What Goes Around Comes Around... And Around...

Michael Stonebraker
Massachusetts Institute of Technology
stonebraker@csail.mit.edu

Andrew Pavlo
Carnegie Mellon University
pavlo@cs.cmu.edu

20 YEARS

ABSTRACT

Two decades ago, one of us co-authored a paper commenting on the previous 40 years of data modelling research and development [188]. That paper demonstrated that the relational model (RM) and SQL are the prevailing choice for database management systems (DBMSs), despite efforts to replace either them. Instead, SQL absorbed the best ideas from these alternative approaches.

We revisit this issue and argue that this same evolution has continued since 2005. Once again there have been repeated efforts to replace either SQL or the RM. But the RM continues to be the dominant data model and SQL has been extended to capture the good ideas from others. As such, we expect more of the same in the future, namely the continued evolution of SQL and relational DBMSs (RDBMSs). We also discuss DBMS implementations and argue that the major advancements have been in the RM systems, primarily driven by changing hardware characteristics.

1 Introduction

?!!?

In 2005, one of the authors participated in writing a chapter for the *Red Book* titled "What Goes Around Comes Around" [188]. That paper examined the major data modelling movements since the 1960s:

- Hierarchical (e.g., IMS): late 1960s and 1970s
- Network (e.g., CODASYL): 1970s
- Relational: 1970s and early 1980s
- Entity-Relationship: 1970s
- Extended Relational: 1980s
- Semantic: late 1970s and 1980s
- Object-Oriented: late 1980s and early 1990s
- Object-Relational: late 1980s and early 1990s
- Semi-structured (e.g., XML): late 1990s and 2000s

Our conclusion was that the relational model with an extendable type system (i.e., object-relational) has dominated all comers, and nothing else has succeeded in the marketplace. Although many of the non-relational DBMSs covered in 2005 still exist today, their vendors have relegated them to legacy maintenance mode and nobody is building new applications on them. This persistence is more of a testament to the "stickiness" of data rather than the lasting power of these systems. In other words, there still are many IBM IMS databases running today because it is expensive and risky to switch them to use a modern DBMS. But no start-up would willingly choose to build a new application on IMS.

A lot has happened in the world of databases since our 2005 survey. During this time, DBMSs have expanded from their roots in business data processing and are now used for almost every kind of data. This led to the "Big Data" era of the early 2010s and the current trend of integrating machine learning (ML) with DBMS technology.

In this paper, we analyze the last 20 years of data model and query language activity in databases. We structure our commentary into the following areas: (1) **MapReduce Systems**, (2) **Key-value Stores**, (3) **Document Databases**, (4) **Column Family / Wide-Column**, (5) **Text Search Engines**, (6) **Array Databases**, (7) **Vector Databases**, and (8) **Graph Databases**.

We contend that most systems that deviated from SQL or the RM have not dominated the DBMS landscape and often only serve niche markets. Many systems that started out rejecting the RM with much fanfare (think NoSQL) now expose a SQL-like interface for RM databases. Such systems are now on a path to convergence with RDBMSs. Meanwhile, SQL incorporated the best query language ideas to expand its support for modern applications and remain relevant.

KEY

Although there has not been much change in RM fundamentals, there were dramatic changes in RM system implementations. The second part of this paper discusses advancements in DBMS architectures that address modern applications and hardware: (1) **Columnar Systems**, (2) **Cloud Databases**, (3) **Data Lakes / Lakehouses**, (4) **NewSQL Systems**, (5) **Hardware Accelerators**, and (6) **Blockchain Databases**. Some of these are profound changes to DBMS implementations, while others are merely trends based on faulty premises.

We finish with a discussion of important considerations for the next generation of DBMSs and provide parting comments on our hope for the future of databases in both research and commercial settings.

DATA IS
FOREVER

2 Data Models & Query Languages

For our discussion here, we group the research and development thrusts in data models and query languages for database into eight categories.

2.1 MapReduce Systems

Google constructed their MapReduce (MR) framework in 2003 as a "point solution" for processing its periodic crawl of the internet [122]. At the time, Google had little expertise in DBMS technology, and they built MR to meet their crawl needs. In database terms, *Map* is a user-defined function (UDF) that performs computation and/or filtering while *Reduce* is a GROUP BY operation. To a first approximation, MR runs a single query:

I GUESS?

```
SELECT map() FROM crawl_table GROUP BY reduce()
```

Google's MR approach did not prescribe a specific data model or query language. Rather, it was up to the *Map* and *Reduce* functions written in a procedural MR program to parse and decipher the contents of data files.

There was a lot of interest in MR-based systems at other companies in the late 2000s. Yahoo! developed an open-source version of MR in 2005, called Hadoop. It ran on top of a distributed file system HDFS that was a clone of the Google File System [134]. Several startups were formed to support Hadoop in the commercial marketplace. We will use MR to refer to the Google implementation and Hadoop to refer to the open-source version. They are functionally similar.

HADOOP IS MAPREDUCE

There was a controversy about the value of Hadoop compared to RDBMSs designed for OLAP workloads. This culminated in a 2009 study that showed that data warehouse DBMSs outperformed Hadoop [172]. This generated dueling articles from Google and the DBMS community [123, 190]. Google argued that with careful engineering, a MR system will beat DBMSs, and a user does not have to load data with a schema before running queries on it. Thus, MR is better for "one shot" tasks, such as text processing and ETL operations. The DBMS community argued that MR incurs performance problems due to its design that existing parallel DBMSs already solved. Furthermore, the use of higher-level languages (SQL) operating over partitioned tables has proven to be a good programming model [127].

A lot of the discussion in the two papers was on implementation issues (e.g., indexing, parsing, push vs. pull query processing, failure recovery). From reading both papers a reasonable conclusion would be that there is a place for both kinds of systems. However, two changes in the technology world rendered the debate moot.

The first event was that the Hadoop technology and services market cratered in the 2010s. Many enterprises spent a lot of money on Hadoop clusters, only to find there was little interest in this functionality. Developers found it difficult to shoehorn their application into the restricted MR/Hadoop paradigm. There were considerable efforts to provide a SQL and RM interface on top of Hadoop, most notable was Meta's Hive [30, 197].

WANT SQL ON HADOOP

The next event occurred eight months after the CACM article when Google announced that they were moving their crawl processing from MR to BigTable [164]. The reason was that Google needed to interactively update its crawl database in real time but MR was a batch system. Google finally announced in 2014 that MR had no place in their technology stack and killed it off [194].

MR DEAD

The first event left the three leading Hadoop vendors (Cloudera, Hortonworks, MapR) without a viable product to sell. Cloudera rebranded Hadoop to mean the whole stack (application, Hadoop, HDFS). In a further sleight-of-hand, Cloudera built a RDBMS, Impala [150], on top of HDFS but not using Hadoop. They realized that Hadoop had no place as an internal interface in a SQL DBMS, and they configured it out of their stack with software built directly on HDFS. In a similar vein, MapR built Drill [22] directly on HDFS, and Meta created Presto [185] to replace Hive.

Discussion: MR's deficiencies were so significant that it could not be saved despite the adoption and enthusiasm from the developer community. Hadoop died about a decade ago, leaving a legacy of HDFS clusters in enterprises and a collection of companies dedicated to making money from them. At present, HDFS has lost its luster, as enterprises realize that there are better distributed storage alternatives [124]. Meanwhile, distributed RDBMSs are thriving, especially in the cloud.

Some aspects of MR system implementations related to scalability, elasticity, and fault tolerance are carried over into distributed RDBMSs. MR also brought about the revival of shared-disk architectures with disaggregated storage, subsequently giving rise to open-source file formats and data lakes (see Sec. 3.3). Hadoop's limitations opened the door for other data processing platforms, namely Spark [201] and Flink [109]. Both systems started as better implementations of MR with procedural APIs but have since added support for SQL [105].

2.2 Key/Value Stores

The key/value (KV) data model is the simplest model possible. It represents the following binary relation:

```
(key,value)
```

A KV DBMS represents a collection of data as an associative array that maps a key to a value. The value is typically an untyped array of bytes (i.e., a blob), and the DBMS is unaware of its contents. It is up to the application to maintain the schema and parse the value into its corresponding parts. Most KV DBMSs only provide get/set/delete operations on a single value.

In the 2000s, several new Internet companies built their own shared-nothing, *distributed* KV stores for narrowly focused applications, like caching and storing session data. For caching, Memcached [131] is the most well-known example of this approach. Redis [67] markets itself as a Memcached replacement, offering a more robust query API with checkpointing support. For more persistent application data, Amazon created the Dynamo KV store in 2007 [125]. Such systems offer higher and more predictable performance, compared to a RDBMS, in exchange for more limited functionality.

The second KV DBMS category are *embedded* storage managers designed to run in the same address space as a higher-level application. One of the first standalone embedded KV DBMSs was BerkeleyDB from the early 1990s [170]. Recent notable entries include Google's LevelDB [37], which Meta later forked as RocksDB [68].

Discussion: Key/value stores provide a quick "out-of-the-box" way for developers to store data, compared to the more laborious effort required to set up a table in a RDBMS. Of course, it is dangerous to use a KV store in a complex application that requires more than just a binary relation. If an application requires multiple fields in a record, then KV stores are probably a bad idea. Not only must the application parse record fields, but also there are no secondary indexes to retrieve other fields by value. Likewise, developers must implement joins or multi-get operations in their application.

BAD FOR COMPLEX DATA

To deal with these issues, several systems began as a KV store and then morphed into a more feature-rich record store. Such systems replace the opaque value with a semi-structured value, such as a JSON document. Examples of this transition are Amazon's DynamoDB [129] and Aerospike [9]. It is not trivial to re-engineer a KV store to make it support a complex data model, whereas RDBMSs easily emulates KV stores without any changes. If an application needs an embedded DBMS, there are full-featured choices available today, including SQLite [71] and DuckDB [180]. Hence, a RDBMS may be a better choice, even for simple applications, because they offer a path forward if the application's complexity increases.

One new architecture trend from the last 20 years is using embedded KV stores as the underlying storage manager for full-featured DBMSs. Prior to this, building a new DBMS requires engineers to build a custom storage manager that is natively integrated in the DBMS. MySQL was the first DBMS to expose an API that allowed developers to replace its default KV storage manager. This API enabled Meta to build RocksDB to replace InnoDB for its massive fleet of MySQL databases. Similarly, MongoDB discarded their ill-fated MMAP-based storage manager in favor of WiredTiger's KV store in 2014 [120, 138]. Using an existing KV store allows developers to write a new DBMS in less time.

KV IS GOOD BASELINE TECH

2.3 Document Databases

The document data model represents a database as a collection of record objects. Each document contains a hierarchy of field/value pairs, where each field is identified by a name and a field's value can be either a scalar type, an array of values, or another document. The following example in JSON is a customer document that contain a nested list of purchase order records with their corresponding order items.

```
{ "name": "First Last",
  "orders": [ { "id": 123, "items": [...] },
              { "id": 456, "items": [...] }, ] }
```

Document data models have been an active field of effort for several decades. This has given rise to data formats like SGML [117] and XML [118]. Despite the buzz with XML databases in the late 1990s, we correctly predicted in 2005 they would not supplant RDBMSs [188]. JSON has since overtaken XML to become the standard for data exchange for web-based applications. JavaScript's popularity with developers and the accompanying ubiquity of JSON led several companies to create document-oriented systems that natively stored JSON in the 2000s.

