Raising the Bar for Data Virtualization

A Whitepaper

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

September 2020

Sponsored by

Table of Contents

1 Introduction

The Requirements for Data Virtualization Have Escalated – Data virtualization technology has been accepted and the use cases are becoming more functionally richer. Gartner has indicated that data virtualization has reached the *Plateau of Productivity*. Data virtualization is not an exotic technology anymore, but has become a mature and widely accepted solution for many organizations.

Data virtualization has become mature technology.

However, since its years of introduction, the requirements for data virtualization technology have escalated for several reasons. First, organizations deploy data virtualization so differently from years ago. For example, the number of users accessing a data virtualization environment has grown; it is being deployed for a wider range of use cases; much more data is being processed; the complexity of applications and queries has increased; and the importance of metadata management has increased.

Secondly, because of the organization's cloud strategies, they want to run their data virtualization platforms on cloud platforms, such as Amazon, Google, and Microsoft Azure. This has introduced a new requirement: data virtualization platforms need to exploit the full power of cloud platforms. Technically, they should be able to really exploit the enormous potential of cloud platforms with respect to the almost limitless performance and dynamic up and down scalability. Additionally, data virtualization platforms need to offer a flexible, pay-by-the-sip fee structure dependent on resource usage. Organizations expect data virtualization platforms not to run 'on' a cloud platform, but 'inside' it.

Extending Data Virtualization – Due to this success, organizations have started to push the limits of what a data virtualization platform should support. As a result, vendors of data virtualization platforms need to raise the bar with respect to many features of their products. In particular, the following ones should be extended and improved:

The bar for data virtualization is raised.

1. Limitless scalability
2. Leverage of cloud platforms
3. Functional extensibility
4. Streaming data
5. Fast software development
6. Transaction processing

This Whitepaper – This whitepaper focuses on these six features. The first two deal with performance and scalability and how well the products can exploit the potential power of cloud platforms. The third, functional extensibility, relates to the ability offered to customers or other parties to extend the product's functionality. Feature four, streaming data, relates to the increasing user demand for working with (near) real-time data. Vendors must be able to improve and extend their product quickly, which is feature five. Allowing data virtualization users to insert, update, and delete on the underlying data sources is the sixth feature is called transaction processing. Each feature is described in detail and the importance is explained.

This whitepaper also describes how *Intenda* has improved the internal architecture of the *fraXses* data virtualization platform and how the product deals with these six features to make it future-proof and suitable for new requirements and workloads.

2 New Data Virtualization Requirements

Raising the Bar for Data Virtualization – Organizations keep raising the bar for all kinds of IT technologies. This is especially true now that we have entered the *big data era*. More data needs to be processed, more users need to be supported, and their requests are becoming more and more complex. This is experienced by vendors of all kinds of technologies and tools, including data science tools, database platforms, messaging technology, and data warehouse automation tools.

Evidently, vendors of data virtualization platforms recognize this customer-push as well; they cannot rest on their laurels. As the bar for data virtualization is raised, they must extend and upgrade their products in all areas. This section describes the key data virtualization features that require extending or upgrading to make them future-proof. Note that this is not an exhaustive list, but these are the key features.

Limitless Scalability – As indicated, the bar is also raised with respect to the workload. New applications accessing a data virtualization platform must support more users, handle more and more complex queries, and process much more data. To make sure that a data virtualization platform does not become the bottleneck in the entire architecture, it must somehow support nearly limitless scalability.

Leverage of Cloud Platforms – Cloud platforms, such as Amazon, Google, and Microsoft Azure, are becoming the preferred platform for most forms of data processing. This is also true for data virtualization. In principle, cloud platforms offer practically limitless resources for storage, processing, and memory. Data virtualization platforms must be able to exploit all these resources to avoid becoming the bottleneck in large environments. Additionally, they must be as resource-efficient as possible to keep the costs low.

Functional Extensibility – As with other products, each new version of a data virtualization platform offers more functionality. Most new functionalities are useful for a large group of organizations. It is important that organizations with very specific functional requirements can easily extend the data virtualization platform themselves. For example, if organizations need to use a very specific and homemade engine for financial calculations, it must be possible to embed that engine within the data virtualization platform.

Streaming Data – There was a time when real-time data analytics was exotic or special, but not anymore. An increasing number of data consumers demands real-time data (or zero-latency data). For some, every second or microsecond counts. When data arrives too late, it may introduce problems or kill an opportunity. We have clearly entered the age of real-time analytics and reporting. Currently, most data virtualization platforms do not support real streaming of data. Most data virtualization platforms require that data is first stored before it can be accessed and processed. Supporting these real-time requirements for streaming data becomes a necessity.

Support for real-time data streaming has become a necessity.

Fast Internal Development – Development in the software industry is not slowing down and organization requirements are getting more complex. It is important that vendors find ways to speed up their own internal software development. Developing all the new functionality itself, may not be fast enough. Other ways must be deployed that lead to faster development of data virtualization functionality, for example, by allowing other parties to develop modules.

Transaction Processing – Currently, the majority of data virtualization platforms is used in read-only environments, such as business intelligence and data science environments. Data is extracted from source

systems and delivered to the data consumers. The number of environments that wants to use data virtualization for inserting, updating, and deleting data in the source systems as well is growing. Therefore, data virtualization platforms need to support *transaction processing* or *write back*. Transaction processing must be as independent of storage technology as query processing already is.

As indicated, this whitepaper focuses on these six features. Others are also important, such as the ones described below.

Query Optimization – It is predominantly the intelligence and smartness of a data virtualization platform's *query optimizer* that determines the overall performance. Therefore, improving the query optimizer is an ongoing process for the vendors. It started on the first day of product development and will continue until the last day of a product's existence. Data virtualization vendors have always devoted much research and development time on improving the query optimizer, and they can't stop now. Examples of several areas where query optimization is expected to improve the coming years:

- More techniques for optimizing distributed joins.
- AI-based algorithms to optimize queries based on the performance of the previous execution of similar queries.
- Dynamic query optimization that changes the execution plan midflight when initial assumptions turn out to be incorrect.

Caching Data – Caching mechanisms and features must be extended. Examples:

- More ETL-like caching mechanisms are needed in case the amount of data to be cached is too extensive and the update frequency is too high for a more classic caching approach.
- Live refreshing is necessary to keep cached data as current as possible.
- Caching mechanisms are needed that keep track of history in case the source systems do not support this feature.

