SingleStore: Simplifying Data Architectures with a Unified Database

A Technical Whitepaper

Rick F. van der Lans
Independent Business Intelligence Analyst
R20/Consultancy

Sponsored by

Table of Contents

1 Introduction

General-Purpose Databases – Most organizations work with databases that can be classified as *general-purpose databases*. These products can support many different types of *use cases*, such as transactional data-entry applications, data warehouses, websites, and portals. To accomplish this, database administrators optimize them for that specific use case. For example, they are optimized to process many complex queries or process a large transactional workload. But they cannot be optimized to support multiple use cases *concurrently*.

General-purpose databases cannot support multiple workloads concurrently.

Therefore, many data architectures, such as the classic data warehouse architecture, consist of several databases, each designed to support a different use case. Evidently, this leads to complex data architectures. A popular example is that one database is developed to support the transactional workload and another to support the reporting and analytical workload, and data is periodically copied from one to the other. Both databases can be developed with the same database product, but are optimized differently.

Drawbacks of Copying Data – In numerous environments that deploy general-purpose databases, data is copied between databases. Copying data has the following drawbacks:

- Copying data increases the data latency experienced by data consumers
- Extracting data from a transactional database can interfere with the transactional workload
- Duplicating data complicates complying with data privacy regulations such as GDPR
- Data stored in copied databases must also be secured as in the original databases, complicating data security
- Processes that copy data can fail resulting in incorrect data and a degraded data quality level

Specialized Databases – Another group of databases is called the *specialized databases*. These are developed and tuned to support a limited number of uses cases, but support them extremely well. Compare this to specialized cars, such as racing cars or large diesel trucks. The main advantage of these specialized products is that they can handle very large workloads. For example, *analytical databases* can process massive query workloads and *transactional databases* can support massive transactional workloads initiated by countless users.

Specialized database servers are optimized for a limited set of workloads.

Specialized databases enable organizations to run use cases that cannot be implemented with general-purpose databases. Unfortunately, this group does not minimize the need to develop multiple databases, as separate databases for separate use cases are still required.

Unified Databases – A relatively new database category is formed by the *unified databases* (or *translytical databases*). These products are designed to support a large transactional *and* analytical workload *concurrently*. Under the hood, they contain features supported by specialized analytical and specialized transactional databases. Their main advantage is that they enable organizations to develop simpler and more flexible data architectures that can still process massive workloads. Examples of how unified databases can simplify data architectures:

Unified databases support multiple, massive workloads concurrently.

- Replacing general-purpose databases with unified databases improves performance and minimizes the need for a complex data architecture
- Databases developed for separate applications can be merged
- Support for a new workload that cannot be developed with available general-purpose databases can now be developed
- Minimizing the need for multiple databases makes it easier to develop new data architectures
- Data warehouse and data marts can be merged which simplifies the data architecture
- Transactional databases can be extended with an analytical workload to support real-time analytical data consumers without having to copy the data
- When existing databases are migrated to a unified database running on a cloud platform the potential scalability of that platform can be exploited much better
- Existing data architectures can be extended to support new transactional and analytical applications

This Whitepaper – This whitepaper describes the need for unified databases and one product in particular *SingleStore*. The following features are described that make the product suitable for concurrent transactional and analytical workloads:

SingleStore is a unified database.

- A table storage structure called *universal storage* designed to support transactional and analytical workloads concurrently
- Strong consistency that ensures a consistent state of data for all the users
- Pipelined and parallel data loading for massive data ingestion
- Columns with JSON data types to store complex-structured data
- A distributed, shared-nothing architecture that provides near-linear scalability
- Compilation of queries to improve query performance
- Support for fast, complex analytical queries

2 The Continuous Growth of Data and Databases

Unique Data or Redundant Data? – The amount of data that organizations store continues to grow. However, most of the stored data is not new or original data. Much of it is stored several times. For example, data about a specific customer can be stored in several transactional systems, a staging area, a data warehouse, several data marts and in a data lake. Even within one database, data is often stored multiple times to support different data consumers. Additionally, business users may have extracted this customer data from a central database and copied it to private files and spreadsheets. The growth of data is enormous within organizations, but a large part of it is non-unique, redundant data.

Much of the stored data is non-unique, redundant data.

Note that some redundant data is not stored in its original form, but in an aggregated or compressed form. For example, data marts often contain aggregated data.

More and More Databases – Along with the growth of data, the number of databases managed by organizations is also growing. It was common to store data in transactional databases and in a data warehouse, but nowadays data is also stored in data marts, data lakes, data hubs, data lakehouses and cloud-based object storage technologies.

Several reasons exist why so many databases have been developed and why the data was not stored in one database just once.

- Many transactional applications that are purchased by organizations come with their own database. The use of all these *application databases* has resulted in a myriad of independent databases and *transactional silos*. Data stored in these silos is not shared by applications; an application cannot access the database of another application. The effect is that data needs to be copied to other databases. Synchronizing the data between these databases is quite a technological challenge. In most cases, synchronization takes place when data from all those application databases is copied to, for example, a data warehouse or data hub. Next, the data is linked and made consistent.

Application databases have resulted in transactional silos.

- Due to the increasing need of organizations to analyze data, special databases such as data warehouses and data marts have been developed to support these analytical workloads. Transactional databases are not tuned for this workload, the data structures of the tables are not designed to support analytics, and the additional analytical workload would interfere with the existing transactional workload; it slows down transactions and slows down queries. Therefore, the analytical workload is commonly shifted to data warehouses and data marts, which circumvents the performance and interference problems. Additionally, data warehouses are designed and optimized to speed up analytical queries. For example, tables in data marts are commonly designed using a star schema technique that would never be used for the tables in transactional databases. Figure 1 shows the high-level architecture of a classic data warehouse architecture in which data is copied and stored multiple times.

- Many transactional database products are not designed to support analytical workloads. For example, many NoSQL databases that excel at supporting transactional workloads are weak at supporting analytics. Therefore, data from such transactional databases almost always needs to be copied to a database that can handle an analytical workload.

- New forms of data usage have recently led to the introduction of new types of databases, such as data lakes, data hubs and data lakehouses. This has also led to more databases and more redundant data storage.

