What is a Materialized View?

A Materialized View in Tinybird is a combination of a Pipe and a Data Source. Instead of publishing a Node as an API Endpoint, you can **write the results of a query into another Data Source.**

Unlike other databases, Materialized Views in Tinybird are incremental and triggered upon ingestion. This means that, as you ingest new rows, data is automatically processed by the materialization query, combined with the previous result, and written to the Materialized View. Because of this, there is no schedule to maintain for refreshing data in a Materialized View.

Materialized Views are processed at ingestion time, making them an excellent way to preprocess data before being used in API Endpoints. Materialized Views are critical for reducing both latency and cost-per-query.

What should I use Materialized Views for?

There are three common use cases where Materialized Views shine: aggregations, schema changes and use-case optimized sorting keys.

Aggregations: you can find significant cost & performance optimization using Materialized Views when your API Endpoints need to do some kind of aggregation (e.g., `max()`, `sum()`, `avg()`). With Materialized Views you preaggregate data at ingest time, meaning your API Endpoints need to process significantly less data to serve the request.

> Note: if the query contains an aggregation function like `max()`, `avg()`... your query will be rewritten to use `-State` or `SimpleAggregationFunction()` modifiers, and the Data Source Engine will be `AggregatingMergeTree` too.

Schema changes: often, the shema of your incoming data does not match your desired output schema, e.g., dropping columns, combining some columns, extracting fields from JSON, type conversions, joining sources, etc. It may make sense to do this preparation in a Materialized View, transforming the data once at ingest time, meaning your API Endpoints need to process significantly less data to serve the request.

Use-case optimized sorting key: sorting keys allow Tinybird to efficiently skip data that is not needed to answer a query. Choosing an appropriate sorting key is critical for cost & performance optimization, however, the best key can vary depending on how you intend to read the data. Materialized Views can allow you maintain the same data with different sorting keys, to optimize for different read patterns.

> Note that you should try to avoid big scans in Materialized Views and, when using JOINs, you should do them with a subquery to the Data Source you will join to, not the whole Data Source.

Creating Materialized Views in the UI

Write your pipe, using as many Nodes as needed. In the top right corner of the Tinybird UI click on the ▽ next to Create API Endpoint, and select Create Materialized View. Select the node you want to use as output, edit the name of the destination Data Source, adjust things like Engine Type, Sorting Keys... and you're done.

For a detailed explanation of Materialized Views in the UI, chech this guide.

Creating Materialized Views in the CLI

If you are an experienced user you may know how to use the `-State` and `-Merge` combinators in the `.pipe` files, the correct Engine and Aggregation types for each column in the `.datasource` file, but if you need help writing them, the CLI can for sure help there.

Check this section for more details.

Querying Materialized Views

Here it would depend on the type of **Engine** you have in your Data Source.

If it is a regular `MergeTree` because you are not doing aggregations then you're good querying like a normal Data Source.

But, if the engine is `AggregatingMergeTree` ---or `SummingMergeTree` or other special engine---, and you have functions in your pipe with the `-State` **you should use** the `-Merge` modifier, or `max()` and **group by the Sorting Key columns**. More details about this in the Master Materialized Views guide.

For Deduplication use cases with `ReplacingMergeTree`, check Deduplication Strategies guide.

What shouldn't I use Materialized Views for?

There is a bit of a mind shift when using Materialized Views if you come from the batch and scheduled world. When you create a Materialized View, the reference to the original table in your query will only see the small block of data that will be processed each time. That means that the block of data, while being processed, will not be able to see the rest of the rows in the table, so things like window functions, order by... will not work as expected.

Serverless ClickHouse

Spain
Calle del Dr. Fourquet, 27
28012 Madrid

USA
41 East 11th Street 11th floor
New York, NY 10003