Data Source Support – The list of data sources accessible by data virtualization platforms grows continuously. This will and should not stop in the foreseeable future. Additionally, the modules (sometimes called connectors or adapters) of data virtualization platforms accessing the data sources must leverage the performance and functionality of the sources as good as possible. An additional new requirement may be that these modules 'understand' the cryptic source data and transform it automatically into meaningful data. Support for accessing data created by packaged applications needs to be extended as well.

Data Security and Privacy – Due to increasing organization needs for protecting data from unauthorized and improper use, data virtualization platforms need to implement more built-in data protection functionality. For example, from a data privacy perspective, sophisticated functionality for pseudonymizing and anonymizing is required.

Metadata and Master Data – Data virtualization platforms need to offer more functionality for entering, storing, and managing user-defined and business-oriented metadata. For example, functionality for importing metadata specifications from other tools is required. Besides an integration and delivery platform for data, a data virtualization platform must become an integration and delivery platform for metadata as well.

3 Requirement 1: Limitless Scalability

Limitless scalability is the first feature that needs to be extended and is described in detail in this whitepaper.

Porting to Cloud Platforms – According to the Gartner's hype cycle[1], data virtualization has reached the *Plateau of Productivity*. This means that the technology has matured and is being deployed in a wide range of use cases. Now that data virtualization has been fully adopted by the market, the first customers are expanding their use cases and are deploying data virtualization in more intense workloads. Also, new customers are using data virtualization for more complex and data-heavy systems. In both cases, it means that a data virtualization platform needs to be able to handle more users, more requests, more data, more data sources, and more complex requests. This requires that the internal architecture of a data virtualization platform offers a high level of scalability to 'grow' in line with the growing customer demands.

A data virtualization platform needs to run not on *but* within *a cloud platform.*

As indicated, cloud platforms offer almost limitless resources. Is making a data virtualization platform available on cloud platforms sufficient to provide that level of scalability? Not really. Without many changes, most software programs and tools, originally developed for on-premises deployment, can be ported to cloud platforms. Being able to run on cloud platforms has several benefits, however, a straightforward port may not result in leveraging the potential performance and scalability offered by these platforms. Running *on* a cloud platform is not sufficient, running *within* the cloud platform is the goal. A data virtualization platform must exploit cloud platforms to the max. Additionally, the product must be as resource-efficient as possible. Just throwing resources at the solution to get a higher level of scalability, can make the solution expensive.

Whether a data virtualization platform is able to fully leverage the potential scalability of a cloud platform depends primarily on its internal architecture. The two main challenges are (1) to distribute the processing across as many processors and/or cores as possible to get a high parallelization level and (2) to distribute processing across modules as efficiently as possible to avoid skewed usage of processors, which can lead to one or more processors becoming the bottleneck. This section describes some of the internal architectures supported by data virtualization platforms and how well they support parallelization.

Multi-Threaded Architecture – Multi-threading is a classic technique that allows processing of requests to be executed in parallel; see Figure 1. Instead of having one single process that executes all the processing on one processor (or core), the work is divided over multiple processes (*threads*). In such an architecture, there is always a central, *master process* that receives all the user requests which it to the threads for processing.

How the processing work is distributed between the master and the threads has a major impact on how efficient parallel processing is. Imagine that a SQL database platform executes queries by performing the following tasks: syntax analysis, user authorization checking, view substitution, query optimization, and query execution. Technically, it may be possible to push all this processing to the threads, which means that most of the work is processed by the threads and executed in parallel, which improves the overall workload. This definitely raises the level of parallelization. However, if only query

[1] Gartner, *Hype Cycle for Data Management, 2019*, July 2019; see https://www.gartner.com/en/documents/3955768/hype-cycle-for-data-management-2019

execution is performed by the threads and the rest by the master processes (as is the case in Figure 1), less processing is executed in parallel and the master could become the bottleneck.

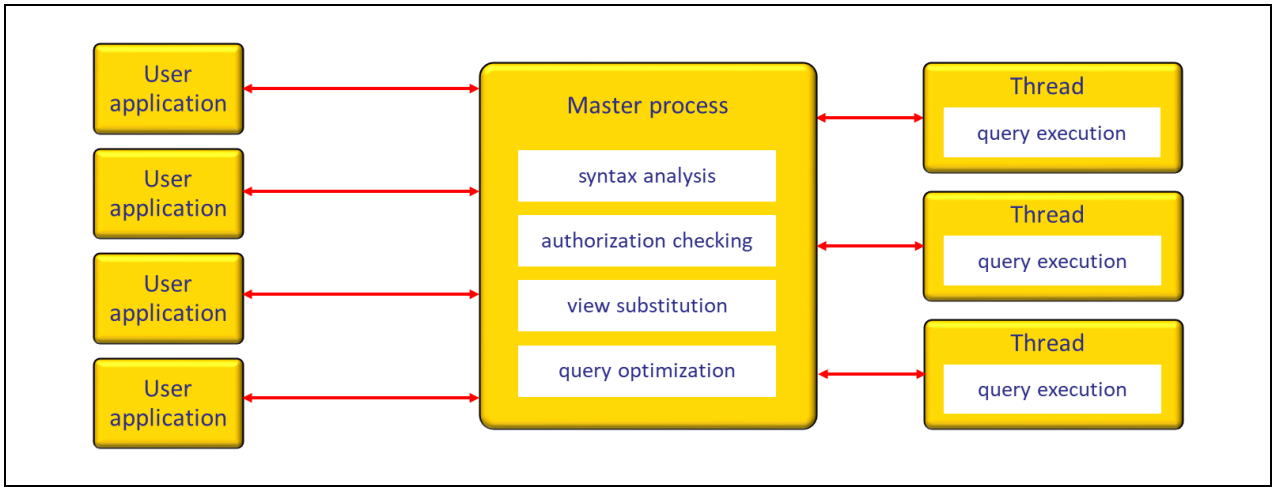

Figure 1 *In a multi-threaded architecture processing of requests is ditributed between a master process and some paralell threads.*

The three main drawbacks of a multi-threaded architecture are: the central master process could become the bottleneck and the single point of failure; the number of threads that can be handled is normally limited and therefore cannot be scaled to hundreds of parallel processes; and it is a resource hungry architecture. The performance is determined by how much of the work is executed by the threads and how much is done by the master process.