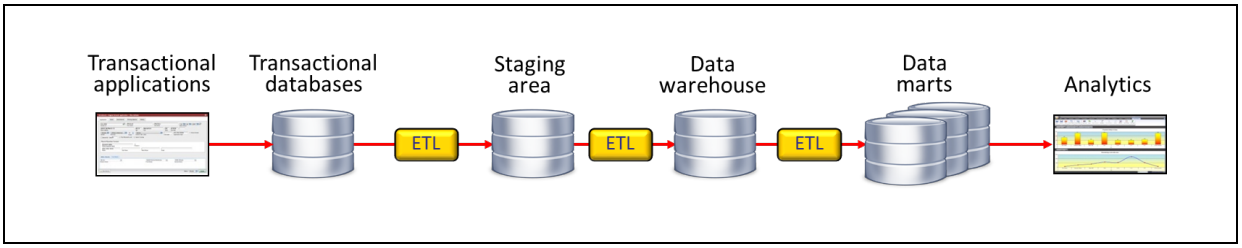

Figure 1 A *classic data warehouse architecture in which data is redundantly stored several times.*

3 Specialization of Databases

General-Purpose Databases – Besides the above reasons described in the previous chapter why so many databases have been developed, there is another: *specialization of databases*.

General-purpose databases are like Swiss army knives.

Most organizations work with databases that can be classified as *general-purpose databases*. These products can support many different types of *use cases*, such as transactional data-entry applications, data warehouses, websites and portals. They are like Swiss army knives. To achieve this, database administrators optimize them for that specific use case. For example, they are optimized to run many complex queries or process a large transactional workload. But they cannot be optimized to support multiple use cases *concurrently*.

The consequence of using general-purpose databases is that many data architectures consist of several databases, each developed to support a different use case. A clear example is the classic data warehouse architecture, in which some databases are developed to support the transactional workload and others to support the reporting and analytical workload, and data is periodically copied from one to the other. Both databases can be developed with the same product, but are optimized differently. Using general-purpose databases can lead to complex data architectures.

Specialized Databases – General-purpose databases are good at many things, but they excel nothing. In this respect, they are like Swiss army knives, because they do not offer the best scissors nor the best corkscrews. For certain workloads, good is not good enough. This is especially true for systems that can be classified as big data systems. Due to the amount of data ingested, the number of transactions, and the complexity of analytical queries they must support, excellence is required and good is not enough.

This has led to the introduction of many *specialized databases*, such as time series, graph and key-value databases. Figure 2 shows a classification of specialized databases. Most of the subcategories on the right in this figure are specialized databases and excel at certain workloads. For example, graph databases are perfect for supporting graph analytics, many NoSQL databases can handle massive transactional workloads and streaming SQL products execute queries on data streams excellently.

The drawback of specialized databases is that the data they contain must always be copied to another database to make it available to other workloads. For example, data stored in a transactional NoSQL database almost always needs to be copied to another database to make it available for reporting and analytics. Graph databases do not support transactions very well, so new data must be copied from another database to the graph database.

Specialized databases have also increased the number of databases.

Therefore, the fifth reason that has led to an increasing number of databases is the use of specialized databases designed for particular workloads.

Unified Databases – More recently, a new group of databases was introduced, *unified databases* (sometimes called *translytical databases*). These products are developed and optimized to support a transactional and analytical workload concurrently without the need to store the data redundantly. *SingleStore* is a representative of this group of unified databases.

Unified databases support a transactional and analytical workload concurrently.

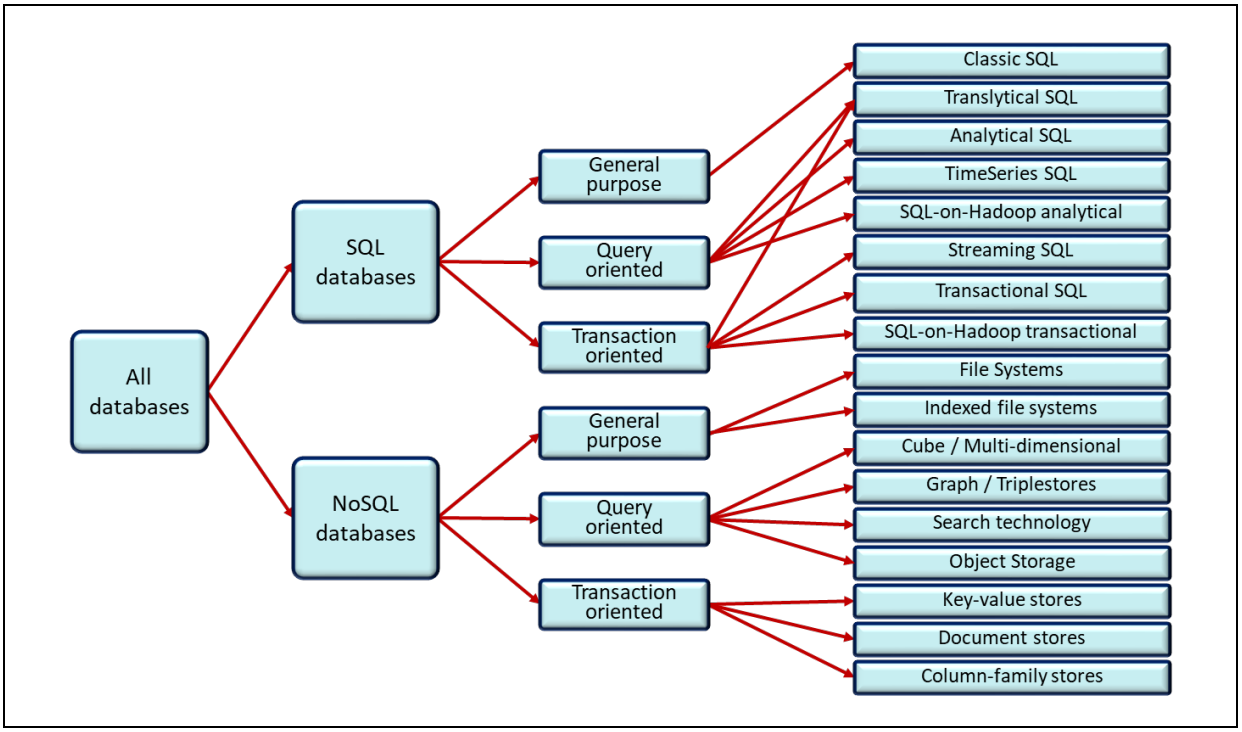

Figure 2 *A classification of databases.*

Note that the use of unified databases is not limited to use cases in which both workloads must be supported. They can also be deployed successfully when the workload is predominantly transactional or analytical. This explains why they are classified in Figure 2 as analytical as well as transactional products.

