Abstract

A NoSQL database provides a mechanism for storage and retrieval of data that uses looser consistency models than traditional relational databases. Motivations for this approach include simplicity of design, horizontal scaling and finer control over availability. NoSQL databases are often highly optimized key–value stores intended for simple retrieval and appending operations, with the goal being significant performance benefits in terms of latency and throughput. NoSQL databases are finding significant and growing industry use in big data and real-time web applications. NoSQL systems are also referred to as "Not only SQL" to emphasize that they do in fact allow SQL-like query languages to be used.

ACID vs BASE NoSQL cannot necessarily give full ACID guarantees. Usually eventual consistency is guaranteed or transactions limited to single data items. This means that given a sufficiently long period of time over which no changes are sent, all updates can be expected to propagate eventually through the system.[citation needed].

Contents

History

Carlo Strozzi used the term NoSQL in 1998 to name his lightweight, open-source relational database that did not expose the standard SQL interface. Strozzi suggests that, as the
current NoSQL movement "departs from the relational model altogether; it should therefore have been called more appropriately 'NoREL'.

Eric Evans (then a Rackspace employee) reintroduced the term NoSQL in early 2009 when Johan Oskarsson of Last.fm wanted to organize an event to discuss open-source distributed databases. The name attempted to label the emergence of a growing number of non-relational, distributed data stores that often did not attempt to provide atomicity, consistency, isolation and durability guarantees that are key attributes of classic relational database systems.

**Taxonomy**

There have been various approaches to classify NoSQL databases, each with different categories and subcategories. Because of the variety of approaches and overlappings regarding the nonfunctional requirements and the feature-set it could be difficult to get and maintain an overview of the nonrelational database market. Nevertheless, the most basic classification that most would agree is one based on the data model. A few of these and their prototypes are:

- **Column**: Hbase, Accumulo
- **Document**: MongoDB, Couchbase
- **Key-value**: Dynamo, Riak, Redis, Cache, Project Voldemort
- **Graph**: Neo4J, Allegro, Virtuoso

**Classification based on data model**

Stephen Yen in his blog post "A yes for NoSQL taxonomy" suggests the following:

<table>
<thead>
<tr>
<th>Term</th>
<th>Matching Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>KV Store</td>
<td>keyspace Flare SchemaFree RAMCloud Oracle NoSQL Database(OnDB)</td>
</tr>
<tr>
<td>KV Store - Eventually Dynamo Voldemort Dynomite SubRecord Mo8onDb DovetailDb</td>
<td></td>
</tr>
</tbody>
</table>
Term | Matching Database
---|---
consistent | TokyoTyrant Lightcloud NMDB Luxio MemcacheDB Actord
KV Store - Ordered | Memcached Repcached Coherence Infinispan EXtremeScale JBossCache Velocity Terracqua
KV Cache | Gigaspaces Coord ApacheRiver
Tuple Store | ZopeDB DB40 Shoal
Object Database | CouchDB Couchbase MongoDB Jackrabbit XML-Databases ThruDB CloudKit Prsevere Riak-Basho Scalaris
Document Store | Bigtable Hbase Cassandra Hypertable KAI OpenNeptune Qbase KDI
Wide Columnar Store | Bigtable Hbase Cassandra Hypertable KAI OpenNeptune Qbase KDI

**Classification based on feature**

Ben Scofield categorized NoSQL databases based on nonfunctional categories ("(il)ities") plus a rating of their feature coverage.\[citation needed\]

<table>
<thead>
<tr>
<th>Data Model</th>
<th>Performance</th>
<th>Scalability</th>
<th>Flexibility</th>
<th>Complexity</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key-Value Stores</td>
<td>high</td>
<td>high</td>
<td>high</td>
<td>none</td>
<td>variable (none)</td>
</tr>
<tr>
<td>Column Store</td>
<td>high</td>
<td>high</td>
<td>moderate</td>
<td>low</td>
<td>minimal</td>
</tr>
<tr>
<td>Document Store</td>
<td>high</td>
<td>variable (high)</td>
<td>high</td>
<td>low</td>
<td>variable (low)</td>
</tr>
<tr>
<td>Graph Database</td>
<td>variable</td>
<td>variable</td>
<td>high</td>
<td>high</td>
<td>graph theory</td>
</tr>
<tr>
<td>Relational Database</td>
<td>variable</td>
<td>variable</td>
<td>low</td>
<td>moderate</td>
<td>relational algebra</td>
</tr>
</tbody>
</table>

It has been suggested that this article be merged into *Comparison of structured storage software*. (Discuss) *Proposed since March 2011.*

**Examples**

**Document store**

Main articles: Document-oriented database and XML database
The central concept of a document store is the notion of a "document". While each document-oriented database implementation differs on the details of this definition, in general, they all assume that documents encapsulate and encode data (or information) in some standard formats or encodings. Encodings in use include XML, YAML, and JSON as well as binary forms like BSON, PDF and Microsoft Office documents (MS Word, Excel, and so on).