Note: From the perspective of query execution, this architecture offers what is sometimes referred to as *inter-query parallelization*. Parts of the query are not automatically processed in parallel.

Multi-Instance Architecture – The *multi-instance architecture* for server products is quite a heavyweight solution. Multiple instances of the full product are executing in parallel; see Figure 2. This architecture needs a load balancer that receives all the user requests and distributes them across the instances. The number of active instances determines the level of parallelization. In principle, each instance itself can have a multi-threaded architecture.

The drawback of this architecture is that the instances are 'large' and resource-hungry processes. Each one is responsible for executing many tasks; as shown in Figure 2. All those tasks are operational inside the instances. However, the functionality of all the tasks is not continuously required. For example, the task responsible for executing queries is more frequently used and consumes much more resources than the task retrieving metadata to check, for example, authorization rules. Unfortunately, these tasks of an instance cannot be independently scaled.

Starting multiple instances for some tasks, such as query optimization, can make sense, because that feature is used frequently by all requests. But this does not apply to other tasks that are less frequently used, such as the task to create new views. But since they are all part of the instance, they are all replicated.

On a cloud platform, it is important that modules are started and stopped dynamically when the module is needed. Because of their size, starting and stopping full instances takes time. As indicated, these are not lightweight processes.

Note: As with the multi-threaded architecture, this architecture offers *inter-query parallelization*.

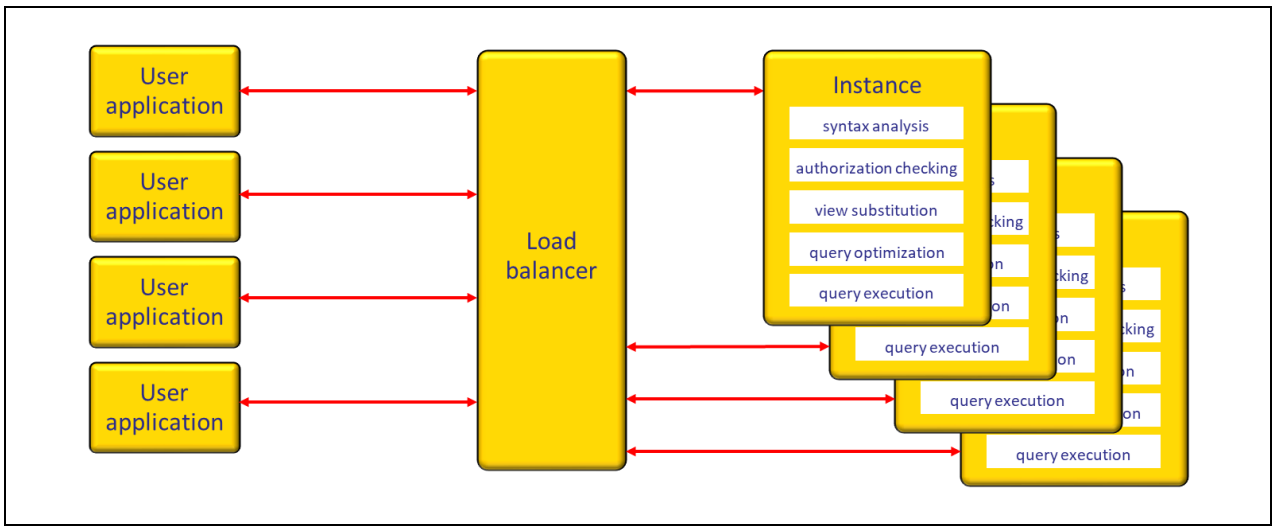

Figure 2 *In a multi-instance architecture processing of requests is distributed by a load balancer across heavy-weight processes.*

Microservices Architecture – An internal architecture that is very suitable for limitless scalability is the *microservices architecture*; see Figure 3. In a microservices architecture all the tasks are not implemented in one big monolithic process, but in different services that all operate independently of each other. On the operating system level, the product consists of many services each functioning as separate processes. For each service several *service instances* can be started. A scheduler or orchestrator exists to receive all the incoming requests. The orchestrator then passes the requests to the necessary service instance. This service instance processes the request. When it needs other services, it sends them requests. Communication between the services is normally implemented using lightweight protocols, such as JSON/REST and Apache Kafka, to send requests and replies back and forth between the service instances.

A microservices architecture can offer limitless scalability.

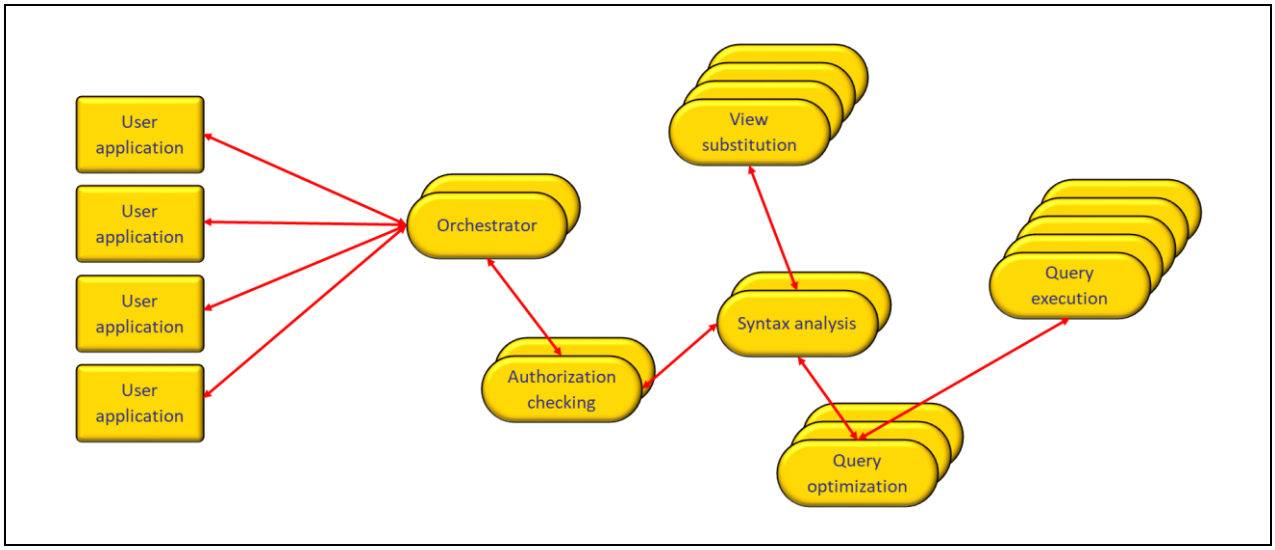

Figure 3 *In a microservices architecture processing of requests is distributed across independent service instances.Different services can have different numbers of instance active.*

The orchestrator determines how many instances of each service need to be started. Of each service, a different number of services can be started, allowing the ones with an expected heavy workload to have more instances than the ones invoked occasionally. Since services are lightweight processes, they can be started and stopped easily and quickly. This makes it easier to dynamically adjust the number of operational service instances based on the workload changes.

