Keebo's Complete Guide to Optimizing Snowflake's Cost and Query Performance

A Survey of Manual and Automated Approaches

Introduction

Snowflake is one of the most popular cloud-based data warehouses. It went public in 2020 with the largest software IPO to date. Snowflake offers a secure and scalable database engine with a unique architecture that decouples compute and storage resources, allowing customers to pay only for the resources they need and use. Unlike traditional (on-prem) data warehouses, Snowflake provides a flexible and fully managed data analytics solution (Software-as-a-Service).

However, Snowflake's pay-as-you-use pricing model and the fact that it is a shared service across the entire organization mean costs can add up quickly. As data volume and computation needs grow, Snowflake costs will too – sometimes astronomically. Users can create automated alerts for when usage exceeds a certain budget, but it's not a long-term solution as organizations still need to meet their growing needs. Ultimately, proper optimization of your Snowflake is the only viable route to preserving resources as you scale.

Optimizing Snowflake Is a Challenge, but Crucial

Performance optimization is always a challenge, but with cloud-based data warehouses (and Snowflake is no exception) it is even more challenging for several reasons. First, compared to traditional data warehouses that have matured over decades, cloud data warehouses are newer offerings focused on ease-of-use by average users, rather than advanced database engineers

who are accustomed to fine-grained tuning knobs. Second, cloud data warehouses are adopted by more modern and data-driven companies. Their user base is more diverse with varying levels of database proficiency. This has increased the chances of poorly written queries, sub-optimal schemas, and inefficient data ingestion patterns. Finally, the ease of scaling up/out a cloud data warehouse has created a shortcut for solving performance problems, causing companies to overlook proper database optimizations at the cost of paying additional compute costs.

Cost optimization is even more challenging when it comes to Snowflake. The reason is Snowflake's pricing model. The overall costs depend on the customer's usage of storage, virtual warehouses (a.k.a. compute), cloud services, and serverless features. However, for most customers, the dominant factor is the compute cost. Simply put, you are charged a certain number of credits (depending on the size of your warehouse) for every second the warehouse is running. Because of this pricing model, optimizing query performance alone may not be sufficient for reducing the overall compute costs.

Nonetheless, Snowflake is one of the most popular cloud data warehouses in the world. Investing in performance and cost optimization will determine your long-term success on the platform. This is critical for not only improving your user experience and adoption of data-driven initiatives across your organization but also for freeing up engineering and financial resources. With optimization, you can funnel those freed up resources into your core business rather than managing your cloud data warehouse.

Since compute costs are at least an order of magnitude greater than storage and cloud services costs for most customers, the rest of this document focuses primarily on optimizing compute costs.

We will present two approaches to optimization: manual and automated. To decide which of these two options is a better fit for your organization, you can use the following guidelines.

Which Optimizations To Use

When to Use Manual Optimizations

Manual optimizations include data ingestion optimizations, query optimizations, schema optimizations, and warehouse optimizations.

For data ingestion, manual optimizations are almost always effective. This is because data ingestion is often overseen by a centralized team of database experts, and ingestion pipelines do not change too frequently in most organizations. As a result, once the input files, formats, and scripts are optimized, there is little need for ongoing monitoring or training of end users.

When it comes to optimizing the schema, queries, or the warehouse itself, manual optimizations are most effective for small workloads and organizations where there are fewer users and queries. While a great start, manual optimizations are not scalable as they are time-consuming, hard to implement, and most importantly, sub-optimal, as we will explain next.

When to Use Automated Optimizations

As mentioned above, manual optimizations are time-consuming, challenging, and less optimal with larger and more complex workloads for several reasons:

1. Most optimizations are a tradeoff that requires a careful analysis of competing factors. One has to carefully weigh the positive and negative impact of any changes while accounting for a large number of complex parameters in the query workload and usage patterns. Rigorously quantifying the impact of each decision can take hours of work from highly skilled engineers. As a result, most rely on simplified rules of thumb, which lead to decisions that are directionally helpful but not necessarily optimal.

2. In most organizations, queries are written by a wide range of users. Even schema changes are not always initiated by a centralized department. This makes it challenging to implement company-wide query or schema optimizations, as it requires training and cooperation of many users with different levels of SQL proficiency, continuous monitoring of queries and schema changes, and enforcing company-wide policies.

