УДК 621.311

ПЕРСПЕКТИВНЫЕ НАПРАВЛЕНИЯ РАЗВИТИЯ БД И ТЕХНОЛОГИИ ВЗАИМОДЕЙСТВИЯ С НИМИ

Длусская А.Ю.,
Научный руководитель - Куприянов А.Б. к.т.н., доцент

С развитием Интернета и облачных технологий, обычные реляционные БД перестали быть универсальными. Базы данных растут и не помещаются на одном сервере, необходимо хранить данные на разных серверах. В такой ситуации сложно использовать оператор соединения таблиц. Единственный вариант использования реляционной БД – это на разные серверы отправлять несвязанные данные.

Поэтому были разработаны не реляционные БД(NoSQL). В зависимости от модели данных можно выделить 4 типа БД: «ключ-значение», документно-ориентированные, БД семейств колонок, графовые БД.

База данных, основанная на хранилище «ключ-значение», считается базой реализацией NoSQL. Они работают путём сопоставления ключей со значениями, между которыми нет ни структуры, ни отношений. Преимущества этих БД – это быстродействие, производительность и масштабируемость. Такие БД используются для хранения изображений, создания специализированных файловых систем. Примеры таких хранилищ — Berkeley DB, MemcacheDB, Redis, Riak, Amazon DynamoDB.

БД семейств колонок расширяют БД «ключ-значение». БД семейств колонок являются двумерными массивами. Каждому ключу прикреплена одна или несколько пар «ключ-значение». В отличии от привычных таблиц в реляционных моделях, эти СУБД не требует предварительного описания структуры данных. Эти системы управления позволить хранить и использовать очень большие объёмы неструктурированных данных. Такие БД используются в случаях, когда БД «ключ-значение» недостаточно и необходимо хранить большие объёмы данных. От БД «ключ-значение» была унаследована масштабируемость. Примерами СУБД данного типа являются: Apache HBase, Apache Cassandra, Apache Accumulo, Hypertable.

Документо-ориентированные базы данных NoSQL пользуются огромной популярностью среди пользователей. Эти СУБД похожи на БД колонок, однако позволяют создать более сложную структуру (документ, вложенный в документ, вложенный в документ.). Документы устраняют некоторые ограничения хранилищ колонок. В целом они позволяют создать документ из произвольной структуры данных любой сложности. Несмотря на множество преимуществ, документо-ориентированные СУБД имеют некоторые недостатки и уязвимости по сравнению с другими СУБД

NoSQL. Например, извлекая значение записи, вы получаете огромный объём данных, а обновление данных негативно влияет на производительность. Находят своё применение в системах управления содержимым, издательском деле, документальном поиске и т. п. Примеры СУБД данного типа — CouchDB, Couchbase, MarkLogic, MongoDB.

Графовые СУБД используют древовидные структуры – графы, которые состоят из узлов и ребёр. Графовые базы данных представляют данные совсем иначе, чем предыдущие три модели. Они используют древовидные структуры – графы, которые состоят из узлов и рёбер. Графовые СУБД соединяют и группируют полученные данные, благодаря чему они намного быстрее справляются с некоторыми операциями. Эти базы данных обычно используются приложениями, которым необходимы четкие границы для подключений. К примеру, при регистрации в любой социальной сети ваш аккаунт связывается с аккаунтами ваших друзей, друзей ваших друзей и т.д. Такую операцию проще всего выполнить при помощи графовой БД. Примеры: Neo4j, OrientDB, AllegroGraph, Blazegraph, InfiniteGraph, Titan.

В не реляционные БД ради масштабируемости и гибкости отсутствует поддержка ACID. ACID – это 4 свойства, характеризующие транзакции: атомарность, согласованность, изоляция, долговечность. Атомарность гарантирует, что ни одна транзакция не будет выполнена частично (транзакция должна иметь тип «все или ничего»). Согласованность – это требования, согласно которому транзакция нормально завершается и фиксирует свои результаты, не противореча согласованности БД. Изоляция означает, что транзакции не могут оказывать влияние друг на друга. Долговечность гарантирует, что после того, как транзакция завершилась и зафиксировала свои результаты в базе данных, система должна гарантировать, что эти результаты переживут любые последующие сбои.

Т.к. не реляционные БД не поддерживают ACID, были разработаны новый тип БД – NewSQL. Этот вид БД соединяется в себе гибкость и масштабируемость NoSQL и транзакционные требования классических БД. В настоящее время существует различные подходы к созданию таких БД: принципиально новая архитектура для хранения данных, новые механизмы хранения MySQL, прозрачное масштабирования.

