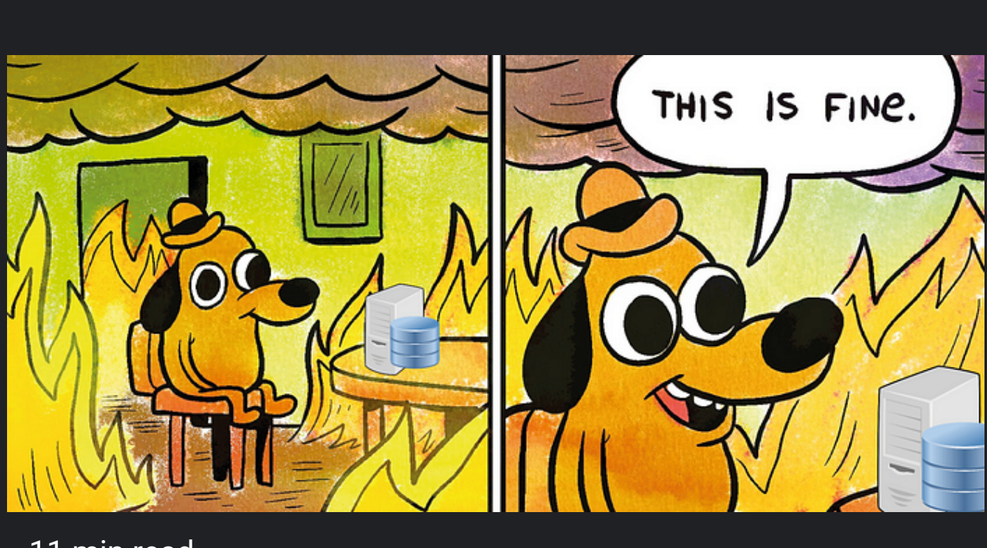
Relational Database Systems Are Becoming A Problem—But What To Do About It? | by Jan Kammerath



~11 min read

·

August 15, 2023 (Updated: August 16, 2023)