The inability of OLTP RDBMSs to scale in the 2000s ushered in dozens of document DBMSs that marketed themselves using the catchphrase *NoSQL* [110]. There were two marketing messages for such systems that resonated with developers. First, SQL and joins are slow, and one should use a "faster" lower-level, record-at-a-time interface. Second, ACID transactions are unnecessary for modern applications, so the DBMS should only provide weaker notion of it (i.e., BASE [179]).

Because of these two thrusts, NoSQL came to stand for a DBMS that stored records or documents as JSON, supported a lower-level API, and weak or non-existent transactions. There are dozens of such systems, of which MongoDB [41] is the most popular.

Discussion: Document DBMSs are essentially the same as object-oriented DBMSs from the 1980s and XML DBMSs from the late 1990s. Proponents of document DBMSs make the same argument as their OO/XML predecessors: storing data as documents removes the impedance mismatch between how application OO code interacts with data and how relational databases store them. They also claim that denormalizing entries into nested structures is better for performance because it removes the need to dispatch multiple queries to retrieve data related to a given object (i.e., "N+1 problem" in ORMs). The problems with denormalization/prejoining is an old topic that dates back to the 1970s [116]: (1) if the join is not one-to-many, then there will be duplicated data, (2) prejoins are not necessarily faster than joins, and (3) there is no data independence.

DENORM DATA HAS LOTS OF DOWNSIDES

WHAT IS SECONDARY INDEX?

Despite strong protestations that SQL was terrible, by the end of the 2010s, almost every NoSQL DBMS added a SQL interface. Notable examples include DynamoDB PartiQL [56], Cassandra CQL [15], Aerospike AQL [9], and Couchbase SQL++ [72]. The last holdout was MongoDB, but they added SQL for their Atlas service in 2021 [42]. Instead of supporting the SQL standard for DDL and DML operations, NoSQL vendors claim that they support their own proprietary query language derived or inspired from SQL. For most applications, these distinctions are without merit. Any language differences between SQL and NoSQL derivatives are mostly due to JSON extensions and maintenance operations.

NOSQL REINVENTED SQL IN THE END

Many of the remaining NoSQL DBMSs also added strongly consistent (ACID) transactions (see Sec. 3.4). As such, the NoSQL message has morphed from "Do not use SQL – it is too slow!" to "Not only SQL" (i.e., SQL is fine for some things).

Adding SQL and ACID to a NoSQL DBMS lowers their intellectual distance from RDBMSs. The main differences between them seems to be JSON support and the fact that NoSQL vendors allow "schema later" databases. But the SQL standard added a JSON data type and operations in 2016 [165, 178]. And as RDBMSs continue to improve their "first five minutes" experience for developers, we believe that the two kinds of systems will soon be effectively identical.

Higher level languages are almost universally preferred to record-at-a-time notations as they require less code and provide greater data independence. Although we acknowledge that the first SQL optimizers were slow and ineffective, they have improved immensely in the last 50 years. But the optimizer remains the hardest part of building a DBMS. We suspect that this engineering burden was a contributing factor to why NoSQL systems originally chose to not support SQL.

2.4 Column-Family Databases

There is another category of NoSQL systems that uses a data model called *column-family* (aka *wide-column*). Despite its name, column-family is not a columnar data model. Instead, it is a reduction of the document data model that only supports one level of nesting instead of arbitrary nesting; it is relation-like, but each record can have optional attributes, and cells can contain an array of values. The following example shows a mapping from user identifier keys to JSON documents that contain each user's varying profile information:

```
User1000 → { "name": "Alice",
             "accounts": [ 123, 456 ],
             "email": "xxx@xxx.edu" }
User1001 → { "name": "Bob",
             "email": [ "yyy@yyy.org", "zzz@zzz.com" ] }
```

The first column-family model DBMS was Google's BigTable in 2004 [111]. Instead of adopting SQL and emerging columnar storage, Google used this data model with procedural client APIs. Other systems adopted the column-family model in an attempt to copy Google's bespoke implementation. Most notable are Cassandra [14] and HBase [28]. They also copied BigTable's limitations, including the lack of joins and secondary indexes.

Discussion: All our comments in Sec. 2.3 about the document model are also applicable here. In the early 2010s, Google built RDBMSs on top of BigTable, including MegaStore [99] and the first version of Spanner. Since then, Google rewrote Spanner to remove the BigTable remnants [98], and it is now the primary database for many of its internal applications. Several NoSQL DBMSs deprecated their proprietary APIs in favor of SQL but still retain their non-relational architectures. Cassandra replaced their Thrift-API with a SQL-like language called CQL [15], and HBase now recommends the Phoenix SQL-frontend [57]. Google still offers BigTable as a cloud service, but the column-family model is a singular outlier with the same disadvantages as NoSQL DBMSs.

2.5 Text Search Engines

Text search engines have existed for a long time, beginning with the seminal SMART system in the 1960s [184]. SMART pioneered information retrieval and the vector space model, now nearly universal in modern search engines, by tokenizing documents into a "bag of words" and then building full-text indexes (aka inverted indexes) on those tokens to support queries on their contents. The system was also cognizant of noise words (e.g., "the", "a"), synonyms (e.g., "The Big Apple" is a synonym for "New York City"), salient keywords, and distance (e.g., "drought" often appears close to "climate change").

VECTOR SEARCH FROM 1960 WOW

The leading text search systems today include Elasticsearch [23] and Solr [70], which both use Lucene [38] as their internal search library. These systems offer good support for storing and indexing text data but offer none-to-limited transaction capabilities. This limitation means that a DBMS has to recover from data corruption by rebuilding the document index from scratch, which results in significant downtime.

All the leading RDBMSs support full-text search indexes, including Oracle [52], Microsoft SQL Server [52], MySQL [43], and PostgreSQL [62]. Their search features have improved recently and are generally on par with the special-purpose systems above. They also have the advantage of built-in transaction support. But their integration of search operations in SQL is often clunky and differs between DBMSs.

NEAT :D

Discussion: Text data is inherently unstructured, which means that there is no data model. Instead, a DBMS seeks to extract structure (i.e., meta-data, indexes) from text to avoid "needle in the haystack" sequential searches.

There are three ways to manage text data in application. First, one can run multiple systems, such as Elasticsearch for text and a RDBMS for operational workloads. This approach allows one to run "best of breed" systems but requires additional ETL plumbing to push data from the operational DBMS to the text DBMS and to rewrite applications to route queries to the right DBMSs based on their needs. Alternatively, one can run a RDBMS with good text-search integration capabilities but with divergent APIs in SQL. This latter issue is often overcome by application frameworks that hide this complexity (e.g., Django Haystack [20]). The third option is a polystore system [187] that masks the system differences via middleware that exposes a unified interface.

Inverted index-centric search engines based on SMART are used for exact match searches. These methods have been supplanted in recent years by similarity search using ML-generated embeddings (see Sec. 2.7).

2.6 Array Databases

There are many areas of computing where arrays are an obvious data representation. We use the term "array" to mean all variants of them [182]: *vectors* (one dimension – see Sec. 2.7), *matrices* (two dimensions), and *tensors* (three or more dimensions). For example, scientific surveys for geographic regions usually represent data as a multi-dimensional array that stores sensor measurements using location/time-based coordinates:

TIL TENSOR

```
(latitude, longitude, time, [vector-of-values])
```

Several other data sets look like this, including genomic sequencing and computational fluid dynamics. Arrays are also the core of most ML data sets.

Although array-based programming languages have existed since the 1960s (APL [142]), the initial work on array DBMSs began in the 1980s. PICDMS is considered to be the first DBMS implementation using the array data model [114]. The two oldest array DBMSs still being developed today are Rasdaman [66, 103] and kdb+ [34]. Newer array DBMSs include SciDB [54, 191] and TileDB [76]. HDF5 [29] and NetCDF [46] are popular array file formats for scientific data.

There are several system challenges with storing and querying real-world array data sets. Foremost is that array data does not always align to a regular integer grid; for example, geospatial data is often split into irregular shapes. An application can map such grids to integer coordinates via metadata describing this mapping [166]. Hence, most applications maintain array and non-array data together in a single database.

Unlike row- or column-based DBMSs, querying array data in arbitrary dimensions presents unique challenges. The difficulty arises from storing multi-dimensional array data on a linear physical storage medium like a disk. To overcome these challenges, array DBMSs must employ indexing and data structures to support efficient traversal across array dimensions.

STILL TRUE?

Discussion: Array DBMSs are a niche market that has only seen adoption in specific verticals (we discuss vector DBMSs next). For example, they have considerable traction in the genomics space. HDF5 is popular for satellite imagery and other gridded scientific data. But business applications rarely use dedicated array DBMSs, which is necessary for any product to survive. No major cloud provider offers a hosted array DBMS service, meaning they do not see a sizable market.

The challenge that array DBMS vendors have always faced is that the SQL includes support for ordered arrays as first-class data types (despite this being against the original RM proposal [115]). The first proposal to extend the unordered set-based RM with ordered rasters was in 1993 [155]. An early example of this was Illustra's temporal (one-dimensional) data plugin [31]. SQL:1999 introduced limited support for single-dimension, fixed-length array data types. SQL:2003 expanded to support nested arrays without a predefined maximum cardinality. Later entrants include Oracle Georaster [4] and Teradata [73]. Data cubes are special-purpose arrays [135], but columnar RDBMSs have eclipsed them for OLAP workloads because of their better flexibility and lower engineering costs [113].

More recently, the SQL:2023 standard includes support for true multi-dimensional arrays (SQL/MDA) that is heavily inspired by Rasdaman's RQL [166]. This update allows SQL to represent arrays with arbitrary dimensions using integer-based coordinates. In effect, this allows data cubes to exist in a SQL framework, but columnar DBMSs now dominate this market.

ARRAYS ARE COOL
BUT WIDE-COLUMN CAN DO IT

2.7 Vector Databases

Similar to how the column-family model is a reduction of the document model, the vector data model simplifies the array data model to one-dimensional rasters. Given that vector DBMSs are attracting the most attention right now from developers and investors (similar to the NoSQL fad), it is necessary to discuss them separately. The reason for this interest is because developers use them to store single-dimension *embeddings* generated from AI tools. These tools use learned transformations to convert a record's data (e.g., text, image) into a vector representing its latent semantics. For example, one could convert each Wikipedia article into an embedding using Google BERT and store them in a vector database along with additional article meta-data:

LOL

```
(title, date, author, [embedding-vector])
```

The size of these embedding vectors range from 100s of dimensions for simple transformers to 1000s for high-end models; these sizes will obviously grow over time with the development of more sophisticated models.

INTERESTING BIKOZ AI

The key difference between vector and array DBMSs is their query patterns. The former are designed for similarity searches that find records whose vectors have the shortest distance to a given input vector in a high-dimensional space. The input vector is another embedding generated with the same transformer used to populate the database. Unlike array DBMSs, applications do not use vector DBMSs to search for matches at an offset in a vector nor extract slices across multiple vectors. Instead, the dominant use case is this similarity search.

