Materialize

WHITEPAPER

Building an Operational Data Warehouse for Real-Time Analytics

Are you pushing your data warehouse to its limits supporting near-real-time scenarios? There's a better way.

Have you ever tried driving real-time updates to customers from your data warehouse? Or optimized a conversion funnel using real-time data? If so, you've probably suffered the pain of pushing your data warehouse past its limits.

Analytical data warehouses are now ubiquitous in every company's data workflow. But the data warehouse was designed to provide historical insights. Even the most modern data warehouses struggle to deliver data quickly and efficiently for near-real-time use cases.

Fortunately, there's an alternative. This white paper gives an overview of the operational data warehouse, how it works, and why it delivers better cost/performance for near-real-time use cases than an analytical data warehouse solution.

How analytical and operational data warehouses differ

The data warehouse exists to provide historical insights across many disparate sources of data. You have data in your OLTP databases - your sources of truth. But you also have SaaS products, events, logging exhaust, and a multitude of other data sources.

All of this data is relevant, valuable, and actionable. But none of these sources know how to play nice with each other. They all feed into your warehouse because that's the only place you can easily bring all this data together.

The data warehouse combines data you couldn't jointly query otherwise. However, the batch-oriented operation of traditional data warehouses ensures significant tradeoffs in latency, freshness, and accuracy.

Analytical vs. operational workloads

These trade-offs don't matter for a large class of work. These are our **analytical workloads**, where we grind through reams of historical data and efficiently produce periodic reports. Traditional analytical data warehouses handle these workloads well.

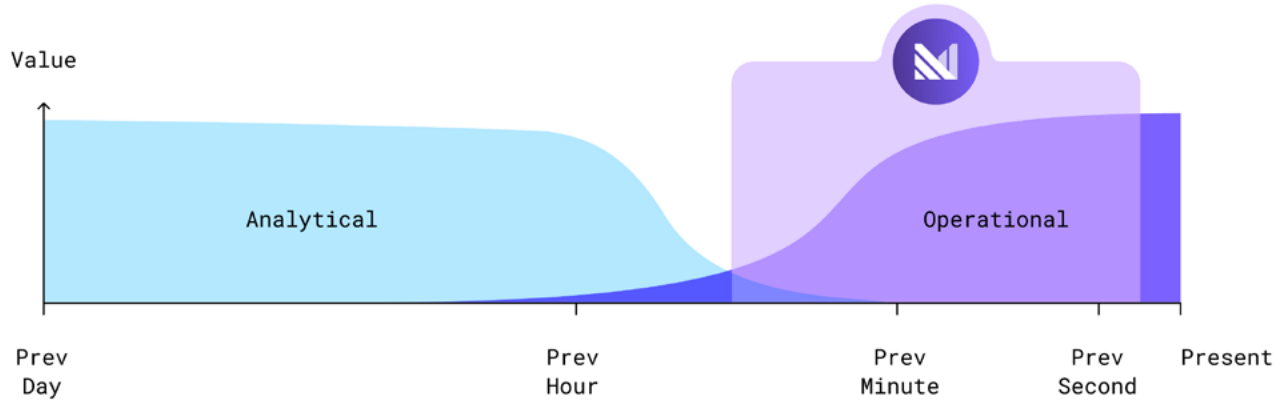
Analytical workloads are different from **operational workloads**, which facilitate the day-to-day operation of your business.

To simplify things, most operational work can be generalized as **automated interventions in the business**. For instance, a bank may process data as it is created to find fraud. If an operational system goes down for the day, it usually means you're forced to offer degraded service or even stop work completely.

Analytical workloads	Operational workloads
Business intelligence exploration	Internal/external alerts & notifications
Ad-hoc exploratory analysis of metrics	Customer segmentation
KPI dashboards	Dynamic pricing & recommendations
Prototyping & exploring hypothetical scenarios	Business automation & workflows
Executive dashboards and reporting	Online feature serving for ML & AI

You can think of the difference between analytical and operational workloads in terms of time spans. Analytical workloads are any workloads that work off of historical data that's anywhere from a day to years old. By contrast, operational workloads work off data from the past month or two, including data that may have just arrived in the past few milliseconds.

