**Нереляционные базы данных**

NoSQL (not only SQL, не только SQL), в информатике— термин, обозначающий ряд подходов, направленных на реализацию хранилищ баз данных, имеющих существенные отличия от моделей, используемых в традиционных реляционных СУБД с доступом к данным средствами языка SQL. Применяется к базам данных, в которых делается попытка решить проблемы масштабируемости (англ. scalability) и доступности(англ. availability) за счёт атомарности (англ. atomicity) и согласованности данных (англ. consistency).

### **История названия**

Изначально слово NoSQL являлось акронимом из двух слов английского языка: No («Не») и SQL (сокращение от англ. Structured Query Language — «структурированный язык запросов»), что даёт термину смысл «отрицающий SQL». Возможно, что первые, кто стал употреблять этот термин, хотели сказать «No RDBMS» («не реляционная СУБД») или «no relational» («не реляционный»), но NoSQL звучало лучше и в итоге прижилось (в качестве альтернативы предлагалось также NonRel). Позднее для NoSQL было придумано объяснение «Not Only SQL» («не только SQL»). NoSQL стал общим термином для различных баз данных и хранилищ, но он не обозначает какую-либо одну конкретную технологию или продукт.

### **Развитие идеи**

Сама по себе идея нереляционных баз данных не нова, а использование нереляционных хранилищ началось ещё во времена первых компьютеров. Нереляционные базы данных процветали во времена мэйнфреймов, а позднее, во времена доминирования реляционных СУБД, нашли применение в специализированных хранилищах, например, иерархических службах каталогов. Появление же нереляционных СУБД нового поколения произошло из-за необходимости создания параллельных распределённых систем для высокомасштабируемых интернет-приложений, таких как поисковые системы.

В начале 2000-х годов Google построил свою высокомасштабируемую поисковую систему и приложения: Gmail, Google Maps, Google Earth и т. п., решая проблемы масштабируемости и параллельной обработки больших объёмов данных. В результате была создана распределённая файловая система и распределённая система координации, хранилище семейств колонок (англ. column family store), среда выполнения, основанная на алгоритме MapReduce. Публикация компанией Google описаний этих технологий привела к всплеску интереса среди разработчиков открытого программного обеспечения, в результате чего был создан Hadoop и запущены связанные с ним проекты, призванные создать подобные Google технологии. Через год, в 2007 году, примеру Google последовал Amazon.com, опубликовав статьи о высокодоступной базе данных Amazon DynamoDB.

Поддержка гигантов индустрии менее чем за пять лет привела к широкому распространению технологий NoSQL (и подобных) для управления «большими данными», а к делу присоединились другие большие и маленькие компании, такие как: IBM, Facebook, Netflix, Ebay, Hulu, Yahoo!, со своими проприетарными и открытыми решениями.

## **Основные черты**

Традиционные СУБД ориентируются на требования ACID к транзакционной системе: атомарность (англ. atomicity), согласованность (англ. consistency), изолированность (англ. isolation), надёжность (англ. Durability).

Атомарность гарантирует, что никакая транзакция не будет зафиксирована в системе частично. Будут либо выполнены все её подоперации, либо не выполнено ни одной. Поскольку на практике невозможно одновременно и атомарно выполнить всю последовательность операций внутри транзакции, вводится понятие «отката» (rollback): если транзакцию не удаётся полностью завершить, результаты всех её до сих пор произведённых действий будут отменены и система вернётся во «внешне исходное» состояние.

Сильная согласованность (Strong consistency) - после завершения обновления, любой последующий доступ к данным (Процессом А, В или С) вернет обновленное значение.

Изолированность - во время выполнения транзакции параллельные транзакции не должны оказывать влияние на её результат. Изолированность — требование дорогое, поэтому в реальных БД существуют режимы, не полностью изолирующие транзакцию (уровни изолированности Repeatable Read и ниже).

