Ask a Question

Types of Databases

Understand various types of databases and their use cases in system design.

We'll cover the following

As we discussed earlier, databases are divided into two types: relational and non-relational. Let’s discuss these types in detail.

Relational databases

Relational databases adhere to particular schemas before storing the data. The data stored in relational databases has prior structure. Mostly, this model organizes data into one or more relations (also called tables), with a unique key for each tuple (instance). Each entity of the data consists of instances and attributes, where instances are stored in rows, and the attributes of each instance are stored in columns. Since each tuple has a unique key, a tuple in one table can be linked to a tuple in other tables by storing the primary keys in other tables, generally known as foreign keys.

A Structure Query Language (SQL) is used for manipulating the database. This includes insertion, deletion, and retrieval of data.

There are various reasons for the popularity and dominance of relational databases, which include simplicity, robustness, flexibility, performance, scalability, and compatibility in managing generic data.

Relational databases provide the atomicity, consistency, isolation, and durability (ACID) properties to maintain the integrity of the database. ACID is a powerful abstraction that simplifies complex interactions with the data and hides many anomalies (like dirty reads, dirty writes, read skew, lost updates, write skew, and phantom reads) behind a simple transaction abort.

But ACID is like a big hammer by design so that it’s generic enough for all the problems. If some specific application only needs to deal with a few anomalies, there’s a window of opportunity to use a custom solution for higher performance, though there is added complexity.

Let’s discuss ACID in detail:

- **Atomicity:** A transaction is considered an atomic unit. Therefore, either all the statements within a transaction will successfully execute, or none of them will execute. If a statement fails within a transaction, it should be aborted and rolled back.
- **Consistency:** At any given time, the database should be in a consistent state, and it should remain in a consistent state after every transaction. For example, if multiple users want to view a record from the database, it should return a similar result each time.
- **Isolation:** In the case of multiple transactions running concurrently, they shouldn't be affected by each other. The final state of the database should be the same as the transactions were executed sequentially.
- **Durability:** The system should guarantee that completed transactions will survive permanently in the database even in system failure events.

Various database management systems (DBMS) are used to define relational database schema along with other operations, such as to store, retrieve, and run SQL queries on data. Some of the popular DBMS are as follows:

- MySQL
- Oracle Database
- Microsoft SQL Server
- IBM DB2
- Postgres
- SQLite

Why relational databases?

Relational databases are the default choices of software professionals for structured data storage. There are a number of advantages to these databases. One of the greatest powers of the relational database is its abstractions of ACID transactions and related programming semantics. This make it very convenient f the end-programmer to use a relational database. Let’s revisit some important features of relational databases:

Flexibility

In the context of SQL, **data definition language (DDL)** provides us the flexibility to modify the database, including tables, columns, renaming the tables, and other changes. DDL even allows us to modify schema while other queries are happening and the database server is running.

Reduced redundancy

One of the biggest advantages of the relational database is that it eliminates data redundancy. The information related to a specific entity appears in one table while the relevant data to that specific entity appears in the other tables linked through foreign keys. This process is called normalization and has the additional benefit of removing an inconsistent dependency.

Concurrency

Concurrency is an important factor while designing an enterprise database. In such a case, the data is read and written by many users at the same time. We need to coordinate such interactions to avoid inconsistency in data—for example, the double booking of hotel rooms. Concurrency in a relational database is handled through transactional access to the data. As explained earlier, a transaction is considered an atomic operation, so it also works in error handling to either roll back or commit a transaction on successful execution.

Integration

The process of aggregating data from multiple sources is a common practice in enterprise applications. A common way to perform this aggregation is to integrate shared database where multiple applications store their data. This way, all the applications can easily access each other’s data while the concurrency control measures handle the access of multiple applications.

