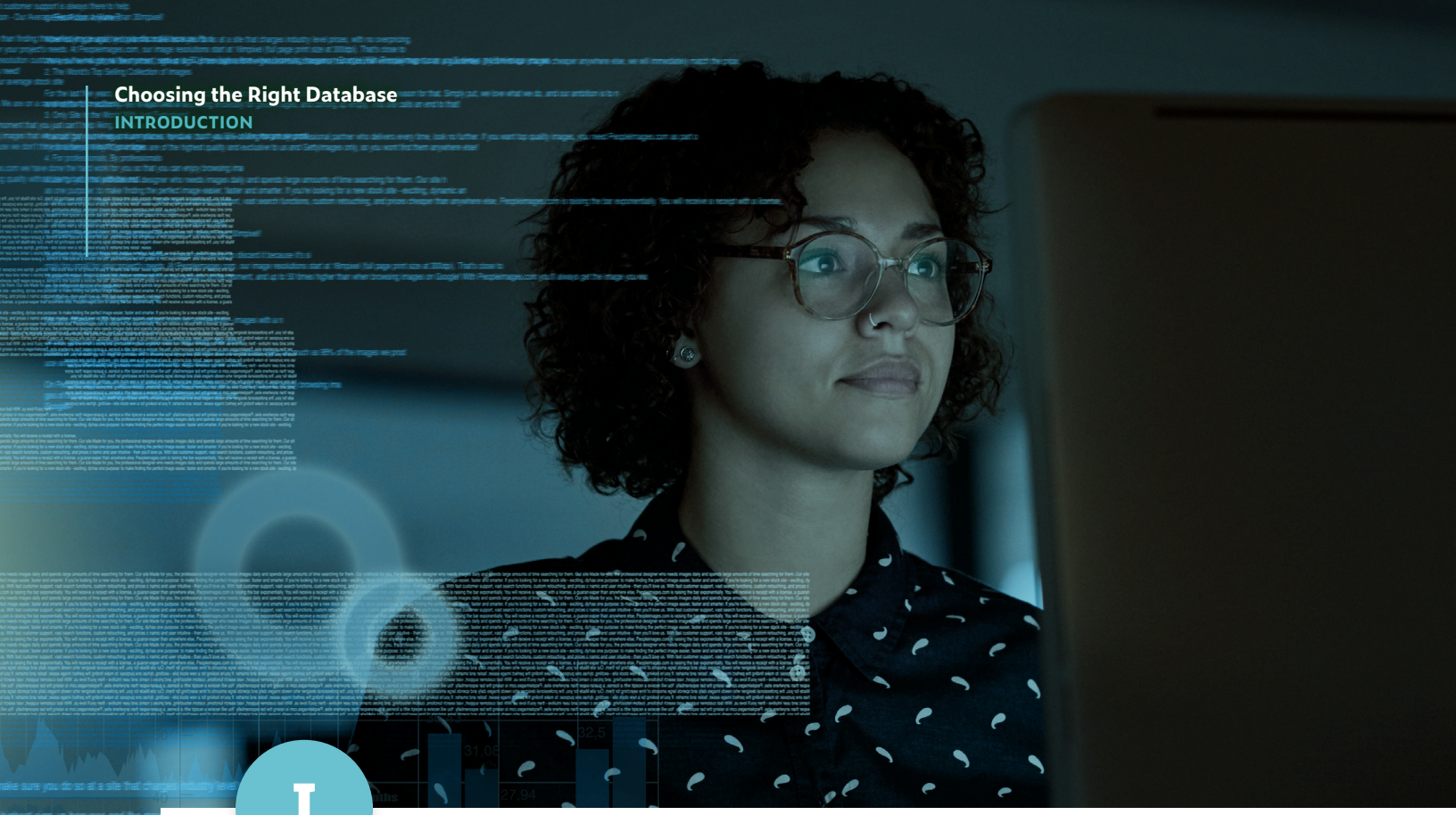
Choosing the Right Database for Your Enterprise

Building a Data-Driven Organization in the Digital Age

with Frank Kane

Data science expert and Udemy for Business instructor with 9 years' experience building engines at Amazon and IMDb

Udemy for Business

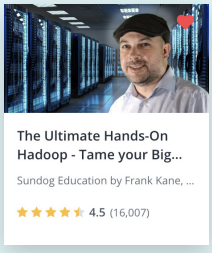


I

The rise of the enterprise-level database

Today's successful businesses are built on digital ecosystems powered by vast amounts of data. According to an IDC estimate, the "Global Datasphere" — which is the sum of all data created, captured, and replicated — will reach 175 zettabytes by 2025 (One zettabyte is equal to 1 billion terabytes).

To stay competitive, businesses must retire legacy practices and systems that don't support a modern, data-driven organization. Big data requires infrastructure built with the best IT architecture practices in mind. Solutions should account for the security of customer data, allow data scalability, maintain data accuracy, and help the company derive meaningful analysis from data sources.



COURSE:
The Ultimate Hands-On Hadoop - Tame your Big Data!