Different time spans are important for analytical vs operational work



The analytical data warehouse lacks several features that are critical for operational work:

- It's not always up to date, because data is ingested in batches over time.
- It's not always immediately responsive, as every individual query requires computation from scratch. Even data warehouses that utilize indexes or materialized views must be updated periodically.
- Its content is not always consistent, and must be manually refreshed by different tools or teams.

Fundamentally, the analytical data warehouse wasn't designed for operational work. This hasn't stopped people from using their analytical data warehouse for real-time data. But, the closer to real-time the use case becomes, the more challenging and cost-prohibitive batch updates, view maintenance, and indexing become.

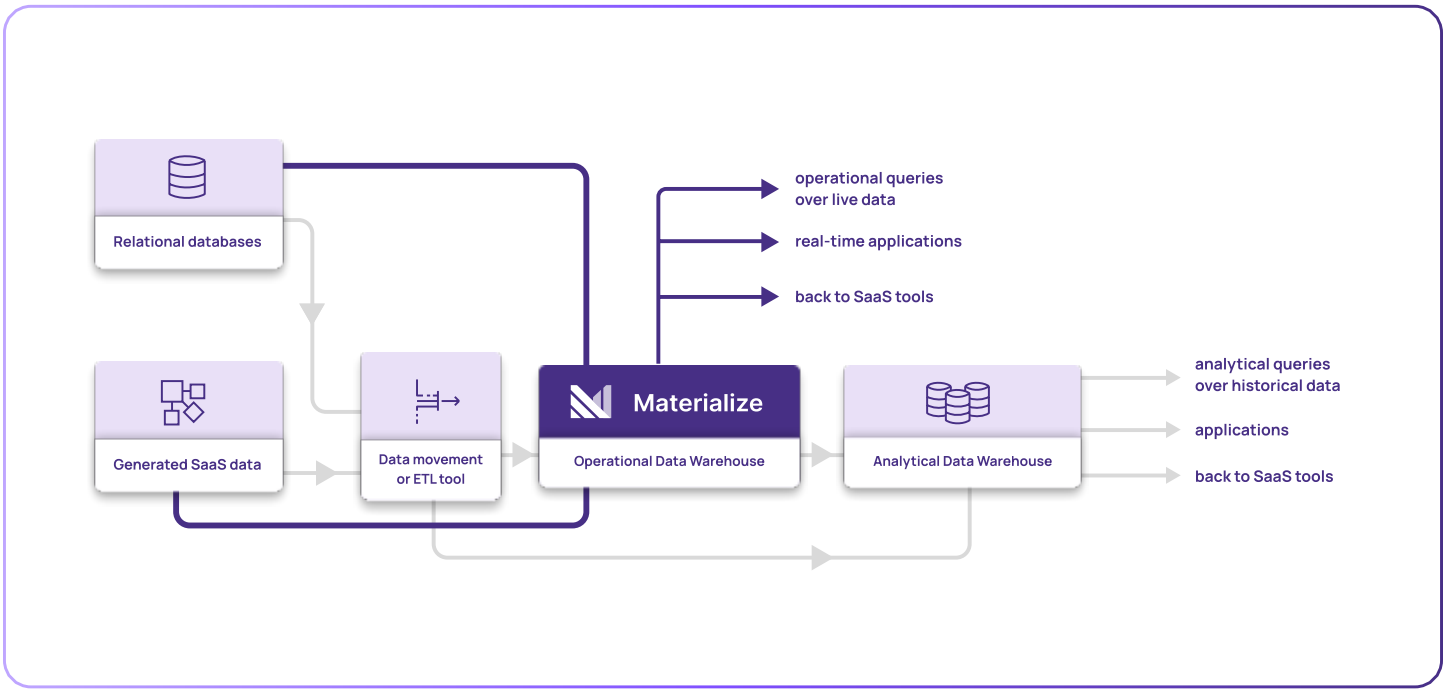
The operational data warehouse

What if you had a data warehouse built for operational workloads? One where:

- Data is ingested continually, and is immediately available to query
- Analytical views are continuously updated and always consistent
- Updated results are seamlessly communicated to downstream systems that can react immediately

In other words, what if you could run operational workloads efficiently out of a data warehouse?

