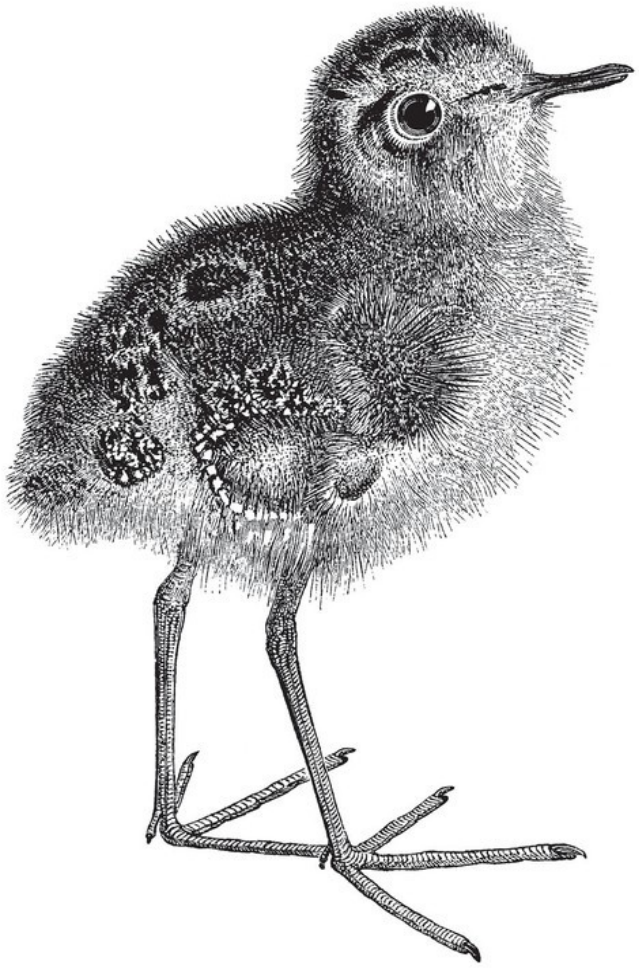
O'REILLY®

Deciphering Data Architectures

Choosing Between a Modern Data Warehouse, Data Fabric, Data Lakehouse, and Data Mesh



Early Release

RAW & UNEDITED

James Serra

Deciphering Data Architectures

Choosing Between a Modern Data Warehouse, Data Fabric, Data Lakehouse, and Data Mesh

---

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

James Serra

Beijing • Boston • Farnham • Sebastopol • Tokyo

Deciphering Data Architectures

by James Serra

Printed in the United States of America.

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

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (http://oreilly.com). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

- Editors: Sarah Grey and Aaron Black
- Production Editor: Katherine Tozer
- Interior Designer: David Futato
- Cover Designer: Karen Montgomery
- Illustrator: Kate Dullea

- September 2024: First Edition

Revision History for the Early Release

- 2023-05-01: First Release
- 2023-06-23: Second Release
- 2023-07-19: Third Release

See http://oreilly.com/catalog/errata.csp?isbn=9781098150761 for release details.

sibility to ensure that your use thereof complies with such licenses and/or rights.

978-1-098-15070-9

Preface

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the author at *jamesserra3@gmail.com*.

I've been in information technology (IT) for 35 years. I've worked at companies of all different sizes, I've worked as a consultant, and I've owned my own company. For the last nine years, I have been at Microsoft as a data architect, and for the last 15 years I have been involved with data warehousing. I've spoken about data thousands of times, to customers and groups.

During my career, I have seen many data architectures come and go. I've seen too many companies argue over the best approach and end up building the wrong data architecture: a mistake that can cost them

millions of dollars and months of time, and put them well behind their competitors.

What's more, data architectures are complex. I've seen firsthand that most people are unclear on the concepts involved, if they're aware of them at all. Everyone seems to be throwing around terms like *data mesh*, *data warehouse*, and *data lakehouse*—but if you ask 10 people what a data mesh is, you will get 11 different answers.

Where do you even start? Are these just buzzwords with a lot of hype but little substance? Or are they viable approaches? They may sound great in theory, but how practical are they? What are the pros and cons of each architecture?

None of the architectures discussed in this book is "wrong." They all have a place, but only in certain use cases. There is no one architecture that applies to every situation, so this book is not about convincing you to choose one architecture over all the others. You will get honest opinions on the pros and cons of each architecture. Everything has tradeoffs, and it's important to understand what those are and not just go with one architecture because it is hyped more than the others. And there is much to learn from each architecture, even if you don't use it. For example, understanding how a data mesh works will get you thinking about data ownership, which can apply to any architecture.

This book provides a basic grounding in common data architecture concepts. There are *so* many concepts out there, and figuring out what to choose and how to implement them is intimidating. I'm here to help you to understand all these concepts and architectures at a high level, so you get a sense of the options and can see which one is the most appropriate for your situation. The goal of the book is to allow you to talk intelligently about the data concepts and architectures, and dig deeper into any of them if they are relevant to the solution you are building.

There are no standard definitions of data concepts and architectures. If there were, this book would not be needed. My hope is to provide standard definitions that help everyone get onto the same page, to make discussions easier. I'm under no illusion that my definitions will be universally accepted, but I'd like to give us all a starting point for conversations about how to adjust those definitions.

I have written this book for anyone with an interest in getting value out of data, whether you're a database developer or administrator, a data architect, a CTO or CIO, or even in a role outside of IT. You could be early in your career or a seasoned veteran. The only skills you need are a little familiarity with data from your work and a sense of curiosity.

For “beginners,” I provide an overview of big data (chapter 1) and data architectures (chapter 2), as well as basic data concepts (Part II). If you’ve been in the data game for a while but need to understand new architectures, you might find a lot of value in Part III, which dives into the details of particular data architectures, as well as in reviewing some of the basics. For you, this will be a quick cover-to-cover read; feel free to skip over the sections that you already know well. Although the focus is on big data, the concepts and architectures apply even if you have “small” data.

This is a vendor-neutral book, except for the appendix about technologies and products. You should be able to apply the architectures and concepts you learn here with any cloud provider. I’ll also note here that I am employed by Microsoft. However, the opinions expressed here are mine alone and do not reflect the views of my employer.

I wrote this book because I have an innate curiosity to understand things and then share them in a way that everyone can understand. This book is the culmination of my life’s work. I hope you find it valuable.

Part I. Foundation

In Part I of this book, I will lay the foundation of deciphering data architectures. Chapter 1 starts with a description of big data, while Chapter 2 provides an overview of the types of data architectures and their evolution. Chapter 3 shows you how you can conduct an architecture design session to help determine the best data architecture to use for your project.

Part I will give you a good starting point to understand the value of big data and the history of the architectures that captured that data. The later chapters will then dig into the details.

Chapter 1. Big Data

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 1st chapter of the final book.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the author at *jamesserra3@gmail.com*.

The number of companies building data architectures has exploded in the 2020s. That growth is unlikely to slow down anytime soon, in large part because more data is available than ever before: from social media, Internet of Things (IoT) devices, homegrown applications, and third-party software, to name just a few sources. According to a 2023 BCG study, "the volume of data generated approximately doubled from 2018 to 2021 to about 84 ZB, a rate of growth that is expected to continue." The researchers "estimate that the volume of data

generated will rise at a compound annual growth rate (CAGR) of 21% from 2021 to 2024, reaching 149 ZB." Companies know that they can save millions of dollars by gathering this data and using it to analyze the past and present and make predictions about the future—but to do that, they need a way to store all that data.

Throughout the business world, the rush is on to build data architectures as quickly as possible. Those architectures need to be ready to handle any future data, no matter its size, speed, or type, and to maintain its accuracy—and those of us who work with data architectures need a clear understanding of how they work and what the options are. That's where this book comes in. I have seen firsthand the result of not properly understanding data architecture concepts: one company I know of built a data architecture at the cost of $100 million over two years, only to discover that the architecture used the wrong technology, was too difficult to use, and was not flexible enough to handle certain types of data. It had to be scraped and started again from scratch. Don't let this happen to you!

It's all about getting the right information to the right people at the right time in the right format. To do that, you need a data architecture to ingest, store, transform, and model the data (big data processing) so it can be accurately and easily used. You need an architecture where any end user, even one with very little technical knowledge, can analyze the data and generate reports and dashboards, instead

of relying on people in IT with deep technical knowledge to do it for them.

Chapter 1 begins by introducing big data and some of its fundamental ideas. I then discuss how companies are using their data, with an emphasis on business intelligence (BI) and how this use grows as a company's data architecture matures.

What Is Big Data and How Can It Help You?

Even though the term "big" is used in big data, it's not just about the size of the data. It's also about all the data, big or small, within your company and all the data outside your company that would be helpful to you. The data could be in any type of format and can be collected in any type of regularity. So the best way to define *big data* is to think of it as *all* data, no matter its size (volume), speed (velocity), or type (variety). In addition to those criteria, there are three more factors you can use to describe data: veracity, variability, and value. Together, they're commonly known as the "six Vs" of big data, as shown in Figure 1-1.

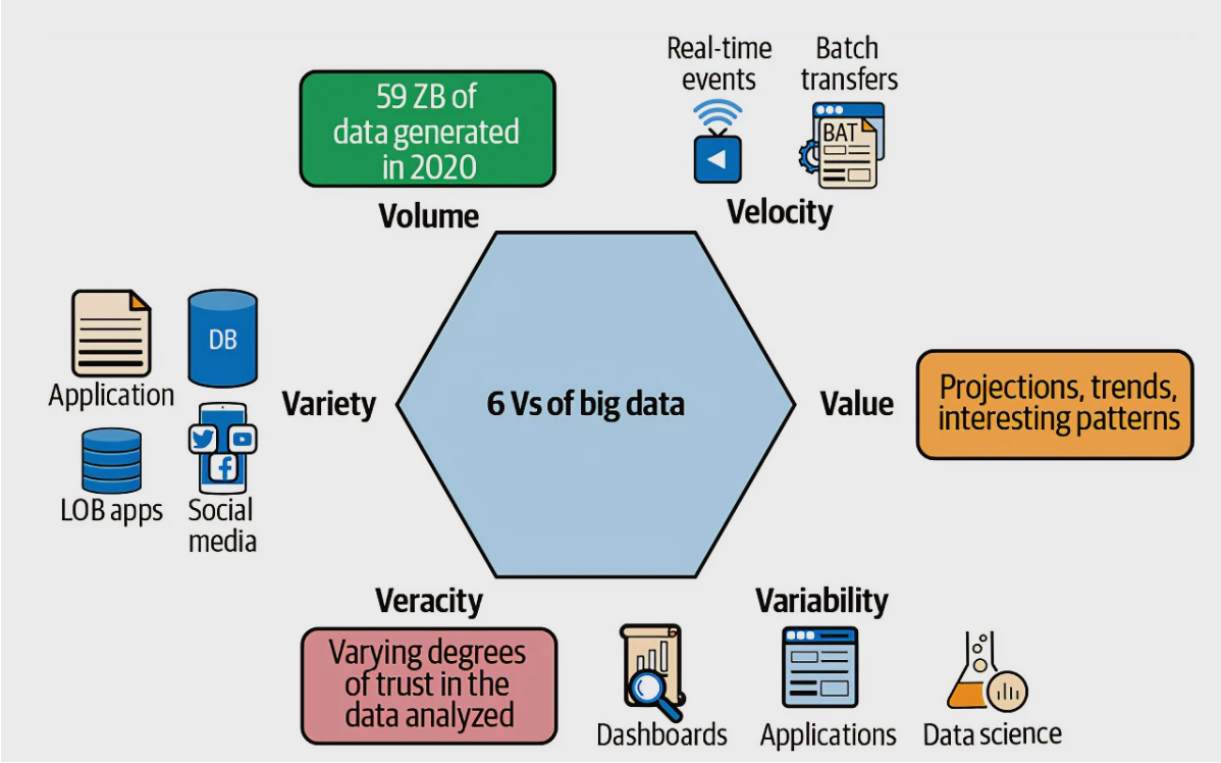


Figure 1-1. The six Vs of big data. Source: The Cloud Data Lake by Rukmani Gopalan (O'Reilly, 2023).

Let's take a closer look at each one:

Volume

Volume is the sheer amount of data generated and stored, which can be anywhere from terabytes to petabytes of data from a wide range of sources that can include social media, e-commerce transactions, scientific experiments, sensor data from IoT devices, and much more. For example, data from an order-entry system might amount to a couple of terabytes a day, while IoT devices can stream millions of events per minute and generate hundreds of terabytes of data a day.

Variety

Variety refers to the wide range of data sources and formats. These can be further broken down into *structured data* (from relational databases), *semi-structured data* (such as logs and CSV, XML, and JSON formats), *unstructured data* (like emails, documents, and PDFs), and *binary data* (images, audio, video). For example, data from an order-entry system would be structured data because it comes from a relational database, while data from an IoT device would likely be in JSON format.

Velocity

Velocity refers to the speed at which data is generated and processed. Collecting data infrequently is often called *batch processing*: for example, each night the orders for the day are collected and processed. Data can also be collected *very* frequently, especially if it's generated at a high velocity or even in real time, such as from social media, IoT devices, and mobile applications.

Veracity

Veracity is about the accuracy and reliability of data. Big data comes from a huge variety of sources. If any sources are unreliable or incomplete, this can damage the quality of the data. For example, if data is coming from an IoT device, such as an

outdoor security camera located at the front of your house that is pointing to the driveway, and it sends a text message to you when it detects a person, it's possible that environmental factors, such as weather could corrupt the data and make it falsely detect a person. Thus, the data needs to be validated when received.

Variability

Variability refers to the consistency (or inconsistency) of data, in terms of its format, quality, and meaning. Processing and analyzing structured, semi-structured, and unstructured data formats require different tools and techniques. For example, the type, frequency, and quality sensor data from IoT devices can vary greatly. Temperature and humidity sensors might generate data points at regular intervals, while motion sensors might only generate data when they detect motion.

Value

Value is the most important "V": it relates to the usefulness and relevance of data. Companies use big data to gain insights and make decisions that can lead to business value, such as increased efficiency, cost savings, or new revenue streams. For example, by analyzing customer data, organizations can better understand their customers' behaviors, preferences, and

needs. They can use this information to develop better targeted marketing campaigns, improve customer experiences, and drive sales.

Collecting big data allows companies to gain insights that help them make better business decisions. *Predictive analysis* is a type of data analysis that involves using statistical algorithms and machine learning to analyze historical data and make predictions about future events and trends. This allows them to be proactive, not just reactive.

You'll hear many companies calling data "the new oil," because it has become an incredibly valuable resource in today's digital economy, much like oil was in the industrial economy. For example, data is like oil because:

- It's a raw material that needs to be extracted, refined, and processed in order to be useful. In the case of data, that involves collecting, storing, and analyzing it in order to gain insights that can drive business decisions.
- It's incredibly valuable. Companies that collect and analyze large amounts of data can use it to improve their products and services, make better business decisions, and gain a competitive advantage.
- It can be used in a variety of ways. For example, if you use data to train machine learning algorithms, you can then use those al-

gorithms to automate tasks, identify patterns, and make predictions.

- It's a powerful resource with a transformative effect on society. The widespread use of oil powered the growth of industries and enabled new technologies, while data has led to advances in fields like artificial intelligence, machine learning, and predictive analytics.
- It can be a source of power and influence, thanks to all of the above factors.

For example, you can use big data to generate reports and dashboards that tell you where sales are lagging, and take steps "after the fact" to improve those sales. You can also use machine learning to predict where sales will drop in the future and take proactive steps to prevent that drop. This is called *business intelligence* (BI): the process of collecting, analyzing, and using data to help businesses make more informed decisions.

As Figure 1-3.2 shows, I can collect data from new sources, such as IoT devices, web logs and social media, as well as older sources, such as line-of-business, enterprise resource planning (ERP), and customer relationship management (CRM) applications. This data can be in multiple formats, such as CSV files, JSON files, and Parquet files. It can come over in batches, say once an hour, or it can be

streamed in multiple times a second (this is called *real-time streaming*).

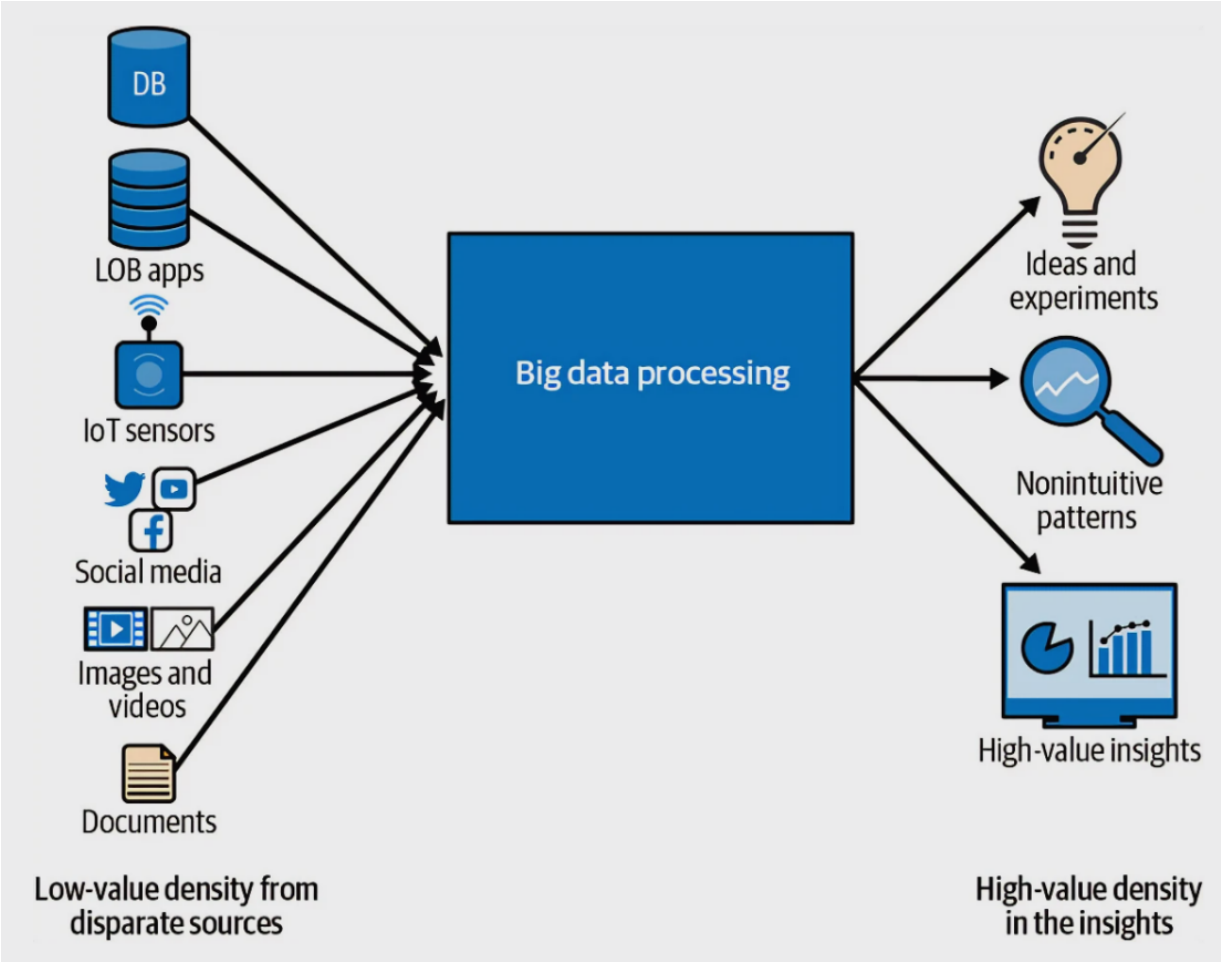


Figure 1-2. Big data processing. Source: The Cloud Data Lake by Rukmani Gopalan (O'Reilly, 2023).

It's important for companies to understand where they are in their journey to use data compared to other companies. This is called data maturity, and the next section shows you its stages so you can understand where your company is.

Data Maturity

You may have heard many in the IT industry use the term *digital transformation*: it means how companies embed technologies across their business to drive fundamental change in the way they get value out of data , and in how they operate and deliver value to customers. The process involves shifting away from traditional, manual, or paper-based processes to digital ones, leveraging the power of technology to improve efficiency, productivity, and innovation. A big part of this transformation is usually using data to improve a company's business, which could mean creating a customer 360 profile to improve customer experience, or using machine learning to improve the speed and accuracy of manufacturing lines.

This digital transformation can be broken into four stages, called the *enterprise data maturity* stages, illustrated in Figure 1-3. While this term is used widely in the IT industry, I have my own take on what those stages look like. They describe the level of development and sophistication an organization has reached in managing, utilizing, and deriving value from its data. This model is a way to assess an organization's data-management capabilities and readiness for advanced analytics, artificial intelligence, and other data-driven initiatives. Each stage represents a step forward in leveraging data for business value

and decision-making. The remainder of this section describes each stage.

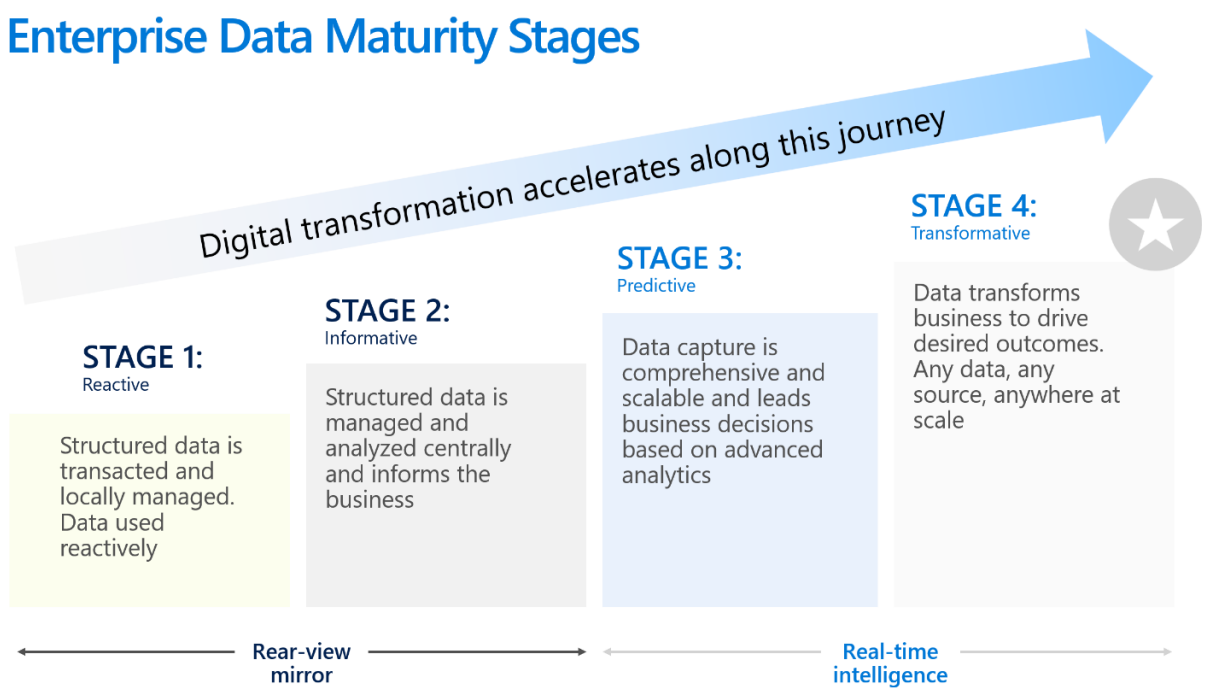


Figure 1-3. Enterprise Data Maturity Stages

Stage 1: Reactive

The first stage is where a company has data scattered all over, likely in a bunch of Excel spreadsheets and/or desktop databases on many different file systems, being emailed all over the place. Data architects call this a *spreadmart* (short for "spreadsheet data mart"): an informal, decentralized collection of data often found within an organization, that uses spreadsheets to store, manage, and analyze data. individuals or teams typically create and maintain spreadmarts are independently of the or-

ganization's centralized data management system or official data warehouse. Spreadmarts suffer from data inconsistency, lack of governance, limited scalability, and inefficiency (since they often result in a lot of duplicated effort).

Stage 2: Informative

Companies reach the second maturity stage when they start to centralize all of their data, making it much easier to analyze the data and create reports from it. Stages 1 and 2 are for *historical reporting*, or seeing trends and patterns from the past, so Figure 1-3 calls them the "rear-view mirror." In these stages, you are reacting to what's already happened.

At stage 2, the solution built to gather the data is usually not very scalable. Generally, the size and types of data it can handle are limited, and it can only ingest data infrequently (every night, for example). Most companies are at stage 2, especially if their infrastructure is still on-prem.[1]

Stage 3: Predictive

Stage 3 is where companies have moved to the cloud and have built a system that can handle larger sizes of data, different types of data, and data that is ingested more frequently (hourly or streaming). They have also improved their decision-making by incorporating machine learning (advanced analytics) to

make decisions in real time. (For example, while a user is in an online bookstore, the system might recommend additional books on the check-out page based on the user's prior purchases.)

Stage 4: Transformative

Finally, at stage 4, the company has built a solution that can handle any data, no matter its size, speed, or type. This is a solution that can last years. Its users never have to say "no" to any request for data, because the architecture can handle it and has the infrastructure capacity to support it. In stage 4, nontechnical end users can easily create reports and dashboards.

Stages 3 and 4 are the focus of this book. In particular, end users doing their own reporting is called *self-service business intelligence*, the subject of the next section.

Self-Service Business Intelligence

For many years, if an end user within an organization needed a report or dashboard, they had to gather all their requirements (the source data needed, plus a description of what the report or dashboard should look like), fill out an IT request form, and wait. IT then built the

report, which involved extracting the data, loading it into the data warehouse, building a data model, and then finally creating the report or dashboard. The end user would review it and either approve it or request changes. This often resulted in a long queue of IT requests, so that IT ended up becoming a huge bottleneck. It took days, weeks, or even months for end users to get value out of the data. This process is now called “traditional BI,” because in recent years something better has developed: self-service BI.

The goal of any data architecture solution you build should be making it quick and easy for any end user, no matter what their technical skills are, to query the data and to create reports and dashboards. They should not have to get IT involved to perform any of those tasks – they should be able to do it all on their own.