·

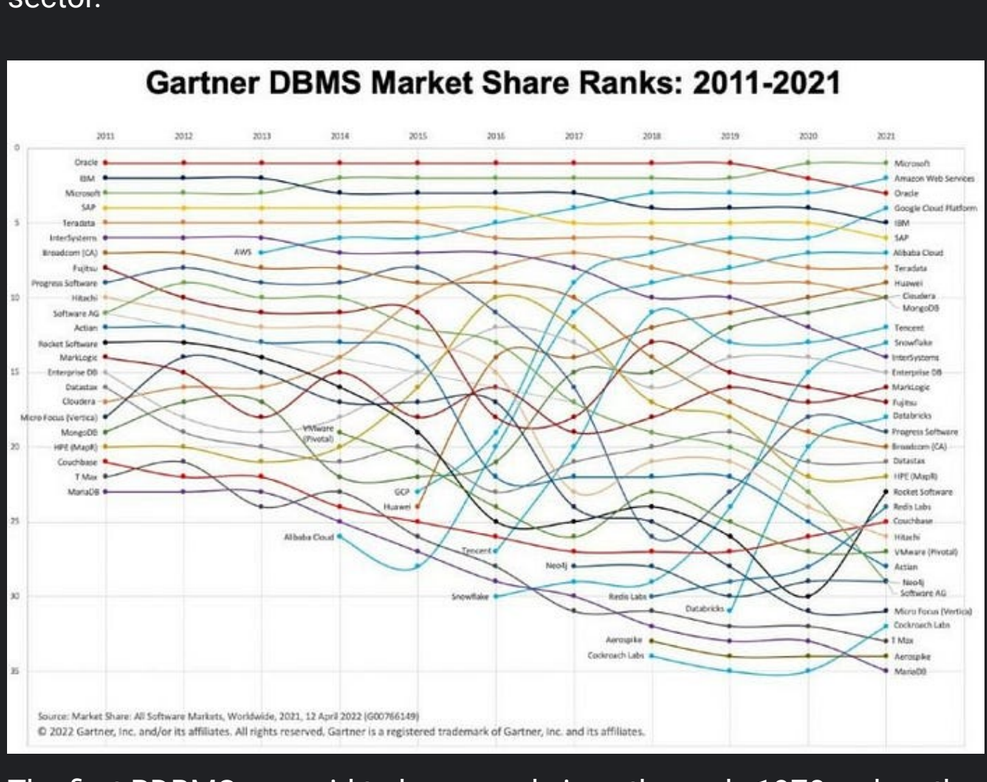
Free: No

My relationship with relational databases relates back to the late 90s. It was part of my first steps with computers and programming, became an essential part of my formal education and studies as a software engineer and constantly followed me through my professional career. I almost crawled through the entire RDBMS rabbit hole and still love it.

During my career, I touched MySQL, Postgres, Oracle, Microsoft SQL Server, DBase, Access, SQLite, DB2, MariaDB, AWS RDS, Azure SQL, Google Cloud SQL and pretty much any RDBMS I could get my hands on. You can't love RDBMS without loving SQL which is a rabbit hole of its own. And not all SQLs are the same. You've got MySQL with its own jargon, you've got Microsoft's T-SQL and the world famous PL/SQL from Oracle. Probably not necessary to mention that they're all not compatible with each other.

It's all relational databases? Always has been.

Trust me, I've seen them all — finance, transportation, hospitality, social media, video streaming services and many others. Regardless of where you go, you'll probably find a relational database. The world seems to run entirely on relational databases filling the pockets primarily for Oracle, IBM and Microsoft. If you need it big, like really big, you're calling up Oracle, IBM or Microsoft. Chances are, you're requirements may also lead you to SAP — especially in the finance sector.



The first RDBMS are said to be around since the early 1970s when the Structured English Query Language *(SEQUEL, later abbreviated to SQL)* was invented. Oracle released its first database in 1979, three years after Honeywell released its Multics Relational Data Store in 1976 — said to be the world's first relational database. In just a couple of years, we'll look back at 50 years of relational database management systems (RDBMS). Unsurprisingly, the RDBMS became the backbone of our modern society and economy. Safe to say that everyone has at least one and everyone is in at least one relational database, unless you live in a cave.

Your social security records, your passport, your police records, your birth certificates and all of that happily reside in massive government owned relational database systems — most likely from Microsoft, IBM, SAP or Oracle. Looking for a trip to the beach? Your tickets, reservations and all of that are in relational databases. Whatever data you give to whatever monstrous organization most likely ends up in a relational database.

