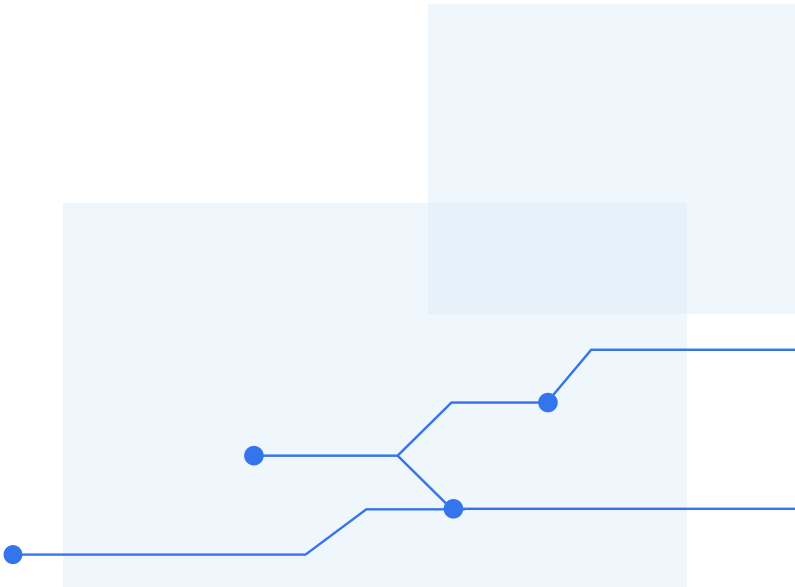
MONTE CARLO

Data Pipeline Monitoring 101

A better way to build reliable, high-quality data pipelines

Table of Contents



I. A New Era of Data Quality

A data renaissance is on the horizon, but to unleash its true potential, data leaders need to prioritize reliability and trust.

Just as SaaS has moved from only powering websites to being trusted with increasingly critical tasks from banking to real-time navigation, in many organizations data is moving from solely powering executive dashboards to creating valuable data applications such as machine learning models and real-time marketing applications.

However, as data becomes increasingly important to modern companies, it's crucial that teams trust their data is accurate and reliable. Unfortunately, poor data quality and inefficient processes are the cold realities for most organizations. Consider:

- The average organization suffers approximately 70 data incidents a year for every 1,000 tables in their environment.
- Data professionals are spending a whopping 40% of their time evaluating or checking data quality.
- Nearly 50% of data professionals estimate their business stakeholders are impacted by issues the data team doesn't catch most of the time, or all the time. They also estimate poor data quality impacts 26% of their companies revenue.

Interested in more industry insights and reactions from experts?

Access our 2022 State of Data Quality ebook

Why is this the case and what can be done to change it? Let's take a look at current data quality monitoring best practices and their limitations.

Current data monitoring approaches and their limitations

For the past decade, the first-and-only line of defense against bad data was testing.

Similar to how software engineers would use unit tests to identify buggy code before it was pushed to production, data engineers would leverage tests–either hardcoded in the pipeline or using an open-source tool like dbt or Great Expectations–to detect and prevent potential data quality issues from moving further downstream.

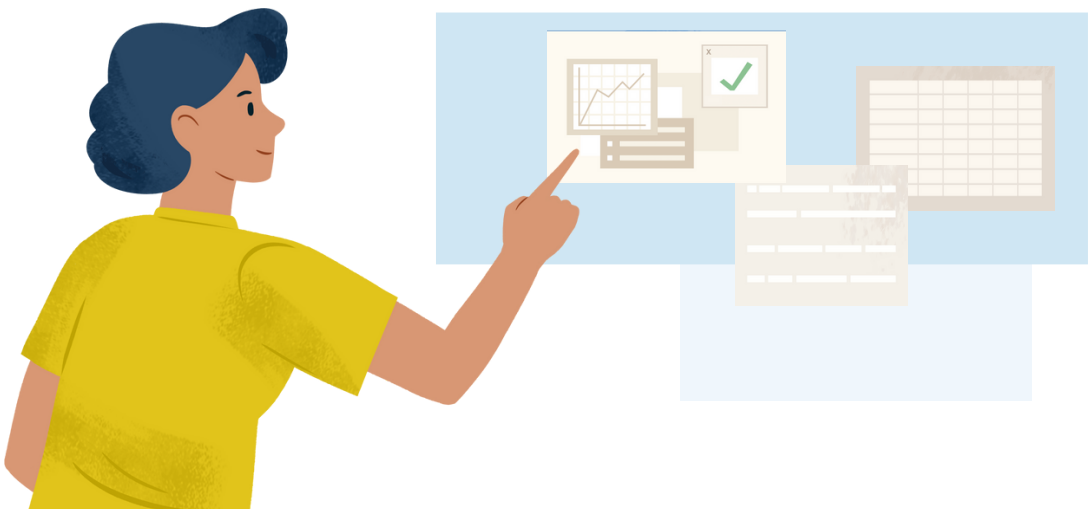
Data testing is a good solution for specific, well-known problems. But just like a spoon is a good tool for eating ice cream and a terrible one for digging a house's foundation, so too is data testing a poor tool for engineering reliable data systems at scale.

For example, threshold setting can be tricky. It is easier to create a test when the column can never have a NULL value than when some percentage of NULLS are expected some of the time.

"Whether it's custom SQL rules or dbt tests, you have to do that upfront configuration. You have to know in advance what it is you're going to monitor, and go through the process of setting it up." said Edward Kent, Technical Lead, AutoTrader UK.

"For us, we have hundreds of data models defined and hundreds of tables built daily. We needed something that would effectively get this off the ground and running without us having to put in that effort."

Read Auto Trader UK's Story



Another challenge is you can't anticipate all the creative ways data can break–there are just too many interdependent, moving parts across too many teams. We call this the unknown unknowns problem.

To emphasize this point, we know of a data team that built hundreds of data tests on a single, business-critical pipeline...and still experienced data incidents with some regularity. Can you imagine trying to scale this process across all of your production pipelines?

"We had a lot of dbt tests. We had a decent number of other checks that we would run, whether they were manual or automated, but there was always this lingering feeling in the back of my mind that some data pipelines were probably broken somewhere in some way, but I just didn't have a test written for it," said Nick Johnson, VP of Data, IT & Security, Dr. Squatch.

Read Dr. Squatch's Story

And of course, the other challenge with data testing is the time required for a task so tedious that execution is typically spotty at best.

Data engineering leads will kick off their weekly meetings and talk about the importance of adding unit and end-to-end tests within all production pipelines until they are blue in the face. Everyone on the Zoom call nods and tells themselves they will rededicate themselves to this effort.

Some teams create tight enough processes that they successfully create and maintain hundreds even thousands of tests. Which creates the problem of...creating and maintaining hundreds even thousands of tests.

Software engineers still use unit tests, but they reached 99.999% uptime thanks to real-time monitoring and observability. It's time for data engineers to utilize these approaches as well.

Why you can't out-architect bad-data

Another misguided data quality approach some data teams will take is to put 100% of their efforts into building systems that don't break. While a noble goal, bad data is inevitable.

Many ETL pipelines are, at their heart, a communication between two different teams, often at different companies," said Rick Saporta, Head of Data Strategy and Insights, The Farmer's Dog.

"When one team makes a change, it affects the other. In the best of instances, you might have strong communication between the different teams, but all pipelines break at some point, and it's hard to anticipate all the different ways that data can break."

Read The Farmer's Dog Story

Consider these scenarios that are completely out of the data engineers control:

- Human error — No data architecture is perfect because these systems involve people and humans aren't perfect. Bugs enter production, wrong numbers get entered, and transformation model mistakes get made.
- Third-party dependencies — Virtually every data ecosystem consists of numerous third-party data sources whose quality cannot be controlled. Even the largest, most disciplined third-party data sources send data late, change how data is delivered, or send bad data at times.
- New initiatives — As one of our colleagues likes to say, "data is like fashion, it's never finished – it's always evolving." Even in a hypothetical world where you have finished the last brushstroke on your architectural masterpiece, there will be new requirements and use cases from the business that will necessitate change.

