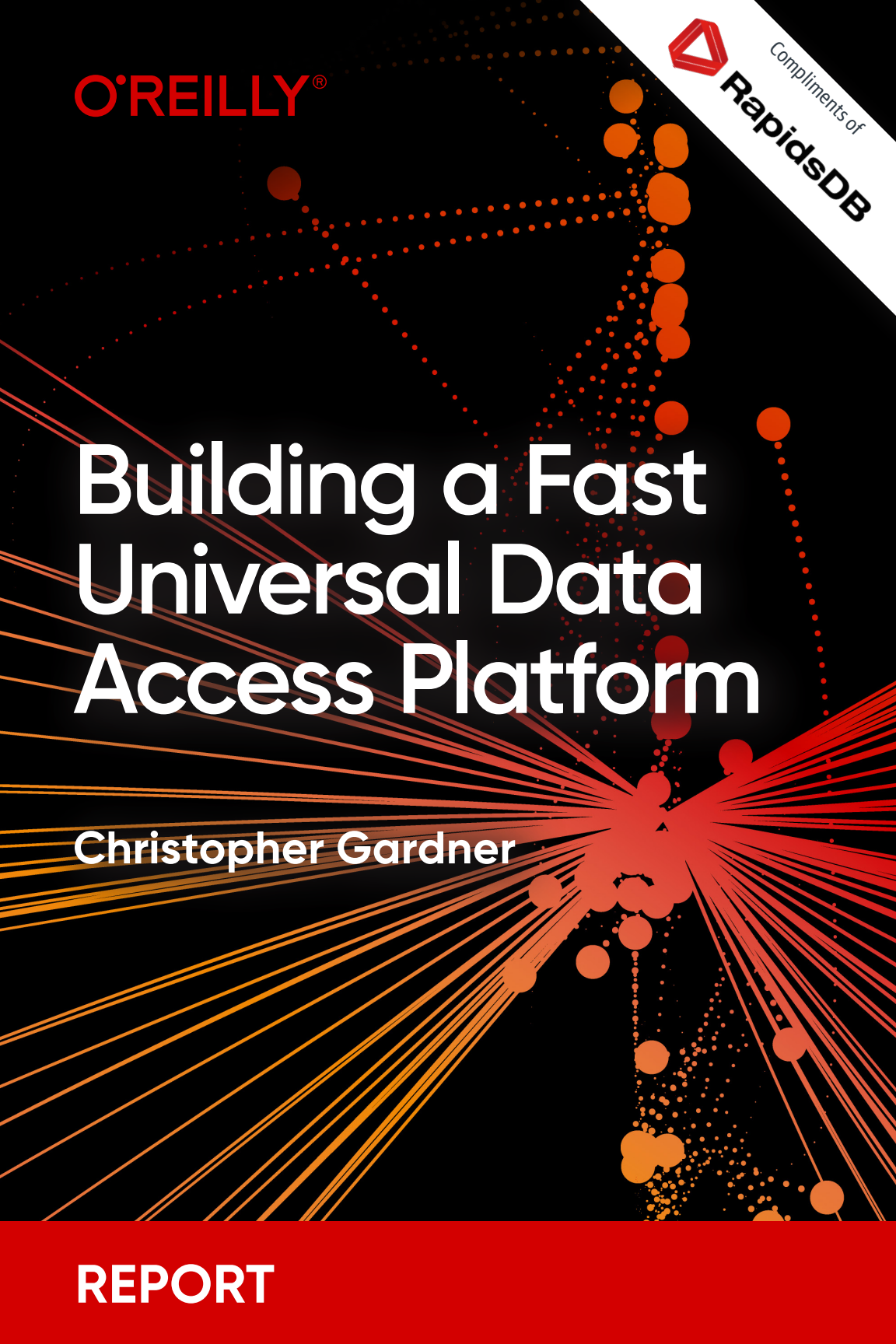
O'REILLY®
Compliments of
RapidsDB
Building a Fast Universal Data Access Platform
Christopher Gardner
REPORT

RapidsDB

Real-Time Analytics Multi-Cloud Lakehouse

Built for the future

RapidsDB | Rapids ParallelAI | Rapids StreamDB | Rapids Lakehouse | Rapids VectorDB

Distributed

Fully distributed computing framework supports cost-efficient horizonal scalability

Real-Time

Dynamic query optimization empowers millisecond-level real-time data processing and computing

In-Memory

In-memory computing provides high-speed data access and processing

AI

Seamlessly embedded AI-in-database library enhances machine learning/deep learning model development and training

Parallel

MPP architecture optimizes subqueries in parallel and handles high throughput and concurrency

SQL

Unified big data analytics platform uses a vendor-neutral dialect of standard ANSI SQL for data integration and interactive queries

Federated

Federated connector system integrates different types of data across a wide variety of data sources without the need for ETL

Secure

Fine grained control over the accessibility of the unified data and powerful auditing controls simplify system administration

Intelligent Data, Enabling Future!

To learn more about the Rapids Lakehouse, please visit www.rapidsdb.com

Patent: 11.347.736 Patent Pending: 17/070,662, 62/928,108, 17/179,149, 63/049,920, 17/179,149

Building a Fast Universal Data Access Platform

Christopher Gardner

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

Building a Fast Universal Data Access Platform
by Christopher Gardner

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

Acquisition Editor: Aaron Black
Development Editor: Gary O'Brien
Production Editor: Beth Kelly
Copyeditor: Audrey Doyle
Proofreader: Brandon Hashemi
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Kate Dullea

August 2023: First Edition

Revision History for the First Edition
2023-08-23: First Release

This work is part of a collaboration between O'Reilly and RapidsDB. See our statement of editorial independence.