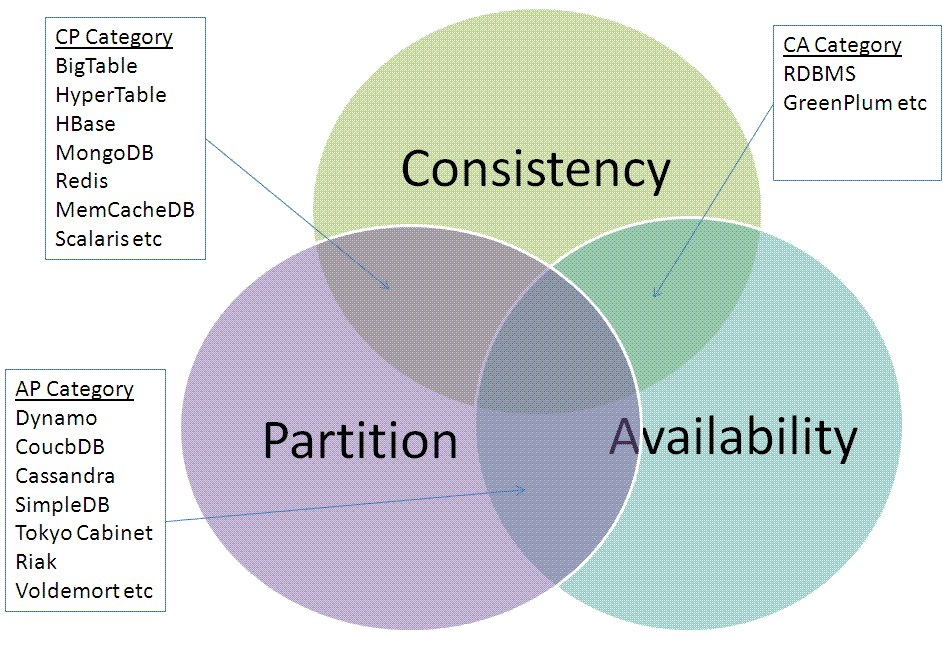
Надежность - независимо от проблем на нижних уровнях (к примеру, обесточивание системы или сбои в оборудовании) изменения, сделанные успешно завершённой транзакцией, должны остаться сохранёнными после возвращения системы в работу. Другими словами, если пользователь получил подтверждение от системы, что транзакция выполнена, он может быть уверен, что сделанные им изменения не будут отменены из-за какого-либо сбоя.

В NoSQL вместо ACID может рассматриваться набор свойств BASE:

* базовая доступность (англ. basic availability) — каждый запрос гарантированно завершается (успешно или безуспешно).
* гибкое состояние (англ. soft state) — состояние системы может изменяться со временем, даже без ввода новых данных, для достижения согласования данных.
* согласованность в конечном счёте (англ. eventual consistency) — данные могут быть некоторое время рассогласованы, но приходят к согласованию через некоторое время.

При сбое в некоторых узлах системы отказ получает только часть приложений, взаимодействующих с вышедшими из строя узлами. В ходе взаимодействия используются протоколы без состояния, что снижает нагрузку на отдельные узлы и позволяет ее перераспределять. Наконец, допустима временная несогласованность данных в разных узлах системы при условии, что информация будет синхронизирована через некоторый обозримый промежуток времени. BASE используется для наиболее общего описания требований к распределенным NoSQL-системам, подпадающих под утверждение теоремы CAP и не удовлетворяющих требованиям ACID.

Термин «BASE» был предложен Эриком Брюером, автором теоремы CAP, согласно которой в распределённых вычислениях можно обеспечить только два из трёх свойств: согласованность данных, доступность или устойчивость к разделению (англ. partition tolerance — расщепление распределённой системы на несколько изолированных секций не приводит к некорректности отклика от каждой из секций).



Разумеется, системы на основе BASE не могут использоваться в любых приложениях: для функционирования биржевых и банковских систем использование транзакций является необходимостью. В то же время, свойства ACID, какими бы желанными они ни были, практически невозможно обеспечить в системах с многомиллионной веб-аудиторией, вроде amazon.com. Таким образом, проектировщики NoSQL-систем жертвуют согласованностью данных ради достижения двух других свойств из теоремы CAP. Некоторые СУБД, например, Riak, позволяют настраивать требуемые характеристики доступности-согласованности даже для отдельных запросов путём задания количества узлов, необходимых для подтверждения успеха транзакции.