Architecturally, an **operational data warehouse** sits just upstream of your conventional analytical data warehouse. It receives data as changes happen and can transform, normalize, and enrich data as it lands.



An operational data warehouse can immediately update maintained views & indexes while informing downstream dependencies. It can then promptly respond to ad-hoc queries against up-to-date data.

Of course, you'll need your analytical data warehouse for historical queries and traditional reporting. The operational data warehouse can help here too, replicating your normalized and enriched data to an analytical data warehouse for longer-term storage and analysis. When appropriate, it can retire old data from the operational store.

Use cases for an operational data warehouse

If you're currently leveraging your analytical data warehouse for any of the following use cases, chances are you could achieve superior performance with lower cost and effort by shifting to an operational data warehouse.

Real-time analytics

An operational data warehouse can receive changes in near-real-time and update the corresponding views and indexes immediately. This enables supporting scenarios that depend on real-time processing, such as status updates from field service personnel or IoT devices, or data received from customer-facing applications.

Example: Real-time delivery notification updates

Automation and alerting

Because data is always fresh in an operational data warehouse, you can more easily perform automation and alerting tasks by transforming incoming data via SQL and reacting to changes via event-driven triggers. This opens up use cases such as risk scoring on transactions, fraud and anomaly detection, and monitoring of devices.

Example: Reducing shopping cart abandonment

Segmentation and personalization

Using real-time data and user behavior, you can support on-the-fly customization of user experiences, including product recommendations, dynamic pricing, and in-app promotions.

Example: Dynamic decision-making with real-time customer data models

Hallmarks of an operational data warehouse

To support operational data workloads, an operational data warehouse must have four fundamental attributes: trust, scale, ease of use, and cost-effectiveness.

Because streaming technologies are often complicated and built-for-purpose, a good operational data warehouse needs to be built to support and deliver data from streams out of the box. This means you can benefit from streamed data without creating bespoke tools to utilize it. Streaming can become an implementation detail rather than a product category.

Trust

Trust consist of interactivity, freshness, and consistency. To exhibit trust, an operational data warehouse must be:

- Responsive, minimizing the time between an operational ask and its completion
- Up to date, immediately reflecting updates to your data as soon as they happen
- Consistent, presenting answers and taking actions that always check out

These properties combine to make an operational data warehouse a trusted surrogate data operator.

Ease of use

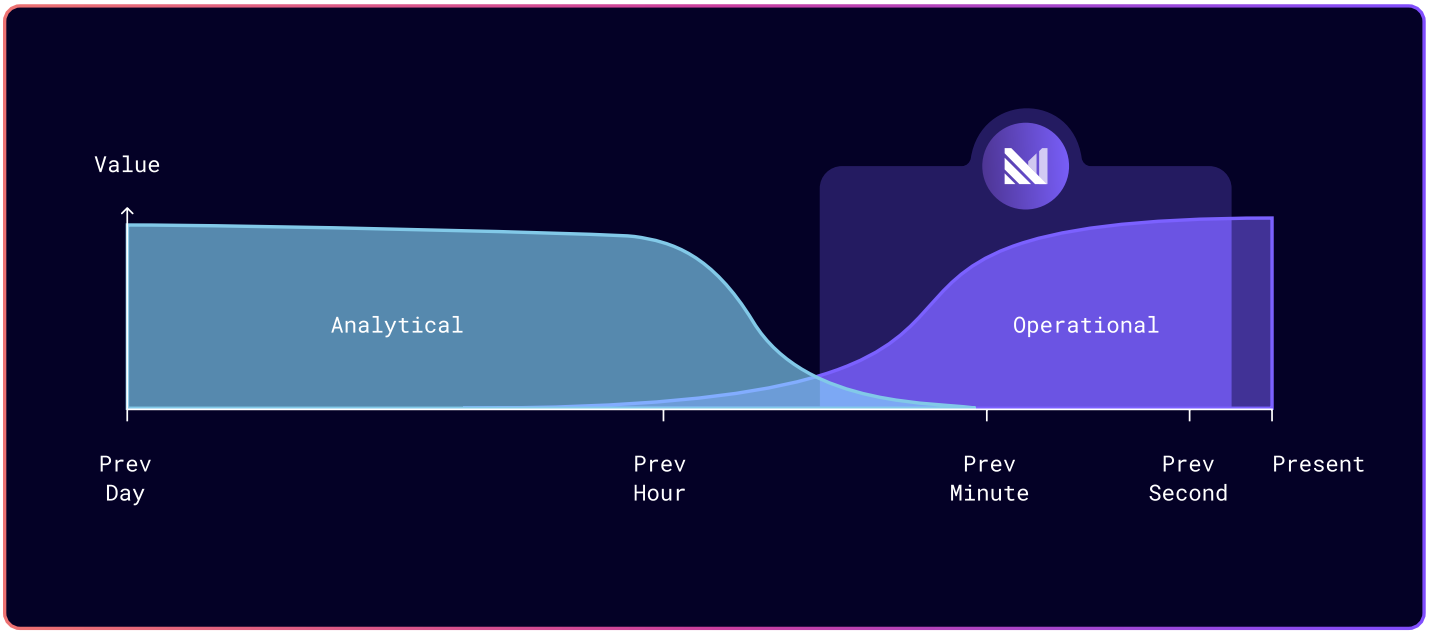
An operational data warehouse should be simple to understand and use. Data arrives and is presented as continually changing tables, over which you use SQL to frame views describing your business logic. You then build indexes that provide ready access to fresh results on top of that.