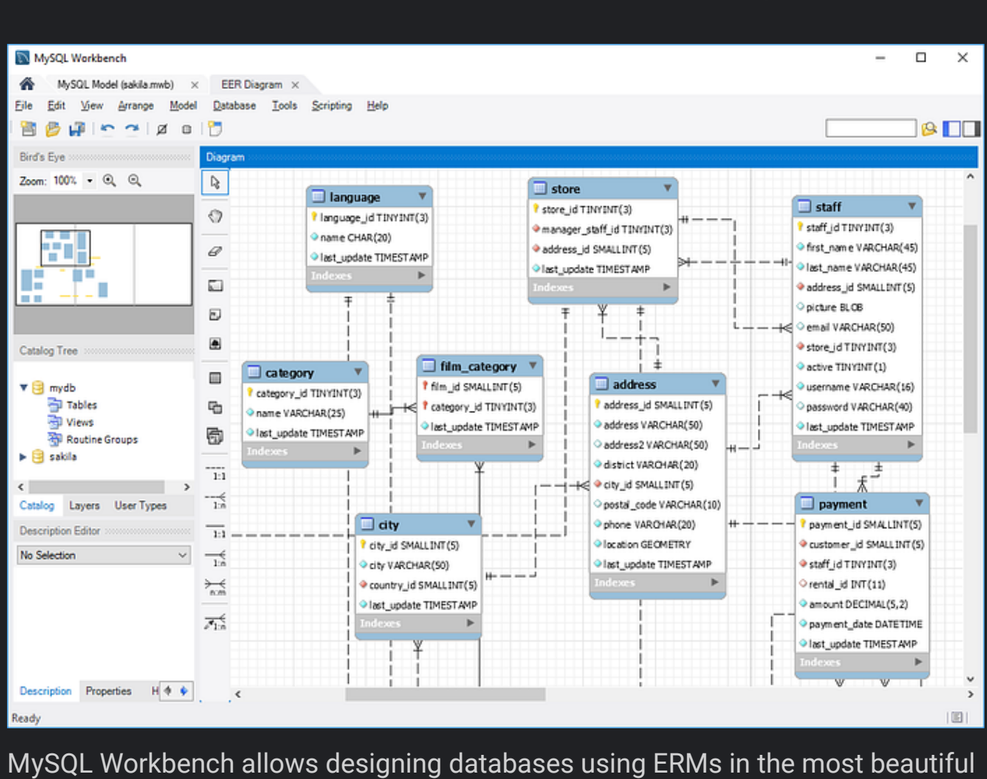
Most database implementations are simple

The majority of databases however exists, in one way or another, in a form comparable to PHP and MySQL or Microsoft Access and VBA *(Visual Basic for Applications)*. These aren't complex database management systems, but rather small applications that just use an RDBMS to store data. For many of them, an RDBMS is a massive overkill to begin with. Only the popularity of relational databases has led developers to pick them. Universities, schools, programming courses all teach SQL and relational databases. Most developers probably come with a tendency to use relational databases.

You might also agree that most software developers aren't good database developers. It's sometimes because they don't care, but mostly because there are really few learning resources out there that teach you how to build relational databases properly. Most universities, schools, books and courses focus on SQL, normalization and transactions. That's it and it shows in the relational databases that live out there in the wild.

> "**An external application shall never even know what tables exist**" — An experienced DBA that retired in 2012 and wish to stay anonymous

The average developer will be shocked at this statement. For experienced database engineers it's the norm to hide away the entire relational database structure behind views and stored procedures.