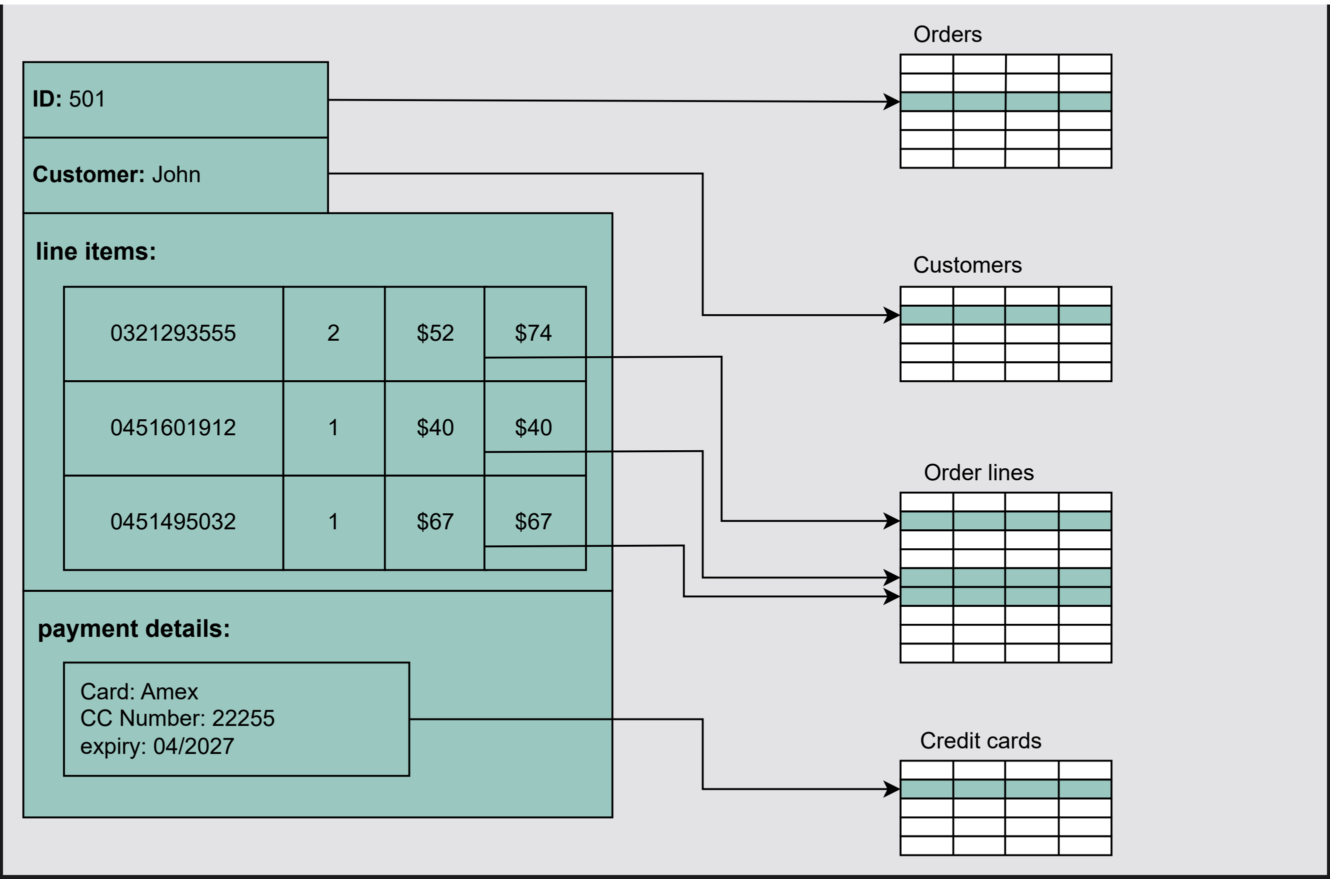
Backup and disaster recovery

Relational databases guarantee the state of data is consistent at any time. The export and import operations make backup and restoration easier. Most cloud-based relational databases perform continuous mirroring to avoid loss of data and make the restoration process easier and quicker.

Drawback

Impedance mismatch

Impedance mismatch is the difference between the relational model and the in-memory data structures. The relational model organizes data into a tabular structure with relations and tuples. SQL operation on this structured data yields relations aligned with relational algebra. However, it has some limitations. In particular, the values in a table take simple values that can’t be a structure or a list. The case is different for in-memory, where a complex data structure can be stored. To make the complex structures compatible with the relations, we would need a translation of the data in light of relational algebra. So, the impedance mismatch requires translation between two representations, as denoted in the following figure:



A single aggregated value in the view is composed of several rows and tables in the relational database

Why non-relational (NoSQL) databases?