A microservices architecture needs a mechanism to receive the requests and invoke the service instances. Two popular mechanisms exist: *orchestration* and *choreography*. Orchestration uses a centralized approach, in which the communication between services is handled by and passes through the central orchestrator. It's like a parcel service where all the received parcels are first shipped to a central warehouse and from there distributed to the recipients. Choreography gives services more freedom to execute requests. The services can interact with each other directly without some centralized module. It would be like everyone sending their parcels directly to the recipients without a central warehouse. Technically, both mechanisms can be used by a product.

Serverless Architecture – Server processes, such as database servers and data virtualization platforms, have a tendency to be started and then wait for requests. When no requests are received, they still run and consume memory, I/O, and CPU resources. A product with a *serverless architecture* does not use server processes. In the ideal world, when such a product does not receive any requests, all the processes are stopped. The next request wakes up the product.

The term serverless causes some confusion, as it can be interpreted as if a product does not run on server machines. This is not correct, serverless means that the product operates with a minimum number of continuously operating server processes. Microservices architectures are normally serverless. If a product has a serverless architecture, it may still access server processes, such as a database server. So, serverless does not mean that there are no server processes, but as few as possible.

A serverless architecture minimizes resource consumption when no requests are processed.

Parallel Query Processing – The multi-threaded, multi-instance, and the microservices architectures are able to execute queries in parallel; inter-query parallelization. However, the execution of one individual query itself is not by definition parallelized. If a query accesses tables with massive amounts of data, it can still take a long time to process. The architecture of a data virtualization platform also needs to support parallel execution of individual queries. This is called *intra-query parallelization*. Especially big data environments demand this form of query execution.

4 Requirement 2: Full Leverage of Cloud Platforms

Every study shows the increasing use of *cloud platforms* for data processing. There are three dominant reasons why organizations invest in cloud platforms:

- Unburdening the organization with regards to management.
- The enormous potential with respect to performance and scalability.
- A flexible fee structure based on resource usage.

This adoption of cloud platforms implies that data virtualization platforms must be available and, more importantly, harness the full power of these platforms. This section explains what this means for data virtualization platforms.

Unburdening the Organization – For many organizations the main benefit of cloud computing is *unburdening*. Take a BI environment as an example. Designing, developing, running, managing, tuning, and changing BI systems requires many software components and a wide range of analytical and technical skills. This is a heavy burden for an organization. Specialists must be employed or hired; machines must be acquired and installed; data needs to be backed up and recovered; new versions of the software must be installed, data center space, computing power and desk space must be made available, and so on. These specialists need to be trained when new products are installed and to keep their analytical skills on par. Running a BI environment on-premises involves a major investment.

By migrating IT systems to the cloud, much of the work described above is no longer of concern to the organization. That work is outsourced to the cloud platform and its vendors and this unburdens an organization.

Limitless Scalability and Performance – Cloud platforms are basically extreme *massively parallel processing* (MPP) environments allowing organizations to exploit almost endless pools of resources. Many tools and applications that were initially developed for on-premises usage, have been made available to run on one or more cloud platforms.

Being able to 'run' on a cloud platform does not mean that the product automatically exploits the full power of the platform. For example, it is very likely that a product with a multi-threaded architecture is limited in the number of threads it can activate. It is normally limited to one or two dozen threads. In other words, it cannot exploit all the processing power available on the cloud platform. This is especially true if most of the tasks are executed by the master process and not by the threads.

It is important that the internal architecture of a data virtualization platform really exploits all the power. In general, a microservices architecture fits the architecture of cloud platforms best, the multi-instance architecture is second best and the multi-threaded architecture is third. A microservices architecture makes products elastic. Depending on changes in the workload, the product can adapt itself by starting and stopping service instances. Starting a service instance is fast and does not degrade performance. A data virtualization platform with a completely monolithic architecture can barely exploit MPP.

Data virtualization platforms need to exploit the full power of cloud platforms.

A product developed for the cloud must have an internal architecture that can *scale up*, meaning when more resources are required, they are dynamically added. But it also should be able to *scale down* when resources are no longer required, because the workload has decreased. A data virtualization platform must be able to scale up and down dynamically. Scaling down needs to be handled gracefully and should never lead to cancellation of running requests.

Flexible Fee Structure – The flexible fee structure offered by cloud platform vendors is sometimes referred to as *pay-by-the-sip*. Organizations only pay for the use of resources, such as I/O, memory, and computing. Ideally, when a product is not processing requests, it should consume zero resources, meaning no charges.

The multi-threaded and multi-instance internal architectures are definitely not serverless architectures. As a result, their server processes are continuously operating, even when no requests are received, and the organization pays for resource usage. In this case, the cloud vendor may support pay-by-the-sip, but the server process is continuously drinking. It is like paying for a movie in a cinema, but not watching it.

A product with a microservices architecture does not consume any or just a minimal amount of resources when it is not executing any requests. And when the number of requests drops, service instances are automatically stopped.

How the product supports scale up and down also influences the costs. When a system does not scale down automatically, but only manually, the delay could lead to unnecessary resource consumption.

Efficient use of cloud platform resources reduces chargeable costs.

The bottom line is that efficient use of cloud platform resources reduces chargeable costs. Therefore, data virtualization vendors must develop their products to minimize resource consumption to dynamically adapt to changing workloads.