To avoid brute force scans for finding the most similar records, vector DBMSs build indexes to accelerate approximate nearest neighbor (ANN) searches. Applications issue queries with predicates on both the embedding index and non-embedding attributes (i.e., metadata). The DBMS then chooses whether to use the non-embedding predicate on records before (pre-filter) or after (post-filter) the vector search.

There are dozens of new DBMSs in this emerging category, with Pinecone [58], Milvus [40], and Weaviate [84] as the leading systems. Text search engines, including Elasticsearch [23], Solr [70], and Vespa [79], expanded their APIs to support vector search. Other DBMSs rebranded themselves as vector databases to jump on the bandwagon, such as Kdb+ [34].

NEAT THAT SAME DBS CAN GENERATE EMBEDDINGS

One compelling feature of vector DBMSs is that they provide better integration with AI tools (e.g., ChatGPT [16], LangChain [36]) than RDBMSs. These systems natively support transforming a record's data into an embedding upon insertion using these tools and then uses the same transformation to convert a query's input arguments into an embedding to perform the ANN search; other DBMSs require the application to perform these transformations outside of the database.

Discussion: Unlike array DBMSs that require a customized storage manager and execution engine to support efficient operations on multi-dimensional data, vector DBMSs are essentially document-oriented DBMSs with specialized ANN indexes. Such indexes are a feature, not the foundation of a new system architecture.

After LLMs became "mainstream" with ChatGPT in late 2022, it took less than one year for several RDBMSs to add their own vector search extensions. In 2023, many of the major RDBMSs added vector indexes, including Oracle [7], SingleStore [137], Rockset [8], and Clickhouse [157]. Contrast this with JSON support in RDBMSs. NoSQL systems like MongoDB and CouchDB became popular in the late 2000s and it took several years for RDBMSs to add support for it.

There are two likely explanations for the quick proliferation of vector indexes. The first is that similarity search via embeddings is such a compelling use case that every DBMS vendor rushed out their version and announced it immediately. The second is that the engineering effort to introduce a new index data structure is small enough that it did not take that much work for the DBMS vendors to add vector search. Most of them did not write their vector index from scratch and instead integrated an open-source library (e.g., pgVector [145], DiskANN [19], FAISS [24]).

We anticipate that vector DBMSs will undergo the same evolution as document DBMSs by adding features to become more relational-like (e.g., SQL, transactions, extensibility). Meanwhile, relational incumbents will have added vector indexes to their already long list of features and moved on to the next emerging trend.

2.8 Graph Databases

There has been a lot of academic and industry interest in the last decade in graph databases [183]. Many applications use knowledge graphs to model semi-structured information. Social media applications inherently contain graph-oriented relationships ("likes", "friend-of"). Relational design tools provide users with an entity-relationship (ER) model of their database. An ER diagram is a graph; thus, this paradigm has clear use cases.

PROPERTY GRAPHS MOST COMMON

The two most prevalent approaches to represent graphs are (1) the resource description framework (RDF) and (2) property graphs [126]. With property graphs, the DBMS maintains a directed multi-graph structure that supports key/value labels for nodes and edges. RDF databases (aka triplestores) only model a directed graph with labeled edges. Since property graphs are more common and are a superset of RDF, we will only discuss them. We consider two use cases for graph DBMSs and discuss the problems that will limit their adoption.

The first category of systems are for operational / OLTP workloads: an application, for example, adds a friend link in the database by updating a single record, presumably in a transactional manner. Neo4j [44] is the most popular graph DBMS for OLTP applications. It supports edges using pointers (as in CODASYL) but it does not cluster nodes with their "parent" or "offspring". Such an architecture is advantageous for traversing long edge chains since it will do pointer chasing, whereas a RDBMS has to do this via joins. But their potential market success comes down to whether there are enough "long chain" scenarios that merit forgoing a RDBMS.

SIMPLER/FASTER THAN MANY JOINS

The second use case is analytics, which seeks to derive information from the graph. An example of this scenario is finding which user has the most friends under 30 years old. Notable entries like Tigergraph [74] and JanusGraph [32] focus on query languages and storage on a graph DBMS. Other systems, such as Giraph [26] and Turi [78] (formerly Graphlab [27]) provide a computing fabric to support parallel execution of graph-oriented programs, typically written by a user.

Unlike queries in relational analytics that are characterized by chains of joins, queries for graph analytics contain operations like shortest path, cut set, or clique

ITS A FEATURE
NOT A PRODUCT

determination. Algorithm choice and data representation will determine a DBMS's performance. This argues for a computing fabric that allows developers to write their own algorithms using an abstraction that hides the underlying system topology. However, previous research shows that distributed algorithms rarely outperform single-node implementations because of communication costs [160]. A better strategy is to compress a graph into a space-efficient data structure that fits in memory on a single node and then run the query against this data structure. All but the largest graph databases are probably best handled this way.

Discussion: Regardless of whether a graph DBMS targets OLTP or OLAP workloads, the key challenge these systems have to overcome is that it is possible to simulate a graph as a collection of tables:

```
Node (node_id, node_data)
Edge (node_id_1, node_id_2, edge_data)
```

This means that RDBMSs are always an option to support graphs. But "vanilla" SQL is not expressive enough for graph queries and thus require multiple client-server roundtrips for traversal operations.

RDBMS OK FOR GRAPH
SQL LESS SO

Some RDBMSs, including MSSQL [3] and Oracle [50], provide built-in SQL extensions that make storing and querying graph data easier. Other DBMSs use a translation layer on top of relations to support graph-oriented APIs. Amazon Neptune [45] is a graph-oriented veneer on top of Aurora MySQL. Apache AGE provides an OpenCypher interface on top of PostgreSQL [10].

More recently, SQL:2023 introduced property graph queries (SQL/PGQ) for defining and traversing graphs in a RDBMS [196]. The syntax builds on existing languages (e.g., Neo4j's Cypher [49], Oracle's PGQL [51], and TigerGraph's GSQL [75]), and shares aspects of the emerging GQL standard [126]. Thus, SQL/PGQ further narrows the functionality difference between RDBMSs and native graph DBMSs.

The question is whether graph DBMS vendors can make their specialized systems fast enough to overcome the above disadvantages. There have been several performance studies showing that graph simulation on RDBMSs outperform graph DBMSs [130, 143]. More recent work showed how SQL/PGQ in DuckDB outperforms a leading graph DBMS by up to 10× [196]. This trend will continue with further improvements in worst-case optimal joins [132, 168] and factorized execution algorithms [100] for graph queries in RDBMSs.

DAMN DUCKDB
IS MAGIC

2.9 Summary

A reasonable conclusion from the above section is that non-SQL, non-relational systems are either a niche market or are fast becoming SQL/RM systems. Specifically:

- **MapReduce Systems:** They died years ago and are, at best, a legacy technology at present.
- **Key-value Stores:** Many have either matured into RM systems or are only used for specific problems. These can generally be equaled or beaten by modern high-performance RDBMSs.
- **Document Databases:** Such NoSQL systems are on a collision course with RDBMSs. The differences between the two kinds of systems have diminished over time and should become nearly indistinguishable in the future.
- **Column-Family Systems:** These remain a niche market. Without Google, this paper would not be talking about this category.
- **Text Search Engines:** These systems are used for text fields in a polystore architecture. It would be valuable if RDBMSs had a better story for search so these would not have to be a separate product.
- **Array Databases:** Scientific applications will continue to ignore RDBMSs in favor of bespoke array systems. They may become more important because RDBMSs cannot efficiently store and analyze arrays despite new SQL/MDA enhancements.
- **Vector Databases:** They are single-purpose DBMSs with indexes to accelerate nearest-neighbor search. RM DBMSs should soon provide native support for these data structures and search methods using their extendable type system that will render such specialized databases unnecessary.
- **Graph Databases:** OLTP graph applications will be largely served by RDBMSs. In addition, analytic graph applications have unique requirements that are best done in main memory with specialized data structures. RDBMSs will provide graph-centric APIs on top of SQL or via extensions. We do not expect specialized graph DBMSs to be a large market.

Beyond the above, there are also proposals to rebrand previous data models as something novel. For example, *graph-relational* [158] is the same as the semantic data model [202]. Likewise, *document-relational* is the document model with foreign keys [199]. Others provide a non-SQL veneer over a RDBMS (e.g., PRQL [64], Malloy [39]). Although these languages deal with some of SQL's shortcomings, they are not compelling enough to overcome its entrenched userbase and ecosystem.

3 System Architectures

There have been major new ideas in DBMS architectures put forward in the last two decades that reflecting changing application and hardware characteristics. These ideas range from terrific to questionable, and we discuss them in turn.

3.1 Columnar Systems

To understand the appeal of columnar DBMSs, we need to explain the origins of the data warehouse (OLAP) market. Beginning in the mid-1990s, enterprises started collecting their customer facing (usually sales) data. Brick-and-mortar retailers (e.g., Walmart) were at forefront of constructing historical sales databases. These companies generally found that a sales data warehouse would pay for itself in better stock ordering and rotation decisions within six months. Such customer facing databases are now omnipresent in enterprises.

Data warehouse applications have common properties that are distinct from OLTP workloads:

1. They are historical in nature (i.e., they are loaded periodically and then are read-only).
2. Organizations retain everything as long as they can afford the storage — think terabytes to petabytes.
3. Queries typically only access a small subset of attributes from tables and are ad-hoc in nature.

Ralph Kimball was an early proponent of star schema data modelling for data warehouses [148, 149]. The idea was to construct a *fact table* that held item-level transactional data. The classic example is a fact table that contains a record for every item purchased in a retail enterprise. Then, one surrounds the fact table with *dimension tables* that contain common information factored out from the fact table to save space. Again, in a retail setting, these dimension tables would include information about customers, products, stores, and time.

Organizing the DBMS's storage by columns instead of rows has several benefits [87]. First, compressing columnar data is more effective than row-based data because there is a single value type in a data block often many repeated bytes. Second, a Volcano-style engine executes operators once per row. In contrast, a column-oriented engine has an inner loop that processes a whole column using vectorized instructions [106, 147]. Lastly, row stores have a large header for each record (e.g., 20 bytes) to track nulls and versioning meta-data, whereas column stores have minimal storage overhead per record.

Discussion: Over the last two decades, all vendors active in the data warehouse market have converted their offerings from a row store to a column store. This transition brought about significant changes in the design of DBMSs. In addition, several new vendors have entered the market in the last two decades with column store offerings, for example Amazon's Redshift [94] and Google's BigQuery [162] along with offerings from independent companies (e.g., Snowflake [121]).

In summary, column stores are new DBMS implementations with specialized optimizers, executors, and storage formats. They have taken over the data warehouse marketplace because of their superior performance.

3.2 Cloud Databases

The rise of cloud platforms in the late 2000s has also greatly affected the implementation (and sales model) of DBMSs. Initial cloud DBMS offerings repackaged on-prem systems into managed VMs with direct-attached storage. But over the last 20 years, networking bandwidth has increased much faster than disk bandwidth, making network attached storage (NAS) attractive as an alternative to attached storage. This has caused a profound rethinking of DBMS architectures for the cloud.