An operational data warehouse works hard in the background to keep these views current. You experience this only through surprisingly prompt and fast access to indexed view contents. Interactive query results come back in strict serialized order as if all were executed in order one at a time. That avoids the otherwise complex application logic you would need to avoid tripping over inconsistent results.

Cost-effectiveness

One of the reasons cloud analytical data warehouses took off is that they significantly drove the costs down for pay-per-query analytics. They did this by optimizing query execution for analytical workloads and utilizing the scalability of the cloud to fit compute resources to workloads.

However, the flip side is that operational workloads on analytical data warehouses don't perform as well. The more transforms required to update incoming data, the worse that standard data warehouse engines tend to perform.



An operational data warehouse addresses this by decoupling cost and freshness. Analytic data warehouses rely on the user to decide the frequency at which a batch transformation query should be recomputed and cached, with more frequent updates coming at the cost of increased complexity and compute usage.

By contrast, an operational data warehouse takes the same SQL and parses it into a dataflow that incrementally maintains the results as the input data changes - bit by bit, rather than everything at once.

Why teams remain stuck on analytics data warehouses

If analytical data warehouses aren't the best way to run operational workloads, why do we use them for that purpose?

We asked our own customers. Here's what we heard back.

The data warehouse is often the first place data can be joined

Because operational source data comes from multiple systems, the analytical value is realized by joining those disparate data sources. I.e., when we see this signal and this other signal, we know we need to take this action. If the two signals arrive from a Software as a Service (SaaS) application and your transactional database, joining the two sources in application logic can get complicated.

In contrast, a single data engineer can set up the loading and integration of data once (sometimes via a few clicks in Fivetran). After that, other teams rarely have to come back with change requests to the pipelines. They just work autonomously in the warehouse, in SQL.

It's appealing to stretch that model to cover operational work.

The SQL that analysts write lives after them

The warehouse is where data analysts and analytics engineers prototype their SQL code. Many operational use cases start with a hypothesis, which we then validate with data. The correct place to do that is on your historical data in your cloud data warehouse.

So data teams find themselves with a fully prototyped use case wondering: how do I get the data out of the warehouse and into my operational tools? The default response is to keep that logic where it is, for simplicity rather than operational effectiveness.

The data warehouse centralizes complex business logic

Keep in mind that this isn't a "SQL vs code" decision. It's often a "SQL vs opaque point and click integrations" or "SQL vs microservices without clear owners" decision. Operational workloads are often hidden in glue code, API configuration, and scripts whose creators have long since left the company.

SQL - especially SQL tracked in git repos and organized in dbt projects - is the most accessible alternative. Since SQL is the language of the data warehouse, business logic can reside there with reduced effort.