Multi-Cloud Strategy – Studies show that many organizations deploy a *multi-cloud strategy*, which means that an organization uses multiple cloud platforms. Flexera[2] reports that "in 2019, 84% of the organizations rely on a multi-cloud strategy." And according to InformationAge[2] "60% of all businesses in the sector expect their IT environment to be multi-cloud, integrating both on-premises and externally hosted cloud infrastructure. Only 18% say they will solely rely on the public cloud." Organizations may have chosen for a multi-cloud strategy on purpose, but it may also be 'enforced' due to, for example, an acquisition. It could also be that organizations have different cloud platforms in use, because some of their software is only available on specific cloud platforms. Sometimes this is a conscious choice, for example, to be able to migrate to another platform when the other one offers more functionality and/or is more interesting in terms of price.

A multi-cloud strategy demands that data virtualization platforms are able to operate on all the different cloud platforms. Data virtualization platforms should not restrict organizations from deploying particular cloud platforms.

5 Requirement 3: Functional Extensibility

Built-in Functionality – Each data virtualization platform comes with many features for extracting, transforming and securing data. They allow developers to use the full power of the popular SQL language. And with some, if SQL is not sufficient, developers can use procedural SQL or lower-level languages, such as Java and C#, to develop functionality.

But what if all that functionality is not sufficient or an efficient and fast solution cannot be developed with the existing functionality? For such situations, it must be possible to extend the existing functionality. One way to do this is to allow developers to extend the built-in functionality by developing stored procured and functions. Unfortunately, some required extensions can be very hard to develop with procedural SQL and if it works, it may be inefficient.

Therefore, a mechanism is required with which external modules can be embedded within the data virtualization platform. It is important that these modules are regarded by the data virtualization platform as *first-class citizens*. For example, if the platform uses a load-balancer for all its own modules, then the external module must be managed by the load balancer as well; if multiple instances of an internal module can be started to support scalability, then it must be possible to start multiple instances of the external module as well; and if specific security rules can be defined for internal modules, it must be possible to do the same for the external module. In other words, data virtualization platforms should not make a distinction between its own and external modules.

[2] Hostingtribunal.com, Cloud Adoption Statistics for 2020; See https://hostingtribunal.com/blog/cloud-adoption-statistics/#gref

Below three examples of modules are described that are too complex to be implemented with the standard features of most data virtualization platforms.

Example 1: Data Science Models – Imagine that a organization has developed an advanced data science model using machine learning techniques that predicts churn risks for customers. This module can be transformed into a service. It must be possible to embed this service as a first-class citizen module and the data virtualization platform should treat it as one of its own native modules. Developers should be able to invoke it as any other module.

Example 2: Data Privacy – Almost every organization stores *personally identifiable information* (PII) and more and more laws and regulations exist for governing the storage and processing of such data. Technically, this means that data needs to be pseudonymized and anonymized. Imagine that social security numbers (SSN) of people must be transformed into an abstract value that does no longer identify them. Additionally, the transformation of SSNs must be consistent across the tables to make it possible to integrate data from multiple tables using the abstract values. It may also be needed to mask certain column values to show only specific parts of the data. Functionality may be needed that uses statistical techniques to determine whether a data consumer can still identify to whom the data belongs. For example, if the employee data also indicates whether someone has won a Nobel prize, and if the result of a report shows only one employee who has been awarded a Nobel prize, it is fairly clear who that is, as most companies do not have that many Nobel prize winners on their payroll, let alone two.

Most existing data virtualization platforms do not offer such features. External modules are needed to support such advanced capabilities.

Example 3: GEO/GIS – More and more applications need support for GEO/GIS functionality. This type of functionality should not be limited to supporting data types, such as 2D and 3D points, lines, circles, and shapes. In addition, functions are required that can process this data intelligently. Examples of such functions are: calculating the distance between points, determining whether areas overlap, mapping zip codes to 2D coordinates, and calculating travel time between points by public transport, car, or foot. Such functions can consist of hundred thousands of lines of code. It would involve quite a number of developers to develop and maintain this. A data virtualization platform needs to be able to embed such modules and make access to them as transparent and simple as access to native functions.

Summary – In particular, very specific customer demands require data virtualization platforms to support a plug-and-play architecture that allows customers to embed their own modules. Potentially, this can lead to a market of homemade modules for a data virtualization platform. For example, if one customer develops a perfect pseudonymization module, it could be made available to other customers.

6 Requirement 4: Processing of Streaming Data

The Era of Streaming Data – For an increasing number of data consumers, data needs to be real-time. For some, every second or microsecond counts. When data arrives too late, it can cause business problems or an loose opportunities. We have entered the era of streaming data and real-time analytics and reporting. There was a time when real-time analysis was exotic or special, but not anymore.

This trend towards real-time data processing is in line with the consumer market for streaming audio and video services. Almost all social media platforms are based on real-time streaming of data to readers and viewers. As a result, more and more data consumers using data virtualization platforms are

demanding access to real-time data. In other words, they expect the data to be streamed.

Different Approaches of Data Delivery – Technically, several approaches exist for delivering data from source systems to data consumers; see Figure 4. These approaches differ in various ways, but this chapter focuses on their difference with respect to data latency as experienced by data consumers. Here, *push* means that data is copied/moved from a source to a target and the former initiates this process. With *pull,* the target initiates the copying/moving of data; it asks for the data.

