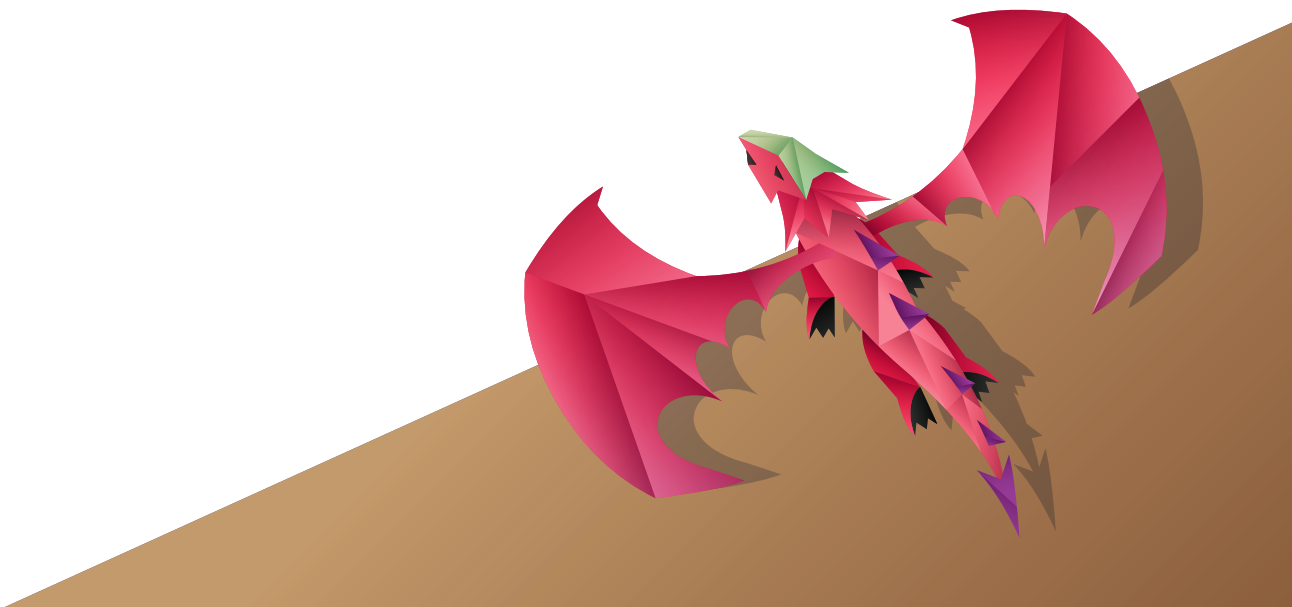
Data Engineering: Inventions and Milestones Along the Way

panoply.io

Table of Contents



A number of really smart people have contributed a lot of knowledge and insight over the years toward the research that comprises modern data engineering. There have been numerous inventions—both hardware and software—that have been critical in serving growing data analysis needs. In this paper, you'll learn about these inventions and how they're contributed to the study of data and the creation of underlying infrastructure necessary to study data.

Data Storage

When people think about "number crunching", they're often thinking about the computing power involved in analyzing mountains of information. Even though they understand that where that information lives is important, it's sometimes still considered an afterthought or that "any old storage" can get the job done.

Innovations and inventions in data storage systems, however, have proven extraordinarily beneficial for data scientists, and data engineers spend considerable time and effort designing data storage systems that can support the "Seven V's of Big Data". In this case, he first two V's—volume and velocity—are the most important. Data storage systems must be able to support the growing capacity needs of data analysis professionals and must also be able to support the fact that data is hitting storage faster.

For data engineers and other IT professionals, these are common scenarios that have been battled for years. There is a constant need for more storage space and storage performance challenges have, until recently, been notoriously difficult to solve. Flash storage has effectively—at least for now—solved many of the storage performance challenges that organizations have faced.

But, data engineers didn't start with flash storage as an option. They didn't even start with hard drives. To look at the history of storage is to take a trip through the forest of trees, magnets, and floating gates. Before we jump into that topic, though, there's an important consideration: in the world of data engineering, storage has three key characteristics:

• **Capacity**

How much actual physical capacity does the medium support? As data sets grow larger, this is really important.

• **Velocity**

How fast can the medium keep up with data ingest? As data is captured, can storage keep up?

• **Retrieval**

When the time comes to analyze data, how quickly can it be retrieved? If you're storing data on a medium that has blazing velocity, but it takes weeks to read data back, it may not be suitable for comprehensive analysis.

Punch Cards

Back before the heyday of hard drives, paper ruled the day. There was no such thing as a spinning disk on which to store data, but companies across the globe yearned for ways to more consistently store their data and automate machinery. In fact, punch card history can be traced all the way back to 1725, when Basile Bouchon used this "technology" to semi-automate a loom.