This goal requires more upfront work: IT will have to contact all the end users to find out what data they need, then build the data architecture with their needs in mind. But it will be well worth it for the time savings in creating the reports. This approach eliminates the queue and the back-and-forth with IT, whose team members generally have little understanding of the data. Instead, the end user, who knows the data best, accesses the data directly, prepares it, builds the data model, creates the reports, and validates that the reports are correct. This is much more productive.

Creating that easy-to-consume data solution results in self-service BI. Creating a report should be as easy as dragging fields around in a workspace. End users shouldn't have to understand how to join data from different tables or worrying about a report running too slowly. When you are creating a data solution, always be thinking about this result: *How easy will it be for people to build their own reports?*

Summary

In this chapter, you learned what big data is and how it can help you and your organization make better business decisions, especially when combined with machine learning. You saw how to describe big data using the "six Vs," and you learned what data maturity means and how to identify its stages. Finally, you learned the difference between traditional and self-service BI, where the goal is for everyone to be able to use the data to create reports and identify insights quickly and easily.

Let me now give you an idea of what to expect in the following chapters. In chapter 2, I will go into what a data architecture is and provide a high-level overview of how the types of data architectures have changed over the years. Chapter 3 is where I show you how to can conduct an architecture design session to help with determining the best data architecture to use.

Part II, “Common Data Architecture Concepts,” gets into more detail about various architectures. In chapter 4, I cover what a data warehouse is and what it is not, and why you would want to use one. I’ll discuss the “top-down approach,” ask if the relational data warehouse is dead, and cover ways to populate a data warehouse. Chapter 5 describes what a data lake is and why you would want to use one, discusses the bottoms-up approach, and then dives into data lake design and when to use multiple data lakes.

Chapter 6 is about common data architecture concepts related to data stores, including data marts, operational data stores, master data management, and data virtualization. Chapter 7 covers common data architecture concepts related to design, including OLTP versus OLAP, operational versus analytical data, SMP versus MPP, Lambda architecture, Kappa architecture, and polyglot persistence. Chapter 8 is all about data modeling, including relational and dimensional modeling, the Kimball versus Inmon debate, the common data model, and data vaults. And in chapter 9 you will read about data ingestion, with sections on ELT versus ELT, reverse ELT, batch versus real-time processing, and data governance.

Part III focuses on specific data architectures. Chapter 10 describes the modern data warehouse and the five stages of building one. Chapter 11 covers the data fabric architecture and its use cases.

Chapter 12 goes over the data lakehouse architecture and the trade-offs of not using a relational data warehouse.

Chapters 13 ad 14 are both about data mesh architectures: there's a lot to talk about! Chapter 13 focuses on the data mesh's decentralized approach and describes what data domains and data products are. Chapter 14 gets into the concerns and challenges of building a data mesh and tackles some common myths of data mesh. It'll help you check if you are ready to adopt a data mesh. It finishes with a look at different topologies for a data mesh and what the future of the data mesh might look like. Chapter 15 looks at why projects succeed and whey they fail, and describes the team organization you'll need for building a data architecture. Chapter 16 is a discussion on open source, Hadoop, the benefits of the cloud, the major cloud providers, and being multi-cloud. Finally, chapters 17 through 19 cover the data architectures of Microsoft Azure, Amazon Web Services, and Google Cloud Platform, respectively.

Now I'm about to revolutionize your data world. Are you ready?

Being "on-prem," short for "on-premises," refers to an organization hosting and managing its IT infrastructure, such as servers, storage, and networking equipment, within its own physical facilities, usually called data centers. This contrasts with cloud-based services, where these resources are hosted and managed by third-party providers such as Azure, Amazon Web Services (AWS), or Google Cloud

Platform (GCP) in remote data centers. I'll discuss the benefits of moving from on-prem to the cloud in chapter 15, but for now, know that transitioning from on-prem servers to cloud is a huge part of most enterprises' digital transformations.

Chapter 2. Types of Data Architectures

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 2nd chapter of the final book.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the author at *jamesserra3@gmail.com*.

It's absolutely vital to spend time up front designing and building the right data architecture. I found this out the hard way early in my career. I was so excited to start building my solution that I breezed over important decisions about the design of the architecture and what products to use. Three months into the project, I realized the architecture would not support some of the required data sources. We essentially had to restart the project from scratch and come up with another

architecture and different products, wasting a ton of money and time. Without the right planning, end users won't get value out of your solution, they'll be angry about the missed deadlines, and your company risks falling farther and farther behind its competitors.

When building a data solution, you need a well-thought-out blueprint to follow. That is where a data architecture comes into play. A *data architecture* defines a high-level architectural approach and concept to follow, outlines a set of technologies to use, and states the flow of data that will be used to build your data solution to capture big data. Deciding on a data architecture can be very challenging, as there is no "one size fits all" architecture. You can't find a stock list of architecture approaches with corresponding products to use by flipping through a book. There's no simple flow chart that you can follow with decision trees that will lead you to the perfect architecture. Your architectural approach and the technologies you use will vary greatly from customer to customer, use case to use case.

The major types of high-level architectural approaches and concepts are exactly what this book is about. It's not a stock list, but it will give you a sense of what they're all about. Although it's useful to separate data architectures into types based on their characteristics, which I will do in this book, that's not the same thing as choosing from a bunch of predefined one-size-fits-all templates. Each data architec-

ture is unique and requires a customized approach to meet specific business needs.

Data architecture refers to the overall design and organization of data within an information system. Predefined templates for data architectures can seem like an easy way to set up a new system quickly. However, data architecture templates often fail to account for the specific requirements and constraints of the system to which they are being applied, which can lead to issues with data quality, system performance, and maintenance. Additionally, the organization's needs and data systems are likely to change over time, requiring updates and adjustments to the data architecture. A standardized template may not be flexible enough to accommodate these changes, which can introduce inefficiencies and limitations in the system.

This chapter gives a brief guided tour of the major types the book will cover: relational data warehouse, data lake, modern data warehouse, data fabric, data lakehouse, and data mesh. Each type will get its own chapter with plenty of detail later in the book.

Evolution of data architectures

A *relational database* is a type of database that stores data in a structured manner, with relationships between the data elements defined by keys. The data is typically organized into tables, with each table

consisting of rows and columns. Each row represents a single instance of data, while each column represents a specific attribute of the data.

Relational databases are designed to handle structured data, and they provide a framework for creating, modifying, and querying data using a standardized language known as Structured Query Language, or SQL. The relational model was first proposed by E.F. Codd in 1972,[1] and since the mid-70s it has become the dominant model for database management systems. Most operational applications need to permanently store data, and a relational database is the tool of choice for a large majority.

In relational databases, where consistency and data integrity are of primary importance, data is usually organized with an approach called *schema-on-write. Schema* refers to the formal structure that defines the organization of, and relationships, between tables, fields, data types, constraints, and tables. It serves as a blueprint for storing and managing data, ensuring consistency, integrity, and efficient organization within the database. Relational databases (and relational data warehouses) require a bit of upfront work before data can land in them. You must create the database and its tables, fields, and schema, then write the code to transfer the data into the database. With a *schema-on-write* approach, the data schema is defined and enforced when data is written or ingested into the database. Data

must adhere to the predefined schema, including data types, constraints, and relationships, before it can be stored.

By contrast, in the *schema-on-read* approach, the schema is applied when data is read or accessed, rather than when it is written. Data can be ingested into the storage system without conforming to a strict schema, and the structure is defined only when data is queried or consumed. This approach offers more flexibility in storing unstructured or semi-structured data, and is commonly used in data lakes, discussed later in this chapter.

At a high level, data architectures provide a framework for organizing and managing data in a way that supports the needs of an organization. This involves defining how data is collected, stored, processed, and accessed, as well as maintaining data quality, security, and privacy. While data architectures can take many different forms, some common elements include:

Data storage

All data architectures need to specify how data is stored, including the physical storage medium (such as hard drives or cloud storage) and the data structures used to organize the data.

Data processing

Data architectures need to define how data is processed, including any transformations or calculations that are performed on the data before it is stored or analyzed.

Data access

Data architectures need to provide mechanisms for accessing data, including user interfaces and application program interfaces (APIs) that enable data to be queried and analyzed.

Data security and privacy

Data architectures need to incorporate mechanisms for ensuring the security and privacy of data, such as access controls, encryption, and data masking.

Data governance

Data architectures need to provide a framework for managing data, including quality standards, lineage tracking, and retention policies.

Overall, the main goal of a data architecture is to enable an organization to manage and leverage its data assets effectively, in order to support its business objectives and decision-making processes.

Table 2-1 provides a high-level comparison of the characteristics of the data architectures I cover in this book, to give you a starting point

to determine which architecture may be the best fit for your use case.

Table 2-1. Comparison of data architectures

Characteristic	Relational data warehouse	Data Lake	Modern Data Warehouse	Da
Year introduced	1984	2010	2011	201
Centralized vs Decentralized	Centralized	Centralized	Centralized	Ce
Storage Type	Relational	Object	Relational and Object	Re and
Schema Type	Schema-on-Write	Schema-on-Read	Schema-on-Read and Schema-on-Write	Sch Re Sch Wr
Data Security	High	Low to Medium	Medium to High	Hig
Data Latency	Low	High	Low to High	Lov
Time to Value	Medium	Low	Low	Lov

Characteristic	Relational data warehouse	Data Lake	Modern Data Warehouse	Da
Total cost of solution	High	Low	Medium	Me Hig
Supported use cases	Low	Low to Medium	Medium	Me Hig
Difficulty of development	Low	Medium	Medium	Me
Maturity of technology	High	Medium	Medium to High	Me Hig
Company skillset needed	Low	Low to Medium	Medium	Me Hig

Relational data warehouse

Relational databases were the mainstay of data storage for several decades. The first relational data warehouse used in production was the Teradata system, developed at Stanford University by Dr. Jack E. Shemer, who founded the Teradata Corporation in 1979. Wells Fargo Bank installed the first Teradata system in 1983 and used it to analyze

financial data[2]. Relational data warehouses became more widely popular in the late 1980s, about 15 years after relational databases came along. As organizations began generating ever more vast amounts of data, it was increasingly challenging to process and analyze that data without a long delay. The reporting limitations of relational databases led to the development of relational data warehouses.[3] A *relational data warehouse* is a specific type of relational database that is designed for data warehousing and business-intelligence applications, with optimized query performance and support for large-scale data analysis. While both relational data warehouses and transactional processing use the relational model to organize data, a relational data warehouse is typically larger in scale and is optimized for analytical queries.

Relational data warehouses have both a compute engine and storage. The compute engine is the processing power used to query the data. The storage is *relational storage*, which holds data that must structured via tables, rows and columns. The relational data warehouse's compute power can only be used on its relational storage – they are tied together.

Some of the most important features of relational data warehouses include transaction support (ensuring that data is processed reliably and consistently), audit trails (keeping a record of all activity per-

formed on the data in the system), and schema enforcement (ensuring that data is organized and structured in a predefined way).

In the 1970s and 1980s, organizations were using relational databases for operational applications like order entry and inventory management. These applications are called *online transactional processing* (OLTP) systems. OLTP systems can make certain changes to data in a database, referred to as *CRUD operations* – which stands for "create, read, update, and delete." CRUD operations form the foundation of data manipulation and management in data architectures. They're essential in designing and implementing data storage systems and user interfaces that interact with data.

CRUD operations require fast response times from the application, so that end users don't get frustrated at how long it takes to update data. You can run queries and generate reports on a relational database that's in use by an operational application, but doing so uses a lot of resources and can conflict with other CRUD operations running at the same time. That can slow everything down.

Data warehouses were invented in part to solve this problem. The data from the relational database is copied into a data warehouse, and users can run queries and reports against the data warehouse instead of the relational database. This way, they're not taxing the system that houses the relational database and slowing the applica-

tion for end users. Data warehouses also centralize data from multiple applications in order to improve reporting, as pictured in Figure 2-1.

Chapter 4 will discuss data warehouses in more detail.

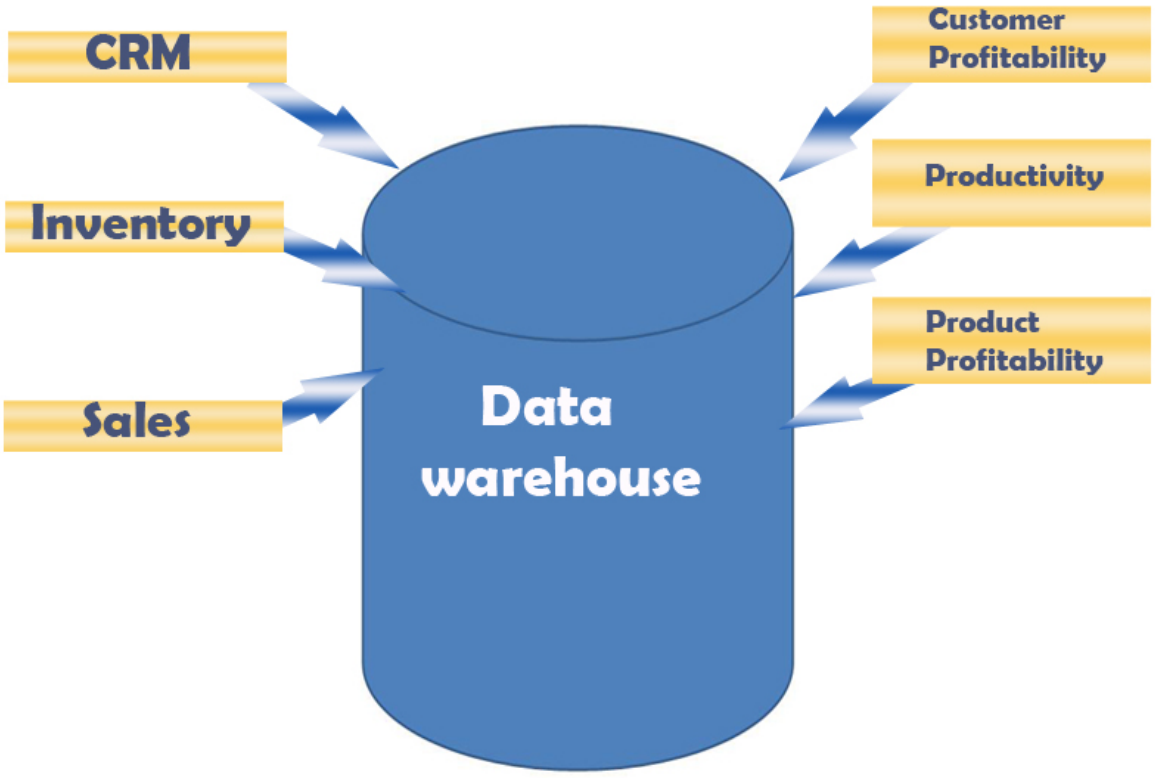


Figure 2-1. Data Warehousing

Data Lake

The data lake is a more recent concept, first appearing around 2010. You can think of a data lake as a glorified file system, not much different than the file system on your laptop. A data lake is simply storage

– unlike a relational data warehouse, there is no compute engine associated with it. Fortunately, there are many compute engines that can work with data lakes, which makes compute power cheaper for a data lake than for a relational data warehouse. Another difference is that while relational data warehouses use relational storage, data lakes use *object storage*, which does not need the data to be structured into tables, rows and columns.

Data lake storage technology started with the Apache Hadoop Distributed File System (HDFS), a free open-source technology[4] hosted almost exclusively on-premises that was very popular in the early 2010s. HDFS is a scalable, fault-tolerant distributed-storage system designed to run on commodity hardware. It is a core component of the Apache Hadoop ecosystem, which I discuss more in the appendix. As cloud computing continued to grow in important, data lakes were built in the cloud using a different type of storage, and most data lakes now exist in the cloud.

As opposed to a relational data warehouse, a data lake is schema-on-read, meaning that no upfront work is needed to put data in the data lake: it can be as simple as copying files into it, like you would with folders on your laptop. The data in a data lake is stored in its natural (or "raw") format, meaning it can go from its source system into the data lake without being transformed into another format. For example, if you export data from a relational database into a file in raw

CSV format, you could store it unaltered in a data lake. If you wanted to store it in a relational data warehouse, however, you'd have to transform it to fit into the rows and columns of a table.

When you copy data files into a data lake, their schema might not be copied along with it, or might be in a different file. So you must define the schema by creating it or pulling it from the separate file, hence the term schema-on-read. As Figure 3-2 shows, data from source systems such as operational application databases, sensor data, and social media data can all land in the data lake. These files could be structured (like data from relational databases), semi-structured (like CSV, logs, XML, or JSON files), or unstructured data (such as from emails, documents, and PDFs). It can even hold binary data (like images, audio, and video).

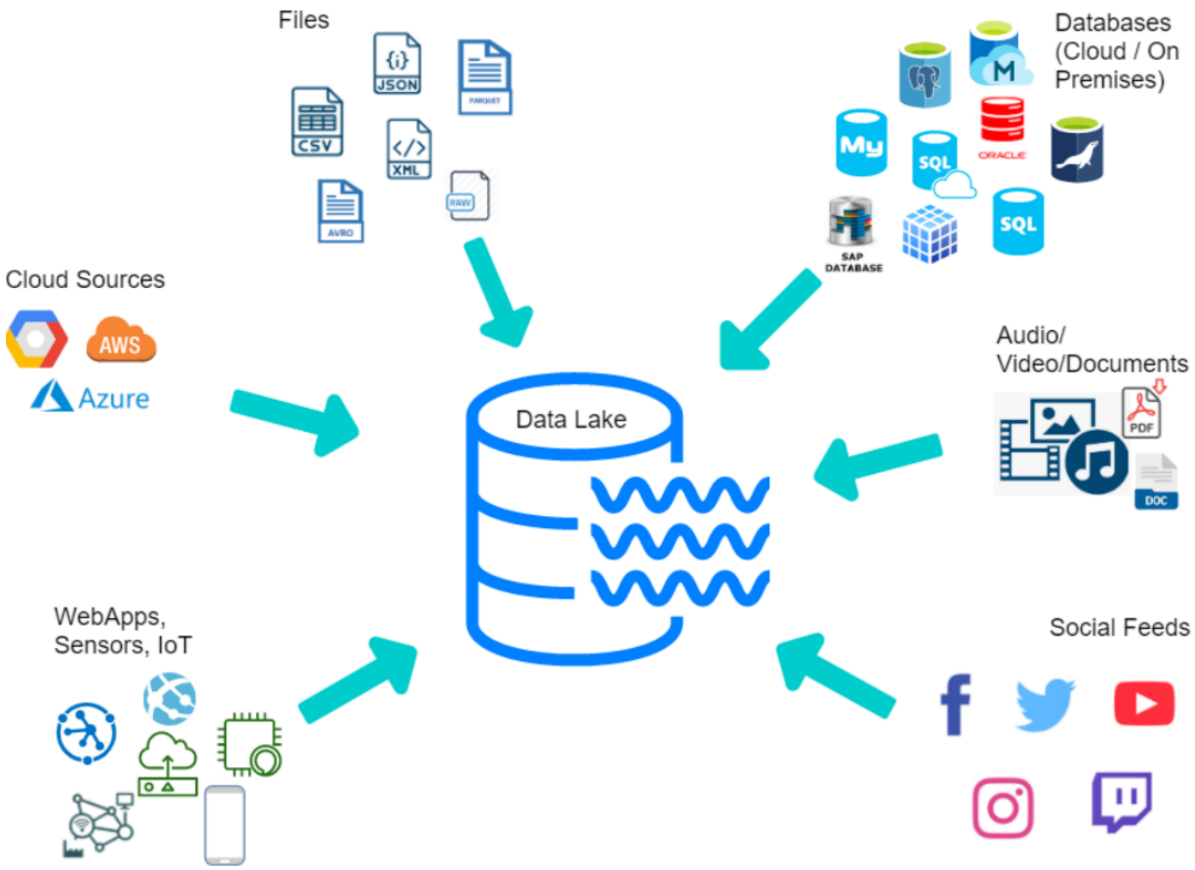


Figure 2-2. Data Lake

Data lakes first started out as the solution to all the problems with relational data warehouses, such as high cost, limited scalability, poor performance, data preparation overhead, and limited support for complex data types. Companies selling Hadoop and data lakes, such as Cloudera, Hortonworks, and MapR, hyped them as if they were filled with unicorns and rainbows that would copy and clean data and make it available to end users with magical ease. They claimed that data lakes could replace relational data warehouses entirely, in a "one technology to do everything" approach. More than a few compa-

nies decided to save money by using free open-source tools for all their technology.

The problem was that querying data lakes isn't actually that easy: it requires some fairly advanced skillsets. IT would tell end-users, "Hey, we copied all the data you need into this data lake. Just go and open a Jupyter notebook and use Hive and Python to build your reports with the files in these folders." This failed miserably, since most end users did not have anywhere near the skillset needed to do all that. Those companies found out the hard way that these complex, difficult-to-use solutions wound up costing more money in hardware and support costs, delays, and lost productivity. In addition, data lakes did not have some of the features people liked about data warehouses, like transaction support, schema enforcement, and audit trails. This resulted in two of the three top data-lake suppliers, Hortonworks and MapR, going out of business.

But the data lake did not go away. Instead, its purpose morphed into a different, but very useful, one: staging and preparing data. Chapter 5 will discuss data lakes in detail.

Modern Data Warehouse

Relational data warehouses and data lakes, on their own, are simplistic architectures. They use only one technology to centralize the data,

with few to no other supporting products. When you use more technologies and products to support relational data warehouses or data lakes, they evolve into the architectures discussed in this and the following chapters. Data lakes failed to replace relational data warehouses, but still had their own benefits in staging and preparing data. Why not have both? Around 2011, many companies started building architectures that place data lakes side-by-side with relational data warehouses to form the data architecture we now call the *modern data warehouse* (MDW), shown in Figure 2-3. The *modern* in *modern data warehouse* refers to the use of newer technologies and approaches to data warehousing.

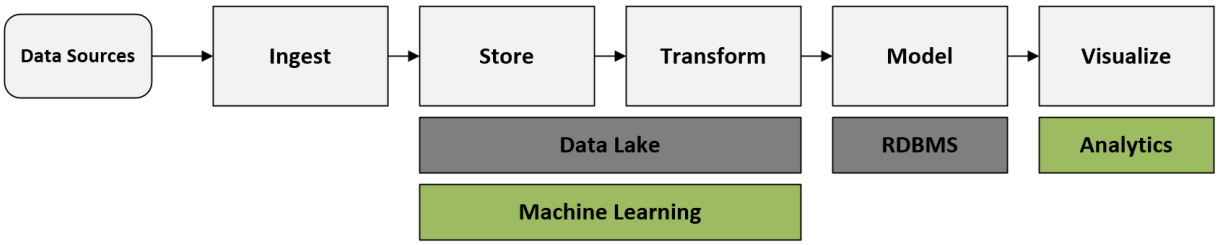


Figure 2-3. Modern Data Warehouse (MDW)

It's sort of a "best of both worlds" approach: the data lake is for staging and preparing data, and data scientists use it to build machine-learning models; the data warehouse is for serving, security, and compliance, and business users do their querying and reporting with it. Chapter 10 will provide much more detail on modern data warehouse architectures.

Data Fabric

Data fabrics started to appear around 2016. You could think of *data fabric* architectures as an evolution of the modern data warehouse architecture, with more technology added to source more data, secure it, and make it available, as well as improving how the system ingests data, transforms, queries, searches, and access data, as Figure 2-4 shows. With all those additions, the system becomes a "fabric": a large framework that can ingest any sort of data. Chapter 10 will go into this in more detail.

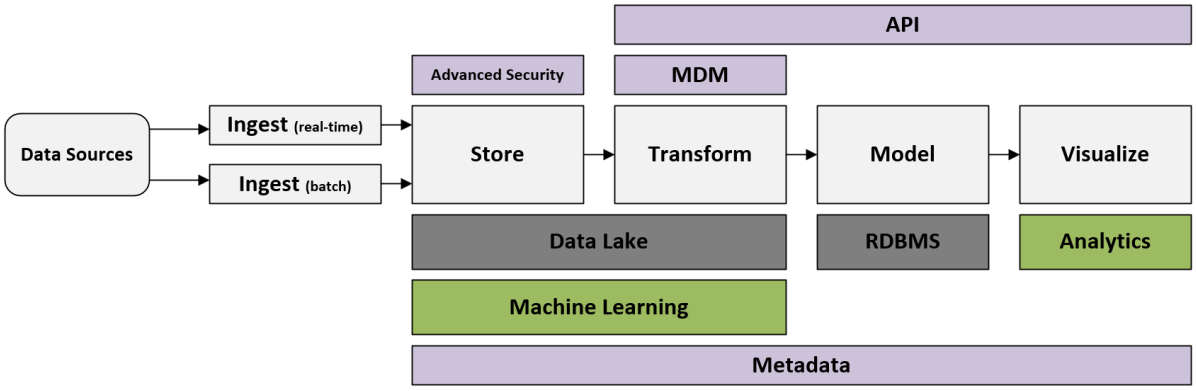


Figure 2-4. Data Fabric

Data Lakehouse

The term *data lakehouse* is a portmanteau (blend) of *data lake* and *data warehouse*. Data lakehouse architectures became popular around 2020, when the company Databricks started using the term. The concept of a lakehouse is to get rid of the relational data ware-

house and use just one repository, a data lake, in your data architecture. All types of data—structured, semi-structured, and unstructured—are ingested into the data lake, and all queries and reports are done from the data lake.

I know what you're thinking: "Wait a minute. You said that data lakes took this same approach when they first appeared, and it failed miserably! What changed?" The answer, as shown in Figure 2-5, is a transactional storage software layer that runs on top of an existing data lake and makes it work more like a relational database. The competing open-sourced options for this layer include Delta Lake, Apache Iceberg, and Apache Hudi. All of this will be covered in more detail in chapter 12.

