электронный журнал
МОЛОДЕЖНЫЙ НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК
Издатель ФГБОУ ВПО "МГТУ им. Н.Э. Баумана". Эл No. ФС77-51038.

 # 02, февраль 2016

УДК 004.652.4+004.656

Организация хранения конфигурации приложения в реляционнной и нереляционной СУБД

***Сафонова А.В.**, студент*
кафедра «Теоретическая информатика и компьютерные технологии»,
Россия, 1050005, г. Москва, МГТУ им. Н. Э. Баумана

Научный руководитель: Вишняков И.Э., ст. преподаватель
Россия, 1050005, г. Москва, МГТУ им. Н. Э. Баумана,
vishnyakov@bmstu.ru

Введение. В наше время, когда объемы информации, которую нужно хранить и обрабатывать, достигли огромных размеров, появляется проблема высоконагруженных систем. В сфере баз данных все большее распространение получает направление NoSQL (No only SQL). Чаще всего под этим термином понимается противопоставление привычным реляционным базам данных, находящимся в использовании уже более 30 лет. Среди них можно выделить такие типы баз данных, как хранилища ключей и значений, объектно-ориентированные, документо-ориентированные, столбцовые и графовые. СУБД, относящиеся к этому направлению, отличаются от реляционных баз данных лучшей масштабируемостью, переносимостью и интеграцией с кодом. Тем не менее SQL базы данных не сходят со своих позиций: они устойчивы, в них легко поддерживать целостность данных, есть механизм транзакций и т.д. Поэтому на сегодняшний день разработчики все чаще сталкиваются с вопросом, какие средства лучше использовать при работе с данными – традиционные и знакомые всем реляционные базы данных или же новые, отвечающие специфическим требованиям NoSQL аналоги. Естественно, ответ на этот вопрос зависит от области применения.

Данная работа посвящена сравнению двух встраиваемых СУБД: SQLite – компактной реляционной базы данных, и LevelDB – хранилища типа ключ-значение. Целью является сравнительная оценка использования каждой из СУБД для решения задачи хранения конфигурации подсистемы обработки потока данных. Требуется проверить, какая из указанных СУБД будет иметь преимущество с точки зрения производительности (скорости выполнения вставки, удаления и обновления данных), простоты и удобства

написания кода при создании БД и формировании запросов.

Описание конфигурации приложения. Очень часто работа приложения сопровождается созданием файлов конфигурации, в которых располагается информация, необходимая для корректного функционирования программы. Независимо от того, какие именно данные содержит конфигурация, важно заметить, что ее общий размер может быть очень большим. Поэтому хранение конфигурации в оперативной памяти может оказаться невозможным, в связи с чем целесообразно пользоваться базами данных.