Occasionally, unified databases are confused with general-purpose databases as both support multiple workloads. They do, but the difference is in the word concurrently. Unified databases support them concurrently, while general-purpose products are tuned for a specific workload, which means that only one workload is properly supported concurrently.

To summarize, Figure 3 shows how unified databases compare to other categories.

4 Challenges of Copying Data

On a more architectural level, copying data introduces the following five challenges.

Copying Data Leads to High-Latency Data – Data is usually copied by batch programs that extract data from a source system, transform it, and load it into a target database. For example, in data warehouse architectures, ETL programs copy data from transactional databases (sources) to a data warehouse (target). To refresh the data, the copying process is scheduled periodically. This means that the data consumers of target databases never work with *zero-latency data*. The refresh rate of copying determines this data latency; the lower the refresh rate, the higher the data latency. Note that some data warehouses still operate with a refresh rate of 24 hours.

Copied data is usually high-latency data.

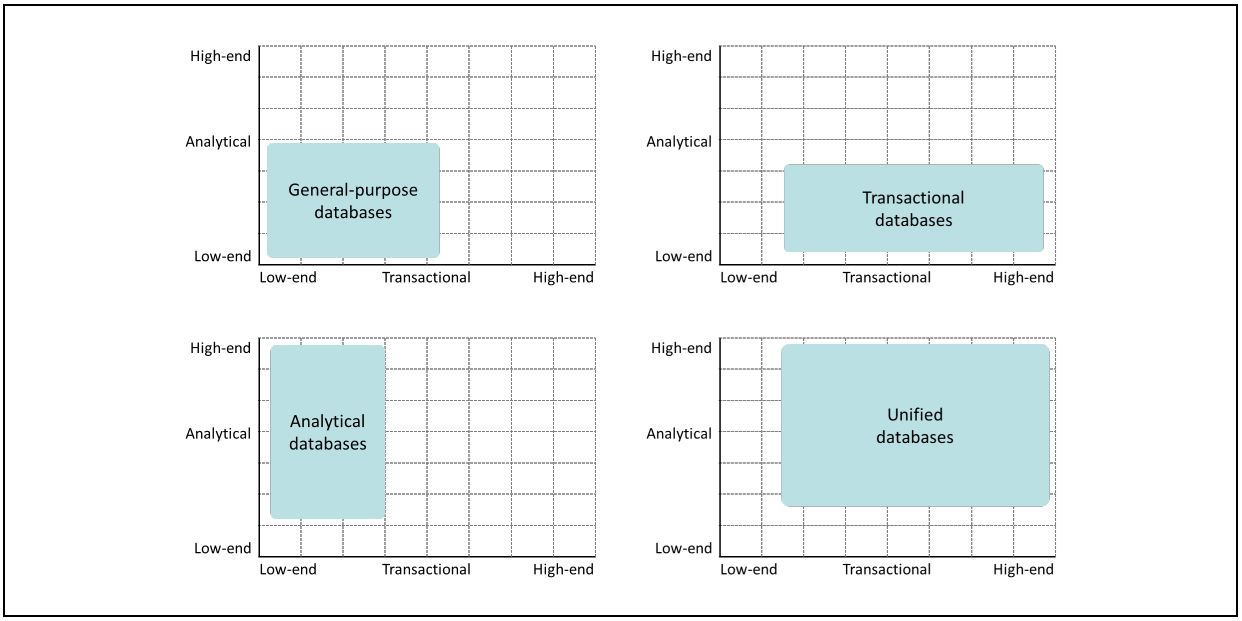

Figure 3 *Comparison of four database categories.*

High data latency is usually not a problem for dashboards and reports used by top executives, because their decisions are not based on the most recent data. They are more interested in the overall trends and not detailed data. Nor is it a problem for most data scientists when they develop their prescriptive and descriptive models. Access to zero-latency data is not required to discover patterns and trends. The last few minutes of new data do not affect overall trends.

However, the number of data consumers requiring low-latency data is increasing. Examples:

- Operational management may need to know what is happening right now. Even five minutes old data may not be sufficient and can lead to unnecessary costs, missed opportunities and lost sales.

- Customers who have access to enterprise data usually require access to low-latency data. For example, if they place a new order using the website and check the status of all their orders a few seconds later, the new order had better show up in the list, otherwise they think that the order has not been processed.

- Suppliers of a retail company may want to see the stock levels of their products in the stores to optimize product transportation. A data latency of one hour can cause products to be delivered too late which can hurt sales.

- In industrial environments where sensor data is produced, it may be a requirement to analyze data instantly as it is generated. For example, if a heat sensor indicates that a machine component is getting too hot, the machine may need to be switched off as soon as possible to prevent irreparable damage. In this example, accessing high-latency data is useless.

- Data science models embedded in transactional applications may require access to zero-latency data. For example, a model that determines whether a particular credit card payment is possibly fraudulent, must be executed immediately upon receipt of the payment. Implementing this model

on old data stored in a data warehouse may be worthless. In this example, access to zero-latency data is required.

Always-On Systems – More and more transactional systems must be operational 24x7. Examples of these *always-on systems* are retail websites, online gaming systems, booking systems, and government systems with which citizens can request data. Copying data from these systems can be challenging for two reasons.

Extracting data from always-on systems can lead to interference.

First, several approaches exist to extract data from transactional databases. Using ETL or ETL-like solutions always leads to interference on the transactional systems. If the interference level is unacceptable, the refresh rate is usually kept low by scheduling the extraction when the workload is low. This increases the data latency for the consumers of the target database.

When *change data capture technology* (data replication) is used to extract data, the interference is minimal. With this approach, the data latency of the target database is extremely low. A drawback of this approach is that the data is often copied unchanged. The data itself is commonly not cleansed or processed and the table structures of the target database are very similar to those of the source database. This makes analyzing the data stored in a target database complex as the table structures are generally designed to support transactions, not analytics.

A second challenge of copying data is to ensure a consistent state of the data. For example, if a new order is inserted in a transactional database and a separate transaction automatically updates the stock level of the products included in the order, both aspects should be reflected in the target database. If the new order is copied, but the stock level is not updated, the target database would be in an inconsistent state. Ensuring this consistent state within a target database can be challenging.

Data Privacy Rules – An organization's own responsibility concerning *personally identifiable data* plus regulations and laws for data privacy, such as *GDPR*, require data privacy rules to be defined on databases to protect this data against unauthorized use. Two recurring rules regarding data privacy are the *right to be forgotten* and the *right to be corrected*. The more duplicates of personally identifiable data stored, the more difficult it is to comply with these rules.