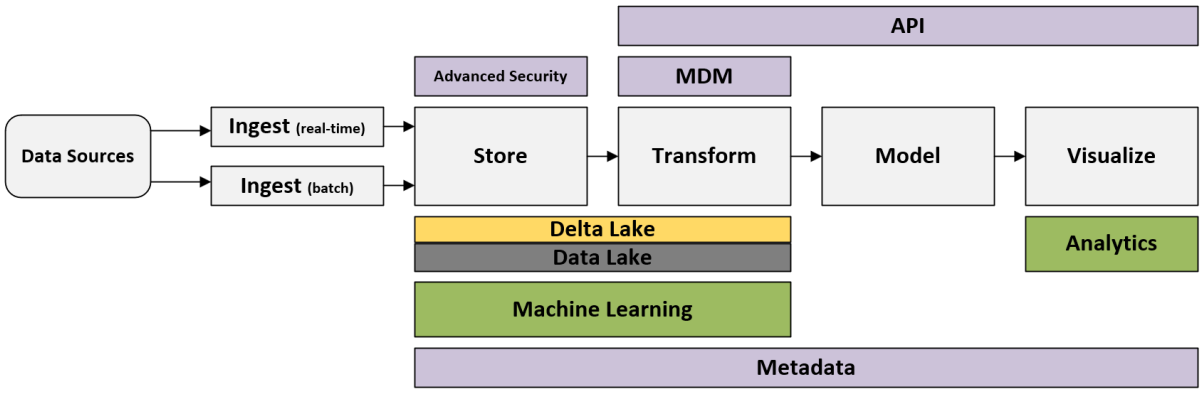


Figure 2-5. Data Lakehouse

Data Mesh

The term *data mesh* was first introduced in a May 2019 blog post by Zhamak Dehghani, founder and CEO of NextData and author of *Data Mesh: Delivering Data-Driven Value at Scale* (O'Reilly, 2022). In December 2020, Dehghani further clarified what a data mesh is and set out four underpinning principles. Data mesh architectures have been an extremely hot topic ever since: getting talked about in tons of blogs, presentations, conferences, and media coverage, and even appearing in the Gartner "Hype Cycle" for data management. There is a lot to like about data mesh architecture, but despite the hype, it is only a fit for a small number of use cases.

The previous three architectures discussed in this book all involved *centralizing* data: copying it into a location owned by IT under an architecture that IT controls. The aim of the data mesh is to solve the main three challenges with a centralized approach: data ownership, data quality, and organizational/technical scaling. In a data mesh, data is kept within several domains within a company, such as manufacturing, sales, and suppliers. Each domain has its own mini-IT team that owns its data, cleans it, and makes it available, as well as its own compute and storage infrastructure. This results in a *decentralized* architecture where data, people, and infrastructure are scaled out – the more domains you have, the more people and infrastructure you get. The system can handle more data, and IT is no longer a bottleneck.

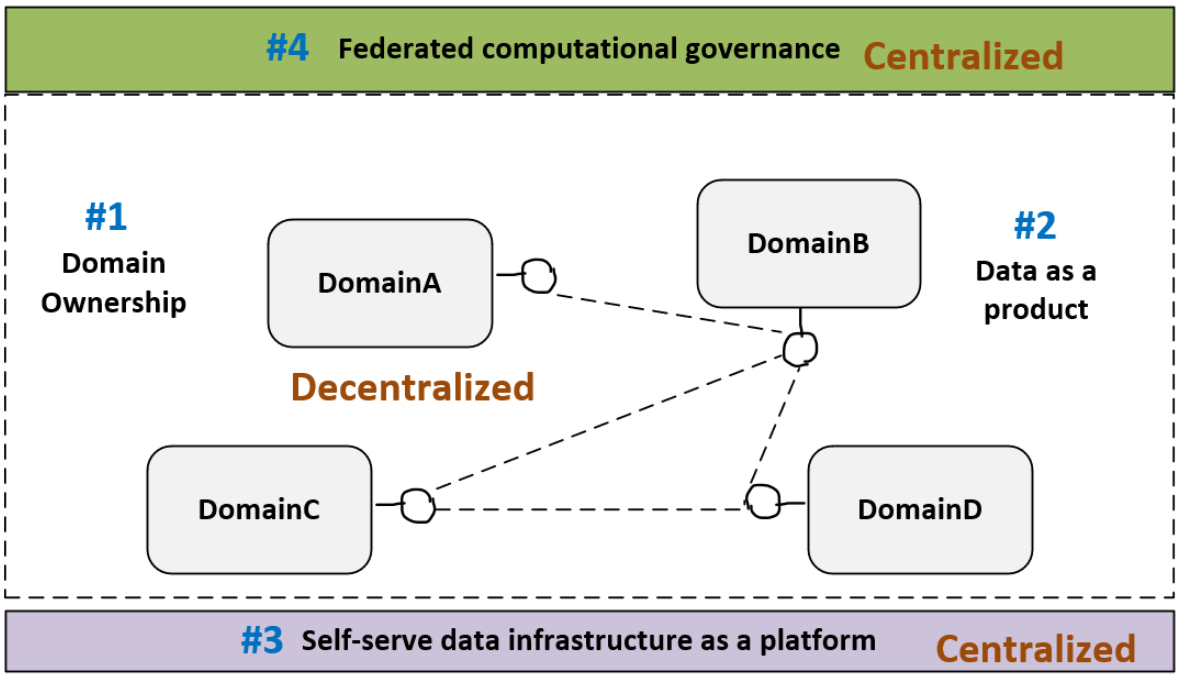


Figure 2-6. Data Mesh

It is important to understand that data mesh is a *concept*, not a technology. There is no "data mesh in a box" that you can buy. Implementing data mesh involves a very big organizational and cultural shift that very few companies are ready for. (Indeed, most companies aren't even large enough to be considered for a data mesh architecture: this is very much an enterprise technology.) Building it requires determining which pieces of already existing technology you can repurpose for it, and which pieces you will have to create on your own. Each domain gets to determine what technologies it will use to build out its part of the data mesh, which could include building a modern data warehouse, data fabric, or data lakehouse. There is much to talk

about regarding data mesh architectures, and I'll do that in chapters 12 and 13.

Summary

Now that you have a high-level understanding of the types of data architectures, the next chapter will talk about how to determine the best data architecture to use: a process called an *architecture design session*.

1. E.F. Codd, "Relational Completeness of Data Base Sublanguages," *Database Systems*, 1972, available at https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.86.9277.

2. Teradata's first system was actually called a "database machine". The term "data warehouse" was not used until the mid 80's

3. A note on language: You might see people refer to the relational data warehouse architecture as *traditional data warehouse* or *relational database management system* (RDBMS). In this book, I mostly use *relational data warehouse*, sometimes shortening it to *data warehouse.* These terms all refer to one and the same thing.

4. *Open source* refers to software whose source code is made available to the public for use, modification, and distribution.

Chapter 3. The Architecture Design Session

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 3rd chapter of the final book.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the author at *jamesserra3@gmail.com*.

I have conducted hundreds of architecture design sessions and have found them to be vital to building a successful data solution. This chapter walks you through how to hold a successful ADS that puts your data solution on the right path.

What Is an ADS?

An *architecture design session* (ADS) is a structured discussion with business and technical stakeholders, driven by technical experts and focused on defining and planning the high-level design of a solution to collect data for specific business opportunities. The first ADS is the start of the architecture process and will lead to many more discussions (including, quite possibly, other ADSs) to support the data-solution project. The ADS should produce two deliverables:

- an architecture (or "blueprint") that can serve as a starting point for the data solution
- a high-level plan of action (including, for example, follow-on demonstrations, proofs of concept, and product discussions

An ADS is not a technical workshop, a technical training, a tech demonstration, or a low-level requirements session.

Why Hold an ADS?

There are many reasons to hold an ADS. First, it often brings together multiple stakeholders from the same organization, including some who have never met face-to-face. Even just the act of getting together in person or virtually can foster creativity and problem solving. Further, this may include stakeholders present who understand their

business very well and who can describe in great detail the specific use cases they are looking to solve.

The ADS also differentiates projects with a solid business case from ‘science projects’ meant to test some technology or product. It flushes out the reasons for the project and, if it indeed a science project, limits the resources devoted to it.

What’s more, an ADS provides a structured framework for ‘thinking big, starting small.’ As noted, this involves coming up with a high-level architecture, but should also involve discussing how you can start small to get quick wins. For example, the architecture may ultimately involve ingesting data from dozens of sources, but the ADS might choose a “low risk, high reward” approach, such as starting with two or three high-value data sources that will require fewer estimated work hours to complete.

You can also include your organization’s customers and/or partner organizations in an ADS. When I was about to hold my first ADS, another Microsoft described the process as “business therapy.” I quickly found out what he meant. We had about a dozen attendees from a customer company, most of whom had never met. We started discussing their current environment, and a few brought up some pain points that were blocking progress. To my surprise, another attendee from that organization chimed in with advice on how to solve the

blocker. Then they solved two more blockers while I sat back and listened. (Of course, my co-workers and I often help with removing blockers, too, but it was enlightening to allow the others to work it out amongst themselves by being quiet.)

You'll learn from your customer in other ways, too, especially if your account team hasn't been as thorough as they might have been. You can dig into any questions you have about their current environment, their pain points, their goals, and other information you'll need.

A customer ADS also gives your account team many ways to follow up, such as demos, proofs of concept, and answers to technical questions, etc. It's a great way to "kickstart" an account that hasn't been getting much traction and accelerate the sales cycle. And if you're working with a partner, such as a consulting firm, consider bringing them into the ADS alongside the customer in order to educate them both on how to use the technology. Make sure to clear it with the customer first, especially if you'll be discussing sensitive information.

Before the ADS

In this section, I'll assume that you are the architect who will be leading the ADS. Even if you're not, though, I recommend you read this

section before participating in an ADS.

Preparing

Set aside at least a day to prepare for an ADS workshop. You'll need to prepare for the logistics of the meeting. Is it physical or virtual? If physical, do you have a big enough room? If virtual, what platform will you use? Does it have the necessary equipment, such as a white-board, audio/video, and other supplies? Who will take care of lunch?

If the ADS will be in-person, and especially if a large group is traveling to get there, I usually set aside a full 7- to 8-hour day, since logistics may make it difficult to get them all in the same room again. If much of the technology is new to the customer, however, 6 or 7 hours is often enough. If you notice they are feeling overwhelmed, end the ADS early, because at that point anything else you cover will be forgotten.

For remote ADSs, two 4-hour days is usually better than one full day. I try to hold the two 4-hour session on back-to-back days or, at least, within a week of each other.

Make sure you find out the project budget and timeline and identify the decision maker. This is important whether the meeting is internal or not, but if you have a long list of customers who want to hold an ADS, it can help you focus on your best prospects. If the ADS is for a

customer, read their latest corporate report for insight on their wider interests and concerns.

I also recommend you hold a pre-call to speak with the account team in your company that is supporting the customer. You'll likely want to discuss:

- details about the customer and the project's background and context
- the customer's problems with their current architecture
- how you can help the customer
- the account team's goals and the outcome they want from the meeting (for example, do they want the customer to request a proof of concept?)
- how well the customer knows your products
- how you will run the ADS and what you expect from the account team

The deliverables from this meeting will be an email to the account team that recaps the call, a draft agenda that they can send to the customer, a seating chart (if the ADS will be held in person), and a reminder for the account team to set up a pre-call with the customer.

Make sure to tell the customer the main reason for the pre-call is to help you all prepare as much as as possible for the ADS, so you can

brush up on topics as needed or bring in subject matter experts (SMEs) to help with areas that you are not familiar with. You'll also share the agenda and together you can make sure it does not include any off-topic items. For example, if the focus of the ADS is on data architectures, discussions of security or DevOps are best taken care of outside the ADS.

The customer pre-call should include a few things:

“Here’s what I understand so far…”

First, recap your understanding of the problem or opportunity, so the customer has a chance to correct any misperceptions.

“Tell me about the attendees…”

Determine who will be in attendance and what roles they play in the organization. This will help you determine how deep or high-level to make the technology discussion.

“Here’s what will happen…”

Provide an agenda and go over it with the customer, but assure them that the discussion can pivot if needed. Explain the approach you will take in running the ADS and set their expectations accordingly. For example, it’s a good idea to tell them to be 100% focused–no phones or emails for anyone involved, including you.

“What do you have for us to look at?”

This is the discovery phase. Encourage them to send you any architecture diagrams for their current solution as well as any architectures they might have drafted for their future solution, as

well as any documents describing their current solution and future plans.

“What would a successful ADS look like to you?”

Make sure you understand their goals going into the ADS.

“Is there anything else that I should be aware of?”

This question can surface questions or goals that might otherwise be missed.

Figure 3-1 shows a sample ADS agenda.

Your Company Name

Architecture Design Session

Modern Data Warehouse

Agenda

9:00 – 9:15	**Welcome and Introductions** Review and outline the goals and outcomes that COMPANY X is looking to drive as part of this strategy briefing	**Architect name**
9:15 – 10:30	**Background / Discovery / Current State Review** Review current data platform and capture future state vision	**All**
10:30 – 12:00	**Modern Data Warehouse Overview** High-level discussion of the modern data warehouse architecture. Ingest, Store, Transform, Model, Visualize.	**Architect name**
12:00 – 12:30	**Lunch**	
12:30 – 2:00	**Data Lake Concept** How to organize the data lake and data governance	**Architect name**
2:00 – 3:45	**Product use case discussion** What products to use, there use cases, and what would the solution look like	**Architect name**
3:45 – 4:00	**Wrap Up Day** Summarize discussions and capture action items	**All**

* Breaks will be added as needed

Figure 3-1. Sample agenda for an ADS about a modern data warehouse project

Make sure to schedule breaks into the agenda, and don't skip them. It may be tempting to bypass a break if you are in a groove with the discussion, but that's usually a mistake. Ask someone on your account team to remind you to take a break in case you forget.

Set aside some time beforehand to get to know the ins and outs of any tools you will use to whiteboard (for example, a Surface Hub) so you aren't figuring things out during the ADS. Pretend you are in an ADS and use the whiteboard to write down goals, pain points, and fol-

low-ups and to draw the architecture, based on your guesses as to how the customer will answer your questions. You can even ask a co-worker to role-play as the customer to help you out.

Inviting Participants

The people who should participate in the ADS will vary a bit depending on whether the customer is internal (a group within your company) or external (an outside company you do business with).

From the customer side, or if it's an internal ADS, from the group for which you are doing the ADS), attendees should include:

- The sponsors (at least one from business and one from IT)
- Business representatives
- Technical representatives
- Project manager
- Advisors, architects, developers, and infrastructure or operations people, as necessary

From your team, include:

- An architect to facilitate the session and make sure the ADS meets its objectives
- From the account team, an account executive, account specialist, and/or cloud solution architect

- Subject matter experts (SMEs) to provide in-depth knowledge on specific topics
- At least one person to take notes (possible a member of the account team)

If the customer wants to talk about a subject area that you are not familiar with, in most cases it's better to bring in an SME instead of trying to learn the topic yourself, especially if your time is limited. Make sure to thank the SME and send their manager a nice email afterward. When a SME attends the ADS, everyone wins: the customer is happy to get expert advice, the account team is happy because the customer is satisfied, and you can learn from the SME.

If you're new to facilitating, you might also want to ask a mentor to participate, to back you up during the ADS (by answering questions that you can't to answer) and to give you feedback afterward on what you did well and what you could do better.

Sometimes I arrange the agenda to allow for people to attend only part of the day. For example, I might plan for the morning to be more about business and discovery, and save the technical side for the afternoon and delve into the proposed architecture and technologies. In this example, the C-level executives would likely attend the morning, and the technical people attend the afternoon. However, anyone who wants to can attend the whole day – maybe some C-level people are

interested in the technical discussion, while some technical people are interested in discovery.

Conducting the ADS

Remember, you're in charge, so it's up to you to set the tone and keep the meeting on course. If someone brings up an off-topic point, say that you can discuss it offline, then write it on the whiteboard for follow-up. Similarly, if you or someone else agrees to follow up on a specific item, add it to the whiteboard. Let the account team take notes—you need to focus on talking with the customer and whiteboarding.

At the halfway point, check the agenda and goals. If you are behind, mention to the customer that the meeting will need to stay on track to get through all the goals. If it seems you will not have time to cover all the goals, ask the customer to re-prioritize the goals, so that any goals you may miss are the lowest priority.

Introductions

At the start of the ADS, have everyone introduce themselves. Ask each participant to state their name, their role, and what they know about the technology you will discuss (if anything). Then, perhaps most important, ask what they want to get out of the ADS (their learn-

ing goals). Write the goals on the whiteboard and prioritize them as a group. If some attendees were not on the pre-call, make clear what the ADS will cover.

Use this time to explain how and why the ADS has been arranged, and set ground rules for the rest of the day. For instance, you might tell them that you'll send them a copy of the final whiteboard, so they don't need to take pictures of it; that it's a very interactive session and you encourage lots of questions; that the agenda is just a recommendation and the group can cover other topics if valuable; and that you can have follow-up meetings to cover topics if you run out of time, so you 're not rushing through important conversations.

Discovery

Discovery is where you spend an hour or two at the beginning of the ADS asking questions about things like:

- the customer's current pain points
- their current technology and architecture
- their future architecture (if any has been discussed)
- any decisions they've already made about technologies, products, or tools they use or plan to use
- current and future use cases
- details on their business

The customer should be doing most of the talking, especially early on in the ADS. You won't learn anything about them if you do all the talking. I always start out asking a ton of questions and getting answers before I go into "education mode," where I start talking for long periods. Even then, I always make sure to pause for questions. You may want to let them know early on that you will stop periodically for questions.

A good architect asks lots of questions. As the saying goes, "you don't know what you don't know." Experienced architects are aware of all the available architectures, techniques, and tools and keep up with the constantly changing landscape of technologies and products. They have the expertise to ask questions that will help them choose the right technologies and products. Discovery is the best way to narrow down the product options to a workable number that you can consider. Then you can come up with the best architecture and products for the particular use case.

By way of analogy, say you're selling medical instruments out of a van filled with a thousand instruments. (It doesn't usually work that way, but bear with me.) You wouldn't just walk up to a doctor, point to your van, and have them look at all the instruments to find the ones they like. Instead, you would ask the doctor a bunch of questions: about their practice area, instruments, and budget, and from those questions you would pull out only a small number of instruments for

the doctor to look at. Architects do the same thing, and the discovery phase of the ADS is a great opportunity to ask your questions while the appropriate people are in the room.

I'll pass on another great bit of advice I received early in my career: As you start asking questions, make sure to "stay off the solution": that is, *don't* talk about architectures or products until you are done asking all the discovery questions. When I first started running ADSs, as soon as the customer mentioned they wanted to build a data warehouse, I would jump right into demonstrating a Microsoft product. I should have waited and asked all of my questions to make sure I was clear on what the customer was trying to accomplish. Instead, I sometimes discovered that the customer really didn't need a data warehouse at all, which meant my efforts on the data warehouse product were wasted.

As you plan your ADS, take a look at this list of recommended questions. You'll notice that none of them is about any specific product. This allows you to begin creating an architecture in a vendor-neutral way. Later on in the ADS, you can ask what products they currently use and what cloud provider they prefer, and then apply products to the architecture.

ADS questionnaire

Here is a list of questions that I have asked in the numerous ADSs I have conducted, where I spend an hour or two at the beginning of the meeting doing “discovery”:

- performance?
- scale?
- storage?
- concurrency?
- query times?

Is your business using the cloud?

> Nowadays, the answer is almost always yes. If not, evaluate why and see if you can overcome it. Many such customers just need to better understand the benefits of the cloud (see chapter 16.3). Others have outdated reasons for not using the cloud that have long been taken care of, like “not enough security.”

Is the data lake you’re considering building a new solution or a migration?

> If this is a migration, then part of the conversation will need to be on how to migrate the existing solution to the new solution.

What are the skillsets of the engineers?

> Knowing what products and technologies the data engineers are familiar with heavily influences the products I recommend.

If they are already familiar with the tool, they'll need less training time, which will likely save costs.

Will you use non-relational data?

In other words, what kind of variety will the data have? If non-relational data is coming from any of the sources, they'll definitely need a data lake, so this is a good time to introduce them to data-lake technologies and design.

How much data do you need to store?

What volume of storage is needed? The size of the data will heavily influence performance, so if it's a large volume of data, you'll need to discuss how to design the architecture (for example, the folder structure of the data lake) to avoid performance issues such as slow queries and reports.

Will you have streaming data?

What kind of velocity is needed? Streaming data sources, such IoT devices, influence the types of products needed to support that kind of data.

Will you use dashboards and/or ad-hoc queries?

Knowing how the data will be used will influence not only the types of products you recommend, but also the system's per-

formance needs. For example, with dashboards, you need millisecond response time so end-users can slice and dice the data without noticeable delays.

Will you use batch and/or interactive queries?

Make sure you understand what types of queries will go against the data. This will influence the type of storage used and the compute needed to ensure acceptable performance.

How fast do the reports need to run? Are there service level agreements (SLAs) in place with specific requirements?

Whether the reports need to run in milliseconds or minutes will affect the architecture, the products used, and how many copies of the data will be needed to get to an acceptable level of performance. For example, the data may need to be aggregated.

Will you use this data in predictive analytics or machine learning?

If so, discuss the ML or analytics products to use and ways to gather the data that will facilitate easily training the ML models.

What are your high-availability and/or disaster-recovery requirements (such as Recovery Time Objectives and Recovery Point Objectives)?

Most cloud providers build in all the high availability the average customer needs, but supporting any specific high-level requirements could require a change in the architecture to. If disaster recovery is required, depending on the objectives (RPOs), you might need to make major additions to the architecture. This is also a good question to start the customer thinking about disaster recovery, if they have none built into their current solution.

Do you need to master the data?

Master data management (MDM) involves creating a single master record for each person, place, or thing in a business, from across internal and external data sources and applications. These master records can then be used to build more accurate reports, dashboards, queries, and machine learning models. I explain MDM in detail in chapter 6. If MDM is needed, you will have to show how it fits into the architecture and recommend products.

Are there any security limitations with storing data in the cloud (for example, defined in your customer contracts)?

This is a very important question: contracts can limit what the organization can do with its customers' data. For example, if they keep data in certain countries and that data can't leave the

country, you will to architect need a solution with multiple data lakes. Similarly, if they cannot store any personally identifiable information in the cloud, you will need to talk about how and when to anonymize that data.

Does this solution require 24/7 client access?

If so, the architecture needs to minimize or eliminate downtime for any of the components. For example, if they're using a relational data warehouse, the solution needs to make sure there is no need for a maintenance window to load and clean data.

How many concurrent users will be accessing the solution at peak times? How many on average?

This is important to know, because some products limit how many concurrent users they support. If the number of concurrent users is over or close to such a limit, your design should avoid those products or combine them with other products to minimize the chance of exceeding a limit.

What is the skill level of the end users?

This will determine what products you recommend: if the skill level is low, you might recommend no-code/low-code products. For example, if I mention Spark and get a blank stare from the customer, I won't even talk about Spark-type products but will

instead focus on easier-to-use products. If the skill level is high, you can consider higher-code options.

What is your budget?

The project budget, of course, has a huge impact on the architecture and the products you choose. If you have a large budget, there are no constraints on the architecture – you can design whatever you wish. In reality, you will often run into customers with low budgets. This forces you to make tradeoffs: for example, reducing financial costs by using less compute, but taking a hit to performance (as data loading and reporting will take longer).

What is your planned timeline?

If the timeline is long, which is usually the case when building a data warehouse solution, meet with the cloud provider to find out what products are in development but haven't been announced yet, so you can educate the customer about anything worth waiting for. The last thing you want is to start building a solution with one product, and then find out 4 or 5 months into development that the cloud provider is about to release a better product.

Is the source data cloud-born and/or on-prem born?

If some of the data sources are on premises, you'll need a pipeline from the on-prem source to the cloud. Depending on the volume of the data, this could force architecture changes, such as using a product that allows for a bigger pipeline.

How much data needs to be imported into the solution every day?

If a lot of data needs to be uploaded each night, transferring it could take too long. Instead of uploading a large file every night, the system may need to upload data on a more frequent basis, such as hourly. Also, look into architectural changes that can speed uploading, such as doing parallel data uploads, compressing the data before uploading, or using products that perform faster file transfers.

What are your current pain points or obstacles with regard to:

- performance?
- scale?
- storage?
- concurrency?
- query times?

You need to know these pain points so you can design an architecture that will address and overcome them. No one wants to pour time and money into building a new solution tthat

leaves them with the same problems, so you'll have to convince the ADS attendees that the solution you're whiteboarding will solve them.

Do you want to use third-party and/or open source tools?

Very few customers will be "all in" to a specific cloud provider and only use that cloud provider's products. So, when you finish the high-level architecture and get to the point of talking about products, you might recommend using third-party and open source tools that the engineers are already familiar with.

Are you okay with using products that are in public or private preview?

Before a product is made generally available (or GA'd), it starts out in private preview (available only to a few invited customers), then public preview (anyone can use it). This is done to make sure the product is as stable and bug-free as possible. The negatives of using a product before it is GA'd include limited functionality, potential bugs, lack of support, unpredictable changes, and uncertainty around pricing. If the company does have some appetite for using private or public preview products, they can get a jump on using a new product or feature.

Some companies have strict policies that forbid using products before they are GA'd. However, this policy may apply only to

products that the company has put into production, which means they can use preview products that will be GA'd by the time the solution is put into production. (It takes a long time to build a data architecture, so this happens a lot.)

What are your security requirements? Do you need data sovereignty?

The answers to these questions can change the architecture in many ways. You might need to build in encryption or extra security features. The requirement to have multiple data lakes because data can't leave a country might apply here, too.

Is data movement a challenge?

Data movement is the process of extracting data from source systems and bringing it into the data warehouse or lake. If the data is coming from many different sources or if there's a mixture of on-prem and cloud sources, your architecture decisions will have to accommodate these various sources. For example, the system might need to use replication software to pull data from a source system into the data lake more quickly.

How much self-service BI would you like?

If the end-users have limited or no technical skills but want to be able to create their own reports, then the architecture will

need to go to extra lengths to put the data in an easily consumable format. This might mean a relational data warehouse (see chapter 4) and/or a star schema (see chapter 8). If the end-users are all very technical, then a data lakehouse (see chapter 12) may be the best architecture.

