7 DuckDB in the Cloud with MotherDuck

This chapter covers

- The idea behind MotherDuck
- Understanding how the architecture works under the hood
- Use cases for serverless SQL analytics
- Creating, managing, and sharing MotherDuck databases
- Tips for optimizing your MotherDuck usage

Up to this point, our focus has been on leveraging DuckDB's capabilities for querying datasets—whether they're stored locally or remotely—directly from our own computers. While this approach addresses a broad range of needs, there are specific scenarios where a remote database server offers additional advantages.

Enter MotherDuck: a solution for enhancing SQL analytics through a simplified scale-up strategy. In this chapter we will learn how MotherDuck enables hybrid query execution, making use of a remotely hosted DuckDB alongside one operating on our own machine.

7.1 Introduction to MotherDuck

MotherDuck is a collaborative serverless analytics platform that lets you query and analyze data in cloud databases and from cloud storage using your browser or any of the DuckDB APIs. *Serverless* in this context means that you as a user won't have to deal with spinning up servers, clusters, or configuring database instances. Instead, you just can create a database and the service will take care of the rest for you.

A closed beta of the platform was launched in June 2023, with general availability introduced in September 2023. MotherDuck works closely with the DuckDB Labs team to ensure the best interoperability and availability of all features in the cloud platform.

You can find the documentation for the MotherDuck service at motherduck.com/docs.

7.1.1 How it works

There are several ways to use MotherDuck. When you sign up to the service, you'll end up in the MotherDuck web UI running in the browser. This UI is running a special DuckDB version in the browser that knows how to communicate with MotherDuck. The UI is both a tool to manage your MotherDuck databases and a notebook-based approach to enter and execute queries and to view their results. We will cover this in Using MotherDuck through the UI.

The other entry points to MotherDuck are of course the CLI and the integrations with languages such as Python. MotherDuck is presented as an opt-in feature to the open-source database DuckDB via an extension. This extension will be automatically loaded when you open a database using the `md:` or `motherduck:` protocol, and will integrate both with the query parser and engine. The parser is enhanced with functionality around database and share management. In the query engine, the extension analyzes if tables are available locally or remotely and then uses the appropriate execution engine and joins the data accordingly. If needed, parts of the local data are sent to the server for joins or filters, or data from the remote side is fetched and joined locally.

MotherDuck's architecture is shown in the figure 7.1 and core to its operation are the following components:

- Service Layer (Sharing, Admin, Services, Monitoring)
- Ducklings (Serverless DuckDB compute instances)
- Catalog (Database and Tables)
- Storage (Internal Storage and Maintenance)

The service layer of MotherDuck provides capabilities like secure identity, authorization, administration, monitoring, billing, and so on.

Where the serverless DuckDB "Duckling" instances execute the "remote" parts of your query, the catalog exposes the databases, tables and views that are managed in the storage layer. The storage is durable, secure, and automatically optimized for best performance. MotherDuck—as with other modern cloud data platforms—separates storage and compute facilities, which will eventually be important for the cost that occurs when using MotherDuck.

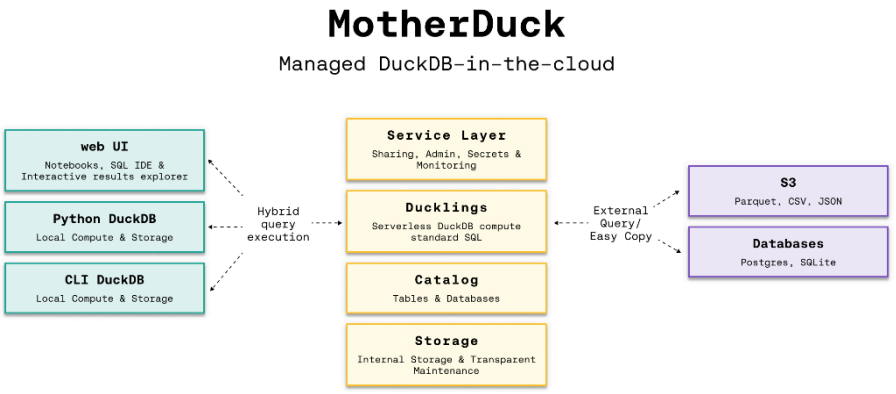


Figure 7.1 Hybrid query execution as implemented within MotherDuck

MotherDuck is designed so that you can focus on your queries, not on the size of the machines you need to spin up in the cloud to make the queries run fast. The separation of the storage- and the compute-layers will have some impact on the cost, and we will discuss this later in section Making the best possible use of MotherDuck.

7.1.2 Why use MotherDuck

First and foremost MotherDuck provides a simplified, performant, and efficient data warehouse based on DuckDB. Most folks are in the long tail of cloud data warehouse users, and don’t need analytics processing capabilities for tens or hundreds of terabytes of hot data. Those use cases can benefit from a simpler, efficient architecture that is not based on a distributed system, but instead uses the cloud as the main data storage and DuckDB as the query engine. Think back to the fictional system in chapters 3 and 4, monitoring energy production. Even if you monitored several hundreds or thousands of sites and their daily output in quarterly hour measurements, it would not be close to a billion records per year and most likely only a few gigabytes in size. This is hardly considered big data and something MotherDuck would be able to deal with easily.

Another way to utilize MotherDuck is using it as a query engine for data lakes made up of heterogeneous sources, such as cold data stored as Parquet or CSV files in S3 or data stored in Apache Iceberg. You can easily join that data with hot data, stored directly in MotherDuck.

