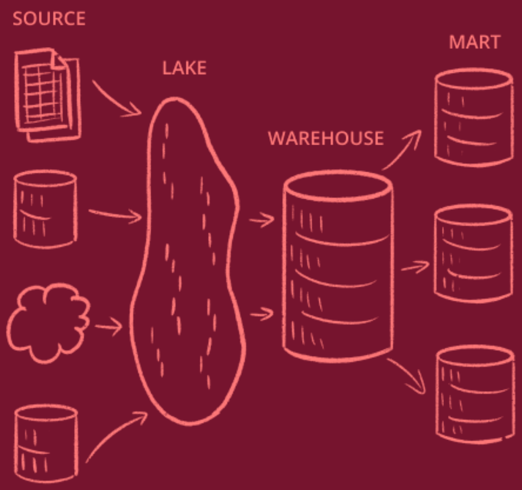
Cloud Data Management

4 Stages for Informed Companies

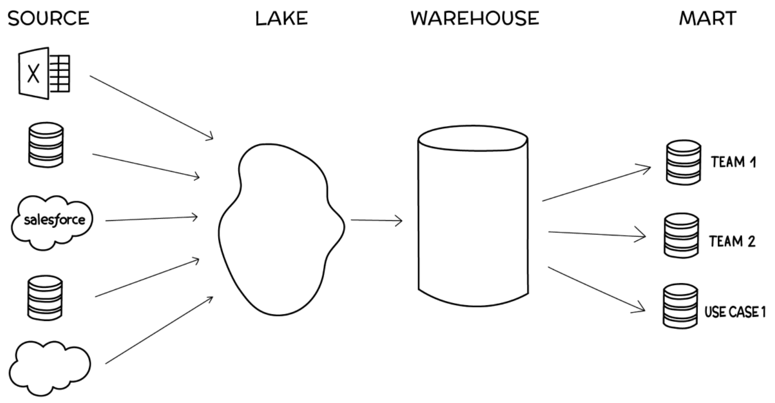


Cloud Data Management

Written by: Dave Fowler, Matt David, Tim Miller, Tracy Chow, Jaime Flores-Lovo, Aaron Aihini, Kostas Pardalis
Reviewed by: Twange Kasoma

Table of Contents

Introduction



Introduction - The 4 Stages of Data Sophistication

Modern companies run on and compete with data. Historically businesses had all of the information they needed walking in and out of their store every day. When customers had requests, frustrations, or buying patterns - the owners and employees were there to ask about them and to directly observe trends.

Over time, companies scaled and most people now work for larger, distributed organizations. We've grown from single shop businesses, to having many locations, and even to having no location at all. The result is that decision makers no longer have direct access to each customer, and must increasingly rely on their data to improve and compete.

Companies today are quite good at collecting data - but still very poor at organizing and learning from it. Setting up a proper Data Governance organization, workflow, tools, and an effective data stack are essential tasks if a business wants to gain from it's information.

This book is for organizations of all sizes that want to build the right data stack for them - one that is both practical and enables them to be as informed as possible. It is a continually improving community driven book teaching modern data governance techniques for companies at different levels of data sophistication. In it we will progresses from the starting setup of a new startup to a mature data driven enterprise covering architectures, tools, team organizations, common pitfalls and best practices as data needs expand.

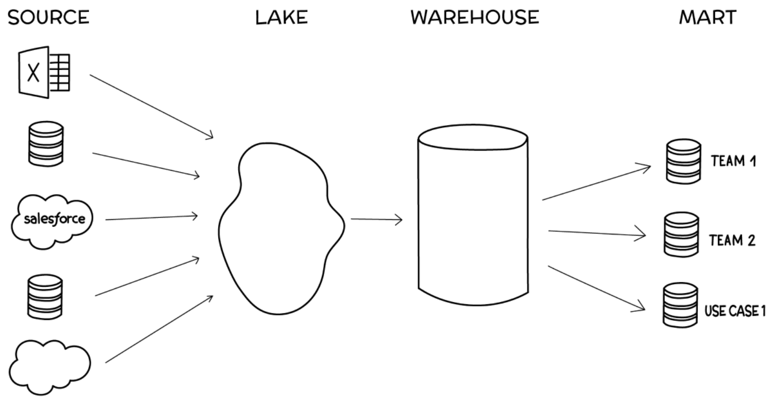
The structure and original chapters of this book were written by the leadership and Data Advisor teams at Chartio, sharing our experiences working with hundreds of companies over the past decade. Here we've compiled our learnings and open sourced them in a free, open book.

The 4 Stages of Data Sophistication

From our experience working with so many organizations we recognized four distinct stages of data sophistication that successful companies go through. These stages happen to be tied to a new piece of the data stack that is needed at each stage, and so we have named these stages after those pieces.

This book is organized in sections covering each of these 4 sequential:

1. Source
2. Lake
3. Warehouse
4. Mart

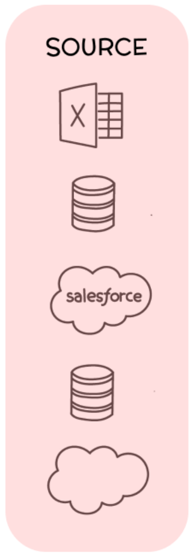


Each vertical stage pictured is a valid stack to operate from, depending on your resources, size, importance and needs of data within your organization. Each has unique benefits, pitfalls and best practices that we'll cover stage by stage.

Your company may not yet need the entirety of this book, but as a growing company's data needs expand, it will be incredibly valuable — and perhaps pivotal — to advance all the way through each of these stages to the Mart stage.

We'll start with an overview of each:

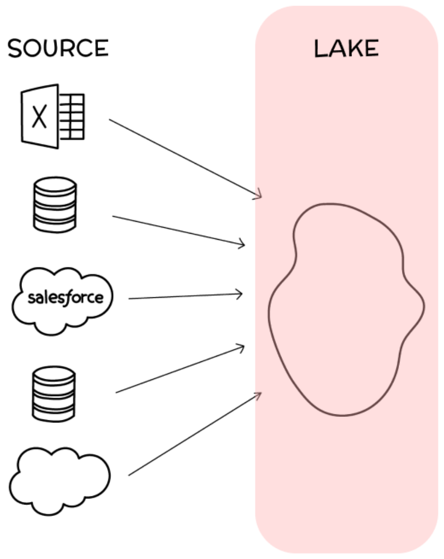
Stage 1. Sources



When you start working with data, you may only have a few sources of interest. Two common early sources are Google Analytics and your application data in whatever PostgreSQL or MySQL database your product runs on. If only a few people at your company need to work with these sources, you might set them up with direct access; it's more simple and agile for them to just work with the data directly.

Stage 2. Lake

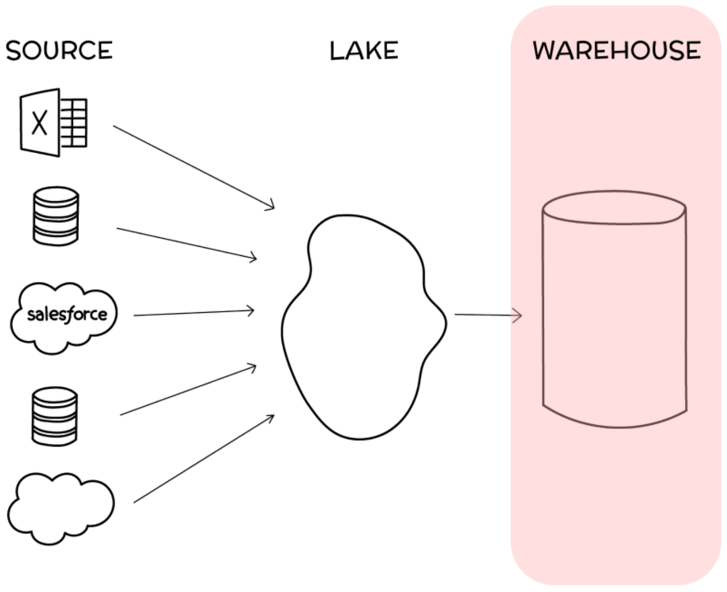
As you start to rely on more data sources, and more frequently need to blend your data, you'll want to build out a Data Lake—a spot for all of your data to exist together in a unified, performant source.



Especially when you need to work with data from applications like Salesforce, Hubspot, Jira, and Zendesk, you'll want to create a single home for this data so you can access all of it together and with a single SQL syntax, rather than many different APIs.

Stage 3. Warehouse (Single Source of Truth)

In the Lake stage, as you bring in more people to work with the data, you have to explain to them the oddities of each schema, what data is where, and what special criteria you need to filter by in each of the tables to get the proper results. This becomes a lot of work, and will leave you frequently fighting integrity issues. Eventually, you'll want to start cleaning your data into a single, clean source of truth.

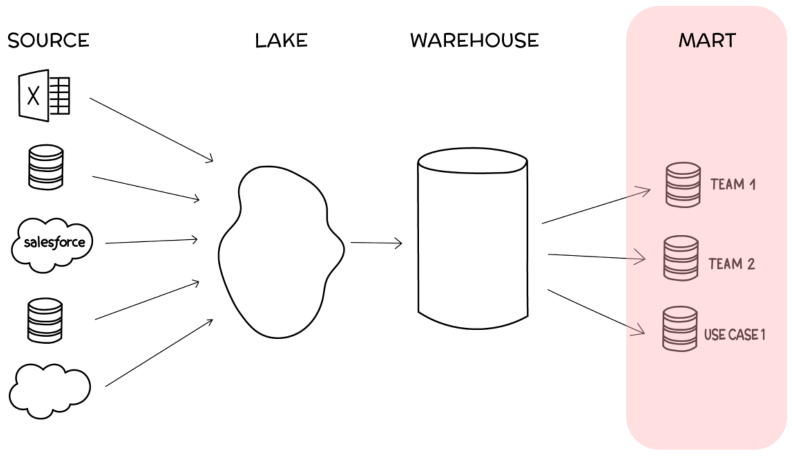


This stage—creating a data Warehouse—has historically been quite a nightmare, and there are many books written on how best to model your data for analytical processing. But these days it's not that hard—and will not only spare you from having to explain all of your schemas' oddities to new team members, but will also save you as an individual time in having to repeat, edit and maintain your own messy queries.

Stage 4. Marts

When you have clean data and a good BI product on top of it, you should start noticing that many people within your company are able to answer their own questions, and more and more people are getting involved. This is great news: your company is getting increasingly informed, and the business and productivity results should be showing. You're also less worried about integrity issues because you've modeled the data, and you're continually maintaining it to be a clean, clear source of truth.

Eventually, however, you'll have hundreds of tables in that source of truth, and users will become overwhelmed when trying to find the data that's relevant to them. You may also discover that, depending on the team, department, or use case, different people want to use the same data structured in different ways. For these reasons, you'll want to start rolling out Data Marts.



Data Marts are smaller, more specific sources of truth for a team or topic of investigation. For example, the Sales team may only need 12 or so tables from the main Warehouse, while the Marketing team may need 20 tables—some of them the same, but some different.

About this Book

Who this book is for

This book is for anyone looking to setup an effective, modern (typically cloud-based) data stack that will truly enable a company to explore and understand the data it collects to have high visibility into their business. It's for people who value their data and realize that a company that is truly informed by their data has significant competitive advantages.

It applies to teams setting up either Centralized or Democratized data workflows with an encouraging bias toward Democratized.

Who this is NOT for

At the moment, this book is not focused on the extreme end of what you might call *Big Data*. As a general guideline for where that might cut off, we'll call that teams collecting more than 100GB of data per day. Those teams may still find this book generally useful (especially for the majority of their tables that are usually significantly less than the event streams), as long as they appreciate that the details and recommendations for implementing a data stack at that scale aren't covered here.

Our eventual goal is to create "Big Data" section inserts throughout the book that will go into the details that those working with enormous datasets will also have to know, while not cluttering the book for the 99% use case.

This book is also not dealing with *AI workflows*, or *realtime operational use cases*. It is purely to build and maintain a reporting and analysis data stack.

Lastly, this book is not written for long time *data experts, vendors and thought leaders*. If it were, we would go into more detail on the pros and cons of each choice and defend why we chose each recommendation. Doing so would greatly clutter the book ruining the experience for those who really need this book and are trying to understand a clear path to a quality data stack.

We instead encourage continued discourse on what constitutes modern and quality data governance practices on blogs, twitter and our Data School slack channel.

How to read this book

The book is structured in 4 stages of sophistication: Sources, Lake, Warehouse and Mart. This is the journey we see companies go through as they grow and their data needs become more sophisticated. Each section starts with a quick excerpt describing when your company may be a good fit for staying at that stage, and when your company should move on to the next stage.

If you find you're already at a later stage, you may have a data lake already setup for instance, you may want to skip ahead to that section. Note though that many best practices are discussed at each stage and are brought up in the stage where they first start to be relevant. It will be assumed that these are already known at the later stages and they won't be repeated. So it may benefit you to at least skim those earlier stages even if you and your company are further ahead.

Disclaimers

While the goal of this book is to be community driven, the initial structure and version was written mostly by the team at Chartio. We don't pretend to have an unbiased view of the world. We've made attempts to use and mention Chartio sparingly - but where BI examples are called for we've used our product as the example. We've also made attempts to be open about our biases - but we may not have caught everything, please let us know if you see any blindspots in the book.

That our experience for this book has been gained by working with well over a thousand modern companies over almost decade, is a great advantage. That the majority of these experiences are working with our own customers, setting up and growing data stacks to be specifically used with our unique product has undoubtedly left some blind spots in the book for those using other products.

Influences

The main books on setting up data stacks are over a decade old, most being pre-cloud and especially pre C-Store warehouses. The state of modern data governance today is largely self-taught and unguided, a reality which is what lead us to write this book.

This book, starting from Chartio, is primarily influenced by first hand experiences and directly working with hundreds of modern cloud-based customers over a decade in BI. The second largest influence is working with our many vendor and consulting partners. Third, are the many community shared posts from a lot of different companies on blogs like Tristan Handy's and in data slack channels. And lastly, it's worth noting some classic books that have influenced (sometimes negatively but mostly positively) the data community.

- Agile Data Warehouse Design by Lawrence Corr
- The Data Warehouse Toolkit by Ralph Kimball
- Information Dashboard Design by Stephen Few - (my review here)

How this book was written

Community driven books aren't that common and we found it was important to setup some guidelines in writing this book:

Single author voice/experience - To avoid the potentially jarring and unnecessary context switches of changing authors each chapter and even throughout each chapter an attempt is made to speak with a single narrator.

Heavily edited - To keep that voice and a flow to the book, it undergoes continually heavy edits. At this point it is still a bit chunky and we're working continually to improve that. We'd love any help with that as you read through (see How to contribute).

No new vocabulary - There is already a lot of jargon in the data world, often created by very talented vendor marketing teams. We try to stick with the most common and simple words here that are already in use.

Stay modern - There are a lot of books for the old way, on old stacks, to work with data. We're defining the current best practices here so we just explain those and forget about the past. In a few cases where it is beneficial to talk about a modern change like ETL moving to ELT we simply teach the ELT in the book and have a chapter in the extras section discussing the choice.

Share examples - Right now this book is fairly light on specific examples. We'd love to add more as we know this is one of the most effective ways of learning. If you have stories to share - do reach out.

It's okay to have an opinion - Almost every part of this book could be contentious to someone, in some use case, or to some vendor. In writing this book it is tempting to bring up the caveats everywhere and write what would ultimately be a very defensive and overly explained book. This would be bad and a bummer to read and way less useful for people reading this book for the advice. Where we have a strong opinion we don't argue it, we just go with it. Where we think the user has a legitimate choice to make - we pose those options.

Stay out of the weeds - This book is intended to be a broad overview and general guideline of how to setup a data stack. We intentionally don't get into the weeds of setting up a Redshift instance, or how to use various BI products. That would clutter the already quite extensive book and repeat a lot of work that is already on the internet.

Stay general - Every company and data use case is a unique snowflake. This book tries to write to the needs of the 95+% of snowflakes. That last 5% is a long tail, and really impossible to cover. Doing so would incredibly clutter the book. Here we stay general and expect readers to be confident enough to deviate from our recommendations when they don't fit them.

How to contribute

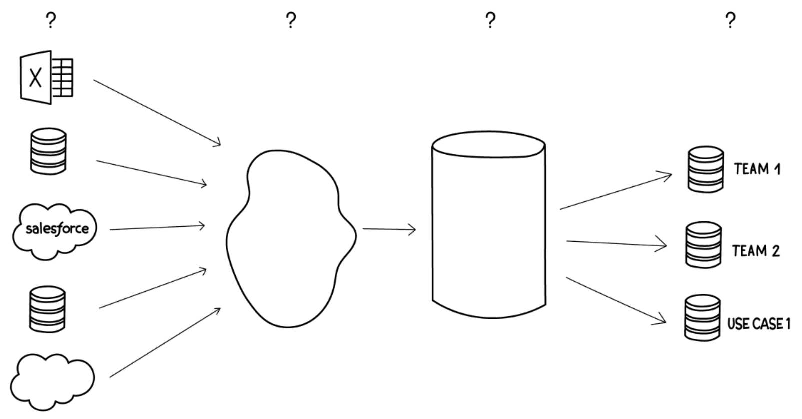
This is an open, community driven book created by many different people. It is hosted on github, and we welcome any and all contributions. Our goal is to continually make this book better and kept modern. We would like to continually expand it, like a wiki, to cover more topics, go more in depth, share more real company examples, and be better reviewed and edited.

Few are complete "experts" in all of the areas of modern data governance, and the landscape is changing all of the time. If you have a story to share, or a chapter you think is missing, or a new idea - email us or create a pull request with the edits on our github repo.

Even if you don't know what specifically to share, but you don't mind sharing your story - please reach out as we are particularly interested in adding specific examples from specific companies as they can be the most powerful method of learning.

What Stage are You at?

Here is a quick guide to finding what stage your team is at. Use the links to jump to that section or view each stage in succession to learn about applying data governance practices at any level.



Stage 1 - Source

Right for you if:

- You have a small team with only a few people using data
- You have minimal data needs at the moment
- You only have data in a few small sources
- The only people who need to make new visuals are fairly technical

You've outgrown if:

- You have data you need access to in multiple places/applications
- You need unique or combined charts/dashboards for cloud application sources like Salesforce
- More than just a few people need access to this data
- You're struggling with performance issues
- You have a set of data that's getting too big for a transactional database
- Non-technical users need to create their own charts

Stage 2 - Lake

Right for you if:

- You need unique or combined charts/dashboards for cloud application sources like Salesforce.
- Your charts and dashboards will be created by a core set of people who will all be able to learn the ins and outs of the structure of the messy data.
- You're intimidated by data modeling (but just don't be - that's why we have this book).
- You cannot spare the time for even light data modeling and are okay, for now, with the technical debt you're taking on.
- You have large sets of data and need more performant queries.

You've outgrown if:

- More than a few people are going to be working with this dataset.
- You want a clean source of truth of your company.
- You don't like fighting with integrity issues.
- You need to separate the structure of the data from the always changing transactional sources.

- You Don’t like Repeating Yourself (DRY)

Stage 3 - Warehouse

Right for you if:

- More than a few people are going to be working with this dataset
- You want a clean source of truth of your company
- You don’t like fighting integrity issues
- You need to separate the structure of the data from the always changing transactional sources.
- You Don’t like Repeating Yourself (DRY)

You’ve outgrown if:

- You want to get democratized - and enable others in your company to explore and understand data themselves
- You’re prepared to teach and enable business users in your company - hopefully using the many resources of the Data School
- You have projects that require different formats of the source of truth for easier use
- Having truly informed employees is important to your company’s competitive success

Stage 4 - Mart

Right for you if:

- You want to get democratized and enable others in your company to explore and understand data themselves
- You’re prepared to teach and enable business users in your company - hopefully using the many resources of The Data School
- You have projects that require different formats of the source of truth for easier use
- Having truly informed employees is important to your company’s competitive success

You’ve outgrown this stage if:

- You can’t really! You can make any number of marts, and even put leveling in your marts if you’d like. Implementing this stage will result in a complete, well architected and governed stack that will continually evolve and support your informed competitive company.

Stage 1 - Source

SOURCE
salesforce

Starting with Source Data

Modern businesses generate tons of data. Product information, customer information, app performance, marketing expenditures, etc. You need to start organizing and analyzing data in order to run your business effectively. At the beginning of a business or while a business remains small, it is sufficient to work with data from production databases, product APIs, and financials directly from their original sources.

This stage is ideal for new companies or teams with minimal data needs. It is inexpensive and relatively easy to tool, implement, and maintain. It is sometimes exciting to build out a sophisticated data stack, but be sure to start here and check that it satisfies your needs before moving on; over-engineering is a costly mistake.

This stage is right for you if

- You have a small team with only a few people using data
- You have minimal data needs at the moment
- You only have data in a few small sources
- The only people who need to make new visuals are fairly technical

You’ve outgrown this stage if

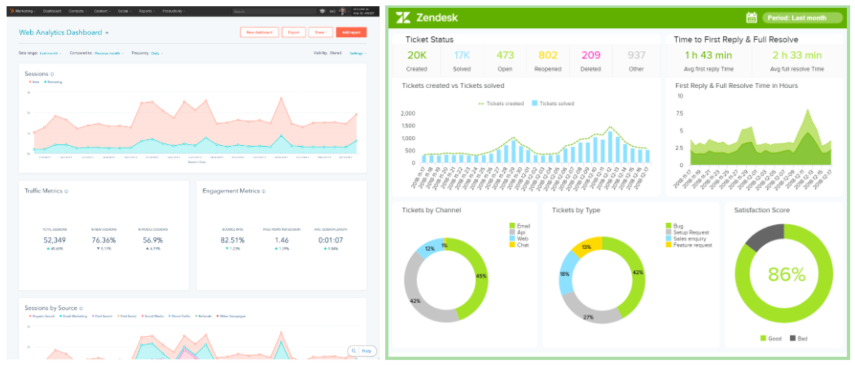
- You have data you need access to in multiple places/applications
- You need unique or combined charts/dashboards for cloud application sources like Salesforce
- More than just a few people need access to this data
- You’re struggling with performance issues
- You have a set of data that’s getting too big for a transactional database
- Non-technical users need to create their own charts

Tools to Analyze Source Data

Data will live in many different places but the methods to analyze them boil down to Application Dashboards, Excel, SQL IDE, Cloud dashboarding tools, Business Intelligence (BI).