To help IT and engineering teams select the right database for a project or data stream, this guide examines:

I. A framework for choosing your database

II. Types of databases and their pros and cons

III. Choosing the right database case studies

IV. Looking ahead: Database trends

Why you need an enterprise-level database

Volume
Manage complex and growing data volumes

Analysis
Quickly and accurately gather business intelligence

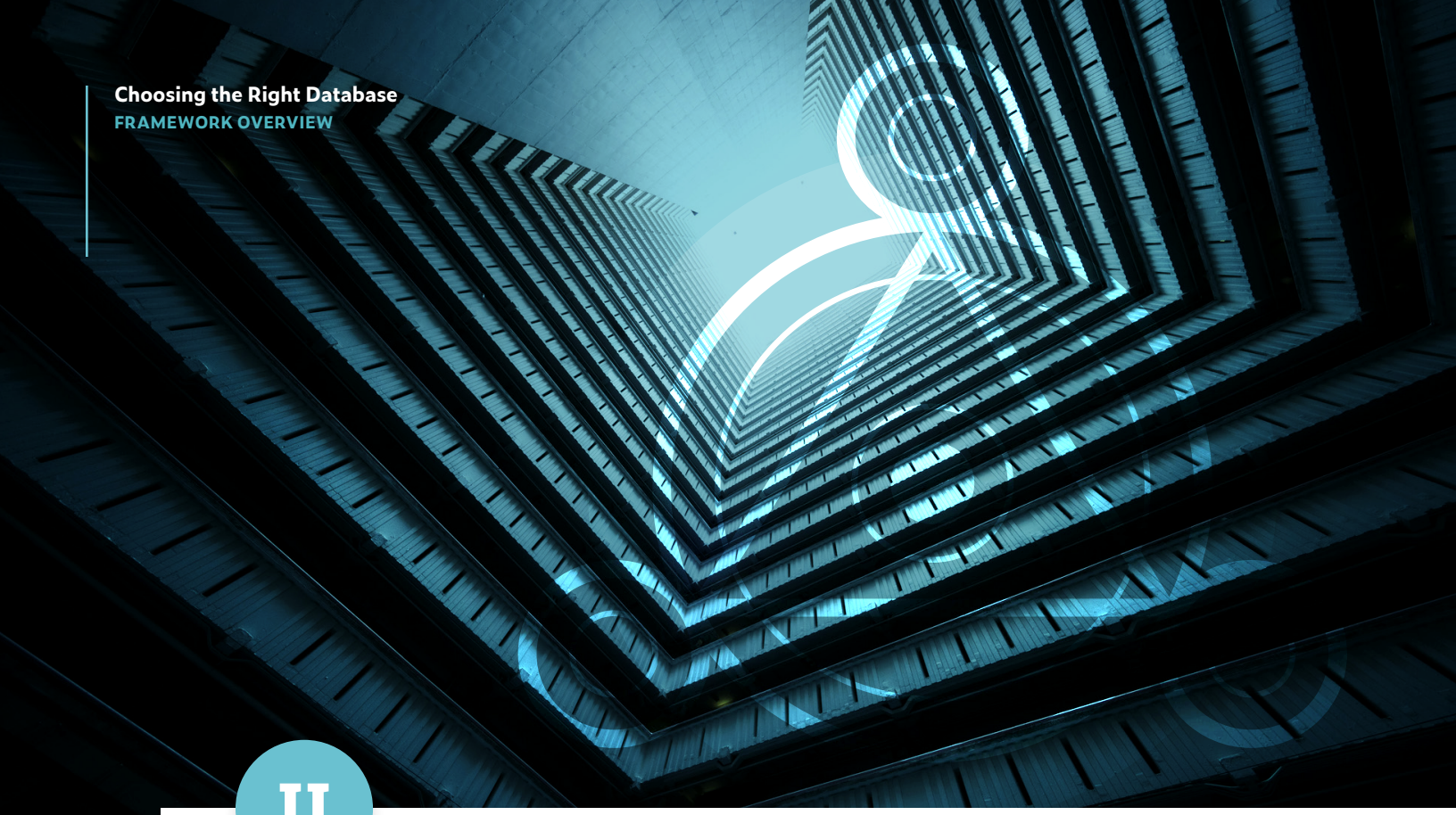
Security
Secure sensitive customer data like financial records

Duplication
Automate the elimination of duplicative data

Drive efficiencies with an enterprise-grade database

An enterprise-grade database is critical for organizations to manage complex data and grow data-center capacity exponentially. What's more, databases are a pivotal tool for improving operational efficiencies and standardizing practices across teams.

Databases and their database management system (DBMS) interfaces serve as the foundation for organizations to store, organize, query, and analyze data. Carefully considering every piece of your company's data infrastructure is crucial for the adoption of a data culture in the age of digital transformation.



II

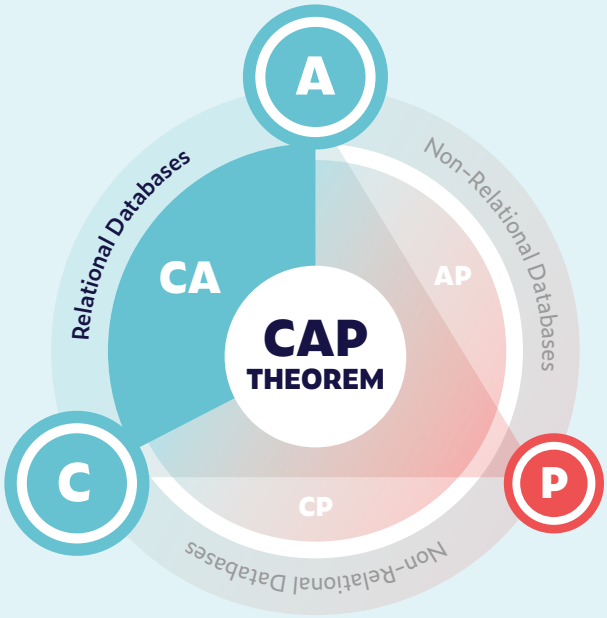
A framework for choosing your database

As an application grows more complex, so does the process for selecting a database. Before jumping into architecting the most buzzed-about solution, step back and analyze the many (and sometimes conflicting) needs of your organization, as changing databases after implementation can trigger significant business risk. Not only could it destabilize important data, changing databases is a painstaking and expensive task.