3. Workloads and databases are living objects. What's optimal today may no longer be optimal next month or after the next software release. In that sense, performance improvement projects are almost a never-ending project. If you invest in manual query optimization, make sure you carefully weigh the potential performance benefits against

the cost of expensive engineering resources that can otherwise be spent in other areas with more impact.

4. Even the most skilled DBAs are still humans: they make mistakes and they get tired and overwhelmed. If you have a large workload with hundreds of thousands of queries, they can only focus on a small subset of them. Even if they manage to magically speed up 1000 of your slowest queries, it may only have a small impact on your overall compute costs. The reason is Snowflake's pricing model charges you based on the number of seconds the warehouse is running. This means speeding up your slowest queries will only reduce your overall cost insofar as it allows for 1) scaling down an unnecessarily large warehouse that was only needed for those slow queries, or 2) reducing the frequency of Snowflake's auto-scaling of your multi-cluster warehouse due to fewer resources being consumed by slow queries.

The automated approach is much more effective for larger workloads because it rigorously accounts for different parameters across hundreds of thousands of queries, thus guaranteeing optimal performance at the lowest cost. Unlike the manual optimizations performed by humans, the automated approach is not error-prone. It is also more scalable, as it does not require hours of effort from highly skilled engineers and DBAs. More importantly, the automated approach can continuously monitor your warehouse and adjust to your workload in real-time. This ensures optimal performance and the lowest costs at all times, even when your workload changes, new applications are introduced, or usage patterns change.

keebo

Manual Approach

1. Ingestion Optimization

Frequent or continuous data ingestion is one of the most expensive operations when it comes to Snowflake. The reason is that Snowflake bills by the number of seconds the warehouse is up and running regardless of what percentage of the warehouse resources are being used.

2. Query Optimization

Inefficient or poorly written queries are the biggest factor affecting Snowflake's compute costs but unfortunately the hardest to fix manually. The reason is simply that a very diverse set of users across the organization use your database. These users have varying degrees of SQL proficiency. Educating hundreds or thousands of users with different backgrounds is a challenging task. Most users do not have any interest in becoming SQL experts. A common approach taken by most organizations is to have a small team of highly experienced DBAs constantly monitor the query logs. They identify the lowest-performing queries and intervene to optimize or rewrite them.

This approach is not very effective for two obvious reasons.

1. First, consider a small or medium-sized organization issuing 100K queries a month (larger organizations might issue tens of millions). Even optimizing the slowest 1% of queries will mean manually inspecting 1000 queries --- a daunting task to say the least!
2. Second, optimizing a small percentage of queries (even if they are the slowest ones) does not necessarily have a big impact on credit usage, again due to Snowflake's billing model. As long as there are other queries running in the warehouse, regardless of whether they are slow or fast, Snowflake will be charging you a certain number of credits depending on the size of your warehouse. This is not to say that optimizing your slowest queries will not affect your credit usage. It does, but indirectly and only to some extent. By speeding up your slowest queries, you may be able to: 1) scale down an unnecessarily large warehouse that was overprovisioned due to users' performance complaints, 2) reduce the likelihood of triggering Snowflake's auto-scaling if you are using a multi-cluster warehouse.

With these disclaimers, below are some of the common mistakes that are easiest to correct and will impact query performance and cost:

- Avoid using `SELECT *` queries when you can. This is because Snowflake is a column-store, and retrieving a subset of columns will drastically improve performance if your table has a lot of columns.
- Avoid nested queries as much as possible. Query optimizers are not very effective in optimizing nested queries. Here's a simple pseudo-example of an unnecessarily nested query:

```
SELECT A FROM T T1
```

```
WHERE T1.B > ANY (SELECT T2.B FROM T T2 WHERE T1.A =T2.A);
```

This query can easily be rewritten as:

```
SELECT DISTINCT A FROM T T1 JOIN T T2 ON T1.A =T2.A

WHERE T1.B > T2.B;
```