Application Dashboards

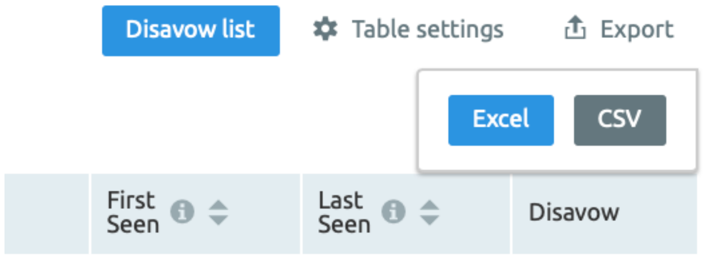
Many modern SaaS applications come built with the same set of fixed dashboards and visualizations to showcase the data they are capturing. These charts are highly tuned to specific use cases and can be quite informative - and maybe all you really need. Some, like Salesforce, even have customizable chart and dashboard creators built-in for high flexibility. These can take you a long way, especially if you don’t need to see this data in combination with other data.



These are also usually well supported by the vendor’s support staff should you have any questions or extra data needs.

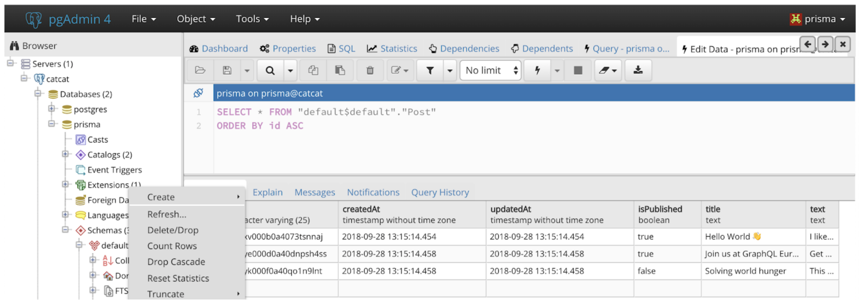
Excel

Many of the applications you are using will allow you to export some of your data into CSV formatted files. You can take this data and open it in Excel to analyze it. While this is an effective way to expand the questions you can ask of the data it is fairly manual and will need to be updated with new data often.



SQL IDE

For data sources such as your production database, you can query it directly from the command line but this can get messy and hard to keep track of queries and results. We suggest setting up an IDE such as PG admin to better handle querying data within a Schema.

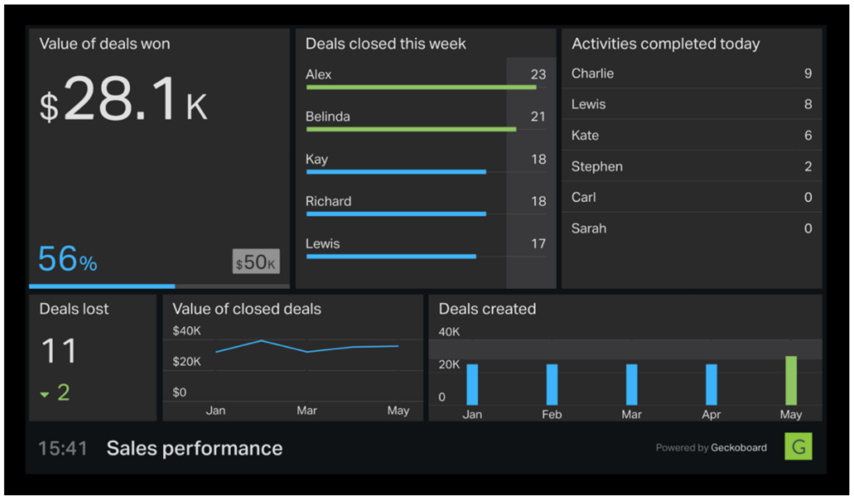


```sql
SELECT * FROM "default$default"."Post"
ORDER BY id ASC
```

acter varying (25)	createdAt timestamp without time zone	updatedAt timestamp without time zone	isPublished boolean	title text	text text
xv000b0a4073tsnnaj	2018-09-28 13:15:14.454	2018-09-28 13:15:14.454	true	Hello World 👋	I like...
ye000d0a40dnpsh4ss	2018-09-28 13:15:14.458	2018-09-28 13:15:14.458	true	Join us at GraphQL Eur...	Get ...
yk000f0a40qo1n9lnt	2018-09-28 13:15:14.458	2018-09-28 13:15:14.458	false	Solving world hunger	This ...

Cloud dashboarding tools

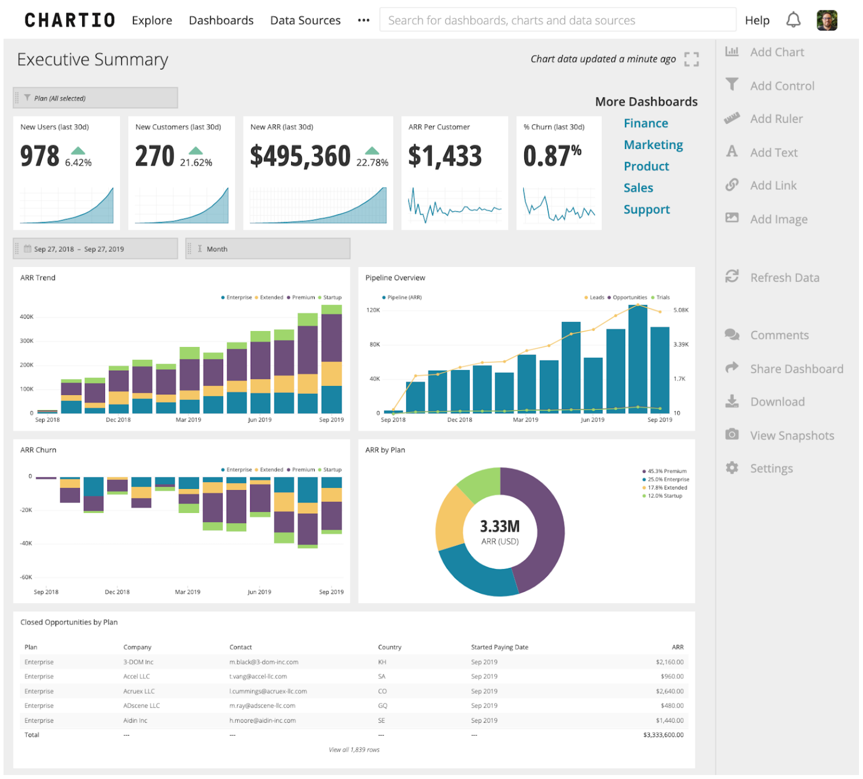
Tools like Geckoboard or Grow allow you to bring in data from cloud applications via APIs to visualize data in simple ways. This allows you to combine data from multiple places into a single dashboard, with simple visualizations to be viewed.



They often come with some nice default dashboards set up automatically for you, and can look great and be great for showing on TVs around your office, keeping teams aware of what's going on. They will be limited in the ease and possibilities of their customizations.

Business Intelligence

For the most power in working with source data, use a flexible self-service Business Intelligence (BI) solution. For this stage be sure to choose an agile product, that allows direct SQL queries when necessary and ideally the ability to connect to and blend data from multiple sources. Those features will be necessary as your data hasn't been consolidated into a single source yet, nor has it been organized in a clean enough way to not need occasional power of a complex SQL query. We're obviously biased, but Chartio is a great choice here.



Closed Opportunities by Plan

Plan	Company	Contact	Country	Started Paying Date	ARR
Enterprise	3-DOM Inc	m.black@3-dom-inc.com	KH	Sep 2019	$2,160.00
Enterprise	Accel LLC	t.vang@accel-llc.com	SA	Sep 2019	$960.00
Enterprise	Acruex LLC	l.cummings@acruex-llc.com	CO	Sep 2019	$2,640.00
Enterprise	ADscene LLC	m.ray@adscene-llc.com	GQ	Sep 2019	$480.00
Enterprise	Aidin Inc	h.moore@aidin-inc.com	SE	Sep 2019	$1,440.00
Total	---	---	---	---	$3,333,600.00

View all 1,839 rows

Summary

Every business has access to data, you need to find how best to view it and analyze it to improve your business. While you can use Application Dashboards, Excel, SQL IDE, or Cloud dashboarding tools; We recommend using a self serve Business Intelligence product to work with a variety of sources at once and be able to write SQL against your database.

Source Data Connections

When a business is getting started with data, people are analyzing it in live systems. While this is ok for tools like Salesforce or Google Analytics, we need to take separate precautions for data in a database.

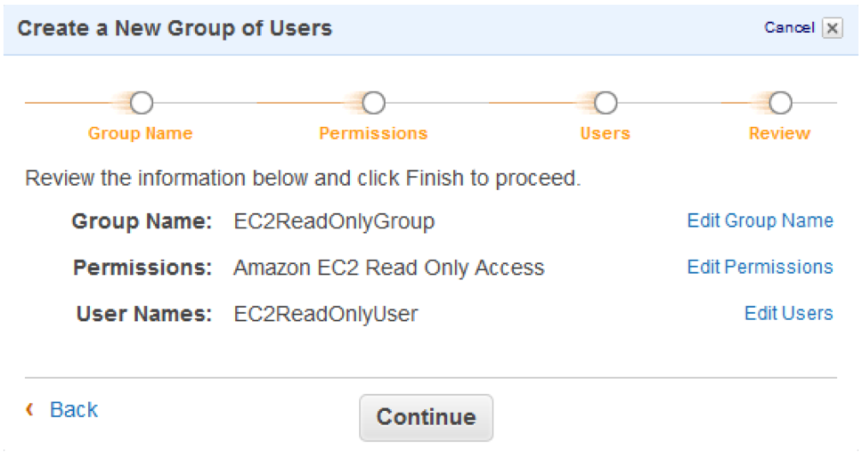
Database Connections

To use any source data in a database you'll want to create:

1. Read-only Access Users (be careful)
2. Read-only Replica

Read-Only User

Create and use a read-only user account to analyze data on your production database. This will prevent users from accidentally making any updates to the data during your analysis, granted this is unlikely but is still a good precaution. It also makes it possible to grant other people access for analysis purposes and guard against their errors.



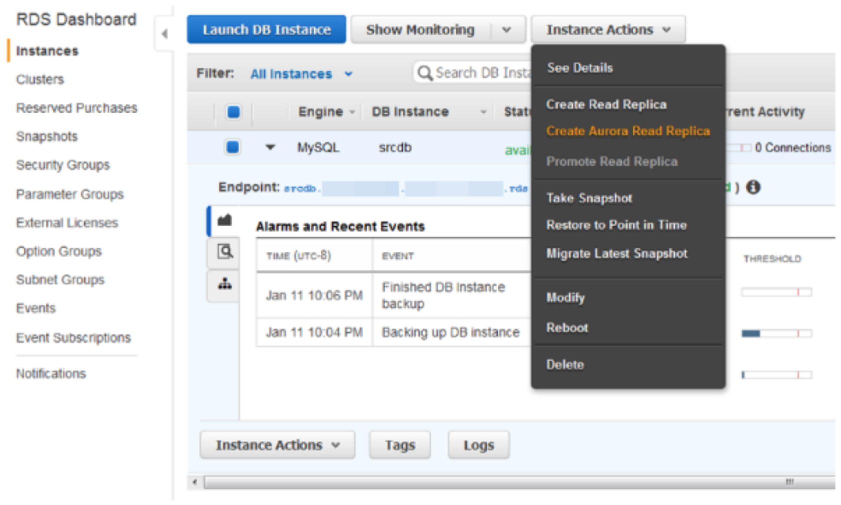
Review the information below and click Finish to proceed.

Group Name:	EC2ReadOnlyGroup
Permissions:	Amazon EC2 Read Only Access
User Names:	EC2ReadOnlyUser

This functionality exists across database providers. However this will effect your app's performance so it is best to separate your analytics from your application.

Read-Only Replica

To query the data without impacting the performance of your application, create a read-only replica of the production database. This creates a copy of your data in a new database which can be queried without concern.



While creating a read-only replica is easy if you're using cloud providers with hosted databases like RDS, it can be hard on other platforms.

These databases may double the cost of your database spend, but they remove the risk of an analytic query affecting your application.

Summary

Even small teams should set up permissions to analyze data:

1. Create Read-Only Users for analyzing data in a database (be careful of impacting the apps performance)
2. Create a Read-Only Replica database to perform analytical queries while removing the performance impacts to the production database.

Source Data Best Practices

If you do get to the stage where you are utilizing a SQL-based BI tool to blend and analyze your data, you should be aware of some helpful best practices. At this stage, with few people exploring the data, it is not recommended to spend a lot of time doing modeling or clean up. However, the following agile best practices will prove helpful.

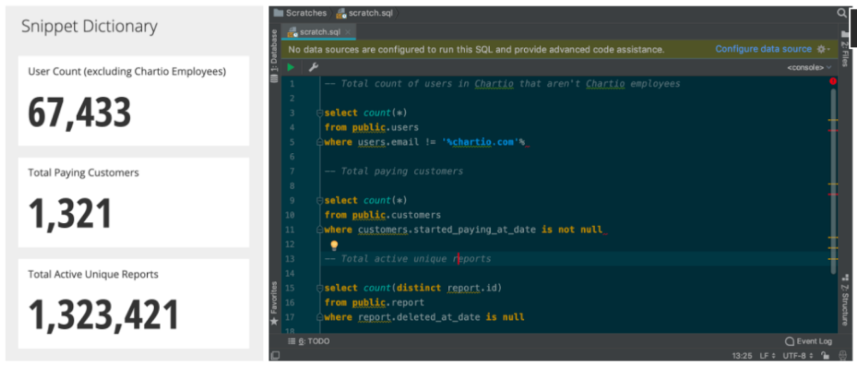
Keep a complexity wiki page

Instead, keep a document or wiki page with warnings about known complexities in the data so that you and others who are training-up can reference and share knowledge. Some example things to keep track of are:

- Descriptions of poorly named columns and tables
- Columns with nulls or largely useless data
- Business logic
- Complex or confusing join paths
- Old or unused columns and tables

Snippet Dictionary

Another useful document to keep is a dictionary of base queries or snippets for common metrics. When working with data that hasn't been modeled yet, you end up having to repeat a lot of the same filters over and over again. It's helpful to be able to grab those easily when needed, rather than re-create them each time.


Snippet Dictionary

User Count (excluding Chartio Employees)
67,433

Total Paying Customers
1,321

Total Active Unique Reports
1,323,421

No data sources are configured to run this SQL and provide advanced code assistance.

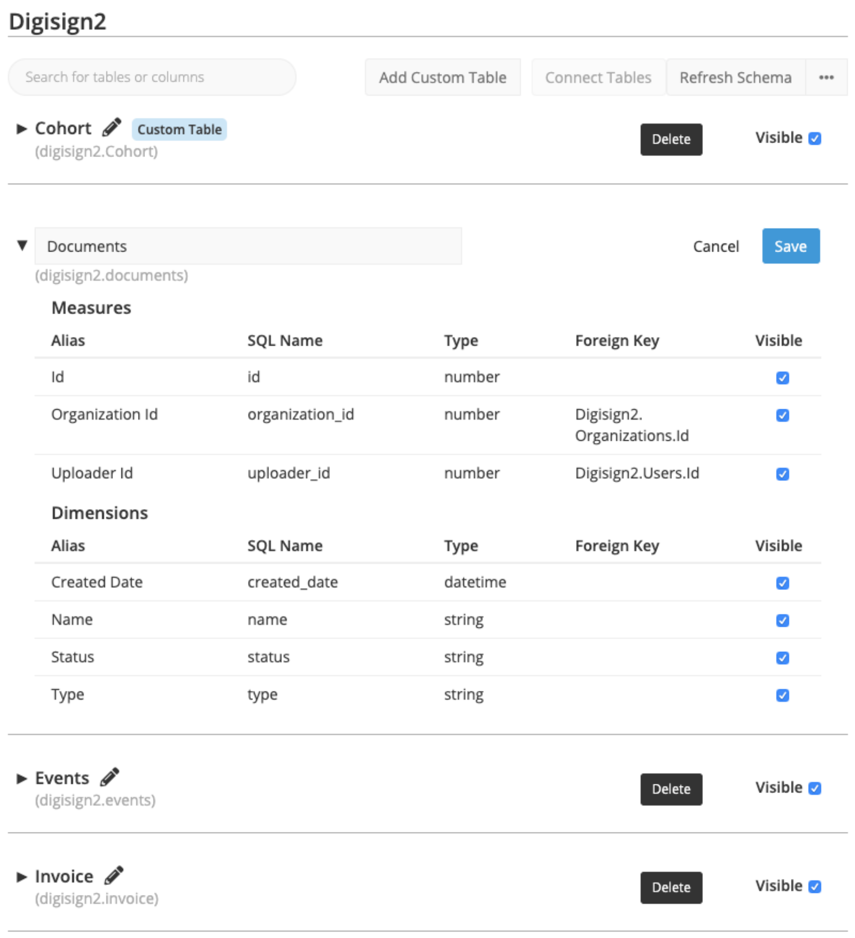
```sql
-- Total count of users in Chartio that aren't Chartio employees

select count(*)
from public.users
where users.email != '%chartio.com'%

-- Total paying customers

select count(*)
from public.customers
where customers.started_paying_at_date is not null

-- Total active unique reports

select count(distinct report.id)
from public.report
where report.deleted_at_date is null
```

These can be stored on a wiki, .sql file, or even as a dashboard of these queries saved as charts. With the right BI product, the dashboard method is ideal, as these snippets are rendered as starter charts that can be easily duplicated and adjusted.

Along with the snippets, it is a best practice to add comments describing how the metric was calculated. These snippets and comments will be a starting point for data models at the Warehouse stage of data sophistication.

BI Layer Meta Modeling

Agile BI products will allow you to do some light modeling of the data at the BI layer. In Chartio for instance, there is a Schema Editor that enables you to quickly rename fields, hide columns, specify join paths, and create custom columns or tables. Time here can be well spent, but if you're doing too much of it, consider moving all the way forward to the Warehouse stage where your efforts will be more universally applied.


Digisign2

▶ Cohort
(digisign2.Cohort)

▼ Documents
(digisign2.documents)

Measures

Alias	SQL Name	Type	Foreign Key	Visible
Id	id	number		☑
Organization Id	organization_id	number	Digisign2. Organizations.Id	☑
Uploader Id	uploader_id	number	Digisign2.Users.Id	☑

Dimensions

Alias	SQL Name	Type	Foreign Key	Visible
Created Date	created_date	datetime		☑
Name	name	string		☑
Status	status	string		☑
Type	type	string		☑

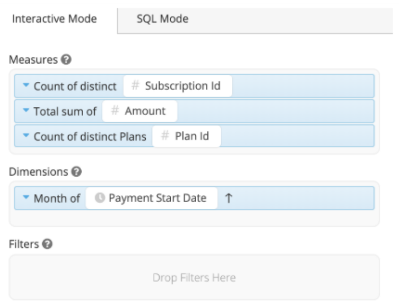
▶ Events
(digisign2.events)

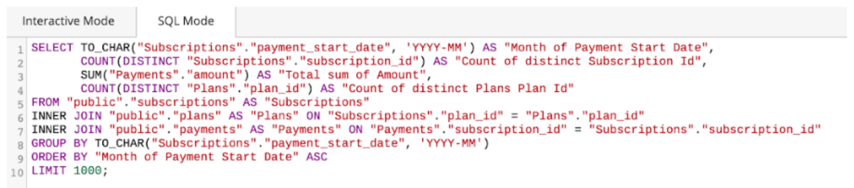
▶ Invoice
(digisign2.invoice)

If you're using Tableau, they have a feature called Data Interpreter that will do this as well.

Visual, Drag & Drop Exploration

In addition to SQL access, some BI products with drag and drop data explorers such as Tableau Desktop or Chartio's Visual SQL will handle writing the SQL for you. Generally exploring data visually is much faster and more intuitive than writing raw SQL. Visual interfaces will handle changing data format strings, join paths, new groupings, unique dialects, etc. which will automatically you a lot of googling and debugging.





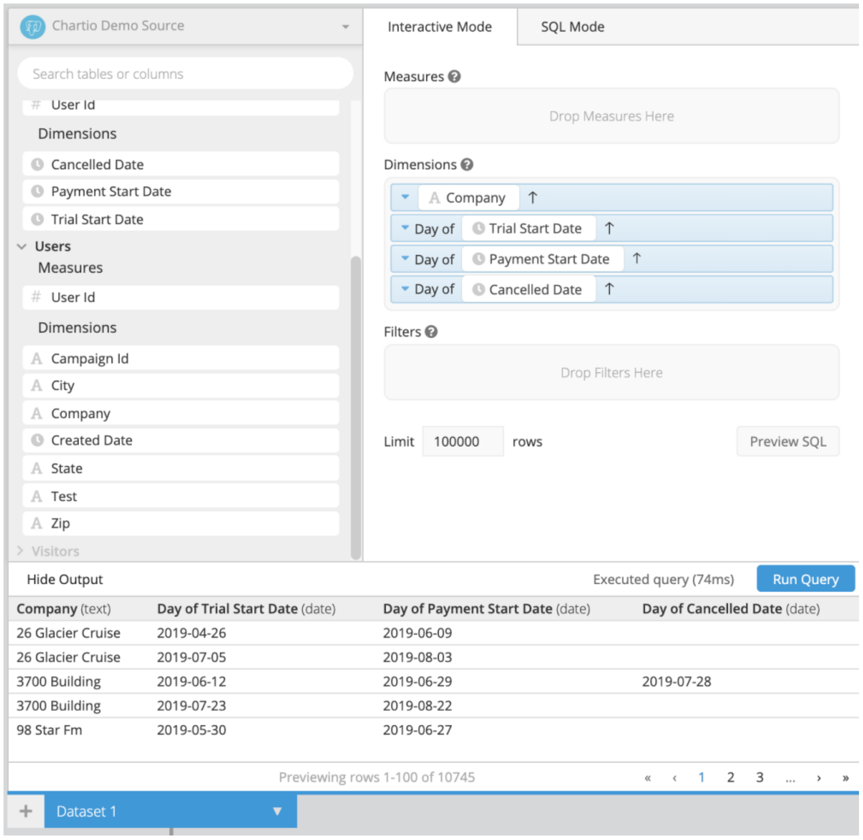
```sql
SELECT TO_CHAR("Subscriptions"."payment_start_date", 'YYYY-MM') AS "Month of Payment Start Date",
       COUNT(DISTINCT "Subscriptions"."subscription_id") AS "Count of distinct Subscription Id",
       SUM("Payments"."amount") AS "Total sum of Amount",
       COUNT(DISTINCT "Plans"."plan_id") AS "Count of distinct Plans Plan Id"
FROM "public"."subscriptions" AS "Subscriptions"
INNER JOIN "public"."plans" AS "Plans" ON "Subscriptions"."plan_id" = "Plans"."plan_id"
INNER JOIN "public"."payments" AS "Payments" ON "Payments"."subscription_id" = "Subscriptions"."subscription_id"
GROUP BY TO_CHAR("Subscriptions"."payment_start_date", 'YYYY-MM')
ORDER BY "Month of Payment Start Date" ASC
LIMIT 1000;
```

Especially at this source stage, you may be dealing with sources in multiple SQL dialects, and a few different APIs. Remembering the syntaxes for all of these sources is just not realistic. One of the largest benefits of these visual interfaces is that they operate in the same way, regardless of the source, and write the correctly formed SQL for you.

