NoSQL Database Types and Use Cases

After completing this reading, you will be able to describe the characteristics of several NoSQL database types, list their use cases, and identify frequently mentioned vendors associated with each NoSQL database type.

As you review each use case, you'll see frequently used NoSQL database vendors mentioned in parentheses following the listed use case. This information is not an endorsement. This information is for market awareness.

Gain the insights you need to be able to explain some of the technical considerations associated the use of MongoDB with a content management system (CMS).

Document store databases

Document-store databases, also known as document-oriented databases, store data in a document format, typically JSON or BSON (binary JSON), where each document contains key-value pairs or key-document pairs. These databases are schema-less, allowing flexibility in data structures within a collection.

Characteristics

- Provides schema flexibility: Documents within collections can have varying structures, allowing for easy updates and accommodation of evolving data requirements.
- Performs efficient create, read, update, and delete (CRUD) operations: well-suited for read and write-intensive applications due to their ability to retrieve whole documents.
- Provides scalability: horizontal scalability by sharding data across clusters.

Use cases

- Content management systems (CMS): CMS platforms like WordPress use document store databases for fast storage and access to content types such as articles, images, and user data. (MongoDB)
- E-commerce: E-commerce platforms need effective management of product catalogs with diverse attributes and hierarchies, accommodating the dynamic nature of e-commerce product listings. (Couchbase or Amazon DocumentDB, using MongoDB compatibility)

Frequently mentioned vendors

- MongoDB
- Couchbase
- Amazon DocumentDB

Key-value stores

Key-value stores are the simplest NoSQL databases, storing data as a collection of key-value pairs, where the key is unique and directly points to its associated value.

Characteristics

- Delivers high performance: efficient for read and write operations, optimized for speedy retrieval based on keys
- Provides scalability: easily scalable due to their simple structure and ability to distribute data across nodes
- Uses caching for fast access
- Provides session management
- Works with distributed systems

Use cases:

- Enhanced web performance by caching frequently accessed data (Using Redis or Memcached)
- E-commerce platforms, software applications, including gaming: Amazon DynamoDB provides a highly scalable key-value store, facilitating distributed systems' seamless operation by handling high traffic and scaling dynamically.

Frequently mentioned vendors

- Redis
- Memcached
- Amazon DynamoDB

Column-family stores

Definition: Column-family stores NoSQL databases, also referred to as columnar databases, organize data in columns rather than rows. These databases store columns of data together, making them efficient for handling large data sets with dynamic schemas.

Characteristics

- Uses column-oriented storage: Data is grouped by columns rather than rows, allowing for efficient retrieval of specific columns.
- Delivers scalability: Distributed architecture for high availability and scalability.

Use cases

- IoT applications manage massive amounts of sensor data efficiently due to their ability to handle time-stamped data at scale, referred to as time-series data analysis. (Apache Cassandra)
- Applications that store and analyze user preferences and behaviors usually deliver personalization. (HBase, part of the Hadoop ecosystem)
- Large-scale data analysis.

Frequently mentioned vendors

- Apache Cassandra
- HBase

Graph databases:

Definition: Graph NoSQL databases are designed to manage highly interconnected data, representing relationships as first-class citizens alongside nodes and properties.

Characteristics:

- Analyzes the data using a graph data model: relationships are as important as the data itself, enabling efficient traversal and querying of complex relationships.
- Fast performance for relationship queries: optimized for queries involving relationships, making them ideal for social networks, recommendation systems, and network analysis.

Use cases:

- Social networks require efficient data management of relationships between users, posts, comments, and likes. (Neo4j)
- Recommendation systems: Organizations need a database structure that can create sophisticated recommendation engines, analyzing complex relationships between users, products, and behaviors for precise recommendations. (Amazon Neptune)

Frequently mentioned vendors

- Neo4j
- Amazon Neptune
- ArangoDB Memcached

Wide-column stores:

Wide-column store NoSQL databases organize data in tables, rows, and columns, like relational databases, but with a flexible schema.

Characteristics:

- Use columnar storage: Data is stored in columns, allowing for efficient retrieval of specific columns rather than entire rows.
- Provide horizontal scalability and fault tolerance.

Use cases:

- Analyzing big data: Efficiently handling large-scale data processing for real-time big data analytics. (Apache HBase used in conjunction with Hadoop)
- Managing enterprise content: Large organizations databases need to manage vast amounts of structured data like employee records or inventory due. (Cassandra)

Frequently mentioned vendors

- Apache HBase
- Apache Cassandra

Expanded use case example: Using MongoDB for a content management system (CMS)

Content management systems (CMS) intelligently collect, govern, manage, and enrich enterprise content, including HTML pages, images, articles, and more. Content management systems help companies deploy their content efficiently and securely across any cloud and within any application.

Good content management means that team members can quickly add, update, and remove content from the database and the associated pages that feature that content. Examples include pushing out breaking news, updating current news, including weather forecasts, pushing advertising content, updating college admission policies, launching new city services, and more.

For example, using MongoDB as a backend database for a content management system (CMS) is a practical choice when you need to manage and serve a variety of content types, especially in scenarios where you expect frequent schema changes or scaling requirements.