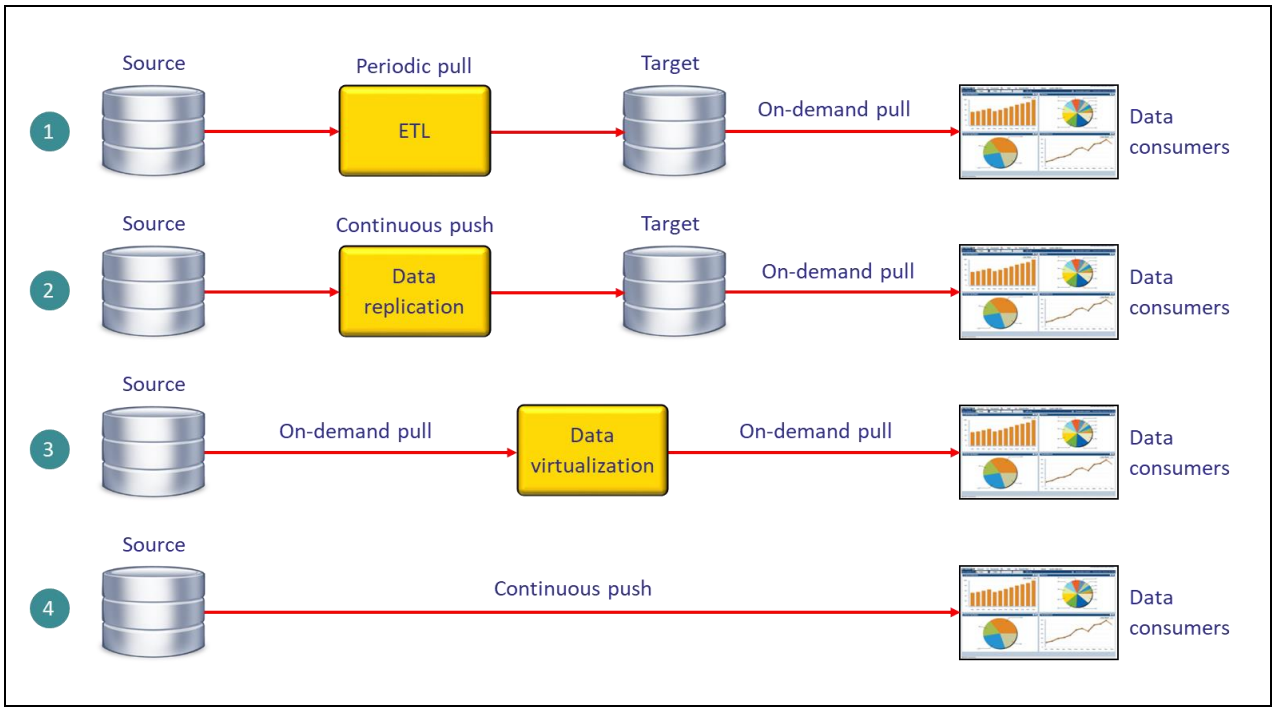

Figure 4 *Different approaches exist for delivering data to data consumers.*

Periodic-pull-and-on-demand-pull – For a long time, ETL has been the primary technology for pushing data from data sources to data consumers. An ETL solution offers a *periodic-pull-and-on-demand-pull approach*. The ETL tool pulls the data in batches (periodically) from source systems and copies it to target databases, such as data warehouses and data marts.

Next, when data consumers need data, it is pulled (on-demand) from the target database. Due to the periodic approach, it may take quite some time before new data is pulled towards the target. In other words, the data latency is high.

Continuous-push-and-on-demand-pull – The second style offers a *continuous-push-and-on-demand-pull approach*. Data replication technology may be used to push data continuously in a near real-time fashion from the source to a target database. Data consumers pull data (on-demand) from the target database. With this approach, the database is close to up-to-date. The frequency of pushing the data to the target determines the data latency experienced by data consumers.

On-demand-pull – Data virtualization offers an *on-demand-pull-approach*. Data is pulled from the source by the data virtualization platform when it receives a request from a data consumer. Extracting data from a data virtualization platform is also based on a pull approach, because data consumers ask the data virtualization platform for the data. The data store accessed by the data virtualization platform

determines the latency of the data. For example, pulling data from a data mart offers a higher data latency than pulling it from a transactional system.

Continuous-push – Real-time data consumers require a real *continuous-push approach*. When data is entered in a source system, somehow it needs to be pushed live to the data consumers. In this case, data consumers do not ask for data, it is constantly pushed to them. This is the world of *streaming data* and streaming analytics. At real-time organizations can analyze business processes and react to that instantly.

Streaming Data and Data Virtualization – A new requirement for data virtualization products is to support the *continuous-push* approach; see Figure 5. A data virtualization platform must be able to 'listen' to incoming data streamed from certain source systems or messaging technologies. Next, it needs to be able to process that incoming data. For example, the incoming data stream must be transformed, extended with other data, decrypted, or integrated with another stream. Such operations on a stream should be defined within the data virtualization platform, preferably transformed in the same way as pull data.

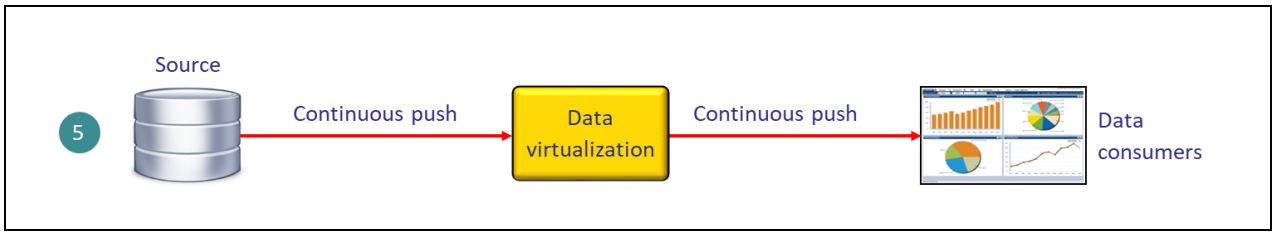

Figure 5 *A style of data delivery in which data virtualization transforms and processes streaming data*.

Next, data virtualization platforms need to be able to push the transformed data onwards to data consumers without any form of data storage. In this case, data consumers need to be in listening mode to be able to respond to that incoming data.

Ideally, it should be transparent to data virtualization developers whether they define transformation rules for pulling data or for streaming data. Except that for streaming some extra specifications may be required that are specific to streaming.

7 Requirement 5: Fast Software Development

From In-house to Outsourced Development – Now that the market has discovered data virtualization and the interest is growing, the customer base is growing and the vendors are getting more requests for functionality. For example, organizations may be asking for support for a GraphQL interface, for master data management capabilities, data pseudonymization functionality, data preparation features, and GEO/GIS support. In addition, data virtualization platforms operate in a network of other technologies, such as database servers, packaged applications, messaging products, enterprise service busses, ETL tools, and BI tools. These product categories are changing and evolving as well, and all this impacts data virtualization.

All these changes mean that functionality must be improved and added to data virtualization platforms much faster. The question is whether the development team can develop all the new features by itself or should a different approach be applied? Can a vendor keep up with all the new software and hardware developments or does development have to outsourced in some way?