These interfaces will be even more useful on top of clean/modeled data in the warehouse and mart stage where you'll enable more people in your company to explore data and answer questions on their own.

Double Check Results

Whenever you are producing visualizations, unexpected or null values can make your analysis incorrect. Always do a quick review of the raw data by sorting each field to see if you need to remove, ignore, or update nulls/outliers so that your analysis is correct.



Dimensions: Company ↑; Day of Trial Start Date ↑; Day of Payment Start Date ↑; Day of Cancelled Date ↑

Limit 100000 rows

Executed query (74ms)

Company (text)	Day of Trial Start Date (date)	Day of Payment Start Date (date)	Day of Cancelled Date (date)
26 Glacier Cruise	2019-04-26	2019-06-09	
26 Glacier Cruise	2019-07-05	2019-08-03	
3700 Building	2019-06-12	2019-06-29	2019-07-28
3700 Building	2019-07-23	2019-08-22	
98 Star Fm	2019-05-30	2019-06-27	

Previewing rows 1-100 of 10745

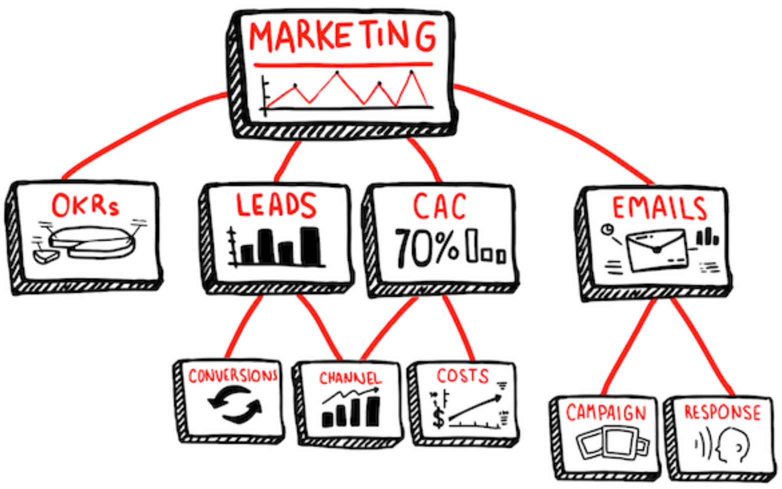
« ‹ 1 2 3 ... › »

Read this post on Finding Outliers with SQL

Keep short Dashboards

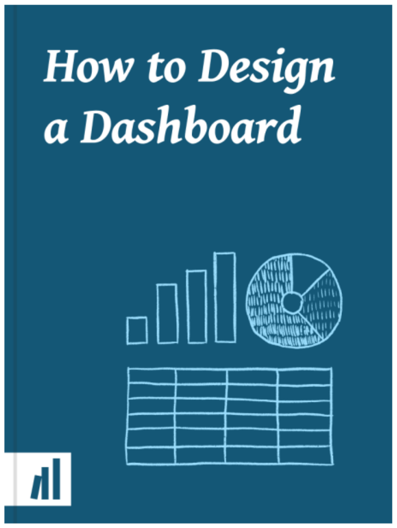
We've noticed that people have a tendency to keep adding more and more charts to existing dashboards. We get it, when you're finding one insight at a time, it never feels like the right time to make a totally new dashboard. The result however is that you end up with really long, disorganized dashboards that run queries for each chart each time.

With a long dashboard, simply checking in on your dashboard to get the latest on one or two of your key charts is going to kick off hundreds of queries. We've written more regarding the best practices of keeping short dashboards here. In short, though, it's best to organize them in smaller groups and create quick links between them.



Design before building

Another common pitfall is starting a dashboard by exploring data first. When you do this, you (while exploring) end up with a collection of what you thought were interesting facts but are in reality probably quite useless in daily monitoring. We've written a whole other book on this pitfall entitled How to Design a Dashboard.



In short - when creating key dashboards that you'll be continually monitoring, first spend some time with pen and paper outlining exactly what you want to see and how. It will save you a lot of time and result in a considerably more useful dashboard.

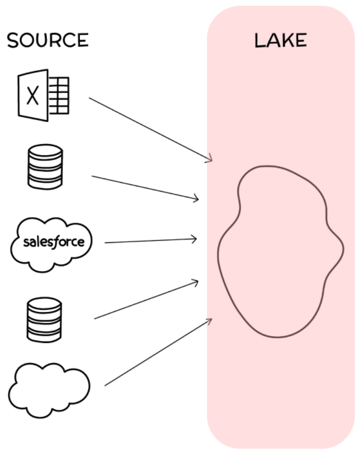
Summary

Analyzing source data can be tricky since it has not been cleaned or modeled. That said, these best practices make it easy

- Keep a short, regularly updated snippet dictionary
- Use BI products to overcome data issues such as weird field names and complex joins
- Double check the data before visualization
- Keep shorter dashboards
- Design dashboards with pen and paper first
- Keep an eye on query performance if you are hitting the production database

Lastly, you should start exploring tools to pipe-in multiple sources of data that are not handled by your SQL-based BI tool to build your Data Lake.

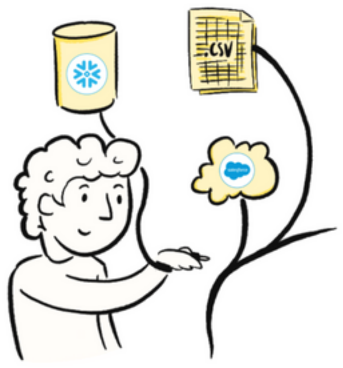
Stage 2 - Lake



Why Build a Data Lake

What is a Data Lake?

A Data Lake is a storage repository of multiple sources of raw data in a single location. In the cloud these are typically stored in cloud c-store data warehouses or in S3 buckets. The data can be in a variety of formats and can be structured, semi-structured, unstructured, or even binary.



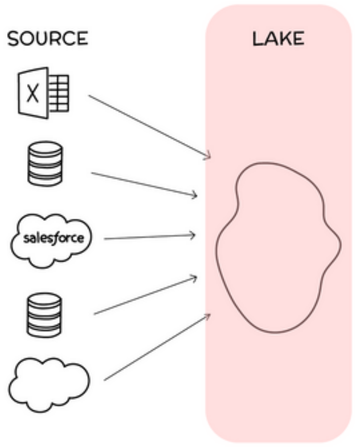
The term Data Lake, after oil lakes (pre-refinery oil), was created to contrast the term Data Mart which described orderly, siloed, and refined data. Having all the data in one place made it easier to work with large data sets and start gaining insights earlier in the data modeling process.

This stage is right for you if:

- You need unique or combined charts/dashboards for cloud application sources like Salesforce.
- Your charts and dashboards will be created by a core set of people who will all be able to learn the ins and outs of the structure of the messy data.
- You're intimidated by data modeling (but just don't be - that's why we have this book).
- You cannot spare the time for even light data modeling and are okay, for now, with the technical debt you're taking on.
- You have large sets of data and need more performant queries.

You've outgrown this stage if:

- More than a few people are going to be working with this dataset.
- You want a clean source of truth about your company.
- You don't like fighting with integrity issues.
- You need to separate the structure of the data from the always-changing transactional sources.
- You Don't like Repeating Yourself (DRY)



Top 4 reasons to build a Data Lake

1) It's unifying

As your data needs expand it becomes harder and harder to work with data kept in multiple different silos. It may make sense from a product perspective for your traffic data to be in Google Analytics, your sales records to be in Salesforce and your trial engagement data to be in some database. However, when you need to analyze your funnel and attribution models you need them all together.

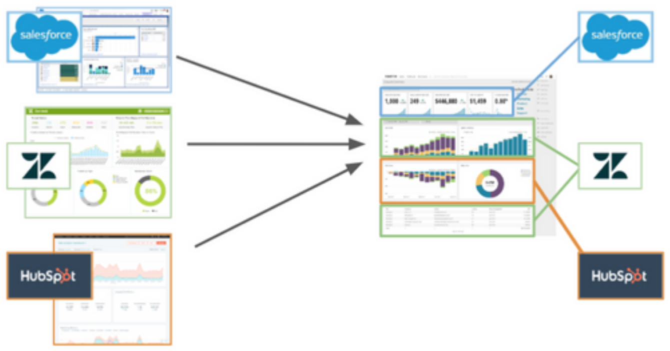
In the source stage, we discussed blending options, but because blends load all pre-join results into the BI product these are extremely limited in how much data can be joined and are not a scalable solution.

In a Data Lake, all data can be combined so it can be analyzed together. This makes gaining insights easier and provides more depth for data exploration.

2) Full query access

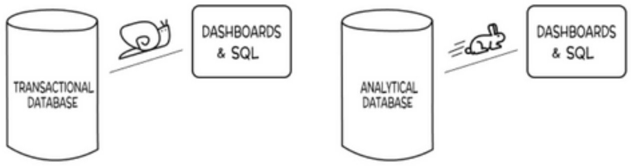
The applications your business uses likely only offer transactional API access to the data. They're not designed for reporting, so unless the data is exported and put into a format you can easily query, you will end up being very limited in what you can pull. These APIs, if used directly for reporting, can also become prohibitively expensive.

If you extract that API data with an ELT product and load it into a Data Lake, you'll have all the power and flexibility of SQL or whatever BI product you use - and the cost won't go up considerably with each chart.



3) Performance

Source data might be from the actual production database which could affect the performance of the application that it is powering. Queries that demand a lot of data such as aggregations are not optimally run on transactional databases.



Data Lakes are built to handle these types of ad hoc analytical queries independently of the production environment. You can scale up resources on a Data Lake to be able to query data even faster.

4) Progress

Getting the data in one spot is a necessary step for progressing to the other stages. It makes working with data so much easier that many BI products require this stage - as they will only connect to a single warehouse source.

In the Warehouse stage, you'll be able to implement proper modeling on top of your Lake. Through modeling the data will be cleaner, which enables more people to use it, causes fewer errors, and creates less repetition of work.

What Engine to Use For a Data Lake

In order to build a Data Lake, we need to choose a database to host it on. Historically, and still today at massive (> 100GB/day) scale, the Lake was stored in a file system like S3 buckets.

Today, with storage being so cheap and warehouses being so scalable, we recommend putting your lake data directly into what is called a Warehouse Engine. This will make creating the Data Warehouse much simpler as we'll cover once we get to that next stage.

What is a Warehouse Engine?

In 2005 a combined group from Brown University, Brandeis University, and MIT released a ground breaking paper know as the C-Store paper introducing a new column store architecture. The many developments in that paper led to a new class of cloud based databases that can very powerfully handle large sets of data.

These engines are geared toward analytic workloads that require larger, but less frequent queries than their transactional counterparts. Transactional databases like PostgreSQL are optimized to do quick reads and writes at incredibly high volumes in order to run the applications that they serve. Analytic use cases query data way less frequently, but their queries are usually more complex and over larger sets of data.

If these are vehicles, transactional databases are motorcycles capable of many quick trips while warehouse engines are semis doing fewer trips but hauling large loads.

Deciding factors

The biggest decision to make when moving from production to a lake is what database you will use. Most people consider:

- Pricing
- Ecosystem
- Performance/Scale
- Maintenance

There are a variety of database pricing models, from being based on storage to being based on the amount of data queried. If your company is strictly using Amazon or Google as your software vendors, this can dictate your vendor choice as well.

The architecture of the Data Lake has implications on how it'll help your operations scale. Differences in the many types of lakes entail columnar vs. row-oriented storage, and having storage and compute together or separated. If there are requirements for ongoing maintenance of your Data Lake you will want to know that as well.

When selecting the right data engine for your organization, you may also consider whether you want an on-premise or cloud solution. More and more businesses are moving to cloud solutions to take advantage of the "as a service" model and save on hardware costs so, we'll focus on cloud databases in this section.

Modern Warehouse Engine Products

Today, there are three dominant choices for cloud based data warehouse engines: Amazon Redshift, Google BigQuery and Snowflake. Note - all of these are similar and based on the C-Store paper.

Amazon Redshift

Pros

Redshift has the benefits of ease of use, speed, and cost. Being a part of AWS, there is full service integration for the wide range of AWS services such as S3 for storage and CloudWatch for infrastructure monitoring. Redshift is generally cheaper than Snowflake or BigQuery, with a couple of pricing options such as paying hourly per node or paying by number of bytes scanned with Redshift Spectrum. It's simple to set up and scale by adding nodes to your cluster and increasing storage and performance.

Redshift is probably the most popular, although it is losing ground to Snowflake. It benefits from being similar in connection and SQL syntax to PostgreSQL.

Cons

Users can often run into concurrency issues with Redshift if it isn't set up properly or if there are high volumes of queries from many users accessing the database. Ongoing maintenance may be required with Redshift to resize clusters, define sort keys, and vacuum data.

Like many AWS services there are ways to customize your configuration with workload management, compression, and partitioning. But these advanced features are not very out of the box. So although Redshift is powerful it may require a dedicated resource from your data engineering team.

Google BigQuery

Pros

BigQuery is not bound by cluster capacity of storage or compute resources, so it scales and performs very well with increasing demands for concurrency (e.g. more users and queries accessing the database). As a fully managed database, BigQuery handles vacuums and resizing on its own which can save time for your data engineers and makes it easy to use and maintain. For businesses using Google products, BigQuery integrates well with Google Drive and Google Analytics.

Cons

Cost is determined per query byte, making it difficult to budget or regulate if you have users running ad hoc queries against the Data Lake. To work around this, you can leverage BigQuery's cost controls, but it can still restrict the amount of analysis you can perform because it limits the queries you can run.

Snowflake

Pros

Like BigQuery, Snowflake has an architecture that separates the compute query engine from data storage. As a result, it is highly scalable at any amount of volume and concurrency. Pricing is based on the storage and compute used on a time-basis with their virtual databases instead of per bytes scanned. Tuning, indexes, and distribution keys aren’t required for queries to be optimized and performant. Because of these reasons, it can be said that Snowflake has many of the benefits of both Redshift and Big Query.

Cons

Snowflake is a relatively new database in the market, so if you are familiar with SQL functions supported by databases like Redshift or Postgres you may find some inconsistencies in the SQL syntax. Snowflake is also generally more expensive.

Database Engines

PostgreSQL

Pros

Unlike the options above, PostgreSQL is an open source database that is free to download. It can easily be spun up on your local server or hosted on various cloud services such as AWS. Postgres also has an ANSI compliant SQL library and supports an extensive library of third-party and user-defined functions. As it’s a transactional database, it has very fast writes and also has fast reads below ~100M rows.

Cons

Postgres is a straightforward, flexible solution that’s different from Snowflake, Redshift, and BigQuery because it is a row-oriented database more suited for processing transactional data over analytical queries. It’s a single database connection not architected for parallel processing, so it generally doesn’t perform as well if you have a data volume of over 1 TB. Consequently, Postgres is great as a database, but is not a good choice for a Data Lake if you have a high volume of data (>1TB).

Recommendation

Selecting a Data Warehouse can be dependent on a number of factors that should be considered before making the investment. If you prefer a cheap, straightforward Data Warehouse you may be tempted to go with PostgreSQL, however it will have trouble scaling as a Data Warehouse.

Redshift is a good choice as a standard cloud Data Warehouse if you have the capacity for a dedicated DBA. BigQuery and Snowflake are both highly scalable solutions considering their architecture. However, if cost or concurrency limits will be an issue for you then Snowflake would be more suitable for your organization.

Remember all of these data warehouses are built on the same C-Store architecture so the differences will not be severe in performance. If you'd like a full benchmarking (though the same final recommendation) do checkout Fivetran's awesome warehouse benchmark.

Extract and Load a Lake

To get data into your Data Lake you will first need to **Extract** the data from the source through SQL or some API, and then **Load** it into the lake. This process is called Extract and Load - or "EL" for short.

There are a number of great modern EL vendors (sometimes called ELT vendors) such as Fivetran, Stitch Data or Blendo.

These EL providers built detailed Extract scripts for the most popular API's and offer a simple experience for extracting and loading your data into your data lake. The process usually involves the setup of a pipeline where credentials are given for both the destination and the data source and some configuration where light transformation is performed, e.g. selecting what tables and fields to sync, hiding some values for privacy reasons, etc.

The setup can be performed with minimal engineering effort in most cases.

Extract Options

Extraction is the phase of pulling data from a data source, through APIs or SQL. We can do a complete extraction of all data available, or we can do an incremental extraction every time we run a sync. A complete extraction will extract all the data from the data source. An incremental extraction will only extract updated records from the data source.

Complete Extraction

A complete extraction is the easiest way since no configuration is required but it has two big disadvantages.

1. You end up with a lot of duplicate data in your data lake
2. You increase the complexity of the next steps in your analytics stack

You will have to figure out what data you actually need in the data lake, so it will require more complex logic to do it and more processing.

Incremental Extraction

The preferred alternative is to do incremental extractions. This is more challenging since you need to check for new or updated rows and account for changing schemas. However, it is typically preferred because much less data being processed and fewer updates will need to be made in the data lake. All cloud ELT vendors support incremental extractions from your sources.

The main downside to incremental extraction is deletions in a data source. It's not easy to detect and implement deletions in the general case. ELT providers do not guarantee consistency under deletions in most cases, in some cases it can be done or it is implemented by the source, e.g. data is never deleted but flagged as `is_deleted` instead.

A complete dump would guarantee that you have always an exact replica of the source state. Keep in mind that in analytics this is not important in the general case, but keeping the deleted records might also be something that is required.

Load Options

However you extract data from your data sources, you need to decide how these changes will be reflected on your destination. You can push changes through to existing data in the data lake or you can store this new data separate from existing data.

Push Changes

If you are using a database system as a data lake, then you can update the data with the pushed changes. This will end up having a close replica of the data from the source system to your Data Lake and it optimizes storage.

Store Separate

The other way is to save the changes without updating the records. This is pretty much the only way you can do it if you use a file system and don't want to add a lot of complexity on your data lake. The benefit of doing this is that you have a history of all the changes that happened on your data.

Multiple schemas

Most EL vendors will insert each source into the lake as a new schema or folder if you're on a file system. This is ideal as your data will still be organized by source and there is no chance of commonly named tables overwriting each other.

It will mean that when querying across these schemas you'll need to remember to specify the schema names in addition to the table names.

```
SELECT * FROM "salesforce"."_user" AS "SFuser" JOIN "zendesk"."user" AS "ZDuser" ON "SFuser'
```

Other Extract and Load routes

Traditional ETL

For the reasons we've outlined here we recommend ELT over ETL. But if you still want to do things the traditional way, with Transformations happening before things are loaded into the Lake you can use products like Xplenty or Amazon Glue.

If you do this, you're essentially doing the Lake and Warehouse stages all in one jump and skipping the Lake piece of the recommended stack. That might sound like a great thing, but it's important to note that it won't save you any money or time - in fact it will likely cost more.

DIY

If you value your time, money, sanity and data integrity don't DIY your own EL scripts. If you DIY you will dedicate precious engineering resources to something that can be done at a fraction of the cost and time using a cloud solution. Your data engineers can work on more important data projects related to your overall data infrastructure and product.

Sometimes, you may need to create custom code for a source that's not widely supported. If you do need to, please at least use a framework like Apache AirFlow. The last thing you want is a mess of scripts and cron jobs deployed haphazardly.

Data Lake Security

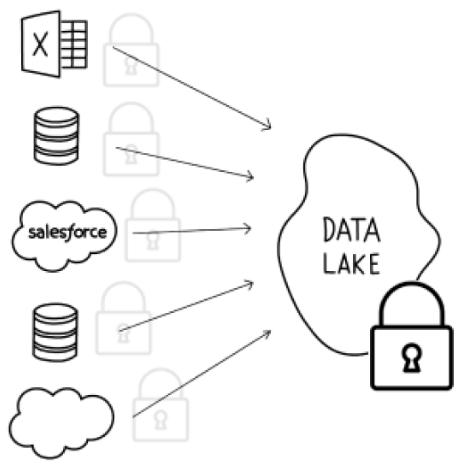
Prior to having a Data Lake, all analysis had to be done by working with data in many different places. These tools required individual logins which are hard to track and maintain the appropriate levels of access.

Data Lakes provide a way to centralize access to data and make it simpler to manage permissions.

Data coming in to the Data Lake is likely not cleaned yet so there may still be sensitive information coming through. Don't Extract or Load sensitive data/columns, this is configurable within the ELT tools. Also be wary of who you grant access to the Data Lake since it gives much more access to all of the data than giving access to a particular source.

Access in central place

Remove access from individual tools, move all access to Data Lake. This will cut down on tickets requesting access and mishaps where people retain access to information they shouldn't have.



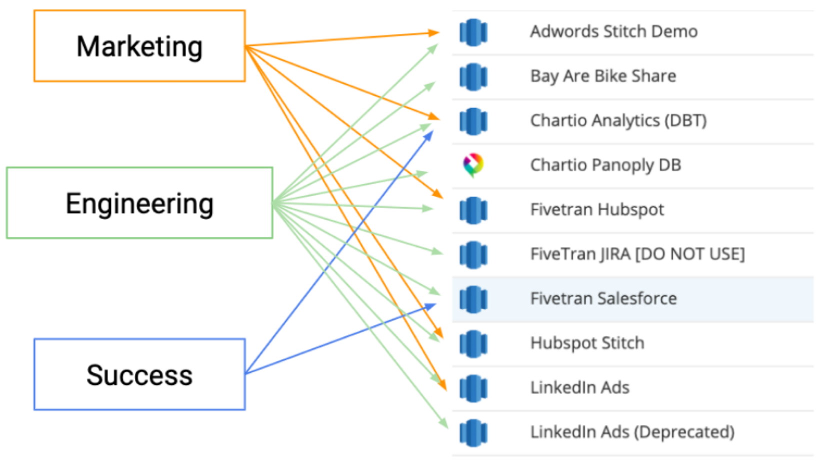
Permission tiers

Now that you can easily grant anyone access to data, you will have more people using that data. While this is advantageous for analysis and exploration you do need to be mindful of what schemas and tables they can see.