MotherDuck can serve as a serverless backend for data applications, dashboards, and APIs. Instead of running analytics queries on the main transactional database, you can use MotherDuck to run those queries on a dedicated analytics database.

Last but not least, MotherDuck allows sharing read-only snapshots of your databases to other MotherDuck users. They can use your sources as they are shared or join data from their instance together with your datasets as well.

Independent of what you are planning to do with MotherDuck, you will need to sign up with their service in order to use their cloud offering, which is what we will do in the next section.

7.2 Getting started with MotherDuck

Throughout the book we did use the CLI or the integration with Python. To get started with MotherDuck, you will need bring up your browser of choice first. Go to motherduck.com and click on the `Sign Up` button. You can create a free account with your GitHub account, Google account, or by providing an email address.

Once you've done that, you'll find yourself in the MotherDuck UI:

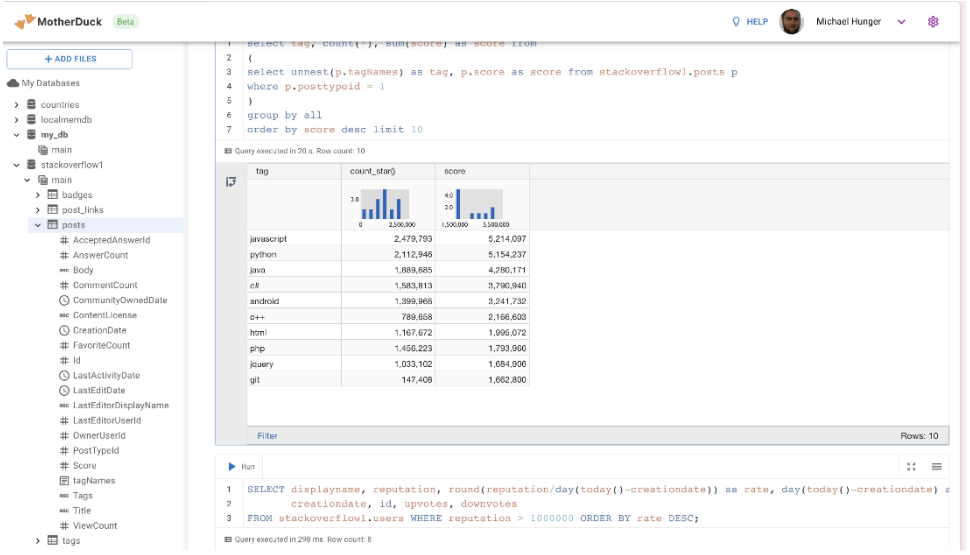


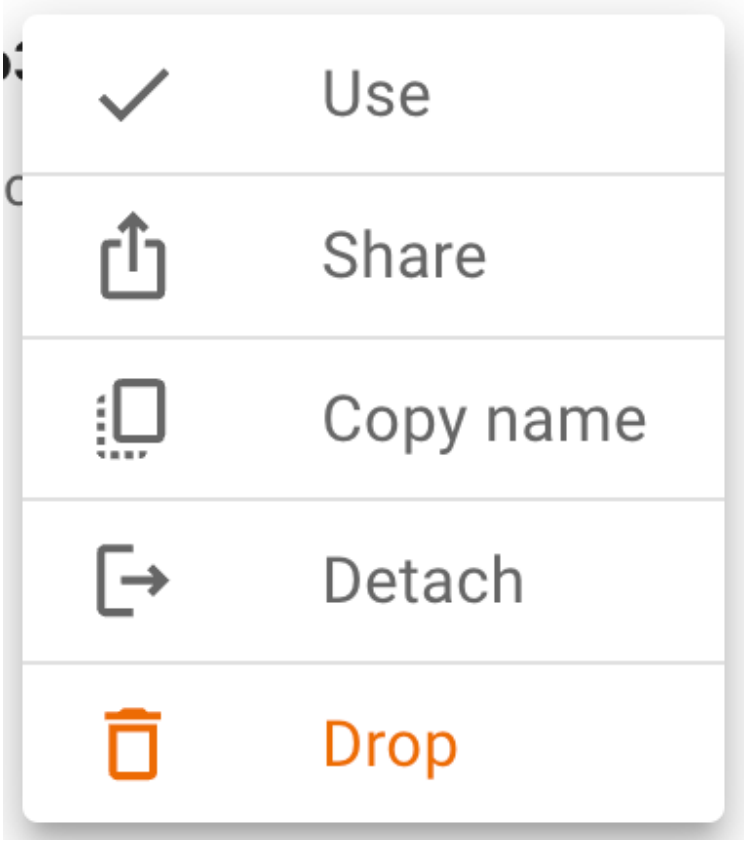
Figure 7.2 MotherDuck UI

7.2.1 Using MotherDuck through the UI

The web-based MotherDuck UI at app.motherduck.com provides a central place for accessing and querying all your remote databases, managing your account settings and storing any secrets necessary for querying remote data sources on S3. It also gives access to your MotherDuck API token, which is needed to access MotherDuck outside the UI.

TIP The database running in the web UI is actually a local, embedded database, too! It is using a version of Duck that is compiled to Web Assembly (WASM), thus running locally in your browser. The queries are executed of course in the MotherDuck cloud, as explained earlier. If you don't have or want a MotherDuck account, you can also use shell.duckdb.org, which behaves essentially like your CLI, but without being able to persist data or connect to MotherDuck.

Results of your SQL queries are cached in the DuckDB instance local to your browser, enabling you to instantly sort, pivot, and filter query results! The UI lists database, their tables with their columns, and uploaded files on the left side. With a context menu you can use, share, drop, detach databases, or copy their name.