978-1-098-15532-2

[LSI]

Table of Contents

CHAPTER 1

Challenges of Universal Data Access

Your company relies on data to succeed. Traditionally, this data came from the business's transactional processes. It was pulled from the transaction systems through an extract, transform, load (ETL) process and into a warehouse for reporting purposes. With the growth of the Internet of Things (IoT), web commerce, and cybersecurity, this traditional data flow no longer suffices. The copious and diverse data available to your company brings challenges with connections, speed, volume, and access. How do you ensure that your company can keep up with today's increasing magnitude of data and insights so that it will be a leader in the field in the future?

What Is Universal Data Access?

The main problems facing businesses today are the volume and variety of data accessible for analysis. It is no longer viable to simply examine the data generated by business processes. Instead, many organizations are starting to look outside their business workflow for information on customer behavior, retail patterns, and industry trends. This supplemental data provides actionable insights, but it also creates a challenge when integrating with business-specific data sources.

Your organization likely generates data and stores it within a single platform. Your daily processes create data as part of normal business flow, and this data is likely stored in a transactional database similar

to that shown in Figure 1-1. From this, the data is translated and transformed into a separate structure, such as a warehouse, where it can be reported more easily. But what happens when data is needed from an external source, such as web analytics, census demographics, or elsewhere? How do you integrate data sources of differing structure, design, or format? How do you ensure peak query performance to provide insights in near-real time?

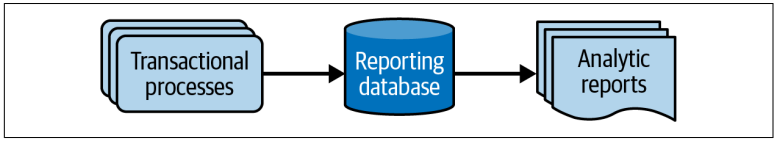

Figure 1-1. Traditional data flow from transactional processes to analytic insights

External data sources are vital to the success of your organization. The challenge lies in how to accurately and efficiently bring all this data together into a usable format. In Figure 1-2, you can see how these additional data sources impact data flow. As many of these sources reside outside your business's transactional process, many are in different formats and structures. Some of them may be in relational databases while others may be in NoSQL databases, flat files or streams. You need special reporting tools to combine this data into a streamlined, usable source.

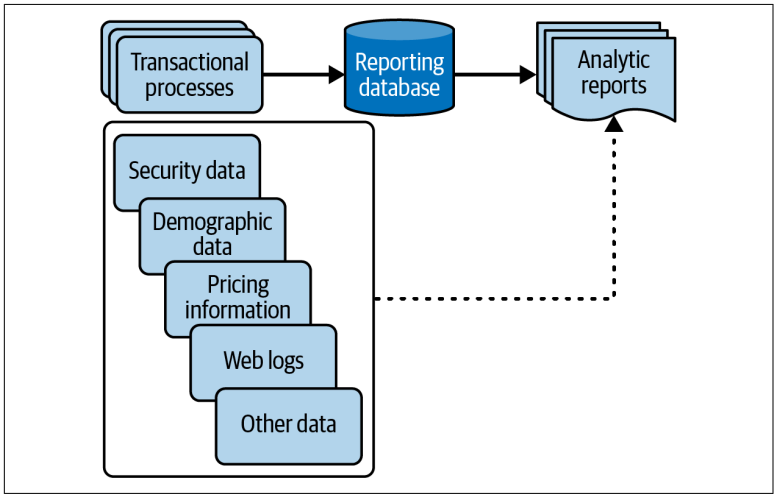

Figure 1-2. Data flow illustrating the gap caused by extracting data from external sources

The process of combining all these different data sources, formats, and structures is what we refer to in this report as *universal data access*. In short, a framework is developed to pull data from multiple sources and combine the data sets into a single schema of queryable views similar to that shown in Figure 1-3. Ideally, this new framework would be dynamic, be easy to access, and rely on the infrastructure of its sources to provide the speed and power needed to make the data accessible in a timely fashion. In truth, however, many of today's applications sacrifice speed for data diversity or vice versa.

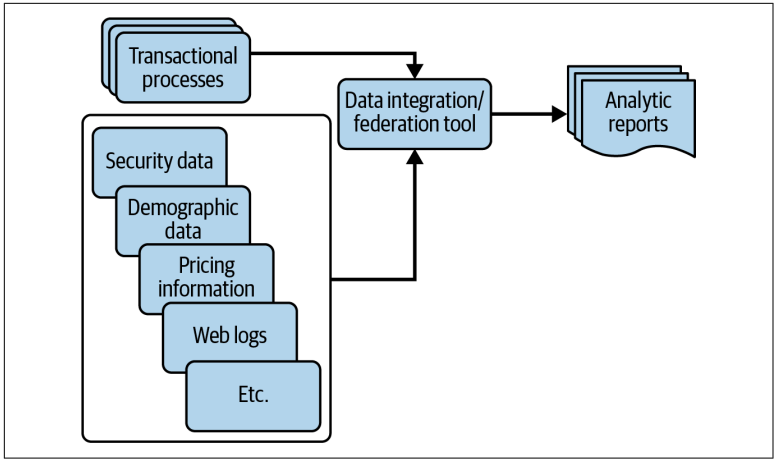

Figure 1-3. Data flow from transactional processes and external sources to an integration or federation tool, consolidating both into a single format

Two of the most well-known software tools to move forward with universal data access are Denodo and Presto. While each is powerful in its own right, they both have drawbacks. Denodo is extremely diverse, providing a huge variety of connectors and allowing users to pull from many different data sources; however, it lacks the performance needed to combine these sources and provide ad hoc queries. Presto is more adept at handling ad hoc queries, but it lacks the diversity of connection types. Let's take a deeper look at the areas of data diversity and data volume, as they dictate the success of combining external and internal data sources.