Преимущества NoSQL:

* Масштабируемость. Горизонтальное масштабирование существующих традиционных СУБД обычно является трудоемкой, дорогостоящей и эффективной только до определенного уровня задачей. В то же время многие NoSQL-решения проектировались исходя из необходимости масштабироваться горизонтально и делать это «на лету». Поэтому эта процедура обычно проще и прозрачнее в NoSQL, чем в РСУБД.
* Производительность БД на одном узле, а не в кластере также является немаловажным параметром. Для многих задач такие свойства традиционных СУБД, как транзакционность, изолированность изменений, надежность в пределах одного узла или даже сама реляционная модель, не всегда нужны в полном объеме. Поэтому отказ от этих свойств (всех или некоторых) позволяет NoSQL иногда добиваться большей производительности на одном узле, чем традиционным решениям.
* Надежная работа в условиях, когда отказ железа или сетевая недоступность – обычное дело, является одним из свойств многих решений NoSQL. Основной способ ее обеспечения – это репликация. Сама по себе репликация отнюдь не является уникальной особенностью NoSQL, но здесь, как и при масштабировании, важную роль играют эффективность и легкость внесения изменений в существующую инсталляцию. Переход БД к работе в режиме репликации – это простая задача для большинства NoSQL-решений.
* Простота разработки и администрирования – также важный аргумент в пользу NoSQL-технологий. Целый ряд задач, связанных с масштабированием и репликацией, представляющих значительную сложность и требующих обширной специальной экспертизы на традиционных СУБД, у NoSQL занимает считанные минуты. Задачи установки и настройки, само использование NoSQL-решений обычно существенно проще и менее трудоемки, чем в случае с РСУБД. Поэтому NoSQL-системы стали очевидным выбором для многих стартапов, где скорость разработки и внедрения является ключевым фактором. NoSQL-решения не обязательно означают замену и полный отказ от РСУБД. Как обычно, инструмент должен выбираться под задачу, а не наоборот.
* Специализированная модель данных. Реляционная модель не обязательно является самым подходящим способом представления данных для всех задач. При разработке приложений уже давно стало нормой использование специальных «прослоек», отображающих реляционную модель на модель данных приложения, и наоборот. Это увеличивает накладные расходы и усложняет систему в целом. NoSQL предлагает широкий спектр моделей данных и их реализаций, остается лишь выбрать оптимальную для конкретной задачи модель: данные в виде «документов» из наборов полей, записей «ключ–значение», графов и т. д.

С другой стороны, согласованность базы данных явно принесена в жертву эффективности. Модель данных приложения не накладывает никаких ограничений; можно создать неограниченное число экземпляров одного и того же объекта. Например, благодаря автогенерации ключей в Google App Engine все экземпляры будут иметь уникальные ключи, но все остальное будет идентично. Кроме того, не поддерживается возможность каскадного удаления, поэтому если применить тот же подход к хранению отношений типа "один-ко-многим", то возможна ситуация, при которой родительский объект будет удален, а дочерние останутся в базе данных. Разумеется, ничто не мешает вам реализовать собственную схему обеспечения согласованности, но в этом-то и кроется проблема: вам придется делать это самостоятельно (примерно так же, как мы реализовывали остальную функциональность).

Таким образом, работа с нереляционными базами данных требует определенной дисциплины. Если, например, начать создавать различные типы соревнований, некоторые с названиями, некоторые без них, одни - со свойством date, другие – c race\_date, то это приведет к головной боли и для вас самих, и для других разработчиков, которые будут использовать ваш код.

Другими характерными чертами NoSQL-решений являются:

* Возможность разработки базы данных без задания схемы.
* Скорость: даже при небольшом количестве данных конечные пользователи могут оценить снижение времени отклика системы с сотен миллисекунд до миллисекунд.

## **Типы хранилищ данных**

Описание схемы данных в случае использования NoSQL-решений может осуществляться через использование различных структур данных: хеш-таблиц, деревьев и других.

В зависимости от модели данных и подходов к распределённости и репликации можно выделить четыре типа хранилищ: «ключ-значение» (key-value store), документно-ориентированные (document store), хранилища семейств колонок (column database), графовые базы данных (graph database).

### **Хранилище «ключ-значение»**

Хранилища «ключ-значение» является простейшим хранилищем данных, использующим ключ для доступа к значению. Такие хранилища используются для хранения изображений, создания специализированных файловых систем, в качестве кэшей для объектов, а также в системах, спроектированных с прицелом на масштабируемость. Примеры таких хранилищ — Berkeley DB, MemcacheDB, Redis, Riak, Amazon DynamoDB.

Такие БД очень производительны, просты в обращении и легко масштабируются.