For running queries there is a "Jupyter Notebook"-like experience, that allows you to write SQL statements with auto-complete, and then run them and see query results rendered in a data grids below the cell. The output data grids support local sorting, select output columns, show histograms and aggregations in the column header, and allow you to pivot and filter the data. The state of the UI with your queries is kept across sessions, so you can close the browser and continue from where you left off at a later date.

The web-based UI is a great work environment for already existing databases, shared databases and applying the lessons learned from chapter 5. There is support for uploading CSV and Parquet files directly from the UI. They will be accessible in any query, and you utilize a "CREATE table AS SELECT" statement or transform them to your needs.

For importing existing databases from the DuckDB CLI, or using MotherDuck from any of the supported language bindings, you'll need to authenticate the CLI or the language binding of choice with MotherDuck.

7.2.2 Connecting to MotherDuck with DuckDB via token based authentication

Please make sure you have MotherDuck account and are logged in before proceeding. The DuckDB triggers the authentication process with MotherDuck at the moment you try to open a shared database instance. This can be a named database or the default one.

> **TIP** When you connect to MotherDuck without specifying a database, you connect to a default database called `my_db`. This is the current database. You can then query any table in this database by just specifying the table name. The `USE` command allows you to switch the current database.

Let's open the default database at MotherDuck for your account by running `.open md:` in the CLI. Unless you're already authenticated it will prompt you with the following message:

```
Attempting to automatically open the SSO authorization page in your
➥ default browser.
1. Please open this link to login into your account:
➥ https://auth.motherduck.com/activate
2. Enter the following code: XXXX-XXXX
```

Your browser will have opened with a device confirmation message similar to that shown in 7.3.

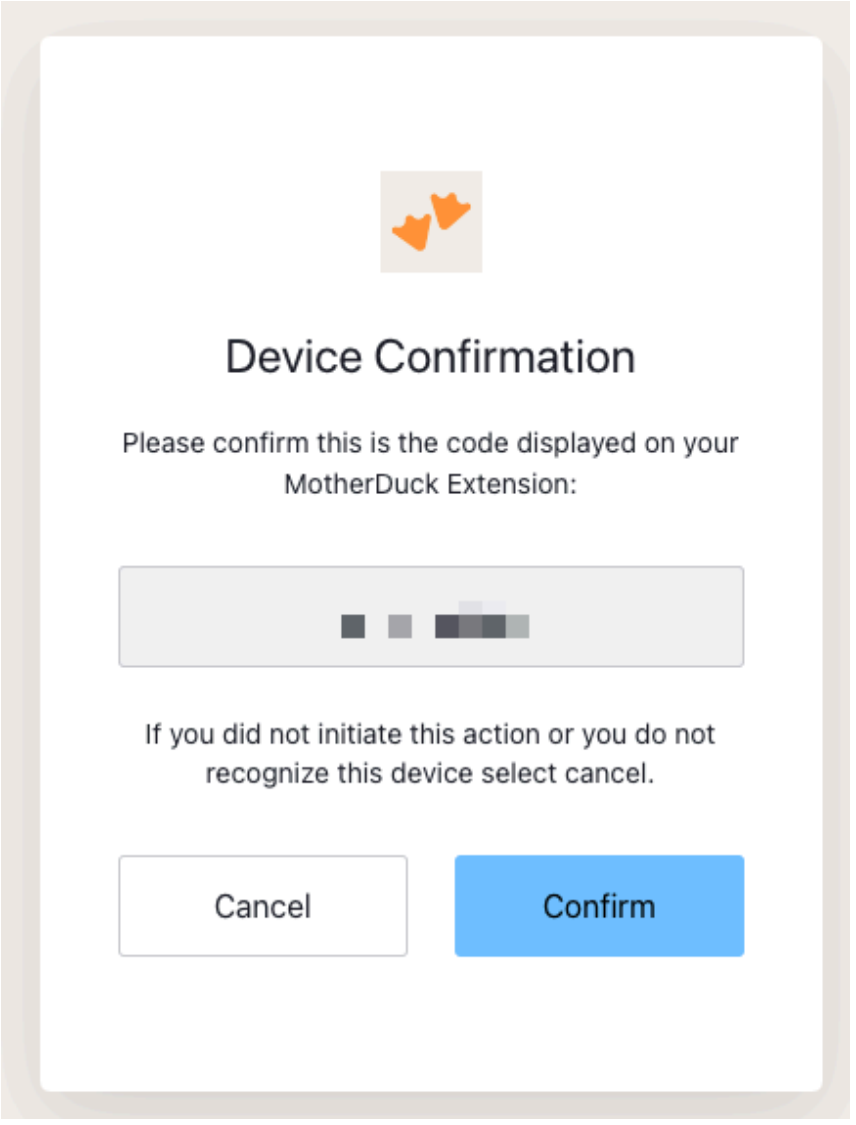


Figure 7.3 MotherDuck Confirmation Message

If you follow through the auth flow to completion, you'll see the following lines in the terminal:

```
Token successfully retrieved [✓]
You can store it as an environment variable to avoid having to log in again:
  $ export motherduck_token='eyJhbGciOiJI..._Jfo'
```

You're now able to access databases on MotherDuck and, as the message says, if you want to be automatically logged in during future sessions, you should add the MotherDuck token as an environment variable. Alternatively, you can use the token as a parameter to the `md:` protocol like this:

```
D .open 'md:?motherduck_token=eyJhbGciOiJI..._Jfo'
```

This URL format is also applicable to the individual language bindings. Think back to chapter 6, in which we discussed connection management from within the integration with the Python ecosystem. To open a connection to MotherDuck from Python, you would use

```
import duckdb

con = duckdb.connect('md:?motherduck_token=eyJhbGciOiJI..._Jfo')
```

Next up we will discuss how to perform various tasks in MotherDuck. It's up to you whether you use the web UI or the CLI, but please make sure you set up your MotherDuck account and can access it with either the UI or the CLI.