Data Diversity

Data within an organization is easier to collect and combine into a single format and data structure, but what happens when data is needed from sources the organization does not control? How do you deal with data that is dynamic, unstructured, or even residing in an entirely different database structure than your organization?

To answer these questions, first we'll examine why external data is relevant to your business. Take, for example, a retailer. A business such as this would need data related to operations such as supply costs, inventory, product demand, and operating costs (employee pay, utilities, etc.). What happens if we look one step beyond the business itself? A retailer can improve sales by utilizing web data to understand customer habits and trends. It can improve marketing by identifying customer demographics and regional tendencies. It could improve transportation costs by comparing delivery services. In fact, huge amounts of data are available to any business beyond its operating core that can improve the bottom line.

This leads to the first issue with data diversity. Traditional data is organized and structured, usually in a centralized database with fixed tables, fields, and data formats. This makes the data easy to store, manipulate, and analyze. Data doesn't just come in strings, integers, or floating decimal values, though. Photos, audio, video, arrays, and many other new types of data are now becoming mainstream, and traditional data structures are not designed to handle these new formats. Your business needs definitions and functionality to assist with storing and analyzing these data types.

This issue is not limited to the format of the sources, either. Many other obstacles come into play. First, the data needs to be extracted from its source and pulled into a repository where it can be combined with other data. This process requires transferring the data over networks, translating data fields, and establishing an architecture that will provide accurate and usable data. All of this takes time, which builds in a level of latency from when the data becomes available in its source system to when it is accessible for reporting within your organization. As we'll see later, data speed is imperative in making timely business decisions.

The second issue with data diversity is data quality and integrity. Adding new data to an already existing system creates potential

challenges. For example, pulling data can sometimes result in duplicate data or incorrect field formats. There are also challenges with changes in the source system impacting the data pull or transformation. This obstacle requires constant monitoring and testing to ensure data quality throughout.

The third issue is security. Data integration makes copies of source systems in another location. Pulling data from external sources may result in additional security needs, especially if that data contains sensitive or private information. Some examples of data that may have potential security issues include protected health data, credit card data, and Social Security numbers. You need to be prepared to handle increased security needs if you use data integration to pull in external data sources with sensitive information.

The final issue with data diversity is adaptability. Data sources will continue to adapt and change. Your company's needs will change as well. Unfortunately, many systems designed to pull in disparate data are not easily adaptable. What happens when your data federation or data integration system does not adapt to a new data source that is imperative to the operation of your business?

All these obstacles combine to create challenges when bringing together varieties of data for reporting. A platform is needed to bring this wide variety of data into a single usable format. Two processes that combine disparate data sources are *data federation* and *data integration*. These processes are accomplished with software that allows multiple varieties and structures of data to be combined and function under a single source. For now, it is enough to understand what the software does. Later in this guide, we will discuss the differences between data integration and data federation as well as evaluate the advantages and disadvantages of each.

Data diversity is not your company's only obstacle as it searches for new and valuable insights. Organizations also face the challenges of data volume.

Data Volume

Consider your existing data structure within your company. How many different tables exist? How many years' worth of data is available? What happens when more and more data is added as your company continues to do business year after year? Are you able

to dynamically add new data types or data sources? As more time passes, additional data is collected, meaning the data stores are dramatically increasing in size. How does your company handle this? What happens when more data is required to do effective predictive and prescriptive analysis for your company?

This brings us to the second-biggest hurdle when it comes to data volume: data processing. Even if you have a large-volume cloud storage option for your data, it still takes time to organize, structure, load, and analyze that stored data. Once the data is loaded, it may still need to be evaluated for quality and integrity. The large volumes of data many companies deal with create obstacles to finding errors or inaccuracies.

There are several vendors in the data storage space and many new methods are emerging to improve data organization, indexing, and performance. Hadoop and Apache Spark are two examples of databases designed for large-volume storage and access. Even these storage structures have issues, though, as they must be able to handle existing data while being extendable to handle future demand. Still, the flexibility and adaptability of cloud storage has led many companies to turn away from storing data in large server banks locally.

Regardless of what tools you use or where you store your data, the biggest hurdle when it comes to data volume is speed. Decisions are made much more effectively when they happen closer to when transactions occur. Parsing, loading, and analyzing large volumes of data take time, which creates the potential for latency. Things like network speed, server resources, and usage volume can all impact the performance of queries against data sources. Knowing this, let's look at why speed is valuable to your business and what obstacles exist with generating near-real-time insights.

Speed of Analytic Operations

Why are rapid insights so valuable to your organization? With increased online commerce, speed is more important than ever. Faster analytics can improve your business in the following ways:

Improved customer experience
: By examining browsing habits and historical purchases, companies can provide their customers with customized experiences in near-real time. These improved experiences improve

customer interactions and increase the likelihood of successful transactions.

Utilizing customer demographics
: Information about customers is key to improving sales. Demographic and regional data provides the tools necessary to generate marketing efforts specific to gender, race, or age groups as well as customers from specific regions within your service area.

Maximizing operational efficiency
: Real-time information on how your company is performing is vital to success. This includes data from transactional sources that allow you to identify where business processes are lagging. Such data may also come from external sources such as suppliers or shipping providers. Access to this data not only allows you to identify what is inefficient, but also helps you preempt potential challenges in the future.