And, although others played roles in this space, it is Charles Babbage that is generally credited as "the father of the computer" thanks to the invention of his Analytical Engine, a punch-card driven machine intended to enable complete general purpose

Ada Lovelace (1815-1832)
World's First Computer Programmer

Although Charles Babbage created the world's first functional computer, it was Ada Lovelace (born Augusta Ada King-Noel, Countess of Lovelace) who created what became the first algorithm to be used by such as machine. As a mathematician, Lovelace's first workable algorithm would have allowed Babbage's Analytical Engine to calculate a sequence of Bernoulli numbers, important figures in number theory. Both in 1815, Lovelace died in 1832 od uterine cancer.

computation. The successor the Babbage's original Differencing Engine, the Analytical Engine was never actually completed, but the design was intended to enable the use of formulas and data via punch cards. Babbage was only able to construct one part of the Analytical Engine before his death in 1871.

Full commercialization of punch card-driven technology was made possible by International Business Machines (IBM). Although they weren't the sole company providing such tools, they were recognized the leader in the space. In fact, thanks to their ubiquity, punch cards were sometimes referred to as IBM cards or Hollerith cards. In 1928, after an internal contest, IBM released what became the signature punch card: the rigid rectangular paper with rectangular holes arranged in an 80-column pattern (Figure 2-1). This 80-column layout would go on to define data and even display layouts for decades to come.

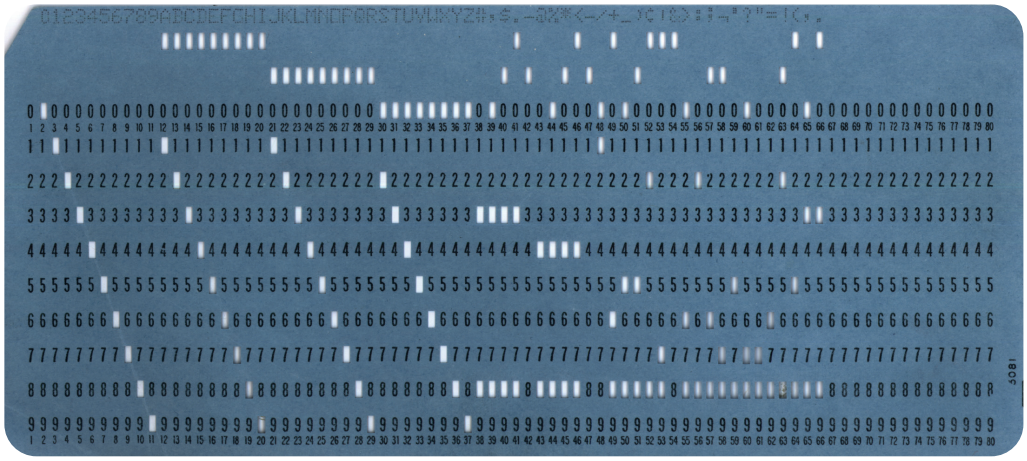


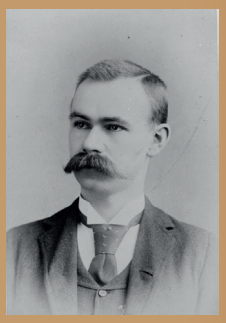
Figure 2-1. A standard issue punch card

Punch card data capacity measurements were dependent on how data was encoded on the card. For a character-based encoding method, storing 80 bytes per card was a typical capacity. For binary encoding, that might jump to 120 bytes (960 bits), but retrieval was more difficult. While 80 or 120 bytes per card may seem paltry by today's standard, keep a couple of things in mind.

First, it wasn't always about the capacity. The ability to automate calculation and business processes was critically important. Second, cards were stored in stacks and

boxes. Let's assume that you have a single box of 2,000 cards. That box stored 160,000 bytes of information, which, considering the simplistic nature of data storage at the time, wasn't necessarily bad.

By the way, punch cards and similar technologies are far from dead. In fact, we still use punch cards even today. Although it was 1999, you may recall the infamous issue of the "hanging chad" that epitomized the 2000 U.S. Presidential election. The machines in use in Florida at the time operated by reading the holes that people punched in their ballots. In a similar fashion, many of the ballot systems used in the 2016 Presidential election operated by reading circles that voters color in to cast their vote. These cards are then fed into a scanner that looks for the colored circle placement and votes are tabulated accordingly.



Herman Hollerith (1860-1929)

Inventor of the Punched Card Tabulating Machine

For decades, the punch card was ubiquitous with big-iron computing and handled data transfer and automation tasks for millions of people and organizations. Herman Hollerith is widely credited with this invention. After founding the Tabulating Machine Company in the late 1800's Hollerith went on to, in 1908, combine his companies with a number of other to form the conglomerate that would ultimately become IBM.

One of Hollerith's original products was a tabulating machine intended to streamline the 1890 census. This machine enabled the U.S. Census Bureau to process the census in just six years, as opposed to the eight years that the less-advanced 1808 census took. This is just one early example of how technology was brought to bear to solve a difficult forward.

Punch cards were certainly an efficient way to store data in a consistent format and, with enough boxes and physical warehouse space, storage capacity was practically limitless. However, in reality, there are physical storage constraints that companies needed to consider. As data volumes increased, punch cards simply couldn't scale to meet demand.

