данных, хранящий только данные, не вошедшие в главный индекс.

Дополнительный индекс переиндексируется быстро, т. к. работает с меньшим количеством данных. Поэтому его и пересчитывать можно чаще. Раз в сутки происходит переиндексация главного индекса. При этом дополнительный индекс сбрасывается.

ElasticSearch

Одним из основных конкурентов Sphinx на рынке поисковых движков является ElasticSearch. В отличие от Sphinx решение ElasticSearch является надстройкой над библиотекой Apache Lucene. Lucene без надстроек мало на что годен, т. к. это библиотека для реализации поисковиков. Она может только индексировать и искать. API для ввода данных, поисковых запросов, кластеризации и прочее отдается на уровень «обертки».

ElasticSearch хорошо масштабируется. Это означает, что можно на лету подключать новые серверы, а движок сам распределит нагрузку между ними. Данные при этом распределятся так, что при отказе

одного из серверов потери не произойдет, а система продолжит работать.

Каждому изменению данных соответствую записи в логах сразу на нескольких серверах кластера, что повышает отказоустойчивость и сохранности данных.

ElasticSearch в отличии от Sphinx даёт возможность загружать обычный JSON-объект, после чего он сам все проиндексирует.

Движок почти полностью управляется через HTTP при помощи запросов в JSON-формате.

У ElasticSearch отсутствуют встроенные системы ограничения доступа. При установке он занимает порт 9200 на все интерфейсы, и это довольно опасная уязвимость. До версии 1.2 было доступно выполнение скриптов в запросах. Таким образом, необходимо учитывать это и закрывать данный порт файрволом.

В список преимуществ Sphinx можно отнести высокую скорость индексации данных. Если вам надо быстро обновить индекс по 10 миллионам документов, это несомненный плюс. Также Sphinx менее требователен к ресурсам сервера. В свою очередь, ElasticSearch имеет удобное и богатое API, которое позволяет сделать поиск более релевантным.

Redis

Помимо уже знакомых нам реляционным СУБД есть базы данных, которые не используют реляционную модель. Redis сохраняет данные в оперативной памяти, а доступ к ним предоставляет по ключу. По желанию пользователя бэкап данных может сохраняться на жёстком диске. Такой подход обеспечивает производительность, в десятки раз большую, чем у реляционных СУБД.

Redis позволяет хранить целые массивы данных, словари, множества уникальных значений, сортированные множества. СУБД позволяет задать время жизни данных, но по умолчанию они хранятся вечно.

Технологическая особенность Redis в том, что это однопоточный сервер. Этот подход упрощает поддержку кода, разрешает проблему атомарности операций и даёт возможность запускать по одному процессу Redis на каждое ядро процессора. При этом каждый процесс будет работать с выделенным портом.

Redis поддерживает репликацию, но в качестве главного сервера может выступать только один сервер. Однако каждый слейв может выступать в роли главного для других. Реплицирование в Redis не вызывает блокировки на серверах. В случае разрыва связи после восстановлении соединения главный и подчиненный сервер производят полную синхронизацию.

Redis поддерживает механизм publish/subscribe. Этот механизм позволяет создавать каналы, на которые можно подписываться и размещать в них сообщения.

Главное ограничение Redis в том, что объем данных хранимых на физическом сервере, лимитирован объемом RAM. Таким образом, хранение в Redis больших данных стоит дорого, но вполне приемлемы следующие варианты применения:

- хранилище сессий пользователей;
- сервер очередей;
- полноценная замена Memcached;
- кэширующее приложение;
- СУБД для небольших приложений;
- хранилище промежуточных результатов вычислений при обработке больших объемов данных.

Практическое задание

1. Установить и запустить СУБД PostgreSQL.
2. * Установить и запустить Sphinx. Проверить его работоспособность.

Дополнительные материалы

1. «PostgreSQL. Для профессионалов» Дж. Уорсли, Дж. Дрейк.

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. https://xakep.ru/2015/06/11/elasticsearch-tutorial/
2. http://chakrygin.ru/2013/03/sphinx-install.html