A NoSQL database is designed for a variety of data models to access and manage data. There are various types of NoSQL databases, which we'll explain in the next section. These databases are used in applications that require a large volume of semi-structured and unstructured data, low latency, and flexible data models. This can be achieved by relaxing some of the data consistency restrictions of other databases. Following are some characteristics of the NoSQL database:

- **Simple design:** Unlike relational databases, NoSQL doesn't require dealing with the impedance mismatch—for example, storing all the employees' data one document instead of multiple tables that require join operations. This strategy makes it simple and easier to write less code, debug, and maintain.
- **Horizontal scaling:** Primarily, NoSQL is preferred due to its ability to run databases on a large cluster. This solves the problem when the number of

concurrent users increases. NoSQL makes it easier to scale out since the data related to a specific employee is stored in one document instead of multiple tables over nodes. NoSQL databases often spread data across multiple nodes and balance data and queries across nodes automatically. In case of a node failure, it can be transparently replaced without any application disruption.

- **Availability:** To enhance the availability of data, node replacement can be performed without application downtime. Most of the non-relational databases' variants support data replication to ensure high availability and disaster recovery.
- **Support for unstructured and semi-structured data:** Many NoSQL databases work with data that doesn't have schema at the time of database configuration or data writes. For example, document databases are structureless; they allow documents (JSON, XML, BSON, and so on) to have different fields. For example, one JSON document can have fewer fields than the other.
- **Cost:** Licenses for many RDBMSs are pretty expensive, while many NoSQL databases are open source and freely available. Similarly, some RDBMSs rely on costly proprietary hardware and storage systems, while NoSQL databases usually use clusters of cheap commodity servers.

NoSQL databases are divided into various categories based on the nature of the operations and features, including document store, columnar database, key-value store, and graph database. We'll discuss each of them along with their use cases from the system design perspective in the following sections.

Types of NoSQL databases

Various types of NoSQL databases are described below:

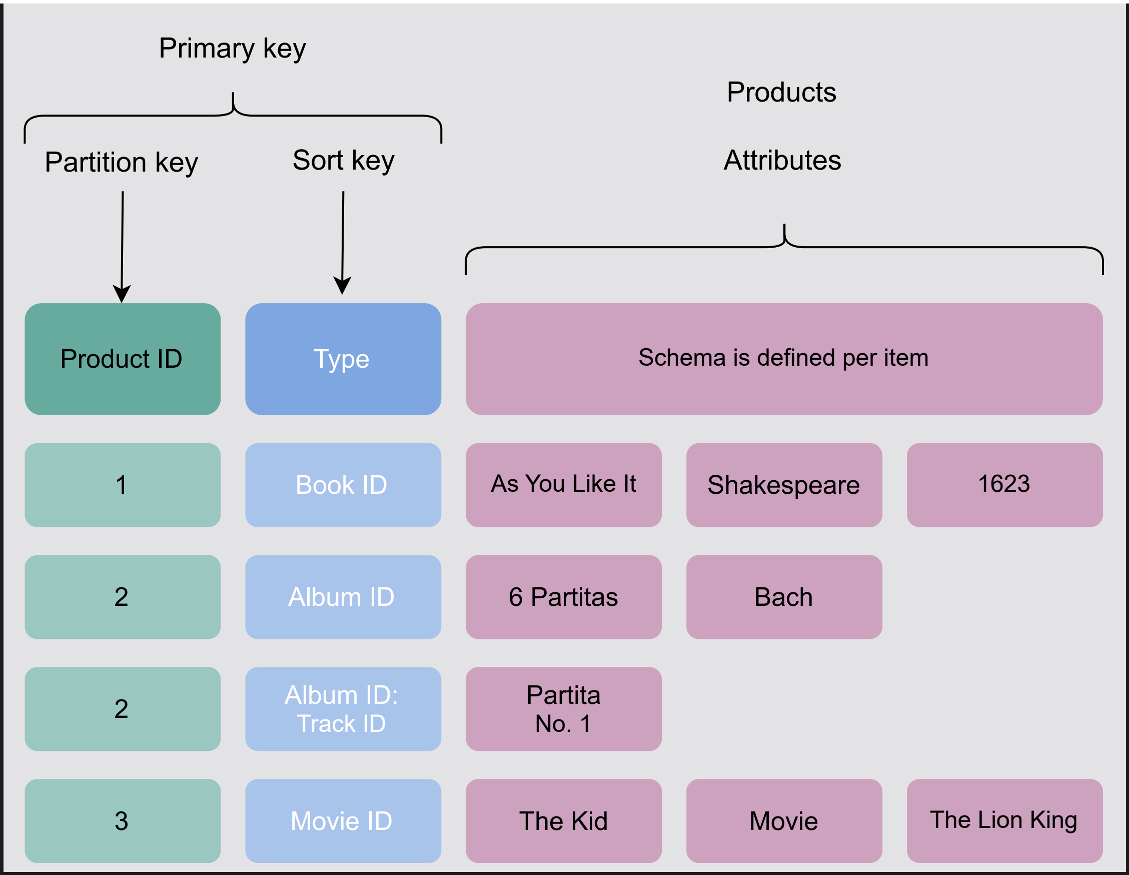
The types of NoSQL databases

Key-value database

Key-value databases use key-value methods like hash tables to store data in key-value pairs. We can see this depicted in the figure a couple of paragraphs below. Here, the key serves as a unique or primary key, and the values can be anything ranging from simple scalar values to complex objects. These databases allow easy partitioning and horizontal scaling of the data. Some popular key-value databases include Amazon DynamoDB, Redis, and Memcached DB.

Use case: Key-value databases are efficient for session-oriented applications. Session oriented-applications, such as web applications, store users' data in the main memory or in a database during a session. This data may include user profile information, recommendations, targeted promotions, discounts, and more. A unique ID (a key) is assigned to each user's session for easy access and storage. Therefore, a better choice to store such data is the key-value database.

The following figure shows an example of a key-value database. The `Product ID` and `Type` of the item are collectively considered as the primary key. This is considered as a key for this key-value database. Moreover, the schema for storin the item attributes is defined based on the nature of the item and the number of attributes it possesses.



Data stored in the form of key-value pair in DynamoDB, where the key is the combination of two attributes (Product ID and Type)

Document database

A **document database** is designed to store and retrieve documents in formats like XML, JSON, BSON, and so on. These documents are composed of a hierarchical tree data structure that can include maps, collections, and scalar values. Documents in this type of database may have varying structures and data. MongoDB and Google Cloud Firestore are examples of document databases.

Use case: Document databases are suitable for unstructured catalog data, like JSON files or other complex structured hierarchical data. For example, in e-commerce applications, a product has thousands of attributes, which is unfeasib to store in a relational database due to its impact on the reading performance. Here comes the role of a document database, which can efficiently store each attribute in a single file for easy management and faster reading speed. Moreover it's also a good option for content management applications, such as blogs and video platforms. An entity required for the application is stored as a single document in such applications.

The following example shows data stored in a JSON document. This data is about a person. Various attributes are stored in the file, including `id`, `name`, `email`, and so on.

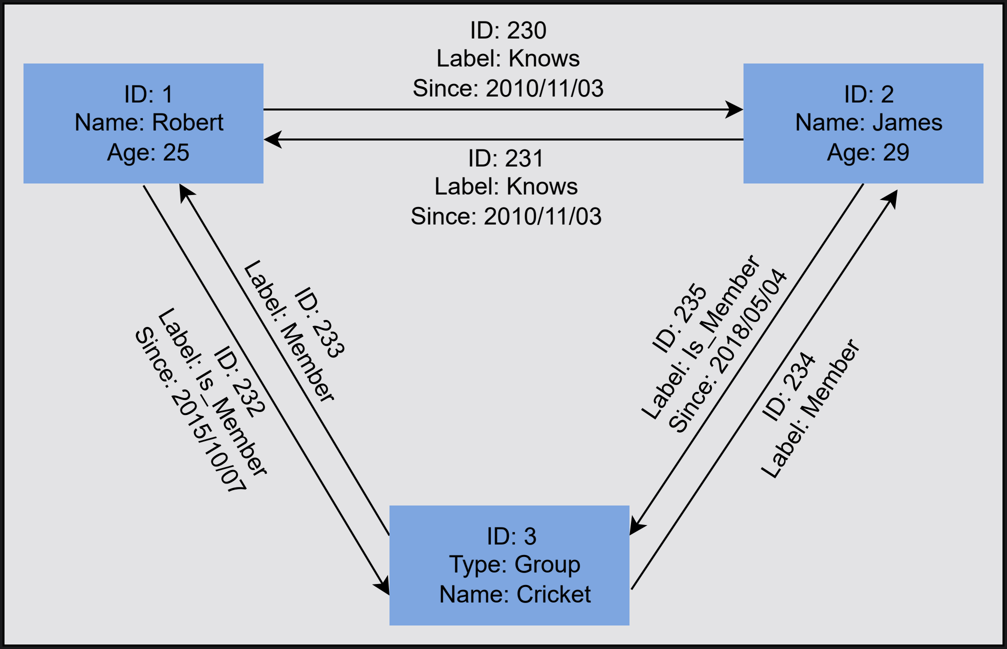
```
{  "id": 1001,
   "name": "Brown",
   "title": "Mr.",
   "email": "brown@anyEmail.com",
   "cell": "123-465-9999",
   "likes": [
      "designing",
      "cycling",
      "skiing"],
   "businesses": [
      { "name": "ABC co.",
         "partner": "Vike",
         "status": "Bankrupt",
         "date_founded": {
            "$date": "2021-12-10" } }]}
```