That's not to say data teams shouldn't invest in best practices like data contracts, certification, or SLAs. Those things do help reduce data incidents.

But data teams do need to approach bad data as inevitably as security teams approach successful cyber breaches, and invest in layered defense and mitigation strategies as appropriate. As we saw in the previous section, it's also the same approach that software engineers have taken towards application reliability for years: data monitoring and observability.

What is data pipeline monitoring? What is data observability?

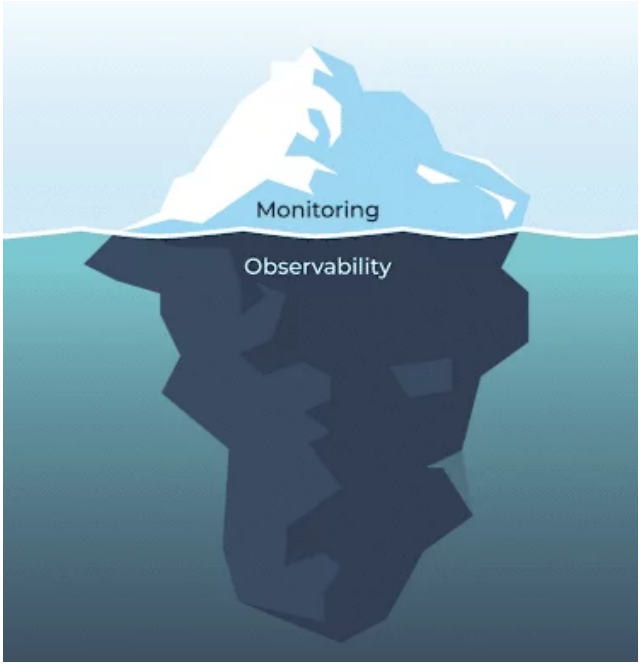
Data pipeline monitoring involves using machine learning to understand the way your data pipelines typically behave, and then send alerts when anomalies occur in that behavior.

This includes:

- Freshness—Did the data arrive when it should have?
- Volume—Are there too many or too few rows?
- Schema—Did the way the data is organized change?

Some tools within the modern data stack, like Airflow for instance, will have the ability to monitor their portion of the ETL pipeline.

While helpful, data teams need to monitor their entire pipeline end-to-end from ingestion to landing and through transformation all the way to consumption in the BI layer.



It is also important that data pipeline monitoring is supplemented with a process for monitoring the data quality itself. This is because while the pipeline may be operating fine, the data flowing through it may be garbage. For example, the data values may be outside the normal historical range or there could be anomalies present in the NULL rates or percent uniques.

Monitoring the data itself can be done automatically with machine learning as well as by setting custom rules, for example if you know a monetary conversion rate can never be negative.

When automated data monitoring is combined with features to accelerate incident resolution, understand the impact of those incidents, and illustrate data health over time, it then becomes data observability. This technology category often includes features such as:

- Automated data lineage—With a real-time map of all system and table dependencies down to the BI layer, data teams can understand the impact of an incident downstream and trace the root cause to the table furthest upstream.

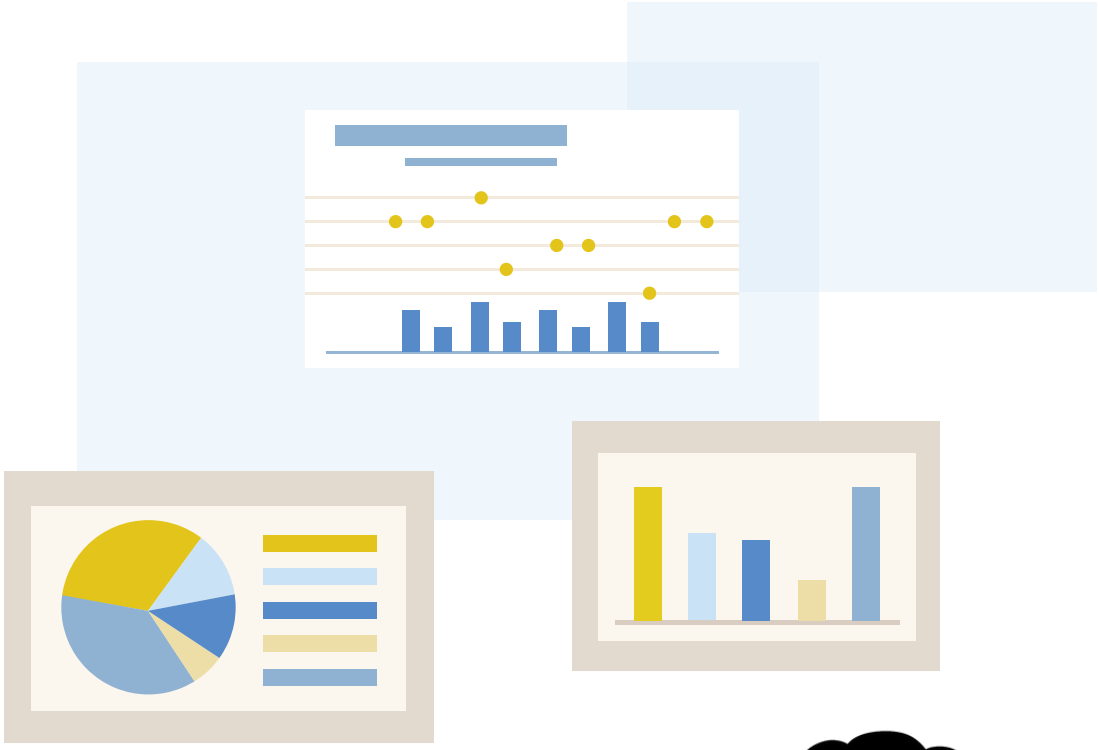
- Anomalous Row Distribution—One way to pinpoint issues with the data itself is to find "bad rows" that break user-defined business logic. While you could manually run multiple queries to segment the data, this can also be automated in a visual format to help accelerate the resolution process.

- Query Change Detection—Did an anomaly arise within a table at the same time as a query acting on that data was modified? Or perhaps at the same time a dbt model was updated? These correlated events are often causal as well.

- Data Insights—Data health insights can move from reactive resolution to proactive prevention by highlighting issues such as unused tables or deteriorating queries. Dashboards that point to particularly problematic areas also enable more optimized resource allocation.

“How well are the systems performing? If there are tons of issues, then maybe we aren’t building our system in an effective way. Or, it could tell us where to optimize our time and resources. Maybe 6 of our 7 warehouses are running smoothly, so let’s take a closer look at the one that isn’t,” said Brandon Beidel, Director of Product Management, RedVentures.

Read Red Venture's Story



II. The Value of Monitoring and Data Observability

In one sense, the value of data pipeline monitoring and data observability is near priceless. An organization cannot be data-driven, and a data team cannot add value, if no one trusts the data.

"Data observability has become a necessity, not a luxury, for us," said Alex. "As the business has become more and more data-driven, nothing is worse than allowing leadership to make a decision based upon data that you don't have trust in. That has tremendous costs and repercussions."

Read Fox Digital's Story

Not only can this lead to strained relationships with internal stakeholders, in cases where the data is or is part of the product, it can impact customer satisfaction and the bottom line.

"Without a [data observability tool], we might have monitoring coverage on final resulting tables, but that can hide a lot of issues," said Adam Woods, CEO, Choozle. "You might not see something pertaining to a small fraction of the tens of thousands campaigns in that table, but the [customer] running that campaign is going to see it. With [data observability] we are at a level where we don't have to compromise."

Read Choozle's Story

Introducing data downtime

That being said, it's unlikely the chief financial officer is going to accept "priceless" when you are building your business case. So let's take a look at how data teams have measured data quality.