Whiteboarding

Use a whiteboard, not presentation slides. An ADS is all about discovery; there's lots of talking, and then you'll move on to whiteboarding. Too many slides, and the ADS becomes just another presentation. Your whiteboard should include a rough diagram of the architecture, as well as places for goals, pain points, and items for follow-up. Figure 3-2 shows what a final whiteboard might look like for an ADS focused on a modern data warehouse:

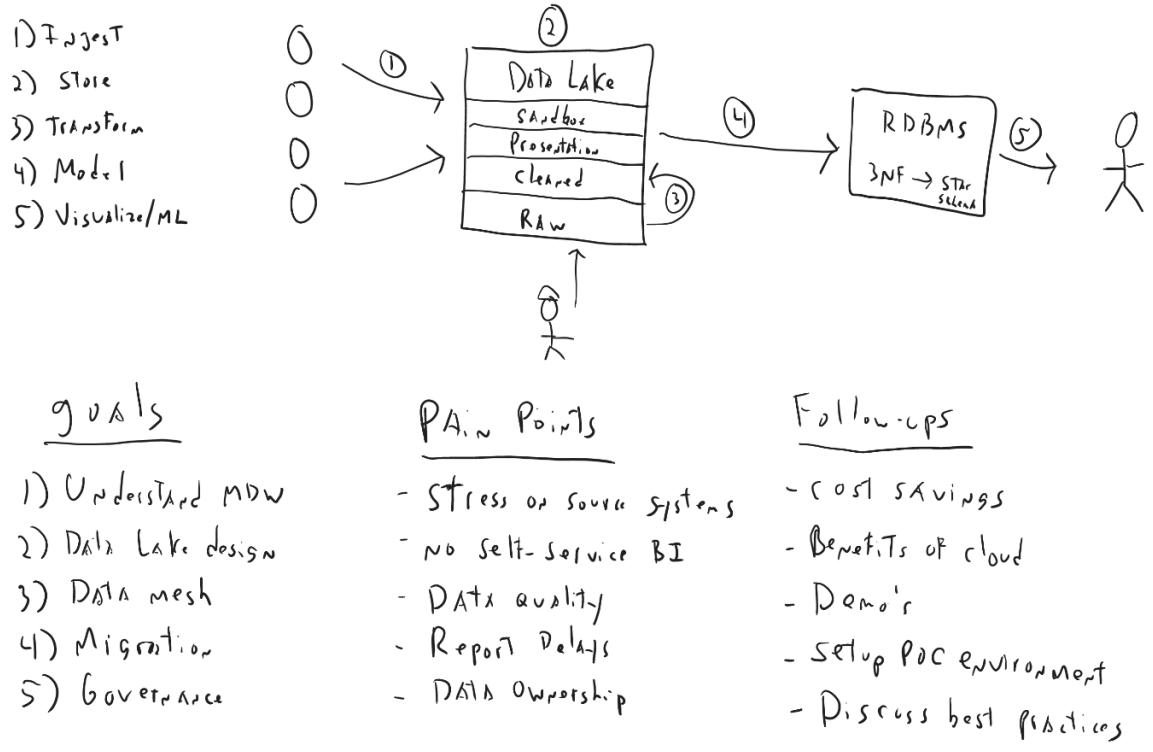


Figure 3-2. A typical whiteboard after the end of an ADS

The whiteboard contains not only the architecture, but prioritized goals, pain points, and “parking lot” items and things to follow up on.

List all the pain points listed on the whiteboard, and make sure to check them off as you solve them. By the end of the ADS, all of the pain points should be addressed in the architecture you have laid out. If not, schedule a follow-up meeting with the customer to address missed pain points. The same applies to goals: if you can’t address them all, schedule a follow-up meeting.

Outcomes/Follow-up

Reserve the last 30 minutes of the ADS to discuss follow-up items.

After the ADS

Debrief with the account team after the ADS. Ask them what went well and what you can do better. This is part of a growth mindset and a great way to get feedback and improve your ADS sessions. You should also communicate the outcomes of the ADS to all stakeholders. This might include a high-level summary for upper management and a more detailed report for those directly involved in the project. Last, collect all the materials you used for the ADS into a digital folder that you can use as a reference for future ADSs. For example, I store all my previous whiteboards; I've found that they're great to review when you're planning an ADS similar to one you have done before.

FOLLOWING UP

Shortly after the ADS, email the customer with the following information:

Summary document

A brief summary of the major points discussed during the ADS

Physical architecture

If you used a digital whiteboard, export the end result to a file that you can send to the customer.

Action items

Any next steps that you agreed on, such as “meet next Tuesday to discuss a proof-of-concept” or “customer will email a diagram of their current architecture”

Parking-lot items and follow-ups

You tracked these on the whiteboard, and in the email you can go into more detail and list who is responsible for carrying out each item. This provides the customer an opportunity to clarify anything you didn’t get quite right.

Survey

> At an in-person ADS, it's best to hand a survey to each participant at the end. That way, most, if not all, attendees will fill it out—especially if you joke with them that they can't leave until they do! If it's a remote ADS, include a link to the survey in your email and tell them it only takes two minutes to complete. Consider making this request during the last break of the day, instead of at the very end of the ADS.

Tips

While it's rare, I've seen my fair share of difficult people decide to challenge me during an ADS. Sometimes a person in the room is afraid of change, afraid a new architecture will put them out of a job (especially when the ADS is about a cloud migration), or simply believes they already have all the answers. If you find yourself dealing with someone like this, don't get into an argument—even if you are "right." Just reply with something that defuses the argument and gets the ADS back on track. For instance: "You have an excellent point and I'd like to dive into that further, but we'll need to do that offline at another time so that we can stay on topic."

Use humor, too. You don't need to tell jokes, but if you're comfortable, try making funny comments throughout the day to lighten the mood

and keep everyone's attention engaged. An ADS can make for a long day, and laughing is the best way to boost people's energy levels.

Be humble–don't come off as a know-it-all. People don't like that. They much prefer people who admit they don't know everything. If you are asked a question you can't answer, just say you don't know but will find out and get back to them quickly. You can research the answer during the lunch break and respond that afternoon. Or find a co-worker who knows the answer and invite them in to answer the question directly and to take follow-up questions. Never try to make up or fudge an answer. You might get caught, and even if you don't, that's just not the right way to treat a customer or a colleague.

Of course, customers aren't the only people who can throw things off. It's not unusual for a member of the account team to talk at length in the hope of looking good in front of the customer, without adding any value to the ADS. If that happens, pull them aside during a break for a friendly word and politely ask them to refrain from talking too much. Tell them you'll make sure to ask them pertinent questions that they can answer during the ADS, so they can get some "face time" with the customer.

Read the room, too. You know who the decision makers are, so keep them focused and engaged. Pay attention to their body: if they go

from looking at you to reading their emails, they're bored. If so, switch to another topic or take a break.

Learn to adjust on the fly. It's rare that I follow an agenda completely; other topics often come up that are much more interesting and useful to the customer. (The challenge is when those topics are unfamiliar to you!) Over time, you will gain the knowledge to be able to pivot and cover just about any topic. It took me about two years of doing ADSs nearly daily to get to that point, so be patient!

Finally, I recommend building up your stamina. When I first started doing full-day ADSs, I found it challenging to keep up my energy—not to mention my voice—for 6 to 8 hours and would collapse when I got home. I talked so much more in ADSs than I was used to, and it took me a while to stop going hoarse! Expect it to take a few months before you can keep up your energy and voice the whole day. My personal favorite tips to keep up stamina and to keep your voice sharp are to get at least eight hours of sleep the day before, stay hydrated during the day by sipping water every few minutes, eat healthy snacks, and exercise regularly during the week.

TIPS FOR CONDUCTING AN ADS REMOTELY

Turn on live captions in your communication software. Not only does this make the session more accessible for everyone, it can also help you understand the conversation and prevent you from having to ask people to repeat themselves.

Sign into the call using two different devices: one for screensharing and whiteboarding, the other to communicate (most communication software supports this). Ideally, your communication device will be a laptop with two or three large monitors. This way, you don't have to constantly minimize windows or move them around, and you can always see the customer as you are whiteboarding and speaking.

I recommend having have a backchannel going with the account team (and *only* the account team) in the chat tool of whatever communication software you are using. This is a great way to communicate with your account team without the customer hearing—for instance, to ask the them to research a question you can't answer, to jump in and talk when needed, or to provide details on who is speaking.

There are also a few tools I recommend. If you are using Microsoft Teams, turn on Speaker Coach, a feature that listens to your audio while you present and provides private real-time feedback and suggestions for improvement, as well as a summary afterward. Only you

can see its live insights; they are not saved in recorded meetings' transcripts.

I also like ZoomIt, a screen-zoom, annotation, and recording tool for technical presentations and application demonstrations. It runs unobtrusively in the tray and activates with customizable hotkeys to let you zoom in on an area of the screen, move around while zoomed, and draw on the zoomed image. It is an excellent way to draw attention to specific portions of your – much better than using your cursor, which can be hard to see.

Summary

An ADS is a vital part of building a data architecture. It helps you align design decisions with business goals, address potential risks and challenges, optimize costs and resources, and foster collaboration among stakeholders.

The end result of an ADS is to build a good data architecture, or even a great one. A good data architecture is robust and scalable and can enable a data architecture that effectively supports data-driven initiatives. Taking good data architecture to *great* data architecture means meeting these needs end-to-end, with feedback from all users in the

data value chain, so the overall data strategy can meet the organization's goals. Building a data solution is a user-focused journey into design and feedback, and it requires the kind of strategy and planning that only an ADS can provide.

Part II. Common Data Architecture Concepts

Before discussing data architectures, it's important to make sure you understand all the data architecture concepts that could be used within an architecture. There are over twenty concepts that I talk about in the upcoming chapters as I find a lot of confusion on these concepts that I hope to clear up, and at the very least it will be a refresher for those that may have not used these concepts in a while. I don't claim that all my definitions of these concepts are universally agreed upon by everyone, but at least this helps to get everyone on the same page to make it easier to discuss architectures.

I have included the relational data warehouse and the data lake under concepts instead of architectures as, while at one time they could have been considered a data architecture when they were the only product used in a solution, now they are almost always combined with other products to form the solution. For example, many years ago there were relational data warehouse products that included the relational storage, compute, ETL software, and reporting software – basically everything you needed bundled together from one vendor. Nowadays, you will stitch together multiple products from possibly

multiple vendors, with each product having a particular focus such as ETL, to complete your data architecture.

Chapter 4. The Relational Data Warehouse

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 4th chapter of the final book.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the author at *jamesserra3@gmail.com*.

By the mid-2000s, I had used relational databases for years, but I had never been exposed to relational data warehouses. I was working as a database administrator (DBA) for a company that used an accounting software package to manage its financial transactions. The reporting from the package was limited and slow, so the company was looking to improve performance, create dashboards to slice and dice

the data, and combine its financial data with data from a homegrown application to get a better understanding of the business.

My employer hired a consulting company to build this thing called a "relational data warehouse"—and, in a decision that changed the course of my career, asked me to help. We generated dashboards that saved the end users so much time and added business insights they'd never had before. When I saw the excitement on their faces, I knew I found my new passion. I changed my career to focus on data warehousing and never looked back.

What Is a Relational Data Warehouse?

A *relational data warehouse* is where you centrally store and manage large volumes of structured data copied from multiple data sources to be used for historical and trend analysis reporting so your company can make better business decisions. It is called *relational* because it is based on the *relational model*, a widely used approach to data representation and organizational for databases. In the relational model, data is organized into tables (also known as relations, hence the name). These tables consist of rows and columns, where each row represents an entity (such as a customer or product), and each column represents an attribute of that entity (like name, price, or quanti-

ty). It is called a *data warehouse* because it collects, stores, and manages massive volumes of structured data from various sources, such as transactional databases, application systems, and external data feeds.

Not all data warehouses are based on the relational model. *Non-relational data warehouses* include types like columnar, NoSQL, and graph data warehouses. However, relational data warehouses are much more popular and widely adopted, primarily because relational databases have been the dominant data management paradigm for decades. The relational model is well-suited for structured data, which is commonly found in business applications. It is also popular due to the widespread use of SQL, which has been the standard language for relational data warehouses for many years.

A relational data warehouse acts as a central repository for many subject areas and contains the *single version of truth* (SVOT). The SVOT is a critical concept in data warehousing that refers to the practice of creating a unified, consistent view of an organization's data. It means that all the data within the data warehouse is stored in a standardized, structured format and represents a single, accurate version of the data. This ensures that all users have access to the same information and eliminates any discrepancies or inconsistencies. It also eliminates data silos, ensuring that all users are working with the same information. This improves decision-making, collaboration, and

efficiency across the organization. It also reduces the risk of errors, inconsistencies, and misunderstandings that can arise from working with disparate, inconsistent data sources.

Imagine you don't have a data warehouse and are generating reports directly from multiple source systems, and maybe even some Excel files. If a report viewer questions the accuracy of the data, wheat can you tell them? The "truth" can be spread out over so many source systems that it's difficult to trace where the data came from. In addition, some reports will give different results for the same data: for example, if two reports use complex logic to pull the data from multiple sources and the logic is updated incorrectly (or not at all). Having all the data in one central location means that the data warehouse is the single source of truth; any questions about the report data can be answered by the data warehouse. Maintaining a SVOT is essential for organizations looking to harness the full potential of their data.

If a data warehouse (DW) is used by the entire company, it's often called an *enterprise data warehouse* (EDW). This is a more comprehensive and robust version of a data warehouse, designed to support the needs of the entire organization. While a standard DW might support a few business units, with many DWs throughout the organization, the EDW uses a wider range of data sources and data types to support *all* business units. The EDW provides a single, unified view of all of the organization's data.

Figure 4-1 illustrates a major reason to have a data warehouse. The diagram on the left shows how challenging it is to run a report using data from multiple applications when you do not have a data warehouse. Each department runs a report that collects data from all the databases associated with each application. So many queries are being run that you're bound to have performance problems and incorrect data. It's a mess. The diagram on the right shows that, with all application data copied into the EDW, it becomes very easy for each department to run a report without compromising performance.

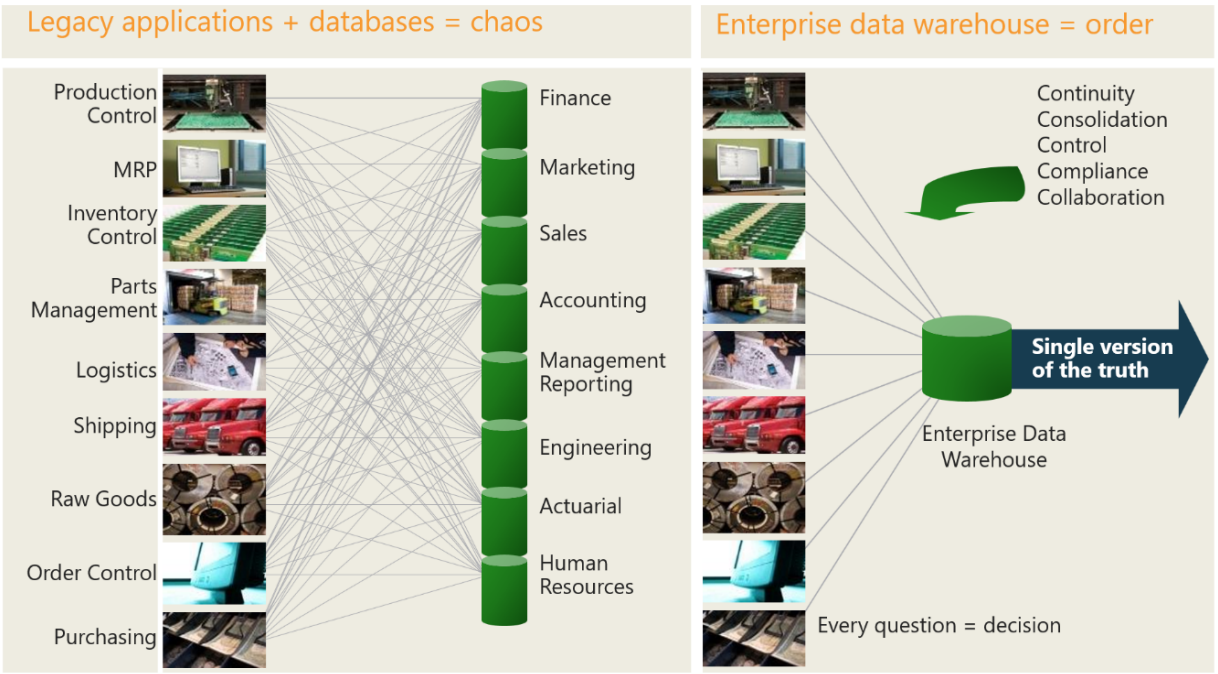


Figure 4-1. Before and after an enterprise data warehouse

Typically, to build a data warehouse, you will create data pipelines that perform three steps, called extract, transform, and load (ETL).

1. The pipeline extracts data from the source systems, such as databases and flat files.
2. The extracted data is then transformed or manipulated to fit the target systems' requirements (in this case, to fit a data warehouse). This can involve cleaning, filtering, aggregating, or combining data from multiple sources.
3. The transformed data is loaded into the data warehouse. A DBA can make the database and field names more meaningful, to make it easier and faster for end-users to create reports.

What a Data Warehouse Is Not

Now that you know what a data warehouse is, let's clarify it more by looking at solutions that should *not* be considered a data warehouse (though I have seen people do so many times):

DW Prefix

A data warehouse is not a just copy of a source database from an operational system with "DW" added to the filename. For example, say you were to copy a database called "Finance" containing 50 operational tables and call the copy DW_Finance, then use those 50 tables to build your reports. This would result in a data warehouse designed for *operational* data when instead you want to design it for *analytical* data. With analytical data, you have better read performance and can create data

models to make it easier for end-users to build reports. (I'll explain more in the next section.)

Views with Unions

A data warehouse is not a copy of multiple tables from various source systems unioned together in a SQL view. (*Unioning* is done via the SQL UNION statement, which combines the results of two or more SELECT statements into a single result set.) For example, if you copied data from three source systems that each contain customers, you'd end up with three tables in the data warehouse called CustomerSource1, CustomerSource2, and CustomerSource3. So you'd need to create a view called CustomerView that is a SELECT statement unioning the tables CustomerSource1, CustomerSource2, and CustomerSource3. You'd repeat this process for other tables, such as products and orders.

Instead, the data from the three tables should be copied into one table in the data warehouse, which required the extra work of creating a data model that fits all three tables. You would likely want to use Master Data Management (MDM, explained in chapter 6) at this point, to prevent duplicates and improve accessibility and performance.

Dumping Ground

> A data warehouse is not a dumping ground for tables. Many times, this arises when a company does not have a DW and an end-user wants to create a report from a subset of data from a couple of source systems. To help them out quickly, a person from IT creates a DW without much thought, copying the data from those two source systems into the DW. Then other end-users see the benefit the first end-user got, and they want additional data from those same source systems and a few others to create their own reports. So once again the IT person quickly copies the requested data into the DW. This process repeats over and over until the DW becomes a jumbled mess of databases and tables.

So many DWs start out as a one-off solution for a couple of users, then morph into a full-blown but poorly designed DWs for the entire company. There is a better way.

Instead, when that first end-user request comes in, assess your company's reporting needs. Find out if the request is really a one-off, or if it should be the start of building an EDW. If it should, this is your chance to show senior leaders why your company needs a DW. If so, be adamant that you need enough up-front time to design a DW that can support many data sources and end-users. (You can use the next chapter in this book to support your case.)

The Top-Down Approach

In a relational data warehouse, you will do a lot of work up front to get the data to where you can use it to create reports. Doing all this work beforehand is a design and implementation methodology referred to as a *top-down approach*. This approach works well for historical-type reporting, in which you're trying to determine what happened (*descriptive analytics*) and why it happened (*diagnostic analytics*). In the top-down approach, you establish the overall planning, design, and architecture of the data warehouse first, then develop specific components. This method emphasizes the importance of defining an enterprise-wide vision and understanding the organization's strategic goals and information requirements before diving into the development of the data warehouse.

Descriptive analytics and diagnostic analytics are two important types of data analysis that are commonly used in business. *Descriptive analytics* involves analyzing data to describe past or current events, often through the use of summary statistics or data visualizations. This type of analysis is used to understand what has happened in the past, and to identify patterns or trends in the data that can help with decision-making.

Diagnostic analytics is used to investigate the causes of past events, typically by examining relationships between different variables or factors. This type of analysis can identify the root causes of problems or diagnose issues that may be affecting business performance.

Suppose a company wants to analyze sales data from the past year. Descriptive analytics would involve calculating summary statistics such as total sales revenue, average sales per day, and sales by product category, to understand what happened during the past year. Diagnostic analytics, by contrast, would involve examining relationships between factors (such as sales and marketing spend, or seasonality and customer demographics) to understand why sales fluctuated throughout the year. By combining both approaches, companies can gain a deeper understanding of their data and make more informed decisions.

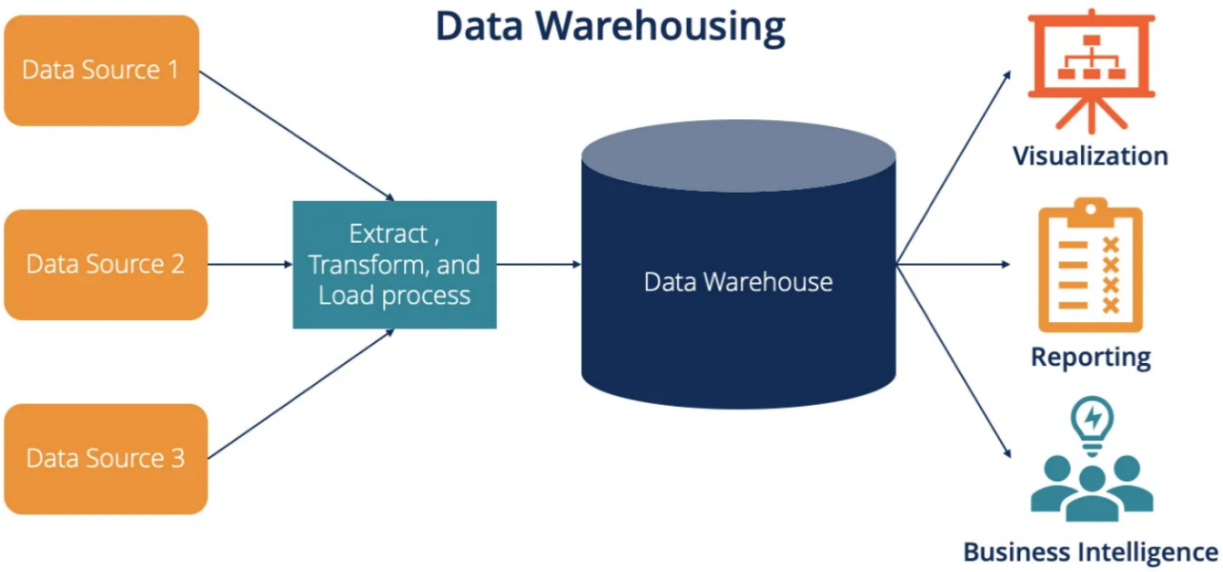


Figure 4-2. The architecture of a data warehouse.

The top-down approach typically involves the following steps:

1. Start with a clear understanding of corporate strategy, and make some theories and hypotheses up front. Make sure you know what questions you want to ask of the data.
2. Define the business requirements: Identify the organization's goals, objectives, and key performance indicators (KPIs). Gather and analyze the information needs of various departments and users. You can also think of this as your reporting requirements
3. Design the data warehouse architecture: Based on the business requirements, create a high-level architecture for the data warehouse, including its structure, data models, and data integration processes. This will be your technical requirements.

4. Develop the data model: Design a detailed data model for the data warehouse, taking into account the relationships between various data entities and the granularity of the data.
5. Build the appropriate databases, schema's, tables, and fields in the data warehouse. This is the previously defined approach called schema-on-write.
6. Data extraction, transformation, and loading (ETL): Develop the ETL processes to extract data from various source systems, transform it into the desired format, and load it into the data warehouse.
7. Develop and deploy BI tools and applications: Implement business intelligence (BI) tools and applications that allow users to access, analyze, and report on the data stored in the data warehouse.
8. Test and refine the data warehouse: Perform testing to ensure data quality, performance, and reliability. Make any necessary adjustments to optimize the system.
9. Maintain and expand the data warehouse: As the organization's needs evolve, update and expand the data warehouse accordingly.

The top-down approach has some advantages, such as a comprehensive view of the organization's data needs, better data consistency, and improved governance. However, it can also be time-consuming and resource-intensive, with a longer time to deliver value com-

pared to a bottom-up approach used by the data lake that will be described later (chapter 5.3), along with how the modern data warehouse architecture combines both approaches (chapter 10).

Why Use a Relational Data Warehouse?

Having a DW makes it so much easier to build any kind of BI solution, since those solutions can pull data just from the DW and will not have to create complex logic to pull data from multiple source systems. Also, they won't have to clean or join the data because that will have already been done by the DW. The BI solution that is built from the DW could be a data mart (which contains a subset of the DW data for a specific group of people), could aggregate data to make queries and reports faster, and could even be used within Microsoft Excel. The bottom line is that with a DW, you already have a solid foundation to build upon.

Let's look in detail at some of the major benefits you can get from using a relational data warehouse:

Reduce stress on the production system

You may have seen this problem before: you get an angry call from an end-user complaining they are trying to insert orders

via the order-entry application, and it is taking forever. You look into it, and it turns out another end-user is running a report via the order-entry application that is hogging all the resources on the server where the order-entry application resides. This is especially prevalent when end-users are allowed to create ad-hoc queries and they come up with very poorly written SQL. By copying the order-entry application database to a DW and optimizing it, you can then have all reports and ad-hoc queries go against the DW and avoid this problem altogether, especially if the end-user needs to run a report that goes against multiple application databases.

Optimize for read access

Application databases are going to be optimized to support equally all the CRUD operations, so the reading of data will not be as fast as it could be. The data warehouse, on the other hand, is a write-once, read-many type of system, meaning it will mainly be used for the reading of data. Therefore, it can be optimized for read access, especially time-consuming sequential disk scans that frequently occur when reports or queries are run. There are many database techniques that can be used to speed up read access in a DW, some to the detriment of write access, which we are not concerned about.

Integrate multiple sources of data

The ability to integrate many sources of data in order to create more useful reports is one of the bigger reasons to build a DW. Having one spot where all the data is located instead of being spread out over various databases not only makes report building easier but greatly improves the performance of reports.

Run accurate historical reports

Without a DW, end-users of applications usually run all their reports on a particular day each month (usually the last day of the month) and save them to disk, so they have hardcopies that they could refer to if they needed to see the report results for a time in the past. This was required, for example, if a customer recently moved to another state and you wanted to look back at a report from a few months ago that listed customer sales-by-state. If you ran a current report, the report would incorrectly show that person's sales in their new state instead of their old state (since their record in the database was updated to show their new state). Hence this requires looking back at a hardcopy report instead of running a current report.

A DW can take care of this by keeping track when a customer moves states (via tracking customer history of the old and new states and start and end dates) as well as any other fields that need to be tracked (for example, employer or income), so that

running a report from a past month would be accurate and saving reports to disk each month would no longer be required.