Further, information retrieval was somewhat unpleasant. While it was easy to store data, getting it back was a chore and was prone to error. Had punch cards not given way to newer technologies, we would not be able to process the mass of data that we do today. Fortunately, newer technologies did come along, and changed the game.

MAGNETIC TAPE

Magnetic tape was a revolution in data storage as it allowed mass quantities of data to be stored and archived for long periods of time. Moreover, magnetic tape was portable, so it also improved data sharing capabilities. For decades, it was very common to see reels of tape being shipped from place to place as organizations shared information with one another.

HARD DRIVE

Reading holes in paper was, at the time, revolutionary, but it was the next evolution in data storage that propelled the data engineering and data science disciplines forward in time. And, again, IBM was at the forefront of this innovation. In 1956, IBM released the RAMAC 305 (Random Access Method of Accounting and Control), a magnetic data storage system that held 5 MB of data. Weighing in at over a ton and costing $50,000 ($10,000 per megabyte), the system consisted of fifty 24-inch storage platters that spun at 1,200 revolutions per minute. IBM advertised the system as being able to store the equivalent of 64,000 punch cards.

The first two words in the name of this device described the innovation: random access. Although the initial model averaged 600 milliseconds of seek time to find a record, this was orders of magnitude faster than anything that came before it. Although RAMAC was no longer produced after 1961, it was this invention that really launched modern computing as we know it. No longer were we bound to serial or paper-based data storage. Data records could be randomly read from this innovative new media device, completely streamlining the ability for data scientists to work their magic.

As you know, today's spinning disks are many, many orders of magnitude larger and faster than those original hard drives. However, they still operate on a similar principle. The disk's read and write heads hover on a cushion of air—or, helium, which is a recent development—while the platters beneath whiz around. Those heads move back and forth and await the arrival of the appropriate tracks and sectors

on the platters at which point the read and write head spring into action either retrieving data from the platter or writing new data to the platter.

Reynold B. Johnson (1906-1998)
Inventor of the IBM Hard Drive

Born in 1906 in Minnesota, Reynold Johnson began his career as a high school science teacher. During this time, he created the world's first electronic test scoring systems that used pencil marks on paper to indicate the choice the tester made when selecting from a list of multiple choice questions.

IBM, then based in Endicott, NY, took notice and bought the invention and commercialized it as the IBM 805 Test Scoring Machine. Ultimately, Johnson test scoring invention went on to help advance the then-popular punch card industry.

Johnson's data storage inventions didn't stop there, though. In 1956, he led the IBM team that was responsible for creating the IBM 305 RAMAC, the first commercially available hard drive. Weighing in at a stunning two tons and holding 5 MB of data, that first hard drive was just the first invention among many that advanced the magnetic data storage industry over the proceeding decades and even into today.

There have also been other major advancements in hard drive technology in recent years. As data capacity needs have continued to expand, disk manufacturers have constantly fought back against the laws of physics in an attempt to cram more and more data into smaller and smaller sections of a disk platter.

- **Perpendicular magnetic recording.**

Think of storage data blocks as dominos. Prior to the introduction of perpendicular magnetic recording, these dominos were laid on their large flat side upon being written to disk. With perpendicular recording, the data blocks are instead storage on their small short side, meaning that far more data could be saved in the same amount of actual plater space.

- **Helium sealed**

Remember, read and write heads run on a little cushion of air above disk platters. However, the air in this equation creates drag and friction, thereby increasing the temperature of the drive. Further, the motor used to spin the platter also has to be

powerful enough to overcome this friction. With helium-sealed drives, helium is used on place of the oxygen/nitrogen mixture we call "air". As a result, there is less drag and less friction, meaning that drives don't get as hot and they don't need motors that are as powerful as air-based disks. The result is a lower cost per GB and, with less heat generated, potential savings in cooling the data center, too.

• Shingled magnetic recording

In a spinning disk, the write head is generally a little bit wider than the read head. This results in data tracks that are written using more area of the disk than strictly necessary for data retrieval. In a disk that uses shingled magnetic recording, data tracks are written such that they slightly overlap with their neighbors. However, since the read head is a bit narrower, it can still read just the data that the user is looking for. The end result is that such disks can store more data than more traditional writing methods.

The point of this section is to demonstrate that we're far from an end-game when it comes to innovation in spinning media. In fact, each year, it seems like disk manufacturers are releasing disks that dwarf the previous year's model. As data sets continue to increase in size, these kinds of capacity increases are really important.

FLASH STORAGE

Even with all of their capacity advances, spinning disks utterly failed when it came to meeting the raw performance needs of modern applications, including analysis tools. A new type of storage was needed and it needed to be fast. It came in the form of what we call flash storage.

Although variations of this solid state storage technology have been around for decades, it's only been in the past ten years or so that the storage technology has been accessible to a wide swath of the market. Prior to that, it was simply too expensive and manufacturers had yet to correct some reliability flaws inherent in the technology.