Start out by creating two users groups on the Data Lake as follows:

1. For admin and engineers (Full Access)
2. For analysts and business users (Relevant Access)

You can prevent the second group from accessing sensitive data in the Data Lake by limiting that group's permissions to only relevant schemas or tables.



In general set broad controls so it is easy to manage. As you grow in sophistication and turn your Data Lake into a Data Warehouse and Data Mart you can create more refined permissions settings.

Data Lake Maintenance

Data Lakes are inherently not very well organized or maintained. They should be relatively low maintenance but there are two areas that will need some attention.

- Data Sources
- Performance

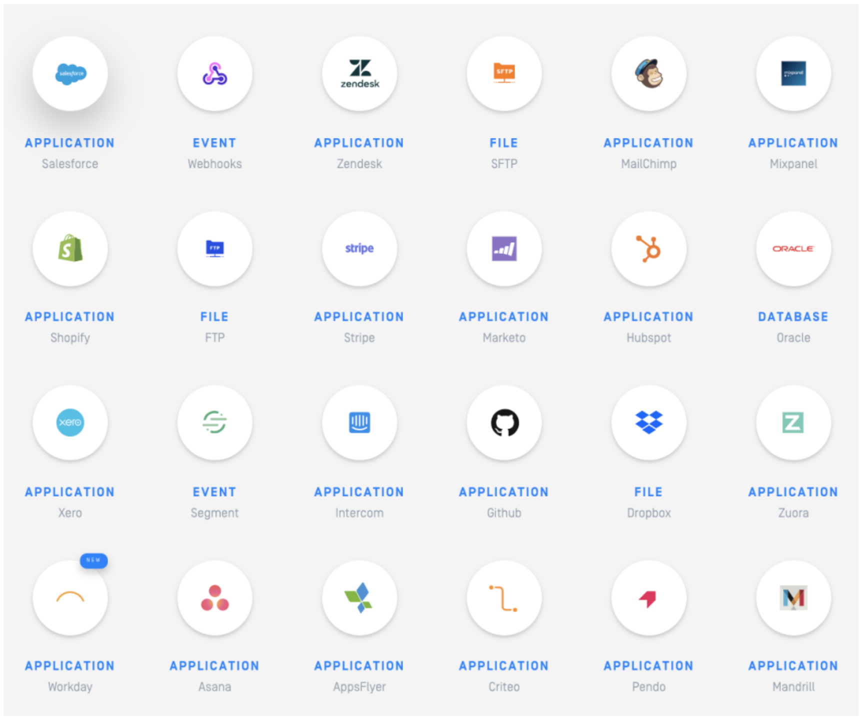
These maintenance activities can be expensive if you extracted and loaded your data with custom scripts. You would need in-depth knowledge of where data is coming from. You would need to know how to work with their API and data structures and potentially have to write a lot of new code when they make an update. Don't Extract and Load manually, use tools like Fivetran, Blendo, or Stitch which will automatically handle these data source updates.

Data Sources

The main place where maintenance issues occur is when the data from the sources changes or the data is not making it from the source into the Data Lake.

Adding new data sources

Ideally, this is as simple as clicking a few buttons inside of an ELT product. Products such as Fivetran, Stitch, and Blendo have large numbers of connectors for different data sources:



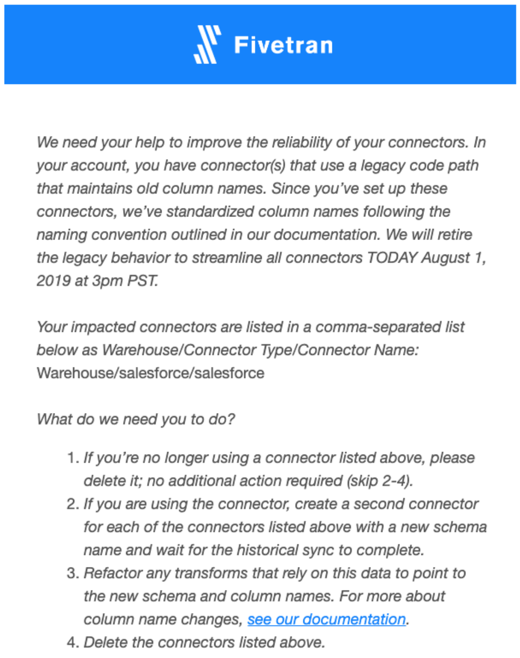
https://fivetran.com/directory

https://www.stitchdata.com/integrations/sources/

https://www.blendo.co/integrations/

Data source updates

Sources change all the time, and ETL tools can handle these for you. This is what they focus on, so they will work to update API calls to make sure the data you're getting is accurate.



Fivetran

We need your help to improve the reliability of your connectors. In your account, you have connector(s) that use a legacy code path that maintains old column names. Since you've set up these connectors, we've standardized column names following the naming convention outlined in our documentation. We will retire the legacy behavior to streamline all connectors TODAY August 1, 2019 at 3pm PST.

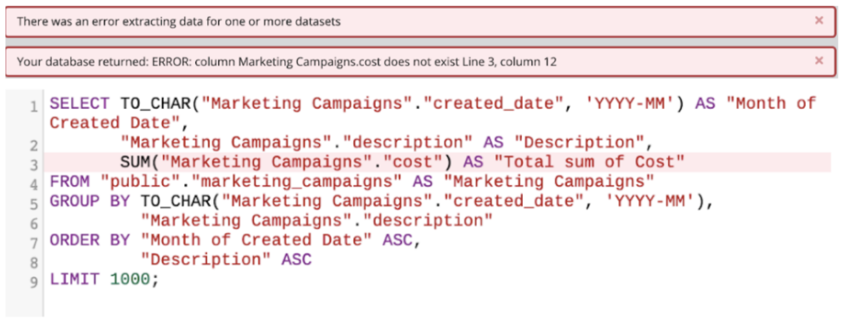
Your impacted connectors are listed in a comma-separated list below as Warehouse/Connector Type/Connector Name: Warehouse/salesforce/salesforce

What do we need you to do?

1. If you're no longer using a connector listed above, please delete it; no additional action required (skip 2-4).
2. If you are using the connector, create a second connector for each of the connectors listed above with a new schema name and wait for the historical sync to complete.
3. Refactor any transforms that rely on this data to point to the new schema and column names. For more about column name changes, see our documentation.
4. Delete the connectors listed above.

Fixing broken connections

Occasionally, you will need to manually reconfigure things. If a data source adds a new field or removes a certain table some of your queries might break. You will need to look into the changes and update your queries to work appropriately.

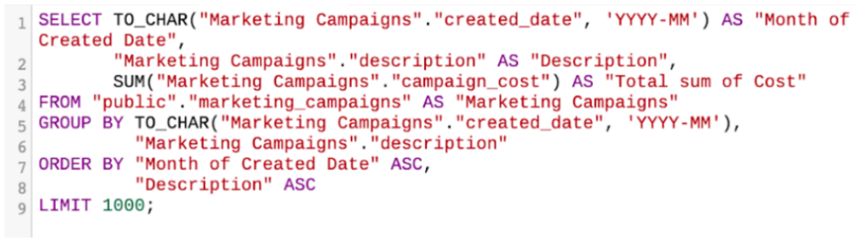


There was an error extracting data for one or more datasets

Your database returned: ERROR: column Marketing Campaigns.cost does not exist Line 3, column 12

```sql
SELECT TO_CHAR("Marketing Campaigns"."created_date", 'YYYY-MM') AS "Month of
Created Date",
       "Marketing Campaigns"."description" AS "Description",
       SUM("Marketing Campaigns"."cost") AS "Total sum of Cost"
FROM "public"."marketing_campaigns" AS "Marketing Campaigns"
GROUP BY TO_CHAR("Marketing Campaigns"."created_date", 'YYYY-MM'),
         "Marketing Campaigns"."description"
ORDER BY "Month of Created Date" ASC,
         "Description" ASC
LIMIT 1000;
```

As shown in the case above, we need to consult the datasource and update the field name in the query. Therefore, to fix the query, we updated "cost" to "campaign_cost" as shown below.



```sql
SELECT TO_CHAR("Marketing Campaigns"."created_date", 'YYYY-MM') AS "Month of
Created Date",
       "Marketing Campaigns"."description" AS "Description",
       SUM("Marketing Campaigns"."campaign_cost") AS "Total sum of Cost"
FROM "public"."marketing_campaigns" AS "Marketing Campaigns"
GROUP BY TO_CHAR("Marketing Campaigns"."created_date", 'YYYY-MM'),
         "Marketing Campaigns"."description"
ORDER BY "Month of Created Date" ASC,
         "Description" ASC
LIMIT 1000;
```

Performance

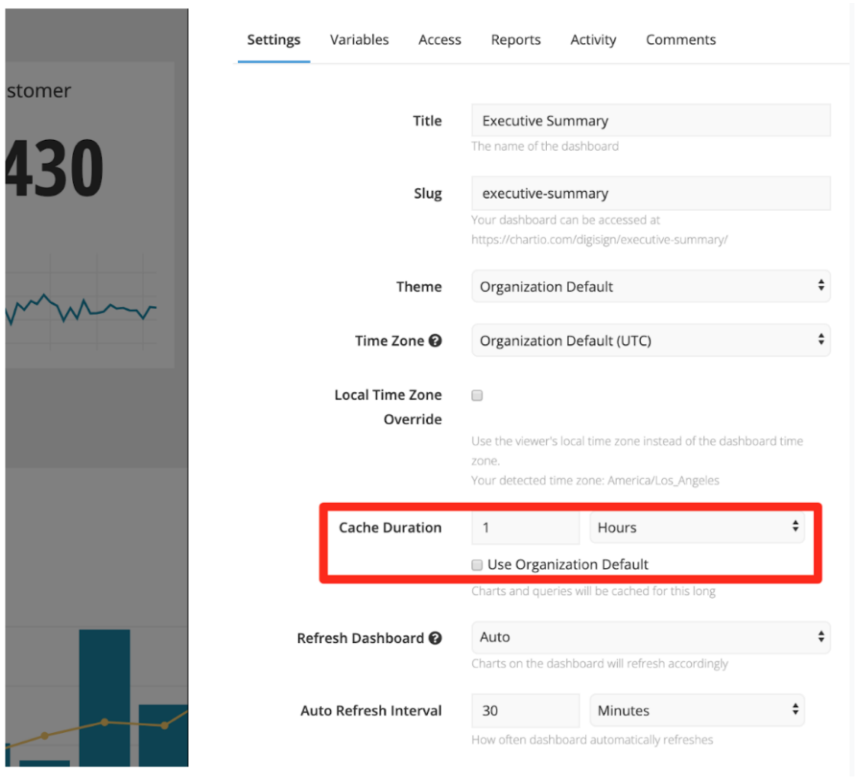
At the Data Lake stage, you should focus your optimization at the dashboard or query level.

Optimize individual queries

There are simple concepts to keep in mind when optimizing queries. Only join what you have to, Select only the columns you will need to analyze, and so on. To dig in deep check out our Book on Optimizing SQL.

Caching

Many BI products allow you to cache data for improved query speeds and less strain on the database itself. While this reduces the real-time nature of your analytical query, you can query the data as much as you would like.



Title: Executive Summary
The name of the dashboard

Slug: executive-summary
Your dashboard can be accessed at https://chartio.com/digisign/executive-summary/

Theme: Organization Default

Time Zone: Organization Default (UTC)

Local Time Zone Override: ☐
Use the viewer's local time zone instead of the dashboard time zone.
Your detected time zone: America/Los_Angeles

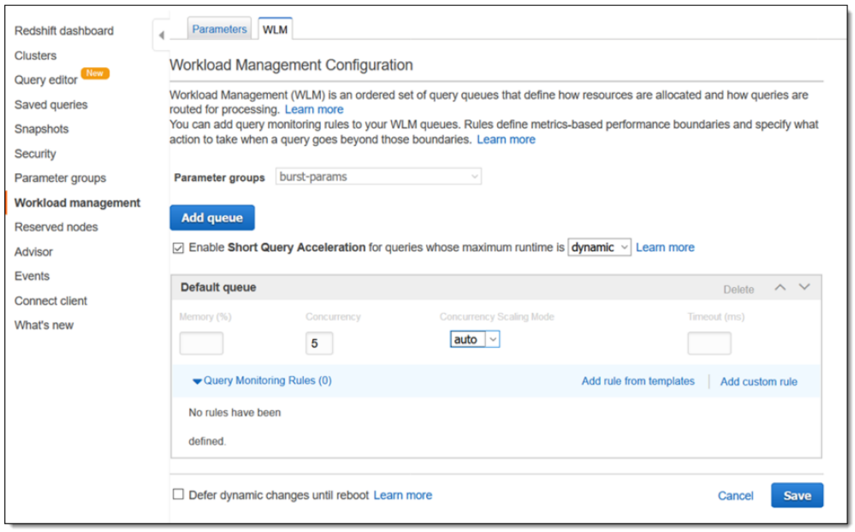
Cache Duration: 1 Hours
☐ Use Organization Default
Charts and queries will be cached for this long

Refresh Dashboard: Auto
Charts on the dashboard will refresh accordingly

Auto Refresh Interval: 30 Minutes
How often dashboard automatically refreshes

Create limits

Some platforms struggle with concurrency, where lots of people are querying the same source at once. Improve query speed in these scenarios by limiting how many queries people can perform on the database. While this can be a blow to people's curiosity or analysis, it quickly solves this performance problem.



Workload Management Configuration

Workload Management (WLM) is an ordered set of query queues that define how resources are allocated and how queries are routed for processing. Learn more
You can add query monitoring rules to your WLM queues. Rules define metrics-based performance boundaries and specify what action to take when a query goes beyond those boundaries. Learn more

Parameter groups: burst-params

☑ Enable **Short Query Acceleration** for queries whose maximum runtime is dynamic Learn more

Default queue

Memory (%)	Concurrency	Concurrency Scaling Mode	Timeout (ms)
	5	auto	

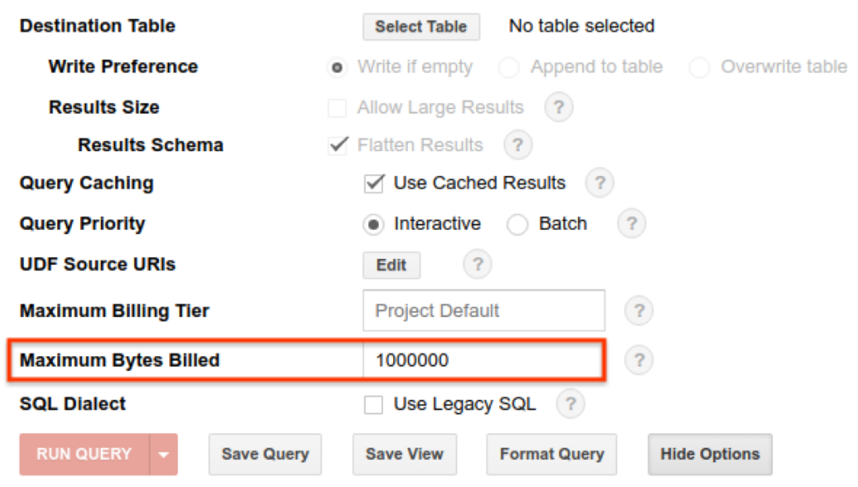
Query Monitoring Rules (0)

No rules have been defined.

☐ Defer dynamic changes until reboot Learn more

Queries can be limited in different ways:

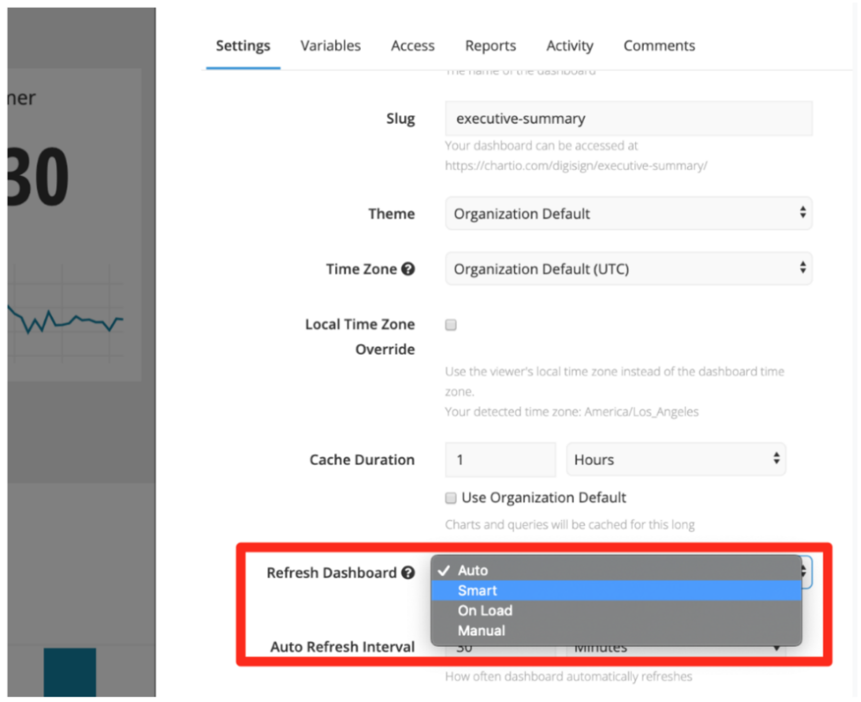
- Limit number of people querying
- Limit queries per day
- Big Query - Set max bytes



Destination Table	Select Table No table selected
Write Preference	[✓ Write if empty] Append to table Overwrite table
Results Size	Allow Large Results
Results Schema	[✓ Flatten Results]
Query Caching	[✓ Use Cached Results]
Query Priority	[✓ Interactive] Batch
UDF Source URIs	Edit
Maximum Billing Tier	Project Default
Maximum Bytes Billed	1000000
SQL Dialect	Use Legacy SQL

Scheduling

Examine how your BI product queries the database. Does it do it automatically on a schedule or is it manual. Tools such as Chartio have options to schedule queries to run at off-peak times to balance the load on the database and Smart Refresh options to prevent queries from running when dashboards aren't actively being viewed.



Slug	executive-summary Your dashboard can be accessed at https://chartio.com/digisign/executive-summary/
Theme	Organization Default
Time Zone	Organization Default (UTC)
Local Time Zone Override	Use the viewer's local time zone instead of the dashboard time zone. Your detected time zone: America/Los_Angeles
Cache Duration	1 Hours Use Organization Default Charts and queries will be cached for this long
Refresh Dashboard	[✓ Auto] Smart On Load Manual
Auto Refresh Interval	30 Minutes How often dashboard automatically refreshes

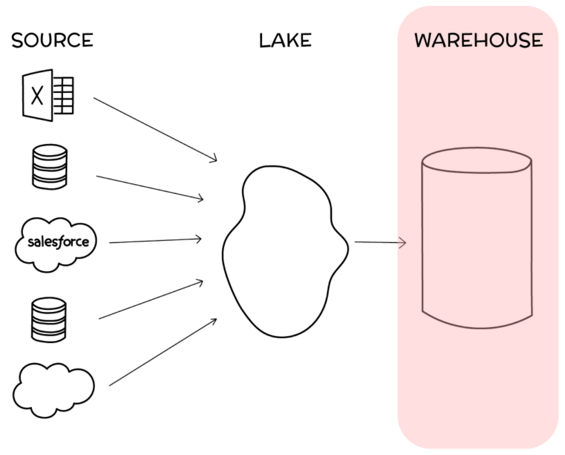
These sorts of tweaks become especially important as more users query the database.

Summary

To keep a Data Lake being useful you need to:

1. Monitor data source connections and update pipelines when necessary. Use an ETL product to make this simple.
2. Keep an eye on performance. More people will be querying the database in different ways. Optimize individual queries that are impacting the database, set up caching to improve speed, create limits to stop people from over-querying, and schedule how your BI tool refreshes queries.

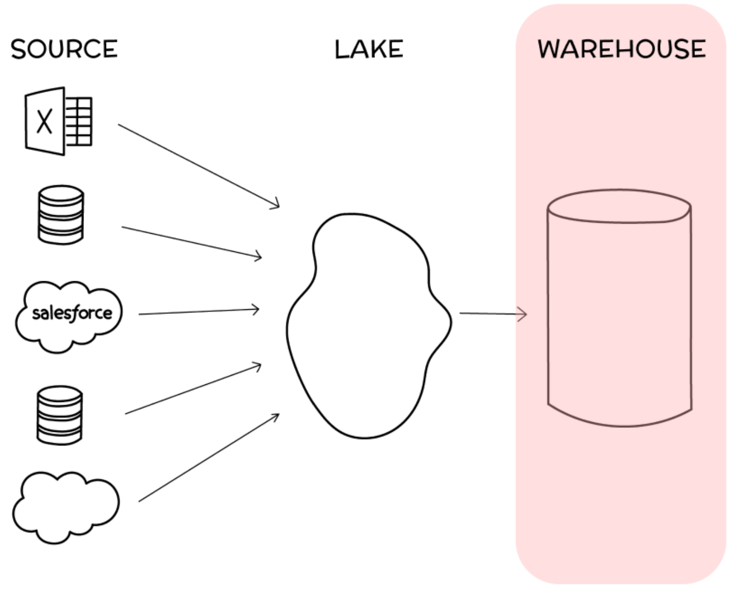
Stage 3 - Warehouse



Why Build a Data Warehouse

What is a Data Warehouse?

A Data Warehouse (also commonly called a single source of truth) is a clean, organized, single representation of your data. Sometimes it's a completely different data source, but increasingly it's structured virtually, as a schema of views on top of an existing lake.



Having a clean unified source of truth enables you to write simpler queries, make fewer errors, work faster (as you're more organized), and repeat yourself much less often.