- Use `LEFT JOIN` instead of nested queries with `IN` or `EXISTS` whenever possible. Here's a simple pseudo-example of a query that's using the latter unnecessarily:

```
SELECT A FROM T1

WHERE T1.B NOT IN (SELECT T2.B FROM T2 WHERE T1.B =T2.B);
```

This query can easily be rewritten as:

```
SELECT A FROM T1 LEFT JOIN T2 ON T1.B = T2.B

WHERE T2.B IS NULL;
```

- Avoid using inequality joins whenever possible, as they are one of the hardest queries to optimize by your data warehouse. Here is a simple pseudo-example:

```
SELECT T1.A, COUNT(DISTINCT T2.A)

FROM T T1 JOIN T T2 ON T1.A <= T2.A

GROUP BY T1.key, T1.A ORDER BY T1.A DESC;
```

The following query achieves the same goal but more efficiently:

```
SELECT A, DENSE_RANK() OVER (ORDER BY A DESC) FROM T
```

- When looking for top-K results, where K is a small constant, using `MIN` (or `MAX`) is more efficient than `RANK()` as the latter requires sorting whereas the former can be processed with a simple scan and often benefits from partition-based pruning:

```
SELECT max(A) AS secondA FROM

(SELECT RANK() OVER (ORDER BY A DESC) AS A_rank, A FROM T)

WHERE A_rank =2;
```

  The following query achieves the same goal but more efficiently:

```
SELECT MAX(A) AS secondA

FROM T WHERE A < (SELECT MAX(A) FROM T);
```

- Avoid applying complex functions and expressions to your join key. In its worst case, a complex join condition will lead to nested loop join. If you need a complex expression as your join key, create intermediate results first before performing a join. Equality joins with simpler expressions allow your query optimizer to choose from a wider range of efficient join implementations. The difference between a join that is internally executed as a hash- or merge-join versus one that is executed as a nested loop can be two orders of magnitude (a few seconds versus tens of minutes).
- Try using `UNION ALL` instead of `UNION` whenever possible. In other words, use UNION only if you really need it. `UNION ALL` keeps all records, whereas UNION has to check for duplicates and only return the unique records. The latter requires extra computation (typically implemented with a sort). What we have observed is that in many cases users are

not entirely sure if the result will have duplicates, so they use UNION simply as a "safety mechanism" in their query. Familiarizing yourself with the integrity constraints in your schema and investing in your data quality eliminate the need for extra computations later on.

- If you are joining tables, try to use ANSI joins as much as possible. They are more likely to be properly optimized by the query optimizer. In other words, instead of:

```
FROM T1, T2 WHERE T1.A = T2.B
```

use the following:

```
FROM T1 JOIN T2 on T1.A = T2.B
```

3. Schema Optimization

There are three types of schema in a relational database: physical, logical (a.k.a. conceptual), and external (a.k.a. view) schemas. A proper database design involves making the right decisions based on the business requirements and the data semantics at all three levels of the schema. In fact, all three schemas can directly impact both data ingestion and query performance, thereby affecting your Snowflake's compute usage.

- Views come with many benefits and are often created for security, logical data independence, or encoding business logic. They also significantly improve query readability and reduce the likelihood of mistakes in query logic. However, an overlooked benefit of views in a large organization is that they reduce the likelihood of

less-experienced users writing inefficient SQL queries. Similar to materialized views (see below), if the DBA team can identify expensive and common building blocks and turn them into efficiently-written views, the less-experienced users in the organization who are more likely to write inefficient queries can simply reuse those views. This usually leads to better overall performance.

Unfortunately, like most beneficial things, views can also be a double-edged sword. Most databases inline the view definition in the queries that reference them, and then pass them to the query optimizer. This means, with a deep hierarchy of complex views, Snowflake's query optimizer is now effectively faced with a highly nested query which is much harder to optimize. Materialized views, when applicable (see below), can prevent some of those problems.

- If you are using Snowflake's enterprise or higher editions, you have access to materialized views. Using materialized views can significantly improve performance if you can identify common building blocks of your queries that are expensive, but may or may not reduce your overall credit usage. This is because Snowflake will pre-compute those building blocks in advance, and thus queries that can use those materialized views are processed faster when they arrive. However, whether materialized views reduce or increase your overall cost depends on the percentage of your queries that benefit from those materialized views versus the frequency at which your underlying data changes causing Snowflake to recompute your materialized views on your behalf. In general, identifying a small set of highly effective materialized views that

benefits the largest number of your slow queries and requires the least amount of update overhead can be a daunting task when done manually. An automated approach is much more effective in analyzing the common building blocks across hundreds of thousands of queries.

