dremio

Dremio Software

Ingest and Manipulate Data Using Dremio and Iceberg

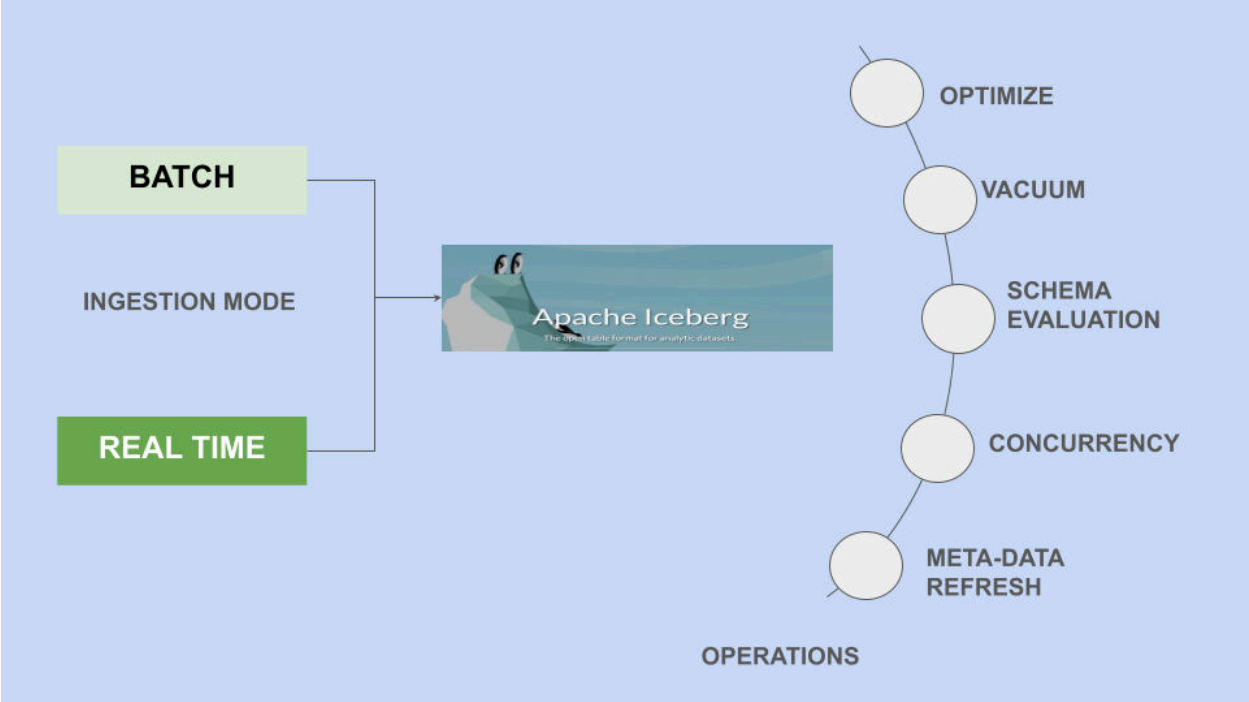
Introduction

Organizations have built data lakes leveraging cloud storage to host vast amounts of structured, semi-structured and unstructured data. However, unlike data warehouses, which allow easy data updates, it's quite a challenge to update data in a data lake.

A data lakehouse combines the flexibility and scalability of a data lake with the structured and optimized processing capabilities of a data warehouse. This document describes the steps and design considerations for building a data lakehouse using Dremio and Apache Iceberg to ingest and manipulate data in the data lake easily and quickly.

> ⚠ NOTICE
> This document assumes you have Dremio version 24.3 or higher installed and running.

Design Considerations for Data Lakehouses



When building a data lakehouse where we want to be able to ingest and manipulate data directly using the Iceberg table format in our data lake, the following operational and functional aspects need to be considered:

- Historic data upload
- DML operations on data (Insert / Update / Delete)
- Rewrite data to optimal file size
- Time travel
- Rollback
- Vacuum
- Schema evolution to accommodate change requests
- Concurrency
- Metadata refresh
- Table rollback in any eventuality

The following sections of this document take you on a walk-through of an example Dremio Data Lakehouse to demonstrate how Dremio addresses each of these aspects.