A JSON file containing data of a businessman

Graph database

Graph databases use the graph data structure to store data, where nodes represent entities, and edges show relationships between entities. The organization of nodes based on relationships leads to interesting patterns between the nodes. This database allows us to store the data once and then interpret it differently based on relationships. Popular graph databases include Neo4J, OrientDB, and InfiniteGraph. Graph data is kept in store files for persistent storage. Each of the files contains data for a specific part of the graph, such as nodes, lin properties, and so on.

In the following figure, some data is stored using a graph data structure in nodes connected to each other via edges representing relationships between nodes. Each node has some properties, like `Name`, `ID`, and `Age`. The node having `ID: 2` has the `Name` of `James` and `Age` of `29` years.



A graph consists of nodes and links. This graph captures entities and their relationships with each other

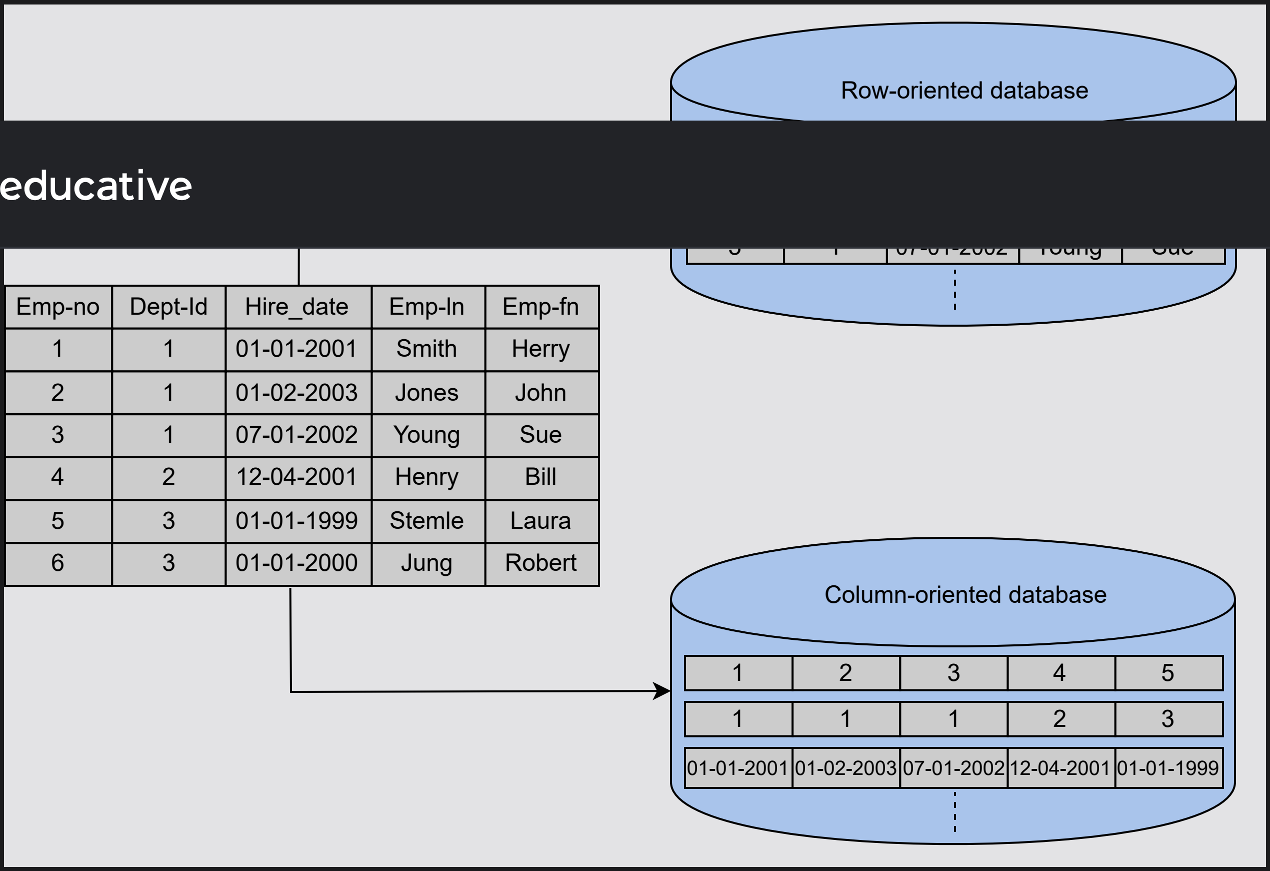
Use case: Graph databases can be used in social applications and provide interesting facts and figures among different kinds of users and their activities. The focus of graph databases is to store data and pave the way to drive analyses and decisions based on relationships between entities. The nature of graph databases makes them suitable for various applications, such as data regulation and privacy, machine learning research, financial services-based applications, and many more.