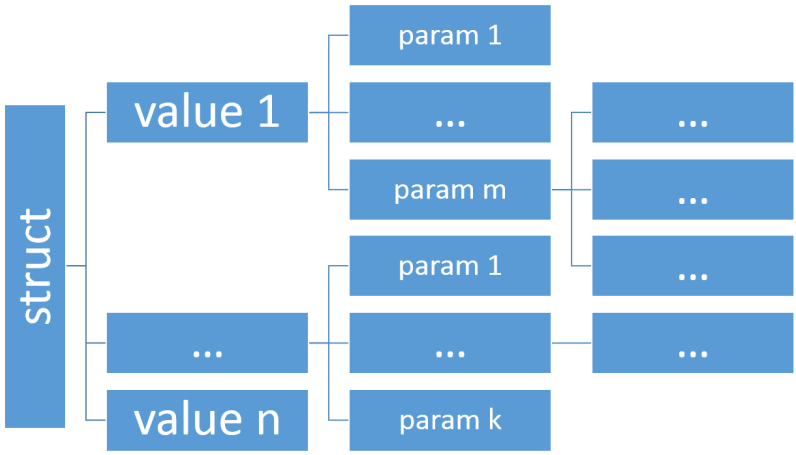

Рис. 1. Структура конфигурации

Рассмотрим структуру абстрактной конфигурации (рис. 1). Предполагается, что она имеет древовидную структуру и обладает несколькими уровнями вложенности. Количество элементов каждого уровня не является фиксированным и может постоянно изменяться (элементы могут добавляться и удаляться), в связи с чем хранение конфигураций в плоском файле является затруднительным. С учетом этого для удобства проектирования модели данных конфигурации можно разделить на две группы: часто изменяемые и редко изменяемые, размер отдельных полей при этом невелик, но общее их число может быть большим и конфигурация может достигать крупных размеров. Кроме того, нужно заметить, что информация, хранящаяся в конфигурации, разнородна. Таким образом, требования к абстрактному описанию конфигурации потока данных можно обобщить следующим образом: это иерархическая структура с произвольным количеством полей, которые могут постоянно меняться.

Исходя из этого, можно сделать вывод, что для хранения подобного набора информации может подойти такой тип NoSQL баз, как хранилище типа ключ-значение.

Привлекательность данного способа заключается в его простоте. Нам не требуется проектировать строгую модель данных, и внесение изменений в набор полей структур не вызовет сложностей.

Выбор СУБД для хранения конфигурации. В настоящее время существует большое разнообразие как реляционных, так и NoSQL СУБД, предоставляющих разнообразные возможности. Принимая во внимание сравнительно небольшой объём хранимых данных и отсутствие необходимости многопользовательского доступа к ним, имеет смысл рассматривать компактные встраиваемые СУБД.

Таким образом, в качестве реляционнного хранилища была выбрана SQLite – встраиваемая кроссплатформенная база данных, которая поддерживает достаточно полный набор команд SQL (но в ней отсутствуют, например, такие возможности, как удаление и изменение столбцов в таблице, хранимые процедуры, поддержка UNICODE) и доступна в исходных кодах, на использование которых нет никаких ограничений. Эта СУБД легко переносима, очень проста и удобна для встраивания, поэтому данная библиотека широко используется в браузерах, музыкальных плеерах и многих других программах [1].

В качестве хранилища типа ключ-значение была выбрана LevelDB. Она, так же как и SQLite, является свободно распространяемой встраиваемой СУБД. Библиотеку LevelDB на C++ можно использовать для разных целей. Например, веб-браузер может обрабатывать с помощью LevelDB кэш недавно посещённых страниц, операционная система — список установленных пакетов и зависимостей между ними, а любое приложение может использовать LevelDB для хранения пользовательских настроек.

Разработчиками LevelDB было произведено сравнение двух описанных выше СУБД [2], результаты которых приведены в таблице 1 и показывают, что производительность СУБД зависит от свойств данных, с которыми производятся операции. Поэтому требуется сравнить целесообразность применимости указанных баз данных на конкретном примере. При этом критериями сравнения будут являться производительность (скорость выполнения вставки, удаления, чтения и обновления элементов конфигурации), простота и удобство написания кода при создании базы данных и формировании запросов.

Таблица 1

Результаты сравнения SQLite и LevelDB

СУБД	Запись 1000 значений размером 100000 байт, операций/с	Запись 1000 партий по 1000 значений, размером 100 байт, операций/с
SQLite	1600	124000
LevelDB	480	840000

Разработка схемы хранения конфигурации в СУБД LevelDB. Одной из особенностей хранилищ типа ключ-значение является более естественная интеграция с кодом и, как следствие, отсутствие необходимости разработки модели данных с точки зрения самого хранилища [3]. Сложно сказать, что это является плюсом, поскольку необходимость такого процесса при использовании реляционной модели обеспечивает независимость данных от приложения. Кроме того, ограничения в реляционных базах гарантируют целостность данных на самом низком уровне, в то время как в хранилищах типа ключ-значение они отсутствуют. Таким образом, данные в хранилище типа ключ-значение должны соответствовать поддерживаемой на уровне приложения схеме хранения, определяемой исходя из решаемой данным приложением задачи.