7.3 Making the best possible use of MotherDuck

In this section you will learn about the features added to DuckDB via the MotherDuck extension that let you interact with MotherDuck and use it to its fullest potential. We start with discussing all features helping you to get your data into the cloud and potentially sharing your databases with colleagues and partners:

- Uploading databases from your local machine to the cloud
- Managing databases (creation, deletion, listing)
- Sharing databases via URL, refreshing the shares, and attaching them to your local DuckDB instance
- Accessing data from S3 buckets

After that, we will discuss how to control whether a query runs completely on a remote database, completely local, or partially remote and local.

Last but not least, MotherDuck ships a couple of AI related features, such as functions that will automatically describe your schema and based on that, generate or fix SQL statements for you. Let's have a look at these features, starting with uploading a local DuckDB database.

7.3.1 Uploading Databases to MotherDuck

Think back to chapter 6, in which we used the Python integration to build up a database containing data about countries. We finished our work there, cleaned up issues in the dataset, and want to share that work now with colleagues. One way to do this is the spreadsheet way of life: Just take the 'countries.duckdb' store, attach it to an email or copy it to a network folder and move on. A better, less fragile way of sharing is MotherDuck. To begin, start your DuckDB CLI and open your database with the following command. If you don't have that store, don't worry, the command will create it for you if it doesn't exist.

```
.open countries.duckdb
```

Let's add a table `cities` to that database, especially if you created a fresh database:

```
CREATE TABLE cities AS
SELECT *
FROM (VALUES ('Amsterdam', 1), ('London', 2)) cities(Name, Id);
```

Before we share the database, we need to detach it first, so that all the locks will be released.

This can either be done by switching back to an in-memory database, using `.open` without arguments followed by a `LOAD motherduck;` to load the extension, or in one go, by calling `.open md:`. The latter will attach the CLI to the MotherDuck default database. We can then create the remote database with the `CREATE DATABASE` statement as shown below:

```
.open md:
CREATE DATABASE "countries" FROM 'countries.duckdb';
```

Depending on the size of your database and the speed of your internet connection, the upload process can take some time. In our experiments the upload of a 16GB database took about 40 minutes from a regular laptop with a home internet connection (40MBit/s upstream). At the time of writing, this is a topic MotherDuck is actively working on and the upload performance will be improved. For now, it may be faster to export your database into Parquet files, upload those to cloud storage, and then create a database in MotherDuck from those files.

Once the database is uploaded we can check that we are able to use it. Your session is still attached to the default database in MotherDuck. That means, you'll need to either prefix your tables with the database name or switch to the new database. `FROM countries.cities` uses the prefix. In the snippet below, we switch to the database first by issuing a `USE` statement. That allows us to omit the prefix to any table in that database, as we did so in all previous examples:

```
USE countries; -- #1
FROM cities;
```

#1 The `USE` statement will look up the database name in the remote MotherDuck catalog

In case you left the CLI in between, you can connect directly to the new database via `.open md:countries`, too. Either way, we can then run a query to check that the data is there. So if we select everything from the `cities` table with `FROM cities`, we should see everything that we just created:

```
┌────────────┬────────┐
│   Name     │   Id   │
│  varchar   │ int32  │
├────────────┼────────┤
│ Amsterdam  │      1 │
│ London     │      2 │
└────────────┴────────┘
```

There are some things to keep in mind when uploading a local database to MotherDuck: **The local and remote name must be different**, otherwise you might get the error below.

```
create database "countries" from 'countries' ;
Error: Catalog Error: Database 'countries' has already been
➥ created and attached
```

If you try to upload your current database, you get the following error.

```
Error: Binder Error: Database "countries.duckdb" is already attached
➥ with alias "countries"
```

We avoided this in our example above by calling `.open md:` which did two things for us: loading the MotherDuck extension and connecting to the default MotherDuck database. Another option is using `.open`, which will switch to an in-memory database.

7.3.2 Creating databases in MotherDuck

In the previous section we uploaded an existing database to MotherDuck. Alternatively you can start building your schema from scratch directly in the cloud. If the content of your schema depends largely on files stored in another public cloud, it would be a waste of time and resources to download them first into a local DuckDB instance and then upload that database to MotherDuck. MotherDuck runs most likely closer to any S3 bucket than your local system, so an ingress from the cloud directly into MotherDuck may save you time and money.

You create a new database in MotherDuck using the `CREATE DATABASE` command. The database name can't have any special characters, only alphanumeric characters and underscores are allowed.

To create a database called `my-test`, we can run the following:

```
CREATE DATABASE "my-test";
```

We can check that the database has been created with the `SHOW DATABASES` command:

```
SHOW DATABASES;
```

```
┌──────────────┐
│ database_name │
│    varchar    │
├──────────────┤
│ my-test       │
└──────────────┘
```

Alternatively, we can use the `.databases` CLI command. We can run either of these commands from our local DuckDB CLI or we can navigate to the MotherDuck UI and run them there.

