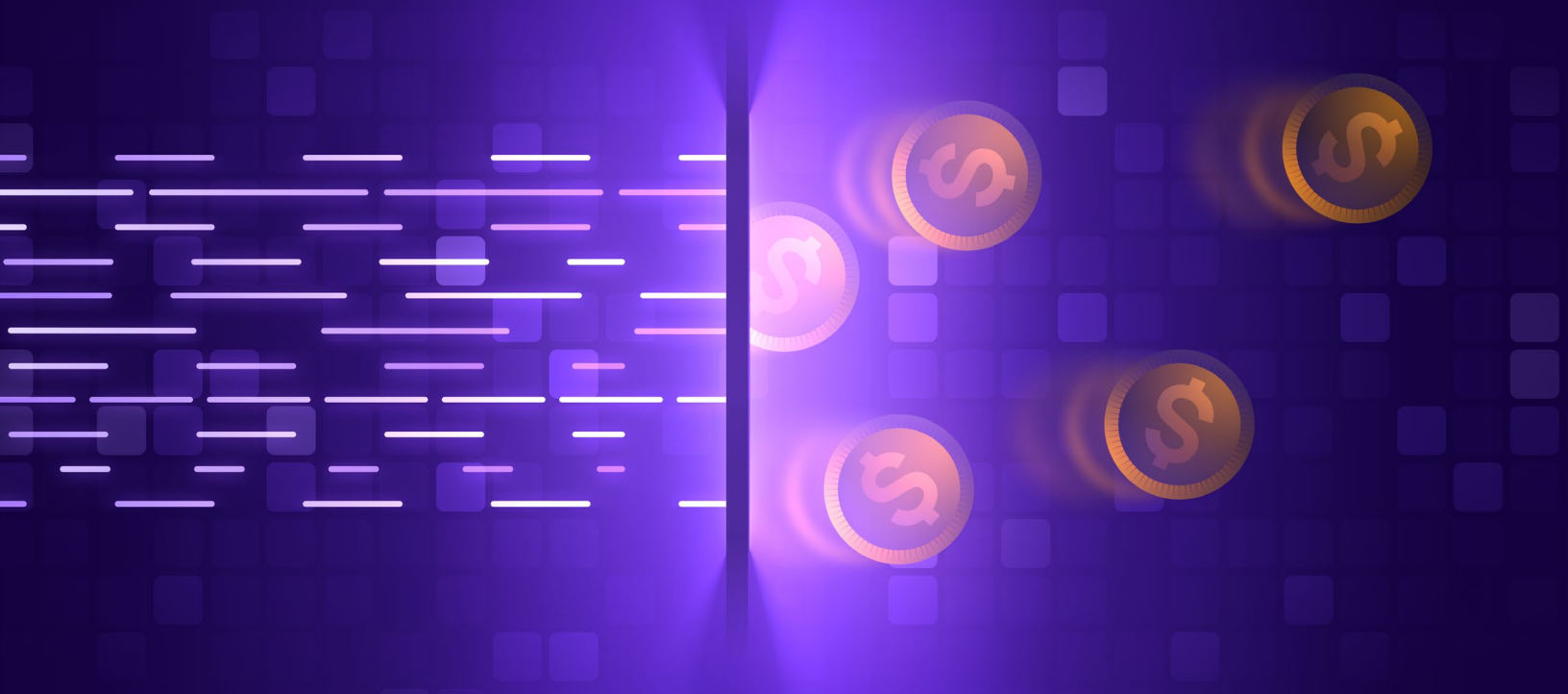
Materialize

WHITEPAPER

Top 6 Strategies for Reducing Data Warehouse Costs



Cost reduction is a key selling point of cloud data warehouses. With serverless architectures, near-infinite compute and storage, and pay-as-you-go pricing, cloud data warehouses can make teams more efficient and cost-effective.

