tinybird

Common use cases

To get the most value from reading about common CLI use cases, we recommend reading the CLI introduction docs first. You can also use the integrated help while you work on the CLI.

Download pipes and data sources from your account

There are two ways you can start working with the CLI. You can either start a new data project from scratch, or if you already have some data and endpoints in your Tinybird account, pull it to your local disk to continue working from there.

For this second option, use the `--match` flag to filter pipes or data sources containing the string passed as parameter.

For instance, to pull all the files named `project`:

PULL ALL THE PROJECT FILES

```
tb pull --match project
[D] writing project.datasource(demo)
[D] writing project_geoindex.datasource(demo)
[D] writing project_geoindex_pipe.pipe(demo)
[D] writing project_agg.pipe(demo)
[D] writing project_agg_API_endpoint_request_log_pipe_3379.pipe(demo)
[D] writing project_exploration.pipe(demo)
[D] writing project_moving_avg.pipe(demo)
```

The pull command does not preserve the directory structure, so all your data files are downloaded to your current directory.

Once the files are pulled, you can `diff` or `push` the changes to your source control repository and continue working from the command line.

> When you pull data sources or pipes, your data is not downloaded, just the data source schemas and pipes definition, so they can be replicated easily.

Push the entire data project

PUSH THE WHOLE PROJECT

```
tb push --push-deps
```

Push a pipe with all its dependencies

PUSH DEPENDENCIES

```
tb push pipes/mypipe.pipe --push-deps
```

Adding a new column to a data source

Data Source schemas are mostly immutable, but you have the possibility to append new columns at the end of an existing Data Source with an Engine from the MergeTree Family or Null Engine. If you want to change columns, add columns in other positions, or modify the engine, you must first create a new version of the Data Source with the modified schema. Then ingest the data and finally point the Pipes to this new endpoint. To force a Pipe replacement use the `--force` flag when pushing it.

Append new columns to an existing Data Source

As an example, imagine you have the following Data Source defined, and it has been already pushed to Tinybird:

APPENDING A NEW COLUMN TO A DATA SOURCE

```
SCHEMA >
    `test` Int16,
    `local_date` Date,
    `test3` Int64
```

If you want to append a new column, you must change the `*.datasource` file to add the new column `new_column`. You can append as many columns as you need at the same time:

APPENDING A NEW COLUMN TO A DATA SOURCE

```
SCHEMA >
    `test` Int16,
    `local_date` Date,
```

```
    `test3` Int64,
    `new_column` Int64
```

> Remember that the only kind of alter column operation supported is **appending new columns to an existing Data Source**, and that the engine of that Data Source must be of the **MergeTree** family.

After appending the new column, execute `tb push my_datasource.datasource --force` and confirm the addition of the column(s). The `--force` parameter is required for this kind of operation.

Existing imports will continue working once the new columns are added, even if those imports don't carry values for the added columns. In those cases, the new columns contain empty values like `0` for numeric values or `''` for Strings, or if defined, the default values in the schema.

Create a new version of the Data Source to make additional add/change column operations

To create a new version of a Data Source, create a separate Datafile with a different name. You can choose a helpful naming convention such as adding a `_version` suffix (e.g. `my_ds_1.datasource`).

How to create materialized views

Materialized Views allow data to be transformed from an `origin` data source to a `destination` data source. There are several scenarios where Materialized Views are really useful and can make a difference in the response times of your analyses, for instance:

- Denormalize several normalized tables into one via a `JOIN`.
- Transform data using an optimized `ENGINE` for a concrete analysis.
- Transform your source data on the fly as you ingest data in your origin data source.

It's important to understand that Materialized Views are **live views** of your origin data source. Any time you `append` or `replace` data to your origin data source, all the destination data sources crated as Materialized Views are properly synced. It means you don't have to worry about costly re-sync processes.

Let's say you have an `origin` data source (`my_origin.datasource`) like this one:

```
ORIGIN DATA SOURCE
```

```
SCHEMA >
    `id` Int16,
    `local_date` Date,
    `name` String,
    `count` Int64
```

And you need an optimized version of this data source that pre-aggregates the `count` for each ID. You should create a new data source that uses a `SimpleAggregateFunction`, which will be a Materialized Views.

First define the `destination` data source (`my_destination.datasource`):

```
DESTINATION DATA SOURCE


SCHEMA >
    `id` Int16,
    `local_date` Date,
    `name` String,
    `total_count` SimpleAggregateFunction(sum, UInt64)

ENGINE "AggregatingMergeTree"
ENGINE_PARTITION_KEY "toYYYYMM(local_date)"
ENGINE_SORTING_KEY "local_date,id"
```

And then you'll write a transformation pipe (`my_transformation.pipe`) like this:

```
TRANFORMATION PIPE


NODE transformation_node

SQL >
    SELECT
        id,
        local_date,
        name,
        sum(count) as total_count
    FROM
        my_origin
    GROUP BY
        id,
```

```
        local_date,
        name
```

Once you have the origin and destination data sources defined and the transformation Pipe you can push them:

PUSH THE MATERIALIZED VIEW

```
tb push my_origin.datasource
tb push my_destination.datasource
tb push my_transformation.pipe --populate
```

Any time you ingest data into `my_origin`, the data in `my_destination` is automatically updated.