It unlocks Software Development Lifecycle (SDLC) best practices

Dev/stage/prod workflows, automated tests, change review via pull requests, CI/CD, centralized logging - all these things are becoming central to the way modern data teams manage a growing scope of responsibility. All of these tasks are simplified in a modern data warehouse.

The analytical data warehouse can serve many needs, but it falters with operational workloads

Reverse ETL tools like Census and Hightouch are evidence that others can succeed running some amount of operational work on an analytical data warehouse. But, over time and with scale, problems develop.

Here's why people try to run operational data in the analytical data warehouse - and how it turns around to bite them.

The data size frog is boiled slowly

Companies leverage "modern data stack" tooling to tackle historical analytics workloads. Since warehouses have lowered the entry-level cost to make themselves viable even for smaller businesses, many are starting this journey earlier and earlier.

Operational workloads can be tempting to run in an analytical data warehouse early on. This is due to the small scale of data involved and the temptation to use existing infrastructure. However, data freshness becomes a problem as datasets grow and the ETL pipeline stretches from minutes to hours.

It's possible to throw money at the problem

Initially, companies can pull (expensive) levers in the warehouse to keep up with operational requirements. They can load data and run their dbt pipelines more frequently, upgrade the resources dedicated to processing, and generally spend more to alleviate freshness issues.

We spoke to a company that prototyped fraud detection logic in their warehouse. Initially, this was workable. They loaded data every 30 minutes. The query took 5 minutes. However, as they grew, the data compounded until the query took more than 30 minutes to complete. Eventually, they were running compute 24 hours a day just to deliver stale fraud-detection data at hourly intervals.

It's possible to throw (engineering) time at the problem

As a last resort, there are upfront pipeline optimizations that can be done on analytical data warehouses. But they gain performance through ever-growing complexity.

dbt has a useful solution for lowering the amount of data you work over: incremental models that let you specify logic to merge only the changed rows. Unfortunately, this requires rewriting your SQL, handling new concepts like late-arriving data, and essentially defining an entire lambda architecture in SQL, with all its associated pitfalls.

Why analytical data warehouses hit limits with operational workloads

Ultimately, serving operational workloads out of a data warehouse is a dead end. There are a few reasons for that.

A batch/orchestrated query model

Somewhere deep in the bowels of a datacenter, servers are repeatedly pulling your entire universe of business data out of object storage, running a massive computation on it, and caching the result. The servers do the same work every time, even when only a few rows of input and output data change.

You can optimize this through the complex work of writing incremental models. However, updating operational outputs when the inputs change is a delicate exercise of chaining together a waterfall of loads, transforms, and reverse ETL syncs.

A fragile serving layer

The first thing every tool querying a CDW does is cache the results. This is because the query interface is just not designed for operational use cases.

There are hard, low limits on query concurrency. Point lookups (SELECT * FROM my_cached_ table WHERE user_id=123;) are costly and not performant when queried directly from the CDW. So, Redis it is. (And then you have to monitor and worry about cache invalidation, which usually adds a surprising amount of staleness.)

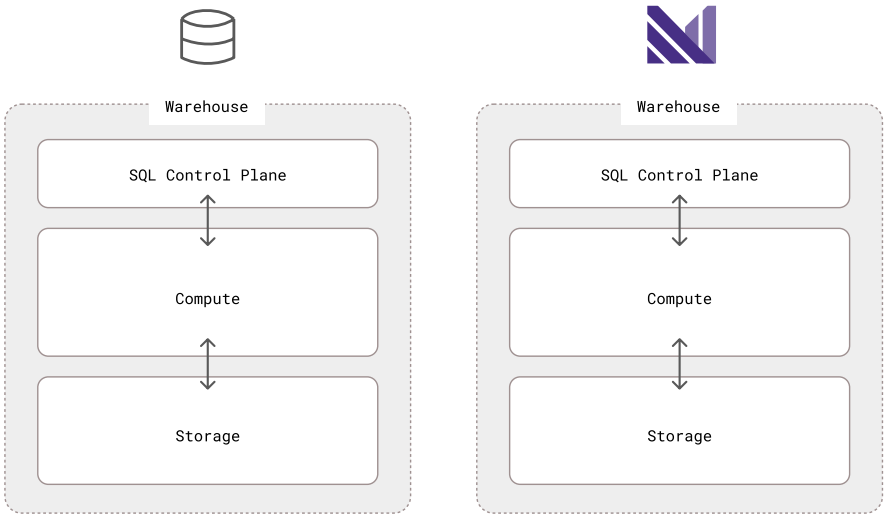
Loaders optimized for infrequent updates