Fast forward to today. The NAND-based flash storage media market has exploded and flash prices, on a dollars per gigabyte basis, have continued to plummet, bringing the technology far closer to a financial reality for more and more organizations. Moreover, flash storage manufacturers have heard the capacity plea and have reacted by making available individual disks with massive storage capacities. As of this

writing, Samsung and Seagate have releases single drives that sport capacities of 16TB and 60TB, respectively.

All of these advancements have created a storage environment that can help organizations easily support the volume and velocity of data projects in this era of big data.

SQL and RDBMS

Storage is really important, but the advancements that have been made over the decades are just part of the data engineering puzzle. The next piece revolves around organization of the data that comprises data projects.

INNOVATION

The database management system (DBMS) and the relational database management system (RDBMS) revolutionized data management by creating an accessible structure that anyone could easily leverage. In addition, these systems enabled reasonable data growth options.

LIMITATIONS

Unfortunately, like most things in IT, SQL and the RDBMS have hit limits as to what they can offer, some of which are by design, and some of which have been discovered over the years. Please note that this section is not intended to say that relational databases are bad. Nothing could be further from the truth. RDBMS tools continue—and will continue—to play a critical role in the enterprise, particularly where ongoing business processes are concerned. However, in the realm of big data, RDBMS tools do have some weaknesses that make them unsuitable for certain needs. RDBMS tools have become just one part of the database arsenal.

OFTEN PROPRIETARY

Some of the world's most well-known RDBMS systems are proprietary products and are expensive to operate. Companies often pay exorbitant sums of money to implement and maintain these systems and it's become simply a cost of doing business. Of course, there are plenty of open source RDBMS tools out there, such as MySQL and PostgreSQL, but many commercial applications rely on closed source RDBMS tools.

Co-founded as Software Development Laboratories in 1977 by Larry Ellison, Bob Miner, and Ed Oates in 1977, the company was renamed to Oracle in 1982. Oracle is credited with the release of the world's first commercially available relational database management system, which hit the market in 1979, as well as with a multitude of other innovations.

Since then, Oracle has grown into a company with a market capitalization of over $180 billion and, as of 2015, held more than 40% of the RDBMS market. Along the way, Oracle has continued to add enterprise class features and capabilities to its platform. Oracle has taken the original RDBMS idea and used it to revolutionize the database market.

SCALE

Today, systems deployed in the enterprise need to be able to scale to levels that, just a few years ago, would have seemed unreasonable. But, given the amount of new data that is being captured and the way that data is leveraged across the enterprise, RDBMS tools have had to create some complex structures in order to continue to support data expansion.

The reality is that scaling relational database management systems is a difficult undertaking. The entire purpose of the original RDBMS was to keep data consistent and available on a single system. Scaling such systems is often achieved via two methods, both of which have serious downsides:

• Get bigger servers.

As your data sets grow and you need more computing and storage resources, you can buy bigger servers. The huge downside here, though, is that you will eventually run into limits. Single servers can only scale so far before you're not able to add any more resources. Those limits might be imposed by the hardware, by the operating system—for example, Windows Server 2016 supports up to 24 TB of RAM—or even by the database software itself. As such, even though modern operating systems and RDBMS tools can grow pretty large, the fact is that data sets are growing larger and

need to be able to reasonably grow beyond the confines of a single system. Worse, as you discover a need to add resources, you often have to take the system offline in order to expand those resources, making the data unavailable for the duration. In many 24/7 operations, this is simply not possible.

• **Clustering.**
Modern RDBMS tools have implemented various clustering and other capabilities in attempts to expand the walls of the server. However, even the best systems, upon inspection, are often found to have some kind of an underlying single point of failure or shared resource that will ultimately impose expansion limits.

In common IT parlance, RDBMS tools are really great when it comes to scaling up, that is, growing within the confines of a server, but they do have challenges scaling out, beyond the walls of the server. In scale out systems, the environment is able to continuously grow across servers, across data centers, and even across geographies

RIGID STRUCTURE

One of the greatest strengths in SQL-based RDBMS tools is also one of its greatest weaknesses: a rigid data structure. If you've worked with traditional databases, you are familiar with the process of defining data structures, which include field names, field types and, sometimes, lengths, relationships, primary keys, foreign keys, and many other standard elements. It is these structures that define the data and that enable consistent consumption and use of that data. These structures have also proven to be generally flexible, at least to a point. For example, suppose you have a customer database and you want to add information about all of the sales orders placed by that customer. To make that work, you simply add a table to the database for sales orders, making sure that you include the customer ID number as one of the fields. By doing so, you're then able to join the sales table with the customer table and run queries that return information from both tables.

As mentioned, this structured data system has revolutionized business and the data consistency provided makes it possible for business processes to have data that can be relied upon for automation and other services. However, modern data consumption efforts are often based around the need to handle unstructured data, for which most RDBMS tools are inadequate.

DATA PARTITIONING AND SHARDING

So, your databases are getting huge and performance is suffering as a result. What do you do? You might consider database partitioning or sharding as another way to be able to grow databases beyond the confines of a single server or single local instance. These capabilities are also used to address complexity inherent in large databases and can be used as a part of an availability strategy.