- Clustering keys are part of your physical schema that have the largest impact on performance when it comes to Snowflake. Each database employs a different data partitioning strategy. Snowflake uses micro-partitions, which are contiguous units of storage, each containing 50-500 MB of uncompressed data. Groups of rows in each table are mapped into individual micro-partitions, compressed, and organized in a columnar fashion. When the rows in each micro-partition are clustered according to some popular attributes ("clustering key"), Snowflake can efficiently skip the irrelevant micro-partitions when processing queries that reference your clustering key. This is quite beneficial for extremely large tables. By default, Snowflake auto clusters your rows according to their insertion order. Because insertion order is often correlated with dates, and dates are a popular filtering condition, this default clustering works quite well in many cases. You can use Snowflake's SYSTEM$CLUSTERING_DEPTH system function to see how effective your micro-partitions are in terms of a specified set of columns. It returns the average depth as an indicator of how much your micro-partitions overlap. The lower this number, the less overlap, the more pruning opportunities, and the better your query performance when filtering on those columns. There are at least two situations in which you should consider using Snowflake's auto clustering or specifying a user-defined clustering key:

a. When you have too many DML statements after your initial data load, your clustering may lose its effectiveness over time: more overlap means fewer pruning opportunities. This can be determined using the above mentioned system function and observing a higher average depth. Snowflake used to allow for manual reclustering in the past but it was expensive, and users had to decide whether performing it outweighed the additional costs. Starting May 2020, Snowflake no longer allows for manual reclustering. Instead, all tables enjoy auto clustering by default, which means Snowflake determines when it is worth reclustering a table. You can check the costs incurred by auto clustering, and suspend it if you want. However, we do not recommend that, as excessive auto clustering costs mean your DML operations are in conflict with your current clustering key. You should consider changing your clustering key instead (see below).

b. Determining an effective clustering key is essential for good query performance on massive tables. To choose the best clustering key, you should choose a subset of columns (or expressions) based on the following considerations:

    i. The cardinality of the columns you choose should neither be too small nor too large. For example, clustering on a column with only 3 distinct values means, in the best case scenario, a query with an equality predicate on that column will run 3x faster (skipping 66% of your data), whereas clustering on a column with 1000 distinct values will mean that an

equality predicate on that column can potentially run 1000X faster. Likewise, a column with extremely high cardinality (e.g., a primary key or timestamp) is not a good candidate for a clustering key either. The reason is that the cost of maintaining a high-cardinality clustering key outweighs the performance benefits of such a clustering key. The best strategy for clustering on high-cardinality columns is to instead use an expression as the clustering key by applying a function that maps the column values into fewer distinct values. For example, you can use `to_date(A)` when A is a timestamp or use `trunc(A)` when A is a number to truncate it to fewer significant digits, e.g., `trunc(123456789, -5)`.

ii. Columns that appear frequently in range or equality predicates in `WHERE` clauses are the best candidates. The second-best candidates are columns that are used as join keys. For example, if you have a common join pattern:

```
FROM T1 JOIN T2 ON T1.A = T2.B
```

A and B may be great clustering keys for T1 and T2, respectively. Finally, columns appearing in your `DISTINCT`, `ORDER BY` and `GROUP BY` clauses can be the next set of good candidates as part of the clustering key, as these clauses often lead a sorting operation too.

iii. You can use multiple columns as your clustering key. For instance, if you have a lot of queries searching by

date and region, you can define your clustering key as the combination of date and region. Snowflake recommends choosing a maximum of 3-4 columns (or expressions) for your clustering key. When you cluster on too many columns, your effective cardinality can be as large as the multiplication of the cardinalities of the individual columns (assuming no correlation).

iv. When using multi-column clusters, the ordering matters. Assuming no correlation, you want to specify the columns in `CLUSTER BY` clause from the lowest cardinality to the highest. This leads to the largest speedup.