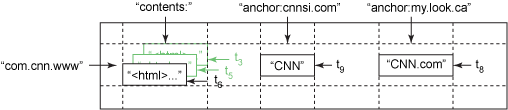
Часто встречающиеся случаи применения:

* Кеширование - быстрое и частое сохранение данных для будущего использования.
* Очередь - некоторые БД типа ключ-значение поддерживают списки, наборы и очереди.
* Распределение информации/задач - используется для реализации паттерна Pub/Sub.
* Живое обновление информации - приложения использующие состояния.

### **Хранилище семейств колонок (или Bigtable-подобные базы данных)**

В этом хранилище данные хранятся в виде разреженной матрицы, строки и столбцы которой используются как ключи. Типичным применением этого вида СУБД является веб-индексирование, а также задачи, связанные с большими данными, с пониженными требованиями к согласованности данных. Примерами СУБД данного типа являются: Apache Hbase, Apache Cassandra, Apache Accumulo , Hypertable, SimpleDB (Amazon.com).

Пример таблицы, в которой URL-адреса используются в качестве ключей строк, а различные аспекты веб-страниц используются в качестве имен столбцов. Содержимое веб-страниц хранится в единственном столбце, который сохраняет несколько версий страницы с метками момента времени, в который они были выбраны.



Хранилища семейств колонок и документно-ориентированные хранилища имеют близкие сценарии использования: системы управления содержимым, блоги, регистрация событий. Использование отметок времени (timestamp) позволяет использовать этот вид хранилища для организации счётчиков, а также регистрации и обработки различных данных, связанных со временем.

Хранилища семейств колонок (англ. column family stores) не следует путать с колоночными хранилищами (англ. column stores). Последние являются реляционными СУБД с раздельным хранением колонок (в отличие от более традиционного построчного хранения данных).

Такие системы баз данных очень эффективны и могут быть использованы для хранения важной информации больших объемов. Может они где то не очень гибки в плане данных, зато они функциональны и производительны.

Основные области применения:

* Хранение неструктурированных, не разрушаемых данных - если вам необходимо хранить большие объемы данных в течение долгого времени, то такие БД очень хорошо справятся с задачей.
* Масштабирование - по задумке такие базы данных легко масштабируются. Они легко справляются с любым объемом данных.

### **Документо-ориентированная СУБД**

Документо-ориентированные СУБД служат для хранения иерархических структур данных. Находят своё применение в системах управления содержимым, издательском деле, документальном поиске и т. п. СУБД данного типа — CouchDB, Couchbase, MarkLogic, MongoDB, eXist, Berkeley DB XML и т.д.

Немного подробней на примере MongoDB:

Основой MongoDB является концепция документа (document), который представляется в виде упорядоченного набора ключей с ассоциированными значениями; коллекция (collection) — это группа таких документов. Если документ является в MongoDB аналогом строки в реляционной базе данных, то коллекция может считаться аналогом таблицы.

Коллекции не имеют схем. Это означает, что документы в рамках одной коллекции могут иметь любое количество различных форм. Например, оба следующих документа могли бы храниться в одной коллекции.

{"empID" : "E12345", "fname" : "John", "lname" : "Smit", "city" : "Sydney", "age": 32}

{"postID" : "P1", "postText" : "This is my blog post"}

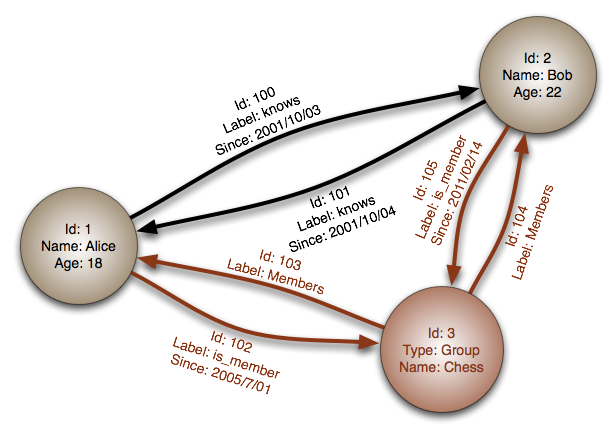
Документ-ориентированные хранилища отлично хранят несвязанную информацию больших объемов, даже если она очень разнится от сущности к сущности.

Преимущества:

* Вложенная информация - документо-ориентированные хранилища отлично работают с глубоко вложенной, сложной информацией.
* Поддержка JavaScript - одна из отличительных особенностей документо-ориентированных хранилищ это то, как они работают с другими приложениями: поддержка JSON.