The `CREATE DATABASE` statement just creates the database, it doesn't change your session to it, so we need to run `USE 'my-test';` first. We can then check that we're connected to this database using the `current_database` function:

```
SELECT current_database();
```

If everything worked out well, you should see the correct database name:

```
┌────────────────────┐
│ current_database() │
│      varchar       │
├────────────────────┤
│ my-test            │
└────────────────────┘
```

Let's now create a table `people` in our MotherDuck database and add a couple of rows:

```
CREATE TABLE people (name varchar, born date);
INSERT INTO people VALUES ('Mark', '1989-01-01'), ('Hannes', '1990-01-01');
```

We can then return a count of the records in the `people` table by running the following query:

```
SELECT count(*)
FROM people;
```

```
┌──────────────┐
│ count_star() │
│    int64     │
├──────────────┤
│            2 │
└──────────────┘
```

We can also run that same query with the database name prefixed to the table name:

```
SELECT count(*)
FROM "my-test".people;
```

And once we're happy that we've got the hang of how MotherDuck works, we can remove that test database. To do that, we'll need to first switch to the default `my_db` database before running the `DROP DATABASE` command.

```
USE my_db;
DROP DATABASE "my-test";
```

The test database `my-test` should no longer appear in the output of `SHOW DATABASES;`.

7.3.3 Sharing Databases

MotherDuck offers the ability to share a read-only snapshot of your databases. This is great not only for sharing the data, but for collaborative analysis and shared functionality. The snapshot will not only contain your data, but all views and with them, all the effort you put into them during creation. Think of it as a spreadsheet on steroids.

To make your data available to others, you can use the `CREATE SHARE` statement. If you run it, you will get a shareable link, that others can connect to using the `ATTACH` command.

TIP The generated links below will be different if you create the shares for the country database based on your MotherDuck account and your database.

Assuming that you followed our initial instructions to create the `countries` database, you could share it like this:

```
CREATE SHARE shared_countries -- #1
FROM countries; -- #2
```

#1 The name of the shared database
#2 The name of the database to share

You will receive a link similar to the following

```
┌──────────────────────────────────────────────────────┐
│                      share_url                       │
│                       varchar                        │
├──────────────────────────────────────────────────────┤
│ md:_share/countries/1acb80cf-d872-4fab-8077-64975cce0452 │
└──────────────────────────────────────────────────────┘
```

The time to create a shared database currently depends on the size of the source database. A 16GB database took about a minute to share.

To attach this database, our friend or colleague will need to have a MotherDuck account, too. They can then run the following command on their DuckDB database:

```
ATTACH 'md:_share/countries/1acb80cf-d872-4fab-8077-64975cce0452'
AS shared_countries;
```

We can describe the contents of a shared database using the `DESCRIBE SHARE` command:

```
.mode line
DESCRIBE SHARE shared_countries;
```

Which results in this output, mirroring the source link and the original name as well as the ids of the database and the latest snapshot:

```
share_name = shared_countries
   share_link = md:_share/countries/1acb80cf-d872-4fab-8077-64975cce0452
database_name = countries
  database_id = 9d7586ac-add9-46dc-a4fb-def6b42f0f7c
  snapshot_id = 041a5ba9-8cf4-471b-af13-1bec75a0b3ce
```

We can also list the shares that we've created:

```
LIST SHARES;
```

At the time of publication, shares are not automatically updated when you change the content of your shared database. Changes to both schema and data must be explicitly propagated through the `UPDATE SHARE` statement from the sharing site:

```
UPDATE SHARE shared_countries;
```

To get rid of any share you are not interested in anymore, run the `DETACH` statement. If you are connected to the share, you need to switch to a different database first:

```
USE my_db;
DETACH shared_countries;
```

We did actually prepare a couple of hopefully interesting shares for you, including the complete database of chapters 3 and 4, containing all tables and views we created throughout the chapters. Get it with:

```
ATTACH
  'md:_share/duckdb_in_action_ch3_4/d0c08584-1d33-491c-8db7-cf9c6910eceb'
AS duckdb_book_ch3_and_4;
USE duckdb_book_ch3_and_4;
SHOW tables;
```

We also ingested a complete dump from StackOverflow, which provides a great dataset to play along with:

```
ATTACH
  'md:_share/stackoverflow/6c318917-6888-425a-bea1-5860c29947e5'
AS stackoverflow_analysis;
USE stackoverflow_analysis;
SELECT count(*) FROM posts;
```

The share contains a whooping number of 58329356 posts and answers. An interesting corpus of data to play along with.

We collected some other interesting shares that you can load into a `sample_data` database via:

```
ATTACH 'md:_share/share_sample_data/23b0d623-1361-421d-ae77-62d701d471e6'
AS sample_data;
```

The tables are prefixed, and you can access them via their fully qualified names e.g. `sample_data.nyc.yellow_cab_nyc_2022_11`:

name	table name	rows	description
Hacker News	`hn.hacker_news`	3.9M	Sample of comments from Hacker News
NYC 311 Complaint Data	`nyc.service_requests`	32.5M	Requests to NYC's 311 complaint hotline via phone and web
Air Quality	`who.ambient_air_quality`	41k	Historical air quality data from the World Health Organization.
Taxi Rides	`nyc.taxi`	3.3M	NYC Yellow Cab Trips data from November 2020
Rideshare	`nyc.rideshare`	18.1M	Ride share trips (Lyft, Uber etc) in NYC

These datasets provide many different topics which you can use to train your SQL skills or explore the possibilities of DuckDB and MotherDuck.

7.3.4 Managing S3 secrets and loading Data from S3 Buckets

In previous chapters we ingested data from files directly available over `http://` or `https://`. Oftentimes people use Amazon S3 for storing files, accessible via the `s3://` protocol.