Different implementations offer different ways of organizing and/or grouping documents:

- Collections
- Tags
- Non-visible Metadata
- Directory hierarchies

Compared to relational databases, for example, collections could be considered as tables as well as documents could be considered as records. But they are different: every record in a table has the same sequence of fields, while documents in a collection may have fields that are completely different.

Documents are addressed in the database via a unique key that represents that document. One of the other defining characteristics of a document-oriented database is that, beyond the simple key-document (or key–value) lookup that you can use to retrieve a document, the database will offer an API or query language that will allow retrieval of documents based on their contents. Some NoSQL document stores offer an alternative way to retrieve information using MapReduce techniques, in CouchDB the usage of MapReduce is mandatory if you want to retrieve documents based on the contents, this is called "Views" and it's an indexed collection with the results of the MapReduce algorithms.

<table>
<thead>
<tr>
<th>Name</th>
<th>Language</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaseX</td>
<td>Java, XQuery</td>
<td>XML database</td>
</tr>
<tr>
<td>Cloudant</td>
<td>Erlang, Java, Scala, C</td>
<td>JSON store (online service)</td>
</tr>
<tr>
<td>Clusterpoint</td>
<td>C++</td>
<td>XML, geared for Full text search</td>
</tr>
<tr>
<td>Couchbase Server</td>
<td>Erlang, C, C++</td>
<td>Support for JSON and binary</td>
</tr>
<tr>
<td>Name</td>
<td>Language</td>
<td>Notes</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------</td>
<td>-----------------------------------------------------------------------</td>
</tr>
<tr>
<td>Apache CouchDB</td>
<td>Erlang</td>
<td>JSON database</td>
</tr>
<tr>
<td>djondb</td>
<td>C++</td>
<td>JSON, ACID Document Store</td>
</tr>
<tr>
<td>ElasticSearch</td>
<td>Java</td>
<td>JSON, Search Engine</td>
</tr>
<tr>
<td>eXist</td>
<td>Java, XQuery</td>
<td>XML database</td>
</tr>
<tr>
<td>Jackrabbit</td>
<td>Java</td>
<td>Java Content Repository Implementation</td>
</tr>
<tr>
<td>IBM Lotus Notes and</td>
<td>LotusScript, Java,</td>
<td>MultiValue</td>
</tr>
<tr>
<td>Lotus Domino</td>
<td>IBM XPages, others</td>
<td></td>
</tr>
<tr>
<td>MarkLogic Server</td>
<td>XQuery, Java, REST</td>
<td>XML database with support for JSON, text, and binaries</td>
</tr>
<tr>
<td>MongoDB</td>
<td>C++, C#, Go</td>
<td>BSON store (binary format JSON)</td>
</tr>
<tr>
<td>Oracle NoSQL Database</td>
<td>Java, C</td>
<td></td>
</tr>
<tr>
<td>OrientDB</td>
<td>Java</td>
<td>JSON, SQL support</td>
</tr>
<tr>
<td>Sedna</td>
<td>XQuery, C++</td>
<td>XML database</td>
</tr>
<tr>
<td>SimpleDB</td>
<td>Erlang</td>
<td>online service</td>
</tr>
<tr>
<td>OpenLink Virtuoso</td>
<td>C++, C#, Java, SPARQL</td>
<td>middleware and database engine hybrid</td>
</tr>
</tbody>
</table>

**Graph**

This kind of database is designed for data whose relations are well represented as a graph (elements interconnected with an undetermined number of relations between them). The kind of data could be social relations, public transport links, road maps or network topologies, for example.