A simple calculation for the estimated number of incidents you have each year (whether you are currently catching them or not) can be done by dividing the number of tables you have in your environment by 15.

You can then multiply this number by your average time-to-detection and average time-to-resolution. If you aren't currently capturing these metrics, don't worry, you are not alone. Our industry research revealed the industry average is about 4 hours and 9 hours respectively–feel free to use or adjust those estimates based on your organization's data quality maturity.

Congratulations, you have just calculated your data downtime! This is the period where data is inaccurate, missing, or otherwise erroneous.

The ROI of reducing data downtime

Now let's calculate how much data downtime costs your organization. The first part of the calculation, labor cost, is relatively straightforward.

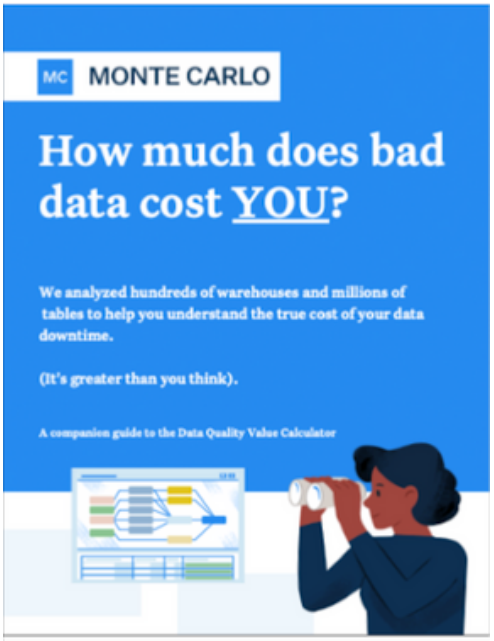
Since we know data quality professionals spend around 40% of their time on inefficient data quality practices, we can use this formula:

Total data engineers x 1804 (average working hours in a year) x $62 (average salary) x .4

The operational cost of poor data quality is a bit harder to quantify. A data incident can be as harmless as a broken dashboard no one uses or as painful as reporting incorrect numbers to Wall Street.

One way to estimate this is to measure the overall risk. If an organization invests in its data team to increase overall efficiency 10% (or insert your own value here) then for each hour of data downtime we can assume the organization's productivity has been reduced 10%. The formula is then:

Overall data downtime x %reduction in hourly revenue generated.



Want more detailed calculations and business case justifications? Access our "How much does bad data cost YOU?" ebook.

The case for build vs buy

It's not uncommon for data teams to consider building their own data quality solutions in-house. Here are a few things to consider in your calculations.

- Expected time-to-value—Like any in-house solution, designing, building, scaling, and maintaining an internal data quality solution will take time, money, and headcount. If you have an extensive data engineering and data science team with a significant amount of extra time on their hands, then building could make sense—but at most companies, lack of work for data teams is rarely an issue. When the data team at a leading insurtech provider investigated building a data observability solution, they realized it would take 30 percent of their data engineering team to build a comprehensive anomaly detection algorithm, a solution that would cost upwards of $450,000/year to build and maintain. They chose to buy instead.

Resource	Monthly cost	Number of resources engaged	Number of months engaged	Total cost
Data Engineer *(build)*	$17,500*	3	9	$472,500
Data Analyst	$14,000**	.5	3	$21,000
Data Engineer *(maintain)*	$17,500*	.25	3	$13,125
				$506,125

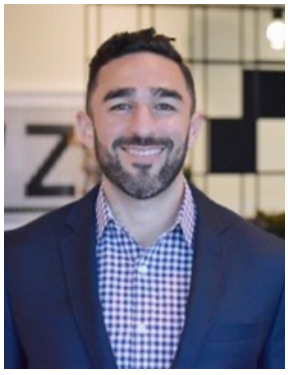
*Data Engineer fully loaded annual cost: $210,000
~$140,000 total pay + additional 50% in benefits + taxes = $210,000
Source: https://www.glassdoor.com/Salaries/san-francisco-data-engineer-salary-SRCH_IL.0,13_IM759_KO14,27.htm

**Data Analyst fully loaded annual cost: $168,000
***$112,000 total pay + additional 50% in benefits + taxes = $168,000*
Source: https://www.glassdoor.com/Salaries/san-francisco-data-analytics-salary-SRCH_IL.0,13_IM759_KO14,28.htm

- Oppportunity cost— When your data engineers are spending their time manually building data tests to account for any and all possible edge cases, those are hours that could have been spent solving customer problems, improving your product, or driving innovation. When you consider that even the most robust testing in the world won't account for about 80 percent of data issues, it pays to consider the opportunity cost of what it means to be building and maintaining these tests instead of working on projects that will actually move the needle for your business.

- Scoping— Before building or buying any data quality solution, you should understand exactly what you are looking to achieve not just tomorrow but in the next 6, 12, or even 18 months. You will need to look into the future and understand:
    - Who will use the tool?
    - What data quality problems do you want to solve?
    - What will be the single source of truth?
    - What about data discovery or lineage?
    - What are your data governance requirements?

"With such high-stakes data in Snowflake, it wasn't an option to not include data observability—getting that out of the box with Monte Carlo allowed us to avoid spending the mental energy on designing something ourselves so that we could spend that energy on building products that help our customers," said Jacob Follis, chief innovation officer, Collaborative Imaging. "It brought us down the path of ensuring data quality faster and prevented us from taking on technical debt."



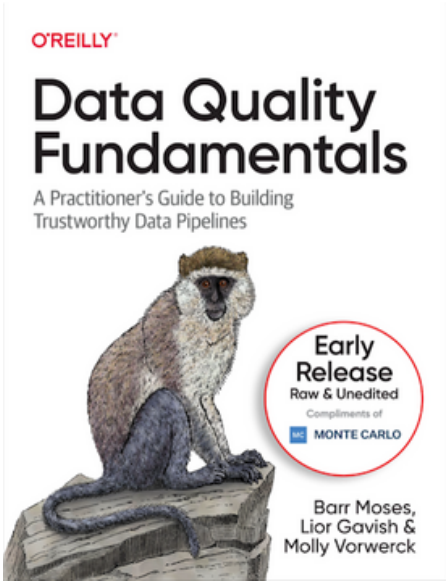
Read Collaborative Imaging's Story

III. How to Build a Data Pipeline Monitor

This chapter will cover how to build data pipeline monitors and is meant for a highly technical reader. The next chapter will cover considerations when buying data pipeline monitoring and data observability solutions. Feel free to skip to the section most relevant to your use case.

Note: the details of your implementation will depend on your choice of data warehouse, data lake, BI tools, preferred languages and frameworks, and so on. This chapter addresses these problems using lightweight tools like SQLite and Jupyter in an example data ecosystem to create a data quality monitor in SQL.

This section has been adapted from O'Reilly Media's Data Quality Fundamentals co-authored by Monte Carlo co-founders Barr Moses, Lior Gavish, and Monte Carlo head of content Molly Vorwerck. The section below was contributed by Ryan Kearns.



While we just review how to build one type of monitor here–in this case data freshness–you can get examples for how to build schema, lineage, distribution monitors and more from the book available for free on our site.

Freshness monitor

Our sample data ecosystem uses mock astronomical data about habitable exoplanets. For the purpose of this exercise, I generated the dataset with Python, modeling anomalies off of real incidents I've come across in production environments. This dataset is entirely free to use, and the utils folder in the repository contains the code that generated the data, if you're interested.

I'm using SQLite 3.32.3, which should make the database accessible from either the command prompt or SQL files with minimal setup. The concepts extend to really any query language, and these implementations can be extended to MySQL, Snowflake, and other database environments with minimal changes.

```
$ sqlite3 EXOPLANETS.db
sqlite> PRAGMA TABLE_INFO(EXOPLANETS);
0 | _id           | TEXT | 0 | | 0
1 | distance      | REAL | 0 | | 0
2 | g             | REAL | 0 | | 0
3 | orbital_period | REAL | 0 | | 0
4 | avg_temp      | REAL | 0 | | 0
5 | date_added    | TEXT | 0 | | 0
```

A database entry in EXOPLANETS contains the following info:

0._id: A UUID corresponding to the planet.
1. distance: Distance from Earth, in lightyears.
2. g: Surface gravity as a multiple of g, the gravitational force constant.
3. orbital_period: Length of a single orbital cycle in days.
4. avg_temp: Average surface temperature in degrees Kelvin.
5. date_added: The date our system discovered the planet and added it automatically to our databases.