But teams can also neglect or misuse the power of data warehouses. The ease and awesome scale of these platforms can magnify problems such as query recomputation, misaligned compute resources, and suboptimal data storage.

These issues, of course, drive up costs. This causes a data warehouse to become a cost center, rather than a cost saver.

However, reducing data warehouse costs is achievable for most teams if they employ the right strategies. By adopting best-practices, teams can lower the cost of their data warehouse without sacrificing performance.

In the following white paper, we'll discuss the real strategies our customers employ to reduce data warehouse costs. By the end, you'll have concrete ideas about how to lower your own data warehouse bill.

What Causes High Costs in Data Warehouses?

In an ideal scenario, a cloud data warehouse reduces costs. Companies don't need to buy physical servers, they pay only for the computation and storage that they use, and they spend less on maintenance costs.

However, overuse, inefficiencies, and substandard practices can cause data warehouse costs to rise unnecessarily. Key drivers of runaway data warehouse costs include:

Compute Costs	Storage Costs	Infrastructure Costs
High costs result from excessive, inefficient, or redundant use of CPU resources.	Costs balloon from storing too much data or storing data suboptimally.	Data pipelines, fixing malfunctions, and other infrastructure issues elevate costs.

We've outlined the top five strategies for mitigating such data warehouse costs in the next section.

Strategies for Reducing Data Warehouse Costs

1. Incrementally Maintained Views

Traditional data warehouses — or 'analytical' data warehouses — typically operate on a pay-per-query pricing model. You're charged every time you run a query. Cost, then, is tied to query frequency, or how often you perform queries on your data.

Analytical data warehouses run on batch processing, meaning that data is not ingested in real-time, but in bulk at set intervals. Queries are also run at intervals, in tandem with batch jobs. When the queries run, you're charged for the associated CPU resources.

But there's a hidden cost to all of this.

Each time you re-run a query, much of the underlying data has already been computed previously. Instead of only computing the data that has changed, traditional data warehouses re-compute all of the data each time. That means you're charged CPU resources to re-compute data you've already queried.

This inherent inefficiency isn't as costly if you're running a query once a day, let's say for an analytics workload. But if you're running a query several times a day, the cost inefficiency starts to eat into your budget.

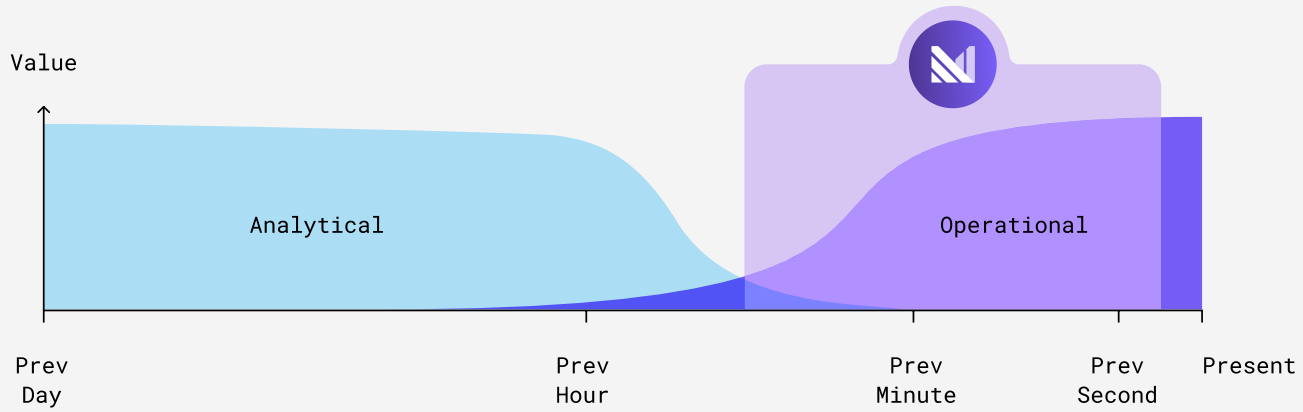
If you're performing real-time use cases such as fraud detection, you need to execute queries every few seconds, rather than every few hours. Real-time workflows demand fresh query outputs in seconds to respond to immediate events.