Ken Thompson (1943-)
Creator of DBM

Long before C came B, a programming language that was, in many way, a descendant of ALGOL. Developed by Ken Thompson and Dennis Ritchie, B was developed around 1969 while Thompson worked at Bell Labs, which was the research arm of AT&T. Although the development of a complete programming language is certainly an impressive feat, Thompson also has another claim to fame and it's one that aligns perfectly with the data engineering focus of this paper.

After B came D... dbm, to be precise. Short for DataBase Manager, the dbm library is a simple database engine that was released by AT&T in 1979. dbm is a pre-relational database included in UNIX operating systems. At the time of dbm's release, UNIX was just eight years old. By bringing dbm to the operating system, a whole new set of capabilities was opened up.

Today, Thompson works at Google as a Distinguished Engineer, where he was a part of the team that developed the Go programming language, which hit the market in 2007.

Partitioning and sharding both work by dividing databases into independent parts, which are then spread over a series of database instances or physical nodes. Simplistically, sharding is just a form of partitioning, but there is confusion on the terms:

- **Partitioning.**

This is simply the act of breaking a database up into smaller parts. Those parts may run locally, but they general share the same underlying database schema.

• **Sharding.**

In general, sharding refers to ways that databases can be horizontally partitioned across multiple computers. The schema is generally replicated to each node and the local partitions/shards use that copy of the schema for their operations.

Partitioning is supported by most of the major RDBMS systems out there, both commercial and open source. Moreover, the limits associated with these capabilities are increasing every year. However, this feature is still a workaround to address an inherent design limit in RDBMS and they can be somewhat complex to understand and to implement.

NoSQL (Not Only SQL) and Open Source Data Science

In recent years, as data sets have become larger, but due more to the fact that they've become far more varied, the need has arisen for tools that could manage data but without the rigid consistency required by traditional relational database systems. Newer systems need to be able to access data by some other means that the tables and fields structures imposed by relational databases. Further, there has been a growing need for data management systems that can scale out and harness the power of dozens, hundreds, thousands, or more, servers to help organizations maintain massive data sets.

On April 4, 1975, a company was founded that would go on to permanently change the face of enterprise computing. This company, Microsoft, has recently turned 43 years old and has managed to continually reinvent itself to stay relevant in a constantly changing world.

In 1989, Microsoft, partnering with Sybase and Asthon-Tate, first jumped into the database market. During that year, SQL Server 1.0 was released for the OS/2 operating system. It wasn't until 1993 that SQL Server left the loving arms of OS/2 and made the move to Windows NT. Today, Microsoft SQL Server controls almost 20% of the relational database market.

SQL Server has come a very long way since those early days and is always adding more capabilities. Moreover, the company is adding SQL

Enter NoSQL. Although the modern NoSQL movement harkens back to only the early 2000's, there have been iterations of these tableless databases for decades. But, with the rise of the Internet and the associated increase in the amount and variety of data being captured, there has been a renewed interest in these products, and, today, they're capturing significant attention in the marketplace.

But, why? Let's start with a bit of database and computer science theory. Bear in mind that there have been thousands of papers and blog posts written on these topics. What we're presenting here is just the tip of the iceberg and is presented as general education. The NoSQL industry is vast, and there are nuances to every implementation.

The CAP Theorem

First conceived by Eric Brewer in 1998, the CAP theorem states that it's impossible for a distributed computer system to simultaneously achieve more than two of the following three characteristics:

- **Consistency (C).**

Every read is able to return the most recent information that has been written. There is a single, current copy of all data available to be read.

- **Availability (A).**

Any node in the cluster is capable of executing queries against data.

- **Partition tolerance (P).**

Even in situations in which nodes are unable to communicate with one another, consistency and availability are maintained.

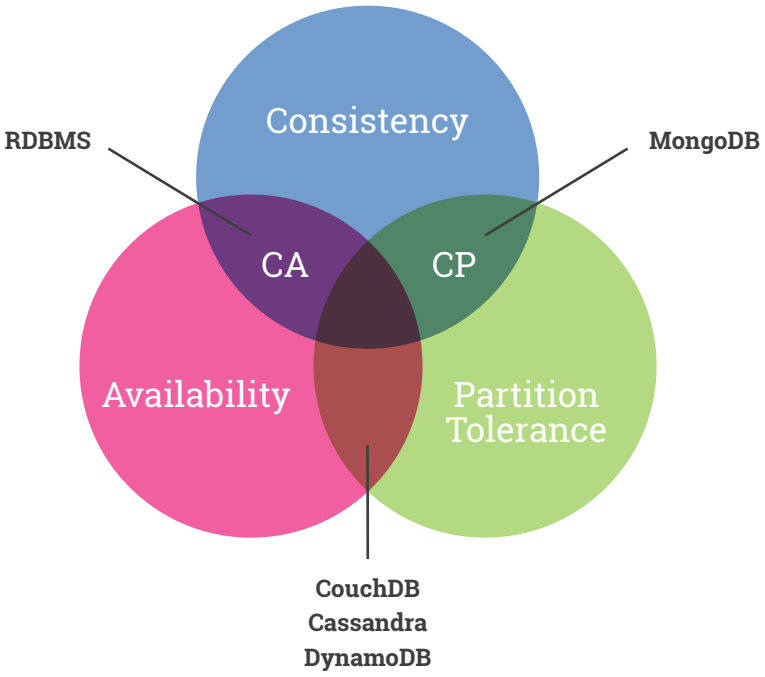


Figure 2-2. The CAP Theorem.