Note that one or more of distance, g, orbital_period, and avg_temp may be NULL for a given planet as a result of missing or erroneous data.

```
sqlite> SELECT * FROM EXOPLANETS LIMIT 5;
```

id	distance	g	orbital_period	avg_temp	date_added
c168b188-ef0c-4d6a-8cb2-f473d4154bdb	34.627303634841		476.480044083599		2020-01-01
e7b56e84-41f4-4e62-b078-01b076cea369	110.196919810563	2.525073623590667	839.8378167897		2020-01-01
a27030a0-e4b4-4bd7-8d24-5435ed86b395	26.695795045452	10.276497001606 7	301.018816321399		2020-01-01
54f9cf85-eae9-4f29-b665-855357a14375	54.8883521129783		173.788967912197	328.644125249613	2020-01-01
4d06ec88-f5c8-4d03-91ef-7493a12cd89e	153.264217159834	0.922874568459221	200.712661803056		2020-01-01

Note that this exercise is retroactive — we're looking at historical data. In a production data environment, data observability is real time and applied at each stage of the data life cycle, and thus will involve a slightly different implementation than what is done here.

The first pillar of data observability we monitor for is freshness, which can give us a strong indicator of when critical data assets were last updated. If a report that is regularly updated on the hour suddenly looks very stale, this type of anomaly should give us a strong indication that something is off. First, note the DATE_ADDED column. SQL doesn't store metadata on when individual records are added. So, to visualize freshness in this retroactive setting, we need to track that information ourselves.

Grouping by the DATE_ADDED column can give us insight into how EXOPLANETS updates daily. For example, we can query for the number of new IDs added per day:

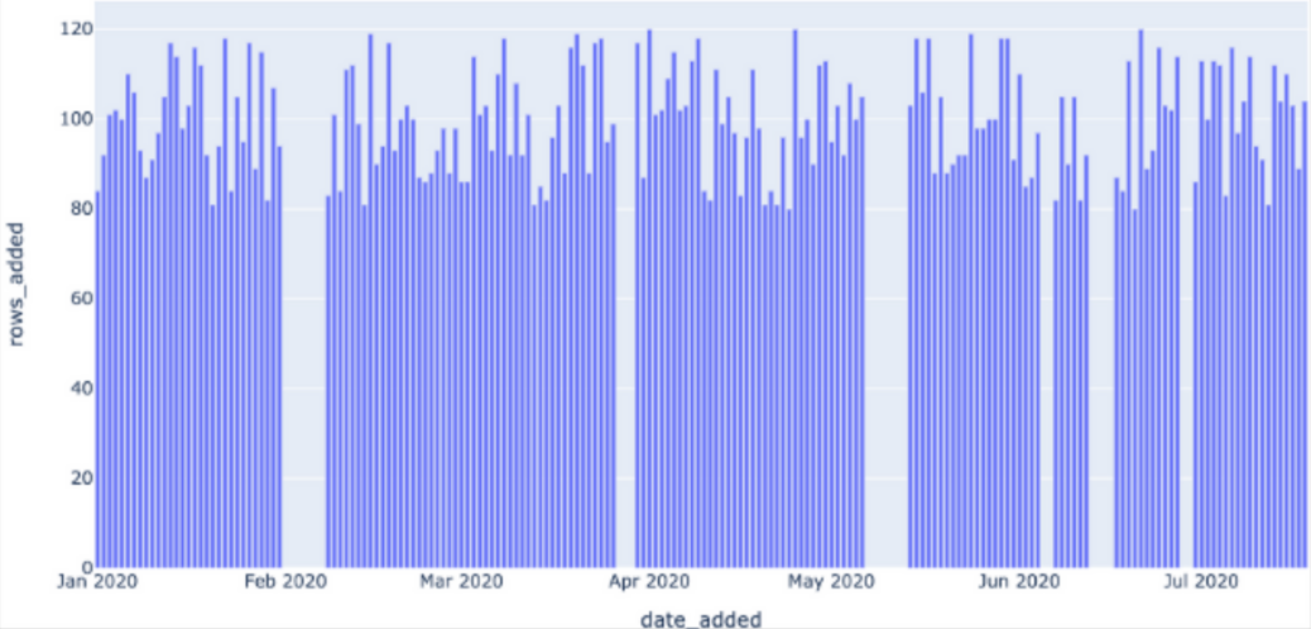
You can run this yourself with $ sqlite3 EXOPLANETS.db < queries/freshness/rows-added.sql in the repository. We get the following data back:

```
SELECT
   DATE_ADDED,
COUNT(*) AS ROWS_ADDED
FROM
   EXOPLANETS
GROUP BY
   DATE_ADDED;
```

date_added	ROWS_ADDED
2020-01-01	84
2020-01-02	92
2020-01-03	101
2020-01-04	102
2020-01-05	100
...	...
2020-07-14	104
2020-07-15	110
2020-07-16	103
2020-07-17	89
2020-07-18	104

Based on this graphical representation of our dataset, it looks like EXOPLANETS consistently updates with around 100 new entries each day, though there are gaps where no data comes in for multiple days.

Recall that with freshness, we want to ask the question “is my data up to date?” — thus, knowing about those gaps in table updates is essential to understanding the reliability of our data.

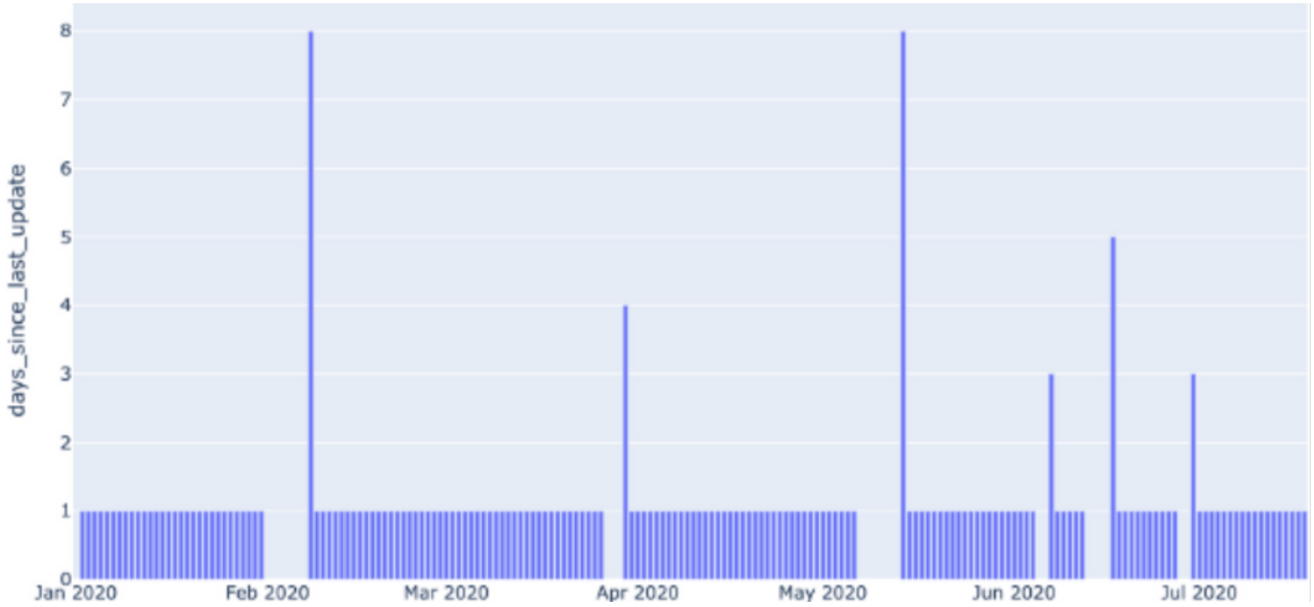


This query operationalizes freshness by introducing a metric for DAYS_SINCE_LAST_UPDATE. (Note: since this tutorial uses SQLite3, the SQL syntax for calculating time differences will be different in MySQL, Snowflake, and other environments).

```
WITH UPDATES AS(
  SELECT
    DATE_ADDED,
    COUNT(*) AS ROWS_ADDED
  FROM
    EXOPLANETS
  GROUP BY
    DATE_ADDED
)


SELECT
 DATE_ADDED,
 JULIANDAY(DATE_ADDED) - JULIANDAY(LAG(DATE_ADDED) OVER(
   ORDER BY DATE_ADDED
```