Redundant data complicates complying with data privacy rules.

In this regard, the architectural design principle called *data minimization* is recommended. Data minimization means that a data architecture is designed to minimize the number of stored duplicates.

Data Security – Stored data must be secured against unauthorized usage. Therefore, security rules and polices need to be defined and managed. But similar (or more strict rules) must be defined for each copy of the data. The more copies of the data, the more complex it is to keep and manage all those security rules consistent.

Redundant data complicates data security.

Failing Copying Process – A challenge that is sometimes overlooked when copying data is that copying processes can fail. They can fail for all kinds of reasons, such as the structure of the loaded data is not according to the predefined schema; the loaded data contains strange or missing values that the ETL program does not expect; the target database is full; the source database does not provide all the required data; or the target database crashes.

A failing copying process can lead to incorrect data in the target database. For example, some organizations do not have procedures that automatically fix problems when ETL programs crash after only half of the data has been loaded. In such organizations, the data engineers discover the problem the next

morning. They start a manual process in which they may delete the loaded data, create a fix, and restart the ETL job. After this unplanned intervention, it can be uncertain whether all the data that had to be deleted has actually been removed, and whether or not too much data has been removed. An additional side effect is that the new data is delayed before it becomes available, which increases data latency.

5 The Impact of Including Unified Databases in Data Architectures

Unified databases are not merely technologies that accelerate workloads, it is not just a matter of performance and scalability. As important is that they can simplify data architectures by minimizing the number of databases that need to be designed, developed, managed, optimized, tuned and secured. This chapter describes some typical examples of the impact of unified databases on data architectures.

Accelerating Existing Systems – IT systems may have reached their limits with respect to performance and scalability. This can limit an organization in several ways. Users may not be allowed to run certain analytical queries; they may experience long response times for their queries; and data-entry users may also experience delays. If users are external customers or suppliers, it can even harm certain business processes and lead to customers switching to other suppliers.

Replacing an existing database with a unified database can accelerate systems, enabling higher performance and better scalability. A unified database can support more concurrent transactions, more users, more data ingestion, more complex queries, and so on. Removing data access restrictions gives an organization the freedom to deploy their data.

Unified databases can free organizations in deploying their data.

Merging of Databases – For many applications a separate database is developed. The main drawback of such an approach is that all those isolated databases need to be developed, tested, managed, optimized, tuned, secured, and so on, which is quite a time-consuming task. Additionally, many of the databases contain similar data, which leads to many data flows between those database to keep similar data synchronized.

Merging all those database into one unified database simplifies database management, minimizes redundant storage of data, and reduces the number of data flows; see Figure 4.

Sometimes databases cannot handle the existing workload, so multiple copies of the same database are installed, allowing the entire workload to be distributed across all of the databases. This leads to many databases, redundant data storage and data flows to synchronize them. A scalable unified database can drastically simplify such a data architecture.

Introducing a New Workload – A new transactional or analytical workload can be too intensive for the databases an organization uses. For example, the data ingestion rate of a new transactional system that streams massive amounts of sensor data into a database can be too high, or the processing of an analytical workload consisting of ultra-complex SQL queries produced by data scientists may be unacceptably slow. In such cases, an organization can decide to deploy a unified database to support one or both workloads. In other words, a unified database enables an organization to develop applications that the technology in use cannot support.

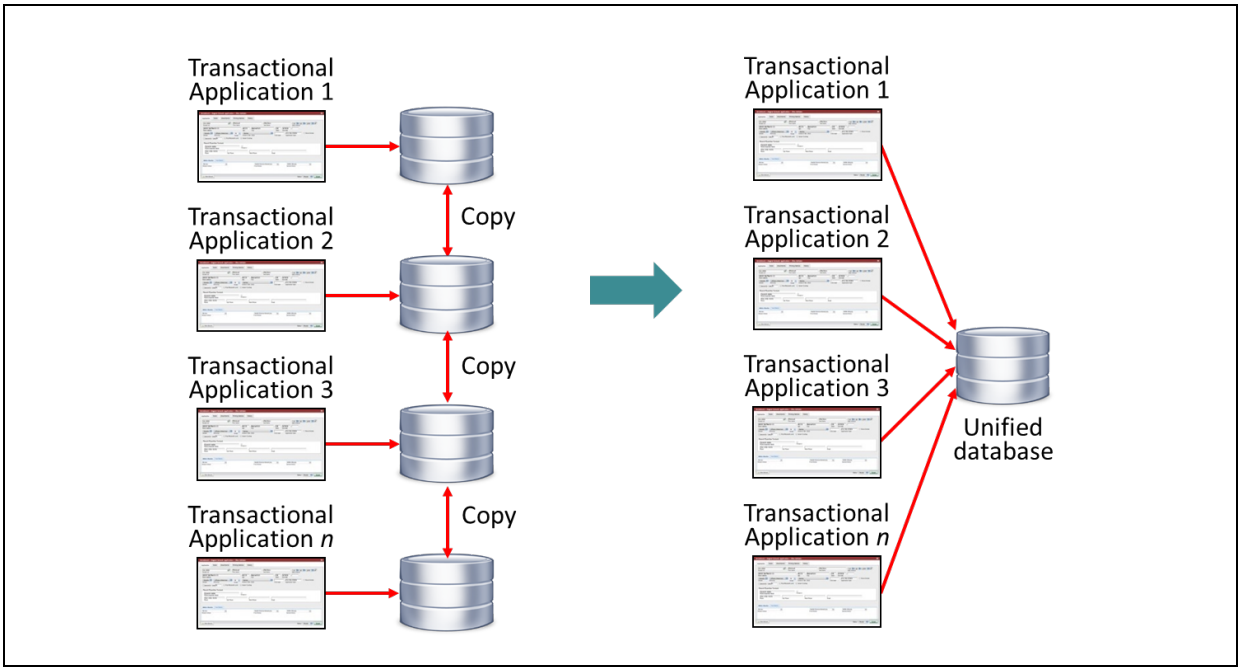

Figure 4 *A unified database can replace a set of databases.*

Simplifying New Data Architectures – Unified databases can simplify new data architectures by reducing the need to work with two separate databases, one for the transactional and the other for the analytical workload; see Figure 5. This simplifies the architecture as there is no need to develop, manage and secure two databases, nor no need to develop, manage and monitor a solution that periodically copies data from one to the other.