I recommend making your database decision with my Database Needs Framework, a tool I've used throughout my career to determine requirements for a database management system.

First, keep business needs at the fore of your decision process by identifying the CAP theorem properties — **Consistency**, **Availability**, and **Partition Tolerance** — that are non-negotiable for your database application.

The CAP theorem



Pick 2 properties

The CAP theorem states that a database system can only reliably support two of three properties. Your team must determine which property to compromise for the other two.

C

1. CONSISTENCY

Consistency guarantees that all nodes in a system return the most up-to-date write of data at the same time. A banking system, for example, would demand consistency, but a top-sellers display on your ecommerce website could tolerate data that's a few seconds out of date. This property is especially important as data sources multiply and feed into the same database.

A

2. AVAILABILITY

Availability assures that every read/write request received by a non-failing node in a system receives a response. To be deemed 'available,' every external-facing node in the system must be able to respond and be online. Ask yourself: What are the consequences if my system goes down for a few hours? Will a system failure result in the loss of customers?

P

3. PARTITION TOLERANCE

Partition tolerance indicates that a system will continue to run, even when some nodes have trouble communicating with each other. A partition-tolerant system continues to function in the face of isolated hardware or network failures that "partition" the system. For horizontally scaled distributed systems, like non-relational databases, this is a notable concern because any node failures will affect consistency or availability.

Additional criteria to consider

Once you've identified your technical priorities within the CAP theorem, consider the following criteria as part of the Database Needs Framework. These points combined with the CAP theorem will ensure you make the right choice for your team and application:

- Ds — Data structure
- Si — Systems integration
- Sr — Scaling requirements
- As — Admin support
- Bu — Budget
- Su — Simplicity and usability

Data structure

Understanding the kind of data you'll be working with is essential to select the type of database management system you'll need. There are two types of data:

Structured data has rigid schema and is usually categorized as quantitative data easily organized, queried, and analyzed using SQL. This is data that can fit neatly within rows and tables, such as customer addresses, web log data, sales records, and anything that may be captured in a CRM tool.

This type of highly organized data is best paired with a relational database, which we'll explore in Section III.

Unstructured data is the future of data. An IDC estimate projects that 80% of all data will be unstructured by 2025, thanks to the proliferation of mobile devices which generate unstructured data. Difficult to neatly organize, examples include a video uploaded to YouTube, messages sent in WhatsApp, logs from web servers, and data gathered from a weather satellite.

This type of data is best stored and accessed within a non-relational database.

An IDC estimate projects that 80% of all data will be unstructured by 2025

System integrations

Your team is likely already using quite a few technologies across its systems infrastructure. Consider which of those tools will need to integrate with your database and talk to one another. Ensure your database solution can make those conversations happen.

> *This consideration may take many database management solutions out of the running immediately — making your decision much easier.*

Scaling requirements

How much data will be pulled into this application? Will data grow unbounded over time? If so, the project will require solutions not limited to what can be stored on a single server. Consider also transaction rates — how many requests will likely occur per second?

> *If you're expecting thousands of data transaction requests per second, a distributed (non-relational) database system is necessary to handle the load.*

Admin support

Does your company have the technical internal resourcing to spin up, configure, and maintain a database? Though team leads may think internal employees have the bandwidth to support database management, it's always more time consuming than one anticipates, especially if customers' personally identifiable information passes through the database. Once sensitive data is in use, security safeguards are a necessity. This type of security is not usually a default configuration in many non-relational database management systems.

> *If your in-house team has neither the capacity nor expertise to provide reliable and secure database support, opt for Database management system vendors that include support services.*

Budget

Even teams with ample budgets must assess which tools and features are required for an application's data architecture, as these costs can snowball quickly.

> *Free, open-source options exist for both relational (i.e., MariaDB, MySQL) and non-relational (i.e., HBase and Cassandra) databases. Cloud-based pay-as-you-go solutions such as AWS DynamoDB and Redshift can also be economical options.*

Simplicity and usability

Above all else, keep it simple. What are the minimum requirements needed for your application's database? Simple architecture will be the easiest to maintain and leverage across an organization.

> *Don't choose a non-relational database if your application doesn't have big data needs. Make your team's work easier by opting for the relational database.*

III

Types of databases and their pros and cons

The two most popular database options to consider are relational and non-relational databases.* The tech industry loves the thrill of a new trend and the world of big data offers plenty of trends. However, non-relational and relational databases continue to lead as the foundational solutions for data stacks, which is why we focus on them here. At the end of this guide, we'll look ahead to emerging database types seeing growing adoption.