Columnar database

Columnar databases store data in columns instead of rows. They enable acces to all entries in the database column quickly and efficiently. Popular columnar databases include HBase, Hypertable, and Amazon Redshift.

Use case: Columnar databases are efficient for a large number of aggregation and data analytics queries. It drastically reduces the disk I/O requirements and the amount of data required to load from the disk. For example, in applications related to financial institutions, there's a need to sum the financial transaction over a period of time. Columnar databases make this operation quicker by just reading the column for the amount of money, ignoring other attributes of customers.

The following figure shows an example of a columnar database, where data is stored in a column-oriented format. This is unlike relational databases, which store data in a row-oriented fashion:



Emp-no	Dept-Id	Hire_date	Emp-ln	Emp-fn
1	1	01-01-2001	Smith	Herry
2	1	01-02-2003	Jones	John
3	1	07-01-2002	Young	Sue
4	2	12-04-2001	Henry	Bill
5	3	01-01-1999	Stemle	Laura
6	3	01-01-2000	Jung	Robert

Column-oriented and row-oriented database

Drawbacks of NoSQL databases

Lack of standardization

NoSQL doesn't follow any specific standard, like how relational databases follow relational algebra. Porting applications from one type of NoSQL database to another might be a challenge.

Consistency

NoSQL databases provide different products based on the specific trade-offs between consistency and availability when failures can happen. We won’t have strong data integrity, like primary and referential integrities in a relational database. Data might not be strongly consistent but slowly converging using a weak model like eventual consistency.

Let’s assess our understanding of the lesson’s content with the following question.

Imagine we’re designing a recommendation engine for a social networking platform with millions of users. The goal is to provide personalized recommendations for users to connect with others based on their interests, activities, and connections. Which database would be a natural fit for modeling and querying the complex relationship in a social network and why?

Provide your answer in the following interactive widget:

Want to know the correct answer?

Database for personalized recommendations

Provide your answer here!

Choose the right database

Various factors affect the choice of database to be used in an application. A comparison between the relational and non-relational databases is shown in the following table to help us choose:

Relational and Non-relational Databases

Relational Database	Non-relational Database
If the data to be stored is structured	If the data to be stored is unstructured
If ACID properties are required	If there’s a need to serialize and deserialize data
If the size of the data is relatively small and can fit on a node)	If the size of the data to be stored is large

> **Note**: When NoSQL databases first came into being, they were drastically different to program and use as compared to traditional databases. Though, due to extensive research in academia and industry over the last many years, the programmer-facing differences between NoSQL and traditional stores are blurring. We might be using the same SQL constructs to talk to a NoSQL store and get a similar level of performance and consistency as a traditional store. Google’s Cloud Spanner is one such database that’s geo-replicated with automatic horizontal sharding ability and high-speed global snapshots of data.

Quiz

Test your knowledge of the different types of databases via a quiz.

1 Which database should we use when we have unstructured data and there’s a need for high performance?

A) MongoDB