Unified databases can offer zero-latency data to reporting and analytics.

An additional, important benefit is that in such an architecture analytical data consumers work with zero-latency data, because inserted data is instantly available to them.

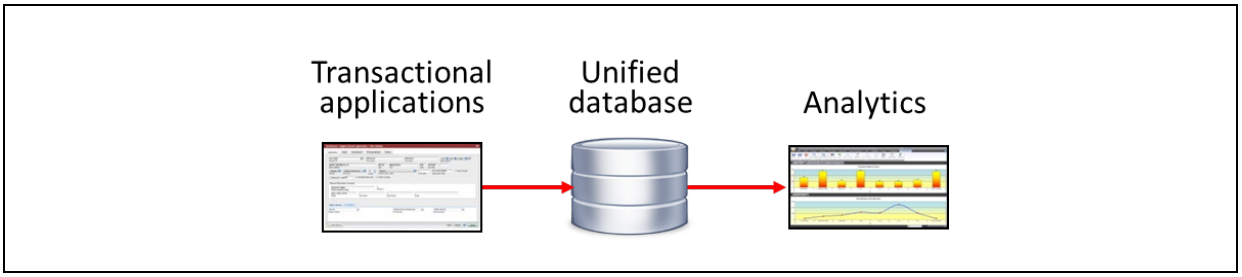

Figure 5 *Unified databases enable the development of simple, new data architectures that support transactional and analytical workloads concurrently without data redundancy.*

Merging of a Data Warehouse and Data Marts – Unified databases also belong to the category of analytical databases. They are more than capable of supporting a large, complex and varied analytical workload. Therefore, as with an analytical database, there is less need to develop a physical data warehouse plus a few physical data marts. Normally, data marts are added to the data architecture to accelerate performance by distributing the analytical workload across multiple databases (data marts) and by designing data structures (star schemas) optimized for analytical workloads.

Existing data warehouse architectures can be simplified by using a unified database to implement the central data warehouse without data marts; see Figure 6. In this case, the entire workload is not distributed across multiple data marts, but is processed by one central data warehouse. If typical data mart data structures are required, such as star schemas, they can be implemented virtually using SQL views.

Unified databases minimize the need for physical data marts.

The advantages of this approach are that data management tasks are simplified, data is copied less often and the data latency is reduced (one copying process less).

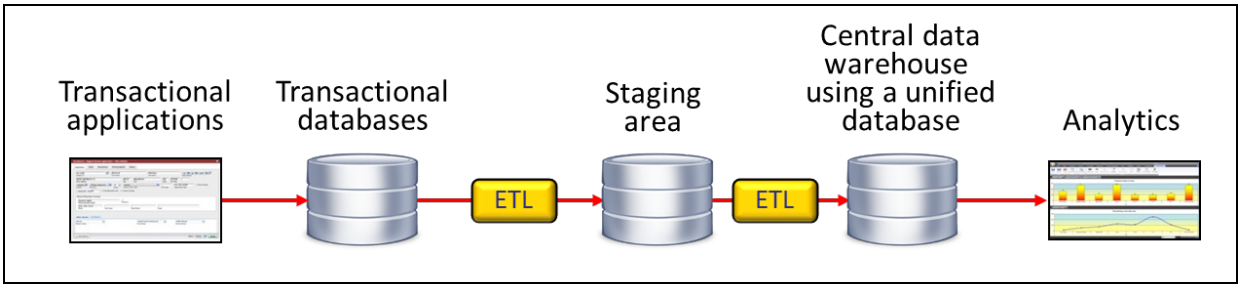

Figure 6 *Unified databases enable the removal of data marts.*

This architecture can be simplified even further by using the same unified database for the analytical and the transactional workload. This also simplifies database management and minimizes the need to copy data.

Extending a Transactional System with an Analytical Workload – If an existing transactional database is developed with a general-purpose or transactional database, it is unlikely to be able to support a high-end analytical workload. This would cause too much interference. A solution is to replace the database in use with a unified database. The transactional workload continues to run and the data becomes available for analytics. Such an architecture can even support real-time analytics.

Developing Analytics-Ready Transactional Databases – New transactional applications can be developed on a unified database, even if no analytical workload on the data is known and planned. When analytical requirements arise, there is no need to design, develop and manage additional databases and processes to copy the data. The unified database is analytics-ready.

Unified databases can make transactional databases analytics-ready.

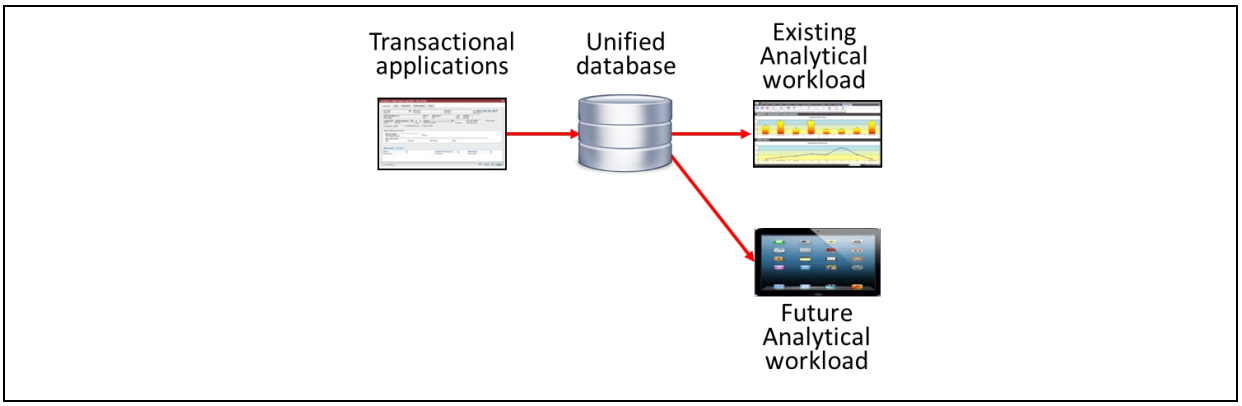

Figure 7 *Unified databases are future-proof solutions.*

Migrating to the Cloud – Some organizations that migrate their databases to the cloud stick with the same database product. This is technically feasible, as a cloud version is available for most databases. However, the fact that a product is also available on cloud platforms, does not automatically mean that it can exploit the scalability power of the hardware infrastructure of cloud data centers. It all depends on the internal architecture of the database product.

Unified databases such as SingleStore can fully exploit the potential scalability of cloud platforms.