Identifying and preventing data threats
: Network and security threats are becoming more and more frequent in the business world. Real-time threat assessment and mitigation is imperative to ensuring that your company and customer information is secure, safe, and private.

Improving competitive advantage
: Tracking pricing trends can benefit your business by providing near-real-time information on competitor pricing, customer feedback, and sales analysis. This data can then be used to adjust prices dynamically, ensuring that your business remains competitive in the market.

In each of these examples, latency is an obstacle to valuable insights. Your company needs to adapt as quickly as possible to the business landscape to attract and keep customers, produce products or services more efficiently, and identify network and security threats as quickly as possible. So how do you overcome the latency that can be inherent in large amounts of data from diverse sources?

There are multiple approaches to improving data speed. The first is to throw resources at the problem. Many companies adopt this method to ensure that critical processes continue to run as more and more resources are needed. In short, if the memory, processor, or storage is insufficient to meet the needs of the data, simply add more. While this works in many cases, there are limitations to how

many of these things can be added to manage the data. Additionally, this method significantly increases the cost of maintaining the data. Additional hardware requires additional processing power, storage space, and human resources to maintain it all.

Speed can also be improved by converting the data into a unified format. As mentioned before, data integration and data federation will combine multiple varieties of data sources to provide a single unified data source. This allows data analysts, data scientists, and report writers to analyze a single source rather than face the challenges associated with disparate data sources. Ultimately, a unified data format improves the speed to gaining insights, providing time for analysis rather than spending time figuring out a blend.

There are other ways to improve speed as well. Indexing is a common solution that improves speed by creating a directory of where certain information is housed. This improves query times by pre-identifying where to look for the requested information. To be effective, indexing needs to happen regularly, as data and sources change.

Certain database management systems (DBMSs) are also effective in improving query performance against data sources. Columnar database management systems such as Amazon Redshift and Google Cloud convert row-based data into a columnar format, allowing for faster indexing and data retrieval as data can be searched for within a specific column rather than searching through every row. Unfortunately, columnar databases are far more difficult to load than row-based databases due to their structure.

There are numerous reasons to improve speed, but the data sources that provide the necessary data to achieve these results are large in size, are varied in format, and require complex queries to generate valuable insights. Which methods are most effective for adapting to large and diverse data sets? How does your company overcome these obstacles to ensure that the data needed for insights is not only accurate and accessible but also timely?

CHAPTER 2

Building a Framework for Data Diversity and Universal Access

We've established three primary goals for our data ecosystem. First, we need it to handle data of varying types and sources while also being able to rapidly adapt as new sources become available. Second, we need to be able to handle large quantities of data. Our solution needs to be large enough to handle all our existing data and flexible enough to expand as more data gets added. Finally, we need our solution to be able to return queries as quickly as possible to give our business the ability to get rapid insights. Knowing these challenges, what are the solutions that will help us achieve our goals?

Federated Query System

The underlying goal of pulling in larger volumes and varieties of data is to create insights from multiple combinations of sources, expanding your knowledge of how your company runs and identifying ways to improve. The challenge is that many of these data sources require different connection types, languages, or even translations to make them usable. A federated query system does that work behind the scenes and provides a single point of connection for users attempting to utilize the data.

Federated query systems resolve the challenges of diverse data. Using an overlying system to connect to diverse data sources solves many data storage and reporting challenges. The most obvious problem this solves is the need for multiple drivers, multiple logins,

and complex blends associated with connecting to different types of data. It also eliminates some of the need for data knowledge, as end users no longer have to know which data source has which data.

It is important to point out that data federation and data integration are not the same thing. Data integration pulls data from multiple sources and replicates it in a single, unified format for querying. Data federation does not duplicate the data. Rather, it provides a data model that connects to the disparate data sources, allowing you to query them directly through a single source, similar to the structure shown in Figure 2-1. It also means end users only need to be familiar with one type of query language.

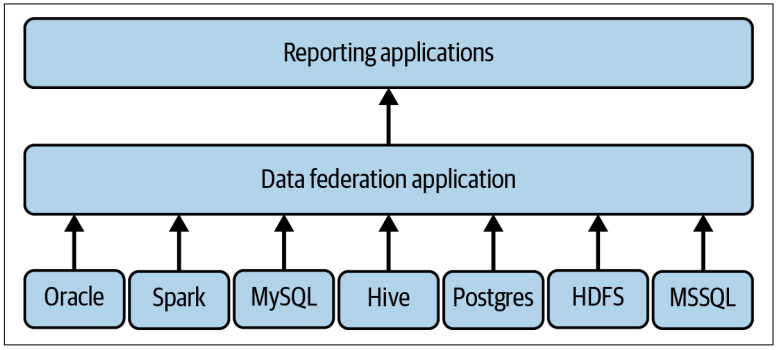

Figure 2-1. Example layering of data federation tools to provide reporting applications access to data in multiple formats

A federated system can also address the issue of security. Unlike data integration systems, data federation does not require data replication. This means security can be passed from the source systems to the end user rather than relying on a separate set of security for the replicated version.

Pluggable Data Connectors

The framework being developed across multiple data sources also needs to be flexible enough to expand and adapt to changes in business requirements. Take, for example, a business that adopts a web-based retail interface. In addition to traditional sales data, the business needs to examine the success of the online storefront. This may include data such as page clicks, time on pages, and resulting sales. More than likely, this data will not be in the same format as that from a traditional brick-and-mortar store. It may be a different

database, in a different format, and of different granularity than what has been tracked and recorded previously.