In many other industries it has become the norm not to develop all the components themselves, but to assemble them. Most vendors have become assemblers. For example, in the car industry, companies such as Volkswagen do not develop their own shock absorbers and GPS systems. This is left to specialized companies who do research and development for those specific components. Those vendors can divide research and development costs across multiple customers. Bosch, for example, develops and sells sensors that are used by several car manufacturers. The component manufacturers will keep on improving their components. Evidently, the car manufacturers inform the component developers what the required specifications are.

An Extensible Architecture – To be able to support all the new organization requirements, data virtualization platform vendors must somehow mimic the outsourcing approach of product development. Data virtualization vendors cannot develop everything in house. This means that that the product needs an internal architecture that allows modules from other vendors to be incorporated.

This approach of development increases the development speed for several reasons. The group of developers working on the data virtualization platform increases. Imagine that a data virtualization platform needs a scheduler to simulate ETL functionality. The development group needs to be extended with specialists in this field. They need to study the topic and determine how to add this functionality to the main product. If the data virtualization platform would allow existing ETL modules to be embedded, the functionality can be added much faster. Indirectly, more people will be working on the data virtualization platform. The vendor supplying the ETL module will probably have specialists in this area who are likely able to develop this functionality more quickly. Additionally, they can continue to improve this module independently of the data virtualization vendor. The latter can adopt a better version when needed. Note that these adopted components can be open source or commercial software.

8 Requirement 6: Transaction Processing

Most data virtualization platforms are deployed in read-only environments. For most current data consumers this is sufficient. For their use cases, there is no need to insert or update data using the data virtualization platforms. This will change with the coming of concepts, such as the microservices architecture and *data fabrics*. Future data consumers need to be able to change data as well. For example, an app used by organizations may allow them to change their own address or to place new orders. Another example is a more BI-type environment in which users want to enter budgeting and forecasting data themselves to be able to compare the real data coming from a data warehouse with the budget and forecast numbers.

Data virtualization platforms need to support transaction processing.

Data virtualization platforms need to support *transaction processing* (or *write-back*) functionality. The added value of changing data with a data virtualization platform is, as always, *data abstraction*. In this case, developers of the applications can use SQL or a service-interface of the data virtualization platform to insert, update, and delete data. They are completely shielded from how data is stored, where it is stored, what technology is used for storing data, and what the interface and language is. This speeds up development and makes it possible for organizations to change data storage technologies or the data structures without having to change application code.

9 How fraXses Supports New Requirements

Introduction to fraXses – *fraXses* is one of the newer data virtualization platforms on the market. It was introduced in 2015. It supports all the features expected from a data virtualization platform, such as:

- It allows for *data federation* of a heterogeneous set of data sources, ranging from SQL sources via Hadoop, packaged applications, and NoSQL to simple files and spreadsheets.
- It uses SQL-99 as the key language for defining all the transformations, filters, aggregations, joins, and cleansing operations to be applied to a data source. Most development is done through an intuitive, non-technical interface that generates data source specific queries.
- Data security rules can be defined centrally.
- It uses virtual tables as the key building block. These are called *data objects* in fraXses. These virtual tables can be accessed through different technical interfaces, including JDBC/SQL and JSON/REST.
- Dependencies between virtual tables can be presented graphically allowing for detailed *lineage and impact analysis*.
- Virtual tables can be cached to minimize access to data sources.

See also the whitepaper *Unifying Data Delivery Systems Through Data Virtualization*[3] for a more detailed description and general overview of fraXses.

In the summer of 2020, a new version was released with a new microservices architecture, supporting the new data virtualization features described in this whitepaper:

- Limitless scalability
- Leverage of cloud platforms
- Functional extensibility
- Streaming data
- Fast software development
- Transaction processing

Limitless Scalability – The previous version of fraXses was already based on a microservices architecture. The architecture of the current version has been significantly improved to support this flexible and scalable architecture, to offer (almost) limitless scalability, and to exploit cloud platforms.

fraXses has a microservices and serverless architecture.

When developing microservices, engineers need to find the perfect balance between the number of network messages between the services and the size of the messages. Too many small services sending many inter-service messages can decrease performance and scalability, and the same applies to large services with a limited number of inter-service messages (semi-monolithic). In the new version of fraXses this balance has been optimized. Functionality has been rearranged across services and more functionality has been implemented as services. The internal architecture of fraXses is primarily a *serverless architecture*.

The product uses an *orchestrator* to distribute the work across all the service instances. It is responsible for starting more instances of a service when the workload demands it. It stops service

[3] R.F. van der Lans, *Unifying Data Delivery Systems Through Data Virtualization*, October 2018; see https://www.r20.nl/WhitepaperFraXsesUnifiedV1.pdf

instances when the workload decreases. This mechanism allows fraXses to offer dynamic up and down scalability. Starting and stopping services can be done manually or automatically. The entire product does not have to be stopped for this. The orchestrator uses a *round-robin* and *load-on-service* approach to divide the work across multiple instances of the same service.

Apache Kafka is used internally to optimize data traffic between service instances. This allows fraXses to exploit and benefit from the performance and robustness of this messaging technology, and also from the cloud platforms.

Another feature related to limitless scalability relates to speeding up queries on big data. Queries that need to process massive amounts of data are pushed by fraXses for execution to Apache Spark. The benefit is that Spark distributes query processing across available cores and memory. The effect is that when fraXses runs on a cloud platform with an almost limitless amount of cores and memory available, theoretically, query execution can be distributed across hundreds of cores.

Full Leverage of Cloud Platforms – With its microservices architecture, fraXses can leverage the parallelization capabilities of cloud platforms. In such an environment, the challenge for a product is to manage all the instances of all the services. Instead of developing a proprietary solution, fraXses opted for proven technology called *Kubernetes*. The group developing Kubernetes defines the product as follows: "Kubernetes itself is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available." The technology was initially developed by Google.

fraXses uses internally Kubernetes and Docker.

FraXses also uses *Docker*, a container system that is available on all the popular cloud platforms and other platforms, including *bare metal servers* (also so known as *single-tenant servers*). Being a container system and not a virtual machine, it is much lighter and typically consumes less resources than virtual machines.