Unified databases, such as SingleStore, can exploit these highly scalable infrastructures and enable organizations to take full advantage of them. Therefore, migrating from an on-premises, general-purpose database to a cloud-based unified database can have several advantages, but the most dominant is scalability.

Augmenting Existing Data Architecture – Unified databases can also be used to augment existing data architectures; see Figure 8. The unified database is used to support new transactional and analytical applications, to integrate data from the existing and new systems and to offload workload from the existing systems.

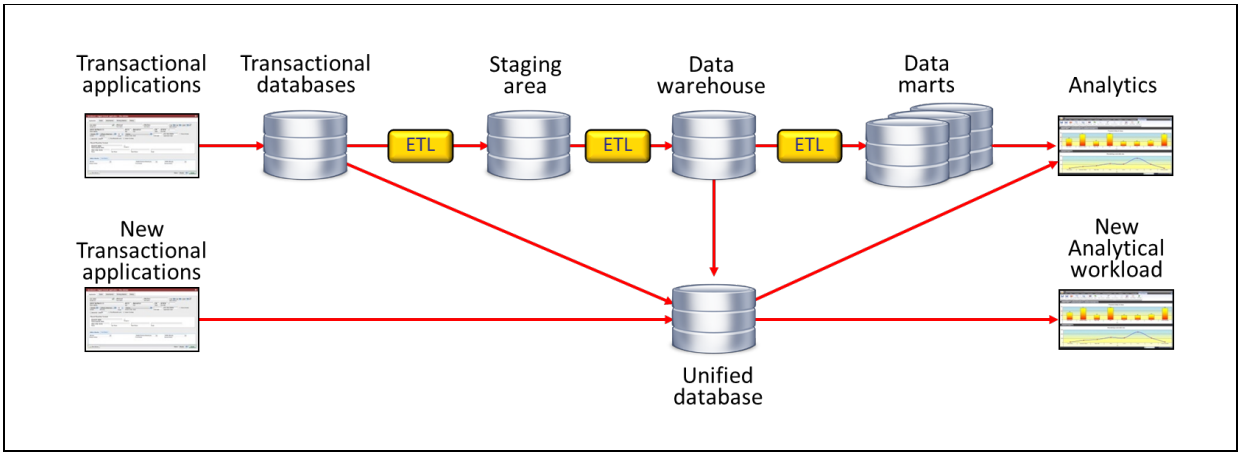

Figure 8 *Unified databases can augment existing data architectures to support new workloads.*

6 How Does SingleStore Support Multiple Workloads?

This chapter describes the unified database *SingleStore*. For developers, SingleStore is like any other SQL database. It supports tables, columns, keys, SQL queries, and so on. But internally, the product is quite unique and designed differently from most other SQL databases. It is different in that it is designed to support a transactional and analytical workload *concurrently,* making it a unified database The descriptions of the SingleStore features are described in the following three sections:

- Section 1: the main feature that enables a concurrent transactional and analytical workload
- Section 2: features for supporting transactional workloads
- Section 3: features for supporting analytical workloads

Note that this whitepaper only includes features that differentiate the product from general-purpose and other databases and that make SingleStore a unified database.

6.1 Support for Transactional and Analytical Workloads

The key feature of SingleStore that enables a concurrent transactional and analytical workload is *universal storage*, which is a new table type. The data in tables of this type is physically stored, organized, and accessed to support transactions *and* analytics. But before universal storage can be described, another table type called *rowstore* needs to be explained.

In-Memory, Record-Oriented Storage for Fast Transaction Processing – SingleStore offers a table type designed to support fast transaction processing called rowstore. It has three main characteristics.

First, rowstore tables are kept into memory. Transactions on these tables are processed purely in memory, which is much faster than if each time the data needs to be updated, inserted or deleted, it has to be extracted from or written to disk. To ensure that these transactions do not get lost or damaged when systems crash, a SingleStore background process continuously writes those changes back to disks ensuring 100% safety.

SingleStore keeps transactional data in memory in a record-oriented format.

Second, the data in these tables is kept in memory in a *record-oriented format* and not in a *column-oriented format*. A record-oriented format is known to accelerate transaction processing, because most transactional applications are record-oriented in nature, as a transaction is always the insert, update and/or delete of a small set of records. Therefore, it makes sense to keep the data in memory in the same record-oriented format.

Third, the concurrency control mechanism implemented in SingleStore is *lock-free* or *non-blocking*. Transactional applications do not block analytical applications, or vice versa. Because SingleStore does not need to keep track of locks, internal database administration is simplified and fewer central processes are required. If SingleStore were not lock-free, it would not be able to efficiently exploit an MPP hardware infrastructure, as is often found in cloud data centers. More on this later in this section.

Universal Storage – In addition to the rowstore table type, which accelerates transactional workloads, SingleStore offers a second table type designed to support analytics called *universal storage*. This table type is the main feature of SingleStore that enables a concurrent transactional and analytical workload.

Universal storage tables are optimized for processing transactional and analytical workloads concurrently.

To support complex analytical queries, SingleStore stores the data of a universal storage table in a column-oriented format on disk. Compression techniques are used to minimize I/O. Column-oriented storage is optimized to support complex queries on very large tables, especially tables that are too large to fit in memory. Hash indexes defined on these tables are used to efficiently retrieve specific rows.

Universal storage tables contain a columnstore (for analytics) and a rowstore (for transactions).

Internally, the universal storage table type also includes a rowstore. So, a universal storage table consists partly of a *columnstore* and partly of a rowstore. When inserts, update or deletes are processed, they are executed on the rowstore of the universal storage table. The performance and scalability of rowstores are inherited by this embedded rowstore. After a while, the rowstore data is deleted and added to the column-store.

The results of queries executed on a universal storage table include the relevant data stored in the columnstore and the relevant data stored in the rowstore. In other words, although the data of universal a storage table is spread across the columnstore and the rowstore, to the users and developers it feels as if it is one table. Universal storage tables can be joined together and they can be joined with rowstore tables, no restrictions apply, they act as 'normal' tables.

The important message is that tables of the universal storage type can run analytical and transactional workload concurrently and fast. This is what makes SingleStore a unified database. Rows can be inserted at the speed of the rowstore and can be queried for analytics at the speed of the columnstore.

6.2 Support for Transactional Workloads

The previous section describes the rowstore and universal storage tables types. Both are designed to support a transactional workload. This section describes the additional transactional features SingleStore offers.