This stage is right for you if:

- More than a few people are going to be working with this dataset
- You want a clean source of truth of your company
- You don't like fighting integrity issues
- You need to separate the structure of the data from the always changing transactional sources.
- You Don't like Repeating Yourself (DRY)

You've outgrown this stage if:

- You want to get democratized - and enable others in your company to explore and understand data themselves
- You're prepared to teach and enable business users in your company - hopefully using the many resources of the Data School
- You have projects that require different formats of the source of truth for easier use
- Having truly informed employees is important to your company's competitive success

Six reasons to build a Data Warehouse

1. **Easier** to understand and query - simplified single model. No more duplicate tables, confusing column names, or mysterious values.
2. **Faster** for the data team to use. Less time is needed to clean and transform data to perform analysis.
3. **Approachable** to work with for business users. Complex joins have been reduced and the correct column is obvious.
4. **Trusted**, consistent source of answers. Everyone generates insights from the same data; no more varying answers to the same question.
5. **Maintainable** with less time and effort. After adopting naming conventions and a style guide you can maintain as you add data.
6. **Separated** from transactional data schema. Queries don't affect app performance, and aren't affected by rapid changes in the data.

This section of the Data Governance book will explain why you should create a Data Warehouse, and how to implement it so that you get all the benefits it can deliver to your business. Before we dive in deep, let’s look at the data issues you face with a Data Lake.

The Problem with Data Lakes

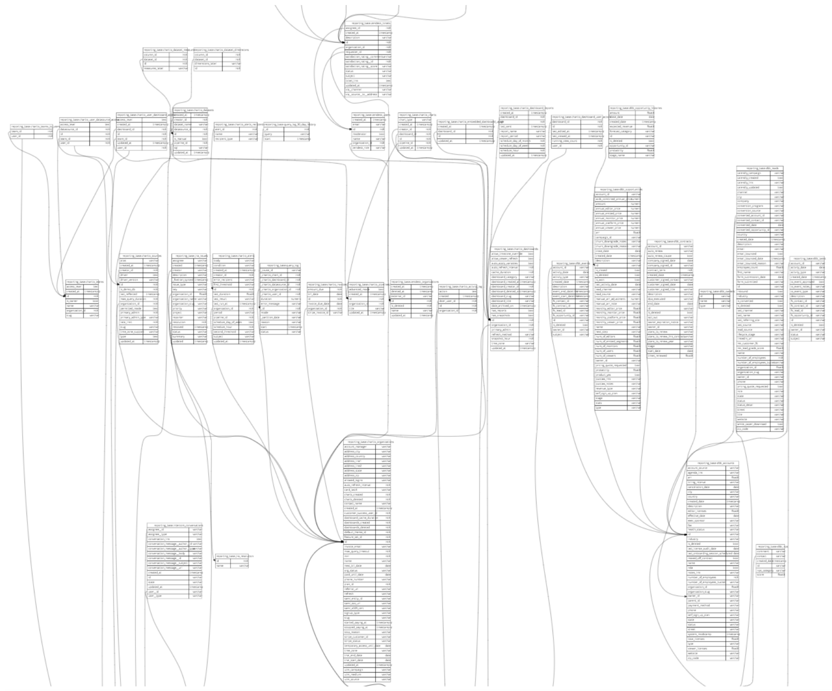
Typically, organizations reach roadblocks in making sense of the data in their Data Lake. When the amount of data in your Data Lake reaches a level of complexity with irrelevant, unstructured data, it will be too confusing and messy for non-data analysts to use. Data is inconsistent and unstructured, so it can be error-prone with users using the wrong columns or calculating metrics incorrectly.

When organizations have an initiative to empower users outside of the data science or engineering team to leverage data, they will move to a Data Warehouse. What differentiates a Data Warehouse from a Data Lake, or other source, is that the Data Warehouse will provide a cleaner view of the data and is easier for users to query.

To illustrate the difference, imagine all of your inventory is under one roof, but in a big pile. It’s unsorted, and some of it’s rotten. You may know where everything is in the pile, but if you want to work with others you’ll need to organize it. A proper warehouse requires shelves and organization that makes sense to anyone using it.

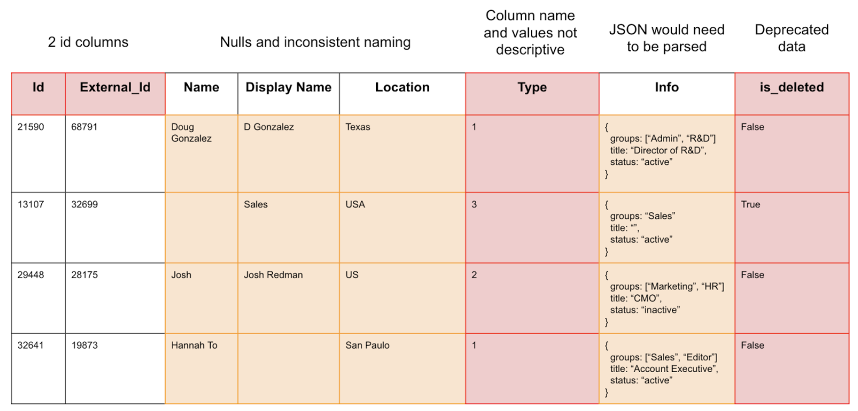
Take an example of a database tracking a product’s users and usage data. The raw data may be difficult to understand for the average user because of things such as bad naming conventions, complex data types, data inconsistencies, irrelevant data and highly normalized tables.

Example Data Lake Schema:



This example was designed as a transactional schema, not for analysis. Without a tool such as Chartio, navigating this schema for analysis would be incredibly challenging. However with Chartio you only need to focus on cleaning up tables to get much more value out of your data. Focus on making Data Lake tables easy to understand.

Example Data Lake table:


2 id columns
Nulls and inconsistent naming
Column name and values not descriptive
JSON would need to be parsed
Deprecated data

Id	External_Id	Name	Display Name	Location	Type	Info	is_deleted

21590	68791	Doug Gonzalez	D Gonzalez	Texas	1	{
						groups: [“Admin”, “R&D”]
						title: “Director of R&D”,
						status: “active”
						}
							False

13107	32699		Sales	USA	3	{
						groups: “Sales”
						title: “”,
						status: “active”
						}
							True

29448	28175	Josh	Josh Redman	US	2	{
						groups: [“Marketing”, “HR”]
						title: “CMO”,
						status: “inactive”
						}
							False

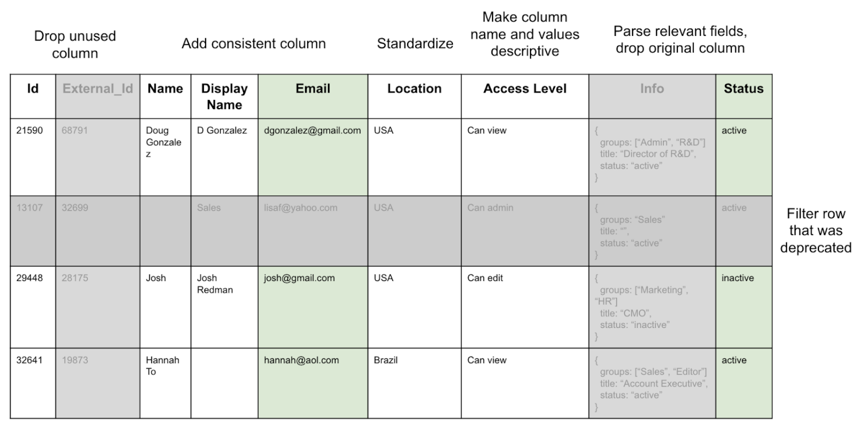
32641	19873	Hannah To		San Paulo	1	{
						groups: [“Sales”, “Editor”]
						title: “Account Executive”,
						status: “active”
						}
							False

We can see multiple columns with issues that would be difficult for an analyst to understand or make use of. Let’s see how this table could be reconfigured to become useful for analytics.

How Data Warehouses differ from Data Lakes

Use modeling to create a clean version of the schema where all of the above inconsistencies and points of confusion are addressed. This will be your company’s Data Warehouse. It will enable more users to understand and use the data. You can remove a few irrelevant tables for analysis but most of the focus should be on cleaning up columns.

Let’s take the table above and apply some simple transformations to it.


Drop unused column
Add consistent column
Standardize
Make column name and values descriptive
Parse relevant fields, drop original column
Filter row that was deprecated

Id	External_Id	Name	Display Name	Email	Location	Access Level	Info	Status

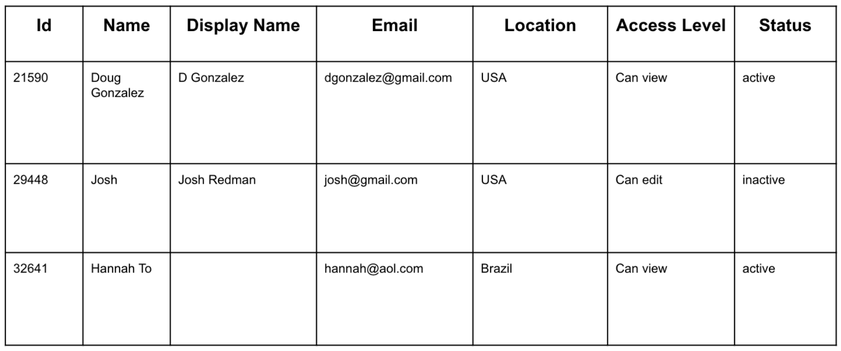
21590	68791	Doug Gonzale z	D Gonzalez	dgonzalez@gmail.com	USA	Can view	{
							groups: [“Admin”, “R&D”]
							title: “Director of R&D”,
							status: “active”
							}
								active

13107	32699		Sales	lisaf@yahoo.com	USA	Can admin	{
							groups: “Sales”
							title: “”,
							status: “active”
							}
								active

29448	28175	Josh	Josh Redman	josh@gmail.com	USA	Can edit	{
							groups: [“Marketing”, “HR”]
							title: “CMO”,
							status: “inactive”
							}
								inactive

32641	19873	Hannah To		hannah@aol.com	Brazil	Can view	{
							groups: [“Sales”, “Editor”]
							title: “Account Executive”,
							status: “active”
							}
								active

We can see how dropping columns, adding columns, filtering rows and clarifying columns make the data much more straightforward to use and interpret. Now, people without experience with the data have a much easier time coming up to speed and will make fewer mistakes. Let’s look at the final table without all the editing markup.



Id	Name	Display Name	Email	Location	Access Level	Status
21590	Doug Gonzalez	D Gonzalez	dgonzalez@gmail.com	USA	Can view	active
29448	Josh	Josh Redman	josh@gmail.com	USA	Can edit	inactive
32641	Hannah To		hannah@aol.com	Brazil	Can view	active

This is the promise of a Data Warehouse: clear tables that can be used by anyone without having to dramatically alter the schema.

Summary

Data warehouses make your data:

- Easier to understand and query - simplified single model

- Faster for the data team to use
- Approachable to work with for Business Users
- Trusted, consistent source of answers
- Easier and less timely to maintain

Data Lake data is the pile of products in your building.

Data Warehouse is those same products sorted, shelved, and tagged.

Data Warehouse Architecture

When multiple people ask the same question using the same data and get varying answers, it creates doubt in all of the data in your organization. Additionally, it's demoralizing for everyone and time-consuming to figure out the right answer. Unfortunately, this is typical when data has not been cleaned up into a Single Source of Truth.

People get inconsistent results because:

- Data sources change
- Schemas are overly complex
- Table and column names are confusing
- Metrics need to be derived from the data

What is a Data Warehouse Architecture?

A Data Warehouse is a database where the data is accurate and is used by everyone in a company when querying data.

The promise of a Single Source of Truth is accuracy across your organization.

This is an obvious thing that any company wants, yet a lot of companies struggle to deliver. Creating a Single Source of Truth requires data engineering effort. Let's explore the problems that a Single Source of Truth solves, issues to watch out for, and best practices.

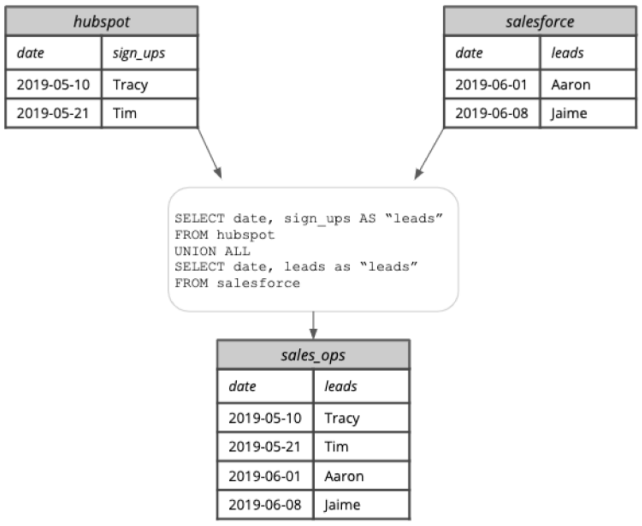
Data Sources

Before you even build a Single Source of Truth, your company will likely have data sources that overlap in terms of what they track. You will also have data from dormant data sources in your Data Lake that is still needed for certain analyses.

Imagine you were tracking sign-ups via Hubspot and after a year you decided to switch to Salesforce. That means that prior to your switch, the Salesforce data will be empty. Moreover, the Google Analytics data might not be as well synchronized between your Hubspot data and your Salesforce data. When an analyst attempts to query for sign-ups, it will be unclear which data source they should use.

Consolidate Data Sources

When your company has used multiple tools to track the same type of data, if you can, migrate the data from the previous tools into the latest tool. If this is not an option, use the data warehouse to create a table which UNIONs the data from both sources. This ensures the historical records are not lost and creates one location for relevant metrics. This will require some renaming and cleaning to accomplish.



hubspot

date	*sign_ups*
2019-05-10	Tracy
2019-05-21	Tim

salesforce

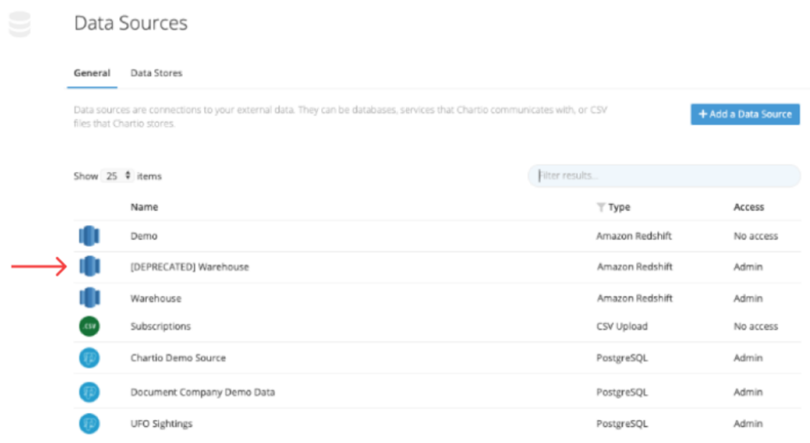
date	*leads*
2019-06-01	Aaron
2019-06-08	Jaime

```
SELECT date, sign_ups AS “leads”
FROM hubspot
UNION ALL
SELECT date, leads as “leads”
FROM salesforce
```

sales_ops

date	*leads*
2019-05-10	Tracy
2019-05-21	Tim
2019-06-01	Aaron
2019-06-08	Jaime

In addition, if you want to maintain access to old/unused data sources from your Data Lake in your Data Warehouse, you can label data sources as deprecated or approved to help guide people during their analysis.



Data Sources

General Data Stores

Data sources are connections to your external data. They can be databases, services that Chartio communicates with, or CSV files that Chartio stores.

+ Add a Data Source

Show 25 items

Filter results...

Name	Type	Access
Demo	Amazon Redshift	No access
[DEPRECATED] Warehouse	Amazon Redshift	Admin
Warehouse	Amazon Redshift	Admin
Subscriptions	CSV Upload	No access
Chartio Demo Source	PostgreSQL	Admin
Document Company Demo Data	PostgreSQL	Admin
UFO Sightings	PostgreSQL	Admin

Simplify the Schema

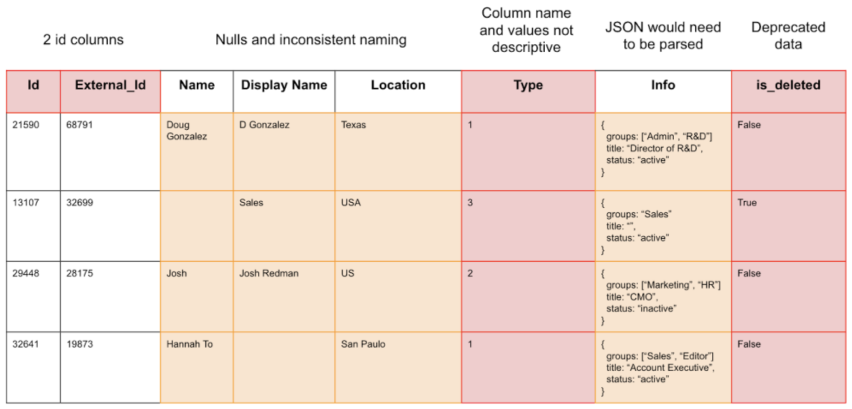
In a Data Lake, the schema reflects in transactional logic of an application and follows best practices (such as a 3rd normal form) so that updating values will not produce errors. But this type of schema can be difficult to navigate and many tables will never be used in an analysis. In the past, books recommended using dimensional modeling to reduce the schema complexity, make it easier to run queries, and enhance performance. Today, due to advances in BI tools such as Chartio and Data Warehouse technologies, dimensional modeling is no longer worth the effort.

Simple Schema

We create a Single Source of Truth by creating views on top of the existing schema. There is no need to move away from 3rd normal form. The main thing we want to do to simplify the schema is to exclude tables from the new views that only contain app specific logic and are not useful for analysis. If you want to make it even easier to work with a specific set of data, you can create a wide table (view) that does all the joins. This can sit alongside the cleaned up normalized version of the Data Warehouse.

Simplify Tables and Columns

Table and column names are typically created by engineers to be used by the application the data comes from. Table names, column names, and even a column's purpose for being in the table can be confusing to others. This makes it challenging for business users to analyze the data without consulting the engineer. We can review the table we referenced in Why Build a Data Warehouse:



2 id columns		Nulls and inconsistent naming			Column name and values not descriptive	JSON would need to be parsed	Deprecated data
Id	External_Id	Name	Display Name	Location	Type	Info	is_deleted

21590	68791	Doug Gonzalez	D Gonzalez	Texas	1	{
						groups: ["Admin", "R&D"]
						title: "Director of R&D",
						status: "active"
						}
							False

13107	32699		Sales	USA	3	{
						groups: "Sales"
						title: "",
						status: "active"
						}
							True

29448	28175	Josh	Josh Redman	US	2	{
						groups: ["Marketing", "HR"]
						title: "CMO",
						status: "inactive"
						}
							False

32641	19873	Hannah To		San Paulo	1	{
						groups: ["Sales", "Editor"]
						title: "Account Executive",
						status: "active"
						}
							False

- Having multiple Id columns can be confusing.
- Nulls can produce unexpected results during aggregations.
- Inconsistent naming reduces confidence that the data is correct, and makes it hard to aggregate and group the data.
- Non-descriptive Column names and values will require the analyst to ask an engineer for clarification.
- Most analysts are not able to use regex to parse out valuable information from JSON data.

- Deprecated data flags are often missed by analysts, so this leaves room for error in aggregations.

To address these issues we need to keep the analyst/business user in mind and make all of the fields easy for them to interpret. The first step is to develop guidelines for how you want to clear up the data.

Naming convention and style guide

When going through and recreating the schema with views of the relevant tables you should also clean up what's in each table. Exclude irrelevant columns and rename any columns that are confusing. Naming conventions help people analyze data consistently because they clarify what each column is and how it can be used.

Simplify

It's quite common for raw data to be extremely complex. Data was typically meant to be consumed by applications and not directly by business users. By taking some time to simplify data, we can greatly improve business user success when querying.

Best Practice	Reason
Only include fields with obvious analytical purpose	It's best to start modeling with only the most relevant columns, excluding any columns that has no immediate or obvious analytical purpose.
Extract relevant data from complex data types	Application data sources may contain JSON, arrays, hstore and other complex data types. These are typically hard to query from business intelligence tooling and should have relevant data extracted into new columns. Example: Supposed a table books contains an id column and the following JSON column. `{ title: "Moby Dick", author: "Herman Melville", genres: ["novel", "fiction"] }` The resulting modeled books table would contain an id, title, and author columns. Genres could be modeled as an independent table, reduced to a single genre based on custom rules, or some other method.
Change flags and crypto abbreviations to meaningful values	It's common for application databases to have flags or cryptic abbreviations in columns that work well for the application and terrible for a business user. It's important to transform these values into easy, human readable values. Some examples: • Boolean values 0 and 1 should be transformed to relevant strings, such as true and false or on and off. • Flag values should be transformed into relevant strings. If a column billing_status has three numeric values (i.e. 0, 1, 2) that represent some status, they should be transformed into a relevant business concept such as Active, Not Active, Delinquent. • Cryptic values should also be transformed into easy to understand business concepts.
De-normalize where possible	Applications typically have highly normalized tables to prevent duplicates, reduce space, and make modification easier. This typically makes it harder for business users to browser the schema however because the complexity of the joins may be hard to follow. Build wider tables where appropriate, collapsing common concepts into a single table. Some examples could be: • Combine the sources, sources_redshift, sources_postgres, and sources_myself tables into a single sources table with the lowest common denominator of values that make sense for a business user.

	• Combine the users and addresses tables into a single users table since addresses are meaningless on their own. This simplification requires trial and error and you may not always get it right.

Cleaning

Data is messy and requires some cleaning to ensure accurate results. Cleaning prevents common problems that might cause a query to produce incorrect results.

Best Practice	Reason
Attempt to eliminate NULLs	NULL values have unexpected consequences in SQL (is "string" <> NULL?). It's best to remove all nulls with values. Some examples: • Change all NULL values in the first_name column to the string Blank. • Change all NULL values in the last_login_type column to the string Never Logged In for customers that have never logged in.
Fix common data inconsistencies	Bad data always makes its way into raw data sources. Whether it is misspellings or just junk data, it is important to clean up the data as much as possible. Some examples: • State names that have a mix of abbreviations, full names, and junk data should be transformed into a single, consistent format such as the full state name. • Phone numbers might be garbage text entered by users to avoid getting phone calls.
Follow Naming Conventions	Schemas, tables, and columns should all be named in accordance with naming conventions listed below. At a minimum, names should be human readable and be representative of the data type and values stored.
Remove irrelevant data	Rows that are irrelevant for various reasons should be removed entirely from the data set. Some examples could be: • Employee testing • Fraud or spam • Inactive Obviously, if analysis is being done on fraud or spam, that data should not be removed but in most causes, if a row would always be excluded from a query, go ahead and remove it in modeling.
Change Data Types	Modeling is a great time to change data types to more appropriate types. Unix timestamps could be converted from int columns to datetime for example.