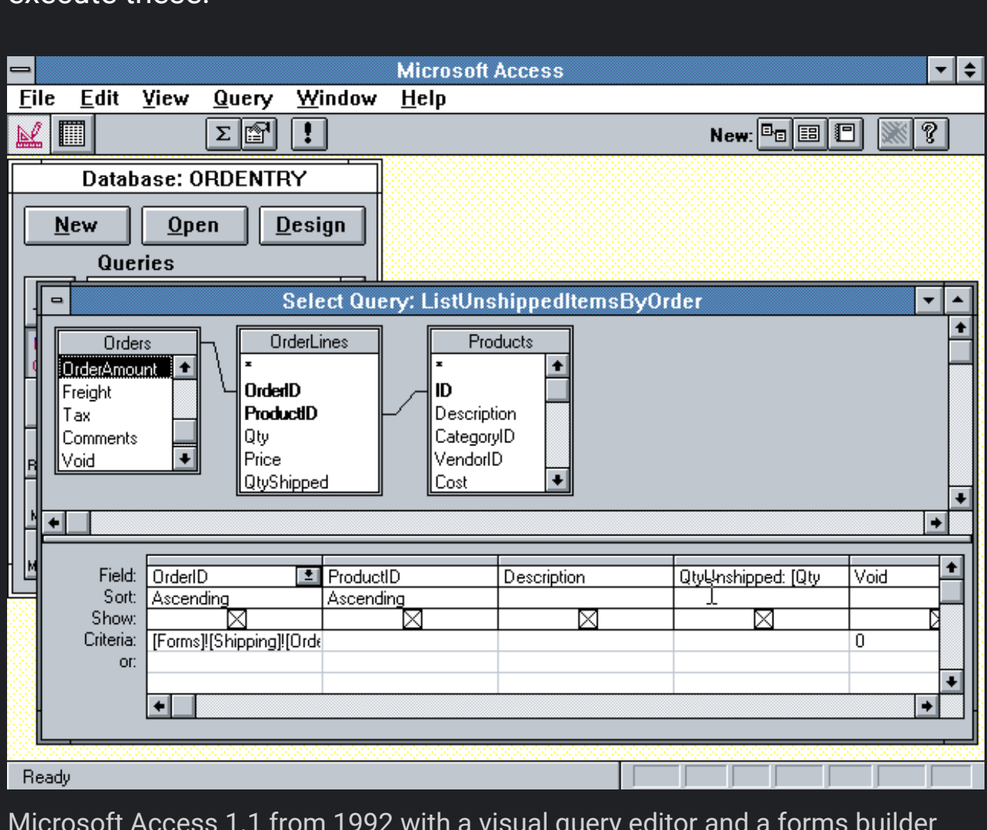


MySQL Workbench allows designing databases using ERMs in the most beautiful way

Database monsters have become irreplaceable

In the 1980s all organizations moved into relational databases. By all, I really mean all organizations there are on this planet. Chances are, if you search long enough, you may still find government organizations that haven't gotten a computer yet. Often, these organizations use custom build databases that reside in mainframe computers for decades, ramping up data and support fees from the manufacturers and vendors.

These custom built databases would contain dozens of intertwined tables invisible to the outside world. An endless amount of triggers, functions, procedures and views would not just organize the storage, but run all business processes of that organization. The applications on the application layer provide the interface for the average joe to work with the database. Yet, these applications mostly don't operate any businesses processes, but merely call stored procedures to execute these.



Microsoft Access 1.1 from 1992 with a visual query editor and a forms builder

Since the database consultants from the 80s have retired decades ago, most of these custom built database systems live out there with their SQL application code mostly unmaintained. For many large organizations, these database applications have become blackboxes. They have no clue what they do and how they work in detail, let alone how they should be maintained. Yet, businesses heavily rely on these applications which by now have become irreplaceable. Reverse engineering and rearchitecting these applications has become to only way for an horrendously large amount of organizations. Those "legacy database migration projects" often come at ridiculously high cost in the multiple millions.

Imagine being an insurance company that has absolutely no clue on how the risk for an individual contract is actually calculated on their mainframe. They couldn't tell their customers what effect a specific claim would have on their premiums. The number of organizations that have no clue how that software runs their business is frightening and hilarious at the same time. Frightening only if you're a customer and it's your data in that blackbox.

What's the problem with relational databases?

I personally came across businesses where non-technical employees were referring to the central relational database as "the Oracle" or "the DB2". Simply because it was such a constraint for the IT department that every change request that affected the RDBMS would become a task for months instead of a few days — with the IT department blaming "the Oracle". The central database became the central point of failure. Sales promotions would go downhill when the database couldn't serve them. Adding a column to a table required regular attendance of sunday prayers. Let's better not start about query performance.

The problem? A relational database and the principles of designing these push you towards centralization of your data inside that database. Your relational database grows as your business grows with the junk data it produces. You will ultimately arrive at a point where your business becomes economically unable to move off that relational database.

The relational database is from a different era

Relational database management systems were invented in an era when the computer looked nothing like what we got today. The use cases were entirely different and the volume these systems had to deal with would fit into anyones pockets today.