MotherDuck—and DuckDB—speak that protocol as well, but they need your secrets to authenticate on your behalf against Amazon S3. When using MotherDuck you can store your secrets in their systems, so that they are available in all sessions that are connected to MotherDuck. You either do this via the web UI or a dedicated statement, the `CREATE OR REPLACE SECRET` statement:

```
CREATE OR REPLACE SECRET (
  TYPE S3,
  KEY_ID 'access-key',
  SECRET 'secret-key',
  REGION 'us-east-1'
);
```

Once you've done that, you can query data in a secure S3 bucket just like any CSV, Parquet file we queried over `http` or the file system before:

```
CREATE OR REPLACE TABLE mytable AS
FROM 's3://...';
```

Once you've finished working with the bucket, you'll want to remove the secret.

```
DROP SECRET (TYPE s3);
```

There are a couple of things to keep in mind when creating a secret:

- With DuckDB prior to 0.10.0 you can only have one `SECRET` object
- You can only use permanent S3 secrets—temporary S3 secrets, that are only valid during a session, are currently not supported

7.3.5 Controlling from where data is ingested and optimizing MotherDuck usage

The costs of running a cloud based solution has multiple dimensions, such as compute, storage, and data ingress and egress. The MotherDuck extensions gives you close control over where a function is to be executed: In the cloud or on your local machine. If you want to process a large Parquet file that you already have on your local machine, it's pointless uploading it to S3 first and then running the processing in MotherDuck, as you will pay for both the computational costs at MotherDuck and the egress cost at S3. A fast internet connection given, you are most likely better off doing this locally and just have the upload from your machine to MotherDuck. The extension enhances all functions starting with the `read_` prefix—such as `read_json` and `read_csv_auto`—to support the `MD_RUN` parameter. That parameter lets you control where the function will be executed, and it supports the following values:

- `MD_RUN=LOCAL` executes the function in your local DuckDB environment
- `MD_RUN=REMOTE` executes the function in MotherDuck-hosted DuckDB runtimes in the cloud
- `MD_RUN=AUTO` executes remotely all `s3://`, `http://`, and `https://` requests, except those to `localhost/127.0.0.1`. This is the default option.

For example, if you wanted to query the DuckDB IPs dataset on the MotherDuck-hosted DuckDB runtime, you could execute the following query. In the example we use `.timer on` to get the query execution time:

```
.timer on
SELECT count(*)
FROM read_csv_auto(
  'https://github.com/duckdb/duckdb/raw/main/data/csv/ips.csv.gz',
  MD_RUN=REMOTE
);
```

As of writing the query took less than a second when running in the MotherDuck cloud. Using `MD_RUN=LOCAL` instead, it took close to two seconds on the author's slow internet connection.

MotherDuck's pricing beyond the free tier will be a base price of $25 for 100GB storage and 100 compute hours, with exceeding usage costing for cold storage ($0.08 per GB per month) and for query execution ($0.40 per hour). But this is only for cloud usage, not your local execution, see https://motherduck.com/pricing/.

Cold storage is the persistent storage for your databases and files.

Hot storage is used to execute your queries, equivalent to memory usage, and it can be limited to a maximum value, fine-tuned to your needs. The hot storage is automatically scaled up to the upper limit and metered per second and Gigabyte.

The separation actually makes a lot of sense, as studies show that a huge percentage of the data that gets processed is less than 24 hours old. By the time data gets to be a week old, it is probably 20 times less likely to be queried than from the most recent day. Additionally, the workload sizes are often smaller than overall data sizes. For example dashboards are usually built from aggregated data: Any data from the last hour can be freshly aggregated, so that you won't miss out on the latest changes. Anything reaching back later than a week can be stored pre-aggregated as an additional table. The aggregate will usually have way less rows and consume a lot less storage.

If you want to keep your costs in control make sure to not store superfluous data directly in MotherDuck but load/process it as needed, and make sure to set a maximum for the hot storage. The amount of hot storage available does affect the performance of your queries.

7.3.6 Querying your data with AI

While SQL is somewhat close to English language, it's not always easy to master and if you're new to it, some constructs might be outright intimidating.

> **TIP** Fun fact: An earlier name of SQL was SEQUEL (**S**tructured **E**nglish **QUE**ry **L**anguage), which was a pun on QUEL, another query language based on the relational model. The name got dropped later due to trademark issues.

Queries in a structured language can however be nicely generated, and you will be delighted to hear that MotherDuck offers a generative AI feature that allows you to query your data using natural language. It is able to describe your data and will generate SQL statements for you or fix existing ones. The feature works by sending the database schema along with a detailed prompt and your question to a large language model (LLM) that then generates the requested SQL statement and optionally executes it.

From our experience it works quite well, and you can try it out on any of the test datasets. The following example uses the StackOverflow dataset, attached with the following statements:

```
ATTACH
  'md:_share/stackoverflow/6c318917-6888-425a-bea1-5860c29947e5'
AS stackoverflow_analysis;
USE stackoverflow_analysis;
```

You can get a description of the database schema by calling the `prompt_schema` procedure.

```
.mode line
CALL prompt_schema();
```

The results of running this a couple of times are as follows:

```
summary = The database contains tables for storing data related to votes,
➥ tags, posts, post links, badges, users, and comments.
Run Time (s): real 3.672 user 0.007355 sys 0.002674
```

It takes a couple of seconds to run and the result isn't all that impressive—we could have gotten the same output by looking at the table list. If call the function a second time, you most likely observe that we get a different response:

```
summary = The data in the database is about votes, tags, posts, post links,
➥ badges, users, and comments.
Run Time (s): real 3.054 user 0.007354 sys 0.003175
```

This can be initially surprising until we consider that an LLM is a probabilistic model that isn't guaranteed to return the same response each time.