Naming Conventions

Initially there will be a variety of naming conventions used for tables, columns, and values. Creating a standard for all of these makes it easier for others to find and understand the data they are looking for.

Best Practice	Reason
Plural Table Names	A table of Leads should be titled "Leads" not Lead. When there are more than two words on the last needs to be pluralized: opportunity_histories
id as primary key	A simple numeric primary key labeled id should be standard for all tables.
foreign keys follow [tablename]_[id] format	ForeignKeys should follow this format to make it very clear on where the table is linking to. If there are two foreign keys to the same table you can preopend a name

to them following the format:

[uniquename]_[tablename]_[id].

An accounts table linking to a users table with both a billing contact and a main owner would look like this:

Accounts

owner_user_id
billing_contact_user_id

Start columns with a _ if they are needed but should be hidden for Visual mode.

If there are columns you need in the model for joining or other purposes but don't want visible by default in visual mode you can prefix them. They will otherwise be treated just as any other column.

Let's say you didn't think the foreign keys in the accounts table above needed to be shown in Visual mode. You can simply prefix them as shown below. The relationships will still be detected. It's a best practice not to show the foreign keys visually.

Accounts
id
name
_owner_user_id
_billing_contact_user_id

This should not be used for columns you're on the fence about needing. Those just shouldn't be included. These are for columns that are needed for querying purposes but have no use in a Visual setting - primarily foreign keys.

Lower case, underscored naming

Our data model needs to be easily editable in SQL mode so we should follow conventions that make editing raw SQL easier. Therefore, we should attempt to have column names like *id, first_name, last_name,* and *last_login_type* instead of more human readable forms in the model. Chartio will handle that conversion.

Publish a style guide and distribute it among all of your employees to make adoption of known terms much easier.

Metrics

There are a lot of different ways to measure how a business is performing. Some are fairly well known, such as Monthly Active Users or Number of Trials Started. In most businesses, getting an accurate count on a metric is difficult because you need to filter out irrelevant data:

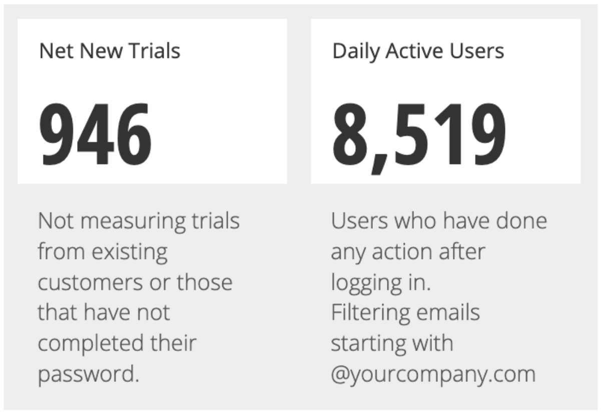
- Test accounts
- Internal Company emails
- Non product-related page visits
- Users that are no longer employed by a client company

Not filtering out the right data will negatively affect your analysis. Presenting to others who have a conflicting analysis of their own will cause everyone to lose trust in the data.

Another more subtle problem with metrics is abbreviations. If Monthly Active Users is abbreviated as MAU in the database, it may be misinterpreted in someone else's analysis. Do not assume everyone understands the abbreviation for the metrics you are reporting.

Create a Standard Metrics Dashboard

To define the calculation of a metric, create a Dashboard with this metric in it and provide text on the dashboard to explain how it was calculated and what has been filtered out. Make this easily searchable!


Net New Trials
946
Not measuring trials from existing customers or those that have not completed their password.
Daily Active Users
8,519
Users who have done any action after logging in.
Filtering emails starting with @yourcompany.com

Another approach is to pre-calculate the metric in a view in the Single Source of Truth database. We recommend doing this through a SQL-based modeling tool such as dbt or Dataform. Defining the metric in the database will remove most, if not all, of the confusion.

To eliminate any remaining confusion on using the metric in your analysis, many SQL-based modeling tools can add a data dictionary to the data model. This allows the author of the data model to write out comments on why it was calculated that way and why certain data was filtered out.

Storing the metric in the database through modeling allows you to control changes in the data and the definitions systematically. You will still need to communicate changes, but they will be documented if anyone needs to check on their own.

Summary

- Create a Single Source of Truth and give employees access to it and only it
- Make your data intuitive through naming conventions and style guides
- Simplify the Schema by excluding app-specific logic tables
- Simplify table and column names: define them by their spoken language titles instead of technical jargon
- Centralize the control and accuracy of metric calculations through SQL-based modeling

References.

- Getdbt.com

Data Warehouse Security

At the warehouse stage, more groups than just the centralized data team will commonly have access. You must use data governance to safeguard certain pieces of sensitive information from being accessed by the wrong people in your organization. Many security regulations mandating data access rules have been passed, such as GDPR, and many companies have industry standard compliance rules that they adhere to as well, like SOC and HIPAA.

Whether it is personally identifiable information (PII) or financial information, sensitive data is much more prevalent throughout a product's journey and your Data Warehouse than one might think. Preventing the exposure of such information is key and can be approached in a variety of ways.

Every company stores information that cannot be exposed to everyone who works in the company. When moving from a Data Lake to a Data Warehouse more people will gain access to data. You need to ensure that sensitive information is aligned to what is being stored, how it's restricted in the Data Warehouse, and how it can be accessed via your BI tools.

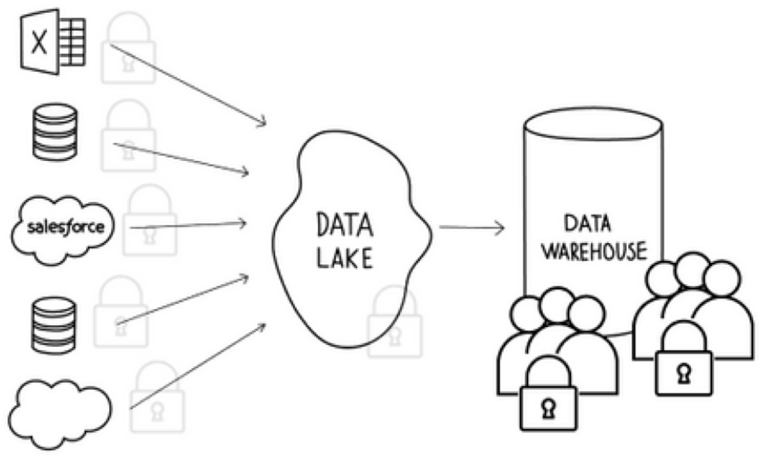
There are multiple ways this can be handled and multiple questions to be answered:

- Where is sensitive data (PII and financial) currently handled?
- Will this sensitive data still be present in the Data Warehouse and then cleaned up?
- How will this information be removed or restricted from the exposed datasets - scripts on the way to the warehouse, data marts created from the warehouse?

These questions need to be answered before you connect these sources to your BI tool.

Within large companies, often times all internal data is considered confidential. Even internally, departments are on a need-to-know basis regarding data in other departments. Issues arise when a company connects its Data Warehouse to its BI platform or grants query access across different departments in general. This leads to sensitive data potentially being exposed to unauthorized users.

How to secure sensitive data on the database



The most direct way to limit access to the proper people is to enforce rules on the database level. This can be done through creating slave read-only replicas, creating custom user groups, and encrypting sensitive data.

Slave Read-Only

Set up your warehouse to be read-only by default. This prevents any dangerous SQL write statements from being executed on your data.

Custom User Groups

Regardless of whether you create the slave read-only warehouse, create a new user group that has read access only. You can choose to exclude access to specific tables or columns of data from that new user group. In addition, you can restrict access to row-specific data. Row-level permissioning allows you to give full access to tables containing sensitive information but restricts which rows and values the person querying can see. Depending on the underlying database, configuring row-level permissions differs slightly.

A great example of when to use row-level permissioning is adhering to HIPAA compliance when accessing a hospital's dataset. Each doctor within this hospital has access to their own patients' records for analysis and review. However, we want to prevent every doctor from having access to every patient's medical records. Implementing access controls at the row level by account/patient ownership (whether that is patient id, patient name, etc.) will prevent doctors from having the chance to access a patient's personal information they don't need. You can apply this example to other groups as well: sales teams, customer tracking, employee records, etc.

Encrypt Columns

If you need to group or aggregate by sensitive data you can create encrypted versions of the data. Then users can create summary tables where sensitive metrics, like financial data, can be aggregated to a level that is appropriate for different departments to see and analyze. The level of security you implement will limit what type of analysis can be performed on the data, but does ensure that the sensitive data is protected.

How to secure sensitive data in a BI tool

Now that we have secured the underlying database, we need to ensure that there are no loopholes in the BI tool. Even setting up the right permissions on the database does not ensure sensitive data won't be inappropriately shared through a dashboard or report. This type of issue can be difficult to prevent, so the common strategy is to set policies with users of the BI tool and regularly audit who is accessing and viewing what data.

Consistent account audit/clean up

Projects change, roles change, and use cases change. Any of these changes can impact employee permissions. Outdated permissions can lead to compliance and privacy issues. Periodically reviewing and updating permissions is a best practice to protect sensitive data.

During an audit you should check all the previous questions we talked about:

- Who has access to which data sources?
- Who has access to sensitive row-level information?
- Who is on the admin team or has admin access?
- Who has access to or is viewing dashboards and reports containing sensitive data?

BI tools offer answers to these questions through varying levels of usage information in-app for the admins to monitor and review. If you do not have all of the pieces of information necessary, talk to the support team at the BI tool you're using. They can help pull the necessary information so you can make informed decisions on the security of your account. Feel free to push the boundaries and you may be surprised at what information is available when you ask.

Summary

Create sensible limits on the database by removing edit access and filtering what data users have access to through custom user groups, and finally encrypting sensitive data. Regardless of the precautions you take you should still perform regular audits to verify who has access to what and where sensitive data is being exposed.

Data Warehouse Implementation

Now that we've established what changes we want to make and decided on what engine to use for our Data Warehouse, let's go through the process of getting data from the Lake into the Warehouse. While this sounds complicated, it's only comprised of using SQL to create Views.

Why SQL

We recommend using SQL to perform all transformations. It's the standard language for relational database management systems (which is what a Data Warehouse should be) and it's the environment you are probably using for your Data Lake. Working in a SQL-based model is ideal because a variety of tools and platforms already exist to write and execute queries. Also, data engineers, analysts, and some business users already understand how to use it.

Why Views

Views allow us to quickly reformat what the data looks like without needing to build a new Data Warehouse or incurring costs from storing any additional data. Unless you are dealing with massive amounts of data there are not significant performance gains in creating new tables or materializing the views.

Use a Modeling tool: dbt

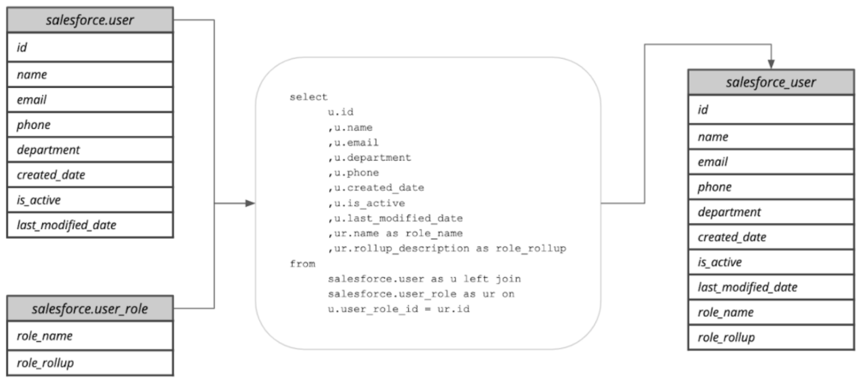
Instead of writing the views directly on the database (which is an option) we recommend using dbt for creating your SQL views. dbt provides many features to help you keep a clean Data Warehouse such as version control, logging, and much more.

Data Lake to Data Warehouse View Examples

Here is an example of applying a transformation to move from a Data Lake to a Data Warehouse. First, we build a query to combine a couple of Salesforce objects into a single table. For example, using information about an individual and their role within a client company can give you more insight into how you may want to interact with that person.

So, getting information on that person's role into the same table as his/her contact along with some basic demographic information, will save the end user some time in querying the Data Warehouse.

That query might look like this:

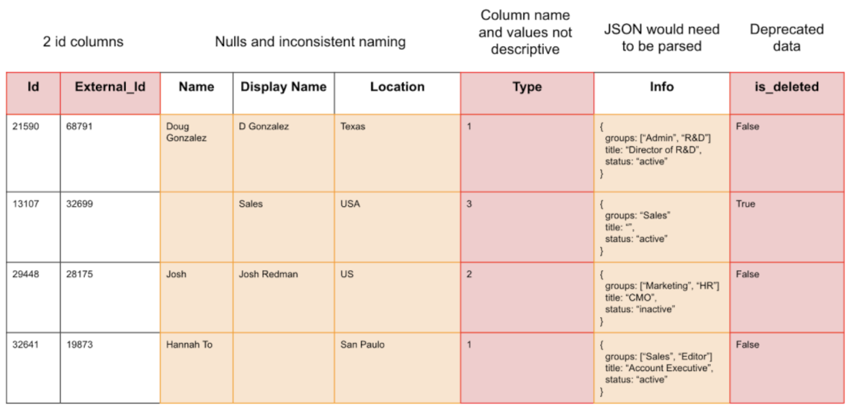


```
select
        u.id
        ,u.name
        ,u.email
        ,u.department
        ,u.phone
        ,u.created_date
        ,u.is_active
        ,u.last_modified_date
        ,ur.name as role_name
        ,ur.rollup_description as role_rollup
from
        salesforce.user as u left join
        salesforce.user_role as ur on
        u.user_role_id = ur.id
```

We are choosing a subset of the total possible columns and rolling up/denormalizing the table a bit to make it easier for others to query. To make this code into SQL that builds our Data Warehouse, we need to add CREATE VIEW. So the query would actually be:

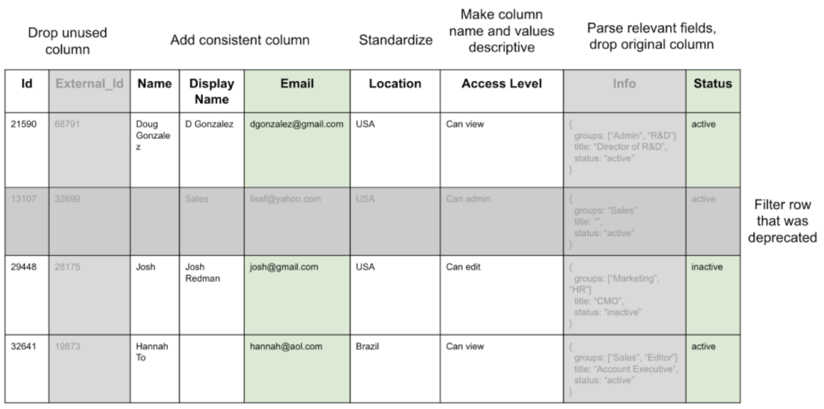
```
CREATE VIEW salesforce_user AS
SELECT
        u.id
        ,u.name
        ,u.email
        ,u.department
        ,u.phone
        ,u.phone
        ,u.created_date
        ,u.is_active
        ,u.last_modified_date
        ,ur.name as role_name
        ,ur.rollup_description as role_rollup
FROM
        salesforce.user as u
        left join salesforce.user_role as ur on u.user_role_id = ur.id;
```

If we go back to the example first introduced in the Why Build a Data Warehouse article we can walk through all of the transformations described in one SQL query. So let's look at that messy table with all of the hard to understand/query fields.



2 id columns		Nulls and inconsistent naming			Column name and values not descriptive	JSON would need to be parsed	Deprecated data
Id	**External_Id**	**Name**	**Display Name**	**Location**	**Type**	**Info**	**is_deleted**
21590	68791	Doug Gonzalez	D Gonzalez	Texas	1	{ groups: ["Admin", "R&D"] title: "Director of R&D", status: "active" }	False
13107	32699		Sales	USA	3	{ groups: "Sales" title: "", status: "active" }	True
29448	28175	Josh	Josh Redman	US	2	{ groups: ["Marketing", "HR"] title: "CMO", status: "inactive" }	False
32641	19873	Hannah To		San Paulo	1	{ groups: ["Sales", "Editor"] title: "Account Executive", status: "active" }	False

We then want to make all of the following changes:



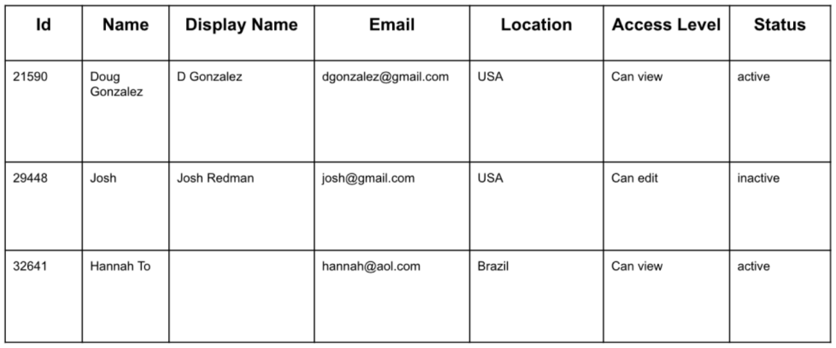
Drop unused column		Add consistent column			Standardize	Make column name and values descriptive	Parse relevant fields, drop original column		
Id	**External_Id**	**Name**	**Display Name**	**Email**	**Location**	**Access Level**	**Info**	**Status**	
21590	68791	Doug Gonzale z	D Gonzalez	dgonzalez@gmail.com	USA	Can view	{ groups: ["Admin", "R&D"] title: "Director of R&D", status: "active" }	active	
13107	32699		Sales	lisaf@yahoo.com	USA	Can admin	{ groups: "Sales" title: "", status: "active" }	active	Filter row that was deprecated
29448	28175	Josh	Josh Redman	josh@gmail.com	USA	Can edit	{ groups: ["Marketing", "HR"] title: "CMO", status: "inactive" }	inactive	
32641	19873	Hannah To		hannah@aol.com	Brazil	Can view	{ groups: ["Sales", "Editor"] title: "Account Executive", status: "active" }	active	

We can create this as a series of SQL statements in a dbt file of common table expressions with a final CREATE VIEW query at the bottom:

```
-- drop unused column External_id
WITH t1 AS (
        SELECT Id, Name, Display Name, Email, Location, Type, Info, Status
        FROM dl_table
),

-- Add consistent column Email
t2 AS (
        SELECT Id, Name, Display Name, Email, Location, Type, Info, Status, is_deleted
        FROM t1
        JOIN dl_email
        ON t1.Id = dl_email.Id
),

--Standardize Location column
t3 AS (
        SELECT Id, Name, Display Name, Email,
        CASE WHEN Location = "US" THEN "USA"
                WHEN Location = "Texas" THEN "USA"
                WHEN Location = "Sao Paulo" THEN "Brazil"
                ELSE Location
                END AS "Location",
        Type, Info, Status, is_deleted
        FROM t2
)

--Make column names and values descriptive for Type
t4 as (
        SELECT Id, Name, Display Name, Email, Location,
        CASE WHEN Type = "1" THEN "Can view"
                WHEN Type = "2" THEN "Can edit"
                WHEN Type = "3" THEN "Can admin"
                END AS "Access Level",
        Info, Status, is_deleted
FROM t3
)

--Parse relevant fields, drop original column for Info
t5 as (
        SELECT Id, Name, Display Name, Email, Location, Access Level,
                CASE WHEN Info = "%active" THEN "active"
                        WHEN Info = "%inactive" THEN "inactive"
                        END AS "Status",
                is_deleted
FROM t4
)

-- filter row that was deprecated from is_deleted, and drop column
t6 as (
        SELECT Id, Name, Display Name, Email, Location, Access Level, Status
        FROM t5
        WHERE is_deleted != True
)

-- create view for Data Warehouse
CREATE VIEW dw_table AS
        SELECT *
        FROM t6
```

For a given table we suggest managing all transformations step by step in common table expressions with notes describing what is happening at each step.

We now have a clean view of the original data



Id	Name	Display Name	Email	Location	Access Level	Status
21590	Doug Gonzalez	D Gonzalez	dgonzalez@gmail.com	USA	Can view	active
29448	Josh	Josh Redman	josh@gmail.com	USA	Can edit	inactive
32641	Hannah To		hannah@aol.com	Brazil	Can view	active

A Data Warehouse may still have a few issues in the data but the vast majority should be handled with obvious work arounds.

Summary

- Create Views for your Data Warehouse
- Lightly clean and denormalize your data so that it is easier to query
- Use a modeling tool such as dbt to manage these transformations

Defining a Data Governor

As more people depend on data in their daily workflow, organizations are pressured to think critically about the quality of data being provided. Having a small team field all data questions will not scale, so companies must move from a centralized data organization to a decentralized one.

Why you need a Data Governor

According to Gartner, Data Governance is an effective program to manage and control the ever-growing amount of data in order to improve business outcomes. It helps ensure that the quality of data is high and compliance standards are adhered to. This does not happen with process alone; you need a Data Governor to drive and maintain Data Governance principles.

If your organization has built a Data Warehouse and has any of the following data services or restrictions, we strongly recommend appointing at least one Data Governor:

- Self-service dashboards
- Operates in an industry with regulations and compliance procedures
- Has large data sources spanning different departments
- Strives for operational intelligence

Without oversight, employees will misinterpret data, sensitive data may be shared inappropriately, employees will lack access to necessary data, and employees' analysis will often be incorrect. A Data Governor will maintain and improve the quality of data and ensure your company is compliant with any regulations. It is a vital role to have for any informed company.

Data Governors for Data Governance

With the exploding volume of data within companies, it has become extremely difficult for a small technical team to govern an entire organization's data. As this trend continues, these Data Scientists and Analysts should transition themselves from their traditional reporting responsibilities to those of Data Governors.

In a traditional reporting role, their day was filled with answering questions for various business groups around their needed metrics. The shift to Data Governors finds them instead creating cleaned, documented data products for those end business groups to explore themselves.

This is called Democratized Data Governance, where the technical team (traditionally data gatekeepers) handles the technical aspects of governance and share the responsibilities of analytics with the end business groups.