Restructure and rename tables

Many times, the application databases have table and fields names that are very difficult to understand, especially for older ERP and CRM products (think tables names such as `T116` and field names like `RAP16`). In the data warehouse, you can copy the data from those source tables into something much easier to understand (for example, `Customer` instead of `T116`). You can also likely come up with a better data model for all the tables. This will make it much easier for end-users to create reports, as they don't have to translate cryptic table and field names.

Protection against application upgrades

Imagine you don't have a DW and instead create reports against an application database. Everything is running fine and then all of a sudden, many reports start giving errors. It turns out the application went through an upgrade, installing a new version that renamed a bunch of tables and fields. So now you must go through each and every report, out of hundreds, and rename the changed tables and fields. That could take months,

resulting in a lot of upset end-users. Even after that, any reports you missed might still give errors.

A DW can protect you against this. After an application upgrade, only the ETL that copies data from the application databases to the DW needs to be updated, a quick task. The reports do not have to be changed. End-users won't see any new data until the ETL is fixed, but their reports don't give errors.

Reduced security concerns

Without a DW, your team would need to give each end-user security access to each application database they needed to use for reporting purposes. There could be dozens; the process could take weeks, and sometimes they still might not be allowed access. With a DW, each end-user would only need access to the appropriate tables, which would be much faster and easier.

Keep historical data

Many production systems limit the amount of historical data they keep (for example, the last three years). They do this to save space and improve performance, and in some cases for regulatory compliance. They usually purge older data yearly or monthly. A DW can hold all of the history, so you never have to

worry about running a report for older years and not finding any data.

Master Data Management (MDM)

As you collect data from multiple source systems, many times you will need to use MDM to remove duplicate records for such things as customers, products, assets, etc. (see chapter 6 for a larger explanation on MDM). The DW is the perfect place to perform MDM. Also, many of the MDM tools allow you to create hierarchies (for example, Company –> Department –> Employee), adding more value to mastering the data.

Improve data quality by plugging holes in source systems

You will find that a lot of the data you get from the various source systems needs to be cleaned, despite what the owners of the applications say (many times I have heard them say "our data is clean", only to be proved wrong every time). For example, an order entry application may require a customer birth date, and if the person entering the data does not know the customers birth date, they might enter a date in the future or a data more than 100 years old just so the order can be entered. Or maybe the application does not check the accuracy of the two digits entered for a state code. There are always dozens of "holes" in the source system. The DW is where you would

clean the data but also notify the people who maintain the applications of the holes in their system so they can fix them to prevent bad data being entered in the future.

No IT involvement needed to create reports

This goes back to the previous chapter on self-service BI – building a proper DW will remove the need to get IT involved with building reports and leave it up to the end-user. Without IT as the bottleneck, reports and dashboards can be built sooner. And IT will be thankful they can work on more interesting projects than creating reports!

Drawbacks to Using a Relational Data Warehouse

There are always tradeoffs, and below are the drawbacks to consider when building a relational data warehouse:

Complexity

Data warehouses can be complex and time-consuming to design, build, and maintain. This can require specialized skills and resources, which can increase costs.

High costs

Implementing a data warehouse can be expensive, requiring significant investments in hardware, software, and personnel. Ongoing maintenance and upgrades can also add to the cost.

Data integration challenges

Integrating data from various sources can be challenging, as it may involve dealing with different formats, structures, and data quality issues. This can result in additional time and effort being spent on data cleaning and preprocessing. In addition, certain data, such as streaming data from IoT devices, is too challenging to ingest into a relational data warehouse so becomes a lost opportunity for more insights

Time-consuming data transformation

To load data into a data warehouse, it may need to be transformed to conform to the warehouse's data model. This process can be time-consuming, and errors in data transformation can lead to inaccurate analysis.

Data latency

Because data warehouses are designed to handle large volumes of data, they can be slower to process than other types of databases. This can result in data latency, where the data in the warehouse is not up to date with the most recent changes.

Maintenance window

Usually, with a relational data warehouse, you need a maintenance window. Loading and cleaning data is very resource-intensive, and users trying to run reports at the same time will experience very slow performance. So users must be locked out of the warehouse while this maintenance is going on, which prevents 24/7 access. If any problems occur during the maintenance window, such as a failed ETL job, you may have to extend the maintenance window. If users try to run reports and are still locked out, you'll have upset users who can't perform their jobs.

Limited flexibility

Data warehouses are designed to support specific types of analysis, which can limit their flexibility for other types of data processing or analysis. This can require additional tools or systems to be integrated with the warehouse to meet specific needs.

Security and privacy concerns

Storing large amounts of sensitive data in a centralized location can increase the risk of data breaches and privacy violations, necessitating strong security measures.

Populating a Data Warehouse

Because the source tables that are fed into a data warehouse change over time, the data warehouse needs to reflect those changes. This sounds simple enough, but there are many decisions to make: how often to *extract* (or pull) the data, what extract method to use, how to physically extract the data, and how to determine the data that has changed since the last extraction. I'll discuss each of these briefly.

How Often to Extract the Data

How often you need to update the DW largely depends on how often the source systems are updated, and how timely the end-user needs the reporting to be. Often end-users don't want to see data for the current day, preferring all data since the prior end-of-day. In this case, you can run your jobs to extract the data from the source systems via ETL tools each night after the source system databases are done being updated, creating a nightly maintenance window to do all the data transfer. If the end-users require updates during the day, then a more frequent extract, say hourly, will be required.

One thing to consider is the size of the data for each extract. If it is very large, updating the DW may take too long, so you might want to split it up into smaller chunks and do more frequent extracts and updates (for example, hourly instead of daily). Also, it may take too long

to transfer large amounts of data from the source systems to the data warehouse, especially if the source data is on-prem and you don't have a large pipeline from the source system to the internet. This is another reason why you may want to go from a large nightly transfer to hourly during the day.

What Extraction Method to Use

There are two methods for extracting data from source systems. Let's look at each one.

Full extraction

In a full extraction, all the data is extracted completely from one or more tables in the source system. This works best for smaller tables. Because this extraction reflects all the data currently available on the source system, there is no need to keep track of changes, making this method very easy to build. The source data is provided as-is and you don't need any additional information (for example, timestamps).

Incremental extraction

In incremental extraction, you're pulling only the data that has changed since a specified time (such as the last extraction or the end of a fiscal period), instead of the whole table. This

works best for large tables, and only when it's possible to identify all the changed information (discussed below).

With most source systems, you'll use a combination of these two methods.

How to Extract the Data

Whether you're doing a full or an incremental extraction, there are two ways to extract the data: online and offline.

In online extraction, the extraction process can connect directly to the source system to access the source tables, or to an intermediate system that stores the data in a preconfigured manner (for example, transaction logs or change tables).

However, direct access to the source system is not always available. In such cases, the data is staged outside the original source system and is created by an extraction routine originating from the source system (for example, a mainframe performing an extract routine on a table and landing it in a folder in a file system). The extracted data is usually in a flat file that is in a defined, generic format (for example, CSV or JSON).

How to Determine What Data Has Changed Since the Last Extraction

Unfortunately, for many source systems, it can be difficult to identify the recently modified data, and therefore it can be challenging to do an incremental extract. Below are several techniques for identifying recently modified data and implementing incremental extraction from the source systems that can work in conjunction with the data extraction techniques just discussed. Some techniques are based upon the characteristics of the source systems; others may require modifications to the source systems. The source system's owners should carefully evaluate any technique prior to implementation.

Timestamps

Timestamps are the most preferable option and the easiest to implement. The tables in some operational systems have timestamp columns with the time and date that a given row was last modified, making it easy to identify the latest data. In relational databases, many times this column is given the data type `timestamp` or `datetime`, along with a column name like `Timestamp` or `Last Modified`. The source application will then populate this column. You can set up the relational database to default to the current date when the record is saved, or add database triggers to populate the column.

Change Data Capture

Most relational databases support *Change Data Capture* (CDC), which records the INSERTs, UPDATEs, and DELETEs applied to database tables, and makes a table record available of what changed, where, and when based on the relational database's transaction log. If you need near-real-time data warehousing, where you can process changes to the source system and have them reflected in the data warehouse in a few seconds, CDC can be the key enabling technology.

Partitioning

Some source systems use range partitioning, in which the source tables are partitioned along a date key, which makes it easy to identify new data. For example, if you are extracting from an orders table partitioned by day, it is easy to identify the current or previous day's data.

Database Triggers

You can add a trigger for INSERT, UPDATE, and DELETE on a single table and have those triggers write the information about the record change to a 'change table'. This is similar to Change Data Capture, so use CDC if you are using a database product that supports it; otherwise, use triggers.

MERGE Statement

The least preferable option is to do a full extraction from the source system to a staging area in the DW, then compare this table with a previous full extract from the source system using a MERGE statement to identify the changed data. You will need to compare all source fields with all destination fields (or use a hash function). This approach likely won't have a significant impact on the source system, but it can place a considerable burden on the DW, particularly if the data volumes are large. This option is usually the last resort if no other options are possible.

The Death of the Relational Data Warehouse Has Been Greatly Exaggerated

Around the early 2010s, people in the IT industry started questioning whether the relational data warehouse was needed anymore, asking, "Is the relational data warehouse dead?" Many people understood this as asking if businesses still need DWs– they do, as this chapter points out. But the question is really about the data warehouse architecture: can you just use a data lake, or should you use both a data lake and a relational data warehouse?

When data lakes first appeared, they were built on Apache Hadoop technology, and it was largely Hadoop vendors pronouncing that the relational data warehouse dead. Just put all your data in the data lake and get rid of your relational data warehouse, they advised. As mentioned in chapter 2, the projects that attempted to do that all failed.

For many years I felt that relational data warehouses would always be needed, because data lakes were all Hadoop-based and there were just too many limitations. But once solutions like delta lake (see chapter 12) became available, and data lakes began using better, easier-to-use products than Hadoop (see appendix 2.1), I started to see some use cases where a solution can work without a relational data warehouse. That type of solution is a data lakehouse architecture that will be covered in chapter 12.

However, there are still plenty of use cases where a relational data warehouse is needed. And while data lake technologies will continue to improve and reduce or eliminate the concerns of bypassing a relational data warehouse (see chapter 12.2), we will never see the end of relational data warehouses completely. I think there are three reasons for this. First, it's still harder to do reporting off a data lake than from a DW. Second, relational DWs continue to meet the information needs of users and provide value. Third, many people use, depend on, and trust DWs and don't want to replace them with data lakes.

Data lakes offer a rich source of data for data scientists and self-service data consumers (“power users”) and serve the needs of analytics and big data well. But not all data and information workers want to become power users. The majority continue to need well-integrated, systematically cleansed, easy-to-access relational data that includes a historical log that captures how things have evolved or progressed over a period of time. These people are best served with a data warehouse.

Summary

This chapter covered the first widely used technology solution to centralize data from multiple sources and report off of it: the relational data warehouse. The relational data warehouse revolutionized the way businesses and organizations managed their data by providing a centralized repository for data storage and retrieval, enabling more efficient data management and analysis. With the ability to store and organize data in a structured manner, relational data warehouses allow users to generate complex queries and reports to be generated quickly and easily, providing valuable insights and helping with critical decision-making.

Today, the relational data warehouse remains a fundamental component of many data architectures and can be seen in a wide range of

industries, from finance and healthcare to retail and manufacturing. The following chapter discusses the next technology to become a major factor in centralizing and reporting on data: the data lake.

Chapter 5. Data Lake

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 5th chapter of the final book.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the author at *jamesserra3@gmail.com*.

Big data started appearing in unprecedented volumes in the early 2010s, due to the increase in sources such as sensors, videos, and social media that output semi-structured and unstructured data. Semi-structured and unstructured data hold a phenomenal amount of value – think of the insights contained in years' worth of customer emails! However, relational data warehouses at that time could only handle structured data. They also had trouble handling large amounts of data or data that needed to be ingested often, so they were not an

option for storing these types of data. This forced the industry to come up with a solution: data lakes. Data lakes can easily handle semi-structured and unstructured data as well as managing data that is ingested often.

Years ago, I spoke with analysts from a large retail chain who wanted to ingest data from Twitter to see what customers thought about their stores. They knew customers would hesitate to bring up complaints to store employees but would be quick to put them on Twitter. I helped them to ingest the Twitter data into a data lake and assess the sentiment of the customer comments, categorizing them as positive, neutral, or negative. When they read the negative comments, they found an unusually large number of complaints about dressing rooms being too small, too crowded, and not private enough. As an experiment, they decided to remodel the dressing rooms in one store. A month after the remodel, they found an overwhelming number of positive comments about the dressing rooms in that store, along with a 7% increase in sales. That prompted them to remodel the dressing rooms in all of their stores, resulting in a 6% increase in sales nationwide and millions more in profit. All thanks to a data lake!

What is a data lake?

The term *data lake* is a metaphor to describe the concept of storing vast amounts of raw data in its natural format. Just as a lake holds water without changing its nature, a data lake holds data without the need to structure or process it first. Also, just as lakes hold various wildlife and plant species, a data lake holds different types of data (structured, semi-structured, and unstructured). Contrast that with the data warehouse, where data is more structured and processed, like bottled or packaged goods in a warehouse.

Once the data is in a data lake, it must be cleaned, joined, and possibly aggregated in order to make it useful. This is where some type of *compute* (that is, the processing power required to manage, manipulate, and analyze data) must connect to the data lake, transform the data, then put it back into the data lake.

Why use a data lake?

There are many reasons for using a data lake, especially alongside a relational data warehouse (DW). As mentioned before, you can quickly store data in a data lake with no up-front work needed, called "schema-on-read" (see chapter 2.4.2). This will allow you to access the data much faster. That can help by allowing power users to run reports quickly (for faster ROI) and giving data scientists quick access to data to train a machine learning model. It's also useful for in-

vestigating data. If an end user asks you to copy source data into a DW, you can quickly copy the source data to the data lake and investigate it to make sure it has value before you put forth the effort of creating the schema in the data warehouse and writing the ETL.

I mentioned in chapter 4 that typically, with a DW, you have a maintenance window at night where you kick the end users off and then load the source data into DW staging tables. You then clean the data and copy it into the DW production tables. The problem is that if you run into problems and maintenance takes longer than expected, so when end users try to access the data in the morning, they're locked out. A better option is to transform the data in the data lake instead of in the DW. The benefits of this include:

- Saving money since the DW compute is usually much more expensive than other types of compute that you can use on the data in the data lake
- Allowing for extreme performance for transformations if needed by having multiple compute options where each compute can be accessing different folders containing data and running in parallel (or running in parallel on the same data)
- Because of the flexibility to use many different types of compute options on the data in the data lake, you can refine the data in many more ways than you could with the DW. For example, using a compute that has code libraries with pre-built routines to do

complex transformations that are very difficult or impossible to do using SQL in the DW
- No DW maintenance window needed, allowing the DW to be used 24/7 for queries, and avoiding users competing for the same compute resources if you try and run reports while data transformations are going on, slowing each other down (since data transformations are large resource hogs)

A data lake is an inexpensive way to store an unlimited amount of data. Unlike on-prem storage, cloud providers have unlimited storage, so you never have to worry about running out of storage with your data lake. Also, cloud storage is relatively very cheap, and most cloud providers have multiple storage tiers to save even more money.

The data lake allows you to collect any data "just in case" you might need it in the future (call it data stockpiling). Data stockpiling is rarely done with a DW since storage in a DW is usually much more expensive than a data lake, and having more data in the DW can affect performance. And because storage is so cheap in a data lake, data is rarely deleted unless for regulatory reasons.

Because it acts as a centralized place for all subjects of data, a data lake can be the "single version of the truth." If any and all data lands in the data lake before it is copied elsewhere, the data lake can be the place to go back to if there are any questions about the accuracy

of the data in all the other places it is used, such as queries, reports, and dashboards. A data lake also makes it easy for any end-user to access any data from any location and use it many times, for any analytic needs and use cases they wish. When you use a data lake alongside a DW, the data lake becomes the single version of the truth instead of the DW. This is because more data will be in the data lake than in the DW, since it is much cheaper and there is unlimited storage.

Furthermore, a data lake is a place to land streaming data, such as from IoT devices, which is nearly impossible to land in a DW.

The data lake also acts as an online archive. For example, you could keep the last three years of order data in the DW and order data older than three years in the data lake. Most of the technology used to build a DW will allow you to query data in a data lake using SQL, so users can still use it in reports (with the trade-off of slower performance). This can save costs and avoid running out of storage space in the DW.

The data lake is also a place where you can backup data from the data warehouse in case you need to restore data to the DW due to accidental deletion, corruption, an incorrect WHERE clause on an UPDATE query, etc.

In addition, you can integrate differently structured data into a data lake: everything from CSV and JSON files to Word documents and media files. Data can be extracted from those files to give you more value.

With a DW, the raw data copied from a source system into staging tables in the DW and transformed and then loaded into the production tables is usually deleted after a day or two to save space in the limited relational storage. A problem then arises if an ETL error is discovered from a ETL run many days ago and needs to be rerun, but the raw data has been deleted, necessitating having to go back to the source system and possibly affecting its performance. By doing the transforming in the data lake, where storage is cheaper, you can keep a long history of the raw data and never have to go back to the source system.

Bottoms-up approach

Because a data lake is schema-on-read, little upfront work needs to be done to start using the data. This works really well if you don't know what questions to ask of the data yet – you can quickly explore the data to find those relevant questions. This results in what is referred to as a *bottom-up approach*, as seen in figure 5-2, where you collect data up front before generating any theories or hypotheses.

This is very different from the top-down approach of a relational data warehouse, as seen in figure 5-1.

Figure 5-1. Top-down approach

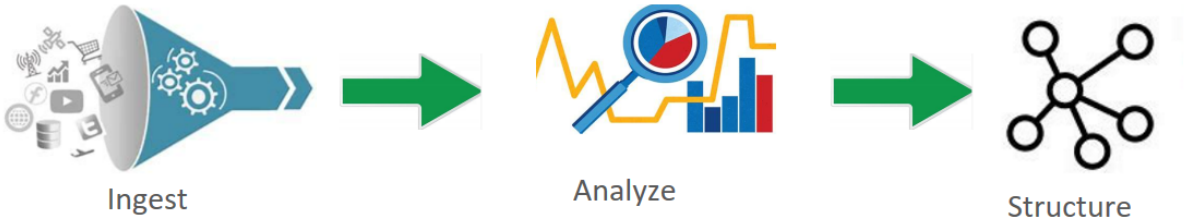


Figure 5-2. Bottom-up approach

This allows data scientists, who typically use software that prefers data in files, to use machine learning on the data to determine what will happen (*predictive analytics*) and how can we make it happen (*prescriptive analytics*).

Predictive analytics is a type of data analytics that makes use of data, statistical algorithms, and machine learning techniques to predict future outcomes based on historical data. The goal is to go beyond what has happened to provide a best assessment of what will happen in the future. It includes a variety of statistical techniques, such as

data mining, predictive modeling, and machine learning, to analyze current and historical facts to make predictions about future or otherwise unknown events. This allows you to be proactive, not just reactive. For instance, predictive analytics can be used in healthcare to forecast patient readmission rates, in retail to predict future sales, or in banking to predict loan defaults.

Prescriptive analytics goes a step further than predictive analytics. It utilizes optimization and simulation algorithms to advise on possible outcomes. Prescriptive analytics not only predicts what will happen in the future but also suggests actions to take to affect those outcomes. The goal is to provide advice based on predicted future scenarios to optimize decision-making. Prescriptive analytics can suggest decision options on how to take advantage of a future opportunity or mitigate a future risk, and illustrate the implications of each decision option. For example, in the logistics industry, prescriptive analytics can be used to find the best routes for delivery and even suggest alternate routes in case of unexpected road closures.

Predictive and prescriptive analytics were mainly how data lakes were used at first, which overcame the difficulty of trying to do advanced analytics with the traditional data warehouse. But now data lakes are used for much more, as indicated in the previous section.

If you find valuable data in the data lake when exploring and you want to make it easily accessible to end-users, you can always model it later by copying it to a relational data warehouse. *Data modeling* is like creating a blueprint for organizing and understanding data. It helps define what data is important, how it's related, and how it should be stored and used in a relational data warehouse. Data lakes lack tools for data modeling, while DWs have had modeling tools for years.

Best practices for data lake design

Designing a data lake should be a time-consuming process. I often find that companies don't spend enough time thinking through all their use cases when designing their data lake, and later have to redesign and rebuild it. So, make sure you think through all the sources of data you will use now and in the future, understanding the size, type, and speed of the data. Then absorb all the information you can find on data lake design and choose the appropriate design for your situation.

A data lake generally doesn't enforce a specific structure on the data when it's ingested, which is one of its key characteristics. This is different from a traditional database or data warehouse, which requires data to be structured or modeled beforehand. However, to make data usable and prevent the data lake from becoming a "data swamp" (an

unorganized and unmanageable collection of data), it's important to apply some organization and governance practices. This chapter introduces some best practices to get you started.

The first best practice is you should logically divide the data lake into multiple layers (also called zones) corresponding to increasing levels of data quality, as shown in figure 5-3. Otherwise, you will have all your files in one folder, which will greatly affect performance, manageability, and security.

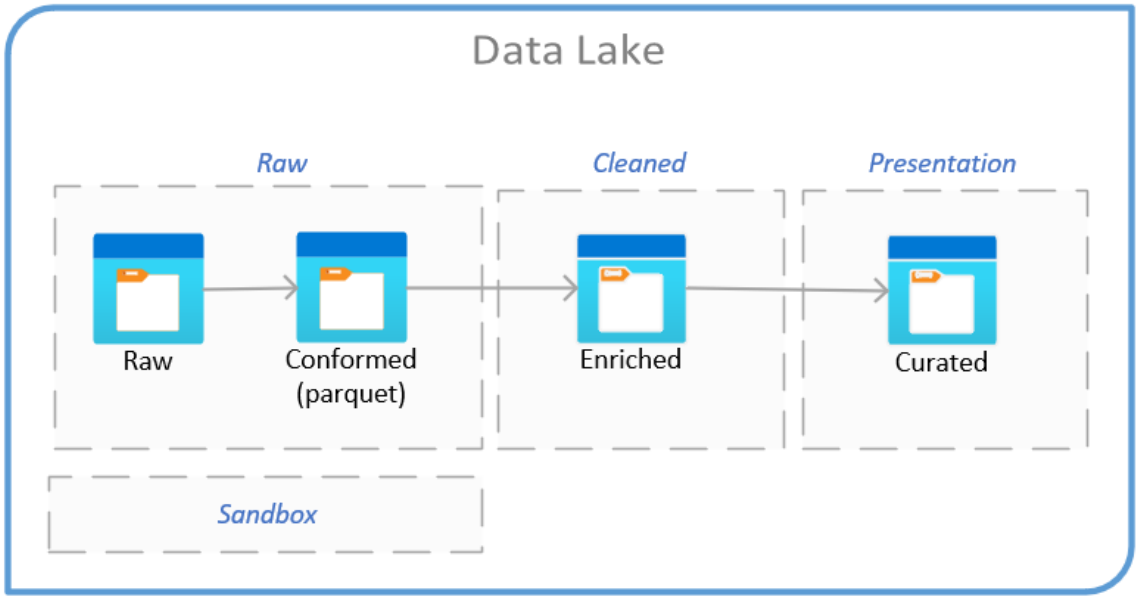


Figure 5-3. Data lake layers

Raw data layer

Raw events are stored for historical reference, usually kept forever (immutable). Think of the raw layer as a reservoir that stores data in its natural and original state (no transformations applied). It's unfiltered and unpurified. This layer is also called the *bronze layer*, *staging layer*, or *landing area.*

Conformed layer

Many times the data in the raw layer will be stored as different types, such as CSV, JSON, and Parquet (a common storage file format optimized for efficient big data processing). The conformed layer is where all the file types are converted to one format, usually Parquet. Think of it as having two reservoirs, one of salt water and one of fresh water, and using desalination to turn the salt water into fresh water. This layer is also called the *base layer* or *standardized layer.*

Cleansed data layer

Raw events are transformed (data is cleaned, integrated, and consolidated) into directly consumable data sets. Think of the cleansed layer as a filtration layer. It removes impurities and can also involve enrichment. The aim is to uniform the way files are stored in terms of encoding, format, data types and content (such as strings and integers). This layer is also called the *sil-*

ver, transformed, refined, integrated, processed, or *enriched layer.*

Presentation data layer

Business logic is applied to the cleansed data to produce data ready to be consumed by end-users or applications, typically in a format that's easy to understand and use. It might involve aggregations, summaries, or files that have been transformed into a specific layout for use in reporting tools, often including information about the data (metadata) within each file. This layer is also called the *application, workspace, trusted, gold, secure, production ready, governed, curated, serving, analytics,* or *consumption layer.*

Sandbox data layer

Optional layer to be used to "play" in, usually for data scientists. It is usually a copy of the raw layer. This layer is also called the *exploration layer, development layer,* or *data science workspace.*

This is just one way of setting up the layers in your data lake, and I have seen many other excellent designs, often with additional layers.

For an example using the layers in figure 5-2, let's consider a retail company that has different data sources like point-of-sale (POS) sys-

tems, online sales, customer feedback, and inventory systems. The four layers would contain:

Raw layer

- Raw logs from the POS system, including every transaction detail such as timestamp, items purchased, quantities, prices, total amount, cashier ID, store location, etc.
- Online sales data from the website or app, such as user ID, items purchased, quantities, prices, total amount, timestamps, etc.
- Inventory data from various warehouses and stores, including items, quantities, restock dates, etc.
- Raw customer feedback collected through surveys, reviews, and ratings.

Conformed layer

- POS and online sales data are transformed into a common schema (for example, common naming and formats for items, common time format, common store ID system, etc.).
- Inventory data is transformed to align with the common item names and store IDs used in the sales data.
- Customer feedback data is transformed into a common format, aligning with the common store ID system, and per-

haps extracting common elements from the feedback for analysis.

Cleaned layer

- POS and online sales data with any errors removed or corrected (such as inconsistencies in how items are named, errors in quantities, missing timestamps, etc.).
- Inventory data with standardization in item naming, and errors or inconsistencies corrected.
- Customer feedback data is cleaned to remove irrelevant or erroneous responses, standardizing formats, etc.

Presentation layer

- A consolidated sales report showing total sales per store, per region, or per day/month/year. It might also break down sales by item or category.
- An inventory report showing current stock levels for each item in each store or warehouse, as well as restocking schedules.
- A customer feedback report summarizing the feedback, maybe with a sentiment analysis score for each store or product.