Analytical data warehouses eventually hit cost and freshness limits when they run real-time use cases. In addition to performance issues, you're also constantly recomputing the same data every few seconds. This is when the cost of query recomputation starts to become unmanageable.

So what's the solution?

Operational data warehouses (ODWs) can significantly reduce the cost of query recomputation. ODWs combine streaming data, SQL support, and incrementally maintained materialized views to decouple cost from query freshness.

To power real-time use cases, operational data warehouses continuously transform streams of raw data into actionable outputs. ODWs allow you to execute SQL queries on fresh data continuously, so you can run real-time operational use cases.



To do this, operational data warehouses such as Materialize leverage incrementally maintained materialized views. Materialized views incrementally refresh query outputs, so you don't need to recompute the underlying data constantly.

Curious about how it works? Let's do a quick whiteboarding session!

Imagine we have a table containing customer purchases called **purchase**.

Customer	date	amount
John	8-7-2020	$30
Sally	8-10-2020	$10
Beth	8-10-2020	$50

Let's calculate the total revenue of this table using the following query:

```
> SELECT SUM (amount)
  FROM purchase;
```

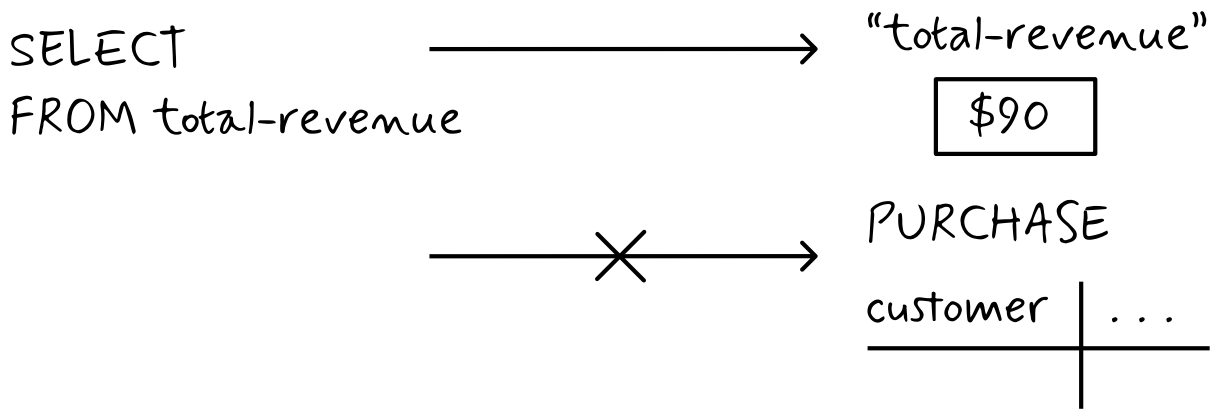
Now let's create a materialized view based on this query.

```
> CREATE MATERIALIZED VIEW total-revenue
  AS SELECT SUM (amount)
     FROM purchase;
```

Unlike queries that calculate their results from tables and views, queries that read from materialized views do not recalculate their results each time.

When this statement is executed, your data warehouse will run your query once to calculate the result. Then, it will physically store that result in a newly created database object — in our case an object named **`total_revenue`**.

Now when you query **`total_revenue`**, your data warehouse will return the stored results without performing any additional computation.



This means that unlike querying a view directly, you only pay for the cost of your query once when creating a materialized view.

But there's a catch. If the underlying data to the query is updated, the materialized view is out-of-date. Let's say we add another entry to our **`purchase`** table. However, since the query was run before the

Customer	date	amount
John	8-7-2020	$30
Sally	8-10-2020	$10
Beth	8-10-2020	$50
Connor	8-11-2020	$20
		$110

"total-revenue"

$90

newest purchase was added to the **purchase** table, the result calculated and stored in **total_revenue** does not take this new purchase into account.