Guided process: tb materialize

Alternatively you can use the `tb materialize` command to generate the target `.datasource` file needed to push a new Materialized View.

The goal of the command is to guide you through all the needed steps to create a Materialized View. Given a Pipe, `tb materialize`:

- Asks you which Node of the Pipe you want to materialize. By default, it selects the last one in the Pipe. If there's only one, it's automatically selected, skipping asking you. From the selected query, the commands guesses the best parameters for the following steps.
- It warns you of the errors the query has, if any, that prevents it from materializing. If everything is correct, it continues.
- It creates the target Data Source file that will receive the results of the materialization, setting default engine parameters. If you are materializing an aggregation you should make sure the `ENGINE_SORTING_KEY` columns in the `.datasource` file are in the right order you are going to filter the table.
- It modifies the query to set up the materialization settings and pushes the Pipe to create the materialization. You can skip the pipe checks if needed as well.
- It asks you if you want to populate the Materialized View with existing data. If you select to populate, it will ask you if you want to use a subset of the data (to populate faster and check if everything is correct) or fully populate with all existing data.
- It creates a backup file of the Pipe (adding the `_bak` suffix to the file extension). We do this because the command modifies the file. It completes the aggregate functions with `-State` combinators (see docs) when corresponding and adds the target Data Source name. We keep the backup file in case you want to recover the original query.

The command generates and modifies the files involved in the materialization. If you run into an error or you need to modify something in the materialization, you will have already the files, that can serve you as a better starting point than creating everything on your own. We recommend you to double check the generated datafiles.

For the same case as above:

Given the same `origin` datasource (`my_origin.datasource`) we need only to write the query that will materialize:

TRANFORMATION PIPE

```
NODE transformation_node

SQL >
    SELECT
        id,
        local_date,
        name,
        sum(count) as total_count
    FROM
        my_origin
    GROUP BY
        id,
        local_date,
        name
```

See that this time we don't need to define and create first the destination Data Source, only the query.

We then run `tb materialize`. As you can see, the command will guide through all the process.

GENERATE FILES TO PUSH A MATERIALIZED VIEW

```
❯ tb materialize pipes/my_transformation.pipe mv_transformation

This feature is under development and released as a beta version. You can re
** Pushing the pipe my_transformation to your Workspace to analyze it
** Running my_transformation
** 'my_transformation' created
** Analyzing the pipe my_transformation
** Created backup file with name my_transformation.pipe_bak
** => Saved local file mv_transformation.datasource
Delete the Data Source from the Workspace and push mv_transformation.datasou
```

```
** Running mv_transformation
** 'mv_transformation' created
** => Saved local file my_transformation.pipe
[1] Push my_transformation.pipe and override if it exists
```

After the command is successfully run, you'll end up with: - A new datasource and its corresponding `.datasource` file: `mv_transformation` - A modified pipe `my_transformation.pipe` file, adapted to materialize from the chosen query. - The materialized view is properly created in the workspace, next time you insert data in `my_origin` the `transformation_node` SQL will be triggered, so data will be aggregated on the fly in the `mv_transformation` Materialized View.

How to force populate materialized views

Sometimes you want to force populating a materialized view, most likely because you changed the transformation in the pipe and you want the data from the origin data source to be re-ingested.

POPULATE A MATERIALIZED VIEW

```
tb push my_may_view_pipe.pipe --force --populate
```

You'll get as a response a Jobs API `job_url` so you can check its progress and status.

Debug mode

Particularly when you work with Pipes that make use of several versions of different data sources, you might need to double check which version of which data source the Pipe is pointing at before you push it to your Tinybird account.

To do so, use the `--dry-run --debug` flags like this:

DEBUG MODE

```
tb push my_pipe.pipe --dry-run --debug
```

Once you've validated the content of the Pipe, push your Pipe as normal.

Automatic regression tests for your API endpoints

Any time you `--force` push a Pipe which has a public endpoint that has received requests, some automatic regression tests are executed.

If the previous version of the endpoint returns the same data as the version you are pushing, the CLI checks for the top ten requests. This can help you validate whether you are introducing a regression in your API.

Other times, you are consciously `--force` pushing a new version which returns different data. In that case you can avoid the regression tests with the `--no-check` flag:

AVOID REGRESSION TESTS

```
tb push my_may_view_pipe.pipe --force --no-check
```

When pushing a Pipe with a public endpoint, the endpoint will be maintained based on the node name. If the existing endpoint node is renamed, the last node of the Pipe will be recreated as an endpoint. The latter option is not an atomic operation: the endpoint will be down for a few instants while the new endpoint is created.

Integrations

- Amazon S3
- Kafka Data Streams
- Google Cloud Storage
- Google BigQuery
- Snowflake
- Confluent

Use cases

- In-Product Analytics
- Operational Intelligence
- Realtime Personalization
- Anomaly Detection & Alerts
- Usage Based Pricing
- Sports Betting/Gaming
- Smart Inventory Management
- Serverless ClickHouse

Spain
Calle del Dr. Fourquet, 27
28012 Madrid

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