Another best practice is that within each layer, there should be a folder structure, usually different for each layer, that can be divided up in many different ways for different reasons. Here are some examples:

Data Segregation

Organizing data based on source, business unit, or data type makes it easier for data scientists and analysts to locate and use the relevant data.

Access Control

Different teams or individuals within an organization may have different levels of access to data. By structuring folders based on user roles or departments, organizations can implement fine-grained access control policies.

Performance Optimization

Organizing data in a specific way can lead to improved performance. For instance, certain data processing or querying operations may be faster if the data is grouped based on specific characteristics.

Data Lifecycle Management

Data often has a lifecycle, from ingestion to archival or deletion. Different folders might be used to segregate data based on its

stage in the lifecycle.

Metadata Management

Folders can be used to manage and segregate metadata from the raw data. This segregation can simplify metadata management and speed up data discovery.

Compliance Requirements

In many industries, compliance requirements dictate that certain types of data be stored and managed in specific ways. Different folder structures can help organizations meet these requirements.

Backup and Disaster Recovery

Having different folder structures can assist in creating a strategic backup and disaster recovery plan. Certain folders might be backed up more frequently or retained for longer based on their importance.

Data Versioning

Different folders might be used to manage different versions of the same dataset.

Data Partitioning

Data can be partitioned by key attributes for quicker query performance.

Ingestion and Processing Needs

Based on the source, data might need different processing pipelines. Different folders can help manage and streamline these processes.

To satisfy one or more of the reasons above, you may divide the folders up in one or more of the ways below:

- Time partitioning (year/month/day/hour/minute)
- Subject area
- Security boundaries (such as department or business area)
- Downstream app or purpose
- Data retention policy (temporary and permanent data, and applicable period or project lifetime)
- Business impact or criticality (high, medium, low)
- Owner/Steward/SME
- Probability of data access (for recent or current data versus historical data)
- Confidential classification (such as public information, internal use only, supplier/partner confidential, personally identifiable information, or sensitive)

Figure 5-4 shows an example folder structure for the raw and cleaned zones:

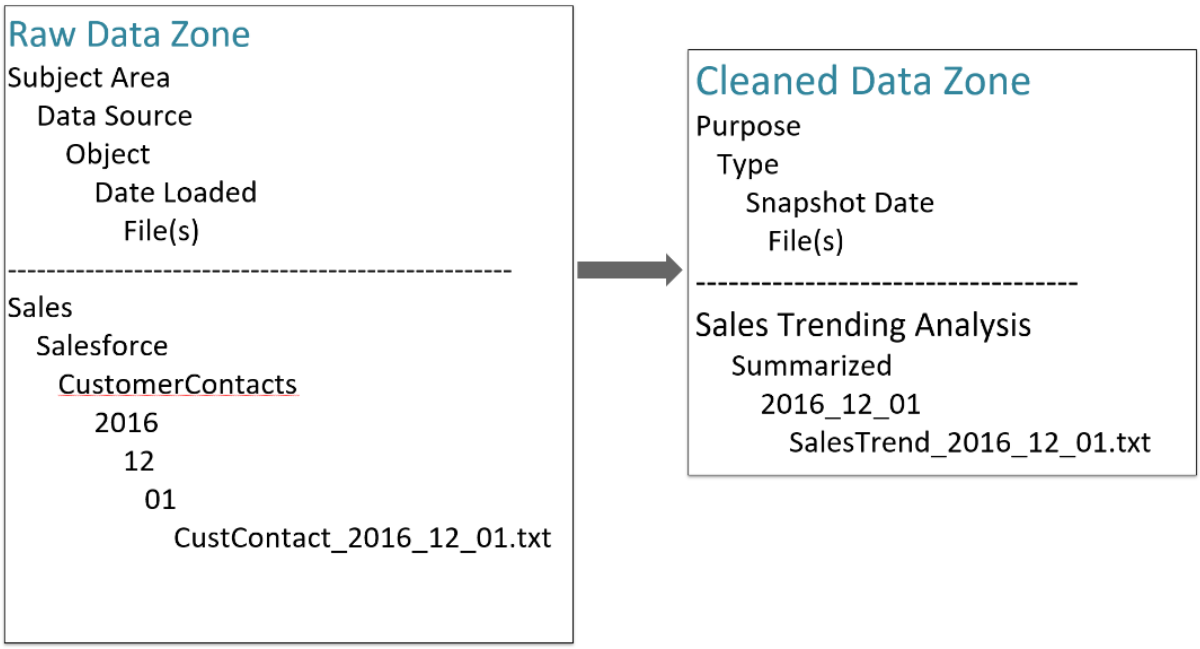


Figure 5-4. Folder structure in data lake zones

Most times all these layers are under one cloud subscription. There are some exceptions, such as if you have specific requirements for billing, if you'll hit some subscription limit, or if you want separate subscriptions for development, testing, and production.

Most customers create a storage account for each layer (so three layers means three storage accounts), all within a single *resource group* (a container that holds related resources for a solution). This isolates the layers to help make performance more predictable and allows for different features and functionality at the storage account level.

Typical features set at each storage account level include lifecycle management, which allows you to reduce costs by automating data management tasks, (like transitioning data across different storage tiers or deleting data when it's no longer needed). You can also set firewall rules to only allow access to certain trusted groups or individuals or to prevent hitting an account's storage or throughput limit.

Most data lakes use cloud storage access tiers, with the raw and conformed layers using the archive tier, the cleansed layer using the cold tier, and the presentation and sandbox layers using the hot tier. Each tier offers a different balance of storage cost and access costs, with the hot tier being the most expensive for storage and archive having the highest retrieval cost.

I recommend putting auditing or integrity checks in place to make sure the data is accurate as it moves through the layers. For example, if the data is finance data, you might create a query that sums up the total of the day's orders (row count and sales total) and compares the values to the source data to make sure that those values are equal in all the data layers. If they are not equal, that indicates a problem with the pipelines you created to move and transform the data. You'll need to fix that and rerun the pipelines.

Multiple data lakes

Ideally, you'd just create one large data lake and use that for all your data, which would simplify finding and combining data for queries or reports. But there are many reasons you might need to create multiple physically separate data lakes (as shown in figure 5-5) instead of just having different sections in the same data lake. Most of the reasons I discuss next aren't possible (or are at least much less convenient) unless the data lakes are physically separate.

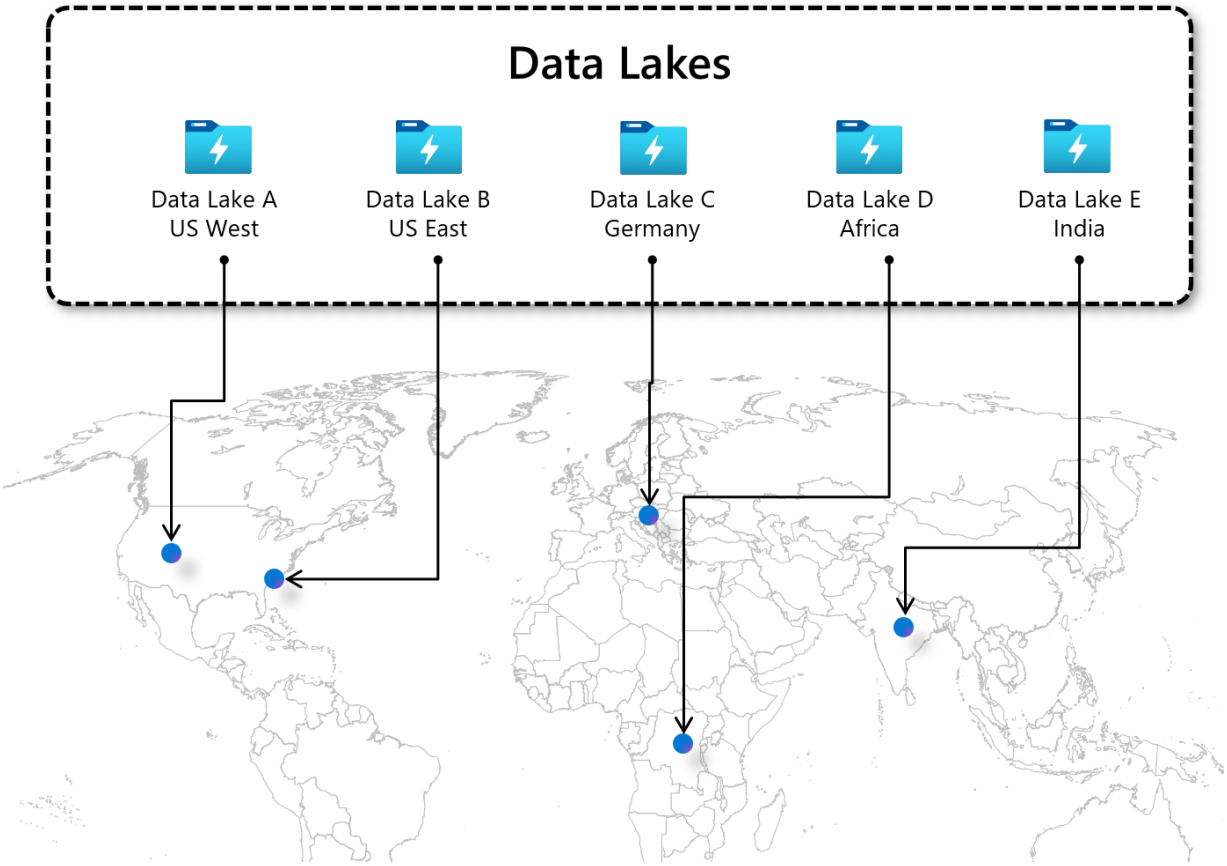


Figure 5-5. Separate data lakes across the world

Advantages

I've grouped the reasons into five sections, after which I'll discuss the disadvantages of having multiple data lakes.

Organizational Structure and Ownership

Maintaining multiple data lakes can be advantageous for many reasons, most of which relate to organization structure and ownership. One such reason can be the company's organizational structure, wherein each organization retains ownership of its own data, a feature typically associated with a data mesh (see chapter 12).

Another reason is that different teams or departments may require their own distinct data lakes for specific use cases or projects.

Further, a company may benefit from having a source-aligned data lake as well as a consumer-aligned data lake. Data from a particular source can be collected into a data lake with minimal transformations, which works well for those who understand the source data, such as those in manufacturing when the data is sourced from a manufacturing process. However, those outside of manufacturing may find the data difficult to understand. In such cases, you might copy the data to a consumer-aligned data lake and transform it to make it more understandable. Thus, having separate data lakes for source-aligned and consumer-aligned data instead of one data lake can simplify the data handling process.

Compliance, Governance, and Security

The decision to have multiple data lakes instead of just one is significantly influenced by a variety of factors falling under the broad categories of compliance, governance, and security. For instance, multi-regional deployments can necessitate multiple data lakes because of data residency or sovereignty requirements vary across different regions. For example, data originating from China cannot be exported outside the country, so it requires a unique data lake within the region.

Another crucial consideration is that multiple data lakes allow you to separate sensitive or confidential data from less sensitive data. More restrictive and stringent security controls can be applied specifically to these sensitive data lakes, enhancing the overall data security.

A further reason is that if some individuals have elevated access privileges, separate data lakes allow you to limit the scope of those privileges, confining them to the particular data lake in which the individual is working. This promotes a more secure data environment.

Finally, diverse governance and compliance requirements can be a driving force for having multiple data lakes. Certain regulations, such as GDPR and HIPAA, necessitate unique data-management standards and practices. With multiple data lakes, organizations can man-

age their data according to the specific governance and compliance requirements that apply to each data set, which can be particularly vital in highly regulated sectors.

Cloud Subscription, Service Limits, and Policies

Having multiple data lakes can also be advantageous for reasons primarily associated with cloud subscription, service limits, and policies. For example, they can help circumvent cloud providers' subscription or service limits, quotas, and constraints, such as a cap on the maximum number of storage accounts per subscription. In such scenarios, if you were to operate with only a single data lake, your needs could exceed those limits.

Moreover, multiple data lakes allow you to implement distinct cloud policies. Cloud providers typically offer hundreds of different policy options which can be tailored individually for each data lake. This flexibility aids in ensuring compliance with company policies. An example might be a specification that storage accounts should have infrastructure encryption, a rule which may vary from one data lake to another within your operation.

Finally, it can be more straightforward to track costs for billing purposes when you maintain separate data lakes, each with its own cloud subscription. This level of granularity in cost tracking offers a clear

advantage over alternative methods provided by cloud services, such as using tags for each resource.

Performance, Availability, and Disaster Recovery

Performance, availability, and disaster recovery also provide numerous reasons to consider employing multiple data lakes. One such reason is improving latency. Using a single data lake for a global set of end users can be very slow for those who are far away. By situating a data lake in the same region as the end-user or application querying the data, you can significantly decrease the time it takes to access data.

Maintaining multiple data lakes with duplicate copies of the data in different regions significantly enhances disaster-recovery capabilities. If a data lake in one region becomes inaccessible, end-users can be redirected to an alternate region holding identical data.

Having multiple data lakes also provides the flexibility to implement different data-recovery and disaster-recovery strategies for different types of data. Critical data that demands constant availability could be hosted in a data lake whose data is replicated across multiple lakes in different regions, while less critical data that can withstand periods of inaccessibility could be stored in a single data lake for significant cost savings.

Finally, with multiple data lakes, you can implement different service levels for distinct types of data. For instance, you could dedicate one data lake to storing and processing high-priority data, which requires low-latency access and high availability, and another data lake to lower-priority data, where higher latencies and lower availability are tolerable. This strategy optimizes the cost and performance of your data-management infrastructure by allowing you to use less expensive storage and processing resources for lower-priority data.

Data Retention and Environment Management

Multiple data lakes can make data retention and environment management more efficient. For instance, by segregating the data lakes dedicated to development, testing, and production environments, organizations can ensure that each environment has its own isolated space for data storage and processing. This minimizes the risk of interference or conflicts between different stages of the data lifecycle.

Another advantage of having multiple data lakes is that you can implement distinct data-retention policies. Legal or regulatory requirements often dictate the need to retain data for specific periods. If you have separate data lakes for different types of data, you can easily enforce diverse retention policies tailored to those categories. This approach allows for efficient management of data retention, ensuring

compliance with various regulations while optimizing storage resources.

Disadvantages

Using multiple data lakes can increase the complexity and cost of your data management infrastructure and require more resources and more expertise to maintain, so if you have a choice, it's important to weigh the benefits against the costs. However, in some cases you will have no choice; for example, if you're designing around a need for data sovereignty, you'll have to have multiple data lakes.

Properly transferring data between multiple data lakes while maintaining its consistency may require additional integration and management tools (such as Azure Data Factory, Informatica, or Apache NiFi). Finally, having multiple data lakes adds the performance challenge of combining the data when a query or report needs data from multiple lakes: if the lakes are physically located very far apart, possibly even in different parts of the world, it can be time-consuming to copy all the data into one location.

Summary

This chapter provides a comprehensive exploration of the data lake: a large-scale data storage and management solution, capable of holding raw data in its native format. I've discussed their role, their design principles, and why you might consider maintaining more than one data lake in certain contexts. Unlike relational data warehouses, data lakes allow quick data storage without any up-front preparation or transformation. This ensures a rapid and effortless data ingestion process, particularly useful in today's era of Big Data.

I discussed the benefits of using a data lake, emphasizing the flexibility and speed they offer for data storage. This aspect, along with their capability to accommodate a diverse range of data types (structured, semi-structured, and unstructured) is a key advantage, especially in situations where quick, scalable, and diverse data collection is crucial.

The chapter also introduced the bottoms-up approach, an important methodology in which you collect data collection before generating any theories or hypotheses. This approach, contrary to the traditional top-down strategy, fosters a more agile, data-centric decision-making process.

A substantial part of the chapter focused on the design of a data lake, introducing the concept of logically dividing a data lake into multiple layers. This layering structure helps to manage data in an organized

manner, paving the way for efficient data retrieval and analytics, and can also aid in managing security, access controls, and data lifecycle management.

The chapter concluded by exploring reasons why an organization might choose to have multiple data lakes. The potential benefits include increased security, improved regulatory compliance, and better data management across distinct business units or specific use cases.

In Chapter 6, we'll move on to look at data stores.

Chapter 6. Data Storage Solutions and Processes

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 6th chapter of the final book.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the author at *jamesserra3@gmail.com*.

In the digital age, data has become the lifeblood of organizations. But, as any seasoned data professional knows, simply having data isn't enough. The real value lies in how effectively this data is managed, stored, and processed. That's why this chapter is a comprehensive guide to navigating the intricate world of data management. I'll take an in-depth look at how data storage solutions and processes operate within diverse data architectures.

The components of data management are like gears in a complex machine: each fulfills a unique role, yet they all work synchronously to achieve a common goal. I'll start with some of the most practical storage solutions: data marts, operational data stores, and data hubs. The second half of the chapter reveals the broad spectrum of processes involved in managing and leveraging data effectively, examining concepts like master data management, data virtualization, data catalogs, and data marketplaces.

For instance, while data marts provide department-specific views of data, operational data stores offer a broader, real-time view of business operations. On the other hand, master data management ensures the consistency and reliability of data, with data virtualization providing a unified, abstracted view of data from various sources. The data catalog serves as the reference guide, making data discovery a breeze, while the data marketplace opens doors for data sharing or even monetizing. Meanwhile, the data hub acts as the system's data epicenter, organizing and managing data from different sources for unified access. Each of these components can stand alone in its function. However, when integrated, they can form a powerful, holistic data strategy, driving your business towards informed decision-making, enhanced operational efficiency, and a competitive advantage.

Whether you're a seasoned data scientist, an aspiring data professional, or a strategic decision-maker, understanding these key com-

ponents and learning to orchestrate them in harmony can unlock the true potential of your organization's data.

Data storage solutions

Let's dive in, shall we? This isn't just about storing and processing data--it's about setting the stage for data-driven transformation.

Data marts

A *data mart*, as shown in Figure 6-1, is a subset of the data warehouse. It is usually designed as a focused repository of data that is optimized to meet the specific needs of a department or a business line within an organization (like finance, HR, or marketing). Because of its narrower scope, a data mart can provide users with a more streamlined and accessible view of information.

It can be challenging for users to navigate large and complex data warehouses that house vast amounts of data. Data marts address this challenge by extracting and consolidating relevant data from the data warehouse and presenting it in a more user-friendly manner. This way, users within a department can locate and retrieve the data they need easily.

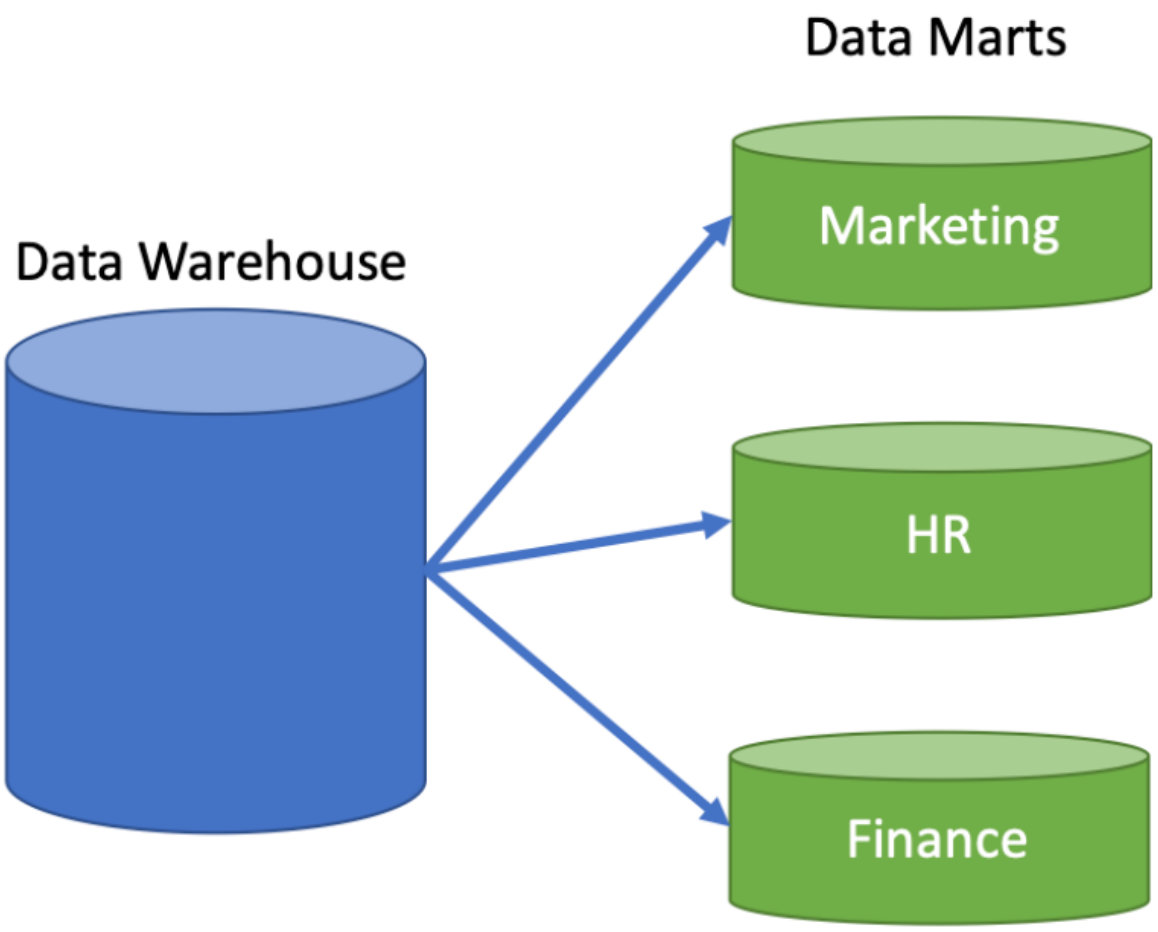


Figure 6-1. Data marts

One of the key advantages of data marts is that they can be created and maintained by individual departments, rather than relying solely on the IT department. Each data mart can be structured differently to accommodate the needs of the department it serves. This allows business users more control over the data: they can tailor it to their unique requirements and use the data structures that are most relevant to their domain.

Department-specific data marts provide business users quicker, easier access to the data they need for their day-to-day operations and decision-making processes. They can perform analytical queries, generate reports, and gain insights into their department's performance without having to navigate through the entire data warehouse.

Moreover, data marts promote data governance and security. Each department can define and enforce access policies for its data mart, ensuring that only authorized individuals have access to specific data sets. This helps to protect sensitive information and ensure compliance with data privacy regulations.

Having delved into data marts, with their targeted, department-specific insights, we now shift our attention to operational data stores.

Operational data stores

The *operational data store* (ODS) has a broader scope than a data mart, providing an enterprise-wide, real-time view of data. While data marts offer valuable, in-depth insights into specific areas, the ODS presents a comprehensive look at the organization's operational data. Each tool offers a different perspective: if a data marts lets you look at your data through a microscope, the ODS lets you view it through a wide-angle lens.

The purpose of an ODS is to integrate corporate data from different sources to enable operational reporting in (or near) real-time. It is not a data warehouse. It provides data much faster than a data warehouse can: every few minutes, instead of daily.

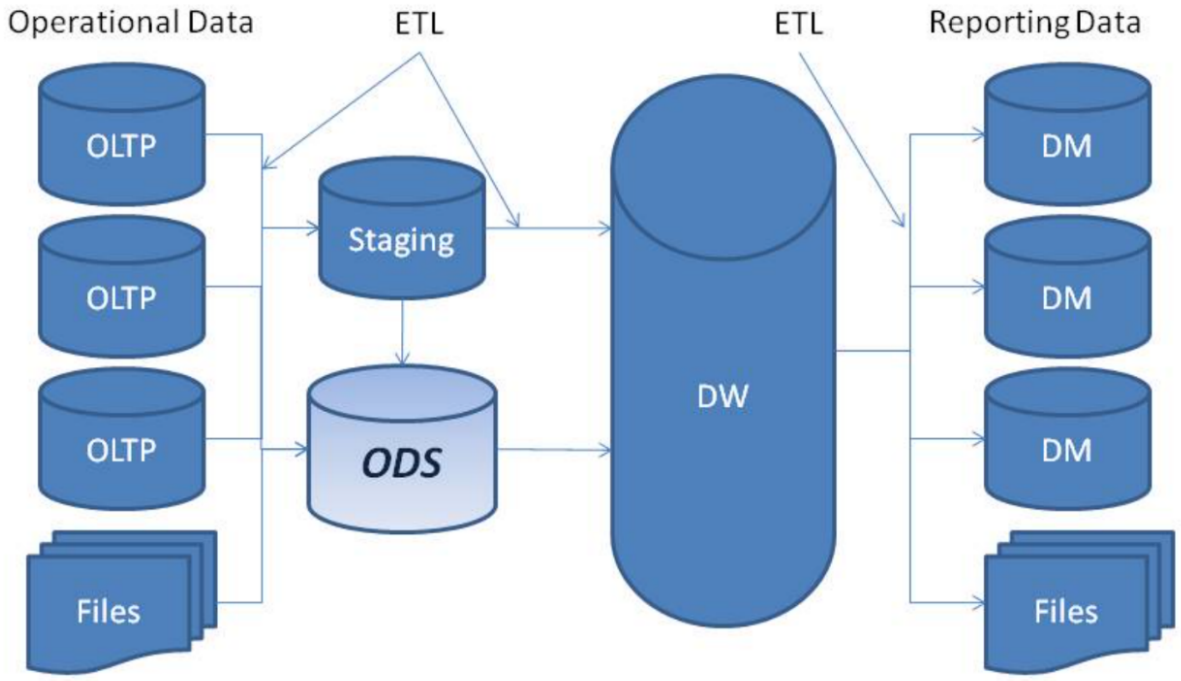


Figure 6-2. Operational Data Store (ODS)

An ODS might be a good fit for your organization if the data you're currently using for reporting in the source systems is too limited, or if you want a better and more powerful reporting tool than what the source systems offer. If only a few people have the security clearance needed to access the source systems but you want to allow others to generate reports, an ODS may be a good solution.

Usually, data in an ODS is structured similarly to its structure in the source systems. During integration, however, the ODS solution can clean, normalize, and apply business rules to the data to ensure data integrity. This integration happens quite frequently throughout the day, at the lowest level of granularity. Normally, you won't need to optimize an ODS for historical and trend analysis, since this is done in the data warehouse.