date_added	DAYS_SINCE_LAST_UPDATE
2020-01-01	0
2020-01-02	1
2020-01-03	1
2020-01-04	1
2020-01-05	1
...	...
2020-07-14	1
2020-07-15	1
2020-07-16	1
2020-07-17	1
2020-07-18	1

The resulting table says “on date X, the most recent data in EXOPLANETS was Ydays old.” This is information not explicitly available from the DATE_ADDED column in the table — but applying data observability gives us the tools to uncover it.



Now, we have the data we need to detect freshness anomalies. All that’s left to do is to set a threshold parameter for Y — how many days old is too many? A parameter turns a query into a detector, since it decides what counts as anomalous (read: worth alerting) and what doesn’t.

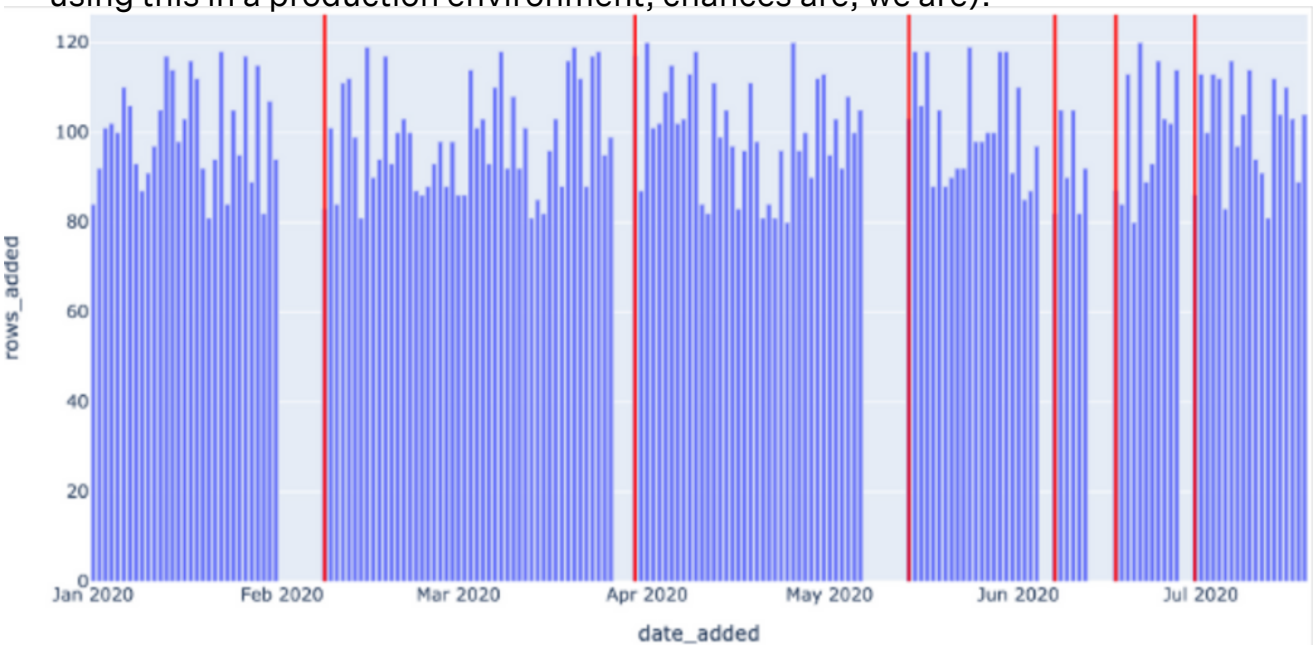
```
WITH UPDATES AS(
  SELECT
    DATE_ADDED,
    COUNT(*) AS ROWS_ADDED
  FROM
    EXOPLANETS
  GROUP BY
    DATE_ADDED
),


NUM_DAYS_UPDATES AS (
  SELECT
    DATE_ADDED,
    JULIANDAY(DATE_ADDED) - JULIANDAY(LAG(DATE_ADDED)
      OVER(
        ORDER BY DATE_ADDED
      )
    ) AS DAYS_SINCE_LAST_UPDATE
  FROM
    UPDATES
)
```

DATE_ADDED	DAYS_SINCE_LAST_UPDATE
2020-02-08	8
2020-03-30	4
2020-05-14	8
2020-06-07	3
2020-06-17	5
2020-06-30	3

Freshness detections!

The data returned to us represents dates where freshness incidents occurred.

On 2020–05–14, the most recent data in the table was 8 days old! Such an outage may represent a breakage in our data pipeline, and would be good to know about if we're using this data for anything worthwhile (and if we're using this in a production environment, chances are, we are).

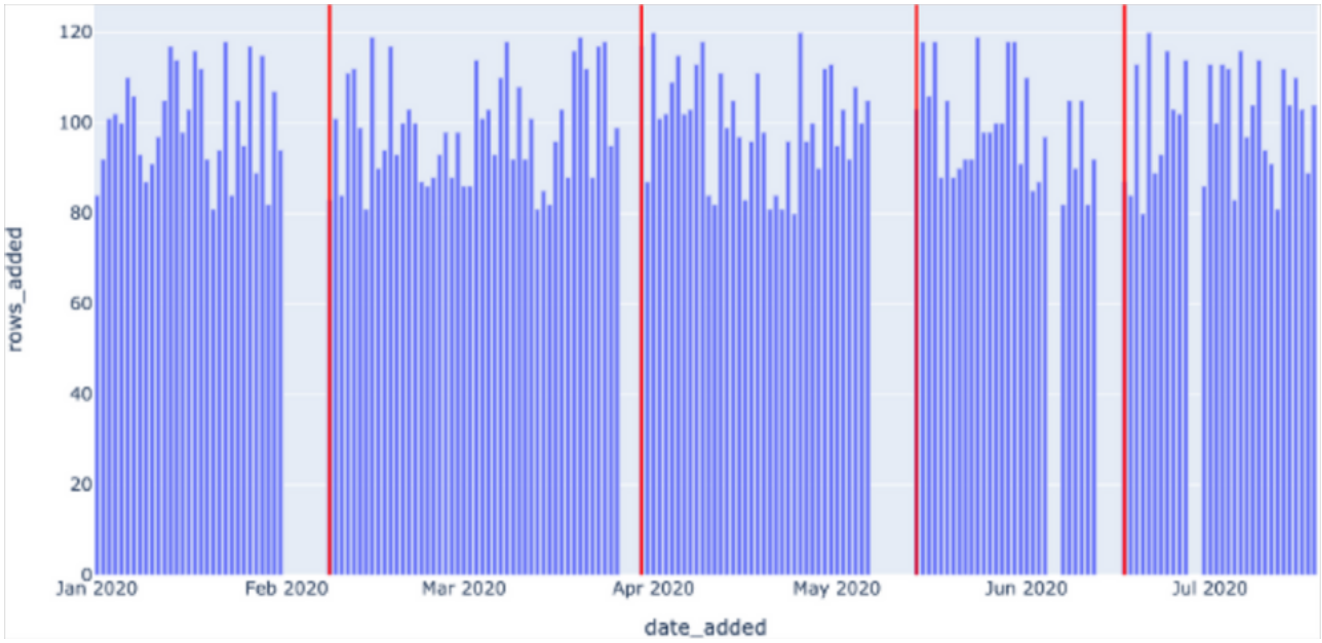


Note in particular the last line of the query: DAYS_SINCE_LAST_UPDATE > 1;.