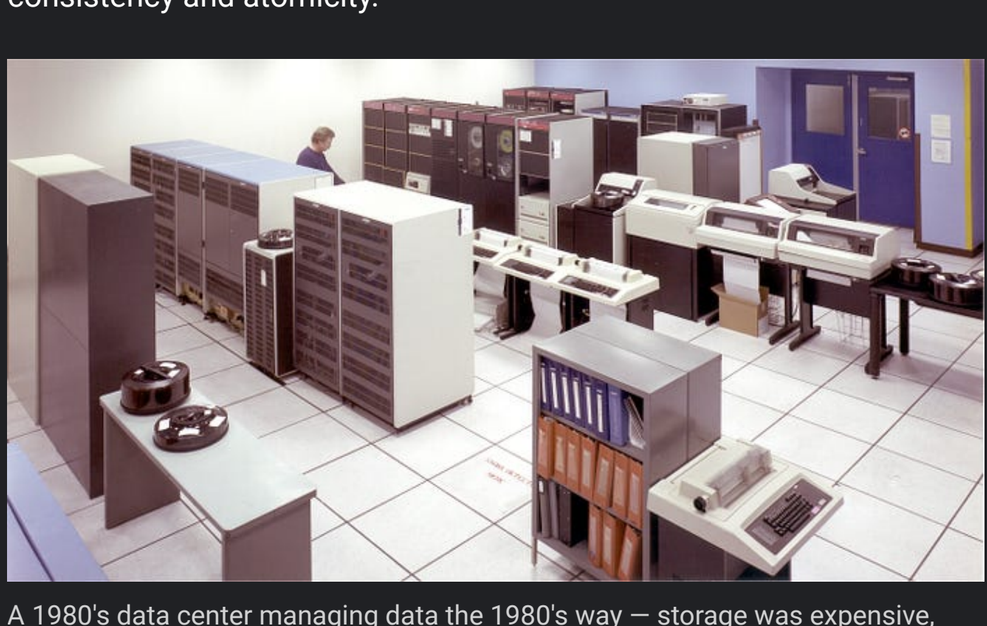
I highly recommend Rick Houlihan for his presentations about databases and the thoughts behind the future of database technology. You should definitely check out his various presentation on YouTube: Rick Houlihan on YouTube. The following interview that he gave in the Software Engineering Daily pretty much sums it up.

> Jeff Meyerson (Founder Of Software Engineering Daily): *There are several explanations for why NoSQL rose in popularity after SQL had been the predominant database type. We've explored some of those different theories on this show. Give me your historical perspective for why NoSQL became popular.*

> Rick Houlihan (MongoDB, ex-AWS): *Well, sure. I mean, really what it comes down to, again, as people started to process volumes of data, the relational database that we've used for so many years turned out to not scale so well. That really points back to the reason why it was invented in the first place, and the relational database came into being because, again, we're experiencing data pressure, it was the cost of processing data that was preventing us from scaling, and the relational database decreased the pressure on the storage systems, because the normalized data model de-duplicated that data and allowed us to free up storage, so to speak, which was really the most expensive resource in the data center 3 or 4 years ago.*

> *But now, today, fast forward, we pay pennies per gigabyte and we're paying dollars per our CPU minutes, and really the CPU is no longer just this fixed asset that's kind of spinning in idle loop when it's not doing anything else. It's an asset that we can use to do other things. So joining data and running complex queries is really not something that we'd like to spend our money on, so to speak.*

RDBMS are also very strong when you have structured data that requires ACID-compliance. However, a number of use cases do not require ACID compliance at all. These include video streaming, gaming, social media, internet search and many more. All these use cases favour speed and performance over ACID compliance with consistency and atomicity.



A 1980's data center managing data the 1980's way — storage was expensive, really expensive

An internet search engine does not need to show every user the latest results and not every user requires the same result. Hence, ACID compliance is totally irrelevant for the use case of an Internet search engine. No one in his right mind would use an RDBMS for a large scale Internet search engine or a social media site.