The problem also works its way into upstream tools, services, and even APIs two degrees from the analytical warehouse. Every loading service is designed to build up a batch of updates and merge it as infrequently as possible.

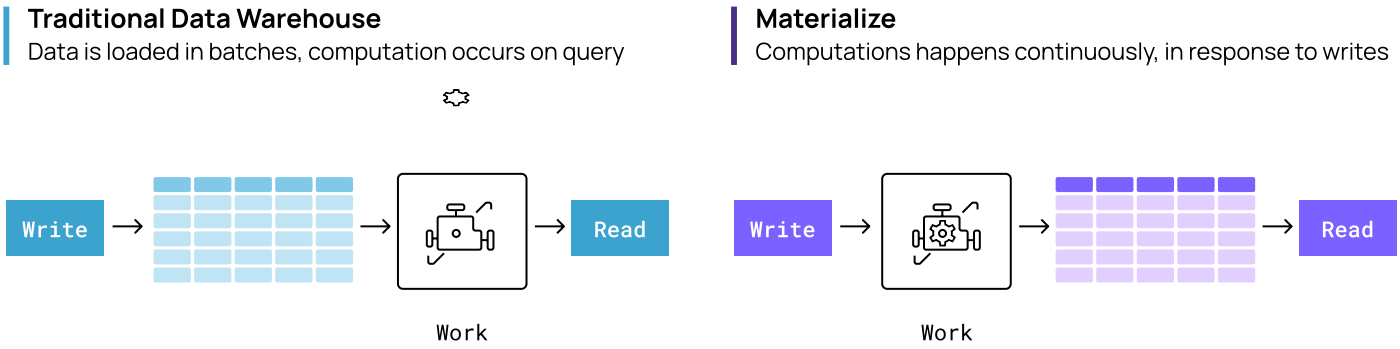
Materialize: The real-time operational data warehouse

We designed Materialize to solve these issues for operational workloads.

Materialize takes the same strengths of analytics data warehouses: (1) ELT of streaming data with cheap scalable object storage, (2) on-demand scaling with compute isolation, (3) compatibility with SDLC workflows. As with data warehouses, you manage these all via the same SQL control layer.



But Materialize changes an analytical data warehouse's constraints by swapping in an engine that delivers real materialized views: continuously updated SQL views created via incremental computation.



Materialize also adds a serving layer that embraces high-frequency reads and proactively pushes downstream systems updates. With these architectural changes, moving certain operational workloads from the warehouse to Materialize can drive major improvements in:

- **Freshness** - data freshness measured in seconds means use cases that were non-starters on data warehouses are now possible.
- **Business Value** - increasing data acts as a value multiplier on many use cases.
- **Cost Efficiency** - due to the incremental model's increased efficiency.

You can achieve similar gains, in theory, by switching to a stream processor. But that carries a steep engineering cost that effectively involves developing your own streaming database.

When to use Materialize: Business use cases

As we've discussed, not every workload currently on your analytical data warehouse needs to move to an operational data warehouse like Materialize. However, for those workloads that would benefit, moving to Materialize will deliver increased performance with less effort at a lower price point.