All major cloud vendors offer NAS via *object stores* (e.g., Amazon S3) with some DBMS functionality (e.g., replication, filtering). Beyond better economics compared to direct-attached storage, object stores have several advantages that compensate for the cost of the added network link. First, because the compute nodes are disconnected from the storage nodes, a system can provide per-query elasticity; the DBMS can add new compute nodes dynamically without having to reshuffle data. It also allows the DBMS to use different hardware for its storage nodes than compute nodes. Second, the system can reassign compute nodes to other tasks if a DBMS is underutilized. On the other hand, in a shared-nothing DBMS, a node must always be online to handle incoming query requests. Lastly, pushing down computation into the storage nodes is possible (and generally advantageous). This execution strategy is known as "pushing the query to the data" versus "pulling the data to the query" and is well understood in DBMSs.

Generally, the first two ideas are called "serverless computing", and was introduced for cloud-native DBMSs by Snowflake [121]. Other vendors have moved or are in the process of moving to a serverless environment for their cloud offerings. Effective utilization of this model requires a hosted multi-node environment in which multiple DBMS customers are grouped onto the same node(s) with a multi-tenant execution scheme.

Discussion: The advent of cloud databases is another example of "what goes around comes around". Multi-node shared-disk DBMSs are an old idea that historically tended not to work out well. However, it is back in vogue with technology change (faster networking) and moving to the cloud. In addition, time-sharing services were popular in the 1970s when computers were big and expensive. Cloud platforms are big time-sharing services, so the concept is back after a few decades. Since enterprises are moving everything possible to the cloud, we expect this shared-disk to dominate DBMS architectures. Hence, we do not foresee shared-nothing architectures resurfacing in the future.

The cloud has profoundly impacted DBMSs, causing them to be completely re-architected. The movement of computing from on-prem to the cloud generates a once-in-a-lifetime opportunity for enterprises to refactor codebases and remove bad historical technology decisions. A cloud environment also provides several benefits to vendors that are not possible with on-prem deployments. Foremost is that vendors can track usage trends for all their customers: they can monitor unexpected behavior, performance degradations, and usage patterns. Moreover, they can push incremental updates and code patches without disrupting service.

From a business perspective, open-source DBMSs face the danger of becoming too popular and being monetized by the major cloud providers. The public spats between Amazon and ISVs like MongoDB [153] and Elasticsearch [101] are notable examples.

3.3 Data Lakes / Lakehouses

Another trend that the cloud platforms fomented is the movement away from monolithic, dedicated data warehouses for OLAP workloads and towards *data lakes* backed by object stores. With legacy data warehouses, organizations load data into the DBMS, which the system stashes in managed storage with proprietary formats. Vendors viewed their DBMSs as the "gatekeepers" for all things related to data in an organization. However, this has not been the model of many organizations, especially technology companies, for the last decade.

With a data lake architecture, applications upload files to a distributed object store, bypassing the traditional route through the DBMS [167]. Users then execute queries and processing pipelines on these accumulated files using a *lakehouse* (a portmanteau of data warehouse and data lake) execution engine [93]. These lakehouse systems provide a unified infrastructure supporting SQL and non-SQL workloads. The latter is crucial as the last decade has shown that data scientists and ML practitioners typically use Python-based notebooks that use Panda's DataFrame API [159] to access data instead of SQL. Several projects leverage DBMS methods to optimize DataFrame processing, including Dask [181], Polars [61], Modin [177], and Bodo [198].

Instead of using DBMS-specific proprietary file formats or inefficient text-based files (e.g., CSV, JSON), applications write data to data lakes using open-source, disk-resident file formats [203]. The two most popular formats are Twitter/Cloudera's Parquet [55] and Meta's ORC [53, 140]. Both of them borrow techniques from earlier columnar storage research, such as PAX [90], compression [87], and nested-data (JSON) shredding [121, 161]. Apache Arrow [11] is a similar binary format for exchanging in-memory data between systems. Open-source libraries for reading/writing these formats allow disparate applications to create data files that other systems then parse and consume, thereby enhancing data sharing across services and business units.

DATA LAKE IS SQL ON FILES
NOT JUST BIG DB

Discussion: Data lakes are the successor to "Big Data" movement from the early 2010s, partly led by the popularity of MR systems (Sec. 2.1) and column stores (Sec. 3.1). At first glance, a data lake seems like a terrible idea for an organization: allowing any application to write arbitrary files into a centralized repository without any governance is a recipe for integrity, discovery, and versioning problems [167]. Lakehouses provide much-needed control over these environments to help mitigate many problems with meta-data, caching, and indexing services [93]. Additional middleware that tracks new data and supports transactional updates, such as Delta Lake [92], Iceberg [6], and Hudi [5], make lakehouses look more like a traditional data warehouse.

Data lakes introduce new challenges to query optimization. DBMSs have always struggled with acquiring precise statistics on data, leading to poor query plan choices [154]. However, a data lake system may completely lack statistics on newly ingested data files. Consequently, incorporating adaptive query processing strategies is imperative in the cloud to enable a DBMS to dynamically modify query plans during execution based on observed data characteristics [97, 105, 163].

All the major cloud vendors now offer some variation of a managed data lake service. Since data lake systems backed by object stores are much cheaper per gigabyte than proprietary data warehouses, the legacy OLAP vendors (e.g., Teradata, Vertica) have extended their DBMSs to support reading data from object stores in response to this pricing pressure. Several independent systems are also in this space, including Databricks [105], Dremio [21], PrestoDB [63], and Trino [77].

3.4 NewSQL Systems

In the late 2000s, there were multiple distributed NoSQL DBMSs available designed to scale horizontally to support online applications with large number of concurrent users [110]. However, many organizations could not use these NoSQL systems because their applications could not give up strong transactional requirements. But the existing RDBMSs (especially open-source ones) were not able to (natively) scale across multiple machines. In response, *NewSQL* systems arrived in the early 2010s seeking to provide the scalability of NoSQL systems for OLTP workloads while still supporting SQL [95, 171]. In other words, these new systems sought to achieve the same scalability of NoSQL DBMSs from the 2000s but still keep the RM and ACID transactions of the legacy DBMSs from the 1990s.

SCALE + ACID

There were two main groups of NewSQL systems. The first was in-memory DBMSs, including H-Store [144, 189] (commercialized as VoltDB [83]), SingleStore [69], Microsoft Hekaton [128], and HyPer [146]. Other startup offerings included disk-oriented, distributed DBMSs like NuoDB [47] and Clustrix [17].

Discussion: There has yet to be a dramatic uptake in NewSQL DBMS adoption [96]. The reason for this lackluster interest is that existing DBMSs were good enough for the time, which means organizations are unwilling to take on the costs and risk of migrating existing applications to newer technologies. Companies are more risk-

OLTP MORE
RISK-AVERSE

averse with changing OLTP DBMSs than with OLAP. If an OLTP DBMS fails, companies cannot execute the transactions they need to generate revenue. In contrast, an OLAP DBMS failure could be limited to temporarily inconveniencing an analyst or data scientist.

There were other restrictions in NewSQL DBMSs, such as only supporting a subset of standard SQL or bad performance on multi-node transactions. Some NewSQL products, like Microsoft's Hekaton, were only available as an extension to a legacy DBMS, requiring the faster engine to use the slower DBMS's interfaces.

NewSQL vendors also incorrectly anticipated that in-memory DBMS adoption would be larger in the last decade. Flash vendors drove down costs while improving storage densities, bandwidth, and latencies. Higher DRAM costs and the collapse of persistent memory (e.g., Intel Optane) means that SSDs will remain dominant for OLTP DBMSs.

TRANSACTIONS GOOD

The aftermath of NewSQL is a new crop of distributed, transactional SQL RDBMSs. These include TiDB [141], CockroachDB [195], PlanetScale [60] (based on the Vitess sharding middleware [80]), and YugabyteDB [86]. The major NoSQL vendors also added transactions to their systems in the last decade despite previously strong claims that they were unnecessary. Notable DBMSs that made the shift include MongoDB, Cassandra, and DynamoDB. This is of course due to customer requests that transactions are in fact necessary. Google said this cogently when they discarded eventual consistency in favor of real transactions with Spanner in 2012 [119].

3.5 Hardware Accelerators

There has been a hunt for a cost-effective hardware accelerator for DBMSs for the last 50 years. The promise is obvious: specialized hardware designed for a DBMS should easily outperform a conventional CPU.

In the 1980s, vendors fabricated custom hardware to accelerate DBMSs and marketed them as *database machines* [107]. Britton-Lee released the first commercial accelerator product (IDM/500) in 1981 [192] that contained a conventional CPU with a hardware accelerator that offloaded portions of a query's execution. This accelerator targeted a small subset of the execution path, and was not cost-effective. Teradata introduced its own database machine that provided network hardware for sorting in-flight tuples (Y-net [1]), but it was dropped for a software-only solution [85]. All other custom hardware DBMS acceleration during the 1980s failed.

Instead of building custom hardware for DBMSs, the last 20 years have been about using commodity hardware (FPGAs, GPUs) to accelerate queries. This is an enticing idea: a vendor can get the benefits of a DBMS accelerator without the cost of fabricating the hardware.

Netezza was one of the first FPGA-based DBMSs that started in the late 1990s as a fork of PostgreSQL. It used an FPGA to accelerate searches on disk-resident pages, but originally could not search in-memory pages. Netezza corrected this limitation in a later version [2]. Swarm64 attempted to sell a FPGA accelerator for PostgreSQL but switched to a software-only architecture without the FPGA before they were acquired [91]. Vitesse's Deepgreen DB [81] is the only remaining FPGA-enhanced DBMS available from an ISV.

There is more activity in the GPU-accelerated DBMS market. Notable GPU DBMSs include Kinetica [35], Sqream [35], Brytlyt [13], and HeavyDB [48]. If data does not fit in GPU memory, then query execution is bottlenecked on loading data into the device, thereby rendering the hardware's parallelization benefits moot.

Discussion: There are several conclusions that we can draw from the above analysis. First, these systems are all focused on the OLAP market and only for RDBMSs; there are essentially no data model implications to the discussion in this section. Also, OLAP workloads will continue to move aggressively to the cloud, but special-purpose hardware is not likely to find acceptance unless it is built by the cloud vendor.

Creating custom hardware just for a DBMS is not cost-effective for most companies. Commodity hardware avoids this problem but there is still the challenge of integrating the hardware into a DBMS. The reason why there are more GPU DBMSs than FPGA systems is because there are existing support libraries available for GPUs (e.g., Nvidia CUDA [169]). But cloud CPU-based compute resources are incredibly cheap due to economies of scale. The success of any accelerator is likely to be limited to on-prem databases, but this market is not growing at the same rate as cloud databases.