This company needs a system that will adapt to the online change, and universal access allows this to happen. Instead of connecting separately to online data and trying to determine how to blend it with existing data, the online data is added to the federated data system with the addition of a simple connector. The users don't need to learn how to query web data; instead, they continue to use the federated query system to pull both online and local sales data.

Data connectors eliminate the need for additional logins, query languages, security, and more. Plugging in data connectors allows your business to adapt and change as data requirements shift. This means adaptability and scalability. It also means your system can handle new data types through user-defined types and functions, which we will discuss later.

Additionally, these connectors can independently adapt to the source system. If our fictional retail business is pulling web data and that web source suddenly changes, only the connector needs to be adjusted. The system can also adapt by adding additional connectors.

Support for Cloud, Hybrid Cloud, and On-Premises Deployments

Since the framework is able to connect to multiple sources and file types, there are no restrictions for where it is hosted. It could be on premises, on the cloud, or both. As long as the system has the means to access the data, it can be hosted in a variety of environments. This adaptability also addresses the second portion of our requirements: the capability to handle large and growing data sets.

A data federation system can connect to multiple data sources, file types, and structures, but once it does, it needs a place to store the resulting data. Data federation does this virtually.

User-Defined Types and Functions

It's very likely that your company utilizes common types of data. In most traditional systems, these would be data types such as integers, floating decimal values, strings, booleans, and dates. A federated

system should be able to handle these types of data easily, but what happens when your data systems become more complex due to adding new data sources or changing your functional business? What happens when your company needs new data types such as URLs, geographic coordinates, or images? A federated system needs to be able to quickly adapt to new structures as well as the requirements and rules that govern how they can be used.

The following example illustrates this idea perfectly. Think of an integer data type. It has specific requirements such as being numeric and being a whole number (no decimal places). It can also be interacted with in specific ways such as with `MIN`, `MAX`, `SUM`, and `AVG` functions. The definition and the corresponding functions that apply to this data type are standard, but what happens when new data types arrive?

For a system to be truly dynamic, users need a way to define new data types along with the functions that manipulate and interact with them. A system that allows user-defined types and functions can be extensible, flexible, and consistent. It allows for data types that are otherwise not available within a traditional database, such as images, movies, and more. These defined types and functions can then be stored and reused via a connector with other data areas and queries.

CHAPTER 3

Meeting the Performance SLAs for Making Business-Critical Decisions

Pulling data from multiple sources is necessary to make effective decisions about your business; however, these data sources are not as useful if they cannot be queried and cannot return data quickly—data delays result in missed opportunities and failure to identify issues. So companies look for ways to increase the variety and volume of data being used, as well as ways to improve the performance of that data and return insights in speeds that are closer to real time. Let's look at some of the common methods for improving database performance.

Dynamic Clustering

The first method for improving database performance is *dynamic clustering*. In short, a central server acts as a delegator, shifting the system's demands across a bank of worker servers. This distributes the query requests and data returns across multiple server nodes. Each server in the cluster provides data on system performance. This allows the workload manager to determine where to assign requests dynamically as resources become available.

By distributing the query across multiple nodes, the system distributes the workload across multiple servers, thus increasing the processing power and memory devoted to query returns. Because the clustering is dynamic, the delegating server can shift the work

among multiple nodes, ensuring that the queries processed are run as efficiently as possible.

Dynamic clustering provides an additional benefit: redundancy. If any cluster nodes go down, the remaining clusters are available to pick up the load. This provides replication and prevents the downtime that would occur during a normal server outage, ensuring that your company has reportable data quickly and reliably.

Clustering also allows your company to scale easily. If your business's data demands change at any time, additional node clusters can be rolled up to meet the increased need. Combining this with a federated query system ensures that any new data can easily be added, analyzed, and available for querying and reporting. It also ensures that reporting happens in as near to real time as possible, meaning that your data-driven insights are actionable and useful.

Dynamic Query Optimization

There are two different approaches to query optimization. The first approach is *static query optimization*, whereby the SQL is submitted to the system and an execution plan is created to generate the results. With larger and more complex queries, static optimization often lacks the necessary information about the tables being queried. Additionally, the statistics about those tables might be inaccurate or lacking. The result is a query execution plan that is slow or long running, delaying the valuable information needed from the data. This issue is exacerbated when organizations look to add nonrelational database connections to their analyses.

The second approach is referred to as *dynamic query optimization*. The first step in this process is identical to that in static query optimization. Once the initial execution plan is created, though, the dynamic query optimization process continues during the course of execution. The result is a query execution plan that is continuously and dynamically updated as additional information about the queried tables is discovered. In short, the dynamic query is able to adjust the assumptions made by the initial execution plan to pull data more efficiently and quickly.

To better understand this, let's look at an example. Assume a large query is submitted to a system with dynamic query optimization. Initially, the system has very little knowledge of the size or

cardinality of the data being queried. The optimizer does its best to guess what will be the fastest way to process the query and return results. As the query continues, however, information about the data becomes available with regard to size and structure. The optimizer can reevaluate performance based on these new pieces of information and improve the performance by shifting parts of the query to other resources or recompiling during the process. Figure 3-1 depicts both approaches to query optimization.