4. Warehouse Optimization

As previously mentioned, data ingestion can be an expensive operation due to Snowflake's billing model.

- To minimize your credit usage, you can take the following steps depending on whether you are using a dedicated warehouse for your data ingestion or sharing one with other applications:
    a. If you use a dedicated warehouse for data ingestion, you can explicitly start it right before the data load process starts and stop it immediately after. However, if you have enabled auto-resume for your warehouse and auto-suspense is set to a very small period (say 5 minutes or less), this is unnecessary because Snowflake does it automatically. However, in that case, you need to make sure that no other applications or

queries will mistakenly hit your warehouse after the load process completes. If it happens, they will resume (i.e., wake up) your warehouse and you will be paying again until the auto suspense kicks in.

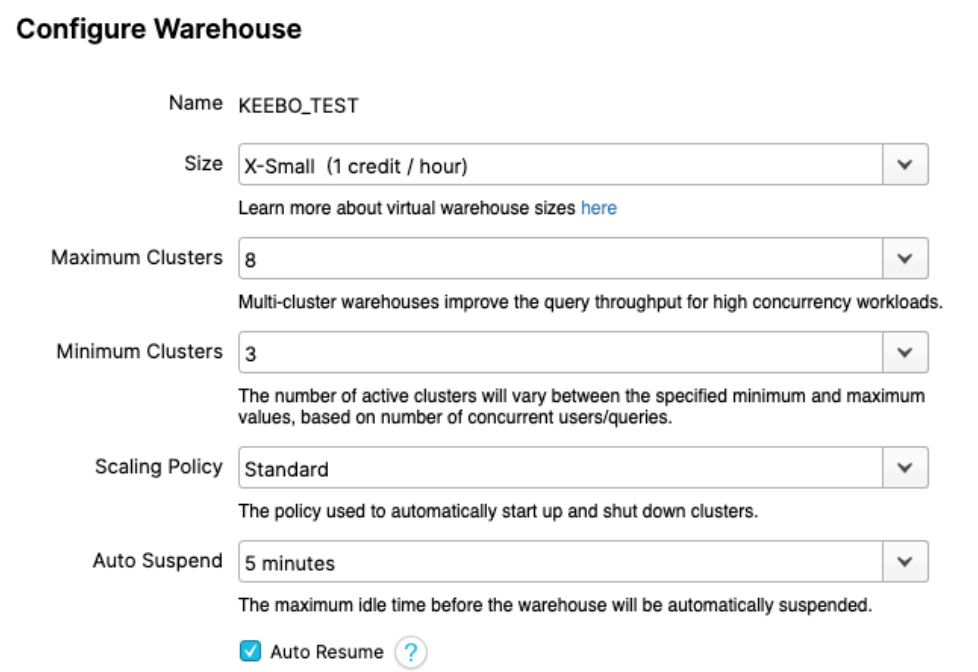
b. If you are sharing the warehouse with other applications, you can scale up your warehouse right before the data load and scale it back down right after. These steps help minimize credit usage.

- One of the most important decisions that directly affect your compute cost is rightsizing your Snowflake instance type, which is referred to as warehouse size in Snowflake Terminology. Snowflake offers the following instance types currently, each costing a different number of credits per hour. Although Snowflake bills by the second, each time a warehouse wakes up (i.e., resumes), Snowflake charges a minimum of 60 seconds.

WAREHOUSE SIZE	CREDITS/HOUR
X-SMALL	1
SMALL	2
MEDIUM	4
LARGE	8
X-LARGE	16
2X-LARGE	32

3X-LARGE	64
4X-LARGE	128
5X-LARGE	256
6X-LARGE	512

If you are using Snowflake's Enterprise edition or higher, you also have access to multi-cluster warehouses, a feature that allows for automated scale out. Regardless of your warehouse size, you can set a minimum and a maximum number of clusters along with a scaling policy that allows Snowflake to automatically start additional clusters on your behalf (up to the maximum specified) when the workload exceeds the current warehouse resources and automatically suspend them (down to the minimum specified) when the additional warehouses are no longer needed.