Even if one could get an accelerator to market that showed orders of magnitude improvement over existing technologies, that only solves half the problem needed for adoption and success. A hardware-only company must find somebody to add support for its accelerator in a DBMS. If the accelerator is an optional add-on to the DBMS, then adoption will be low and thus a DBMS vendor will not want to spend engineering time on supporting it. If the accelerator is a critical component of the DBMS, then no vendor would outsource the development of such an important part to an outside vendor.

The only place that custom hardware accelerators will succeed is for the large cloud vendors. They can justify the $50–100m R&D cost of custom hardware at their massive scale. They also control the entire stack (hardware and software) and can integrate their hardware at critical locations. Amazon did this already with their Redshift AQUA accelerators [102]. Google BigQuery has custom components for in-memory shuffles [89].

In spite of the long odds, we predict that there will be many attempts in this space over the next two decades.

HARDWARE DB ACCELERATORS
NOT WORTH IT UNLESS YOURE
CLOUD VENDOR

3.6 Blockchain Databases

As of this writing, a waning database technology fad is blockchains. These are decentralized log-structured databases (i.e., ledger) that maintain incremental checksums using some variation of Merkle trees. These incremental checksums are how a blockchain ensures that the database's log records are immutable: applications use these checksums to verify that previous database updates have not been altered.

The ideal use case for blockchain databases is peer-to-peer applications where one cannot trust anybody. There is no centralized authority that controls the ordering of updates to the database. Thus, blockchain implementations use a BFT commit protocol to determine which transaction to apply to the database next.

At the present time, cryptocurrencies (Bitcoin) are the only use case for blockchains. In addition, there have been attempts to build a usable DBMS on top of blockchains, notably Fluree [25], BigChainDB [12], and ResilientDB [136]. These vendors (incorrectly) promote the blockchain as providing better security and auditability that are not possible in previous DBMSs.

LOL SCAM

Discussion: We are required to place trust in several entities in today's society. When one sells a house, they trust the title company to manage the transaction. The only applications without real-world trust are dark web interactions (e.g., money laundering). Legitimate businesses are unwilling to pay the performance price (about five orders of magnitude) to use a blockchain DBMS. If organizations trust each other, they can run a shared distributed DBMS more efficiently without wasting time with blockchains. To the best of our knowledge, all the major cryptocurrency exchanges run their businesses off traditional RDBMSs and not blockchain systems.

Blockchain proponents make additional meaningless claims of achieving data resiliency through replication in a peer-to-peer environment. No sensible company would rely on random participants on the Internet as the backup solution for mission-critical databases.

There is possibly a (small) market for private blockchain DBMSs. Amazon's Quantum Ledger Database (QLDB) released in 2018 [65] provides the same immutable and verifiable update guarantees as a blockchain, but it is not decentralized (i.e., no BFT commit protocol). Amazon built QLDB after finding no compelling use case for a fully decentralized blockchain DBMS [108].

NOBODY NEEDS BLOCKCHAIN

3.7 Summary

The key takeaways from the major technological thrusts in database systems are as follows:

- **Columnar Systems:** The change to columnar storage revolutionized OLAP DBMS architectures.
- **Cloud Databases:** The cloud has upended the conventional wisdom on how to build scalable DBMSs. Except for embedded DBMSs, any product not starting with a cloud offering will likely fail.
- **Data Lakes / Lakehouses:** Cloud-based object storage using open-source formats will be the OLAP DBMS archetype for the next ten years.
- **NewSQL Systems:** They leverage new ideas but have yet to have the same impact as columnar and cloud DBMSs. It has led to new distributed DBMSs that support stronger ACID semantics as a counter to NoSQL's weaker BASE guarantees.
- **Hardware Accelerators:** We do not see a use case for specialized hardware outside of the major cloud vendors, though start-ups will continue to try.
- **Blockchain Databases:** An inefficient technology looking for an application. History has shown this is the wrong way to approach systems development.

LOL

4 Parting Comments

Our analysis of the last two decades in databases has several takeaways. Unfortunately, some of these are repeats of the warnings from the 2005 paper.

Never underestimate the value of good marketing for bad products. The database market is highly competitive and lucrative. This competition drives vendors to claim that their new technologies will solve all sorts of problems and change developers' lives for the better. Every developer has struggled with databases before, so they are especially amenable to such marketing. Inferior DBMS products have succeeded via strong marketing despite the existence of better options available at the time: Oracle did this in the 1980s, MySQL did this in the 2000s, and MongoDB did this in the 2010s. These systems got enough traction early on to buy them time to fix the engineering debt they accumulated earlier.

Beware of DBMSs from large non-DBMS vendors. One interesting aspect in the last ten years of databases is the trend of tech companies building DBMSs in-house that they then spin out as open-source projects. All these systems started life as purpose-built applications for a tech company. The company then releases the DBMS as an open-source project (often pushed to the Apache Foundation for stewardship) in hopes to achieve "free" development from external users.

Some times they come from large companies that can afford to allocate resources to developing new systems. Notable examples include Meta (Hive [197], Presto [63], Cassandra [14], RocksDB [68]) and LinkedIn (Kafka [33], Pinot [59], Voldemort [82]). Other systems are from start-ups building a data-intensive product where they felt the need to also build a DBMS. The most successful examples are 10gen (MongoDB) and PowerSet (HBase), but there also many failed endeavors.

This trend to avoid "not invented here" software is partly because many companies' promotion path favors engineers who make new internal systems, even if existing tools are sufficient. But this perversion led many teams without DBMS engineering experience to undertake building a new system. One should be wary of such systems when a company first open-sources them, as they are almost always immature technologies.

Do not ignore the out-of-box experience. One of the salient selling points of many non-relational DBMSs is a better "out-of-box" experience than RDBMSs. Most SQL systems require one first to create a database and then define their tables before they can load data. This is why data scientists use Python notebooks to analyze data files quickly. Every DBMS should, therefore, make it easy to perform in situ processing of local and cloud-storage files. DuckDB's rising popularity is partly due to its ability to do this well.

Vendors should also consider additional challenges that customers will inevitably face with databases, including physical design, knob tuning, schema design, and query tuning. There is a crucial need for what one of us calls "self-driving" DBMSs [173].

Developers need to query their database directly. Most OLTP applications created in the last 20 years primarily interact with databases via an abstraction layer, such as an endpoint API (e.g., REST, GraphQL) or an object-relational mapper (ORM) library. Such layers translate an application's high-level requests into database queries. ORMs also automatically handle maintenance tasks, such as schema migrations. One could argue that since OLTP developers never write raw SQL in their applications, it does not matter what data model their DBMS uses as these layers hide it.

ORMs are a vital tool for rapid prototyping. But they often sacrifice the ability to push logic into the DBMS in exchange for interoperability with multiple DBMSs. Developers fall back to writing explicit database queries to override the poor auto-generated queries. This is why using a RDBMS that supports SQL is the better choice.

The impact of AI/ML on DBMSs will be significant. How DBMSs should interact with modern AI/ML tools has recently become a crucial question, especially with the advent of LLMs (e.g., ChatGPT). Although this field is moving rapidly, we offer a few initial comments.

There is a resurgence in using natural languages (NLs) to query databases due to advancements in LLMs at converting NL to query code (e.g., SQL) [133]. Some have even suggested that such AI-powered query interfaces will render SQL obsolete. NL interfaces are an old research topic that dates back to the 1970s [139], but which historically has poor outcomes and thus little widespread use [88]. We acknowledge LLMs have impressive results for this task but caution those who think NL will replace SQL. Nobody will write OLTP applications using an NL, as most generate queries using ORMs. For OLAP databases, NL could prove helpful in constructing the initial queries for exploratory analysis. However, these queries should be exposed to a dashboard-like refinement tool since English and other NLs are rife with ambiguities and impreciseness.

There is a reluctance to depend on current LLM technology for decision-making inside the enterprise, especially with financial data. The biggest issue is that the output of an LLM is not explainable to a human. Second, LLM systems require more training data than "traditional" ML systems (e.g., random forests, Bayesian models). Companies generally cannot outsource the creation of training data for these models to unskilled people. For these reasons, the uptake of LLMs for enterprise data will be cautiously slow.

Lastly, there is a considerable amount of recent research on using AI/ML to optimize the DBMSs [174]. Examples include ML-oriented query optimizers [152, 156], configuration tuners [200, 204], and access methods [151, 193]. Although such ML-assisted optimizations are powerful tools to improve the performance of DBMSs, it does not obviate the need for high-quality systems engineering.

5 Conclusion

We predict that what goes around with databases will continue to come around in upcoming decades. Another wave of developers will claim that SQL and the RM are insufficient for emerging application domains. People will then propose new query languages and data models to overcome these problems. There is tremendous value in exploring new ideas and concepts for DBMSs (it is where we get new features for SQL). The database research community and marketplace are more robust because of it. However, we do not expect these new data models to supplant the RM.

NEW TOOLS RESEARCH
SQL TAKES BEST

Another concern is the wasted effort of new projects reimplementing the same components that are not novel but necessary to have a production-ready DBMS (e.g., config handlers, parsers, buffer pools). To accelerate the next generation of DBMSs, the community should foster the development of open-source reusable components and services [112, 176]. There are some efforts towards this goal, including for file formats (see Sec. 3.3), query optimization (e.g., Calcite [104], Orca [186]), and execution engines (e.g., DataFusion [18], Velox [175]). We contend that the database community should strive for a POSIX-like standard of DBMS internals to accelerate interoperability.

We caution developers to learn from history. In other words, stand on the shoulders of those who came before and not on their toes. One of us will likely still be alive and out on bail in two decades, and thus fully expects to write a follow-up to this paper in 2044.

References

[1] TeraData Forums. https://downloads.teradata.com/forum/database/what-is-the-difference-between-a-ynet-and-bynet, September 2011.

[2] Netezza TwinFin Architecture. https://www.iexpertify.com/learn/netezza-twinfin-architecture/#.YYq5_S1h17Y, April 2020.

[3] Graph processing with sql server and azure sql database. https://docs.microsoft.com/en-us/sql/relational-databases/graphs/sql-graph-overview, 2021.

[4] Georaster in oracle database. https://www.oracle.com/a/tech/docs/georaster-2021.pdf, mar 2021.

[5] Apache Hudi. https://hudi.apache.org/, 2023.

[6] Apache Iceberg. https://iceberg.apache.org/, 2023.

[7] Oracle introduces integrated vector database to augment generative ai and dramatically increase developer productivity. https://www.oracle.com/news/announcement/ocw-integrated-vector-database-augments-generative-ai-2023-09-19/, sep 2023.

[8] Introducing vector search on rockset. https://rockset.com/blog/introducing-vector-search-on-rockset/, apr 2023.

[9] Aerospike AQL. https://docs.aerospike.com/tools/aql, 2024.

[10] Apache AGE. https://age.apache.org, 2024.

[11] Apache Arrow. https://arrow.apache.org, 2024.

[12] BigchainDB. https://www.bigchaindb.com/, 2024.

[13] Brytlyt. https://brytlyt.io/, 2024.

[14] Apache Cassandra. https://cassandra.apache.org, 2024.

[15] The Cassandra Query Language (CQL). https://cassandra.apache.org/doc/latest/cassandra/cql/, 2024.

[16] ChatGPT Plugins. https://openai.com/blog/chatgpt-plugins, March 2024.