Next, let's check out some of the aspects of managing content using a content management system, specifically using MongoDB.

Content structure using MongoDB

In MongoDB, you represent content as documents. Each document corresponds to a piece of content, such as an article, image, video, or page. You can use the subdocuments in the document to organize the content hierarchy and structure.

Example of structuring: Storing a blog post

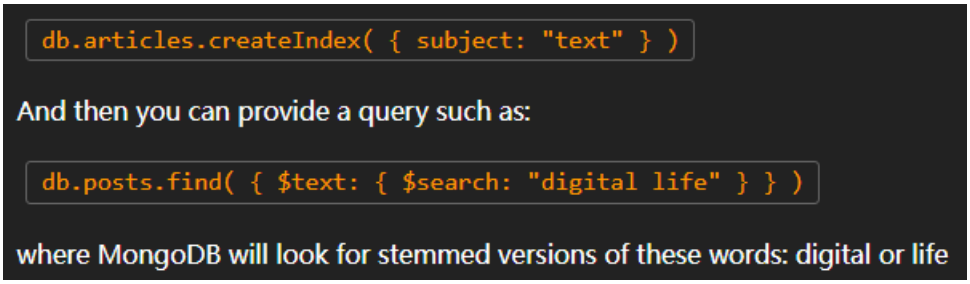
When storing a blog post, you will store core attributes like title, content, created at, and the image URL. Then, using an array field, you can store tags. The comments on that post are stored as an array of objects.

```
// Collection: posts
{
"\_id":1,
"title":"Sample Blog Post",
"content":"This is the content of the blog post...",
"author":{
"name":"John Doe",
"email":"john@example.com",
"bio":"A passionate blogger.",
"created\_at":"2023-09-20T00:00:00Z"
},
"created\_at":"2023-09-20T08:00:00Z",
"tags":["mongodb","blogging","example"],
"comments":[
{
"text":"Great post!",
"author":"Emily Johnson",
"created\_at":"2023-09-20T10:00:00Z"
},
{
"text":"Thanks for sharing!",
"author":"James Martin",
"created\_at":"2023-09-20T11:00:00Z"
}
]
}
```

Metadata and indexing using MongoDB

You can use the indexing capabilities of MongoDB to optimize content retrieval. You can create indexes on fields commonly used for filtering or searching, such as keywords, publication date, or content type, or use MongoDB's text index support for text search queries on fields containing string content. Text indexes improve performance when searching for specific words or phrases within string content.

For example, you want to provide searching capability on the content of your blogs. You will first create a text index:



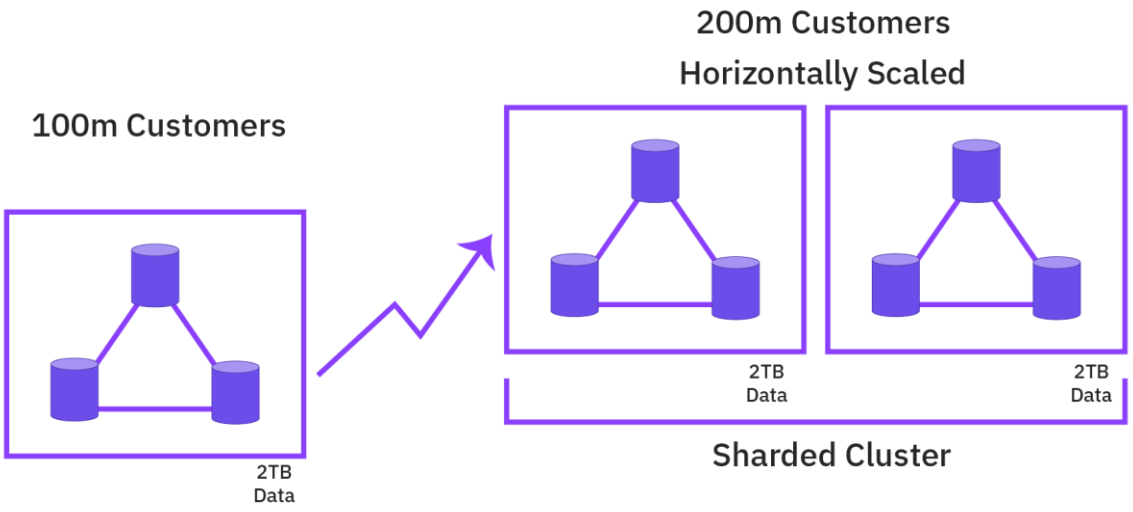
where MongoDB will look for stemmed versions of these words: digital or life

Scaling your CMS using MongoDB

As your CMS grows, MongoDB can help you scale. You can use sharding for horizontal scaling or use zone-based sharding for global distribution.

Using sharding for horizontal scaling (increased capacity)

Let's consider a company that currently has 100 million customers. This company expects to expand its customer base to 200 million customers. This increase in the number of customers means that the company will need to double its IT data storage hardware. The company can scale vertically, which can cost exponentially more as the hardware cost isn't linear with the performance. The following diagram shows that the company can scale horizontally and use sharding to manage the databases.



However, the use of sharding for horizontal scaling provides the company with double the throughput at double the cost.