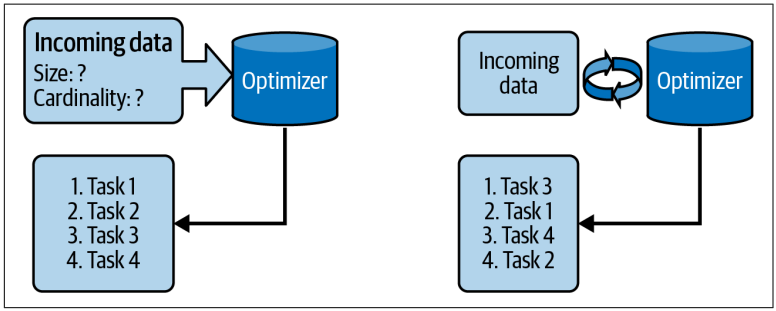

Figure 3-1. Static query optimization (left) and dynamic query optimization (right)

Dynamic queries continuously work to improve performance. The results that are gathered, such as statistical information about tables, cardinality, and size, can then be applied to future queries utilizing these same table resources. The system also analyzes statistics on query performance to see if the process meets thresholds or needs to be reconfigured. This ensures that the queries are routinely updated and verified to be the most efficient as the data changes and updates. The system can run additional queries to improve performance during downtimes. Let's dig further into query optimization to understand why it's so important.

Humans write queries from the perspective of requesting data from specific places, joining it to other data, restricting it, and returning the results. Logic from humans only provides the "what" of the query, and it's up to the optimizer to determine the "how." Let's imagine a typical query with multiple joins and restrictions and use it to understand how query optimization works and how data being pulled can be run in multiple ways.

First, let's set up our query. We will pull data from Table A and join it with data from Tables B and C. We will restrict our results to only

include values of *y* from Table A not equal to 5 and values of *z* from Table C equal to 8, as shown in Figure 3-2. The query, as written in SQL, would be something like this:

```
select sum(B.x)
from A, B, C
where A.k = B.k and B.k = C.k and A.y >< 5 and C.z = 8
group by B.g;
```

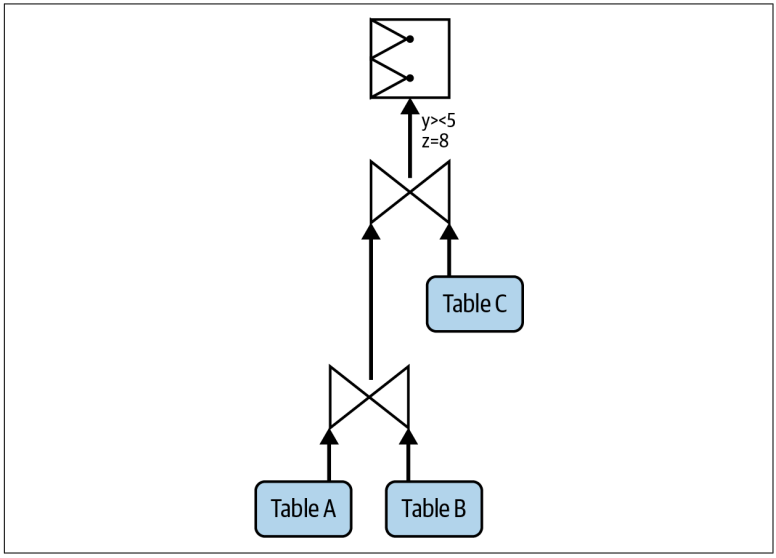

Figure 3-2. An example query consisting of data from Table A joined to Tables B and C, with the values of y and z being restricted

In this query, there are multiple paths the optimizer can take to return the results. Does it pull the data from A and then from B? Does it pull it from A and then C? When do we restrict the results for *y* and *z*? Assuming limited information about the tables was available at execution, a static query optimization might perform the join between Tables A and B, then join the results to Table C before limiting the results based on *y* and *z*. This is not the most optimal approach. Let's examine how a dynamic query optimization process might improve performance.

First, the dynamic query optimization process would examine the query and build an execution plan. As the query starts running, the optimizer will likely determine that less work needs to happen if the rows being pulled are limited to begin with. Thus, it would restrict the results from Table A by *z* and Table C by *y* before

doing any joins. This is called *moving the predicate down in the plan*. Finally, as the query is running, the optimizer might realize that Table A is very large and Table C is very small. It may then determine that joining Tables A and C first before joining Table B would improve performance. Information about tables such as size and cardinality of the data from this query would then be recorded for any future queries including Tables A and C. By acquiring information about the tables, moving the predicates down, and rearranging the joins as shown in Figure 3-3, dynamic query optimization improves the runtime performance of the query.

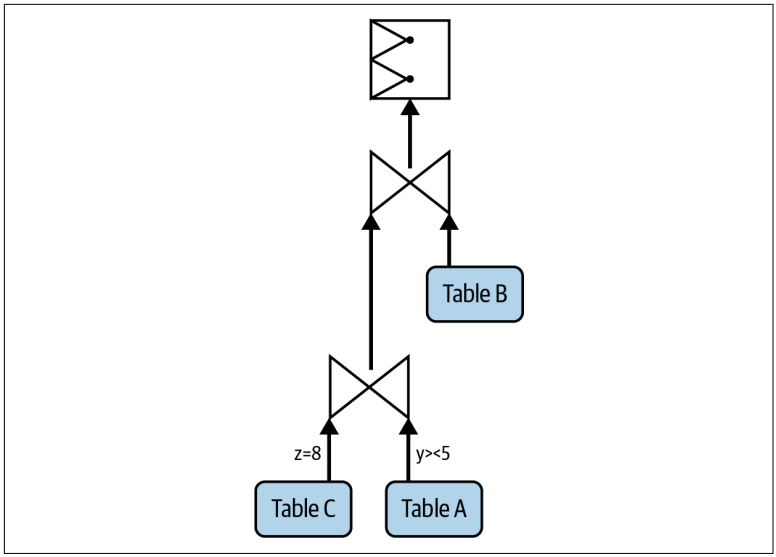

Figure 3-3. Query optimization rearranging the order of predicates and joins to improve performance

Minimizing Data Movement Using Intelligent Query Pushdown

A federated query system provides the ability to query multiple different data sources as though they were one single source, but doing so can be CPU, memory, and network intensive on the system doing the federation. When large amounts of data need to be combined across a variety of systems, additional steps are required to mitigate the burden on the federated system where the query originated. This is where intelligent query pushdown comes into play.