[17] Clustrix. https://clustrix.com, 2024.

[18] Apache Arrow DataFusion. https://arrow.apache.org/datafusion/, 2024.

[19] Microsoft DiskANN. https://github.com/microsoft/DiskANN, 2024.

[20] Django Haystack. https://django-haystack.readthedocs.io, 2024.

[21] Dremio. https://dremio.com/, 2024.

[22] Apache drill. https://drill.apache.org, 2024.

[23] Elasticsearch. https://www.elastic.co, 2024.

[24] FAISS – Facebook AI Similarity Search. https://ai.facebook.com/tools/faiss/, 2024.

[25] Fluree. https://flur.ee/, 2024.

[26] Apache Giraph. https://giraph.apache.org, 2024.

[27] Graphlab. https://en.wikipedia.org/wiki/GraphLab, 2024.

[28] Apache Hbase. https://hbase.apache.org, 2024.

[29] The hdf5 library & file format. https://www.hdfgroup.org/solutions/hdf5, 2024.

[30] Apache Hive. https://hive.apache.org, 2024.

[31] Informix extensions and datablade modules. https://www.ibm.com/docs/en/informix-servers/12.10?topic=informix-extensions-datablade-modules, 2024.

[32] Janusgraph. https://janusgraph.org/, 2024.

[33] Apache Kafka. https://kafka.apache.org/, 2024.

[34] kdb+. https://kx.com/, 2024.

[35] Kinetica. https://www.kinetica.com/, 2024.

[36] LangChain. https://langchain.com, 2024.

[37] LevelDB. https://github.com/google/leveldb, 2024.

[38] Apache Lucene. https://lucene.apache.org, 2024.

[39] Malloy - Experimental Language. https://github.com/looker-open-source/malloy, 2024.

[40] Milvus. https://milvus.io/, 2024.

[41] MongoDB. https://mongodb.com, 2024.

[42] Mongodb – querying with sql. https://docs.mongodb.com/datalake/admin/query-with-sql/, 2024.

[43] MySQL – InnoDB Full-Text Indexes. https://dev.mysql.com/doc/refman/8.0/en/innodb-fulltext-index.html, 2024.

[44] Neo4j. https://neo4j.com/, 2024.

[45] Amazon Neptune. https://aws.amazon.com/neptune/, 2024.

[46] Network Common Data Form (NetCDF). https://www.unidata.ucar.edu/software/netcdf/, 2024.

[47] Nuodb. https://nuodb.com, 2024.

[48] Heavydb. https://www.heavy.ai, 2024.

[49] openCypher. https://opencypher.org, 2024.

[50] Oracle graph database. https://www.oracle.com/database/graph/, 2024.

[51] PGQL – Property Graph Query Language. https://pgql-lang.org/, 2024.

[52] Oracle Text. https://www.oracle.com/database/technologies/datawarehouse-bigdata/text.html, 2024.

[53] Apache ORC. https://orc.apache.org/, 2024.

[54] Paradigm4 platform overview. https://www.paradigm4.com/technology/scidb-platform-overview/, 2024.

[55] Apache Parquet. https://parquet.apache.org/, 2024.

[56] Partiql – sql-compatible access to relational, semi-structured, and nested data. https://partiql.org/, 2024.

[57] Apache Phoenix. https://phoenix.apache.org, 2024.

[58] Pinecone. https://www.pinecone.io/, 2024.

[59] Apache Pinot. https://pinot.apache.org/, 2024.

[60] PlanetScale. https://planetscale.com/, 2024.

[61] Polars. https://www.pola.rs, 2024.

[62] PostgreSQL – Full Text Search. https://www.postgresql.org/docs/current/textsearch.html, 2024.

[63] PrestoDB. https://prestodb.io/, 2024.

[64] PRQL – A Proposal for a Better SQL. https://prql-lang.org/, 2024.

[65] Amazon Quantum Ledger Database (QLDB). https://aws.amazon.com/qldb/, 2024.

[66] The rasdaman raster array database. http://www.rasdaman.org, 2024.

[67] Redis. https://redis.io/, 2024.

[68] RocksDB. https://rocksdb.org, 2024.

[69] Singestore. https://www.singlestore.com/, 2024.

[70] Apache Solr. https://solr.apache.org/, 2024.

[71] SQLite. https://www.sqlite.org, 2024.

[72] Sql++ – the next-generation query language for managing json data. https://www.couchbase.com/sqlplusplus, 2024.

[73] Teradata – creating an array data type. https://docs.teradata.com/r/S0Fw2AVH8ff3MDA0wDOHlQ/un3kj~t3qMDO66LF4YXuiw, 2024.

[74] Tigergraph. https://www.tigergraph.com/, 2024.

[75] Tigergraph – gsql. https://www.tigergraph.com/gsql/, 2024.

[76] Tiledb. https://tiledb.com, 2024.

[77] Trino. https://trino.io/, 2024.

[78] Turi. http://turi.com/, 2024.

[79] Vespa. https://vespa.ai/, 2024.

[80] Vitess. https://vitess.io, 2024.

[81] Vitesse Deepgreen DB. https://www.vitessedata.com/products/deepgreen-db/, 2024.

[82] Project Voldemort. https://www.project-voldemort.com, 2024.

[83] Voltdb. https://www.voltactivedata.com/, 2024.

[84] Weaviate. https://weaviate.io, 2024.

[85] Dbc 1012. https://en.wikipedia.org/wiki/DBC_1012, 2024.

[86] YugabyteDB. https://www.yugabyte.com/, 2024.

[87] D. J. Abadi. *Query Execution in Column-Oriented Database Systems*. PhD thesis, MIT, 2008.

[88] K. Affolter, K. Stockinger, and A. Bernstein. A comparative survey of recent natural language interfaces for databases. *VLDB J.*, 28(5):793–819, 2019. doi: 10.1007/s00778-019-00567-8.

[89] H. Ahmadi. In-memory query execution in google bigquery. `https://cloud.google.com/blog/products/bigquery/in-memory-query-execution-in-google-bigquery`, Aug 2016.

[90] A. Ailamaki, D. J. DeWitt, M. D. Hill, and M. Skounakis. Weaving relations for cache performance. In *VLDB*, volume 1, pages 169–180, 2001.

[91] G. Anadiotis. Open source postgresql on steroids: Swarm64 database acceleration software for performance improvement and analytics. `https://www.zdnet.com/article/open-source-postgresql-on-steroids-swarm64-database-acceleration-software-for-performance-improvement-and-analytics/`, apr 2023.

[92] M. Armbrust, T. Das, L. Sun, B. Yavuz, S. Zhu, M. Murthy, J. Torres, H. van Hovell, A. Ionescu, A. Łuszczak, et al. Delta lake: high-performance acid table storage over cloud object stores. *Proceedings of the VLDB Endowment*, 13(12):3411–3424, 2020.

[93] M. Armbrust, A. Ghodsi, R. Xin, and M. Zaharia. Lakehouse: a new generation of open platforms that unify data warehousing and advanced analytics. In *Proceedings of CIDR*, page 8, 2021.

[94] N. Armenatzoglou, S. Basu, N. Bhanoori, M. Cai, N. Chainani, K. Chinta, V. Govindaraju, T. J. Green, M. Gupta, S. Hillig, E. Hotinger, Y. Leshinksy, J. Liang, M. McCreedy, F. Nagel, I. Pandis, P. Parchas, R. Pathak, O. Polychroniou, F. Rahman, G. Saxena, G. Soundararajan, S. Subramanian, and D. Terry. Amazon redshift re-invented. In *Proceedings of the 2022 International Conference on Management of Data*, SIGMOD '22, pages 2205–2217, 2022. doi: 10.1145/3514221.3526045.

[95] M. Aslett. How will the database incumbents respond to NoSQL and NewSQL? The 451 Group, April 2011.

[96] M. Aslett. Ten years of NewSQL: Back to the future of distributed relational databases. The 451 Group, June 2021.

[97] S. Babu and P. Bizarro. Adaptive query processing in the looking glass. In *CIDR*, pages 238–249, January 2005.

[98] D. F. Bacon, N. Bales, N. Bruno, B. F. Cooper, A. Dickinson, A. Fikes, C. Fraser, A. Gubarev, M. Joshi, E. Kogan, A. Lloyd, S. Melnik, R. Rao, D. Shue, C. Taylor, M. van der Holst, and D. Woodford. Spanner: Becoming a sql system. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, pages 331–343, 2017. doi: 10.1145/3035918.3056103.

[99] J. Baker, C. Bond, J. C. Corbett, J. Furman, A. Khorlin, J. Larson, J.-M. Leon, Y. Li, A. Lloyd, and V. Yushprakh. Megastore: Providing scalable, highly available storage for interactive services. In *Proceedings of the Conference on Innovative Data system Research (CIDR)*, pages 223–234, 2011.

[100] N. Bakibayev, D. Olteanu, and J. Závodný. Fdb: A query engine for factorised relational databases. *Proc. VLDB Endow.*, 5 (11):1232–1243, jul 2012. doi: 10.14778/2350229.2350242.

[101] S. Banon. Amazon: NOT OK - why we had to change Elastic licensing. `https://www.elastic.co/blog/why-license-change-aws`, jan 2021.

[102] J. Barr. AQUA (Advanced Query Accelerator) – A Speed Boost for Your Amazon Redshift Queries. `https://aws.amazon.com/blogs/aws/new-aqua-advanced-query-accelerator-for-amazon-redshift/`, Apr 2021.

[103] P. Baumann. A database array algebra for spatio-temporal data and beyond. In *Next Generation Information Technologies and Systems, 4th International Workshop, NGITS'99*, volume 1649 of *Lecture Notes in Computer Science*, pages 76–93, 1999. doi: 10.1007/3-540-48521-X_7.

[104] E. Begoli, J. Camacho-Rodríguez, J. Hyde, M. J. Mior, and D. Lemire. Apache calcite: A foundational framework for optimized query processing over heterogeneous data sources. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, pages 221–230, 2018. doi: 10.1145/3183713.3190662.

[105] A. Behm, S. Palkar, U. Agarwal, T. Armstrong, D. Cashman, A. Dave, T. Greenstein, S. Hovsepian, R. Johnson, A. Sai Krishnan, P. Leventis, A. Luszczak, P. Menon, M. Mokhtar, G. Pang, S. Paranjpye, G. Rahn, B. Samwel, T. van Bussel, H. van Hovell, M. Xue, R. Xin, and M. Zaharia. Photon: A fast query engine for lakehouse systems. In *Proceedings of the 2022 International Conference on Management of Data*, SIGMOD '22, pages 2326–2339, 2022. doi: 10.1145/3514221.3526054.

[106] P. A. Boncz, M. Zukowski, and N. Nes. Monetdb/x100: Hyper-pipelining query execution. In *CIDR*, pages 225–237, 2005.

[107] H. Boral and D. J. DeWitt. Database machines: An idea whose time passed? A critique of the future of database machines. pages 166–187, 1983. doi: 10.1007/978-3-642-69419-6_10.

[108] T. Bray. AWS and Blockchain. `https://www.tbray.org/ongoing/When/202x/2022/11/19/AWS-Blockchain`, nov 2019.