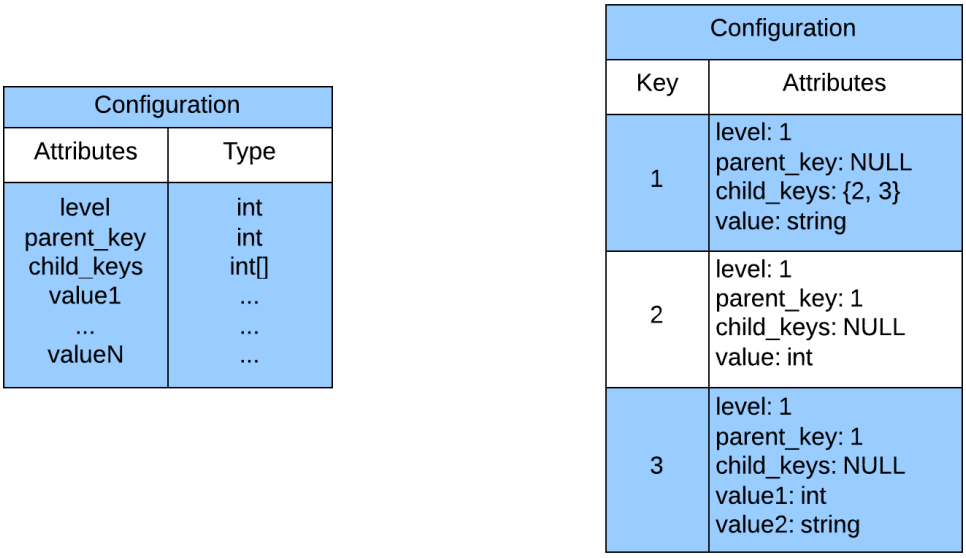

Рис. 2. Представление дерева в хранилище типа ключ-значение

Разнородность полей полностью вписывается в парадигму хранилищ рассматриваемого типа, согласно которой в качестве значения могут храниться данные произвольных типов и структуры. Таким образом, отдельное поле конфигурации и вся относящаяся к нему служебная информация будет храниться в виде одного значения (рис. 2). В качестве ключа можно использовать некоторое суррогатное значение. Поддержка разветвленной структуры конфигурации будет осуществляться следующим образом: внутри одной записи, соответствующей определенному ключу, можно выделить такие обязательные поля, как номер уровня вложенности, ключ поля-родителя и массив ключей полей-потомков. При реализации такой схемы хранения важно следить за сохранением свойств дерева: обеспечить обязательное наличие родителей у всех записей, кроме

корневой, и отсутствие циклов внутри общей структуры. То есть для любого значения в поле, соответствующем ключу родительской записи не может содержаться значение, принадлежащее множеству ключей записей, стоящих выше данного в дереве, и сама запись не может быть своим родителем или потомком. При этом операция удаления вершины должна работать таким образом, что при удалении некоторого узла дерева, соответствующее ему поддерево, также должно удаляться из общей схемы.

Разработка схемы хранения конфигурации в СУБД SQLite. Основная проблема в случае использования реляционной СУБД заключается в хранении деревьев в базе данных. На первый взгляд это кажется непростой задачей, для которой реляционные базы не предназначены, тем не менее существуют различные подходы к представлению иерархических структур в SQL, некоторые из которых описаны в [4]. Рассмотрим несколько методов, обращая внимание на принцип выполнения вставки и удаления узлов и поиск определенного поддерева.

Первый способ называется «Список смежности» (Adjacency list model) и является прямой реализацией представления графа в виде списка смежности. Данный метод подразумевает наличие двух основных таблиц: одна из них отвечает за узлы дерева, а вторая — за связь между ними.