Here, 1 is a model parameter — there's nothing "correct" about this number, though changing it will impact what dates we consider to be incidents. The smaller the number, the more genuine anomalies we'll catch (high recall), but chances are, several of these "anomalies" will not reflect real outages. The larger the number, the greater the likelihood all anomalies we catch will reflect true anomalies (high precision), but it's possible we may miss some.

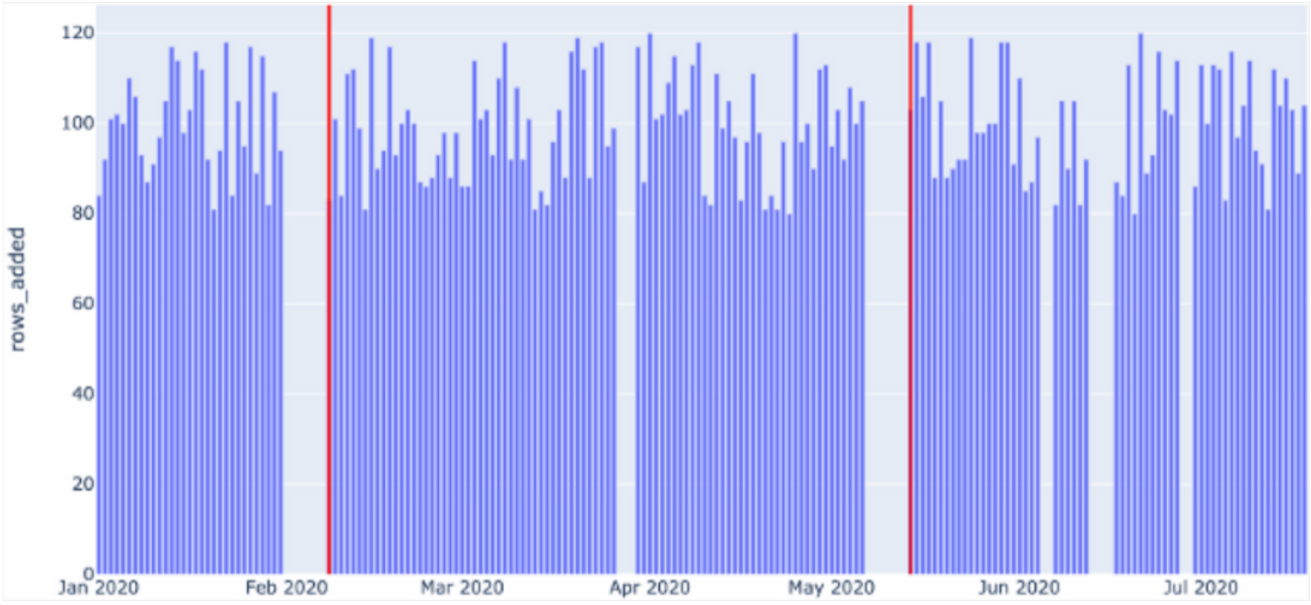
For the purpose of this example, we could change 1 to 7 and thus only catch the two worst outages on 2020–02–08 and 2020–05–14. Any choice here will reflect the particular use case and objectives, and is an important balance to strike that comes up again and again when applying data observability at scale to production environments.

Below, we leverage the same freshness detector, but with DAYS_SINCE_LAST_UPDATE > 3; serving as the threshold. Two of the smaller outages now go undetected.

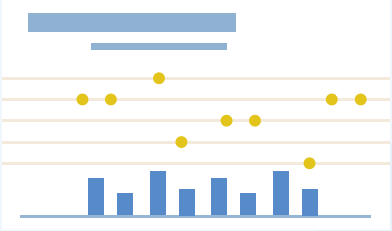


Note the two undetected outages — these must be fewer than 3-day gaps.

Now we visualize the same freshness detector, but with DAYS_SINCE_LAST_UPDATE > 7; now serving as the threshold. All but the two largest outages now go undetected.



Just like planets, optimal model parameters sit in a “Goldilocks Zone” or “sweet spot” between values considered too low and too high.



What's next?

This brief tutorial intends to show that "data observability" is not as mystical as the name suggests, and with a holistic approach to understanding your data health, you can ensure high data trust and reliability at every stage of your pipeline.

In fact, the core principles of data observability are achievable using plain SQL "detectors," provided some key information like record timestamps and historical table metadata are kept. It's also worth noting that key ML-powered parameter tuning is mandatory for end-to-end data observability systems that grow with your production environment.

However, keep in mind our earlier section on build vs. buy and the costs involved with creating and maintaining your own data pipeline monitoring. At the very least, our recommendation is to demo some commercial solutions prior to engaging on your build.

IV. How to Get Started With Data Pipeline Monitoring

Data observability platforms are the newest layer in the modern data stack, helping teams monitor the health of critical data assets and pipelines while building organizational trust in data at scale.

Because it's a nascent category, many questions arise on what the must-have features are or how different solutions should be evaluated.

After working with hundreds of organizations during the pre and post purchase process, we have put together four key areas that drive value and user satisfaction.

Time-to-value

Data teams should get immediate value out of data observability tools. Full stop.

They should be easy and quick to deploy, and fast to start detecting anomalies or other issues with your data, instead of a weeks-long process in manually configuring monitors. Even the most efficient data engineering team doesn't have time for that.

At a fundamental level, a data observability platform should monitor your data's health and alert your team when pipelines break or jobs don't run well before your downstream data consumers notice the issue in their dashboards or queries.

Knowing what issues to monitor for can be overwhelming which is where a ML driven approach is critical. Instead of relying on manual tests to check for data quality issues, your data observability platform should intelligently surface when changes are noticed.

A tool that provides fast time to value will quickly address any questions regarding data freshness, volume, distribution, and schema changes over time, and provided domain data quality dashboards for technical and business users alike. The solution will also reduce time to monitor assets over time and provide a single pane for data governance needs.

"What was really exciting to me when we first got set up was just how we flipped this switch, and suddenly we had all kinds of insights into what was happening in our warehouse," said Dylan Hughes, senior software engineer, Prefect.

Read Prefect's Story

End-to-end pipeline visibility

Your data observability platform should give you unprecedented visibility into your data ecosystem so you can proactively identify, root cause, and resolve data issues from source to orchestration and transformation tooling, to BI layer.

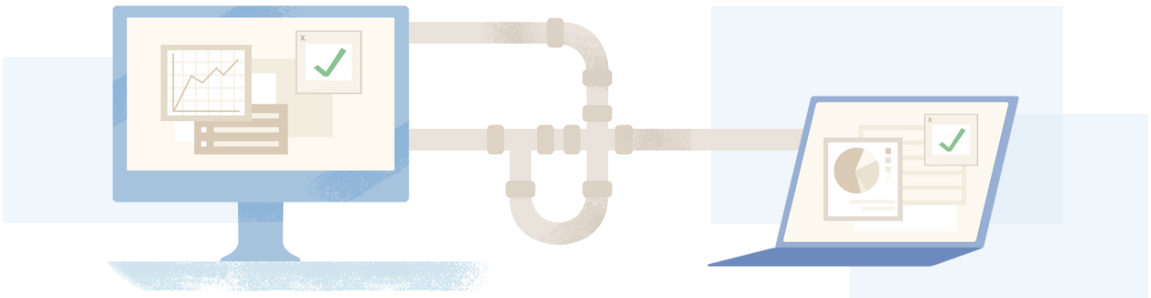
Your most critical asset? Automated field-level lineage.