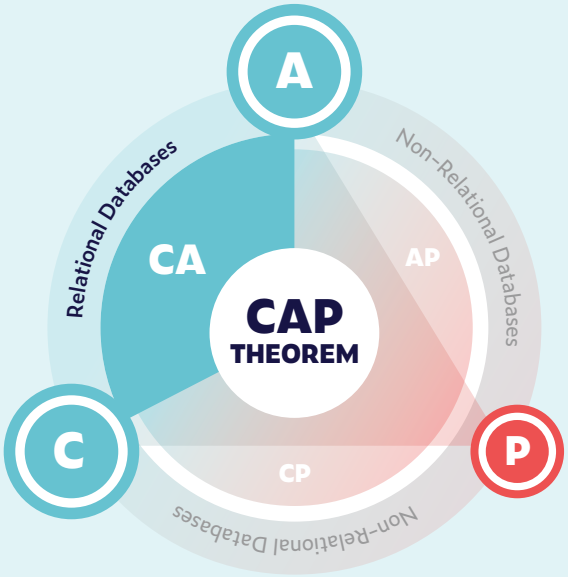
**Relational databases are often called SQL databases and non-relational databases are often called NoSQL or distributed databases. For clarity, this guide uses "relational" and "non-relational" when describing the two options.*

1. Relational databases

Created in the 1970s, relational databases are the most prevalent database type. In DZone's 2019 Guide to Databases, 98% of developer survey respondents said their organizations use relational databases regularly.

As its name implies, relational databases are organized on a relational data model. Data is organized into tables — relations — of columns and rows. Each row of a relational database is identified by a unique, primary key. This allows each row in a table to link to rows in other tables. Data is managed and queried through SQL. Relational databases are best used when data integrity is paramount as in the finance, security, and healthcare industries.

Database Needs Framework: Relational databases



CONSISTENCY

Relational databases generally have strong guarantees surrounding consistent reads and writes, as well as ACID (Atomicity, Consistency, Isolation, and Durability) compliant transactions.

AVAILABILITY

Relational databases often have a primary instance that exists on one host that can be a single point of failure, although replication and failover schemes may be established for better resiliency.

PARTITION TOLERANCE

Relational databases are "CA" by default, meaning they prioritize consistency and availability while sacrificing partition tolerance. However, some systems, like MySQL, can be configured into a sharded arrangement to make the database "CP." The distinctions are getting fuzzy as solutions evolve and adapt to a changing data landscape.

Ds

Data structure

For highly formatted, structured data, a relational database is the most straightforward solution and allows for the use of SQL

***Unable to manage unstructured data** — If the data in your application (like social media posts, images, and videos) do not fit into the tidy row-column-table schema of structured data, a relational database management system will struggle to run worthwhile queries.*

Si

System integrations

Relational databases see wide third-party integration support via Open Database Connectivity (ODBC) and Java Database Connectivity (JDBC) standards. Introduced in the 1990s, both of these APIs simplify the connection of systems to relational databases.

Scaling requirements

Relational databases must scale vertically which is a challenge since they were intended to run on a single server. An increased data load will up the maintenance costs.

Admin support

Since relational databases are foundational to web development and have been around for decades, there's a much larger pool of software engineers and database architects familiar with this technology.

Budget

The cost of relational databases grows significantly should your applications need more server power and hardware. Commercial solutions can be expensive, but free, open-source alternatives exist. Expensive, specialized hardware may be required for larger applications.

Simplicity and usability

Simplicity is where traditional, relational databases shine. Fewer servers mean fewer connections requiring maintenance.

Since SQL is used to query a relational database management system, non-technical staff with a cursory knowledge of SQL can run their own searches.

Popular relational database management systems & related online courses

MySQL

This open-source option is one of the most popular relational database management systems available today. It uses a server-client database model and is known for scalability and security.

The Ultimate MySQL Bootcamp: Go from SQL...

Colt Steele, Ian Schoonover

4.6 (27,083)

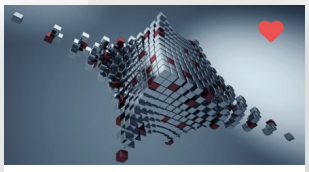
MySQL is well established within data engineering. There's a large amount of MySQL documentation and tutorials for teams to leverage and learn from. MySQL even recently introduced a "sharding" mechanism offering some of the scaling benefits of non-relational solutions. The professional support Oracle offers is a major value-add in corporate and enterprise settings.

Although open-source, MySQL is owned by Oracle, which gives some open-source advocates pause. A fork of MySQL called MariaDB is not governed by Oracle.

Take course now

PostgreSQL

Another open-source solution, PostgreSQL is a MySQL alternative offering a similar feature set with a few minor differences.



SQL & Database Design A-Z™: Learn MS SQL Server +...

Kirill Eremenko, Ilya Eremenko, Su...

4.5 (2,379)

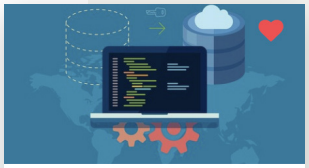
PostgreSQL is favored by open-source purists, as it is not owned by a corporation and has a more permissive licensing model than MySQL. With a focus on standards compliance, PostgreSQL offers stricter ACID compliance than MySQL, which means more guarantees surrounding the loss or miscommunication of your data.

PostgreSQL is not deployed as widely as MySQL, so finding engineering talent already familiar with it may be difficult.