Skip Lists for Fast Point Queries – A *point query* retrieves a single business object, such as a customer record, an invoice, or a patient file. Transactional applications typically execute many point queries and it is important that they are executed fast. This requires some form of direct or semi-direct access. Scanning full tables or large partitions to extract just one or two records is often too slow.

Most databases use *Btree indexes* for implementing semi-direct access to data. Btree indexes avoid full table scans when executing point queries. The disadvantage of these indexes is that they slow down inserting, deleting and updating of data.

Skip lists accelerate point queries without slowing down the transactions.

As indicated, SingleStore keeps table data in memory. It can use table scans to execute point queries, but even scanning large tables in memory can take a while. Therefore, SingleStore offers *skip lists*[1] to provide semi-direct access to in-memory data. Skip lists provide semi-direct access to single records or sets of records. Skip lists also work well with in-memory data. The key advantage is that they do not slow down the inserts, deletes and updates as much as Btree indexes would, making them very suitable for transactional workloads.

Strong Consistency for Consistent Data Views – SingleStore supports *strong consistency*, sometimes referred to as the *ACID* (atomicity, consistency, isolation, durability) property. Strong consistency implies that when multiple users access the same data at the exact same time, they always see the same data. Products that do not support strong consistency, but for example *eventual consistency*, cannot ensure this. Strong consistency is a necessity for many transactional applications.

Pipelines for Fast Data Ingestion – SingleStore supports several features to accelerate mass data loading. First, it offers a built-in ETL feature called *Pipelines*. Pipelines load data directly into the files while bypassing the SQL layer. When a table is partitioned, Pipelines enable *parallel loading of data*.

With Pipelines data can be loaded in parallel.

Second, more and more data is delivered in JSON (or a similar) format. This type of data can be loaded in two ways, it can be converted into flat relational structures before loading or it can be stored in a column with a JSON data type. The advantage of this second approach is that there is no delay caused by transforming each incoming record to a flat data structure during the loading process.

A Distributed, Shared-Nothing Architecture for Scalability – Processing individual transactions quickly is important, but it is equally important to process many concurrent transactions quickly. A unified database needs to support a high level of scalability and preferably *linear scalability*; when the number of concurrent transactions increases, individual users should not experience a performance drop.

[1] Wikipedia, *Skip list*, March 2021; see https://en.wikipedia.org/wiki/Skip_list

The level of scalability of some database products is limited because they are built around a few central processes that are responsible for most of the work, such as authorization checking, query optimization, lock administration, metadata management and merging the results of the worker processes. Such heavy central processes can easily become the bottleneck and minimize the database's scalability. Such a central process is comparable to one toll booth on a 6-lane highway. Just before the tool booth, all lanes converge forcing cars to pass the toll booth one by one. If there are not that many cars on the highway, there will be no problems, but when traffic increases, the toll booth becomes the bottleneck.

SingleStore supports a *distributed, shared-nothing architecture* that runs on a server cluster that offers near-linear scalability. SingleStore works with several aggregator processes. Each client application connects to an *aggregator process*. Each aggregator works with all the *leaf processes*. This enables independent scaling of user connections (through the aggregator processes) based on data size, query performance and throughput scaling (through the leaf processes).

SingleStore supports a distributed, shared-nothing architecture offering almost linear scalability.

The leaf processes manage the stored data. A SingleStore database is the sum of all the data stored by the leaf processes. Figure 9 shows a simplified view of SingleStore's internal architecture.

The processing of SQL statements is distributed across as many leaf processes as possible. In other words, when the performance starts to decline, more aggregator or leaf processes are started to keep the performance stable. Compare this to opening more toll booths that all operate independently.

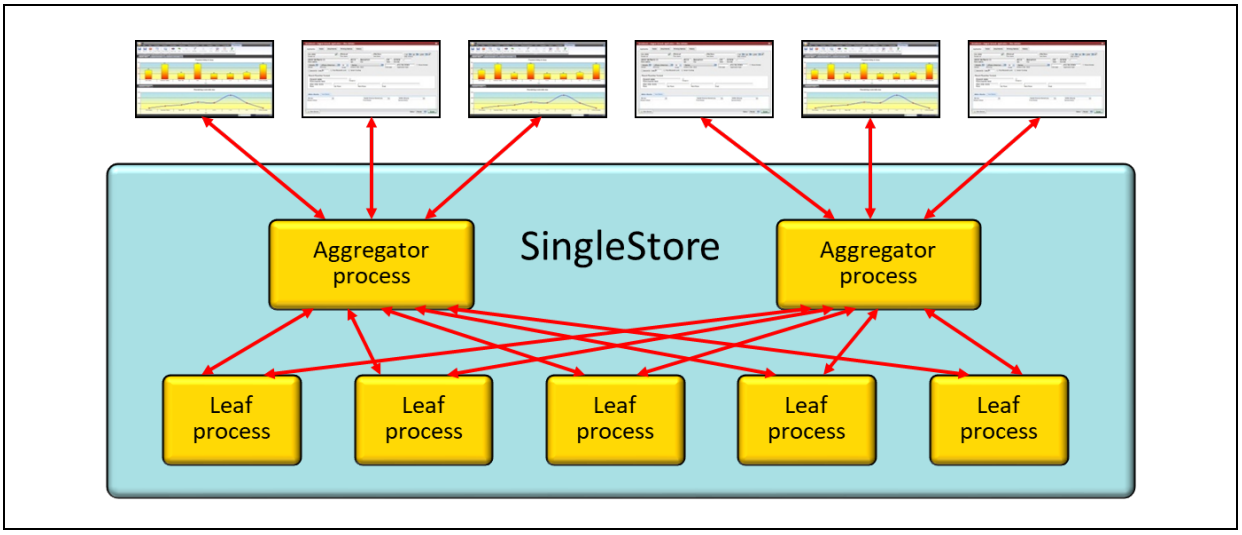

Figure 9 *The distributed, shared-nothing architecture of SingleStore support a high-level of scalability.*

6.3 Support for Analytical Workloads

Section 6.1 describes the universal storage tables type, which is designed to support an analytical workload. This section describes the additional analytical features that SingleStore offers.

Advanced Query Optimization Techniques for Fast Queries – Query performance predominantly depends on the *query optimizer* of the database. The smarter the query optimizer, the better the performance. SingleStore supports all the smart query optimization techniques found in other SQL products. The query

optimizer uses *cost-based models* and statistical information to determine the best *execution plan*. The query optimizer knows how data is distributed across all the leaf nodes.