Increasing the size of your warehouse, say from Large to X-Large, is called “scaling up”. Increasing the number of clusters, say from a 2-cluster warehouse to a 3-cluster warehouse, is called “scaling out”. In theory, because of Snowflake’s pricing, the credit cost of running an X-Large warehouse for an entire hour is no different than running a Large warehouse with 2 clusters both running for the entire hour. In practice, however, there is a world of difference both in terms of performance and cost. Below is our recommendation of when to use which one.

- First, pick a warehouse size that can efficiently process “most” of your queries. Provisioning a warehouse for your slowest query is too expensive, as it would mean most of the time you will be paying for underutilized compute resources. If you expect heavy queries (e.g., a daily ingestion task) during a specific time window, you can scale up your warehouse size programmatically only for the duration of the

heavy queries. If your heavy queries are scattered throughout the day, it is much harder to achieve both good performance for those queries and remain cost-efficient. You need to rely on the automated approach in those scenarios.

- If you have a lot of queries hitting your data warehouse concurrently, you are better off going with a multi-cluster. This is because, almost always, the number of queries fluctuates during different hours of the day. A multi-cluster will allow for significant savings by turning off the clusters during times when your load goes down and bring them back up when you hit your peak load.
- As a rule of thumb, scaling up helps when you have slow queries but scaling out helps when you have a lot of concurrent queries. The most common mistake we observe is relying on `TOTAL_ELAPSED_TIME` to determine whether you have slow queries. The `TOTAL_ELAPSED_TIME` can be misleading as it is typically the symptom and not the cause. To find out the root cause, you need to look at the following fields in the `QUERY_HISTORY` view:
    a. `COMPILATION_TIME:` a large compilation time is typically indicative of overly complicated queries, e.g., highly nested queries with a lot of `UNION` clauses.
    b. `EXECUTION_TIME:` a large execution time means the query is an expensive one. A large number of queries with large execution times is a signal that you need to either optimize those queries or scale up to a larger warehouse.
    c. `QUEUED_PROVISIONING_TIME`: a large number of queries with a non-zero queued provisioning time means the cluster is being

suspended too frequently. This means you need to increase your "Auto Suspend" interval.

d. `QUEUED_OVERLOAD_TIME`: This is a crucial piece of information to look at. Oftentimes, large `TOTAL_ELAPSED_TIME` is simply due to `QUEUED_OVERLOAD_TIME` rather than a large `EXECUTION_TIME`. Whenever there are a lot of queries with considerable `QUEUED_OVERLOAD_TIME`, it means a lot of faster queries are queued up behind slower queries. To address this, you should either scale out (by increasing the number of clusters in your multi-cluster warehouse) or separate the slow and fast queries into separate warehouses. Both solutions address your performance problems at the cost of increasing your compute cost, but the latter (separation of the warehouses) is almost always a more expensive approach.

- There are several great reasons for creating separate warehouses for your different applications, users, and queries. Part of Snowflake's popularity is that it makes creating separate warehouses accessing the same data extremely simple. For example, you may separate your workloads for security reasons, to shield your users from heavy-duty tasks (e.g., data ingestion), or even allocating different budgets to each department (e.g., ease of accounting). What you and your organization need to know is that **separating your workload into multiple warehouses will almost always increase your compute costs.** The reason is simple. Snowflake bills you by the number of seconds that your warehouse was running multiplied by the credits/second for that warehouse size. To illustrate why separating your workload into multiple clusters can drastically reduce your costs, consider the following toy example.

Assume you have a query Q1 arriving at 9:00 am that runs for 60 seconds in your Large warehouse. You have a second query Q2 arriving at 9:01 am that runs for 120 seconds in the same warehouse. Even if you have aggressively set your auto suspend to be 3 minutes, you will be paying for your cluster running from 9 am to 9:06 am. A Large warehouse costs 8 credits/hour, so you will be paying 8*6/60=0.8 credits for these two queries sharing a Large warehouse. Now let's compare this to a situation where you have separated your workload into two warehouses, W1 and W2, where Q1 is served by W1 and Q2 is served by W2. To deliver the same performance as before, W1 and W2 need to be Large warehouses too. Assuming queries arrive at the same time for simplifying our analysis, now you will be paying for W1 running from 9:00 to 9:04 (4 mins) and for W2 running from 9:01 to 9:06 (5 mins). This means in total you will be paying for running Large warehouses for 9 mins, namely 8*9/60=1.2 credits. The only way to keep the costs comparable in this example is to downsize W1 and W2 to Medium warehouses, which will mean roughly doubling the latency of Q1 and Q2!