Due to the microservices architecture, the use of technologies such as Docker and Kubernetes, fraXses can benefit from the flexible fee structure (pay-by-the-sip) of the cloud. When the workload decreases, service instances are automatically stopped which minimizes resource consumption. In fact, if there is no workload at all, it is only the orchestrator and certain other services that are still operational, but those are small services consuming little resources.

During the execution of services, fraXses logs every message and service invocation. If the workload increases, the amount of logging can become massive. If fraXses is installed on a cloud platform, it uses the platform's native storage technology for logging and it uses *Ceph* for bare metal deployment. Ceph is an advanced storage platform for object, block, and file-level storage. It uses software abstraction layers to decouple the data from the physical hardware storage. It integrates well with technologies such as Kubernetes and microservices architectures, making it fit well with the fraXses architecture.

Functional Extensibility – fraXses can be extended with services developed by customers or other parties. The examples described in Chapter 7 can all be implemented as services to become an intrinsic part of fraXses. Externally developed services become first-class citizens services in the fraXses microservices architecture. Developers will not notice the difference between original fraXses services and externally developed services.

fraXses works with so-called *smart wrappers* in which all services are executed, including their own. Through those wrappers external services inherit all the features of the fraXses microservices architecture. For example, when a new service communicates with other services it uses Kafka, but note that service developers do not need to know, this is all

Smart wrappers are used to embed external modules.

hidden from them. In fact, if it is ever decided to change the messaging technology fraXses uses, developers do not have to change one line of code of the service, they can be embedded without adjustments. This is all hidden by the smart wrapper. The orchestrator manages these services as well and they can also dynamically scale up and down. Another feature that all the services inherit is, for example, that messages sent are also automatically encrypted and logged. But again, if it is decided to replace the current storage technology by another storage, it has no impact on the services.

The smart wrapper technology allows developers to write the services in their preferred language. Java, Python, Scala, and Rust can all be used.

Streaming Data – For streaming data applications, fraXses supports specific services. These services exploit the Apache Kafka messaging technology. Messages from source systems can be pushed into fraXses, fraXses can then process the messages and then push them onwards to an application that listens to incoming messages. Such an application can also use another messaging technology or an enterprise service bus. How the data is processed is defined with views.

Fast Software Development – To speed up software development, fraXses uses several open source software modules. The product exploits them fully and when improved versions are released, fraXses automatically exploits the new capabilities. Additionally, as indicated, the microservices architecture is flexible enough to allow external parties and customers to develop modules that are embedded in fraXses and operate as first-class citizens.

Transaction Processing – fraXses supports transaction processing. Updates, inserts, and deletes can be executed on the views resulting in changes made to the underlying source systems. Developers are completely shielded from the interfaces and SQL languages supported by the source systems. Source systems do not have to be SQL-databases, but can be NoSQL systems as well. Heterogeneous transaction processing is not supported yet.

Development and Deployment – Features not described in this whitepaper pertain to development by developers and product deployment.

The development of the data virtualization environment has been simplified in several ways in the new version of fraXses. First of all, the development environment in which views are developed can be classified as a *low-code environment* which improves productivity and maintenance. Most specifications can be defined without writing code.

The smart wrappers have simplified the development of services. They allow developers to concentrate on the core functionality of the services and not on more technical aspects. The smart wrappers, in which services operate, together with the orchestrator take care of all aspects related to operating within the larger architecture.

Deploying the environment itself has become easier because of the use of popular technologies such as Kubernetes, Docker, and Harbor, with which many organizations have experience. Kubernetes provides management, up and down scaling, upgrades, and rollback features. Docker containers allow images to be built, along with packaged code, that can be run in any environment that has Docker installed. This makes porting increasingly simple, but also for running multiple fraXses servers for different purposes, such as OTAP. In all these environments, the code running in the containers has a consistent execution environment in which to run. Harbor is a container image registry providing role-based access control, image vulnerability scanning and trusted image signing to all fraXses images. The three together, make deployment easy. In other words, the product does not introduce a new deployment style to those within the organization who are responsible for deployment.

10 Closing Remarks

The bar has clearly been raised for data virtualization, because organizations keep demanding more. The functionality, performance, and scalability must be improved. With a revamp of their internal architecture, fraXses did raise the bar for itself. Its microservices architecture fits perfectly with the technology of cloud platforms. fraXses does not run ‘on’ the cloud, it runs ‘within’ the cloud. It can fully utilize cloud platforms, resulting in almost limitless scalability. The use of popular technologies on cloud platforms, such as Kafka, Docker, and Kubernetes, makes it even more suitable for the cloud. Additionally, the microservices architecture and the smart wrappers allow customers and other parties to easily extend the existing functionality.

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[4].

He has presented countless seminars, webinars, and keynotes at industry-leading conferences. For many years, he has served as the chairman of the annual *European Enterprise Data and Business Intelligence Conference* in London and the annual *Data Warehousing and Business Intelligence Summit*.

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[5] all published at B-eye-Network.com.

He is the author of several books on computing, including his new *Data Virtualization: Selected Writings*[6] and *Data Virtualization for Business Intelligence Systems*[7]. Some of these books are available in different languages. Books such as the popular *Introduction to SQL* is available in English, Dutch, Italian, Chinese, and German and is 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[8], 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.

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

About Intenda

Intenda, founded in 2001, is a software company providing the unique fraXses platform to the global market. Intenda clients are ensured that they will be aligned with the latest technology trends in the market.
Their purpose is to provide business applications and technology that makes a difference to the world, using innovative thinking and tailor-made solutions that allow their clients to drive their businesses forward.
They see data inside out, and they are all about data. Data drives business, that is what Intenda specializes in.
With offices located in Africa, Europe, the United Kingdom and the USA, they have the capacity to service businesses on a broad scale across industry sectors and locations. For more information, visit www.intenda.tech or email info@intenda.tech.

About fraXses

fraXses was created to address the fast-moving issues business are facing with the ever-increasing data volumes, variety, the complexity of data sources and advances in technology that have required a different approach to data and transactional systems.
Using a discover, configure, and deliver methodology, the fraXses platform enhances data access and reduces the need for development with a low/no-code approach. This framework empowers the business to gain much greater value from their data.
fraXses provides an end-to-end solution for data virtualization and federation across multiple sources, technologies and locations as well as providing a data lake, IoT and data pipelining. The platform is built on a microservices architecture, which allows endless scalability options.