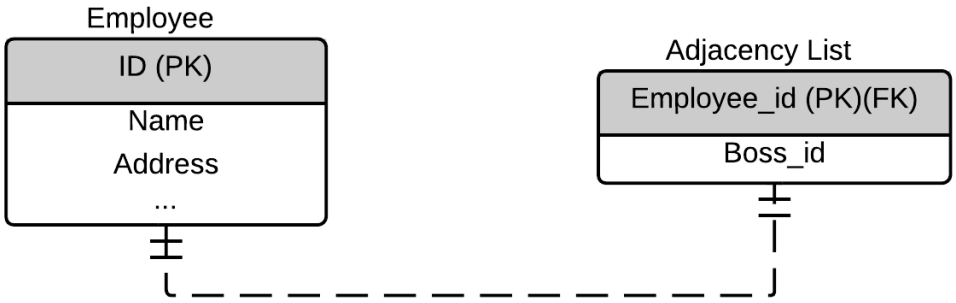

Рис. 3. Простейший пример использования списка смежности

На рисунке 3 представлен пример иерархической структуры управления некоторой организацией. Уникальный идентификатор в таблице Employee выбирается из множества натуральных чисел. В таблице Adjacency List, определяющей отношение подчинения, записано соответствие идентификаторов руководителей и подчиненных. Поскольку в дереве количество ребер на единицу отличается от количества вершин, то каждому

идентификатору Employee_id должно соответствовать единственное значение Boss_id, за исключением первого, корневого элемента, в котором оно инициализируется пустым значением. Все эти условия необходимо соблюдать при заполнении таблиц.

Сильной стороной данного метода является вставка новых элементов в дерево, поскольку для этого достаточно вставить пару (Employee_id, Boss_id) в таблицу Adjacency List. Недостатком данного способа является отсутствие единого метода извлечения поддерева. Конечно, можно спускаться в цикле по дереву, перебирая все пары (Employee_id, Boss_id) в базе, и выбирать нужные. Но этот способ будет показывать очень низкую скорость при больших размерах деревьев. При фиксированной известной глубине искомого поддерева можно использовать соединение таблицы Adjacency List с самой собой на основании равенства полей Employee_id из одной таблицы и Boss_id из другой. Также можно найти поддерево заданной глубины при помощи рекурсии. Но все эти способы не позволяют выводить полное поддерево, не зная его глубины. Поскольку при удалении одного узла должно удаляться все соответствующее ему поддерево, данный метод также оказывается не очень удобным.

Organisation		
Employee	ID	Path
'Albert'	A	A
'John'	B	AB
'Ann'	C	AC
'Fred'	D	ACD
'Chuck'	E	ACE

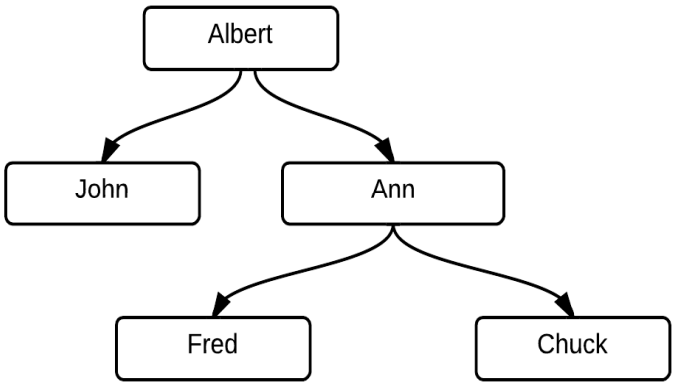

Рис. 4. Таблица для метода «Перечисление пути» и дерево, которое она описывает

Другой способ называется «Перечисление пути» (Path Enumeration Model) и основывается на том, что от корня до любого элемента дерева существует только один путь. Именно он помещается в строку таблицы, соответствующую данному узлу (рис. 4). Получается такой путь посредством конкатенации ключей записей, которые в рассматриваемом примере имеют тип char. Поиск осуществляется при помощи строковых функций и предикатов. Вставка узла в дерево осуществляется так же просто, как и в предыдущем способе – нужно только найти путь от корня до новой вершины. Для этого

достаточно взять уже сформированный путь родителя и присоединить к его концу ID вставленного элемента.

При реализации этого метода обеспечение свойств дерева основывается на следующем. Деревья являются ациклическими графами, а, следовательно, символы в пути не могут повторяться, и максимальная длина пути не может превышать общее число вершин поддерева. При вставке такие ограничения нетрудно обеспечивать средствами SQL. Для удаления поддерева требуется удалить все вершины, в пути к которым содержится ключ, соответствующий удаляемому поддереву. Аналогичным образом можно делать и выбор поддерева. Недостаток этого метода заключается в необходимости использования инструкции LIKE, у которой предикат начинается с '%', а эта инструкция является затратной с точки зрения времени.