Figure 2-2 depicts the diagram that is commonly used to illustrate the CAP Theorem, what it means, and how various products in the market fit into the equation. For example, consider traditional relational databases, such as MySQL, PostgreSQL, or SQL Server. They meet the A & C, availability and consistency requirements, but not P, or partition tolerance.

The theorem is not perfect and has its detractors and even Brewer himself has written clarifying material around CAP since developing it, but, in general, it's a good illustration for why there is a current need for different kinds of systems to support different kinds of data.

Moving from ACID to BASE

Consistency and availability are critical elements in CAP. Even the P in CAP—Partition tolerance—refers back to these two elements. When it comes to overall database design, there are multiple philosophies at play and what you choose is dependent on your overall requirements.

Let's start with a bit of data chemistry and discuss ACID. ACID is a mnemonic device that outlines core requirements for every database transaction:

Eric Brewer
Information Theorist

Eric Brewer is a tenured professor of computer science at UC Berkley, although, since 2011, he has served as Google VP of Infrastructure. In 1998, Brewer's CAP Theorem made its first appearance, although it took until 1999 for the renamed CAP Principle to be published. Brewer is also credited with coining the term BASE, which stands for "Basically Available, Soft State, Eventually Consistent" and is intended to demonstrate the different between "big data" databases and traditional ACID-based relational ones. Brewer has won a number of awards, including his 2007 induction as a Fellow of the Association for Computing Machinery as well as his 2007 induction into the National Academy of Engineering for the design of highly scalable internet services.

• **Atomicity**

Many transactions involve multiple discrete pieces of data. For example, suppose a customer orders a product from your web site and wants it shipped. The complete transaction would require that the customer's sales history is updated, the inventory database is updated, and the delivery record added to a delivery manifest. If any one part fails, bad things happen. The customer may never see the order on his history page. Your inventory might end up inaccurate. Or, the customer may not have his order delivered. Atomicity is a requirement that, should any one of the database updates for any part of a transaction fail, the entire transaction is failed.

• **Consistency**

Atomicity leads to consistency in data and is very closely related. Consistency imposes a requirement that any failures in a transaction will result in all data being returned to the state it was in prior to the transaction being initiated. So, if a transaction fails half way through executing, any changes that were made up to that point will be rolled back.

• **Isolation.**

To prevent potential inconsistency, any transactions that are in progress, but not yet completed or fully committed must remain fully isolated from other transactions. If transactions are performed concurrently, the end result must be the same as if the two transactions were performed one after the other.

• **Durability.**

Even in the event of a systems failure, any committed data must be available, correct, and current.

When you consider the importance of your data, these requirements make a great deal of sense. However, as databases are distributed, the CAP Theorem states that not all of these elements can be guaranteed to be maintained.

In chemistry, the opposite of an acid is a base and, in the world of databases, there is a similar phenomenon. BASE is the antithesis of ACID and carries with it a number of different characteristics:

• **Basically Available.**
The system ensures that data will be available in that there will be a response to every request for data. However, that request may result in a failure to actually retrieve that requested data or the data could still be in an inconsistent state. In short, basically available means that the database is working most of the time.

• **Soft State.**
Thanks to the fact that these systems operating with eventual consistency (defined below), the ongoing state of the database may change even when there is no active input taking place. So, whereas ACID-compliant systems are always rigid in state, BASE-centric systems can be considered soft in state.

• **Eventually Consistent.**
The database doesn't have to be perfectly up to date all the time. It's sufficient that the database will eventually be consistent once it stops accepting input. Take an accounting department, for example. There is an end of year process that these departments undertake to ensure that all of the transactions for the prior year have been properly recorded. It's understood that, only after the books are closed—that is, all transactions have been added—will all data be consistent. In the meantime, it's acceptable to run queries against the "good enough" data that's entered on a daily basis.

It's easy to see that BASE has far looser characteristics than ACID. Neither is necessarily "wrong" but the choice of the kind of database to choose—ACID- or BASE-compliant—is dependent on the kind of application you intend to support. For a customer sales tracking system, an ACID-compliant database is necessary. However, for other use cases, BASE-compliant databases may be sufficient or even desired. For example, consider Etsy. In order to analyze the mass of log data that the company generates, they deployed a Hadoop cluster. This cluster analyzes vast quantities of data to learn about user behavior so that they can make more targeted recommendations. This data does not need to be perfectly aligned to a strict structure. It just has to be in good enough shape for an analysis tool to figure out what's happening. But, more importantly, it has to be able to deal with huge data sets, which Hadoop is great at doing.