In a typical federated system, the federation server would retrieve data from the various data sources and proceed to execute the query. In this process, the resources of the federation server would be used to collect and combine the multiple data sources as illustrated in Figure 3-4. For example, a join between tables from separate databases would first be pulled to the federation server, which would then process the join.

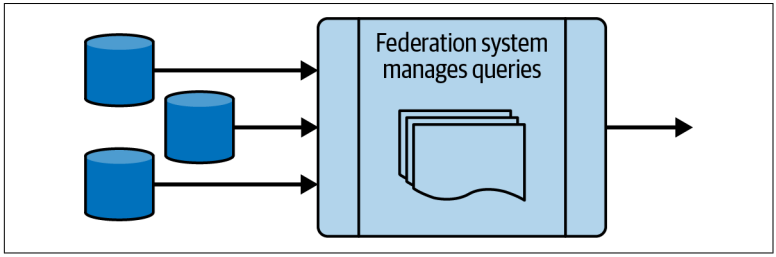

Figure 3-4. In a typical federated system, the system itself executes each query and returns the data from the source data systems

This idea of query pushdown is to push as much of the query process as possible to the underlying data source servers. This not only shifts some of the processor and memory demands to another server, it also limits the amount of data being passed over the network. We mentioned an example of pushdown when we discussed dynamic query optimization. Some federated systems will push the query along with predicates down to the source server. Let's return to our previous example:

```
select sum(B.x)
from A, B, C
where A.k = B.k and B.k = C.k and A.y >< 5 and C.z = 8
group by B.g;
```

If we assume that Table C resides on an Oracle database, the federation server can push that part of the overall query to the server on which Table C resides:

```
select k
from C
where z=8;
```

This pushes the query processing to the Oracle server hosting Table C. Once the Oracle server completes the query, it can then run the predicate, which limits the amount of data returned across the network to the federated server. This increases query performance

by distributing the workload and reducing the amount of data transferred over the network, as illustrated in Figure 3-5.

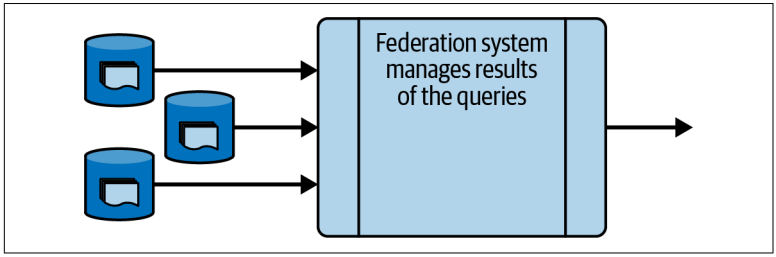

Figure 3-5. Intelligent query pushdown shifts all or some of the query process to the server hosting the data

Some solutions, such as RapidsDB, take this a step further by utilizing their connector-based approach to federation, where each connector understands the capabilities of the data source for which the connector is responsible. This is known as *intelligent query pushdown*. With intelligent query pushdown, the federation server will push down not only simple queries and predicates, but also complex subqueries to be completed on the remote data source server.

Returning to the previous example, if Tables A and C resided on an Oracle database, then the query:

```
select A.k
from C,A
where C.k = A.k and C.z=8 and A.y >< 5;
```

could be pushed down to the Oracle database server.

To support intelligent query pushdown, the system has to be able to break down a query into a set of subqueries, each contained completely within a single data source, and then decide which parts of that subquery can be pushed down. Submitted queries that have calls to functions only supported by that federation server would be run on that federation server. Federation servers only push down those parts of the subquery that could be executed by the data source server. For example, assume the following query:

```
select fastest(x)
from A
where A.y > 8;
```

If the function `fastest` was not supported by the remote data source, it would need to be handled by pushing down the query:

```
select x
from A
where y > 8;
```

The federated system would then apply the function `fastest` to the data returned from the data source. This type of pushdown can become very complex.

Execution at Machine-Code Speed

Queries are written with data in mind from the perspective of a human being, but that perspective is not the fastest or most efficient for machines relying on 1s and 0s. To make queries move even faster, the logic needs to be shifted from a human language to a language that is faster for a computer to use and digest. That language is machine code. If a query can be converted from the language it was written in (such as SQL) to machine language, no level of code interpretation needs to happen. The process of converting from SQL to machine code is called *compilation* and is performed by compiler software.

The benefits of this approach are twofold. First, executable programs are able to fully utilize the processor and memory resources within a server and are not restricted by the program hosting the query. Second, the operating system has built-in capabilities to improve executable performance, meaning the optimization can continue to the level of the server. An example of such a compiler is LLVM, which was started as a research project at the University of Illinois.

Some companies take the optimization process a step further. For example, RapidsDB relies on Java bytecode, which is then executed by the Java Virtual Machine (JVM). The JVM has a just-in-time compiler that analyzes the code and determines how to compile the Java bytecode to provide the most efficient method of execution. Each time it executes a chunk of code, it triggers a countdown. When it reaches a predefined optimization level, the bytecode is compiled into machine code. This means the more often a piece of bytecode is run, the more likely it is to get compiled into machine code. For RapidsDB, the engineers have determined the best way to write Java bytecode such that the just-in-time compiler will compile the bytecode into machine code to get the optimum performance.