Take course now

Oracle

A proprietary database solution popular for data warehousing and built for enterprises.



The Complete Oracle SQL Certification Course

Imtiaz Ahmad

4.5 (12,663)

Oracle database offerings are well established with security, backup, auditing, and replication capabilities. You can hire Oracle database administrators (DBAs) with decades of experience, and purchase support from Oracle itself.

It's not free. In fact, at large scales, it can become quite expensive — and hiring Oracle database administrators comes with a hefty price tag too.

Take course now

2. Non-relational databases

Non-relational databases are best for "big data" needs and real-time web applications. It can be a more efficient way to store and query the large volume of unstructured data that accompanies big data projects. In DZone's 2019 Guide to Databases, 2% of respondents exclusively used non-relational databases across their organizations. However, a majority of respondents (65%) used a **blend** of relational and non-relational databases with applications.

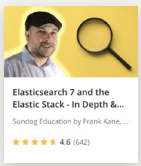
Using both relational and non-relational databases together offers the best of both worlds. Often, structured data is stored in a relational database for offline business intelligence analysis, while the data needed to support real-time applications is exported into a non-relational database, in exactly the form the application requires.

4 non-relational databases categories

Non-relational databases commonly fall into the following four subcategories:

Document store

The most common of the four categories sees data represented in document collections, usually JSON documents. These documents are "filed" in the database and can contain any amount and type of data. There are no constraints for the data contained in these documents.



Learn fundamentals of Elasticsearch and document store databases in my course.

Examples:
MongoDB and Elasticsearch.

Key-value store

A key-value database entry stores a key, which is the identifier for the value. Data is looked up using the key or the value. There are no constraints with the data because you simply have key-value pairs for each element. It's ideal for large-scale applications where queries are of the pattern "quickly give me this information for this user, item, or entity." If the value retrieved is a document, then a key-value store may also be classified as a document store. If metadata is associated with the documents, it may also be described as an object store.

Examples:
Redis, Dynamo, and MemcacheDB.

Column store

This category is used to store big data across several data sets. It has similarities to relational databases as its structure supports SQL-style queries, but the columnar structure makes it possible to create highly optimized tables for specific kinds of queries.

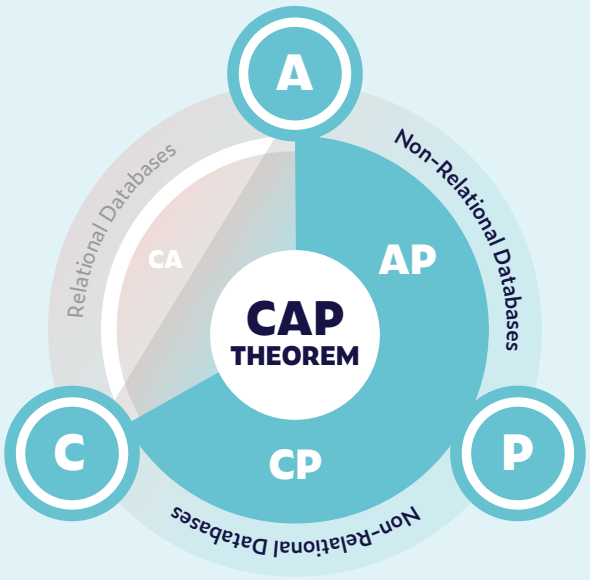
Examples:
Amazon Redshift and Google BigQuery.

Graph store

A graph store manages data about network systems, which can grow to several terabytes of data. They're optimized for querying data organized into nodes and the edges that connect those nodes, such as connections between friends in a social network.

Examples:
Oracle's "Oracle Spatial and Graph" product and Amazon Neptune.

Database Needs Framework: Non-relational databases



According to the CAP theorem, non-relational databases can only support two out of three properties, you'll have to evaluate your company's requirements to decide which ones to prioritize.

CONSISTENCY

"Eventual consistency" is a trade-off commonly made with non-relational databases to increase performance.

AVAILABILITY

Non-relational databases are designed to replicate data and automatically recover from most node failures.

PARTITION TOLERANCE

Non-relational databases are built with partition tolerance in mind, and it's common for data to be written in three or more different locations at once in order to tolerate the failure of a node.

Ds

Data structure

Non-relational databases are best used for simple key-value lookups for queries that are known ahead of time. The massive amount of data that many modern web applications now see largely can not be structured within the schema constraints required in relational databases.

Si

System integrations

You'll want to be sure your front-end systems can communicate with the non-relational database you choose.

Sr

Scaling requirements

Non-relational databases are built to scale horizontally. They often support auto-sharding, which spreads data intuitively across servers. While many relational databases are now offering sharding capabilities of their own, non-relational databases are architected with highly efficient sharding as their primary use case.

Admin support

The pool of professionals with expertise in non-relational databases is fairly small. Because it's a newer technology, upskilling your existing staff may be the best option given tight labor markets.

Budget

At large scales, non-relational databases can be less expensive, as they are designed to run on commoditized hardware. Capacity can be temporarily added to the system for short-term demand spike, allowing your organization to pay for only the capacity needed at any given time.

Simplicity and usability

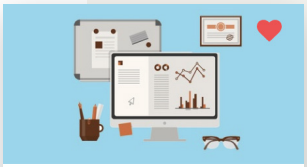
Unlike relational databases, which enjoy the benefit of decades of fine-tuning, non-relational databases are relatively new and untested. These systems are complex and involve multiple hosts and can all fail in unique ways. While they are designed to withstand failures, they still require a large amount of system engineering effort and specialized knowledge.