Instead of using a SQL statement to query for the most popular tags, we will use plain English in the next example: What are the most popular tags? This is of course not a valid SQL statement, so we must indicate that via a special pragma called `prompt_query`. A pragma is special directive that tells a compiler or query parser how it should process its input. Here the input should be process as a prompt:

```
.mode duckbox
pragma prompt_query('What are the most popular tags?');
```

Without showing us how, we get 10 rows back and they are actually meaningful:

```
┌────────────┬─────────┐
│  TagName   │  Count  │
│  varchar   │  int64  │
├────────────┼─────────┤
│ javascript │ 2479947 │
│ python     │ 2113196 │
│ java       │ 1889767 │
│ c#         │ 1583879 │
│ php        │ 1456271 │
│ android    │ 1400026 │
│ html       │ 1167742 │
│ jquery     │ 1033113 │
│ c++        │  789699 │
│ css        │  787138 │
├────────────┴─────────┤
│ 10 rows    2 columns │
└──────────────────────┘
-- Run Time (s): real 3.763 user 0.124567 sys 0.001716
```

While that is a correct answer, we might be curious about the SQL query that was used to compute the answer. We can find out the SQL that might have used for that (keeping in mind that probabilistically it could have been slightly different) by using the `prompt_sql` procedure.

```
.mode line
call prompt_sql('What are the most popular tags?');
```

```
query = SELECT TagName, Count
FROM tags
ORDER BY Count DESC;
Run Time (s): real 5.425 user 0.010331 sys 0.005074
```

Looks good, it's even smart enough to use the table columns and ordering and limit to get "most popular" tags. The runtime for these AI prompts is between 2 and 10 seconds and the majority of the time is spent inside the Large Language Model (LLM).

Let's see how it deals with a more involved question: Which 5 posts have the most comments, what is their title and comment count?

```
.mode duckbox
pragma prompt_query("Which 5 questions have the most comments, what is the
  post title and comment count");
```

```
┌──────────────────────────────────────────────────────────┬──────────┐
│                          Title                           │ comments │
│                         varchar                          │  int64   │
├──────────────────────────────────────────────────────────┼──────────┤
│ UIImageView Frame Doesnt Reflect Constraints             │      108 │
│ Is it possible to use adb commands to click on a view by find│      102 │
│ How to create a new web character symbol recognizable by html│      100 │
│ Why isnt my CSS3 animation smooth in Google Chrome (but very │       89 │
│ Heap Gives Page Fault                                    │       89 │
└──────────────────────────────────────────────────────────┴──────────┘
Run Time (s): real 19.695 user 2.406446 sys 0.018353
```

And let's check what query was used. It is interesting that it detects or knows that all entries in the `posts` table with `PostTypeId = 1` are questions and not answers. Perhaps it knew that because it was trained on the StackOverflow dataset?

```
.mode line
call prompt_sql("Which 5 questions have the most comments, what is the
  post title and comment count");
```

```
query = SELECT p.Title, COUNT(c.Id) AS comment_count
FROM posts p
JOIN comments c ON p.Id = c.PostId AND p.PostTypeId = 1
GROUP BY p.Title
ORDER BY comment_count DESC
LIMIT 5;
Run Time (s): real 4.795 user 0.002301 sys 0.001346
```

Figure 7.4 shows what it looks like in the MotherDuck UI.