The data in the ODS is closer to real time than the data required by the data warehouse. You can use an ODS as one of the staging areas for the data warehouse, at least for the data that it maintains (see Figure 6-2). I recommend that you first reconcile and cleanse that data, populating the ODS in near real time to provide value to the operational and tactical decision makers who need it. You can then also use it to populate the data warehouse, which typically has less demanding load cycles. You will still need the staging area for DW-required data that is not hosted in the ODS, but in following this recommendation, you reduce the number of ETL flows and staging-area sizes. That helps improve performance without sacrificing value or function. Table 6-1 summarizes the most important differences between an ODS and a data warehouse.

Table 6-1. *Differences between ODS and data warehouse*

	Operational Data Store (ODS)	Data Warehouse
Best Suited For	Granular, low-level queries against detailed data	Complex queries against summary or aggregated data
Purpose	Operational reporting; current or near real-time reporting	Historical and trend analysis reporting on a large volume of data
Data Duration	Contains a short window of data	Contains the entire history of the organization's data
Decision Making	Supports operational and tactical decisions on current or near real-time data	Provides feedback on strategic decisions, leading to overall system improvements
Data Load Frequency	Might load data every few minutes or hourly	Might load data daily, weekly, monthly, or quarterly

Use case

Let's say you're running inventory reports for a fictional retail chain called ShoesForLess. If you only have one retail store with one database to track orders, you could just run the inventory report against that database. But if you have many retail stores, each of which tracks orders in its own database, the only way to get up-to-the-minute accurate inventory reports is to combine the order data from those databases. This is important for letting customers know which items are out of stock, as well as restocking items quickly when supply is low. A data warehouse would not be real-time enough for that situation.

Now suppose a customer service representative gets a call from a customer asking whether a particular style and size of shoe is available. The representative queries the ODS to retrieve real-time data on availability for that type of shoe (for example, "Men's Running Shoe, Size 10") and return the current stock level.

On the other hand, if the store's marketing manager wants to understand sales trends for that shoe style over the past year, they can use the data warehouse. They run an aggregated query that sums up the total sales of "Men's Running Shoe" for each month within the past year. This provides a high-level view of trends that can inform strategic marketing and inventory decisions.

Data hubs

A *data hub* is a centralized data storage and management system that helps an organization collect, integrate, store, organize, and share data from various sources and provides easy access to analytics, reporting, and decision-making tools. Most significantly, data hubs eliminate the need to use hundreds or thousands of point-to-point interfaces to exchange data between systems. While data warehouses and data lakes have the primary (and often exclusive) goal of serving data for analytics, data hubs are also valuable as a means of communication and data exchange among operational systems (Figure 6-3).

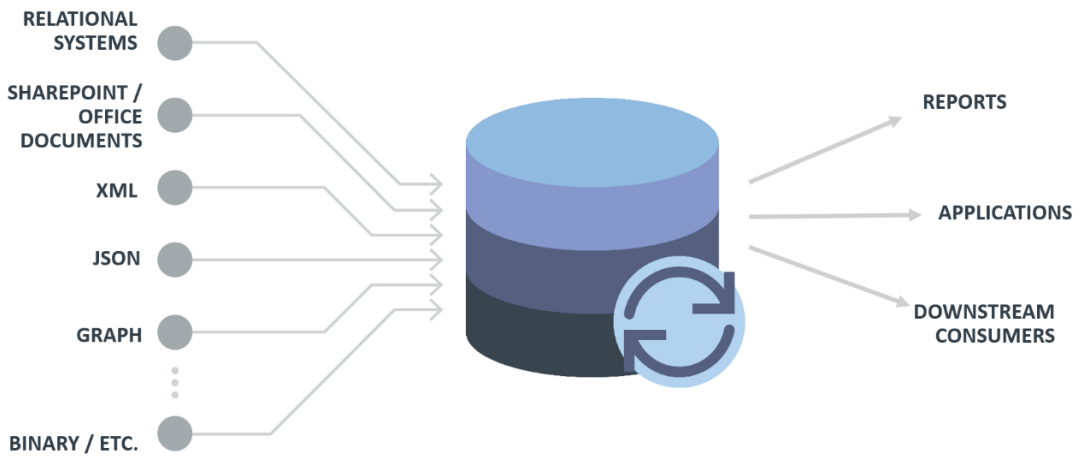


Figure 6-3. Data Hub

Complementary approaches

Data hubs are often used in conjunction with data lakes and warehouses to create a comprehensive data-management infrastructure. As you learned in chapters 4 and 5, data lakes store large volumes of raw, unprocessed data, while data warehouses store structured, processed data optimized for analytical processing. A data hub complements these systems by acting as an intermediary for data exchange, collaboration, and governance. To clarify, let's look at how the data hub interacts with and compares to the other approaches in this chapter. (Table 6-2 summarizes this comparison.)

A data hub stores copies of source data and is focused on managing and distributing, whereas a data catalog does not store actual data and is primarily concerned with metadata management, data lineage, enforcing data governance policies, and facilitating data discovery.

A data hub can serve as a source for a data marketplace, which does not store data. The hub is focused on managing and distributing data within an organization, not the commercial transactions of a data marketplace. It is a central platform that uses more storage systems than a data lake or RDW. Data lakes typically use object storage to store data, while RDWs use relational storage; data hubs can combine storage systems like relational databases, NoSQL databases, and data lakes.

Like a data lake, a data hub supports structured, semi-structured, and unstructured data, as shown in figure 6-5, whereas a RDW only supports structured data. Data hubs are more likely to provide built-in mechanisms for data cataloging, lineage, and access controls, where a data lake often integrates third-party tools or custom solutions instead.

In terms of processing, a data lake uses compute to clean data and land it back in the data lake. RDWs do extensive data processing, including ETL operations, to clean, normalize, and structure data for analytical purposes. With a data hub, however, data is usually transformed and analyzed outside the hub using external tools or applications.

Table 6-2. *Comparison of data solution features*

Feature	Data Hub	Data Lake	Relational Data Warehouse	Data
Primary focus	Data storage, integration, distribution	Data storage, processing, analytics	Data storage, processing, analytics	Meta mana data disco
Data storage	Raw or lightly processed data	Raw and processed data	Highly structured, processed data	Meta only
Data structure	Structured, semi-structured, unstructured	Structured, semi-structured, unstructured	Structured	Meta only

Feature	Data Hub	Data Lake	Relational Data Warehouse	Data
Data processing	Limited processing, mainly integration and distribution	On-demand processing, analytics	Extensive ETL, complex queries	None
Data sharing	Built-in	Third party	Third party	None
Data ownership	Within organization or related groups	Within organization or related groups	Within organization or related groups	With orga or re grou
Use cases	Data ingestion, integration, distribution	Big data analytics, machine learning, AI	Complex analytics, reporting, decision-making	Data disco data enfo data gove polic

In summary, a data hub is a flexible and versatile solution for ingesting, managing, integrating, and distributing data from a wide variety of types and sources. By contrast, a data lake is focused on storing and processing large volumes of raw data for use in analytics and machine learning, and a data warehouse is optimized for structured data storage for use in complex analytics, reporting, and decision-making.

Data processes

This chapter has explored the landscape of data storage solutions, understanding how they provide versatile, efficient ways to organize and access data. However, storing data is only one piece of the puzzle. To truly leverage the potential of data, we must also understand how to effectively manage it. So, we now transition from storage solutions to the intricacies of data processes, where we will discover the strategies and techniques to manipulate, govern, and capitalize on our data assets.

Master data management

Master data management (MDM) is a set of technology, tools, and processes that enable users to create and maintain consistent and accurate lists of master data (such as customer, product, and supplier lists). It involves creating a single master record for each person,

place, or thing in a business, from across internal and external data sources and applications. You can then use these master records to build more accurate reports, dashboards, queries, and machine-learning models.

Most MDM products have very sophisticated tools to clean, transform, merge, and validate the data to enforce data standards. They also allow you to build hierarchies (for example, Company –> Department –> Employee), which you can use to improve reporting and dashboards. The first step is that you copy the data you want to master into the MDM product. The product then cleans and standardizes the data and creates a master record for each entity (say, a customer) that you are trying to master. It removes most duplicates automatically, but some of the work will have to be done manually, with someone reviewing the records that might be duplicates. The master records are then copied back into the data lake or DW. This master record is also known as a "golden source" or "best version of the truth."

If you are creating a star schema (see Chapter 8), the mastered records would be your dimension tables, which would then be joined to your fact tables (which are not mastered).

Use case

Let's go back to ShoesForLess, the retail chain where each store has its own customer database. The company collects the customer data from all of those databases into a data warehouse. Because some customers have purchased items from more than one retail location, the DW will include duplicates of some customer records.

If the customer's name is spelled exactly the same in both records, you can easily filter out duplicates. But if the name is misspelled or different (for example, it contains a middle initial, a hyphen, or a suffix like "Junior or "Senior" in one record but not the other), the filter will not recognize them as duplicates. (These are called *near-duplicates*.)

If you don't implement MDM, the first time the end-user receives a report from the DW, they will see the same customers showing up multiple times and will likely question the report's accuracy. You will have already lost their trust—and once you do, it is very difficult to win it back. So what's the solution?

Data virtualization and data federation

Data virtualization goes by several names: you might see it referred to as a *logical data warehouse*, *virtual data warehouse*, or *decentralized data warehouse*. It is a technology that allows you to access and combine data from various sources and formats by creating a logical view of the data, without replicating or moving the actual data. It pro-

vides a single access point for all data, enabling real-time data integration and reducing the time and costs associated with traditional data-integration approaches such as ETL or ELT (see Chapter 9).

Data federation and *data virtualization* are very similar terms that are often used interchangeably, and you are unlikely to get into trouble if you mix them. However, there are subtle differences between the two. Data federation focuses on being an umbrella organization consisting of smaller, fully or partially autonomous subset organizations that control all or part of their operations. (The United States' system of federal and state governments is a good example of a federation.) Like data virtualization, data federation creates a united view of data from multiple data sources. Data virtualization, however, is a broader concept that encompasses data federation as well as data transformation, caching, security, and governance.

The better data virtualization tools provide features that help with performance, such as query optimization, query pushdown, and caching. You may see tools with these features called *data virtualization* and tools without them called *data federation*.

Data virtualization can replace a data warehouse or ETL (see Chapter 4). Let's look at how.

Virtualization as a replacement for the data warehouse

In some cases, especially when there are many data sources or they are changing constantly, data virtualization can be an alternative to a data warehouse. When you do this, there is no copying and storing the data in a central location.

Figure 6-4 shows how most data virtualization products work. The grid (step 1) represents an end-user's query results: a list of items that contains just addresses. When the end-user scrolls through the grid to see more data, the grid requests items that are currently not loaded from the data-virtualization engine (step 2). If the engine has a cache of data available, it returns the cache immediately and the grid simply updates to display the data. If you request data from outside the cache, the data-virtualization engine passes the request to the remote data source (step 3) where it is processed. The newly fetched data is returned to the engine (step 4) and then passed to the grid. In both situations, the grid always remains completely responsive.

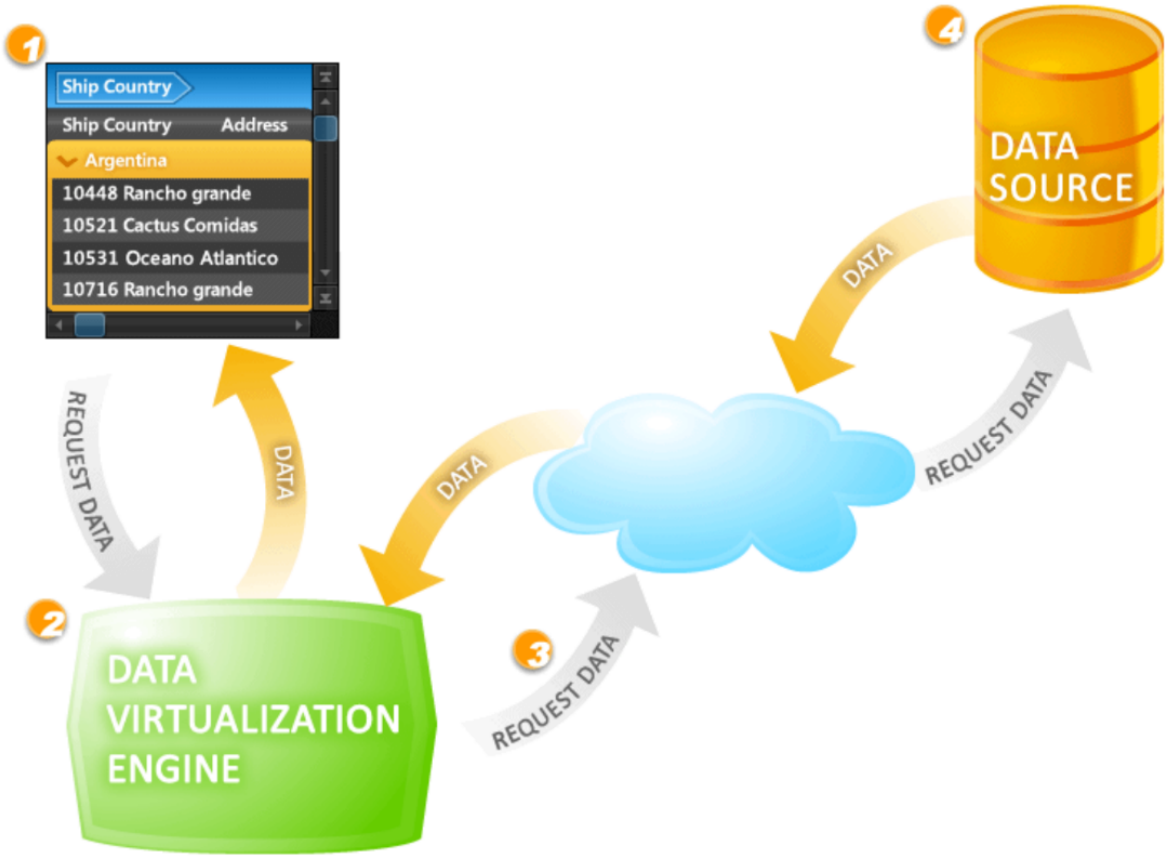


Figure 6-4. Data virtualization

The main advantage of a data virtualization solution is that it optimizes for speed to market. It can be built in a fraction of the time it takes to build a data warehouse, because you don't need to design and build a DW and ETL to copy the data into it, and you don't need to spend as much time testing it. Copying the data (as with a DW) means higher storage and governance costs, more ETL flows to build and maintain, and more data inconsistencies, so using virtualization can save you a lot of money.

Data virtualization supports *federated queries*, in which you can issue a single query that retrieves data from multiple sources and formats and combines the results into a single result set.[1]

However, there are some major drawbacks to data virtualization and federated queries. If you're considering a data virtualization solution to replace your DW, I recommend you ask the following questions:

- What level of performance do I need? Is this a solution I could use for a dashboard that needs sub-second response times, or is it more for operational reporting?
- How much will the data virtualization solution affect the performance of the source system? Will my query run slowly if another end-user runs a query against the same data source that consumes all the resources of the server? Does it push down the query?[2]
- Do I need to install something on each server that contains a data source I want to use?
- Does the solution use the index of each technology on the data store, or does it create its own indexes?[3] Does it use the database statistics from each data source for queries?[4]
- Can I restrict which users can access each data source for security purposes?
- How does the solution handle near-duplicates?
- Where and how will the data be cleaned?

- Will reports break if the source system is changed?

In addition to these questions, there are other valid reasons why a physical data warehouse may be required, as discussed in chapter 4, such as running accurate historical reports, renaming tables, and keeping historical data.

Virtualization as a replacement for ETL or data movement

If you're building a data warehouse, should you move all the source data into the data warehouse, or keep the source data where it is and create a virtualization layer on top of some of it?

The most common reason to move data is if you plan to aggregate or transform it once and then query the results many times. Another is if you will frequently be joining data sets from multiple sources and need super-fast performance. In these scenarios, I generally recommend data warehouse solutions.

However, at times, you might have ad-hoc queries that don't need to be super-fast. And you could certainly have a data warehouse that uses data movement for some tables and data virtualization for others.

Table 6-3 compares data movement and data virtualization.

Table 6-3. Comparison of data movement and data virtualization

	Development & Operations Costs	Time to Solution	Security	Data Fres & Q
Data Movement	Costly to build and maintain ELT jobs Duplicate data storage costs	Takes time to build and run jobs	Creating copies of the data makes it more vulnerable to hackers	ETL pipe mak data ETL intro data
Data Virtualization	Reduces ongoing maintenance and change management	Allows for rapid iterations and prototyping	Data is kept in one secure place, minimizing the attack	Data que alwa “fres accu

	Development & Operations	Time to Solution	Security	Data Freshness
Minimizes storage costs		surface	from source	

Data virtualization offers several benefits. First, it provides a comprehensive data lineage, ensuring you can clearly understand the data's path from its source to the presentation layer. Second, including additional data sources becomes seamless, eliminating the need to modify transformation packages or staging tables. Last, data virtualization software presents all data through a unified SQL interface. This allows users easy access regardless of the source, which could be flat files, spreadsheets, mainframes, relational databases, or other types of data repositories.

While this table shows the benefits of data virtualization over data movement, those benefits may not be enough to overcome the sacrifice in performance and other drawbacks of not using a relational data warehouse, listed in chapter 4. Also, keep in mind that the virtualization tool you choose may not support *all* of your data sources.

Use cases

Here are some reasons you might choose data virtualization:

There are regulatory constraints on moving data

As you learned in chapter 5, Chinese data sovereignty regulations hold that data located in China has to stay in China and can only be queried by a person currently located there. But say you're in the US and try to query customer data in China from your application. The application can retrieve metadata *about* the customer data in China, but it can't ingest that data. You could use virtualization software to keep the data within the country by only letting a person inside the country query it. Anyone outside China trying to query the data would either get an error message or just not see any results.

An end-user outside the company wants to access customer data

Instead of you sending them a copy of the data, they can use virtualization software to query it. This lets you control who accesses what data and know how much they've accessed.

There is reference data in a database outside of the DW that changes frequently

Instead of copying that data into the DW again and again, you could use virtualization to query it when you're joining the external database's data with the DW data.

You want to query data from different data stores and join it together

Virtualization software usually supports many data sources (say, a data lake, relational database, and NoSQL database) and can handle this.

You want your users to do self-service analytics via a virtual sandbox

A *virtual sandbox* is a secure and isolated environment where users can explore, analyze, and manipulate data without affecting the underlying production data or systems. For example, say you're using a virtual sandbox to analyze both customer data and third-party data. Instead of copying that data into the virtual sandbox, you could use virtualization to query it *within* the virtual sandbox. This minimizes the number of copies made of the data, and you avoid having to create ETL.

You want to build a solution quickly to move source data

Instead of using a traditional ETL tool to copy the source data to a central location and then creating reports from that, you can just create reports that use the virtualization software. It will run queries against the source data and do the centralization automatically. This means you can build a solution a lot faster, since you don't have to use an ETL tool (which can be very time-consuming).

Data Catalogs

A *data catalog* is a centralized repository, typically housed in the cloud, that stores and organizes metadata about all of an organization's data sources, tables, schemas, columns, and other data assets. It functions as a single source of truth for users to discover, understand, and manage data that is located in application databases, data lakes, relational data warehouses, operational data stores, data marts, and any other data storage form.

A data catalog typically includes the following information about each data asset:

- Information about the source, such as location, type, and connection details
- If within an RDW, table and schema information, such as structure, relationships, and organization, and column information, such as data types, formats, descriptions, and relationships between columns
- If within an object store, such as a data lake, file properties (storage, folder name, filename)
- Data lineage (how the data arrived from its origin), including information on any transformation, aggregation, and integration the data has undergone

- Data governance and compliance details, such as data quality, ownership, and policies
- Search and discovery tools for users to search, filter, and find relevant data

A data catalog helps you manage data more effectively by providing visibility into the data landscape. This, in turn, facilitates better collaboration between teams and more informed decision-making. It also helps ensure data quality, security, and compliance by making it easier to track data lineage, enforce data policies, and maintain a clear understanding of data assets.

There are many products on the market for creating data catalogs, some of which have been around for nearly ten years. The most popular products are Informatica Enterprise Data Catalog, Collibra Catalog, and Microsoft Purview.

Data Marketplaces

A *data marketplace*, sometimes called a *data exchange*, is an online platform where data providers and data consumers come together to buy, sell, and exchange data sets.[5] These marketplaces are designed to help data consumers (like businesses, researchers, and developers) discover, evaluate, and purchase data for purposes such as analysis, machine learning, business intelligence, and decision-mak-

ing. Providers usually include organizations, governments, and individuals who have collected or generated valuable data sets.

A data marketplace typically includes a data catalog. Many marketplaces assess and improve the quality of data sets to ensure they are accurate, complete, consistent, and easy to use. Some provide tools and services for cleaning, transforming, and enriching data, which can save users time and effort in preparing it for analysis. (This is particularly valuable when you're integrating data from multiple sources or dealing with incomplete or inconsistent data.) Once a consumer makes a purchase, the marketplace may offer tools to help them access the data and integrate it into their workflows, applications, or systems. Some even allow users to customize data sets by combining data from multiple sources or filtering it based on specific criteria, so they can access only the data they need.

Data marketplaces should have robust security measures in place to protect user data, ensure compliance with data protection regulations, and maintain the confidentiality of sensitive information. Some offer ways to obfuscate private or sensitive data easily. Data marketplaces' pricing structures and licensing agreements clearly define the terms of use and ensure legal compliance for both data providers and consumers.

Many data marketplaces also offer built-in analytics and data-visualization tools that allow you to analyze and visualize (or simply preview) data directly within the platform. Other common features and tools include:

- Ratings and reviews of data sets
- Recommendations for other data sets you might like
- Shopper profiles for easy reordering, favorites lists, and so on
- Collaboration and community
- Training and support

From the providers' perspective, data marketplaces offer a way to monetize data assets, generating new revenue streams. This can be particularly beneficial for organizations that collect large amounts of data but don't have the resources or expertise to analyze and monetize it themselves.

Data marketplaces have increased in popularity as the demand for data-driven insights grows and more diverse data sets continue to become available. There are not nearly as many products available for creating data marketplaces as there are for data catalogs, since marketplaces just started appearing in the early 2020s. The most popular are Snowflake Data Marketplace, Datarade Marketplace, and Informatica Cloud Data Marketplace. Both providers and consumers can benefit from a more efficient and transparent exchange of data,

which can foster innovation of new data-driven products and services.

Summary

In this chapter, you explored various approaches to data stores, all of which play pivotal roles in today's data-driven business environment.

I discussed data storage, including data marts, operational data stores, and data hubs, then changed focus on processes. You learned about master data management (MDM), a crucial approach that focuses on managing core business entities to ensure data accuracy, uniformity, and consistency across multiple systems and processes. I then moved on to discuss data virtualization, data catalogs, and data marketplaces.

By understanding these different strategies and their functionalities, you can better align your data architecture to your organization's needs and drive more significant business value from your data.

The difference between data federation and federated queries is that data federation is more about the system's architecture, while federated queries are the mechanism used to extract information from this system. In practice, if you have a data federation system in place, you are using federated queries to retrieve your data.

In data virtualization, the term *push-down query* refers to the capability to push certain parts of a query, such as filtering or aggregation, to the data sources themselves. This allows the data virtualization layer to offload processing tasks and retrieve only the relevant results. The benefits include enhancing performance and minimizing data transfer; the tradeoff is that it uses compute resources on the server where the data source resides.

Database indexes are a type of data structure that improves the speed of data-retrieval operations on a database table. They work much like book indexes, providing users a quick way to access the data they're seeking.

Database statistics are a set of information that a *database management system* (DBMS) maintains about the data stored in a database. These statistics help the DBMS optimize query performance by informing how it decides on the most efficient way to execute a given query.

Sometimes, however, a data marketplace is internal to a company and is simply used to help its users find and exchange data.

Chapter 7. Approaches to Design

A NOTE FOR EARLY RELEASE READERS

With Early Release ebooks, you get books in their earliest form—the author's raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 7th chapter of the final book.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the author at *jamesserra3@gmail.com*.

This chapter explores design approaches to handling and organizing data, and how these methods help build powerful, adaptable, and reliable systems for data management. In simple terms, we'll learn about the strategies that help us decide how data should be stored, processed, and accessed, enhancing the speed and dependability of our data systems.

To clarify, let's draw a distinction between data design and data modeling, the subject of the next chapter. Think of *data design* as like

building a city. It's about deciding where the buildings go, which roads connect different parts of the city, and how traffic flows. On the other hand, *data modeling* is more like designing individual buildings—it's about arranging rooms, deciding how they connect, and what purpose each room serves.

In this chapter, we'll look at different types of data designs. We'll compare and contrast methods like OLTP and OLAP, which are essentially different ways of processing and analyzing data. We'll also explore concepts like SMP and MPP, which are strategies to process data more efficiently. Then, we'll learn about Lambda and Kappa architectures, which are blueprints for handling large amounts of data in real-time. Lastly, we'll talk about an approach called polyglot persistence, which allows us to use different types of data storage technologies within the same application.

My aim here isn't to push one approach as the best solution for every situation, but to help you understand the strengths and weaknesses of each. This way, you can choose or combine the right methods based on your specific needs. By the end of this chapter, you'll have a stronger grasp on how to create efficient data systems that can adapt to changing needs and technological advancements.

Online transaction processing (OTLP) versus online analytical processing (OLAP)

Online transaction processing (OLTP) is a type of informational system or application that processes online create, read, update, and delete (CRUD) transactions (see chapter 2) in a real-time environment. OLTP systems are designed to support high levels of concurrency, which means they can handle a large number of transactions at the same time. They typically use a relational model (see chapter 7.2) and are optimized for low latency, which means they can process transactions very quickly. Examples include point-of-sale applications, e-commerce websites, and online banking solutions.

In an OLTP system, transactions are typically processed quickly and in a very specific way. For example, a customer making a purchase at a store would be considered a transaction. The transaction would involve the customer's account being debited for the purchase amount, the store's inventory count being reduced, and the store's financial records being updated to reflect the sale. OLTP systems use a variety of DBMSs to store and manage the data, such as Microsoft SQL Server and Oracle.