Limited to no visibility into data pipelines upstream of a data assets makes troubleshooting manual and time consuming.

With data observability, data teams should be able to see the entire data lineage upstream and across warehouse projects of a BI report or materialized tables by simply searching in a centralized UI.

Lineage provides information to end users on dashboard staleness and other incidents due to downstream issues, and offers proactive troubleshooting to help address issues in a timely manner.

A strong data observability solution will integrate with solutions at the ingestion, orchestration, transformation, repository, and BI layers.



"I told my boss, 'I wish there were someplace that I could take a table and see everything it connects to," said Daniel Rimon, head of data engineering at Resident. "I could know, it affects this and this, but not here, and here's the dashboard it goes to ultimately. I need a very big map. There has to be some kind of thing like this!"

Read Resident's Story

Deep, broad, and custom coverage

Data observability solutions should proactively (and automatically) alert you to freshness and volume anomalies, as well as schema changes that are then logged in a central catalog.

Your platform should use metadata such as query logs and schema change logs to automatically alert when changes break your data pipelines. By using this metadata instead of the data itself, your platform provides an additional layer of security and reduces costs by limiting data monitoring on field values requiring direct queries to only your most critical assets (more on that later).

This is table stakes for any data observability platform. Just as software engineers rely on ML-powered monitoring to alert them to any outages or downtime, data teams should be able to rely on monitoring to automatically notify them of any anomalies in data health the minute a table is added to their environment rather than needing to set thresholds, opt it into monitoring, or waiting for an internal consumer or customer to notice something's gone wrong.

However while automation is critical for freshness, volume, and schema-related incidents, your organization knows your data assets best. As a result, your data observability platform should include the flexibility for data engineers, data analysts, and data scientists to set their own rules to monitor data according to business needs or as new pipelines and projects are introduced.

User-defined, machine learning-powered monitors are best deployed using the most well-defined, understood logic to catch anomalies that are frequent or severe. It is typically expressed through a SQL statement.

A strong data observability solution will proactively alert you to field-level anomalies, such as an increase in the number of dimensions, a change in the % of unique values, or whether a value is within historical range, through Field Health and Dimension tracking monitors in a timely manner, with no-code configuration.

Beware— ensure your data monitoring isn't overly narrow! When you only deploy user-defined and machine learning monitors on your most important tables:

- You miss or are delayed in detecting and resolving anomalies more evident upstream.
- Alerting on a key table, oftentimes dozens of processing steps removed from the root cause, will involve the wrong person and give them little context on the source of the issue or how to solve it.
- Without understanding the dependencies in the system, your team will waste time trying to determine where to focus their monitoring attention. Maintaining that as the environment and data consumption patterns change can also be tedious and error prone.

All we really had to do was sign up, add the security implementation to give Monte Carlo the access that it needed, and we were able to start getting alerted on issues. Monte Carlo gave us that right out of the box."

Read Clearcover's Story

Accelerated root cause analysis across all three levels

Data observability solutions should accelerate resolution across the three levels where anomalies are introduced:

- System root causes: System or operational issues are found when there is an error introduced by the system or tools that customers apply to the data during the extraction, loading and transformation processes.

An example of this could be an Airflow check that took too long to run causing a data freshness anomaly. Another example could be a job that is relying on accessing a particular schema in Snowflake, but it doesn't have the right permissions to access that schema. This is why it is critical for your data observability solution to integrate with multiple solutions across the modern data stack and surface error logs to consolidate this troubleshooting behind a single pane of glass.

- Code: The second type of data incident root causes are code related. An example would be is there anything wrong with your SQL or engineering code? An improper JOIN statement resulting in unwanted or unfiltered rows perhaps? Or is it a dbt model that accidentally added a very restrictive WHERE clause that resulted in a reducing number of rows of output data triggering a volume anomaly?

  If you can find a query or dbt model was modified around the approximate time an anomaly was introduced, it's a promising sign you've found your root cause. This process can be expedited with data monitoring and log analysis across your stack.

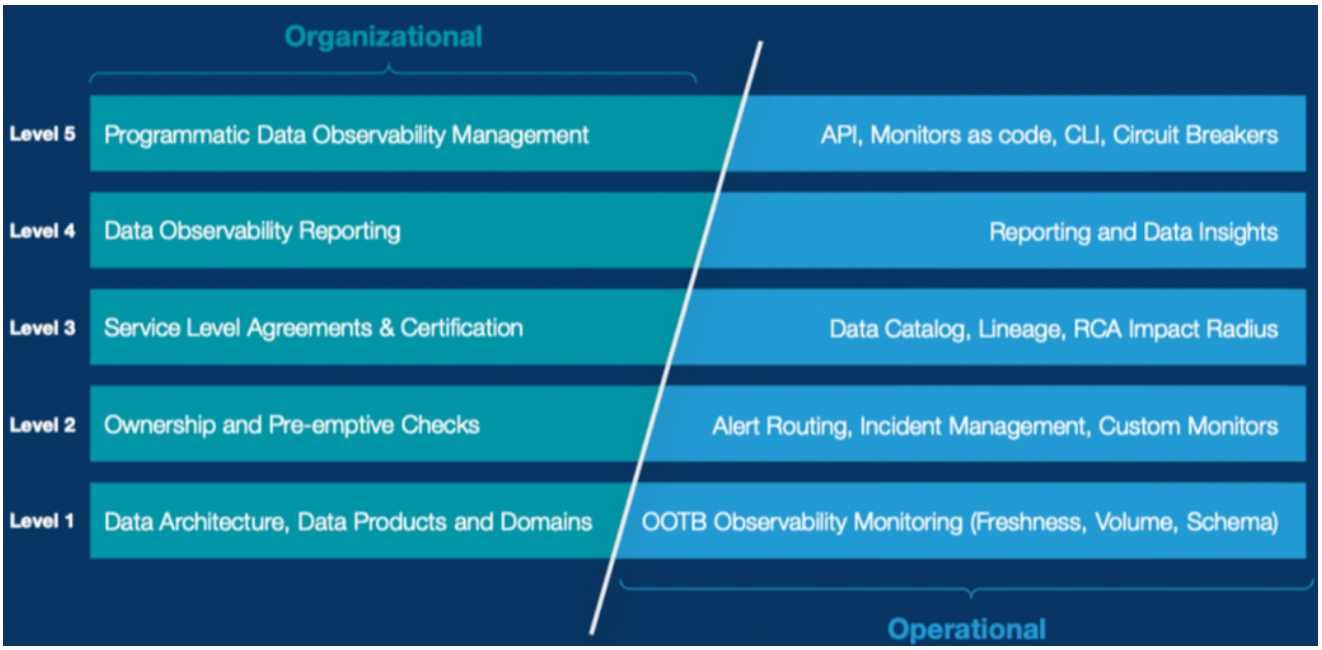
- Data: System and code issues are also very typical in software engineering, but in the wonderful world of data engineering, there can also be issues that arise in the data itself making it a more dynamic variable.

  For example, it could be a consumer application where the customer input is just wacky. Let's say you are an online pet retailer and someone enters their dog weighs 500 pounds instead of just 50. A strong data observability solution needs to be able to monitor for these types of field or metric anomalies and have features like anomalous row distribution that can help pinpoint where bad data is entering the system.

"With Monte Carlo's broad coverage and automated lineage...our team can identify, understand downstream impacts, prioritize, and resolve data issues at a much faster rate," said Ashley VanName, general manager of data engineering, JetBlue.

Read Jet Blue's Story

V. Operationalizing Data Pipeline Monitoring With Data Observability



Let's say at this point you are either in the final stages of building or buying your data monitoring solution. It's time to start thinking about how you are going to leverage your new capabilities to scale data quality across your organization.

We've found there are five levels of operational maturity that are best tackled in sequential order. Let's take a look at each.

Level zero

Level zero, if you haven't already taken this step when building your business case, is to align on the overall organizational goals and KPIs of this initiative. How are you going to measure the success over time or show the value it has provided to the business?