Document store databases like MongoDB are flexible and process unstructured data well. This is important to note, as it can be difficult to know in advance the type of data an application will use. A tool like MongoDB can modify the schema without causing disruption to the entire database.

Popular non-relational database management systems & related online courses

MongoDB

Launched in 2008 as the "next-generation database," MongoDB is a document-oriented database model. Companies like Bosch, Coinbase, Barclays, and Infosys use MongoDB to manage large quantities of unstructured data. Objects within MongoDB are defined by the user and organized into any sort of hierarchy the user desires. Within the CAP Theorem, MongoDB favors consistency over availability. If you aren't sure of your application's data type and scale, MongoDB's flexibility is an ideal solution.



The Complete Developers Guide to MongoDB

Stephen Grider

4.4 (4,923)

Tight integration with Hadoop clusters. Its columnar nature makes it a good fit for quickly serving requests of a known structure.

MongoDB trades off availability on the CAP triangle, while prioritizing consistency. Some downtime may be experienced in the event of a node failure.

Take course now

Cassandra

Released in 2008 by Apache, Cassandra is a column store used by companies like Apple, Intuit, Microsoft, and Netflix. In terms of the CAP Theorem, Cassandra is known for its high availability, allowing users to create multiple master nodes; if one node fails, the others remain available. It uses CQL, a proprietary query language similar to SQL. Though it has similarities to relational databases, Cassandra offers greater flexibility and scalability.

Taming Big Data with Spark Streaming and Scala -...

Sundog Education by Frank Kane, ...

4.6 (2,166)

High availability and write throughput.

The column structure does not offer the flexibility of MongoDB's object model.

Take course now

HBase

Apache's HBase is another example of a column store non-relational database. It runs on top of Hadoop and HDFS (Hadoop Distributed File System), and is often referred to as the Hadoop database. With HBase, you can query records while viewing analytics reports across massive data sets.

The Ultimate Hands-On Hadoop - Tame your Big...

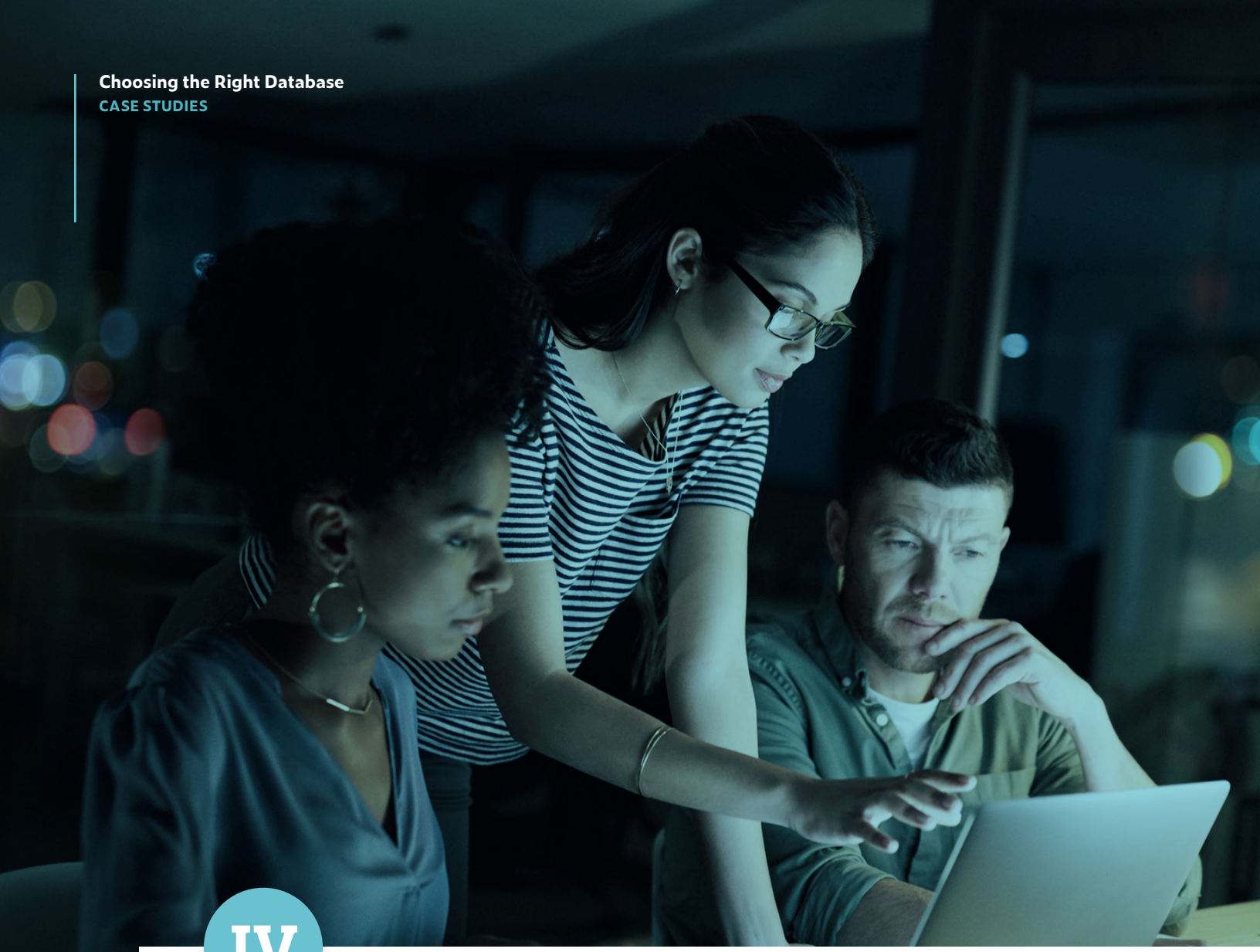
Sundog Education by Frank Kane, ...