Generally, there are two mechanisms for refreshing materialized views: complete refreshes or incremental refreshes.

Complete refreshes re-run the query underlying a materialized view to completely recalculate and replace the stored results. This requires a full recomputation of the query.

By contrast, incremental refreshes keep materialized views up to date by only performing work on data that has changed. Operational data warehouses allow you to perform incremental refreshes.

Rather than rescan each row of the purchase table to calculate the sum, an incrementally maintained materialized view would only do the following work:

COMPLETE REFRESH:

SUM (30,10,50,20) → 110

INCREMENTAL REFRESH:

SUM (90,20) → 110

Since query recomputation can account for significant compute spend, you can instead use **incrementally maintained materialized views** in operational data warehouses to keep the spend down.

Of course, batch data warehouses can perform incremental refreshes with dbt incremental models. But with Materialize, the refresh is continuous. Each new input record updates the query output immediately.

With Materialize, you don't have to recompute the same data every time you execute a query. This can save you a significant amount of money, especially if you're performing real-time use cases.

2. Offload Real-Time Use Cases

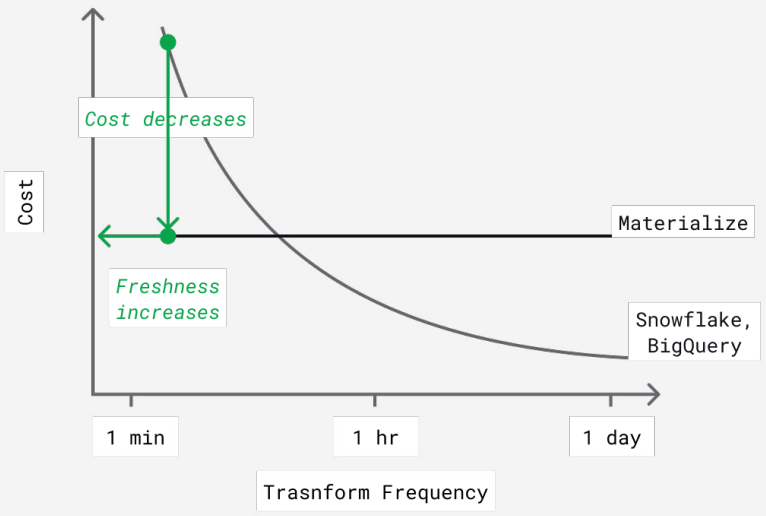
With historical data and flexible querying, analytical data warehouses allow you to develop SQL logic for real-time use cases. But operationalizing these real-time workloads on your analytical data warehouse is difficult.

With analytical data warehouses, these real-time workloads require constant query recomputation and spikes in compute usage. This is a very expensive and inefficient way to leverage CPU resources.

You can avoid these costs by **offloading real-time use cases** to an operational data warehouse such as Materialize.

Materialize operates with static, always-on compute resources, making it far cheaper to run real-time use cases. Analytical data warehouses, on the other hand, charge per query and rely on inefficient spikes in compute for fast responses.

For work that demands freshness: Materialize removes the cost‹ ›freshness correlation



Once you've offloaded your real-time use cases to Materialize, you can start to consolidate and optimize your query workloads on your analytical data warehouse as well. Combine your workloads to maximize compute clusters and minimize costs.

Here's an example.

Let's say you're optimizing compute on your analytical data warehouse. If you run 100 complex queries in a short timespan on an expensive compute cluster, you'll save more money than running less queries on smaller compute clusters. You're actually using less compute resources in the first example.

For your analytical warehouse, you could potentially cut your computation costs significantly with similar optimizations. Many times, you just need to schedule workloads more strategically and run them in parallel.

By offloading your real-time use cases to Materialize, you can optimize your compute usage across your data stack, and reduce your data warehouse bill.