БД с новой архитектурой изначально рассчитаны на распределенную архитектуру и многопоточность. Одним из ключевых факторов в повышении производительности является использование оперативной памяти или новых видов дисков (флэш-память/SSD), которые являются хранилищем первичных данных. Данное решение может осуществляться программно (VoltDB, NuoDB) либо на уровне железа (Clustrix).

Чтобы преодолеть проблемы масштабируемости MySQL, было создано ряд движков основанных на MySQL. Положительная сторона — использование интерфейса MySQL, но есть плохая сторона — не поддерживается миграция данных из других баз данных (включая старый

MySQL). Примеры реализации — Xeround, GenieDB (коммерческие) TokuTek; и Akiban, MySQL Группа NDB и др. (opensource)

БД с прозрачным масштабированием сохраняют базы данных OLTP в своем оригинальном виде, но обеспечивают особенность расширения, с прозрачной группировкой и гарантирующую масштабируемость. Другой подход должен обеспечить прозрачное распределение данных между физическими серверами, чтобы также улучшить масштабируемость. БД Schooner MySQL, Continuent Tungsten и ScalArc следуют первому подходу, тогда как ScaleBase и dbShards следуют второму подходу. Оба подхода позволяют повторное использование существующих наборов и экосистемы, и избегают потребности переписать код или выполнить любые миграции данных. Примеры реализаций — ScalArc, Schooner MySQL, dbShards (коммерческий) ScaleBase; и Continuent Tungsten (opensource).

Были разработаны технологии для упрощения работы с БД: LINQ, Entity Framework и NHibernate. Эти технологии предназначены для упрощения работы с БД и компиляцией запросов. Суть технологий заключается в том, что работа с таблицами БД ведётся как с классами C#, с полями этих таблиц – как со свойствами классов, а синтаксис SQL-запросов, который в ADO.NET раньше нужно было вставлять в код C# в виде команд, заменен на более удобный подход с LINQ. C# берет на себя обязанности по преобразованию кода C# в SQL-инструкции.

LINQ (Language-Integrated Query) представляет простой и удобный язык запросов к источнику данных. В качестве источника данных может выступать объект, реализующий интерфейс IEnumerable (например, стандартные коллекции, массивы), набор данных DataSet, документ XML. Но вне зависимости от типа источника LINQ позволяет применить ко всем один и тот же подход для выборки данных.

Существует несколько разновидностей LINQ:

- LINQ to Objects: применяется для работы с массивами и коллекциями
- LINQ to Entities: используется при обращении к базам данных через технологию Entity Framework
- LINQ to Sql: технология доступа к данным в MS SQL Server
- LINQ to XML: применяется при работе с файлами XML
- LINQ to DataSet: применяется при работе с объектом DataSet
- Parallel LINQ (PLINQ): используется для выполнения параллельной запросов

В 2009 году разработка LINQ и Entity Framework были переданы одной команде. Стало очевидно, что оба решения нацелены на решение одних и тех же задач, а следовательно будут конкурировать друг с другом. Начиная с версии платформы .NET 4.0, рекомендованным решением становится

именно LINQ к Entities. Кроме того, на основании информации, полученной от пользователей, наиболее употребляемые возможности LINQ к SQL будут добавлены и в LINQ к Entities. В результате чего произойдет постепенное слияние решений.

При работе с Entity Framework предоставляется три способа взаимодействия с БД:

Database-First - подходит для проектировщиков баз данных - сначала вы создаете базу данных с помощью различных инструментов (например, SQL Server Management Studio), а затем генерируете EDMX-модель базы данных (предоставляет удобный графический интерфейс для взаимодействия с базой данных в виде диаграмм и объектную модель в виде классов C#). В данном случае вам нужно работать с SQL Server и хорошо знать синтаксис T-SQL, но при этом не нужно разбираться в C#.

Model-First - Подходит для архитекторов - сначала вы создаете графическую модель EDMX в Visual Studio (в фоновом режиме создаются классы C# модели), а затем генерируете на основе диаграммы EDMX базу данных. При данном подходе не нужно знать ни деталей T-SQL ни синтаксиса C#.

Code-First - подходит для программистов - при данном подходе модель EDMX вообще не используется и вы вручную настраиваете классы C# объектной модели (данный подход поддерживает как генерацию сущностных классов из существующей базы данных, так и создание базы данных из созданной вручную модели объектов C#). Очевидно, что это подходит для программистов, хорошо знакомых с синтаксисом C#.

NHibernate — ORM-решение для платформы Microsoft .NET, портированное с Java. NHibernate позволяет отображать объекты бизнес-логики на реляционную базу данных. По заданному XML-описанию сущностей и связей NHibernate автоматически создает SQL-запросы для загрузки и сохранения объектов.