The above example is contrived, but hopefully, it illustrates why Snowflake's pricing model makes it expensive to separate workloads. This is not to say that the entire organization should be sharing a single warehouse. In fact, there are many good reasons for using multiple warehouses for different departments. However, these decisions must be made with the full understanding that they will come at a financial cost. Like most of the other optimizations described here, analyzing if and how to scale up or out requires a lot of investigative work.

In our case studies, analyzing queries and rightsizing decisions can take significant effort from the database team. In the end, because of

the sheer complexity of analyzing hundreds of thousands of queries and all the factors that need to be accounted for, decisions on size and number of warehouses you use for each application is never going to be "optimal". In other words, you are always going to leave money on the table. However, even running some basic experiments might help you directionally with improving your Snowflake setup. If you really want to retain an optimal Snowflake setup at all times, you need to rely on the automated solution

- Choose Snowflake's auto-suspend parameter carefully. This parameter allows for automatically suspending a warehouse whenever there is no activity for a specified period. You can also enable auto resumption which means your warehouse is automatically resumed as soon as a new query arrives. The auto suspension can help reduce your compute bill, but if the interval is not chosen carefully can have the opposite impact. Most Snowflake customers, misled by the fact that a suspended warehouse can often be resumed in 1-2 seconds, end up selecting an aggressively small auto suspension period, say 3 minutes or even less. The reason why such a small value can have a drastic impact on performance, and thereby cost, is that whenever a warehouse is suspended its entire cache is dropped! This means even if the warehouse is resumed after 1-2 seconds, the subsequent queries may take significantly longer simply because they now have to read data from cold storage (e.g., S3) rather than local memory. Depending on your workload's access pattern and working set, it may take a while before the relevant data for the upcoming queries is cached again. There is at least one order of magnitude of performance difference between processing a query on cached data versus cold data. Choosing an optimal auto suspend interval requires

analyzing the average gap between your subsequent queries across various applications and users. Strike a balance between an exceedingly large interval versus an overly aggressive parameter.

- Make sure you always have some resource monitors in place to avoid unexpected credit usage. Snowflake allows you to monitor the credit usage per day, week, month, or year, whereupon reaching a quota an automated alert is sent or the corresponding warehouses are automatically suspended. We recommend that you have multiple monitors in place to ensure each department or application stays within its allotted credit budget for the month. You can also define a smaller daily quota to prevent extreme mistakes. For instance, if a department is meant to spend no more than 3000 credits a month, it is a good idea to also define a lower budget, say 500 credits, for their daily usage. You should not set the daily limit too small, say 3000/30=100, to accommodate natural workload fluctuations. If you do not have any daily monitors in place, then a user mistakenly starting a massive warehouse without suspending it or an application bug issuing continuous queries may quickly burn through all your credits for the month. Unfortunately, you cannot define quotas on a per-user basis, but in its most granular form, you can define them on a per-warehouse basis.

Keebo's Automated Approach

While manual optimizations are most effective for small workloads and organizations where there are fewer users and queries, they are not scalable or effective for dealing with larger workloads. Manual optimizations are time-consuming, hard to implement, and sub-optimal for several reasons (as explained earlier). In this section, we will introduce Keebo's automated approach to cost and performance optimization.

What Is Keebo?

Keebo is a turn-key data-learning platform that sits invisibly between the data warehouse and the users and **automatically optimizes** the data warehouse, **speeding up analytical queries by one or two orders of magnitude** and/or **drastically reducing compute costs**. Keebo is not a data warehouse but rather a **platform-independent warehouse optimizer**, which means it can accelerate or reduce the cost of existing data warehouses/lakes and can support any BI tool or existing application. Keebo is also a transparent **drop-in** solution. It does not require any data migrations or any modifications to existing tools or applications.