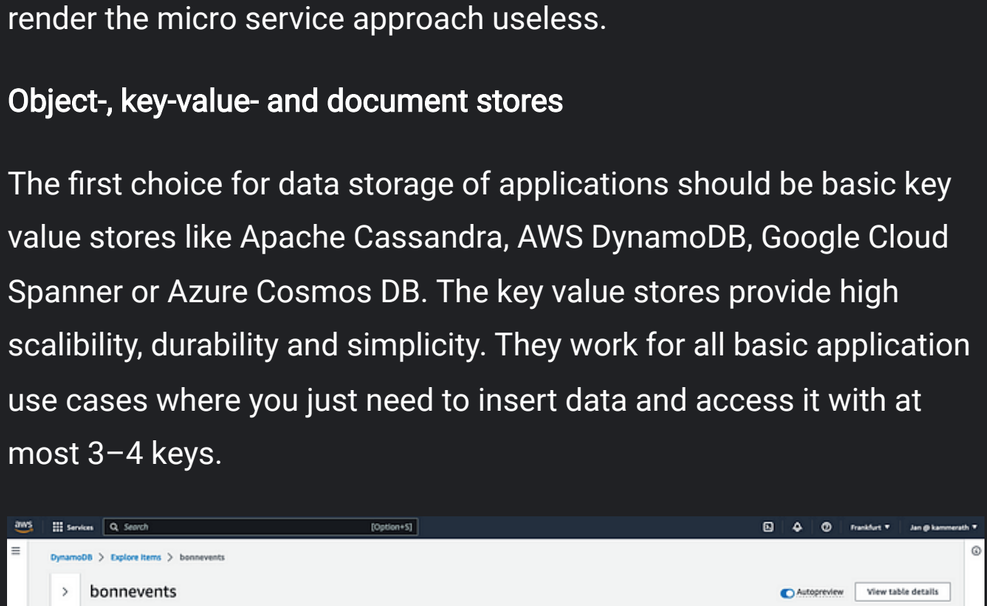
The solution? Purpose built systems.

It's obvious that a general purpose database with a "one fits all" attitude hardly achieves superiority in any use case. Trying to use RDBMS for transactions, search, analytics and any other use case will most likely never have the optimal result for any of these. Hence, the obvious elephant in the room are purpose built solutions. These can be databases, even relational ones, but may also be other systems such as dedicated search engines or even custom software.

The approach of using purpose built data management only works if you strictly adhere to microservice architectures and don't build *"micro serviced monoliths"* in which you have micro services all working on a single centralized data management system like a relational database. It's a common occurence to find micro service architectures paired with monolithic databases which completely render the micro service approach useless.

Object-, key-value- and document stores

The first choice for data storage of applications should be basic key value stores like Apache Cassandra, AWS DynamoDB, Google Cloud Spanner or Azure Cosmos DB. The key value stores provide high scalibility, durability and simplicity. They work for all basic application use cases where you just need to insert data and access it with at most 3–4 keys.



A simple Dynamo table for a local event calendar

Should your data require more complex querying, e.g. search or analytics, you can always switch to a dedicated search engine or analytical system by streaming the data from your key-value-store to the other system. If you don't need querying at all and just require simple data storage, going with an object store such as AWS S3, Azure Blob Storage or Google Cloud Storage is a best practice approach.

Document stores such as MongoDB or AWS DocumentDB try to provide an alternative to relational databases although they often come with the same principles, just not relational. Just moving from tables to documents may still present you with the same issues you had before.