Before we wrap up this section, understand that this ACID-BASE spectrum is not nearly as black and white as it might first appear. There is actually a whole lot of grey to work with. There are ACID-systems that can do some BASE things, and vice versa.

But, the point overall is that there is a need to be able to support big data sets. These data sets often thrive at the BASE end of the spectrum and that also happens to be where NoSQL shines.

NoSQL history

First of all, NoSQL doesn't mean that SQL is simply jettisoned. SQL is useful, but so are other tools. NoSQL actually stands for Not Only SQL, although that definition has morphed a bit over time used to mean "non SQL" or non-relational. That said, even some NoSQL systems today have relational characteristics, so the modern Not Only SQL definition is the most accurate.

The original NoSQL term was coined in 1998 by Carlo Strozzi. In that year, Strozzi made available his database product—NoSQL—which has a claim to fame that it didn't directly expose a SQL interface, although it was a relational database under the covers.

In 2009, the term NoSQL was redefined and it morphed into its current form, where the relational component is either eliminated or significantly deemphasized.

In a NoSQL system, traditional tables and columns are mostly avoided in favor of other kinds of data. Read on to learn more.

Structures

As you may expect, given the broad data needs present in the world, there are a lot of different ways that NoSQL systems can be structured. There are four primary categories into which NoSQL systems fall, each of which is described in the following sections.

Please note that these categories are generalities only. While some NoSQL tools will fall purely in just one of these buckets, some tools will straddle multiple categories and will have characteristics from each one.

Key/Value Stores

The simplest of the NoSQL structures, key/value stores a unique identifier along with an associated value. The key can be just about anything, although it should be unique, and, for performance reasons, it should be relatively short, but it doesn't have to be. Some key/value DBMS tools impose size limits to keys, but these are often specified in terms of megabytes, so "limits" seems to be relative.

The value portion of the key/value pair can be any data that needs to be associated with the key. Figure 2-3 an example of a key/value database that stores a U.S. state name as the key and, for the value, the state abbreviation and capitol.

Key	Value
New York	NY, Albany
Maryland	MD, Annapolis
Pennsylvania	PA, Harrisburg

Figure 2-3. A simple Key/Value database example.

The beauty of key/value stores is their simplicity and sheer speed and scalability. They are fast and the value portion doesn't require a schema. You can store whatever you like. When you need to store a lot of low complexity information, a key/value store is a good option.

However, key/value stores aren't always the best choice if you need to eventually join the value side of the equation to some other database. Because that data isn't necessarily structured and is not always consistent, it can be tough to rely on these for complex data needs.

In the marketplace Amazon DynamoDB is an example of a database that support key/value stores.

Founded in 1994, Amazon has become the world's largest and most disruptive retailer. Beyond retail, however, Amazon has also forever altered the IT landscape by commoditizing what we now call cloud computing. As a part of this service, Amazon makes available three primary database offerings:

Amazon DynamoDB. A NoSQL key/value store database.

Amazon RDS. RDS enables you to create a MySQL (open source), Oracle, SQL Server, or PostgreSQL (open source) database in the cloud.

Amazon Redshift. A petabyte-scale data warehouse that is intended to integrate with third party analysis tools, such as Panoply. Redshift leverages columnar storage.

Document Stores

Key/value stores are pretty simple affairs, but they can't satisfy all data needs. Document stores store data with structure. Under the hood, they're still considered key/value stores, but with the value element encoded using something semi-structured like Extensible Markup Language (XML) or JavaScript Object Notation (JSON), although binary data, such as a PDF file or a Microsoft Word document, can also be stored as values.

The term "semi-structured" was used in the preceding paragraph because document stores don't have to enforce value consistency. So, for one value, you could have an XML document and for the next one, JSON.

In Figure 2-4, you can see a sample of an XML record that might be stored in a document database.

```
<customer>
  <customerid>1400</customerid>
  <firstname>Scott</firstname>
  <lastname>Lowe</lastname>
  <address>
  <type>Work</type>
   <street1>1701 Broadway</street1>
   <city>New York</city>
   <state>NY</state>
   <zip>10001</zip>
  </address>
  <phone>(555) 555-1212</phone>
</customer>
```

Figure 2-4. An XML value in a document database.

In Figure 2-5, you can see a sample of an JSON record that could be stored in a document database.

```
{
  "CustomerID": 1400,
  "CustomerName": "Scott",
  "FullAddress": "1701 Broadway, New York, NY
10001",
  "SecretPasscode": "Cats are awesome"
  "FirstOrderDate": "01/15/2014"
}
```

Figure 2-5. A JSON value in a document database.

One common example often associated with document databases is capturing data from various Internet of Things (IoT) devices. Different devices may capture data somewhat differently, so the flexibility to store different kinds of data is nice, and it's possible to store millions of records in a distributed way.

Since they're basically a variant of key/value stores, document stores also have similar drawbacks. They aren't generally a good fit if you need to join stored data with other data elements in your environment.