Третий способ основан на представлении деревьев как вложенных множеств (Nested Sets Model). Чтобы понять эту модель, рассмотрим рисунок 5.

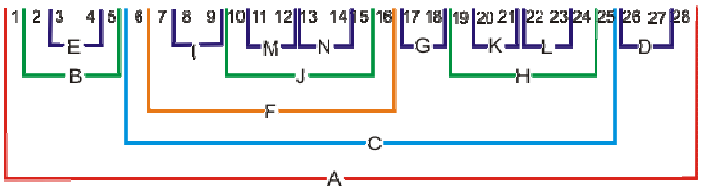

Рис 5. Вложенные множества

Буквы на рисунке отвечают за узлы дерева, а границы определяют вложенность их друг в друга. Также можно заметить, что каждый узел своими границами захватывает часть ряда чисел от единицы до количества вершин, то есть каждому узлу можно сопоставить два числа, соответствующих началу и концу интервала. Эти числа, в которые как раз заложена вся информация о дереве, будем называть правым и левым ключами. Если информацию о ключах занести в базу данных, то работа с деревом намного упрощается. При этом помимо правого и левого ключа в таблицу базы целесообразно добавить номер уровня вложенности данного узла и его уникальный идентификатор (рис. 6).

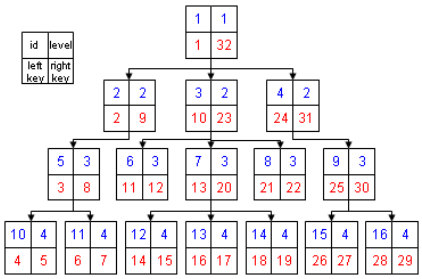

Рис 6. Представление дерева на основе вложенных множеств

В данной модели присутствуют следующие ограничения, которые требуется соблюдать при заполнении таблицы:

1. Левый ключ всегда меньше правого.
2. Наименьший левый ключ всегда равен 1.
3. Наибольший правый ключ всегда равен двойному числу узлов.
4. Разница между правым и левым ключом всегда нечетна.
5. Четность уровня всегда совпадает с четностью ключа.
6. Как правые, так и левые ключи уникальны.

Нетрудно заметить, что при таком подходе очень просто осуществляется выборка вершин поддерева. Для этого нужно выбрать те вершины, у которых левый ключ больше левого ключа корня этого поддерева, но меньше правого ключа. Отсортировав полученные записи по левому ключу, получим вершины дерева в порядке иерархии. С точки зрения хранения конфигурации данный метод является наилучшим, так как наиболее приоритетным действием является выборка поддерева, в то время как изменение структуры не будет являться часто выполняемой операцией.

В дополнение к описанной схеме хранения дерева в базе данных нужно обеспечить для него такие свойства, как неограниченное количество полей и их разнородность. Для этого вынесем значимые поля дерева в отдельную таблицу, содержащую поле со ссылкой на себя. Данный прием позволит обеспечить хранение в узле конфигурации списка значений различных типов. Само же значение будет храниться в таблице, соответствующей типу значения. Для этого можно применить шаблон проектирования категория-подтип, при этом поле, отвечающее за тип, выступит в роли дискриминатора. На рисунке 7 изображена финальная схема данных.