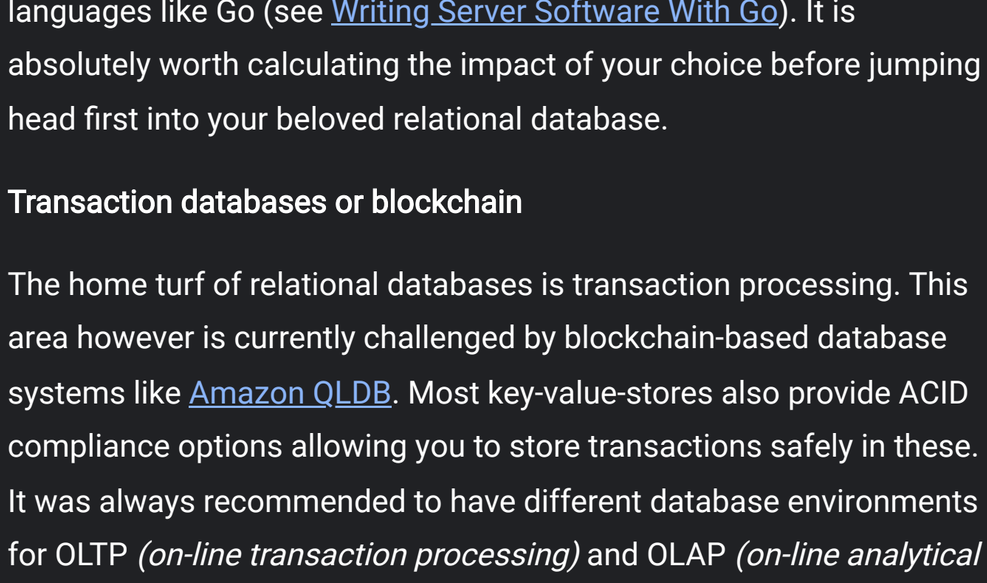
Dedicated or custom built search engines

A common use case for relational databases is search. That's a use case that relational databases are rarely ideal for. Search functionality, in most cases, does not require ACID compliance at all. Purpose built search engine like Lucene, Solr, OpenSearch or ElasticSearch provide significant better performance and have a lower operating cost.

Depending on the use case, the existing offerings from cloud providers such as Google's Cloud Search may already be more suitable for your requirements. If none of that fits your requirements, building dedicated search software tailored to your needs is not too far fetched considering the speed of development that you have with languages like Go (see Writing Server Software With Go). It is absolutely worth calculating the impact of your choice before jumping head first into your beloved relational database.

Transaction databases or blockchain

The home turf of relational databases is transaction processing. This area however is currently challenged by blockchain-based database systems like Amazon QLDB. Most key-value-stores also provide ACID compliance options allowing you to store transactions safely in these. It was always recommended to have different database environments for OLTP *(on-line transaction processing)* and OLAP *(on-line analytical processing)* anyway. Accessing transactions often is done by no more than 3–4 keys, hence key-value-stores may be ideal for transactions as well.



An overview of how Amazon QLDB works

I have personally deployed Amazon QLDB in production and would not go back to relational databases. The advantages of a cryptographically verifyable storage for transactions allows for much greater auditability. While anyone can manipulate a transaction in a relational database, QLDB tracks any change to the records using the strain of the transaction. For financial transaction processing, QLDB is my preferred system. However, it depends on the use case and whether you even need the cryptographic verification for your use case.

Challenge the status quo

I love writing SQL 🐿 with stored procedures, functions, triggers and views. Designing relational databases with MySQL Workbench is fun for me. The newest features in MySQL 8 with geospatial data is amazing. There's so much you can do in relational databases — all in one place. To be honest, I sometimes miss the days when you would just write your entire business application inside MySQL, Oracle or SQL Server. But I need to be honest to myself: It was acceptable in the 80s. We are in 2023, compute and storage have changed and so have our data centers and our applications.

With the vast offering in database systems, key-value and object stores, search engine technology and programming languages, the days of going with a database for decades are over. No more projects with endless debates on whether MySQL, MSSQL, Oracle or Postgres is the right choice. Today's databases and storage are decided upon on a case by case basis. And quite often, I find myself writing a small custom storage strategy that is based on object or key-value stores.

Today, before I implement a software or system, I think about what data is stored and how it is accessed. I then often spent hours and days to find the right approach to data stores. Relational databases are very rarely a part of the solution when I am honest with myself. I've just seen the long-term effects of centralized relational databases too often.

Thank you for reading. Jan

What's your opinion on relational databases and have you caught yourself going with the "I'll just stick to that" mentality?