If you're looking for a document database for your business, there are a number of popular ones out there, including CouchDB, MongoDB, and the aptly named Microsoft Azure DocumentDB.

GRAPH DATABASES

In discussing key/value and document stores, you learned that they aren't that they don't always play nicely with other data. The thing is that there are a lot of reasons that you might want your NoSQL data to know about and be able to work with other elements inside your database. Of course, you could just revert to a relational database, but you also know that those have their own drawbacks.

This is where graph databases come in handy. Graph databases are the product of a host of research into what is known as graph theory. These types of databases provide far more flexibility than other types because they bring to NoSQL some semblance of structure and interrelationships without having to use a full, strict relational database.

With graph databases, there are a couple of concepts you need to understand: nodes and relationships. Each node in a graph database stores information, but it also stores relationship information that can link it to other records in the database. Graph databases are useful when there are finite relationships between nodes.
How does this translate to a real-world use case? Let's use Twitter as an example. In the Twittersphere, each user could be considered a node. For relationships, we just need to look at who you follow and who follows you. Take a look at Figure 2-6. In this figure, you can see that there are three nodes shown, with each being a user in this social network. There are arrows identifying who is following whom in this web of users. These are the relationships. This type of database is incredibly useful, and

necessary, in a world where social media-based relationships are the norm and people have millions of followers. The kind of performance disaster that would ensue with a more traditional database approach would be catastrophic.

Common graph database tools include Neo4j, Titan, and FlockDB. Fun fact: Neo4j is able to provide ACID-compliant services like relational databases. Remember, the ACID-BASE spectrum isn't black and white!

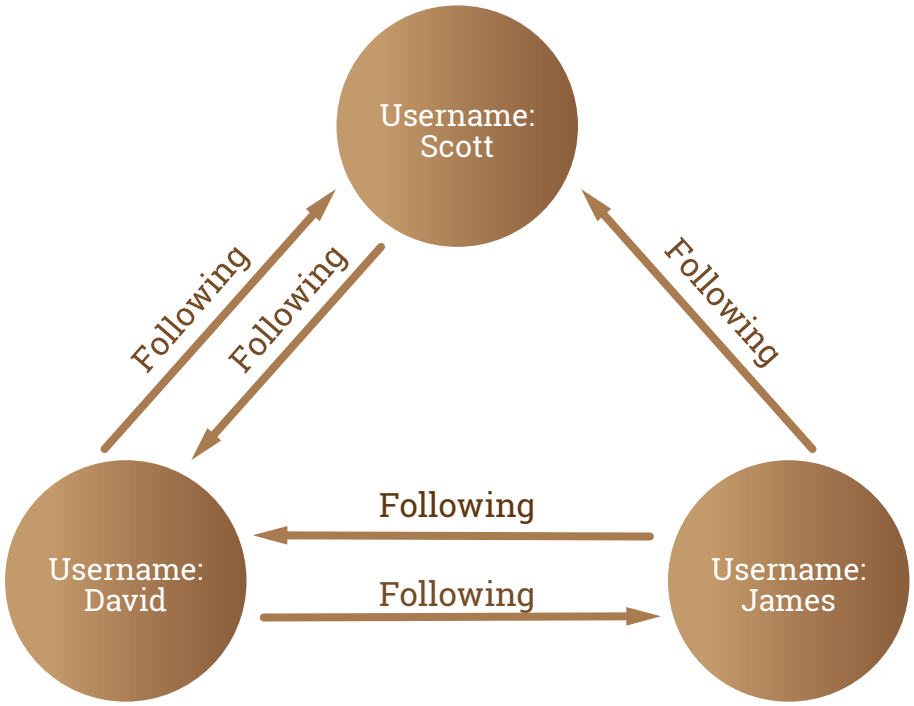


Figure 2-6. A simple Graph database example.

COLUMN STORES

In a column database, data is stored in columns rather than rows. At first glance, you might look at a column store database and think that it's a traditional relational one. However, nothing could be further from the truth. Column-based databases are generally suitable when you need to read just a few columns of information at a time and you don't need to do constant data writes.

Columnar databases are really fast, at least when it comes to reading common information. If you consider how you retrieve traditional relational database information, you're generally pulling one or more fields (columns) from a subset of records (rows). So, to get a single field of information (a column), you need to read through tons of records. In a columnar database, if you were to retrieve a single field, it would impact just one record since data is column based and is indexed based on columns. Yes… it sounds confusing and this attempt at a simplistic explanation may fall flat, but we'll look at an example shortly.

But before we get to that, consider the other side of the database equation: adding data. In a columnar database, as you add new full records that span columns, performance can suffer. Why? In a relational database, when you add a record, you're adding a row of data and, even though that row may have dozens of fields, it's still just a single row operation. When you add such a record to a column database—one that spans fields or columns—you're required to perform a column insert for each and every field. So, adding new data can take a bit longer.

You'll see column databases commonly used for event logging and content management. Apache's Cassandra is probably the most well-known NoSQL column database.

Panoply.io 188 King St.• San Francisco, CA 94107 • hello@panoply.io