[109] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas. Apache flink: Stream and batch processing in a single engine. *The Bulletin of the Technical Committee on Data Engineering*, 38(4), 2015.

[110] R. Cattell. Scalable sql and nosql data stores. *SIGMOD Rec.*, 39:12–27, 2011.

[111] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*, OSDI '06, pages 205–218, 2006.

[112] S. Chaudhuri and G. Weikum. Rethinking database system architecture: Towards a self-tuning risc-style database system. In *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases*, pages 1–10, 2000.

[113] C. Chin. The rise and fall of the olap cube. `https://www.holistics.io/blog/the-rise-and-fall-of-the-olap-cube/`, January 2020.

[114] M. Chock, A. F. Cardenas, and A. Klinger. Database structure and manipulation capabilities of a picture database management system (picdms). *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(4):484–492, 1984. doi: 10.1109/TPAMI.1984.4767553.

[115] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, jun 1970. doi: 10.1145/362384.362685.

[116] E. F. Codd. Further normalization of the data base relational model. *Research Report / RJ / IBM / San Jose, California*, RJ909, 1971.

[117] W. W. W. Consortium. Overview of sgml resources. `https://www.w3.org/MarkUp/SGML/`, 2004.

[118] W. W. W. Consortium. Extensible Markup Language (XML). `https://www.w3.org/XML/`, 2016.

[119] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, M. S. Yasushi Saito, C. Taylor, R. Wang, and D. Woodford. Spanner: Google's Globally-Distributed Database. In *OSDI*, 2012.

[120] A. Crotty, V. Leis, and A. Pavlo. Are you sure you want to use MMAP in your database management system? In *Conference on Innovative Data Systems Research*. www.cidrdb.org, 2022.

[121] B. Dageville, T. Cruanes, M. Zukowski, V. Antonov, A. Avanes, J. Bock, J. Claybaugh, D. Engovatov, M. Hentschel, J. Huang, A. W. Lee, A. Motivala, A. Q. Munir, S. Pelley, P. Povinec, G. Rahn, S. Triantafyllis, and P. Unterbrunner. The snowflake elastic data warehouse. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 215–226, 2016. doi: 10.1145/2882903.2903741.

[122] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *6th Symposium on Operating Systems Design & Implementation (OSDI 04)*. USENIX Association, Dec. 2004.

[123] J. Dean and S. Ghemawat. Mapreduce: A flexible data processing tool. *Commun. ACM*, 53(1):72–77, Jan. 2010.

[124] A. Dearmer. Storing apache hadoop data on the cloud - hdfs vs. s3. `https://www.xplenty.com/blog/storing-apache-hadoop-data-cloud-hdfs-vs-s3/`, November 2019.

[125] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.*, 41(6):205–220, oct 2007.

[126] A. Deutsch, N. Francis, A. Green, K. Hare, B. Li, L. Libkin, T. Lindaaker, V. Marsault, W. Martens, J. Michels, F. Murlak, S. Plantikow, P. Selmer, O. van Rest, H. Voigt, D. Vrgoč, M. Wu, and F. Zemke. Graph pattern matching in gql and sql/pgq. In *Proceedings of the 2022 International Conference on Management of Data*, SIGMOD '22, pages 2246–2258, 2022. doi: 10.1145/3514221.3526057.

[127] D. DeWitt and J. Gray. Parallel database systems: The future of high performance database systems. *Commun. ACM*, 35(6): 85–98, jun 1992. doi: 10.1145/129888.129894.

[128] C. Diaconu, C. Freedman, E. Ismert, P. Larson, P. Mittal, R. Stonecipher, N. Verma, and M. Zwilling. Hekaton: SQL server's memory-optimized OLTP engine. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 1243–1254, 2013. doi: 10.1145/2463676.2463710.

[129] M. Elhemali, N. Gallagher, N. Gordon, J. Idziorek, R. Krog, C. Lazier, E. Mo, A. Mritunjai, S. Perianayagam, T. Rath, S. Sivasubramanian, J. C. S. III, S. Sosothikul, D. Terry, and A. Vig. Amazon DynamoDB: A scalable, predictably performant, and fully managed NoSQL database service. In *USENIX Annual Technical Conference*, pages 1037–1048, July 2022.

[130] J. Fan, A. G. S. Raj, and J. M. Patel. The case against specialized graph analytics engines. In *Seventh Biennial Conference on Innovative Data Systems Research, CIDR*, 2015.

[131] B. Fitzpatrick. Distributed caching with memcached. *Linux J.*, 2004(124):5, aug 2004. ISSN 1075–3583.

[132] M. Freitag, M. Bandle, T. Schmidt, A. Kemper, and T. Neumann. Adopting worst-case optimal joins in relational database systems. *Proc. VLDB Endow.*, 13(12):1891–1904, jul 2020. doi: 10.14778/3407790.3407797.

[133] H. Fu, C. Liu, B. Wu, F. Li, J. Tan, and J. Sun. Catsql: Towards real world natural language to sql applications. *Proc. VLDB Endow.*, 16(6):1534–1547, feb 2023. doi: 10.14778/3583140.3583165.

[134] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. *SIGOPS Oper. Syst. Rev.*, 37(5):29–43, oct 2003. ISSN 0163-5980. doi: 10.1145/1165389.945450.

[135] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-total. In *Proceedings of the International Conference on Data Engineering*, pages 152–159, 1996. doi: 10.1109/ICDE.1996.492099.

[136] S. Gupta, S. Rahnama, J. Hellings, and M. Sadoghi. Resilientdb: Global scale resilient blockchain fabric. *Proc. VLDB Endow.*, 13(6):868–883, 2020. doi: 10.14778/3380750.3380757.

[137] E. Hanson and A. Comet. Why Your Vector Database Should Not be a Vector Database. `https://www.singlestore.com/blog/why-your-vector-database-should-not-be-a-vector-database/`, April 2023.

[138] G. Harrison. How WiredTiger Revolutionized MongoDB. `https://www.dbta.com/Columns/MongoDB-Matters/How-WiredTiger-Revolutionized-MongoDB-145510.aspx`, mar 2021.

[139] G. G. Hendrix, E. D. Sacerdoti, D. Sagalowicz, and J. Slocum. Developing a natural language interface to complex data. *ACM Trans. Database Syst.*, 3(2):105–147, jun 1978. doi: 10.1145/320251.320253.

[140] Y. Huai, A. Chauhan, A. Gates, G. Hagleitner, E. N. Hanson, O. O'Malley, J. Pandey, Y. Yuan, R. Lee, and X. Zhang. Major technical advancements in apache hive. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 1235–1246, 2014.

[141] D. Huang, Q. Liu, Q. Cui, Z. Fang, X. Ma, F. Xu, L. Shen, L. Tang, Y. Zhou, M. Huang, W. Wei, C. Liu, J. Zhang, J. Li, X. Wu, L. Song, R. Sun, S. Yu, L. Zhao, N. Cameron, L. Pei, and X. Tang. Tidb: A raft-based htap database. *Proc. VLDB Endow.*, 13(12):3072–3084, aug 2020. doi: 10.14778/3415478.3415535.

[142] K. E. Iverson. *A Programming Language*. John Wiley & Sons, Inc., 1962. ISBN 0471430145.

[143] A. Jindal, S. Madden, M. Castellanos, and M. Hsu. Graph analytics using vertica relational database. In *2015 IEEE International Conference on Big Data*, pages 1191–1200, 2015.

[144] R. Kallman, H. Kimura, J. Natkins, A. Pavlo, A. Rasin, S. Zdonik, E. P. C. Jones, S. Madden, M. Stonebraker, Y. Zhang, J. Hugg, and D. J. Abadi. H-store: A high-performance, distributed main memory transaction processing system. *Proc. VLDB Endow.*, 1(2):1496–1499, aug 2008. doi: 10.14778/1454159.1454211.

[145] A. Kane. pgvector. `https://github.com/pgvector/pgvector`, 2024.

[146] A. Kemper and T. Neumann. Hyper: A hybrid oltp&olap main memory database system based on virtual memory snapshots. In *Proceedings of the 27th International Conference on Data Engineering*, pages 195–206. IEEE Computer Society, 2011. doi: 10.1109/ICDE.2011.5767867.

[147] T. Kersten, V. Leis, A. Kemper, T. Neumann, A. Pavlo, and P. Boncz. Everything you always wanted to know about compiled and vectorized queries but were afraid to ask. *Proc. VLDB Endow.*, 11(13):2209–2222, jan 2019. doi: 10.14778/3275366.3284966.

[148] R. Kimball. *The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses*. John Wiley, 1996.

[149] R. Kimball and K. Strehlo. Why decision support fails and how to fix it. *SIGMOD Rec.*, 24(3):92–97, 1995.

[150] M. Kornacker, A. Behm, V. Bittorf, T. Bobrovytsky, C. Ching, A. Choi, J. Erickson, M. Grund, D. Hecht, M. Jacobs, I. Joshi, L. Kuff, D. Kumar, A. Leblang, N. Li, I. Pandis, H. Robinson, D. Rorke, S. Rus, J. Russell, D. Tsirogiannis, S. Wanderman-Milne, and M. Yoder. Impala: A modern, open-source sql engine for hadoop. In *CIDR*, 2015.

[151] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, pages 489–504, 2018. doi: 10.1145/3183713.3196909.

[152] S. Krishnan, Z. Yang, K. Goldberg, J. Hellerstein, and I. Stoica. Learning to optimize join queries with deep reinforcement learning, 2018. URL `https://arxiv.org/abs/1808.03196`.

[153] F. Lardinois. Aws gives open source the middle finger. `https://techcrunch.com/2019/01/09/aws-gives-open-source-the-middle-finger/`, jan 2019.

[154] V. Leis, A. Gubichev, A. Mirchev, P. A. Boncz, A. Kemper, and T. Neumann. How good are query optimizers, really? *Proc. VLDB Endow.*, 9(3):204–215, 2015. doi: 10.14778/2850583.2850594.

[155] D. Maier and B. Vance. A call to order. In *Proceedings of the Twelfth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 1–16, 1993. doi: 10.1145/153850.153851.

[156] R. Marcus, P. Negi, H. Mao, N. Tatbul, M. Alizadeh, and T. Kraska. Bao: Making learned query optimization practical. In *Proceedings of the 2021 International Conference on Management of Data*, SIGMOD '21, pages 1275–1288, 2021. doi: 10.1145/3448016.3452838.

[157] D. McDiarmid. Vector search with clickhouse. `https://clickhouse.com/blog/vector-search-clickhouse-p2`, May 2023.

[158] C. McDonnell. The graph-relational database, defined. https://www.edgedb.com/blog/the-graph-relational-database-defined, March 2022.

[159] W. McKinney et al. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56, 2010.

[160] F. McSherry. Scalability! but at what cost? http://www.frankmcsherry.com/graph/scalability/cost/2015/01/15/COST.html, January 2015.

[161] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis. Dremel: Interactive analysis of web-scale datasets. *Proc. VLDB Endow.*, 3(12):330–339, sep 2010. ISSN 2150-8097. doi: 10.14778/1920841.1920886.