The Roles of the Data Governor

As the Data Governor, everything addressed in all the chapters of this book is your responsibility. This is your manual. Your role changes at each stage of sophistication. You bravely lead your company from struggling to get value out of its data to producing accurate insights consistently. Let's step through each of the roles you will play.

1. Data Cleanup and Maintenance

The majority of the technical work of data governance is around collecting, cleaning, and maintaining various data sets. This is a many-part activity that's broken out here in subtypes.

Data Piping (ETL) and Warehousing

Data is going to exist in many different places inside of your organization. A big part of your job may consist of bringing those disparate sets of data together, where people can query across various sources. These combined places are data warehouses such as Google BigQuery or Amazon Redshift, and there are various Extract, Transform and Load (ETL) tools out there such as Stitch and Fivetran.

Schema Cleanup/Modeling

For most companies the team collecting the data is also the team reporting on the data. The people on the team know all the ins and outs of the data. They can, for the most part, remember where the data they needed was and what tricky conditionals they'd have to put in each query (for example, not to count deleted or expired accounts).

But when organizations grow and their need to have access grows as well, the people exploring aren't always the ones that put the data there. So, you have to clean up that data with the non-technical data explorer in mind.

Some BI products have ways to do this internally, but often it's best and more reusable to do this on the database level. Just create new schemas in your database with a file full of your chosen views for that consumer. This is both a usability best practice and a security best practice.

Process and Auditing

Manually created data, such as that coming from your CRM, has a large margin for error. Also, how this data is recorded in CRMs is often determined by business users, not by data teams, so governance and data integrity can be less than ideal. For example, there may be two places sales reps need to manually enter the date of a call, or cancellation tracking may change when a new cancellation policy is put in place. Whenever there's manually entered data, there will likely be discrepancies.

The way of handling this is to audit the data, ensure that it's being recorded properly for the needed reports, and identify and develop missing processes with the managers of the relevant teams.

Documentation

Again, the people exploring the data are no longer the people who put it there in the first place. Ideally you've now created clean, curated, and simple models for specific teams. Even so, you'll still find a lot of benefit in documenting each table and column.

This can be done with a Wiki or leaving comments inside the database schema.

2. Permissions and Organization

Data security is obviously incredibly important. But besides that, permissions can be leveraged for proper organization. Data projects can get messy fast. Not everyone needs access to absolutely everything, especially if there is a clear process for requesting whatever additional information is needed.

Organizations today often strive to be highly transparent, but when over-transparency leads to confusion, it's time to make the tradeoff for curating your team's data experience.

3. Integrity Handling

It happens all the time: two people exploring data end up with two different values for the same metric. This can be one of the must frustrating moments for anyone working with data and can lead to some serious mistrust in the integrity of the data.

There's no way of stopping this, but it can be minimized. If the data is kept clean and well documented this problem should come up much less often. The best way to deal with it is to educate everyone on the fact that the problem does happen and they should expect and embrace it. Just as every product has bugs, every dataset does as well. When these inconsistencies are discovered, you have an opportunity to fix/solve/clarify them as soon as possible. One method of clarification is to build standardized metrics in your Data Warehouse model and point people to them when these discrepancies arise.

Ensure there's a clear process for people to resolve these integrity issues. Be available to them and helpful when it's reported. Maintaining a dataset is like maintaining a garden. There will always be weeds growing and more to do. It will never be perfect, but it can be beautiful.

4. Tool Selection

The Data Governor has to make decisions on what best fits your organization's needs. Be mindful of tools that have high learning curves or have proprietary languages that lock you into a tool. Consider all the pieces of your data analytics stack and make sure tools you are selecting work well together.

5. Education/Enablement

No matter how well you've done your data cleaning, documentation and tool selection, you're still going to have to educate your organization on how to use the data to get accurate and actionable insights.

Here are the things you must educate your organization on:

1. What’s available in the models
2. How to use the BI tool
3. Your process for prioritizing data requests, data sharing, and access
4. Data Basics in databases, tables, data structures and SQL
5. Quality versus Vanity Metrics
6. Chart best practices

Data Warehouse Maintenance

Now that you've setup a Data Warehouse, the next and ongoing step is maintenance. This involves making sure the Data Warehouse objects; columns, tables, views, and schemas are accurate and up-to-date. Maintaining your Data Warehouse is integral for users in your organization to easily and accurately gain insights into your data. If it is not maintained people will query the wrong data and get conflicting results.

As a company's Data Warehouse ages:

- New Metrics need to be tracked
- Some old Metrics are no longer needed
- You will need to grant and remove permissions (more than you'd think)
- Modeling will become un-optimized

These inevitable problems make it difficult for your company to conduct analyses. To prevent these issues, you'll need a data engineer familiar with the Data Warehouse, and how users are querying the source. This article will go in-depth on these issues and how to address them with routine maintenance.

Track New Metrics

Why do new metrics matter?

The way we need to measure our business will change over time. We will launch new products, look into different user behaviors, or try to create a predictive model. We need to track new metrics for these different efforts. Sometimes this means creating a new calculated field or a new column, view, or table.

We may add a new field to track our customer information in Salesforce that is inline with our new company objectives, say, tracking account activities through the services we provide. From here, we can see what services are most popular with our customers, then, offer special promotions on these services during seasonal trends where we see a fall in purchases in order to increase sales.

Why do new metrics cause issues in a Data Warehouse?

When engineers or analysts create new tables, columns, or views to track metrics they do not always follow the naming convention set out for the Data Warehouse. This makes interpretation difficult by an analyst unfamiliar with the new metrics.

This can also create duplicate work – say you created a view for your support team but the view along with the pertinent information inside it do not distinguish exactly what this view is for. Users looking to query this view may not know it exists, so they may recreate this view.

How to add new metrics correctly.

When adding new metrics we need to consider:

- How to add to the Schema
- Backfilling data
- Naming Conventions

How to add to the Schema

Do we only need to add the data to an existing table or should we add to a view or create a whole new view? Let's review what reasons we would do each:

Existing Table

- The new metric can be understood and queried easily without complex joins or being aggregated.

Existing View*

- A view that exists that is relevant to the new metric.
- That view is aggregated in a way that fits this data and its dependent metrics.
- Complex join paths would make it difficult for people to query the new data accurately without the view.

New View*

- No view that exists that is relevant to the new metric.
- No view that is relevant is aggregated in a way that fits how this metric should be aggregated without the view.
- Complex join paths would make it difficult for people to query the new metric accurately.

*Typically it will be added to an existing table as well but it will be queried from the View

Backfilling Data

It is advisable to backfill data whenever we can determine what the values should be.

In a dimension table we might be able to determine the value based off of other columns or there is an obvious default value to plug in. If we are not able to determine or have no obvious default value leave the value as null. However, do consider the impact of nulls, such as in aggregation. If this will impact your query try determining a stand in value to indicate you could not backfill it.

In a measures table the same principles hold, so if it can be determined backfill it. Since measures are more often aggregated the impact of nulls can be even greater. The other negative of nulls in a measures table that has time-based data is that it limits your analysis to when you added the new column. Sometimes this issue can be overcome by bringing in data from previous data sources or by inputting values based on overall statistics or dimensions of each row.

Naming Conventions

We also need to ensure these views, tables and columns follow the Data Warehouse's naming conventions. At Chartio, we follow a naming convention when adding new metrics or updating metrics and making sure the name captures the purpose of the updated metrics (Naming Convention and Style Guide). We recommend publishing your style guide and distributing it among your employees as familiarity with the process keeps the naming convention intact.

Deprecate Old Metrics

Why do old metrics matter?

Metrics may become inaccurate or no longer worth analyzing. We want to prevent that data from being queried so others do reach false conclusions.

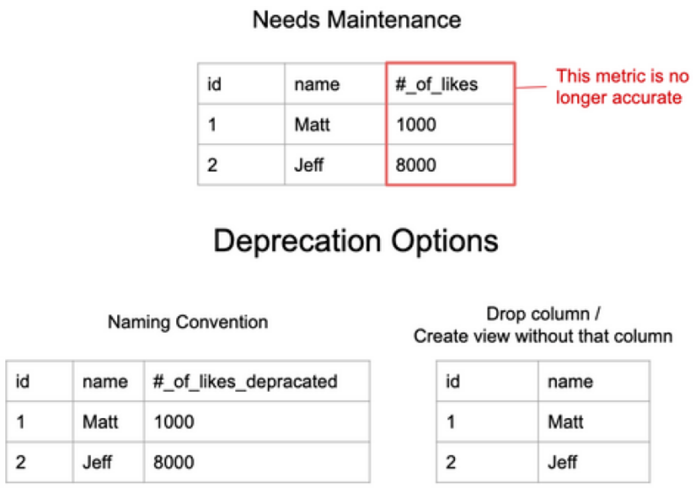
Why do old metrics issues happen?

As companies grow the tools they use to get and move data change. This leads to multiple places where the same type of data can be queried. Since the analyst might not be aware of which source to use they may query the wrong data.

Also business objectives may change. This can affect what data is appropriate for analysis as well. Features or products may have been deprecated as a result and therefore their associated metrics might be misleading if the analyst was unaware of this.

How to deprecate old metrics correctly?

If you're not going to remove columns, tables, or views from user's access right away, we recommend updating the names for these objects to _deprecated, or, _do_not_use. This makes it clear when browsing or querying that they should not be used.



Needs Maintenance

id	name	#_of_likes
1	Matt	1000
2	Jeff	8000

This metric is no longer accurate

Deprecation Options

Naming Convention

id	name	#_of_likes_depracated
1	Matt	1000
2	Jeff	8000

Drop column / Create view without that column

id	name
1	Matt
2	Jeff

This style for old metrics should be incorporated into your company's naming convention style guide. It's also worth letting users know that these metrics are no longer useful through email or with your BI Tool so they're not caught off guard. Again, naming conventions play an integral role in how we keep users from querying data warehouse objects incorrectly.

Handle Permissions

Why do permissions matter?

Not being granted access to the data you need completely halts analysis. Similarly, not removing someone's access can be a legal liability. Having the right permissions is a line that should be carefully considered, but caution should be taken to ensure that it doesn't become a bottleneck.

Consider a scenario where you have users working on multiple BigQuery projects and you're worried about query costs. In order to prevent users from abusing queries and raising the cost, you'll want to create a custom quota. As the BigQuery documentation outlines, a custom quota will manage costs by specifying a limit on the amount of query data processed per day. This can be set at the project-level, or, at the user-level. This type of permissioning is aimed at price reduction, but we can also have permissionings aimed at keeping data intact, or, for privacy concerns.

Oracle offers the ability to recover a dropped table, but, some data warehouses do not have this ability. If you drop a table in PostgreSQL, you won't be able to restore this table unless you restore from a backup. If you can't restore from a backup, there will be no way to recover the dropped table. To avoid this, make sure the proper individuals have the right permissioning on the right Data Warehouse objects.

Why do permission issues happen?

Permission issues happen when the BI Tool or Data Warehouse access does not mirror employee status. This happens when:

- New employees need to query the Data Warehouse
- Employees change roles
- Employees leave
- Special permission
- Account sharing

When you hire analysts, or an individual's role changes which requires more access to the Data Warehouse, you will want to make sure they have the appropriate permissions to analyze metrics in the warehouse. If they lack the proper permissions, this will create a barrier in completing their tasks. Or, if there are no restrictions in what objects they can access in place, those unfamiliar with the Data Warehouse might alter or drop an object.

When employees leave sometimes not all of their accounts are deactivated. You may turn off their email but this would not prevent them from logging into the BI Tool. Their account can remain active which is a security concern.

Sometimes an employee is granted special permission to temporarily gain access to more data. The problem occurs when their temporary access is not revoked after they no longer need it.

Lastly, employees often share accounts to get the access they want for a particular analysis. This is a bad practice as it does not leave a trace as to who is looking at what data.

How to handle permissions correctly.

Making sure your BI Tools access levels mirror the employment status of your employees helps you track users and prevent security issues. The main priority of granting permission is to prevent users from being able to access sensitive information, or, from accidentally deleting data that can't be recovered.

We recommend setting user permissions at a team level because as you scale up your usage and add more users, it's easier to track. Tracking individuals in a small company is trivial since you know everyone by name and when new people are hired and when people leave. In large companies you don't know the vast majority of the people you work with and do not know when their roles or employment status changes.

Sharing accounts is an unfortunate practice that you need to keep an eye on. Sharing accounts makes it impossible to hold users responsible to what action they've carried out. If there needs to be an answer as to why a user dropped a table or updated a column, you might have to ask multiple people to figure it out. This can also be a potential breach of an agreement you may have with your customer and their expectations of how you handle their data. We recommend giving each user a separate account. This ensures security compliance and accountability in the event of an error.

We recommend programmatically adding and removing users to avoid employees from being blocked and to ensure they do not maintain access to data they should not have. If you decide to manually add, remove, or change privileges for users, you'll need to be vigilant in completely removing/updating the permissions.

Tuning to Optimize

Why does ongoing modeling matter?

The amount of data you have will grow as your business grows and your objectives change and you begin to track new metrics. As the data grows, you will need to consider if the way you designed the Data Warehouse objects; schemas, tables, views, and columns still makes sense based on the way users query it.

An indicative sign of needing to revisit Data Warehouse objects or when you should consider remodeling the objects is performance. Performance matters to users, if non-complex queries take too long to run they will stop querying data or start filing tickets to engineering.

Why do warehouses need ongoing optimization?

As your company grows the data that is queried changes too. New data, new analysts, and new business objectives will shift what data is being queried. The original structure of Data Warehouse objects may need to be reconfigured to optimize usage and performance based on how it is queried.

How to continually optimize your warehouse?

Different data warehouses will have options to check performance, but most offer ways to:

- Identify slow queries and add indexes
- Identify common queries and create views

Identify slow queries and add indexes

PostgreSQL has a "slow query" log that lets you set a threshold of an amount of time, if a query takes longer than this threshold a line is sent to the slow query log. From here, the data engineer can determine how to best optimize the most efficient way to run the query, such as, examining the query plan to see how a query is executing and adjust the query to be more efficient.

We can deploy the EXPLAIN command and the EXPLAIN ANALYZE command. The *EXPLAIN* command shows the generated query plan but does not run the query. To see the results of actually executing the query, you can use the *EXPLAIN ANALYZE* command. Based on the output, you can decide to create an index to speed up the query time. To learn more about indexing and the various types, read indexing.

Identify common queries and create views

In PostgreSQL you can use `pg_stat_statements` to group identical queries and find optimization opportunities. The `pg_stat_statements` directive stores queries that are run against your PostgreSQL instance. It saves the query, the execution time, the underlying reads and writes, and the variables. This information allows you to determine what type of data users want so you can optimize frequently used queries.

Creating a view can help users unfamiliar with the structure of the data warehouse by consolidating what they need to query to a single place. For example, you can grant users from a specific department access to a view that reflects all the departmental information they need to query.

In addition, you can get performance benefits if you materialize the view or create a new table. Most of the improvements here will be seen if the query is heavily filtered or if it is aggregated. Users can then query the materialized view or table. You can get an even bigger bump in performance if you add an index to this new materialized view or table.

To learn more about the pg_stat_statements, I recommend reading the following articles; The most useful Postgres extension: pg_stat_statements and pg_stat_statements: The Way I Like It.

Summary

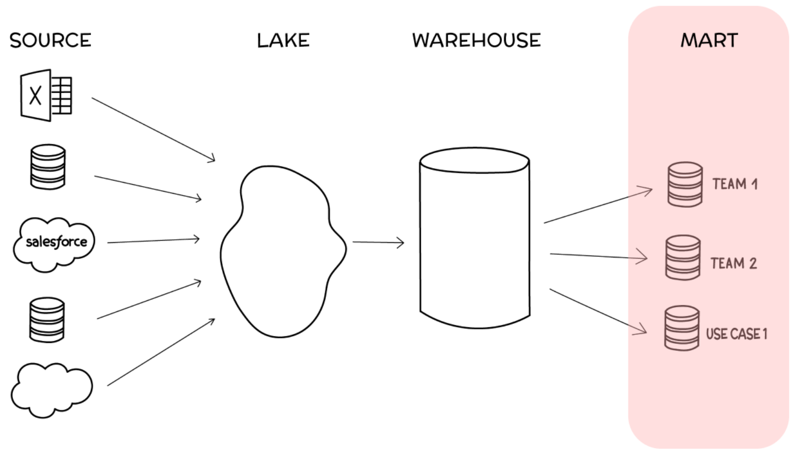
A Single Source of Truth Data Warehouse is a worthwhile investment. However, without maintenance it will fall into disarray and lose its value.

- As metrics are added, make sure they're named properly.
- As metrics are deemed no longer useful, make sure they're removed to avoid confusion.
- As you vet your metrics and find that some need to be updated/pre-aggregated, make sure they're named properly.
- Keeping user permissions appropriate and accurate will free up database admins to focus on important projects as well as avoid data being removed accidentally.
- Considering the restructuring of Data Warehouse objects will help create a suitable structure for analysis and complex querying along with cutting down performance cost.
- The worthwhile investment of a data engineer to perform said maintenance tasks will remove the bottleneck of incorrect analytics from a neglected warehouse.

References:

- Claire Carroll - Fishtown Analytics: Five principles that will keep your data warehouse organized
- Claire Carroll: The difference between users, groups, and roles on Postgres, Redshift and Snowflake
- Don Jones: Three data warehouse maintenance tips for DBAs
- BigQuery Documentation: Creating custom cost controls
- Stack Over Flow: Can I rollback a transaction I've already committed? (data loss)
- CYBERTEC: 3 Ways to Detect Slow Queries in PostgreSQL
- CYBERTEC: pg_stat_statements: The Way I Like It
- Citusdata: The most useful Postgres extension: pg_stat_statements

Stage 4 - Mart


SOURCE
LAKE
WAREHOUSE
MART
salesforce
TEAM 1
TEAM 2
USE CASE 1

Why Build Data Marts

What is a Data Mart?

A Data Mart is a filtered (and sometimes aggregated) subsection of a Data Warehouse to make it easier for a particular group to query data. It provides a smaller schema with only the relevant tables for the group.

This stage is right for you if:

- You want to get democratized and enable others in your company to explore and understand data themselves
- You're prepared to teach and enable business users in your company - hopefully using the many resources of The Data School
- You have projects that require different formats of the source of truth for easier use
- Having truly informed employees is important to your company's competitive success

You've outgrown this stage if:

- You can't really! You can make any number of marts, and even put leveling in your marts if you'd like. Implementing this stage will result in a complete, well architected and governed stack that will continually evolve and support your informed competitive company.

Five reasons to build a Data Mart

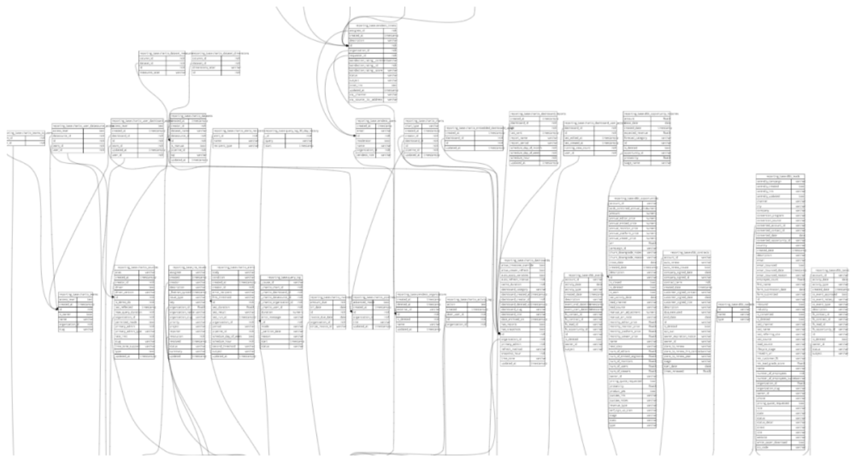
1. **Relevance** to use cases. Limiting the schema to the tables that you need allow you to parse the schema easily.
2. **Accessible** to a variety of people and teams. Data marts allow you to expose more people to data without overwhelming them.
3. **Customized** architecture for different use cases. Aggregations, metric calculations, and PII can all be handled individually for teams.
4. **Maintainable** with less time and effort. Having the data monitored by team leads makes it easier to identify data issues.
5. **Separated** levels of data access. Easily protect sensitive data by limiting what teams can see in their data marts.

This section of the Data Governance book will explain why you should create data marts, and how to implement them so that you get all the benefits they can deliver your business. Before we dive in further let's look at the data issues you are facing with a Data Warehouse.

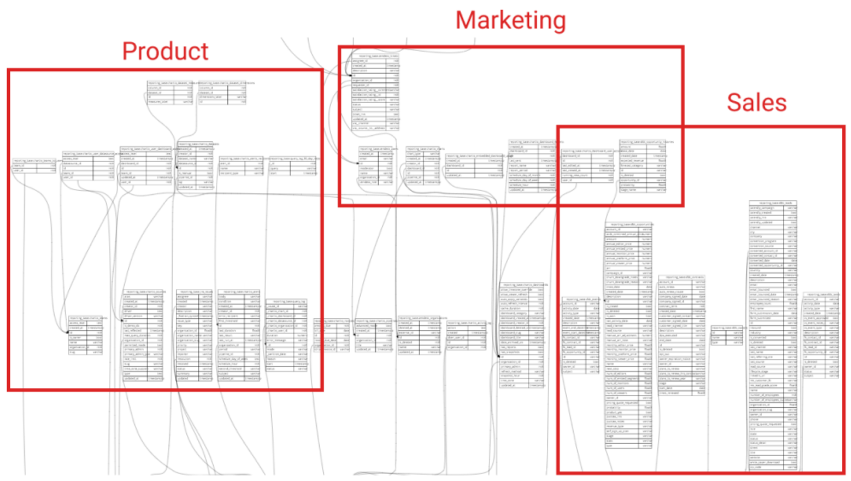
The Problem with Data Warehouses

As an organization scales the amount of data it is tracking, the number of people who want to access it scale too. This results in more people with less context about a large portion of the schema.

We want to go from a complex schema:



To a siloed schema, where each department has the data they need:



So while going from Lake to Warehouse was mostly about cleaning up tables, going from Warehouse to Marts is about cleaning up schemas. Different departments need different parts of the Data Warehouse schema.

Table	Product	Marketing	Sales
Customer	✅	❌	✅
Marketing Channel	✅	✅	❌
Leads	❌	❌	✅

How Data Marts are different from Data Warehouses

Use modeling to create separate schemas where the tables are provided to the appropriate team or individual. These will be your company's Data Marts.