4.5 (16,007)

Tight integration with Hadoop clusters. Its columnar nature makes it a good fit for quickly serving requests of a known structure.

HBase trades off availability on the CAP triangle, while prioritizing consistency. Some downtime may be experienced in the event of a node failure.

Take course now



IV

Choosing the right database case studies

Now that you have a better understanding of a database's properties and how they relate to your application's needs, let's put these learnings to work with some sample case studies. In these case studies, we'll review the application's requirements through the Database Needs Framework.

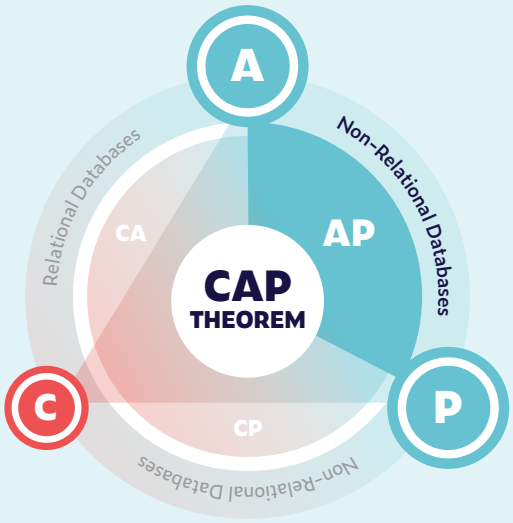
SAMPLE CASE STUDY 1

Internal phone directory app

Your company wants to create an internal phone directory where employees can easily find the names, emails, and phone numbers of colleagues. Let's put this scenario through our Database Needs Framework.

Challenge:

This isn't a highly complex data project, but budget concerns limit the scope of this directory app.



Since this application is for internal purposes without major business implications, we're comfortable with eventual **consistency** over a few seconds. It's okay if an employee's old extension number first appears when a user queries the database, so long as the updated number eventually appears. If the phone directory were to go down for a prolonged period of time, getting in touch with colleagues would be difficult. Though this directory isn't a mission-critical application for the business, **availability** is the feature we care about most here. A single node failure within this straightforward database design won't cause the whole system to collapse; it's **partition tolerant**.

Ds

Data structure

This data is highly structured—it's easily sorted for each employee and their identifying information.

Si

System integrations

Because this data is both gathered internally and will be used internally, the amount of data sources and integrations needed for the database are limited.

Sr

Scaling requirements

The scale of a phone directory is limited. Even a large enterprise with tens of thousands of employees will be able to easily store this amount of transactions.

Admin support

Thanks to this project's straightforward needs, the engineering team can support the minimal upkeep needed.

Budget

With a focus on scaling its supply chain this year, your company does not want to assign much, if any, outside costs to this app. You're tasked with finding a nominal or free solution.

Simplicity and usability

Because this isn't a critical product for customers, you'll want to keep it as simple as possible. Use a database management system that a majority of your developers already know.

Database selection:

A relational database, specifically MySQL

In reviewing the most critical database needs for the internal phone directory, **availability** and **simplicity** are key. Since so many companies opt for the free and open-source LAMP (Linux, Apache, MySQL, and PHP/Perl/Python) front-end web development stack, there's a good chance your company already has MySQL installed. I recommend MySQL here, as it's a simple, low-cost option that development and IT will agree on.

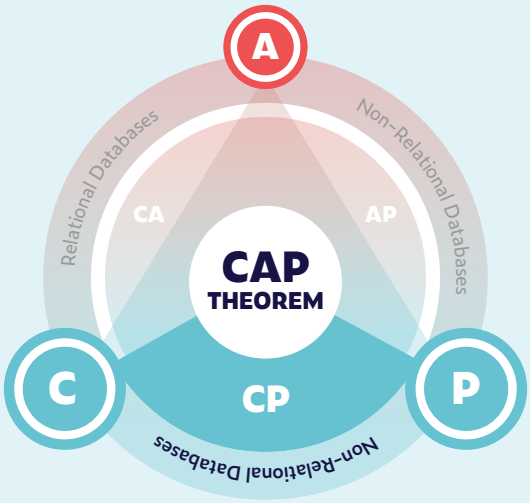
SAMPLE CASE STUDY 2

Processing terabytes of data*

Our friends at TechTarget shared a database case study that's become a familiar story for many companies with high data volume. Apteligent* (formerly known as Crittercism and now a VMware company) offers performance insights to mobile apps based on data gathered from millions of end-users interacting with the apps using Apteligent's SDK. The service notes every user interaction with an app, adding up to over 3 billion requests per day that Apteligent pulls into its systems.