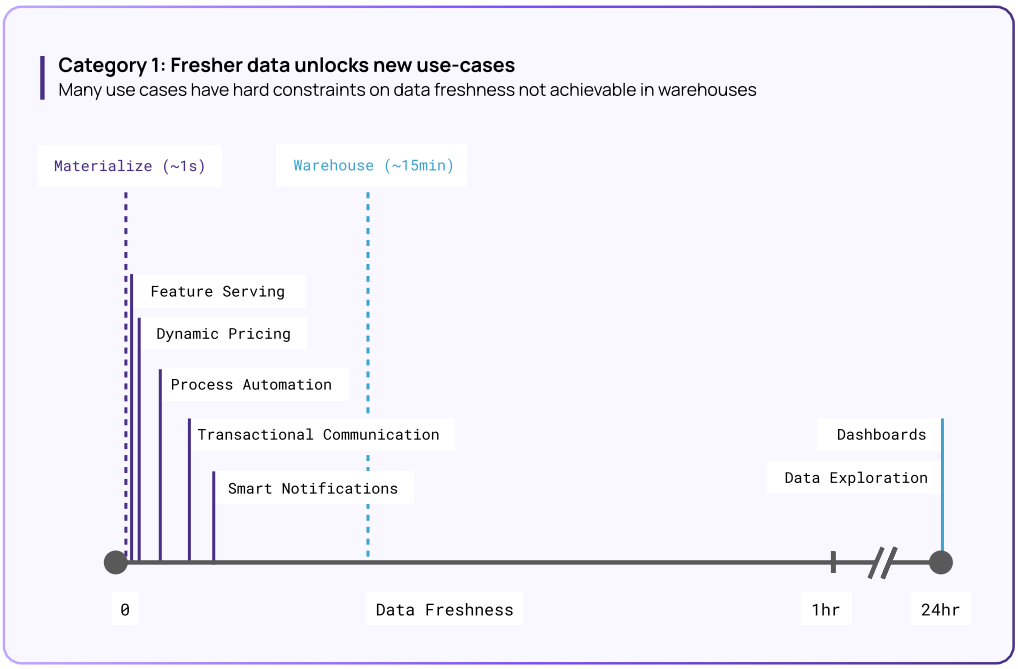
To evaluate whether a warehouse workload is well-suited to Materialize, start with the underlying business requirements: what service or capability does it need to deliver?

Your use case is a good candidate for Materialize if:

- **It benefits from fresh data**

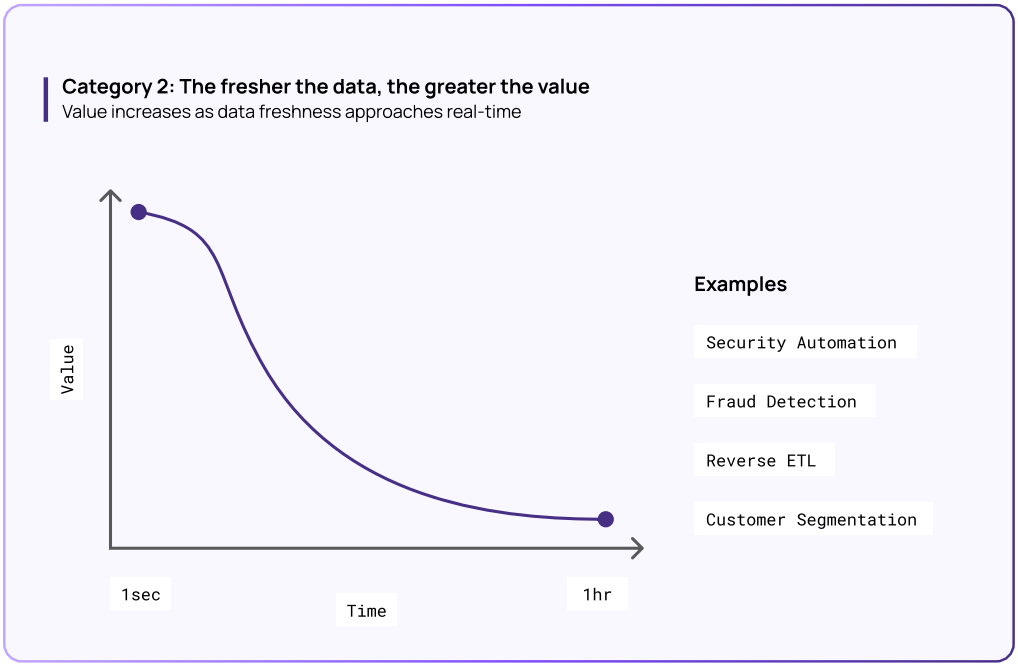
  Fresher data means less time between when the data first originates and when it is incorporated in the results of SQL transformations in the warehouse.