Main article: Graph database

<table>
<thead>
<tr>
<th>Name</th>
<th>Language</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllegroGraph</td>
<td>SPARQL</td>
<td>RDF GraphStore</td>
</tr>
<tr>
<td>Name</td>
<td>Language</td>
<td>Notes</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------------</td>
<td>--------------------------------------------------------------</td>
</tr>
<tr>
<td>IBM DB2</td>
<td>SPARQL</td>
<td>RDF GraphStore added in DB2 10</td>
</tr>
<tr>
<td>DEX</td>
<td>Java, C++, .NET</td>
<td>High-performance Graph Database</td>
</tr>
<tr>
<td>FlockDB</td>
<td>Scala</td>
<td></td>
</tr>
<tr>
<td>InfiniteGraph</td>
<td>Java</td>
<td>High-performance, scalable, distributed Graph Database</td>
</tr>
<tr>
<td>Neo4j</td>
<td>Java</td>
<td></td>
</tr>
<tr>
<td>OpenLink</td>
<td>C++, C#, Java, SPARQL</td>
<td>middleware and database engine hybrid</td>
</tr>
<tr>
<td>Virtuoso</td>
<td>SPARQL</td>
<td></td>
</tr>
<tr>
<td>OrientDB</td>
<td>Java</td>
<td></td>
</tr>
<tr>
<td>Sones GraphDB</td>
<td>C#</td>
<td></td>
</tr>
<tr>
<td>OWLIM</td>
<td>Java, SPARQL 1.1</td>
<td>RDF graph store with reasoning</td>
</tr>
<tr>
<td>VelocityGraph</td>
<td>C#</td>
<td>Fully Tinkerpop Blueprints compliant. Scalable hybrid Object Database and Graph Database</td>
</tr>
</tbody>
</table>

**Key–value stores**

Key–value stores allow the application to store its data in a schema-less way. The data could be stored in a datatype of a programming language or an object. Because of this, there is no need for a fixed data model. The following types exist:

**KV - eventually consistent**

- Apache Cassandra
- Dynamo
- Hibari
- OpenLink Virtuoso
- Project Voldemort
- Riak

**KV - hierarchical**
- GT.M
- InterSystems Caché

**KV - cache in RAM**

- memcached
- OpenLink Virtuoso
- Oracle Coherence
- Redis
- Tuple space
- Velocity
- IBM WebSphere eXtreme Scale
- JBoss Infinispan

**KV - solid state or rotating disk**

- Aerospike
- BigTable
- CDB
- Couchbase Server
- Keyspace
- LevelDB
- MemcacheDB (using Berkeley DB)
- MongoDB
- OpenLink Virtuoso
- Tarantool
- Tokyo Cabinet
- Tuple space
- Oracle NoSQL Database

**KV - ordered**

- Berkeley DB
- FoundationDB
- IBM Informix C-ISAM
• InfinityDB
• MemcacheDB
• NDBM

Object database

Main article: Object database

• db4o
• GemStone/S
• InterSystems Caché
• JADE
• NeoDatis ODB
• ObjectDB
• Objectivity/DB
• ObjectStore
• ODABA
• OpenLink Virtuoso
• Versant Object Database
• WakandaDB
• ZODB

Tabular

• Apache Accumulo
• BigTable
• Apache Hbase
• Hypertable
• Mnesia
• OpenLink Virtuoso

Tuple store

• Apache River
• OpenLink Virtuoso
• Tarantool

**Triple/Quad Store (RDF) database**

• Meronymy SPARQL Database Server
• Virtuoso Universal Server
• Ontotext-OWLIM
• Apache JENA
• Oracle NoSQL database

**Hosted**

• Freebase
• OpenLink Virtuoso
• Datastore on Google Appengine
• Amazon DynamoDB
• Cloudant Data Layer (CouchDB)

**Multivalue databases**

• Northgate Information Solutions Reality, the original Pick/MV Database
• Extensible Storage Engine (ESE/NT)
• OpenQM
• Revelation Software’s OpenInsight
• Rocket U2
• D3 Pick database
• InterSystems Caché
• InfinityDB

**Cell database**

• Boardwalk

**References**


4. Mike Chapple. "The ACID Model".


8. Sandy (14 January 2011). "Key Value stores and the NoSQL movement". http://dba.stackexchange.com/questions/607/what-is-a-key-value-store-database: Stackexchange. Retrieved 1 January 2012. "Key–value stores allow the application developer to store schema-less data. This data usually consists of a string that represents the key, and the actual data that is considered to be the value in the "key–value" relationship. The data itself is usually some kind of primitive of the programming language (a string, an integer, or an array) or an object that is being marshaled by the programming language's bindings to the key–value store. This structure replaces the need for a fixed data model and allows proper formatting."


11. Tweed, Rob; George James (2010). "A Universal NoSQL Engine, Using a Tried and Tested Technology" (PDF). p. 25. "Without exception, the most successful and well-known of the NoSQL databases have been developed from scratch, all
within just the last few years. Strangely, it seems that nobody looked around to see whether there were any existing, successfully implemented database technologies that could have provided a sound foundation for meeting Web-scale demands. Had they done so, they might have discovered two products, GT.M and Caché....."*

12. JBoss Infinispan

13.
https://www.google.com/patents/US7383272?dq=Method+and+system+for+versioned+sharing,+consolidation+and+reporting+information&hl=en&sa=X&ei=wmUFUuyKBsv0iwLmvoCgBA&ved=0CDQQ6AEwAA