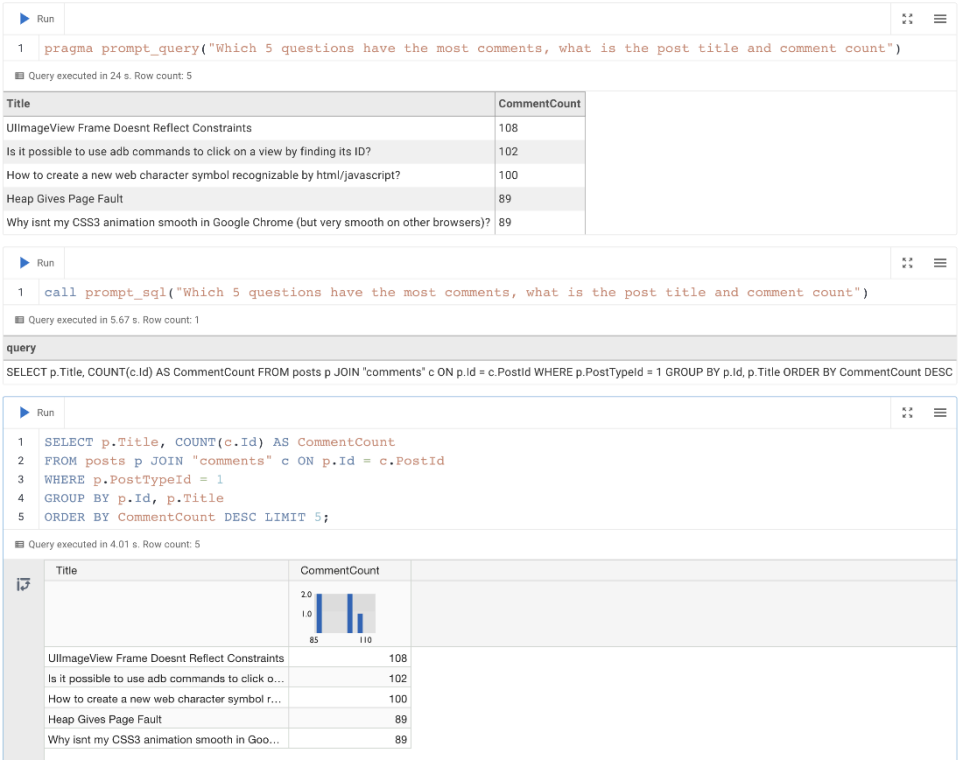


Figure 7.4 MotherDuck UI with AI Queries

Since the comment count is a column in the posts table, the join with the comments table isn't needed. Let's see if we can get it to generate a query using just the posts table by tweaking our prompt:

```
call prompt_sql("System: No joins! User: Which 5 questions have the most
  comments, what is the post title and comment count");
```

```
query = SELECT Title, CommentCount
FROM posts
WHERE PostTypeId = 1
ORDER BY CommentCount DESC
LIMIT 5;
Run Time (s): real 3.587 user 0.001733 sys 0.000865
```

Much better!

You can also use `call prompt_fixup()` to fix a SQL-code for a statement, e.g. the infamous, "I forgot GROUP BY".

```
call prompt_fixup("select postTypeId, count(*) from posts");
```

```
query = SELECT postTypeId, COUNT(*) FROM posts GROUP BY postTypeId
Run Time (s): real 12.006 user 0.004266 sys 0.002980
```

Or fixing a wrong join column name, or two.

```
call prompt_fixup("select count(*) from posts join users on
➥ posts.userId = users.userId");
```

```
query = SELECT COUNT(*) FROM posts JOIN users ON
➥ posts.OwnerUserId = users.Id
Run Time (s): real 2.378 user 0.001770 sys 0.001067
```

We think there's a lot of potential using an LLM together to generate queries for a database, especially when the model can be augmented with the database schema. While it most likely won't replace queries that are written and tuned by specialists for reports and applications, it will make database systems much more accessible to a broader audience.

In early 2024 MotherDuck introduced also "FixIt" a fast SQL error fixer for the UI which is based on the same technology and fixes SQL statements that are syntactically incorrect directly inline in the UI (https://motherduck.com/blog/introducing-fixit-ai-sql-error-fixer/).

7.3.7 Integrations

MotherDuck supports a wide variety of data transfer, business intelligence and data visualization tools, as shown in figure 7.5.

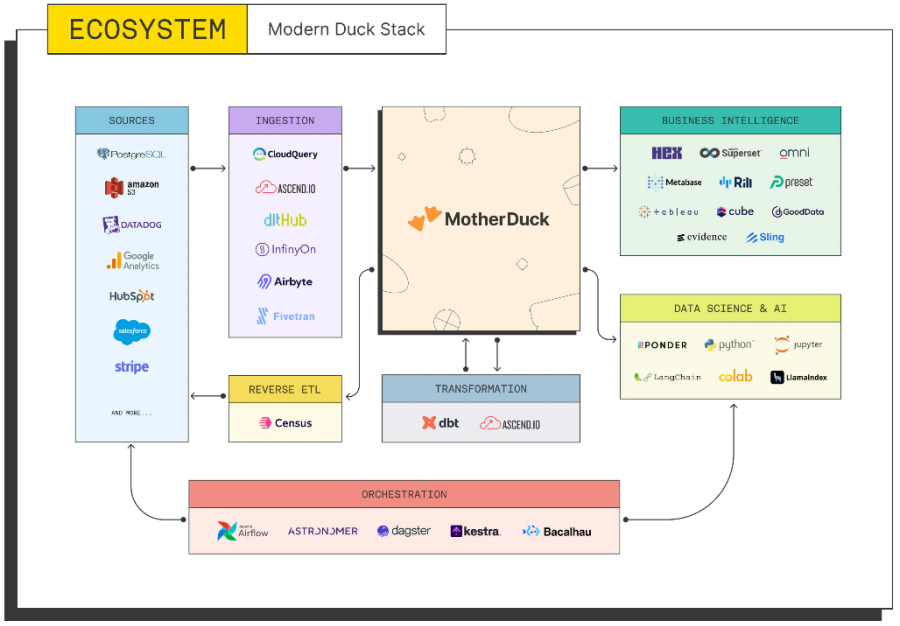


Figure 7.5 MotherDuck Integrations

Going from left to right, with optional transformations in between, MotherDuck can sit in the middle of your pipeline. Sources can either be ingested directly or via additional services, then put into storage at MotherDuck before they are queried either by business intelligence use cases or with specific data science tools. They can serve as a *Retrieval Augmented Generation* (RAG) information retrieval component for LLM models, too.

Additionally, any existing DuckDB integrations and drivers also work with MotherDuck, so everywhere you can use DuckDB, you can make use of MotherDuck by just inserting the `md:` prefix into the database connection string, and appending the `?motherduck_token=<token>` parameter.

7.4 Summary

- MotherDuck is a serverless data analytics platform that makes it easy to query and analyze data in cloud storage from the browser.
- It integrates seamlessly into the DuckDB CLI, the Python, and other language integrations using the `md:` protocol that automatically loads the MotherDuck extension.
- MotherDuck enables you to store structured data, query that data with SQL, and share it with others.

- One key principle to the service is ease of use: You don’t need to configure or spin up instances, clusters, or warehouses. You simply write and submit SQL, in the same tool or from within the same ecosystem when working locally.
- In many cases data can be ingested much faster at MotherDuck than locally due to closer proximity of the sources.
- Local, remote, and shared datasets can easily be joined by using the qualified name of schema and relation.
- The MotherDuck platform also offers support for querying datasets in natural language, allowing people that are not trained in SQL to benefit from the analytical database, too.