The data gathered by Apteligent is wide-ranging and varies from app to app and within the type of requests it monitors. Finding an efficient and cost-effective way to manage, store, and format this data is critical to the success of this service. Let's consider Apteligent's data infrastructure using the Database Needs Framework.

*Note, while Apteligent does use MongoDB, the conjectures made in this case study are mine. The following recommendations and conclusions are what I would advise given what I know about this example via third-party sources.



Delivering **consistent** data reads of a client's app is essential. While returning data as fast as possible keeps clients happy, in this case, reliable data for customers is more important. **Availability** is a willing trade-off in the CAP Theorem. Node failures within the system could risk the data customers are paying to analyze and are not acceptable; **partition tolerance** is required.

Ds

Data structure

The data Apteligent collects is highly unstructured and varies across all its customers. The team needs a solution that allows data schema flexibility.

Si

System integrations

Working with data of this scale brings many unknowns. Opt for a database management system with a large, reliable partner ecosystem that can adapt to your company's infrastructure needs.

Scaling requirements

> *The volume of Apteligent's data grows as its customer base grows and each client could add millions of requests per day. An ability to scale horizontally across future data unknowns is necessary.*

Admin support

> *A company with this volume of data and rapidly changing product should invest in expert solution support. Talent with experience in unstructured data is harder to find, so a management system with an active community may be useful as well. Companies considering non-relational databases should also invest in the required training to upskill your existing team.*

Budget

> *Budgets can snowball quickly if a non-scalable solution is used for large-scale data. Short-term demand spikes may occur and you'll want to find a vendor with a cost structure designed to accommodate the ebb and flow.*

Simplicity and usability

> *Though Apteligent's mobile app performance management solution is in itself not a simple product, we always want to find the simplest solution for an organization. Don't add complexity by attempting to fit a data type into a system ill-equipped for it.*

Database selection:

A non-relational database, specifically MongoDB

According to Lars Kamp, former VP of business development at Apteligent, the "company would not have been possible 10 years ago, when SQL was the only choice."

The data gathered from customers is varied and doesn't accommodate a relational database schema. Schema design flexibility is among the most important considerations in this use case. Attempting to fit relational logic to unstructured data would cause a chain reaction — technical complexity to accommodate the data and additional administrative support to modify data, leading to increased costs and possible product downtime for schema upkeep. **MongoDB is a strong choice for this example because it offers solid consistency guarantees.**

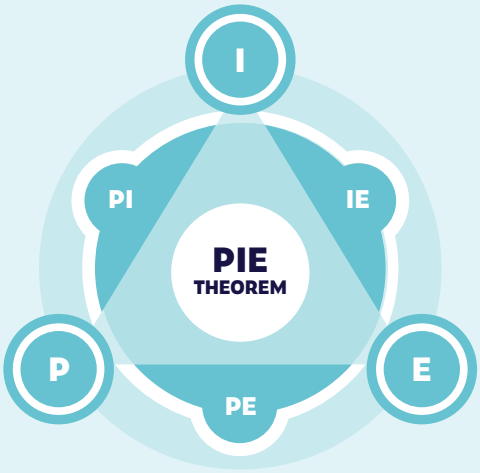
V

Looking ahead: Database trends

It's always a good idea to have an eye toward the future when making technology decisions in the present. Here are some interesting trends I'm watching in this field.

1 The CAP theorem is getting fuzzy

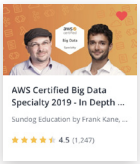
Can you have your cake and eat it, too? Recent advances mean you don't necessarily need to make the usual trade-offs in the CAP theorem triangle. Amazon Redshift, for example, is a relational data warehouse that is fully distributed, horizontally scalable, and highly reliable. MySQL and PostgreSQL offer sharding mechanisms to access the benefits of non-relational databases. And already, most database systems can provide high availability, even if it's due to a nominal trade-off with consistency.



Some big data experts have proposed replacing the CAP theorem with the PIE (**platform flexibility, infinite scale, and efficiency**) theorem as a better reflection of the trade-offs modern system architects must make.

2 The growth of data lakes: Making data structured

Increasingly, pools of unstructured data (such as CSV or TSV files) are being stored in large repositories stored in the cloud. These are called "data lakes." Systems, such as AWS Glue, can impart structure and offer data queries by relational databases without making a copy of the data in the process. This approach offers the benefits of massively scalable, unstructured data, together with the ability to query that data as you would from a relational database.



Certify your team as big data experts with the [AWS Certified Big Data Specialty 2019](#).

3 Search-engine databases

This type of non-relational database uses indexes to categorize data by its similar characteristics. A popular example of this is Elasticsearch, an efficient, scalable data store in addition to a capable search engine. Many organizations use Elasticsearch to store numerical data, while using its "Elastic Stack" tool to visualize and analyze the data. The tool also includes machine learning capabilities to automatically identify anomalous data and tools for transferring data into Elasticsearch at massive scale.

Stay ahead of the competition. Learn more about Elasticsearch:
Elasticsearch 7 and the Elastic Stack - In Depth & Hands On!

4 Time-series databases

Analyzing data for trends over time dictates the need to index your data by time, in time order. Like graph databases, time-series databases serve a specialized need — but it's a common one. These systems are still emerging, but it speaks to a larger trend of using many different, specialized databases for the many different, specialized challenges your organization faces. Some time-series database vendors include InfluxData, kdb+, and Prometheus.