[162] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, T. Vassilakis, H. Ahmadi, D. Delorey, S. Min, M. Pasumansky, and J. Shute. Dremel: A decade of interactive sql analysis at web scale. *Proc. VLDB Endow.*, 13(12):3461–3472, aug 2020. ISSN 2150-8097. doi: 10.14778/3415478.3415568.

[163] P. Menon, A. Ngom, T. C. Mowry, A. Pavlo, and L. Ma. Permutable compiled queries: Dynamically adapting compiled queries without recompiling. *Proc. VLDB Endow.*, 14(2):101–113, 2020. doi: 10.14778/3425879.3425882.

[164] C. Metz. Google search index splits with mapreduce. https://www.theregister.com/2010/09/09/google_caffeine_explained/, September 2010.

[165] J. Michels, K. Hare, K. Kulkarni, C. Zuzarte, Z. H. Liu, B. Hammerschmidt, and F. Zemke. The new and improved sql: 2016 standard. *SIGMOD Rec.*, 47(2):51–60, dec 2018. doi: 10.1145/3299887.3299897.

[166] D. Misev and P. Baumann. Sql support for multidimensional arrays. Technical Report 34, Jacobs University, July 2017. URL https://nbn-resolving.org/urn:nbn:de:gbv:579-opus-1007237.

[167] F. Nargesian, E. Zhu, R. J. Miller, K. Q. Pu, and P. C. Arocena. Data lake management: Challenges and opportunities. *Proc. VLDB Endow.*, 12(12):1986–1989, aug 2019. doi: 10.14778/3352063.3352116.

[168] H. Q. Ngo, C. Ré, and A. Rudra. Skew strikes back: New developments in the theory of join algorithms. *SIGMOD Rec.*, 42(4):5–16, feb 2014. doi: 10.1145/2590989.2590991.

[169] NVIDIA, P. Vingelmann, and F. H. Fitzek. Cuda toolkit. https://developer.nvidia.com/cuda-toolkit, 2020.

[170] M. A. Olson, K. Bostic, and M. I. Seltzer. Berkeley DB. In *Proceedings of the FREENIX Track: 1999 USENIX Annual Technical Conference*, pages 183–191, 1999.

[171] A. Pavlo and M. Aslett. What's really new with newsql? *SIGMOD Record*, 45(2):45–55, Sep 2016.

[172] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker. A comparison of approaches to large-scale data analysis. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 165–178, 2009.

[173] A. Pavlo, G. Angulo, J. Arulraj, H. Lin, J. Lin, L. Ma, P. Menon, T. Mowry, M. Perron, I. Quah, S. Santurkar, A. Tomasic, S. Toor, D. V. Aken, Z. Wang, Y. Wu, R. Xian, and T. Zhang. Self-driving database management systems. In *CIDR 2017, Conference on Innovative Data Systems Research*, 2017.

[174] A. Pavlo, M. Butrovich, A. Joshi, L. Ma, P. Menon, D. V. Aken, L. Lee, and R. Salakhutdinov. External vs. internal: An essay on machine learning agents for autonomous database management systems. *IEEE Data Eng. Bull.*, 42(2):32–46, 2019.

[175] P. Pedreira, O. Erling, M. Basmanova, K. Wilfong, L. Sakka, K. Pai, W. He, and B. Chattopadhyay. Velox: Meta's unified execution engine. *Proc. VLDB Endow.*, 15(12):3372–3384, aug 2022. doi: 10.14778/3554821.3554829.

[176] P. Pedreira, O. Erling, K. Karanasos, S. Schneider, W. McKinney, S. R. Valluri, M. Zait, and J. Nadeau. The composable data management system manifesto. *Proc. VLDB Endow.*, 16 (10):2679–2685, jun 2023. doi: 10.14778/3603581.3603604.

[177] D. Petersohn, S. Macke, D. Xin, W. Ma, D. Lee, X. Mo, J. E. Gonzalez, J. M. Hellerstein, A. D. Joseph, and A. Parameswaran. Towards scalable dataframe systems. *Proc. VLDB Endow.*, 13(12):2033–2046, jul 2020. doi: 10.14778/3407790.3407807.

[178] D. Petkovic. SQL/JSON standard: Properties and deficiencies. *Datenbank-Spektrum*, 17(3):277–287, 2017. doi: 10.1007/s13222-017-0267-4.

[179] D. Pritchett. BASE: An Acid Alternative: In Partitioned Databases, Trading Some Consistency for Availability Can Lead to Dramatic Improvements in Scalability. *ACM Queue*, 6(3):48–55, may 2008. doi: 10.1145/1394127.1394128.

[180] M. Raasveldt and H. Mühleisen. Duckdb: An embeddable analytical database. In *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, pages 1981–1984, 2019. doi: 10.1145/3299869.3320212.

[181] M. Rocklin. Dask: Parallel computation with blocked algorithms and task scheduling. In *Proceedings of the 14th Python in Science Conference*, pages 130–136, 2015.

[182] F. Rusu. Multidimensional array data management. *Found. Trends Databases*, 12(2-3):69–220, 2023. doi: 10.1561/1900000069.

[183] S. Sakr, A. Bonifati, H. Voigt, A. Iosup, K. Ammar, R. Angles, W. Aref, M. Arenas, M. Besta, P. A. Boncz, K. Daudjee, E. D. Valle, S. Dumbrava, O. Hartig, B. Haslhofer, T. Hegeman, J. Hidders, K. Hose, A. Iamnitchi, V. Kalavri, H. Kapp, W. Martens, M. T. Özsu, E. Peukert, S. Plantikow, M. Ragab, M. R. Ripeanu, S. Salihoglu, C. Schulz, P. Selmer, J. F. Sequeda, J. Shinavier, G. Szárnyas, R. Tommasini, A. Tumeo, A. Uta, A. L. Varbanescu, H.-Y. Wu, N. Yakovets, D. Yan, and E. Yoneki. The future is big graphs: A community view on graph processing systems. *Commun. ACM*, 64(9):62–71, aug 2021. doi: 10.1145/3434642.

[184] G. Salton and M. E. Lesk. The smart automatic document retrieval systems–an illustration. *Commun. ACM*, 8(6):391–398, jun 1965. doi: 10.1145/364955.364990.

[185] R. Sethi, M. Traverso, D. Sundstrom, D. Phillips, W. Xie, Y. Sun, N. Yegitbasi, H. Jin, E. Hwang, N. Shingte, and C. Berner. Presto: Sql on everything. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1802–1813, 2019. doi: 10.1109/ICDE.2019.00196.

[186] M. A. Soliman, L. Antova, V. Raghavan, A. El-Helw, Z. Gu, E. Shen, G. C. Caragea, C. Garcia-Alvarado, F. Rahman, M. Petropoulos, F. Waas, S. Narayanan, K. Krikellas, and R. Baldwin. Orca: a modular query optimizer architecture for big data. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 337–348, 2014. doi: 10.1145/2588555.2595637.

[187] M. Stonebraker. The case for polystores. https://wp.sigmod.org/?p=1629, 2015.

[188] M. Stonebraker and J. Hellerstein. *Readings in Database Systems*, chapter What Goes Around Comes Around, pages 2–41. 4th edition, 2005.

[189] M. Stonebraker, S. Madden, D. J. Abadi, S. Harizopoulos, N. Hachem, and P. Helland. The end of an architectural era: (it's time for a complete rewrite). In *Proceedings of the 33rd International Conference on Very Large Data Bases*, VLDB '07, pages 1150–1160. VLDB Endowment, 2007.

[190] M. Stonebraker, D. Abadi, D. J. DeWitt, S. Madden, E. Paulson, A. Pavlo, and A. Rasin. Mapreduce and parallel dbmss: Friends or foes? *Commun. ACM*, 53(1):64–71, Jan. 2010.

[191] M. Stonebraker, P. Brown, A. Poliakov, and S. Raman. The architecture of scidb. In *Scientific and Statistical Database Management - 23rd International Conference, SSDBM 2011*, volume 6809 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2011. doi: 10.1007/978-3-642-22351-8_1.

[192] L. Sullivan. Performance issues in mid-sized relational database machines. Master's thesis, Rochester Institute of Technology, 1989.

[193] Z. Sun, X. Zhou, and G. Li. Learned index: A comprehensive experimental evaluation. *Proc. VLDB Endow.*, 16(8):1992–2004, apr 2023. doi: 10.14778/3594512.3594528.

[194] Y. Sverdlik. Google dumps mapreduce in favor of new hyper-scale analytics system. `https://www.datacenterknowledge.com/archives/2014/06/25/google-dumps-mapreduce-favor-new-hyper-scale-analytics-system`, June 2014.

[195] R. Taft, I. Sharif, A. Matei, N. VanBenschoten, J. Lewis, T. Grieger, K. Niemi, A. Woods, A. Birzin, R. Poss, P. Bardea, A. Ranade, B. Darnell, B. Gruneir, J. Jaffray, L. Zhang, and P. Mattis. Cockroachdb: The resilient geo-distributed SQL database. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD*, pages 1493–1509, 2020. doi: 10.1145/3318464.3386134.

[196] D. ten Wolde, T. Singh, G. Szarnyas, and P. Boncz. Duckpgq: Efficient property graph queries in an analytical rdbms. In *CIDR*, 2023. URL `https://www.cidrdb.org/cidr2023/papers/p66-wolde.pdf`.

[197] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Antony, H. Liu, and R. Murthy. Hive - a petabyte scale data warehouse using hadoop. In *International Conference on Data Engineering (ICDE 2010)*, pages 996–1005, 2010. doi: 10.1109/ICDE.2010.5447738.

[198] E. Totoni, T. A. Anderson, and T. Shpeisman. HPAT: high performance analytics with scripting ease-of-use. In *Proceedings of the International Conference on Supercomputing*, pages 9:1–9:10, 2017. doi: 10.1145/3079079.3079099.

[199] T. Trautmann. Understanding the document-relational database. `https://fauna.com/blog/what-is-a-document-relational-database`, September 2021.

[200] D. Van Aken, A. Pavlo, G. J. Gordon, and B. Zhang. Automatic database management system tuning through large-scale machine learning. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, pages 1009–1024, 2017. doi: 10.1145/3035918.3064029.

[201] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, and I. Stoica. Apache spark: a unified engine for big data processing. *Commun. ACM*, 59(11):56–65, oct 2016. doi: 10.1145/2934664.

[202] C. Zaniolo. The database language GEM. In *SIGMOD*, pages 207–218, 1983.

[203] X. Zeng, Y. Hui, J. Shen, A. Pavlo, W. McKinney, and H. Zhang. An empirical evaluation of columnar storage formats. *Proc. VLDB Endow.*, 17(2):148–161, 2023. URL `https://www.vldb.org/pvldb/vol17/p148-zeng.pdf`.

[204] X. Zhang, Z. Chang, Y. Li, H. Wu, J. Tan, F. Li, and B. Cui. Facilitating database tuning with hyper-parameter optimization: a comprehensive experimental evaluation. *Proc. VLDB Endow.*, 15(9):1808–1821, may 2022. doi: 10.14778/3538598.3538604.