There are a number of general data quality metrics–some like data downtime or time spent fixing pipelines that we have introduced previously in this ebook–that can be used for this purpose. However, the best metric is one that measures what your boss cares about and is aligned with business goals in the long-term. This way you retain executive buy-in and support for your initiative.

At this stage you should also set up regular check-ins with the vendor or internal team responsible for the roadmap and development of your monitoring solution.

Level 1

The first level of operational maturity is to make sure you have basic coverage (freshness, volume, schema) in place across your data environment. For the vast majority of organizations, you will want to roll this out across every data product, domain, and department rather than pilot and scale.

This will accelerate your time to value and help you establish critical touch points with different teams if you haven't done so already. Another reason for a wide roll out is that, even with the most decentralized organizations, data is interdependent. If you install fire depressant systems in the living room while you have a fire in the kitchen, it doesn't do you much good. Also, wide-scale data monitoring and/or data observability will give you a complete picture of your data environment and the overall health. Having the 30,000 foot view is helpful as you enter the second level of operational maturity.

"We started building these relationships where I know who's the team driving the data set," said Lior Solomon, former VP of engineering and data at Vimeo.. "I can set up these Slack channels where the alerts go and make sure the stakeholders are also on that channel and the publishers are on that channel and we have a whole kumbaya to understand if a problem should be investigated."

Read Vimeo's Story

Level 2

At this stage in the game, we want to start optimizing our incident triage and resolution response. This involves setting up clear lines of ownership. There should be team owners for data quality as well as overall data asset owners at the data product and even data pipeline level. Breaking your environment into domains, if you haven't already, can help create additional accountability and transparency for the overall data health levels maintained by different groups.

Having clear ownership also enables fine tuning your alert settings, making sure they are sent to the right communication channels of the responsible team at the right level of escalation.

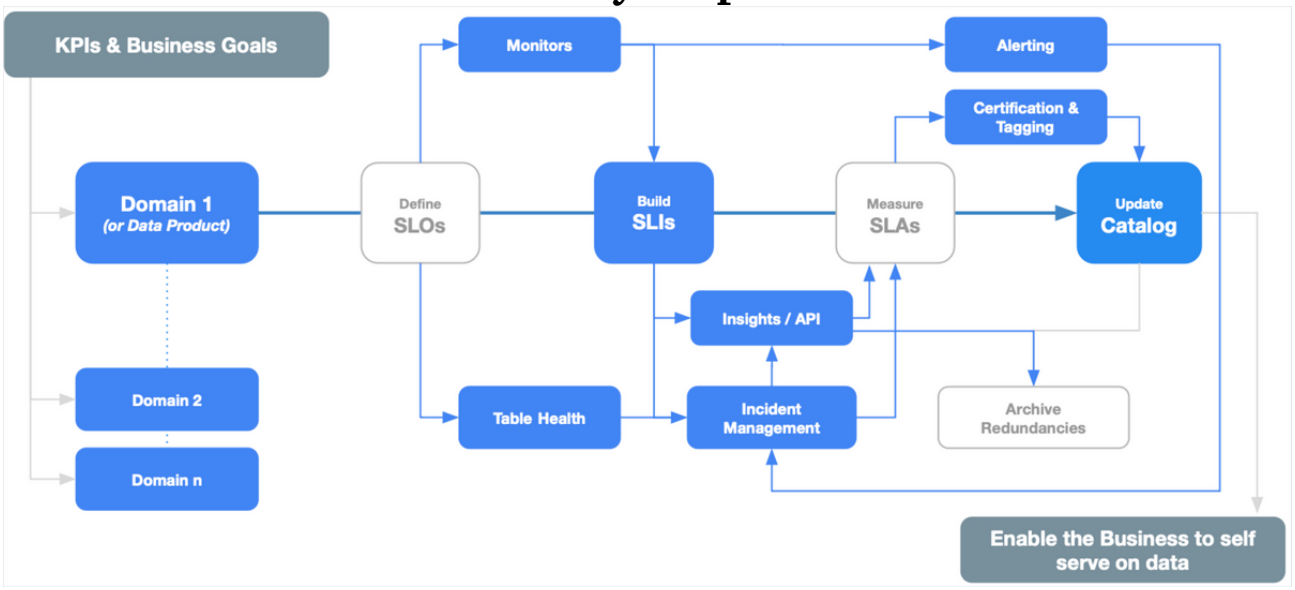
Consider...	Otherwise...	Think through...
Is the monitoring valuable?	You get noise and fatigue	• Is it on meaningful data? • Does it adapt as your business scales? • Can you add the appropriate business context?
Are alerts delivered effectively?	Alerts go unnoticed	• Does it reach the right people? • Is it easily understood? • Can it convey a major outage from a small blip?
Are individuals empowered to act?	You prolong the downtime	• Is there culture/accountability to act on alerts? • Are the tools available to triage and debug issues? • Does leadership have visibility to enforce accountability?
During outages, are the right people informed?	Consumers still 'discover' the issue and you erode trust	• Can you understand who in the business is impacted? • Can you communicate what happened? • Can you communicate when it's resolved?

Level 3

While level two is about fine tuning your ownership and alerting, this level of operational maturity is focused on layering more sophisticated, custom alerts. These can be centered around specific data SLAs–for example if data needs to be fresh at 8:00 am every weekday for a meticulous executive.

Custom monitors and SLAs can also be built around different data reliability tiers to help set expectations to certify "gold" datasets with high reliability, to an ad-hoc data pull for a limited use case that may not be supported as robustly.

How does data observability help with data certification?



Level 4

At this point, we have driven significant value to the business and noticeably improved data quality at our organization. The first three levels have helped dramatically reduce our time-to-detection and time-to-resolution, but there is a third variable in the data downtime formula: number of data incidents.

One of the main goals of level four is to start operationalizing your preventive maintenance. In other words, preventing data incidents before a pipeline breaks. That can be done by focusing on data health insights like unused tables or deteriorating queries. Analyzing and reporting the data quality levels or SLA adherence across domains can also help data leaders determine where to allocate resources.

"Monte Carlo's lineage highlights upstream and downstream dependencies in our data ecosystem, including Salesforce, to give us a better understanding of our data health," said Yoav Kamin, then the director of business performance at Yotpo. "Instead of being reactive and fixing the dashboard after it breaks, Monte Carlo provides the visibility that we need to be proactive."

Read Yotpo's Story

Level 5

The final level is focused on programmatic data quality management. The most sophisticated organizations manage a large portion of their data monitoring through code (monitors as code) as part of the CI/CD process. They also use circuit breakers, which stop pipelines from landing bad data into the warehouse (or any data once they are tripped so proceed with caution!).

Final thoughts

We covered a lot of ground. Some of the key takeaways include:

- Make sure you are monitoring both the data pipeline and the data flowing through it.

- You can build a business case for data monitoring by understanding the amount of time your team spends fixing pipelines and the impact it has on the business.

- You can build or buy data monitoring–the choice is yours–but if you decide to buy a solution be sure to evaluate its end-to-end visibility, monitoring scope, and incident resolution capabilities.

- Operationalize data monitoring by starting with broad coverage and mature your alerting, ownership, preventive maintenance, and programmatic operations over time.

Perhaps the most important takeaway from this guide is that data pipelines will break and data will turn bad. Whatever your next step entails, it's important to take it sooner rather than later. You won't be sorry.

VI. Additional Resources

Don’t let this be the ending point of your data quality journey! Check out more helpful resources including:

- Data Downtime Blog: Get fresh tips, how-tos, and expert advice on all things data.
- O’Reilly Data Quality Framework: The first several chapters of this practitioner's guide to building more trustworthy pipelines are free to access.
- Data Observability Product Tour: Check out this video tour showing just how a data observability platform works.
- Request A Demo: Talk to our team to get a more accurate assessment of your data downtime, its costs, and what level of value you can expect from Monte Carlo.