In-Memory Data Processing

Traditionally, queries rely on disk accesses to pull data and analyze the results; however, hosting the data in memory means the processing skips the disk read process and improves performance dramatically. In-memory systems store the data in RAM across a cluster of servers, allowing them to process the data in parallel. By storing the data in RAM and utilizing multiple servers, data can be processed many times faster than traditional disk-storage databases.

Redis is an example of an in-memory data structure store. But while Redis offers the performance improvement afforded by in-memory data processing, that improved performance comes at the cost of functionality. Redis only allows very simple queries and lacks the ability to do joins or other, more complex data queries. It may improve speed, but if it cannot meet business query needs, its usefulness is limited.

These are several examples of how performance needs can be addressed when working with the wider varieties and larger volumes of data becoming available each day. What does our platform need to accomplish to meet the needs of our changing data environment?

CHAPTER 4
Requirements Summary

A fast universal data access platform must meet the following organizational requirements:

- Handle a wide variety and ever-changing set of data.
- Deal with huge amounts of data that continue to grow with time.
- Provide results to analytical queries as quickly as possible to allow organizations to do analysis and make data-driven decisions in as close to real time as possible.
- Be flexible enough to be on premises, in the cloud, or any combination of the two.

To meet these requirements, the platform needs to utilize multiple components:

Federated query system
: Allows the platform to connect to a wide variety of different data types and structures, and allows for structured and unstructured data.

Pluggable data connectors
: Give the flexibility to add additional connection types as data sources change and update.

Multiple installation options including varying levels of cloud support
: Ensure that the organization's needs are met regardless of whether the installation is on premises or in the cloud.

Dynamic clustering
: Brings multiple worker nodes under a delegating authority to provide additional power, speed, and redundancy.

Dynamic query optimization
: Continuously improves the performance of queries to ensure that data is returned in as close to real time as possible.

Intelligent query pushdown
: Shifts the data processing to the data sources, meaning the work is distributed to the system most adept at processing it. It also potentially reduces the amount of data returned to the federated system by applying query predicates at the source prior to returning the data.

Ability to convert query code to machine code
: Further improves query performance by translating code written by humans into a machine-optimized language.

Case Study

How do all these components work together to form a multiaccess, high-volume, high-performance interface? Let's take a look at an example. China Mobile is a telemarketing business in Asia, providing service for over 1.5 billion mobile service users in 2019. Within the company is China Mobile Guangdong (GMCC), which services over 100 million customers in the Guangdong province of China. The large volume of customers combined with a wide variety of diverse data sources meant that GMCC was struggling to derive insights from terabyte levels of data.

GMCC needed this data to meet customer needs, assess network performance, and optimize company performance; however, pulling insights from the data proved extremely difficult since the data was so large and so siloed. Query performance was slow, and the lack of a universal query language meant business users had to rely on the IT office to generate new reports and data sets. The result? The IT office became overwhelmed by the high demand for data, and vital reports were delayed from days to months. In short, GMCC could not make real-time data-driven decisions due to the volume and complexity of its data environment.

The company approached the issue by addressing portions of the issue independently:

- First, it relied on Hadoop (HDFS) to store all the data. HDFS was able to adapt to the huge volume, holding over 8 TB of data; however, it relies heavily on disc input and output, resulting in increased latency and reduced query speed. HDFS is also batch processing based, meaning ad hoc queries could not be run easily.
- Redis was employed to improve speed by processing data in memory; however, as stated before, Redis limits capabilities to simple queries, reducing its effectiveness.
- Oracle and DB2 were employed, but both rely heavily on the ETL process, which is extremely time consuming and resource intensive. The two database structures are also limited to structured data, meaning they couldn't handle some of the unique data values being pulled in by GMCC.

GMCC turned to RapidsDB to address its data challenges. RapidsDB was able to deal with many of the issues by utilizing its in-memory federated query system. The system is built on a distributed model, meaning multiple machines work in unison to process data and queries. This model not only improved performance, but also provided redundancy while allowing for easy expansion. The system also pulled all the varied data sources under a single unified language, enabling users to access the data using a single query language and login rather than multiple languages and logins.

This implementation provided a significant improvement. Query processing times dropped from minutes to seconds. Additionally, the software relied on a standard SQL language and passed credential authorization to the sub-data sets. This meant the base application controlled access. In short, if the user successfully authenticated to the universal database access platform and had access to the underlying data set, they would be granted access without the need for a separate credential.

This improved performance means GMCC can make queries of its data and receive insights in much closer to real time. The federated query system ensures that existing platforms and future platforms can easily be added, combined, and queried. Also important is that users can do ad hoc queries rather than rely on time-consuming batch processing.

The benefits of a fast universal data access platform are multiple. With the increased number of data sources, data varieties, and data changes, the capability to provide connectivity, storage, and usability at fast speeds is more valuable than ever. A distributed framework combined with pluggable connectors ensures not only that current data obstacles are overcome, but also that your company is prepared and able to adapt to future data challenges.

About the Author

Christopher Gardner is the campus Tableau application administrator at the University of Michigan, controlling security, updates, and performance maintenance. He's also part of a small team responsible for developing and maintaining a suite of university-level dashboards used by campus budget administrators, deans, and university leadership. As part of his job, he leads a monthly daylong Tableau introduction course at the university. Christopher holds a degree in actuarial mathematics from the University of Michigan.