OLTP systems are often contrasted with online analytical processing (OLAP) systems, which are used for data analysis and reporting to support business intelligence and decision making. OLAP systems are optimized for fast query performance, allowing end-users to easily and quickly analyze data from multiple perspectives by slicing and dicing the data in reports and dashboards much faster than with an OLTP system. Think of an OLAP system as "write-once, read-many" (as opposed to CRUD). Often, multiple OLTP databases are used as the sources that feed into a data warehouse, which then feeds into an OLAP database, as shown in Figure 7-1. (However, sometimes you can also build an OLAP database directly from OLTP sources.)

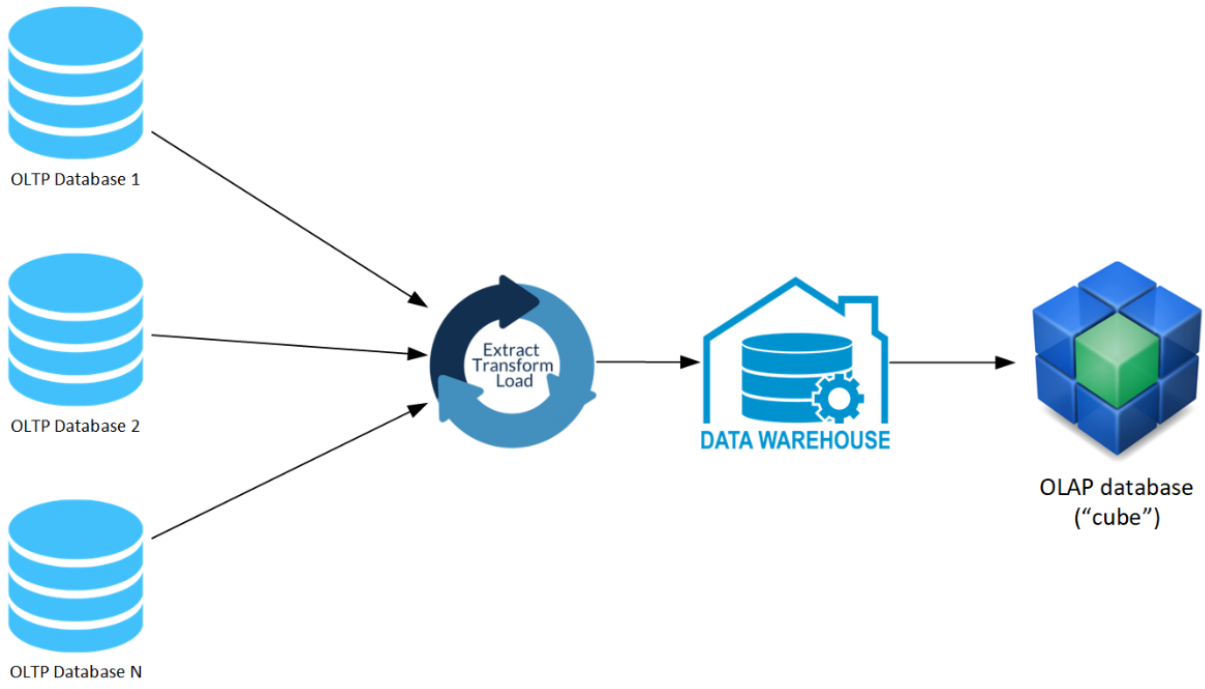


Figure 7-1. OLAP database architecture

An OLAP database is typically composed of one or more OLAP cubes. An *OLAP cube* is where data is pre-aggregated by the cube, meaning that it has already been summarized and grouped by certain dimensions, so that end-users can quickly access it from multiple dimensions and levels of detail without having to wait for long-running queries to complete. Creating an OLAP cube generally involves using a *multidimensional model*, which uses a star or snowflake schema to represent the data. (Chapter 8 will discuss these in more detail.)

For example, a retail corporation might use an OLAP database with OLAP cubes to quickly analyze sales data. This pre-aggregated data is organized by time, allowing for the analysis of sales trends from yearly all the way down to daily. The data is also sorted by location, which facilitates geographic comparisons, such as product sales in different cities. Furthermore, the data is categorized by product, making it easier to track the performance of individual items. If a regional manager needs to quickly review how a particular product category performed in his region during the last holiday season, the OLAP cube has already prepared the data. This allows the manager to avoid waiting for a lengthy query to sort through each individual sale. Instead, the relevant data—sorted by product, region, and time—is readily available, accelerating the decision-making process.

You can also use another, more recent model, called a *tabular data model*, instead of an OLAP cube for similarly fast query performance.

Instead of a multidimensional model, a tabular data model uses a tabular structure, similar to a relational database table, to represent the data. This makes it more flexible and simpler than a multidimensional model. This book refers to the multidimensional model as an "OLAP cube" and the tabular model as a "tabular cube," but please note that the term *cube* is used interchangeably in the industry to denote either of these models.

Table 7-1 compares the differences between these design styles.

Table 7-1. Comparing the characteristics of OLTP and OLAP systems

	OLTP	**OLAP/Tabular**
Processing type	Transactional	Analytical
Data nature	Operational data	Consolidation data
Orientation	Application oriented	Subject oriented
Purpose	Works on ongoing business tasks	Helps in decision making
Transaction frequency	Frequent transactions	Occasional transactions
Operation types	Short on-line transactions: Insert, Update, Delete	Lots of read scans
Data design	Normalized database (3NF)	Denormalized database
Common usage	Used in retail sales and other financial transactions systems	Often used in data mining, sales, and marketing

	OLTP	OLAP/Tabular
Response time	Response time is instant	Response time varies from seconds to hours
Query complexity	Simple and instant queries	Complex queries
Usage pattern	Repetitive usage	Ad-hoc usage
Transaction nature	Short, simple transactions	Complex queries
Database size	Gigabyte database size	Terabyte database size

OLAP databases and data warehouses (see chapter 3) are related but distinct concepts and are often used in conjunction. Through a multidimensional or tabular model, OLAP databases provide a way to analyze the data stored in the DW in a way that is more flexible and interactive than executing traditional SQL-based queries on a DW that contains large amounts of data. Multidimensional models and tabular models are often considered *semantic* layers or models,

which means that they provide a level of abstraction over the schemas in the DW.

Operational and analytical data

Operational data is real-time data used to manage day-to-day operations and processes that is captured, stored, and processed by OLTP systems. You can use it to take a "snapshot" of the current state of the business, to ensure that operations are running smoothly and efficiently. Operational data tends to be high volume and helps in making decisions quickly.

Analytical data comes from collecting and transforming operational data. It is a historical view of the data maintained and used by OLAP systems. With data analytics tools, you can analyze it to understand trends, patterns, and relationships in the data over time. Analytical data is optimized for creating reports and visualizations and for training machine-learning models. It usually provides a view of data over a longer period of time than you get with operational data, often has a lower volume, and is generally consolidated and aggregated. Data is usually ingested in batches and requires more processing time than operational data.

In summary, operational data is used to monitor and control business processes in real-time, while analytical data is used to gain insights and inform decision-making over a longer period of time. Both types of data are essential for effective business management, and they complement each other to provide a complete view of an organization's operations and performance.

Think of OLTP as the technology used to implement operational data, and of OLAP as the technology used to implement analytical data.

Symmetric multiprocessing (SMP) and massively parallel processing (MPP)

Some of the first relational databases used a *symmetric multiprocessing* (SMP) design, where computer processing is done by multiple processors that share disk and memory, all in one server – think SQL Server and Oracle. (This is pictured on the left side of Figure 7-1.) To get more processing power for these systems, you "scale up" by increasing the processors and memory in the server. This works well for OLTP databases, but not so well for the write-once, read-many environment of DWs.

As data warehouses grew in popularity in the 1990s and started to ingest huge amounts of data, performance became a big problem. To help with that, along came a new kind of database design. In a *massively parallel processing* (MPP) design, the database has multiple servers, each with multiple processors, and (unlike SMP) each processor has its own memory and its own disk. This allows you to "scale out" (rather than up) by adding more servers.

MPP servers distribute a portion of the data from the database to the disk on each server (whereas SMP databases keep all the data on one disk). Queries are then sent to a *control node* (also called a *name node*) that splits each query into multiple subqueries that are sent to each server (called a *compute node* or *worker node*), shown on the right side of Figure 7-2. There, the subquery is executed and the results from each compute node are sent back to the control node, mashed together, and sent back to the user. This is how solutions such as Teradata and Netezza work.

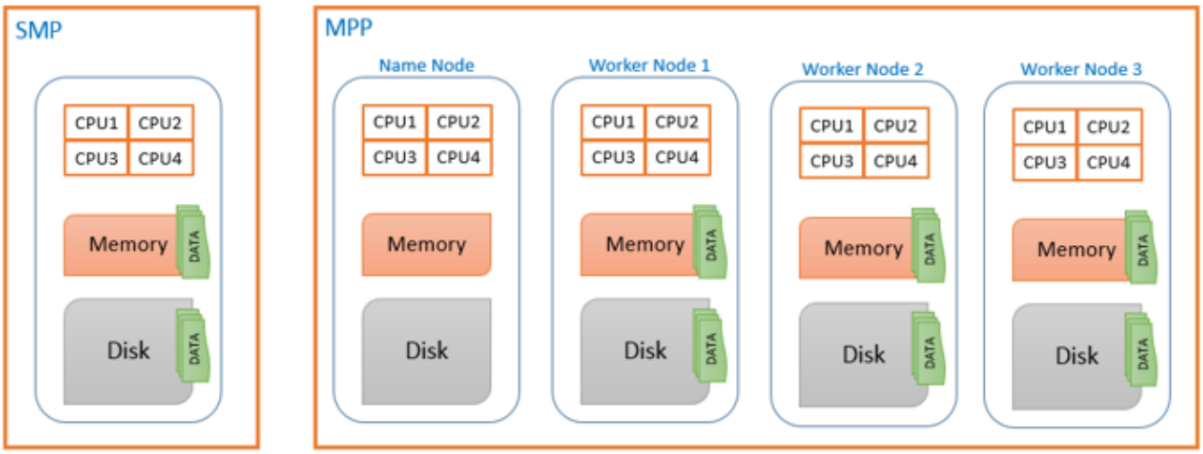


Figure 7-2. SMP and MPP database designs.

By way of analogy: Imagine your friend Fiona has a deck of 52 cards and is looking for the Ace of Hearts. It takes Fiona about 15 seconds, on average, to find the card. You can "scale up" by replacing Fiona with another friend, Max, who is faster. Using Max brings the average time down to 10 seconds—a limited improvement. This is how SMP databases work.

Now imagine you "scale out" instead of up, by replacing Fiona with 26 people, each of whom has only two cards. Now the average time to find the card is just one second. That's how MPP databases work.

SMP and MPP databases started as on-prem solutions that are still prevalent today, but there are now many equivalent solutions in the cloud.

Lambda Architecture

Lambda architecture is a data-processing architecture designed to handle massive quantities of data by using both batch and real-time stream processing methods. This idea is to get comprehensive and accurate views of the batch data and to balance latency, throughput, scaling, and fault tolerance by using batch processing, while simultaneously using real-time stream processing to provide views of online data (such as IoT devices, Twitter feeds, or computer log files). You can join the two view outputs before the presentation/serving layer.

Lambda architecture bridges the gap between the historical "single source of truth" and the highly sought-after "I want it now" real-time solution by combining traditional batch processing systems with stream consumption tools to meet both needs.

The Lambda architecture has three key principles:

A dual data model

The Lambda architecture uses one model for batch processing (batch layer) and another model for real-time processing (stream layer). This allows the system to handle both batch and real-time data, and to perform both types of processing in scalable and fault-tolerant ways.

A single unified view

The Lambda architecture uses a single unified view (called the *presentation layer*) to present the results of both batch and real-time processing to end-users. This allows the user to see a complete and up-to-date view of the data, even though it's being processed by two different systems.

Decoupled processing layers

The Lambda architecture decouples the batch and real-time processing layers so that they can be scaled independently and developed and maintained separately, allowing for flexibility and ease of development.

Figure 7-3 depicts a high-level overview of the Lambda architecture.

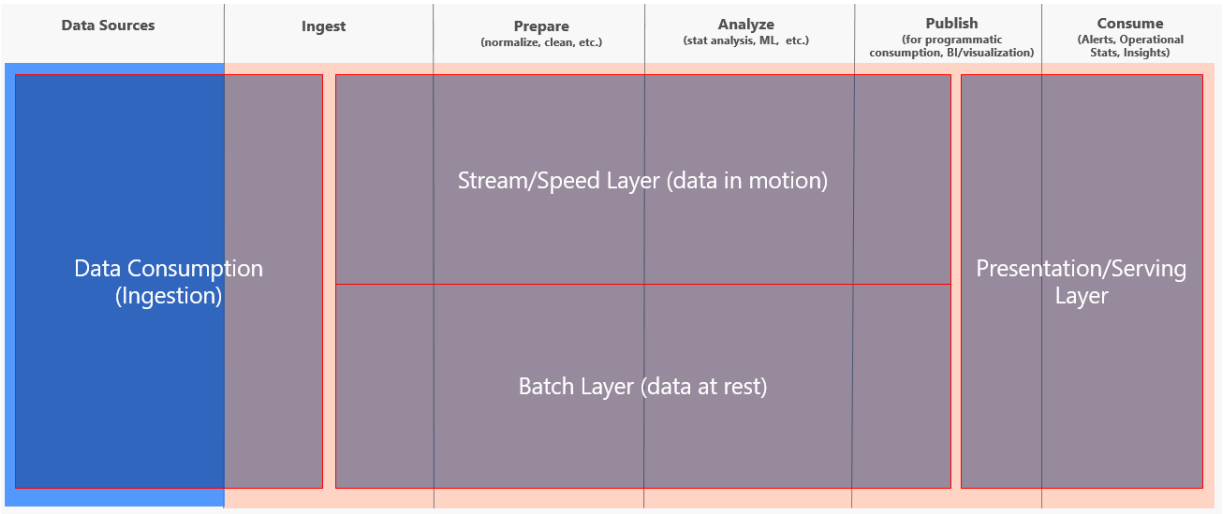


Figure 7-3. Overview of Lambda architecture

On the left of Figure 7-3 is the *data consumption layer*. This is where you import the data from all source systems. Some sources may be streaming the data, while others only provide data daily or hourly.

In the top middle, you see the *stream layer*, also called the *speed layer*. It provides for incremental updating, making it the more complex of the two middle layers. It trades accuracy for low latency, looking at only recent data. The data in here may be only seconds behind, but the trade-off is that it might not be clean. Data in this layer is usually stored in a data lake.

Beneath that is the *batch layer*, which looks at all the data at once and eventually corrects the data that comes into the stream layer. It is the single source of truth, the trusted layer. Here there's usually lots of ETL, and data is stored in a traditional data warehouse or data lake. This layer is built using a predefined schedule, usually daily or hourly, including importing the data currently stored in the stream layer.

At the right of Figure 7-3 is the *presentation layer*, also called the *serving layer*. Think of it as a mediator: when it accepts queries, it decides when to use the batch layer and when to use the speed layer. It generally defaults to the batch layer, since that has the trusted data, but if you ask it for up-to-the-second data (perhaps by setting alerts for certain log messages that indicate a server is down), it will pull

from the stream layer. This layer has to balance retrieving the data you can trust with retrieving the data you want right now.

The Lambda architecture is an excellent choice for building distributed systems that need to handle both batch and real-time data, like recommendation engines and fraud-detection systems. However, that doesn't mean it's the best choice for every situation. Some potential drawbacks of the Lambda architecture include:

Complexity

The Lambda architecture includes a dual data model and a single unified view. That can be more complex to implement and maintain than other architectures are.

Limited real-time processing

The Lambda architecture is designed for both batch and real-time processing, but it may not be as efficient at handling high volumes of real-time data as the Kappa architecture (discussed in the next section), which is specifically designed for real-time processing.

Limited support for stateful processing

The Lambda architecture is designed for stateless processing and may not be well suited for applications that require maintaining state across multiple events. For example, consider a

retail store with a recommendation system that suggests products based on customers' browsing and purchasing patterns. If this system used a Lambda architecture, which processes each event separately without maintaining state, it could miss the customer's shopping journey and intent. If the customer browses for shoes, then socks, and then shoe polish, a stateless system might not correctly recommend related items like shoelaces or shoe storage, because it doesn't consider the sequence of events. It might also recommend items that are already in the customer's cart.

Overall, you should consider the Lambda architecture if you need to build a distributed system that can handle both batch and real-time data but needs to provide a single unified view of the data. If you need stateful processing or to handle high volumes of real-time data, you may want to consider the Kappa architecture.

Kappa Architecture

As opposed to the Lambda architecture, which is designed to handle both real-time and batch data, Kappa is designed to handle just real-time data. And like the Lambda architecture, Kappa architecture is also designed to handle high levels of concurrency and high volumes of data.

The three key principles of the Kappa architecture are:

Real-time processing

The Kappa architecture is designed for real-time processing, which means that events are processed as soon as they are received, rather than being batch-processed later. This decreases latency and enables the system to respond quickly to changing conditions.

A single event stream

The Kappa architecture uses a single event stream to store all data that flows through the system. This allows for easy scalability and fault tolerance, since the data can be easily distributed across multiple nodes.

Stateless processing

In the Kappa architecture, all processing is stateless. This means that each event is processed independently, without relying on the state of previous events. This makes it easier to scale the system, because there is no need to maintain state across multiple nodes.

Figure 7-4 provides a high-level overview of the Kappa architecture.

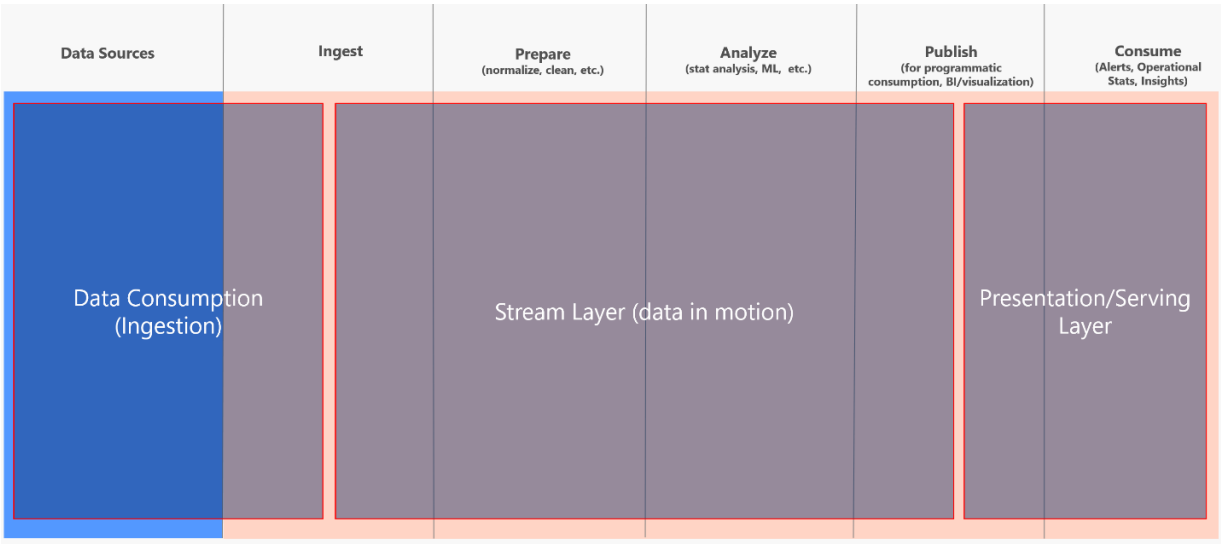


Figure 7-4. Overview of Kappa architecture

The layers in the Kappa architecture are exactly the same as in the Lamba architecture, with the exception that the Kappa architecture does not have a batch layer.

Some potential drawbacks of the Kappa architecture include:

Complexity

The Kappa architecture involves a single event stream and stateless processing, which can be more complex to implement and maintain than other architectures.

Limited batch processing

The Kappa architecture is designed for real-time processing and does not easily support batch processing of historical

data. If you need to perform batch processing, you may want to consider the Lambda architecture instead.

Limited support for ad-hoc queries

Because the Kappa architecture is designed for real-time processing, it may not be well suited for ad-hoc queries that need to process large amounts of historical data.

Overall, the Kappa architecture is an excellent choice for building distributed systems that need to handle large amounts of data in real time and that need to be scalable, fault-tolerant, and have low latency, like streaming platforms and financial trading systems. However, if you need to perform batch processing or support ad-hoc queries, then the Lambda architecture may be a better choice.

Note that the Lambda and Kappa architectures are high-level design patterns that can be implemented within any of the data architectures described in Part III of this book. If you use one of those architectures to build a solution that supports both batch and real-time data, then that architecture supports the Lambda architecture; if you use it to build a solution that supports only real-time data, that architecture supports the Kappa architecture.

Polyglot persistence and polyglot data stores

Polyglot persistence is a fancy term that means using multiple data storage technologies to store different types of data *within a single application or system*, based upon how the data will be used. Different kinds of data are best kept in different data stores. In short, polyglot persistence means picking the right tool for the right use case. It's the same idea behind polyglot programming, which is the idea that applications should be written in a mix of languages to take advantage of different languages' strengths in tackling different problems.

By contrast, a *polyglot data store* means using multiple data stores *across an organization or enterprise*. Each data store is optimized for a specific type of data or use case. This approach allows organizations to use different data stores for different projects or business units, rather than a one-size-fits-all approach for the entire organization.

For example, say you're building an e-commerce platform that will deal with many types of data (shopping carts, inventory, completed orders, and so forth). Instead of trying to store all the different types of data in one database, which would require a lot of conversion to

put everything in the same, you could take a polyglot persistence approach and store each kind of data in the database best suited for it. So, an e-commerce platform might look like the diagram in Figure 7-5.

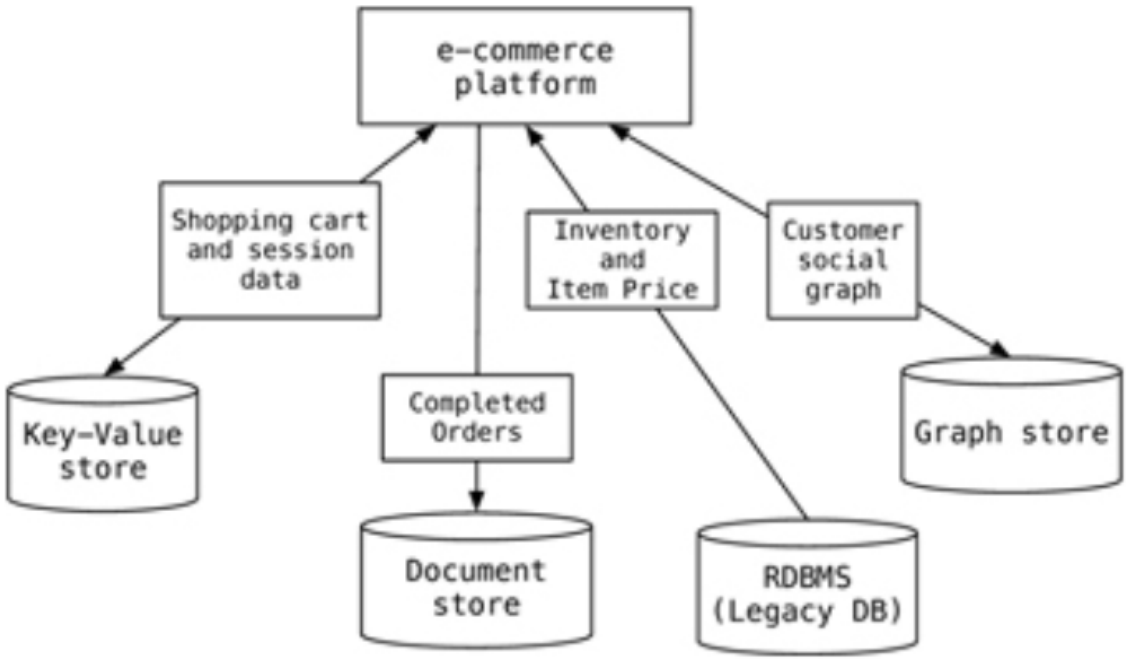


Figure 7-5. An e-commerce platform with a polyglot persistence design.

This results in the best tool being used for each type of data. In Figure 7-4, you can see that the database uses a key-value store for shopping cart and session data (giving very fast retrieval); a document store for completed orders (making storing and retrieving order data fast and easy), an RDBMS for inventory and item prices (since those are best stored in a relational database due to the structured nature of the data), and a graph store for customer social graphs (since it's very difficult to store graph data in a non-graph store).

This will come at a cost in complexity, since each data-storage solution means learning a new technology. But the benefits will be worth it. For instance, if you try to use relational databases for non-relational data, it can significantly slow application development and performance; using the appropriate storage type pays off in speed.

Summary

This chapter explored the architectural concepts and design philosophies that form the basis of effective data systems.

First, you learned about the two primary types of data processing systems: online transaction processing (OLTP) and online analytical processing (OLAP). OLTP systems are designed for fast, reliable, short transactions, typically in the operational databases that power daily business operations. In contrast, OLAP systems support complex analytical queries, aggregations, and computations used for strategic decision-making, typically in a data warehouse. You then learned about the differences between operational and analytical data.

You also learned the differences between symmetric multiprocessing (SMP) and massively parallel processing (MPP) architectures. I then delved into two modern Big Data processing architectures: Lambda and Kappa. Last, I explored the concepts of polyglot persistence and

polyglot data stores, which promote using the best-suited database technology for the specific needs and workload characteristics of the given data.

As we transition into the next chapter, our focus will shift from data storage and processing to the principles and practices of data modeling: the crucial bridge between raw data and meaningful insights. Its approaches, such as relational, dimensional, and the Common Data Model, serve as an underpinning structure that allows you to use and interpret data efficiently across diverse applications.

As we delve into these topics, we'll see how data modeling enables efficient use of the storage and processing solutions we've studied, ultimately serving as a blueprint for transforming raw data into actionable insights.

About the Author

James Serra works at Microsoft as a big data and data warehousing solution architect where he has been for most of the last nine years. He is a thought leader in the use and application of Big Data and advanced analytics, including data architectures such as the modern data warehouse, data lakehouse, data fabric, and data mesh. Previously he was an independent consultant working as a Data Warehouse/Business Intelligence architect and developer. He is a prior SQL Server MVP with over 35 years of IT experience. He is a popular blogger (JamesSerra.com) and speaker, having presented at dozens of major events including SQLBits, PASS Summit, Data Summit and the Enterprise Data World conference.