- **Automating a Tedious Process.** Keebo learns the query patterns and analyzes the underlying data automatically, without the need for any configurations or hints from users or administrators. It also detects data updates and reacts to changes in the workload automatically. In other words, Keebo is a fully automated turn-key solution, freeing up hundreds of hours of engineering time spent on tedious performance optimization tasks.

- **Reducing Costs.** Through a suite of fully automated optimization algorithms, Keebo drastically reduces the computational costs of the underlying data warehouse. Keebo enables organizations and data teams to accomplish more with fewer resources.

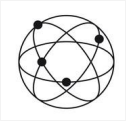
- **Improving User Experience and Productivity.** Instead of your customers and users waiting minutes to see their query or dashboard results, Keebo delivers an answer to their queries or loads their dashboards in 2 seconds or less. Not only does a 2-second response time drastically improve productivity and user experience, but it also lowers the barrier for users who are less technical to create their own queries and dashboards, increasing organizational adoption of data-driven insights.

Keebo's Architecture

Keebo performs its operations on top of your existing data warehouse while appearing as a data warehouse from the client's perspective to avoid the need for any modifications to your existing applications.

Keebo provides the native interface of popular databases (e.g., Postgres, Snowflake, Redshift). Thus, any client, application, or BI tool that connects to those databases can also connect to Keebo. In other words, customers do not need to install any custom drivers to use Keebo.

How to use Keebo?



Step 1: *Connect Keebo!*

Keebo is a turn-key SaaS solution, which means you can sign up and connect it to your data warehouse with just a few clicks. Keebo can connect to any data warehouse. If, in addition to reducing your compute costs, you also want to accelerate your queries, you can route your queries through Keebo to your data warehouse. You can do that by connecting your existing BI or other applications to Keebo using the vendor's own driver, eliminating the need for any migrations. This means, apart from a change of IP address, the BI or application thinks it is still connecting to the underlying data warehouse, and the data warehouse thinks the queries are coming directly from the BI or user application. In other words, Keebo is a drop-in solution compatible with today's BI tools and data warehouses without requiring any data migration or changing a single line of code in existing tools and applications.

You do not have to connect all your applications to Keebo. You can choose which applications, users, or dashboards need to be accelerated and only

connect those to Keebo. Even if you do not connect any of your applications to Keebo, you can still connect Keebo to your data warehouse to optimize its performance and reduce its compute costs.

Step 2: *Let Keebo automatically learn and optimize your workload*

Once Keebo connects to your data warehouse, it automatically starts to learn its "smart models" by analyzing your workload: warehouse settings, database schema, data distribution, and query patterns. This process is called Data Learning, which considers many factors before deciding which optimizations to use:

1) Your data distribution. For example, it identifies columns with high or low cardinality, their ranges, whether they have skewed distribution or not, as well as the correlations among them.

2) Your query distribution. For example, Keebo's Data Learning analyzes your past query logs to identify common join patterns, grouping conditions, or popular filters. It also takes the frequency of each pattern into account and observes how they change over time.

3) Your warehouse settings and past performance. Keebo analyzes the size and parameters of your data warehouse, as well as past query latencies to determine the most effective optimizations.

Keebo uses a fully automated process and does not require any assistance or manual intervention from users in determining and applying its optimizations.

To avoid confusing your users and existing applications, Keebo never modifies your existing database schema. All schema changes, if needed, are

performed under a separate schema, specifically created for Keebo's own operation, which is then used to answer Keebo's rewritten (i.e., optimized) queries.

Keebo continuously monitors the cost and performance of the incoming queries and adjusts its optimizations accordingly to account for any changes in the workload. Optimizations that are no longer beneficial are automatically retired, and new ones are created as needed to accommodate the additional load, new query patterns, or changes to the underlying distribution of the customer data.

Step 3: *Enjoy costs savings and acceleration*

Within days, you should see a drastic reduction of your compute costs (if you use a cloud data warehouse, such as Snowflake) and/or significant acceleration of your queries. Keebo's portal reports KPIs on both cost and performance metrics so you can monitor your RoI and automated optimizations performed by Keebo.

Get In Touch

To learn more about Keebo or a free trial contact us at info@keebo.ai or visit https://keebo.ai