3. Data Mesh

For decades, most data architectures were designed as centralized, top-down infrastructures with a single controlling entity. This entity — typically a data team — implemented the data governance for all business domains, from sales, to marketing, to product.

In this paradigm, costs quickly ballooned. The price of extracting, transforming, and delivering data to business users in their preferred format becomes expensive across so many pipelines, databases, and other infrastructure. In this system, it became difficult to produce, catalog, and find high quality data.

This old paradigm also disempowers domain experts who know their teams best. For instance, instead of allowing the marketing team to control and govern its own attribution data, centralized data architectures outsourced this responsibility to data team members who lacked subject matter knowledge.

Recently, new paradigms such as **data mesh** have emerged in recent years. Data mesh is a decentralized data architecture that organizes data by business domain, rather than as a central, monolithic infrastructure.

With data mesh, domain experts can set the governance policies and protocols that work best for their teams. Team members can access data through self-service and iterate on data quickly, rather than waiting on data experts.

By giving business users direct access to their data, the data mesh framework removes costs associated with older architectures. Some of the ways you can save money with a data mesh architecture include:

- **Data belongs to business owners** - Business owners control their own data, reducing friction between IT and business users. Teams can deliver high quality data products in less time.
- **Data is easier to find** - Data cataloging and data quality tools make it easier for business users to find high quality data.
- **Automated data governance** - Federated computational governance automates data governance, producing high quality data with less manual effort.
- **Eliminates redundancy** - With a holistic view of the business teams, you can eliminate duplicative data and processes.

This last point is important. If you have several different teams working in a decentralized architecture, the cost of duplicative data, queries, and other processes can rapidly grow in your data warehouse.

But with Materialize, teams can leverage a flexible operational data warehouse to power a decentralized data architecture. Teams can collaborate across different business units to build data products, all without raising costs.

Here's an example. With Materialize, you can share materialized views across multiple clusters.

For instance, let's say Team A has a materialized view that Team B wants to incorporate into a real-time data product.

Both teams have their own clusters. However, Team B can leverage Team A's precomputed materialized view without having to use their own compute resources.

Team A can publish their data product as a materialized view. Team B can read those results in their own cluster. Then Team B can implement those results in their own real-time data product.

This allows Team B to avoid running duplicative queries, thus saving money on compute costs. This is what operationalizing the data mesh looks like.

If done correctly, the cost savings of data mesh are real. However, an inefficient implementation can lead to a higher data warehouse bill.

That's why you need a malleable data warehouse such as Materialize to unlock the full potential of decentralized data architecture.

4. Normalized Data Models

Normalization organizes database tables into smaller, more manageable units to reduce data redundancy and improve data integrity. This helps eliminate data duplication, so you can avoid anomalies when you insert, update, or delete data.

Normalized data is divided into normal forms, or rules that ensure data accuracy and consistency. The most implemented normal forms include:

- **1NF** - Eliminates duplicate dependencies
- **2NF** - Removes partial dependencies
- **3NF** - Eliminates transitive dependencies

For example, let's say you have a table that tracks customer orders. You can normalize the table by breaking it into smaller tables, for customer address, name, order, and so on. The tables are linked together with a foreign key.

By removing data redundancy, normalized data enables you to construct more efficient and pliable data models. Normalized data makes modeling a business much easier.

Large joins become a scaling issue for normalized data models. This leads to performance sacrifices with the data model. To add joins, teams must engage in time-consuming revisions of their models. As a result, data models suffer from limitations, and require more work for teams to build.

To fix this performance issue, some teams attempt to denormalize data. Denormalization combines data into larger tables, which can reintroduce redundant data. Queries can access the data faster, improving performance. But denormalized data can corrupt data modeling with duplicative data and other anomalies.

Materialize offers the best of both worlds. With Materialize, you can keep all the benefits of a normalized data model, while still performing large joins.