The table structures should be the same, as the data should have been cleaned at the Data Warehouse stage. Data Marts are not very different from your Data Warehouse since the heavy lifting was already done. Data Marts make it easier for people within departments to navigate the schema and provide extra insight of the data for that department.

Summary

Data Marts make your data:

- **Relevant** to your job and use cases
- **Accessible** to a variety of people and teams
- **Customized** architecture for different use cases
- **Maintainable** with team leads
- **Separated** to protect sensitive information

A Data Lake is a pile of products in your building.

A Data Warehouse is those products sorted, shelved, and tagged.

A Data Mart is those products shipped out to relevant stores for sale.

Data Mart Implementation

As companies grow, the amount of data and the number of sources they have will also increase. This leads to your Data Warehouse having numerous schemas that can become difficult to navigate. Moving from a Data Warehouse to Data Marts reduces the scope of access and makes it easier for users to find the data they need. Data Marts can be created in five steps.

1. Views

Marts should be created with Views, not by creating new tables.

For most companies there is no need to materialize views as the performance should not be that different. However if you are running into performance issues it can be worth trying materialized views.

2. Use the Data Warehouse

Any large cleaning should be avoided at this stage. You should be selecting the relevant views and filtering out unnecessary columns from the Data Warehouse to build out each Data Mart.

```
CREATE VIEW
SELECT *
FROM DataWarehouse.View
```

Most if not all of the cleaning should have occurred when going from the Lake to the Warehouse, if there is a cleanliness issue address it with modeling in the Warehouse stage.

If you do want to do some additional modeling to create aggregations for performance reasons that is fine, and if you want to combine data to make it easier to analyze we recommend using the wide table approach versus implementing something more complex like star schema.

3. No Star Schema

The performance benefits of star schema no longer exist. It is a lot of work to implement. While some people argue it is easier to query after being set up this way, modern BI tools such as Chartio have created interfaces to the data which solve problems such as complex joins.

4. Segment tables

Determine how you are going to split the data into different Data Marts. Common ways include:

- Department
- Product Line
- Use Case
- Region
- Security considerations

Create a matrix that contains the table names and the segments you are splitting up the data by to determine which group has access to what. Then you can create the relevant views for each Data Mart.

5. Access Update

Prior to implementing Data Marts, you likely had provided all of these groups access to the Data Warehouse. You should remove everyone’s Data Warehouse access by default and grant them access to the mart or marts they belong to.

A few people might need to retain access to both. Let them ask so that you know who has access to what. These people may still want to query the Data Warehouse when they want to analyze data that would span multiple marts.

Summary

- Use Views
- Don’t deviate that much from the Data Warehouse Views
- Do not use a Star Schema
- Segment your marts

- Update access to be at the mart level instead of warehouse

Data Mart Maintenance

Now that you have Data Marts set up, you will need ongoing maintenance to get the most out of your data. The first step is to establish a mayor per mart that will be responsible for carrying out the maintenance tasks for their mart:

- Communicate and educate the team
- Identify issues
- Identify new needs

Establish Mart Mayors

This role is similar to the Data Governor for the Data Warehouse. Data Governors delegate down to Mayors, who in turn, take care of Governor tasks at the mart level. Mayors, therefore, communicate with and educate the team using their mart. They are responsible for identifying issues in the data that exist in their mart. They also should be the ones creating requests to get more data sources or tables added to their mart.

An Ideal Mayor

Communicate and educate team

Different teams have different needs, but some common threads include teaching SQL skills or how to use your BI tool. Mayors should document and share data quirks that show up in common queries. Note, however, that you should try to address these quirks with modeling at the Data Warehouse stage.

We wrote a book on how to teach SQL if you need assistance in explaining how JOINs, Aggregations, or subqueries work.

Identify issues

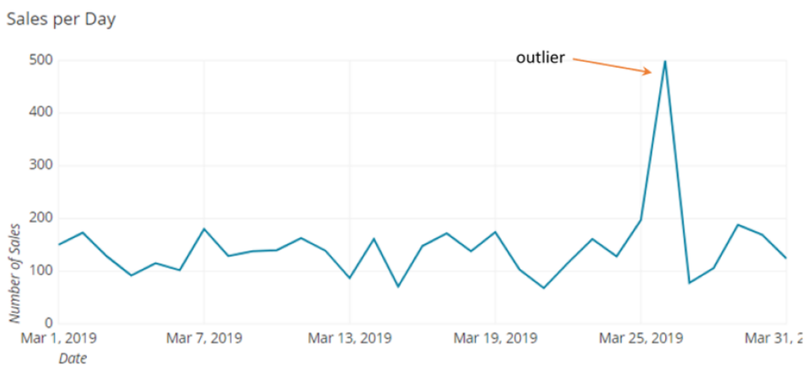
There are two types of issues you will need to investigate as a Mayor of a Data Mart.

- Data that doesn't make sense
- Common data errors

Data that doesn't make sense

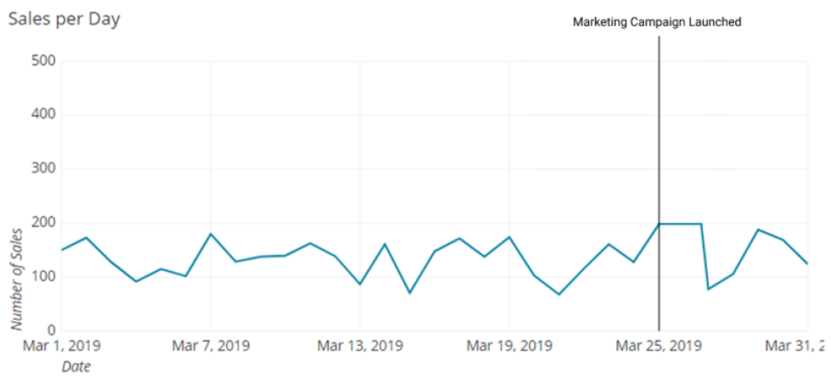
Spike

Something to pay attention to are numbers that are much different from the day before, but not caused by any changes on your end. If traffic to your website doubles in a day, it is likely caused by something and is not just a fluke. The spike may have been caused by a new marketing campaign, a bug, or potentially a Google search algorithm update. You should explore these possibilities in that order.



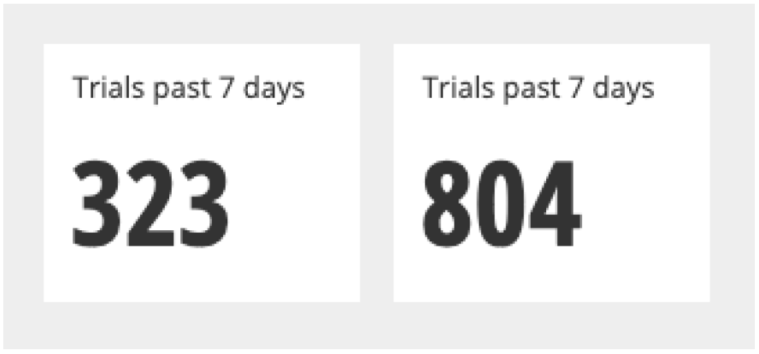
No Spike

Another point of interest are numbers that are not changing even though changes have been made. If you launched a new marketing campaign and the numbers are not going up, that could be due to a bug or poor campaign performance. They should be investigated in that order. Often the tracking was not set up correctly or the link in the advertisement was going to the wrong place.



Conflicting numbers

Sometimes metrics can be showing conflicting numbers even though they are measuring the same thing. If you see the number of new trials in HubSpot and your production database are different, the rule of thumb is to trust the data source closest to the event that is being tracked. In this case, it would be production.



```
SELECT COUNT *
FROM USER
WHERE Trial_Start > NOW()::date – 7
        AND Email != “%chartio.com”
```

vs.

```
SELECT COUNT *
FROM USER
WHERE Trial_Start > NOW()::date – 7
```

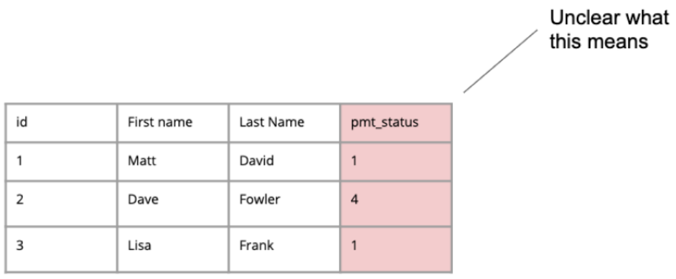
Oftentimes people’s calculation of a metric will differ because of the following reasons: They are calculating it based on a different formula, the data source they are using is different, the data is being filtered differently, or there is an error in their calculation.

Common data errors

- New field or value not cleaned (nulls, encoded, wrong format, etc.)
- No new data
- All queries on a data source erroring out
- Performance

New field or value not cleaned

You will likely notice when there is a new column in one of your views that is not very clear. You should raise this to the Data Governor so that they can apply the necessary cleaning to it at the Data Warehouse stage. Avoid doing additional cleaning at the Mart level because others may need this field as well. Having a single version of it helps to make sure analyses are consistent.

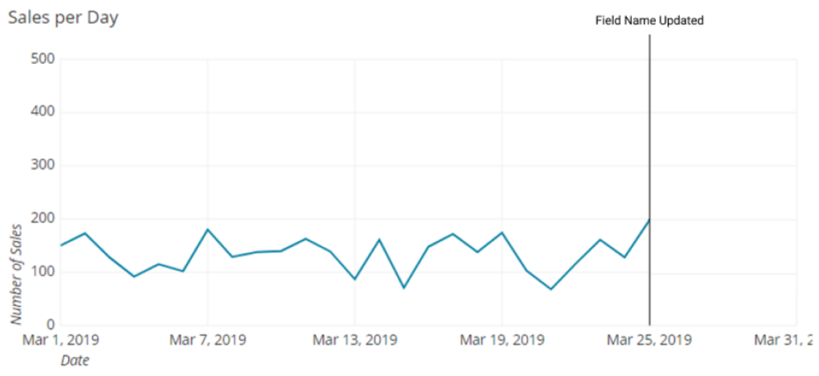

Unclear what this means

id	First name	Last Name	pmt_status
1	Matt	David	1
2	Dave	Fowler	4
3	Lisa	Frank	1

This can also happen when a new option is added to a field and it is encoded in an unreadable way. Follow the same process to get it updated at the Data Warehouse level.

No new data

If your query stops producing data after a specific date, you will need to investigate. This can be caused by a bug, a field being renamed, or the data source changing. This is more common than you would think. For instance, if you update a URL the data associated with the previous name will cut off. You can work around this within your SQL query.



For example, when Chartio moved its URL from Chart.io to Chartio.com we needed to use:

```
SELECT
CASE WHEN page_tracking.url LIKE 'www.chart.io'
        THEN 'www.chartio.com'
    ELSE page_tracking.url
    END AS “Page”,
to_char(page_tracking.viewed_at_date, ‘YYYY-MM’) AS “Month”,
COUNT(distinct page_tracking.view_id) as “Views”
FROM page_tracking
GROUP BY 1
ORDER BY 2 ASC
```

You can also implement this as a more permanent fix at the Data Warehouse stage. One note of warning here is that sometimes you want to preserve this cut off to remember the name was changed, so think through the implications before making this modeling decision. If you aren’t sure why the data cut off, consult the Data Governor or your engineers to find out what is going on.

All queries on a data source are erroring out

This happens for a few reasons: the source has been deprecated, the source had an update changing its data structure, or a bug. This is something to communicate out to your team quickly as it can prevent a ton of data from being used.

Performance

If queries by you or your team start to take over a minute to run, you should investigate. Can the queries be optimized? Do we need to spin up more clusters? Or, do we need to do some pre-aggregation?

These are all fairly advanced solutions. To learn how to optimize the SQL, read our book titled *SQL Optimization*. To spin up more clusters you will need to consult with your engineering team and the Data Governor. To do pre-aggregation you should consult with the Data Governor and create a new view at the Data Warehouse level so others can use this newly formed view.

Identify new needs

Data is never a static thing. As new features roll out, new tools get used, objectives are set, and new data needs will emerge for your team. Do not assume that your mart will be updated when any of these changes happen. You need to proactively advocate to make sure your mart is updated in a timely manner.

Extras

Evaluating Data Stack Technologies

What is a Data Stack?

In software development, a Stack is a combination of technologies that together solve a problem. Rarely does one technology solve it on its own. Stacks are usually given names relevant to the problem that they solve. For example, a Web Stack is the set of technologies that host an application that can be accessed via a website. We need to know about the front end technologies used and the back end. We could say "our Web Stack is Cassandra, Django, React" which gives them a high-level overview.

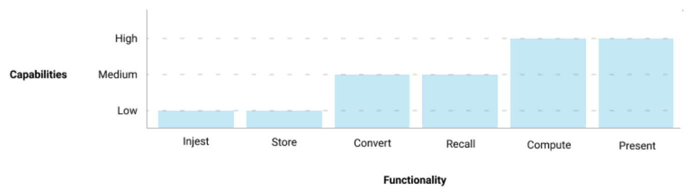
We can apply the Stack concept to describe solutions in domains other than web development. Here we will describe the Data Stack, which takes data from a raw form to an insight.

The Data Analytics Problem

Every software tool spins off data: your application, Google Analytics, Salesforce, and so on. In order to use this data to find insights you need a Data Stack that performs the following functions:

1. Ingestion - Accept new data
2. Storage - Store and retain data
3. Conversion - Prepare data for future use
4. Recall - Enable basic access to data
5. Computation - Calculate metrics from pre-processed data
6. Presentation - Format results into tools that improve human understanding

Normally the elements of a Data Stack are composed to perform these functions in this order, with raw data flowing into the ingestion point and becoming more and more refined until it finally is prepared enough for people to understand. We will use a visual to show how different BI tools map to these functions and what level of capabilities they have for that function. Here is an example graph:

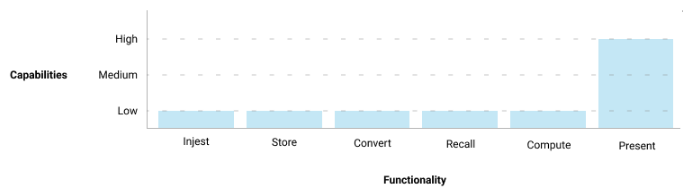


Fitting Technologies Into the Data Stack

Microsoft Excel

Technologies such as Microsoft Excel could be your entire Data Stack. It has the functionality to do each step.

It can ingest a csv, store an xlsx file, convert the data to be more usable, recall the data so it can be used, compute aggregations on the data, and visually present the information in a chart. Let's take a look at how well it performs each part of the Data Stack:



Let's examine why we rated **ingest** low and **present** high.

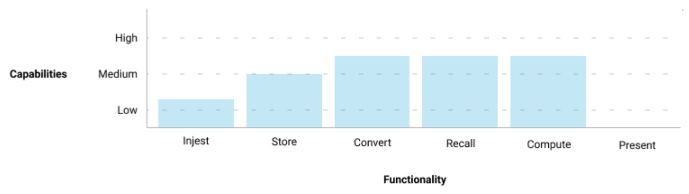
Ingest

- Capacity - Being a desktop application, Excel cannot handle significant volumes of data. It is limited by the computer it is installed on and Excel itself has a maximum row count of 1 million. When tasked with processing a low volume of data Excel can do quite well and is frequently used for analytics.
- Data Formats - Excel can easily work with excel files and comma or tab separated files, but it does not work well with JSON, XML, or many other file types.

Presentation

- Data Visualization - When it comes to presentation, Excel has most data visualization types. It also can be used to present interactive calculations and tables of data for people to consume. Excel's scale limitations are not an issue for this phase alone because visualizations are not built directly from large data sets: a chart that includes thousands of data points will likely be unreadable.

Postgres



Why does Postgres do a better job at **Store** and **Compute**?

Store

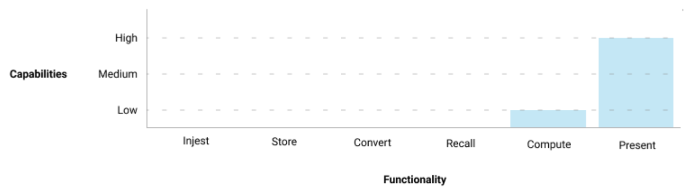
- Postgres can easily store a volume of data that exceed Excel's capacity
- Data stored in Postgres is generally easier to backup

Compute

- It is easier and cheaper to dedicate more powerful hardware to Postgres than to distribute more powerful desktop or laptop computers to staff that use Excel
- Postgres is better at handling relational logic, which can be a significant component of preparing data sets for presentation materials

At this point many readers might be thinking that Excel has advantages in Compute that I have glossed over to make my point. This is true, and the suitability of technology to perform each of these workloads cannot be reduced to a single number. The capabilities of each technology have been oversimplified so that it is easier to make simple illustrations that help to explain how different technologies complement one another.

Chartio



From the above we see that Postgres does not have any ability to **Present** data. A tool like Chartio can help with this:

Compute

- Chartio supports an interface for interacting with data stored in databases like Postgres that is easier to work with than what those databases support by themselves
- Chartio can perform a range of calculations on query results at a small scale, but most of the computational load has to be handled by a database

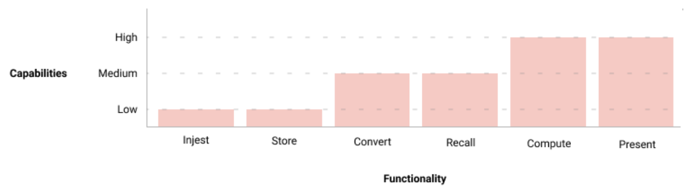
Present

- Chartio has many features related to producing and sharing meaningful and interactive data visualizations

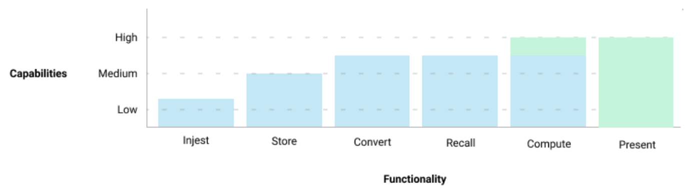
The combination of collection of meaningful data sets from a database and production of good visualizations from these data sets makes Chartio an excellent complement to most databases.

Meeting Your Requirements

Suppose that a mobile game developer wants to track user behavior within the context of the mobile app. This might require tracking dozens of distinct events per session that might represent actions like completing a level, viewing a screen, or clicking on a banner ad. If this game attracts thousands of users per day, the volume of raw activity data will exceed Excel's capability in short order. A detailed analysis of the requirements might produce a rough profile such as this:



Where the need in excess of Excel's capabilities is largely due to raw data volume. Naturally this suggests that a solution that can handle a greater volume of data would be appropriate. A simple combination of Postgres and Chartio might have combined capabilities that look roughly like this:



Growing with the Business

As a business or product grows it is natural for demands on the Data Stack to change. One way to accommodate changing requirements is to make the Stack more capable in key areas by adding or replacing technologies in the Stack. In the next chapters of this book, we will explore the evolution of a Data Stack in more detail. We will explore which technologies enhance each other and which are incompatible.

There are some key factors to keep in mind when building your Stack in a growing organization:

1. Use technologies that are easily configurable to work with a range of complementary technology. Otherwise there may be few choices for critical parts of your Stack.
2. Having one component cover a wide range of functions potentially complicates upgrading it and may make it more likely that it has to be replaced.

Communication is the Goal

More powerful Data Stacks tend to be more complicated, and this complication naturally makes it harder to understand things well enough to know how to contribute or how to evaluate risk. With a written out Data Stack, it is easier to explain how the various technologies work in concert to accomplish the end goal.

This model presents a fairly simplified view of the journey of data from raw feeds to presentation. It allows more people in the organization to understand the problems and solutions of data analytics from a high level.

ETL vs ELT

How should you get your various data sources into the data lake? Well there are two common paradigms for this.

1. **ETL** is the legacy way, where transformations of your data happen on the way to the lake.
2. **ELT** is the modern approach, where the transformation step is saved until *after* the data is in the lake. The transformations really happen when moving from the Data Lake to the Data Warehouse.

ETL was developed when there were no data lakes; the staging area for the data that was being transformed acted as a virtual data lake. Now that storage and compute is relatively cheap, we can have an actual data lake and a virtual data warehouse built on top of it.

We recommend ELT because

1. We care more about Extracting and Loading the data into a common place at the Data Lake stage.
2. Data on a lake will go through heavy transformations during the next stage, so there is no need for complex logic before the data is loaded. The transformation step of ELT happens in the data warehouse.
3. We can end up with a much simpler architecture which means less problems and less maintenance.
4. Data lineage becomes easier to track as complex transformations are not happening prior to loading the data into the lake.

Light transformation of the data before loading the data into the lake might still be necessary:

- **Column Selection:** Select the data that really matters. For example, not everything in Salesforce needs to be synced.
- **Privacy reasons:** for example, filtering out columns that contain PII (personally identifiable information). Instead of filtering, you might want to hash PII data so they can be used for your analytics.

The above transformation cases can be included in the ELT paradigm and are offered by most commercially available ELT vendors.

Acknowledgments & Contributions

This book has been a large effort by a number of people and continues to evolve with community involvement. We aim to keep this book always "Modern" with updates, contributions, reviews easily submitted via email or committed through Github - where the source for this whole site is kept.

Thank you's

I want to give a special thanks to Matt David, the Head of the Data School for working on the start of this for so many hours with me. He has project managed, refined ideas and written the majority of this book and continually recruits community involvement. His passion for educating the masses on data is truly inspirational, and a joy to work with.

I also want to thank Tim Miller, who wrote many chapters here and worked many stressful hours on content with me. Thank you Tim for your constant drive and passion for both learning and educating.

Also I'd like to thank the rest of the Chartio Data Advisor team, who each contributed a chapter and who daily take the learnings in this book directly to our customers.

Thank you Steven Lewis for your always excellent design and illustrations that honor this book and site. Thank you Eleanor Preston for helping me work through this framework on so many customer visits - and persistently waiting for us to get it published!

Thank you Kostas Paradalis of Blendo for your article on ETL vs ELT.

How to contribute

Our goal is to continually make this book better and kept modern. We would like to expand it, like a wiki, to cover more topics, go more in depth, share more real company examples, and be better reviewed and edited.

Few are complete "experts" in all of the areas of modern data governance, and the landscape is changing all of the time. If you have a story to share, or a chapter you think is missing, or a new idea - email us or create a pull request with the edits on our github repo.