Query Compilation for Fast Queries – An important difference with most other products is that SingleStore compiles the queries. For each query, a just-in-time *execution plan* written in the C++ programming language is generated and compiled to machine code. The plan is executed and kept in cache. No time is wasted re-optimizing the query the next time it is executed.

SingleStore compiles queries to avoid repeated optimization of similar queries.

This approach is especially beneficial for SQL statements that are repeatedly executed, such as those implemented in transactional applications and predefined dashboards used by large groups of concurrent users. To minimize the number of cached queries, the execution plan is parameterized. For example, if a query retrieves the address of customer C15, that customer identifier is parameterized so that it can be reused to retrieve customer C16.

Each SingleStore node has its own query cache to prevent writing to and reading from the cache from becoming a bottleneck. Execution plans remain in memory. When they expire, they are written to disk, from where they are retrieved when needed again.

Reference Tables for Fast Joins – Complex queries normally involve joining multiple tables. Some of those tables are large, such as the one with all the sales transactions, and others can be small, such as the table containing all the product categories. To avoid unnecessary join processing, where much data is sent between nodes, a table can be defined as a *reference table*. Each reference table is automatically stored on each node. The system always keeps these tables synchronized. Reference tables improve the performance of complex joins by minimizing the amount of data that needs to be sent between nodes.

Geospatial Data Type and Functions for Geospatial Queries – An increasingly popular category of queries are the *geospatial data queries*. SingleStore supports geospatial data types and functions to intelligently execute geospatial queries. For example, functions are supported that determine the distance between two points and determine whether two polygons overlap.

SingleStore supports geospatial data types, functions and queries.

For databases that do not support geospatial data types and functions it is more complex to formulate geospatial queries and it results in poor performance, as most of the processing is done within the applications and not by the database. With such databases, large amounts of data are sent to the application and the power of the database is not fully exploited. For example, if a query needs to know all the lines that intersect a specific line, SingleStore can find them all and only the selected lines need to be returned to the application. In other systems, all lines are returned to the application that needs to select the intersecting lines. SingleStore's approach reduces performance.

7 Closing Remarks

The Need for Unified Databases – The need to simplify data architectures, provide access to zero-latency data for analytics, be linearly scalable, and exploit cloud platforms has grown. For many years, general-purpose databases have served organizations very well. But for some new applications, good is no longer good enough. Specialized databases, such as analytical and transactional SQL databases, have definitely raised the bar for what can be achieved in terms of workloads. But they have not solved the problem of running analytical and transactional workloads concurrently. This is where unified databases come to the rescue.

The Key Features of SingleStore – SingleStore is a representative of this relatively new breed of databases. It supports the following features to support an analytical and transactional workload concurrently:

- Universal storage
- In-memory, record-oriented storage for transactions
- Skip lists for fast queries
- Strong consistency
- Pipelines for fast data ingestion
- Distributed, shared-nothing architecture
- Advanced query optimization techniques
- Compilation of queries
- Reference tables
- Geo-spatial data types and functions

The Advantages of SingleStore – The advantages of SingleStore are:

- The performance of existing systems can be accelerated
- Databases can be merged to simplify the entire data architecture
- A new workload that raises the bar with respect to analytical or transactional workloads, is easier to support
- Data warehouses can be merged with data marts lowering the data latency and simplifying the data warehouse architecture
- Existing transactional systems can be extended with an analytical workload
- Analytics-ready transactional databases can be developed
- The potential scalability of cloud platforms can be fully exploited
- Existing data architectures can be augmented to make them ready for new workloads

About the Author

Rick van der Lans is a highly-respected independent analyst, consultant, author and internationally acclaimed lecturer specializing in data architecture, data warehousing, business intelligence, big data, database technology and data virtualization. He works for R20/Consultancy (www.r20.nl), which he founded in 1987. In 2018, he was selected the sixth most influential BI analyst worldwide by onalytica.com[2]. He has presented countless seminars, webinars, and keynotes at industry-leading conferences.

Rick helps clients worldwide to design their data warehouse, big data and business intelligence architectures and solutions and assists them with selecting the right products. He has been influential in introducing the new logical data warehouse architecture worldwide which helps organizations to develop more agile business intelligence systems. He introduced the business intelligence architecture called the *Data Delivery Platform* in 2009 in a number of articles[3] all published at B-eye-Network.com.

He is the author of several books on computing, including his new *Data Virtualization: Selected Writings*[4] and *Data Virtualization for Business Intelligence Systems*[5]. Some of these books are available in different languages. Books such as the popular *Introduction to SQL* are available in English, Dutch, Italian, Chinese and German and are sold worldwide. Over the years, he has authored hundreds of articles and blogs for newspapers and websites and has authored many educational and popular white papers for a long list of vendors. He was the author of the first available book on SQL[6], entitled *Introduction to SQL*, which has been translated into several languages with more than 100,000 copies sold.

For more information please visit www.r20.nl, or send an email to rick@r20.nl. You can also get in touch with him via LinkedIn and Twitter (@Rick_vanderlans).

Ambassador of Axians Business Analytics Laren: This consultancy company specializes in business intelligence, data management, big data, data warehousing, data virtualization and analytics. In this part-time role, Rick works closely together with the consultants in many projects. Their joint experiences and insights are shared in seminars, webinars, blogs and whitepapers.

About SingleStore, Inc.

SingleStore is dedicated to helping businesses adapt more quickly, embrace diverse data and accelerate digital innovation by operationalizing all data through one platform for all of their moments that matter. These capabilities are provided as a service on Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform, Red Hat and through your own deployments with SingleStore Managed Service and SingleStore DB. Visit www.singlestore.com or follow us @SingleStoreDB and @SingleStoreDevs.

[2] Onalytica.com, *Business Intelligence – Top Influencers, Brands and Publications*, June 2018; see http://www.onalytica.com/blog/posts/business-intelligence-top-influencers-brands-publications/
[3] See http://www.b-eye-network.com/channels/5087/view/12495
[4] R.F. van der Lans, *Data Virtualization: Selected Writings*, Lulu.com, September 2019; see http://www.r20.nl/DataVirtualizationBook.htm
[5] R.F. van der Lans, *Data Virtualization for Business Intelligence Systems*, Morgan Kaufmann Publishers, 2012; see https://www.r20.nl/DataVirtualization_V1.htm
[6] R.F. van der Lans, *Introduction to SQL; Mastering the Relational Database Language*, fourth edition, Addison-Wesley, 2007.