With Materialize, you can use streaming joins to combine normalized data. Materialize ingests real-time data, and updates materialized views (containing your JOINs) every time the underlying data for the query changes.

This enables you to incrementally join data as it streams into Materialize, instead of all at once upon query execution. By incrementally maintaining joins, Materialize eliminates the scaling issues associated with large joins.

Materialize allows you to maintain a disciplined, normalized data model without making performance sacrifices in terms of joins. When business requirements change, you can simply add another join to the view instead of revising the entire data model.

This extensibility empowers you to build flexible data models that take your business analysis to the next level.

5. Sink Precomputed Results

One of the critical benefits of an operational data warehouse is continuous data transformation. Materialize leverages streaming data, SQL support, and incremental updates to continuously and cost-effectively transform data.

This is ideal for real-time use cases that demand constant computation, such as user-facing analytics and fraud detection. But you can also harness Materialize's transformations throughout your data stack.

With Materialize, you can sink results to external tools. Sinks are the inverse of sources and represent a connection to an external stream where Materialize outputs data.

When a user defines a sink over a materialized view, source, or table, Materialize automatically generates the required schema and writes down the stream of changes to that view or source. In effect, Materialize sinks act as change data capture (CDC) producers for the given source or view.

This enables many use cases in your external data tools. For instance, you can maintain the most up-to-date query results in Materialize, and then sink them with your analytical data warehouse.

This will allow you to access the freshest query results in your analytical data warehouse without having to compute them. Instead of engaging in expensive transformations, the analytical data warehouse can focus on simpler aggregations. You can produce historical analytics based on how the computed results changed over time. This kind of analysis augments your standard historical reporting.

And you won't need to recompute these queries in your analytical data warehouse, saving you money on compute costs. This is how sinking results from Materialize can cut costs across other tools in your data stack.

6. Real-Time Analytics

One of the promises of a modern data warehouse is that multiple teams and users can retrieve, query, and analyze data simultaneously. However, this flexibility can also drive costs upward. Cost centers develop when various teams are working on the same data warehouse, performing tasks in parallel.

For instance, consider a real-time analytics use case on a data warehouse.

Users have a need for near real-time analytics, but this is hard to achieve on batch warehouses. Usage spikes as a result. When users continuously update their dashboards and reports throughout the day, costs for query recomputation grow considerably.

Without set limits on usage, users can continue to refresh their analytics as much as they want, and constantly incur costs. But putting hard limits on usage isn't the answer. Your users are refreshing their dashboards and reports because they need up-to-date results, not because they want to waste money.

They want real-time analytics to perform analysis and make decisions. The goal, then, is not to limit usage. It's to enable real-time analytics without driving up costs.

With operational data warehouses such as Materialize, you can build real-time dashboards and reports without subsuming query recomputation costs. This allows you to cost-effectively generate real-time analytics by incrementally updating materialized views.

Now you can stream your real-time results to BI tools for visualization. Materialize is PostgreSQL wire-compatible and can connect to BI platforms such as Looker, Metabase, and more. Your users can leverage real-time BI without constant data refreshes. But this is just one use case for real-time analytics in Materialize.

Increasingly, we're seeing our customers use Materialize and real-time analytics in their user-facing apps. They're also leveraging these analytics to power their business processes. Onward Delivery shared one of these use cases with us in a Customer Story.:

> "Our data team was serving real-time delivery status and location analytics to customers with two weeks of work and minimal ongoing maintenance."
>
> **- CLAYTON VON HOVEL, DATA ENGINEER AT ONWARD DELIVERY**

Cut Your Data Warehouse Bill. Deploy These Strategies Now!

One of the key benefits of a data warehouse is the potential to unlock cost efficiencies. But in practice, data warehouses often develop their own cost centers, from query recomputation to inefficient storage.

However, you have the power to restrain and eliminate these cost centers. By deploying the strategies we've discussed, you can bring down data warehouse costs, and achieve savings that will power your team's new projects.