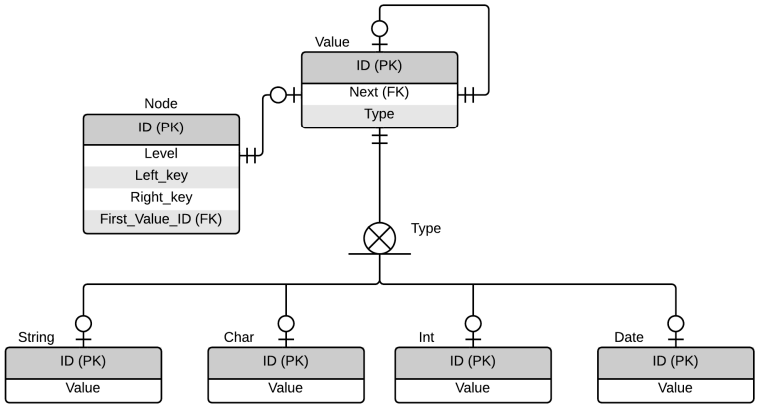

Рис 7. Реляционная модель базы данных для хранения дерева

Тестирование и оценка производительности. Для примера рассматривается приложение, которое по заданной маске пути выводит список файлов и каталогов, располагающихся в конечной директории для каждого найденного пути. Заданной маске, очевидно, может соответствовать несколько путей, а значит, и выведено должно быть несколько списков. При реализации данного приложения используется ряд структур, хранящих информацию об уже обработанных путях и результатах поиска. Требуется обеспечить возможность сохранять промежуточную конфигурацию и загружать ее для продолжения поиска.

Остановимся подробнее на устройстве конфигурации. Она представляется структурой, в которой имеются следующие поля простых типов: маска пути, количество уровней вложенности, список вложенных файлов и папок и массив структур (с полями простых типов), отвечающих за состояние поиска на каждом уровне вложенности. Таким образом, данную конфигурацию можно представить в виде дерева с высотой, равной двум, корнем которого будет являться основная структура, а листьями — элементы массива вложенных структур.

Замеры производительности реализованных функций сохранения и загрузки конфигурации осуществлялись на следующей задаче поиска файлов по пути: «/home/*/*/*/*/*/*/*/*». Конфигурация, соответствующая данному запросу, состоит из 11 узлов и занимает в среднем 3 килобайта (обрабатывается порядка 100 конфигураций). Тестирование производилось на оборудовании, оснащенном процессором Intel Core i5 с тактовой частотой 1.80 ГГц, и оперативной памятью объемом 4 ГБ. Замеры времени при работе с целой конфигурацией приведены в таблице 2, а с отдельными узлами конфигурации - в таблице 3.

Таблица 2

Время выполнения основных операций с целой конфигурацией

СУБД	Время выполнения операции, мс		
	Вставка	Выборка	Удаление
SQLite	7,700	2,940	0,040
LevelDB	4,970	3,230	0,010

Таблица 3

Время выполнения основных операций с узлами конфигурации

СУБД	Время выполнения операции, мс		
	Вставка	Обновление	Удаление
SQLite	0,480	0,020	0,020
LevelDB	0,016	0,016	0,010

Результаты показывают, что по времени выполнения операций LevelDB превосходит SQLite в случае сохранения и удаления полной конфигурации и незначительно отстает при выборке конфигурации из базы. В случае с изменением отдельных узлов конфигурации LevelDB имеет преимущество более чем в 10 раз при вставке и почти в 2 раза при обновлении и удалении узла. При этом LevelDB является намного более удобной с точки зрения хранения иерархических структур и написания кода.

Заключение. При решении задачи организации хранения конфигурации были рассмотрены встраиваемые СУБД SQLite и LevelDB. Разработанные для этих СУБД схемы хранения обеспечивают эффективное с точки зрения выполнения основных операций представление иерархических конфигураций с листовыми узлами произвольных типов. Результаты сравнительного тестирования показывают, что для создания хранилища конфигураций наиболее подходящей базой данных является LevelDB, как с точки зрения производительности, так и с точки зрения простоты и удобства использования.

Для более глубокого изучения производительности созданных баз с точки зрения поставленной задачи, возможно, их тестирование на более крупных конфигурациях (имеющих большее число уровней вложенности и количество узлов), а также изучение влияния количества вставленных записей на скорость выполнения запросов.

Список литературы

[1]. SQLite. Available at: http://www.sqlite.org/cintro.html, accessed 26.03.2015.

[2]. LevelDB. Available at: https://code.google.com/p/leveldb, accessed 26.03.2015.

[3]. Редмонд Э., Уилсон Д.Р. Семь баз данных за семь недель. Введение в современные базы данных и идеологию NoSQL. М.: ДМК-Пресс, 2012. 384 с.

[4]. Celko J. Trees and Hierarchies in SQL for Smarties. Waltham, Massachusetts, USA: Morgan Caufmann, 2012. 277 p.