A need for fresher data can manifest in hard latency budgets or thresholds, above which the use case is not feasible:



Example: Transactional emails must be sent within a minute of customer action. Here, a 30-minute delay would cause confusion and frustration.

Or data freshness can act as a value multiplier, where the value of data goes up as lag decreases:



Example: Fraud detection that operates on data 30 minutes out-of-date is possible, but not as valuable as fraud detection that operates on data accurate up to 1 second ago.

It's well-served by a consistent, push-based system

Because Materialize uses dataflows to update results for each individual change to source data, it's the only warehouse that can efficiently push updates out (via SINKs). This means it can be wired up to take action on behalf of the user exactly when it needs to.

Are you polling a view in your warehouse or running an evaluation at a set interval? If so, you might be better served switching to a push-based model, like this:

1. Stream data in continuously via SOURCEs.
2. Define your "alert" or "trigger" criteria as a SQL view.
3. Materialize runs the source data (and all future changes to it) through dataflows to incrementally update your view results.
4. As the results of your view change, Materialize writes the change out immediately to a downstream system.

Today, Materialize writes changes to Kafka. We are also evaluating support for writing changes to other destinations like Datadog and general HTTP endpoints, opening up an even broader and simpler set of push use cases.

When to use Materialize: technical requirements

Workloads you meet should also fulfill one or more of the following technical requirements.

Warehouse compute is always on, running a predictable workload

Predictable workloads are orchestrated or scheduled work like loading data and running SQL transformations. This is work often handled by dbt run or Airflow versus the kind initiated by end-users.

Constantly running an analytical data warehouse for scheduled jobs is like paying an Uber driver for eight hours a day instead of renting a car. Often, this happens when you're running a batch load and transformation workload hourly or more in a traditional data warehouse.

Why move? Presumably, you're running something often because there is value in fresher data. Unless you're using very advanced dbt incremental models. You're probably doing a significant amount of inefficient recomputation of data that hasn't changed since you last ran the query.

Materialize can deliver efficiency and cost savings by maintaining the same query incrementally, as it only needs to compute the data that changes.

Running into limits or complexity with dbt incremental materializations

Related to the point above, one optimization path that heavy warehouse users take is to convert their table materializations to incremental materializations.

At first glance, this may seem like a good long-term solution. But experienced users, and even dbt themselves, have much to say about the incremental models' limitations.

Incremental models offload the responsibility of tracking which data can possibly change, and which is truly append-only, to the user. This doesn't just apply at the time of creation. It remains the user's responsibility as the schema evolves.

Why move? Not all incremental materializations will magically work better in Materialize. If the scale of the dataset is very large, Materialize may run into limits of its own. But porting an incremental model in a traditional warehouse to a regular materialized view in Materialize equates to handing responsibility for tracking what inputs might change back to the database. That saves data teams a lot of time in debugging and maintenance.

- **Data is slowed down before it enters the warehouse**

  Look at the sources of data you are using in the warehouse. Are you using a service to queue continuous updates into a batch of changes better suited to load into the warehouse? This is a good signal that Materialize can deliver a drop-in capability upgrade by processing the data as it arrives and serving transformed results that are always up-to-date.

  Why move? If you're consuming streams with Kafka/Snowflake, CDC streams from your primary transactional database, or analytics events coming from web, mobile, and server-side sources via services like Segment, Snowplow, and Rudderstack, you may be able to gain performance (fresher data), decrease costs (no separate loading service, less repetitive computation work in the compute layer) and simplify your architecture using Materialize.

Conclusion

Overuse is the hallmark of a great technology. When a tool gives you superpowers, it's natural to use it for every new initiative.

For years, companies have struggled to fit operational data workloads within analytical data warehouses. With an operational data warehouse like Materialize, you can leverage the upsides of an analytical data warehouse for streaming workloads that require up-to-date data. You no longer have to choose between freshness, ease of use, and a low price point.

What's more, Materialize makes this possible using a simple programming model that users of analytical data warehouses will find familiar. You can try it by signing up for a playground environment and running the Materialize Quickstart.