### **Базы данных на основе графов**

Графовые базы данных применяются для задач, в которых данные имеют большое количество связей, например, социальные сети, выявление мошенничества. Примеры: Neo4j, OrientDB, AllegroGraph, Blazegraph (RDF-хранилище, ранее называлось Bigdata), InfiniteGraph , FlockDB, Titan.



Так как ребра графа материализованы, то есть, являются хранимыми, обход графа не требует дополнительных вычислений (как JOIN в SQL), но для нахождения начальной вершины обхода требуется наличие индексов. Графовые базы данных как правило поддерживают ACID, а также имеют различные языки запросов, вроде Gremlin и Cypher (Neo4j).

Такие типы БД хранят информацию совершенно особенно, совсем не как все остальные СУБД.

Часто встречаемые примеры использования:

* Работа со сложно связанной информацией - как было сказано во вступлении, хранилища типа граф отлично справляются со сложно связанной информацией. Например хранения связей между двумя сущностями и целого ряда разноуровневых связей между сущностями не связанных с первыми напрямую.
* Моделирование и поддержка классификаций - такие БД преуспели везде где есть связи. Моделирование данных и классификация различной информации по связям можно с легкостью представить используя эти БД.

## **UnQL**

В июле 2011 компания Couchbase, разработчик CouchDB, Memcached и Membase, анонсировала создание нового SQL-подобного языка запросов — UnQL (Unstructured Data Query Language). Работы по созданию нового языка выполнили создатель SQLite Ричард Гипп и основатель проекта CouchDB Дэмиен Кац. Разработка передана сообществу на правах общественного достояния.

Этот шаг сделан в сторону стандартизации языка запросов к NoSQL базам данных, на данный момент каждая база данных обладает собственным синтаксисом запросов, например:

CQL(Cassandra):

SELECT \* FROM ad\_click WHERE reseller\_id = 'supaboobs' AND day = '2013-11-29';

MongoDB:

db.users.find( { username: “joe”, age: 27 } );

DQL (DynamoDB):

SELECT \* FROM foobars WHERE foo = 'bar' SAVE out.json;

UnQL имеет SQL-подобный синтаксис и поддерживает такие команды, как SELECT, DELETE, INSERT и UPDATE, что делает новый язык запросов привычным для большинства разработчиков. Тем не менее, в отличие от SQL, UnQL обладает рядом расширенных возможностей, позволяющих манипулировать и выбирать информацию в хранилищах документов со сложной и неоднородной структурой. Для определения представления документов используется формат JSON (JavaScript Object Notation).

Вместо таблиц UnQL манипулирует коллекциями разнородных документов, структура которых жестко не определена и может варьироваться в разных документах (структура документа задается в самом документе, общая схема данных отсутствует). Для создания коллекций по аналогии с SQL-выражением "CREATE/DROP TABLE" используется "CREATE/DROP COLLECTION", при этом коллекция служит для логического разделения различных наборов документов.

Присутствующие в каждом документе поля определяются через JSON, в простейшем случае JSON-представление может состоять из одной строковой или числовой переменной. Каждое из таких полей может фигурировать в блоке "WHERE" запроса SELECT, при этом запрос коснется только документов, в которых определены данные поля. В запросе также могут быть заданы критерии сортировки (ORDER BY), группировки (GROUP BY, HAVING), ограничения размера выборки (LIMIT... OFFSET) и объединения (UNION, INTERSECT, EXCEPT). Поддерживается создание индексов (CREATE INDEX) и использование атомарных транзакций (BEGIN, ROLLBACK, COMMIT). Операции добавления, удаления и изменения данных (INSERT, DELETE, UPDATE) могут выполняться как в синхронном, так и в асинхронном (возвращение управления не дожидаясь фактического выполнения) режимах.

**Немного о будущем**

Согласованность данных - самое большое препятствие на пути NoSQL. Пока что нереляционные БД применяются там, где масштабируемость и производительность важнее строгой согласованности. Реляционные СУБД никуда не денутся, NoSQL не заменяет, а дополняет их. А, может, в будущем мы сможем наблюдать постепенное срастание обеих технологий в гибридную экосистему и поглощения молодых лидеров рынка NoSQL производителями традиционных РСУБД.