Sample Data Lakehouse

In this example, we regularly collect sensor data into on-premises Minio storage. This sensor keeps track of the temperature of a device. As you can see from the snippet of data below, the data stored in Minio is in CSV format.

```
ts_year,ts_month,ts_day,ts_hour,ts,metric,data
2024,1,5,18,2024-01-05 18:06:49.000,'Temperature',107.85
2024,1,5,18,2024-01-05 18:06:49.000,'Temperature',67.34
2024,1,5,18,2024-01-05 18:06:49.000,'Temperature',110.26
2024,1,5,18,2024-01-05 18:06:49.000,'Temperature',90.65
2024,1,5,18,2024-01-05 18:06:49.000,'Temperature',74.35
```

As part of creating our Data Lakehouse solution, we wish to store and analyze this sensor data in an Azure ADLS Gen 2 data lake using Dremio and Iceberg.

Table Creation

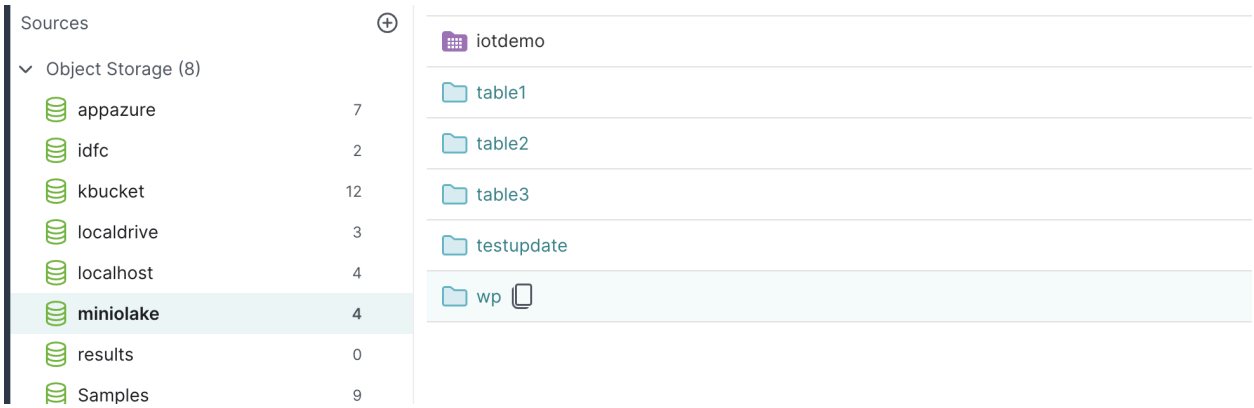
We assume that Dremio is already connected to the Azure Storage data lake source called `appazure`. In Dremio, we can use the `CREATE TABLE` command to create a table called `demotable` in that target `appazure` source. The syntax for this is as follows:

```
CREATE TABLE appazure.demotable (ts_year INT, ts_month INT, ts_day INT, ts_hour INT,
ts TIMESTAMP, metric VARCHAR(100), data DECIMAL(10,2))
```

Historic Data Upload

Dremio provides the `COPY INTO` SQL command to upload large volumes of data to Iceberg tables.

We must load the historic CSV data from Minio into the Iceberg table we created in the previous section. In Dremio, we also have a connection to the Minio data source, which is called `miniolake`, and we can see that the CSV data resides in a folder called `wp`.



By running the following command in Dremio, we can load the historical data into our Iceberg table:

```
COPY INTO appazure.demotable FROM '@miniolake/wp/data-2024-01-05_18:06:49.544348.csv'
```

DML Operations On Data

With data now loaded into the Iceberg table, there are several scenarios where periodic changes might need to be made to the contents of the table:

- Insertion of new records - Use Dremio's `INSERT` statement
- Update of specific existing records - Use Dremio's `UPDATE` statement
- Deletion of specific records - User Dremio's `DELETE` statement
- Bulk insert from files - Use the same `COPY INTO` command referenced previously to load periodic data into the target table.
- Bulk update or insert from another source table - Requires a two-step process. First, use `COPY INTO` to load data to a temporary table, then use `MERGE` to insert or update records from the temporary table to the target table.

Rewrite Data to Optimal File Size

`Insert`, `Update` and `Delete` operations on the Iceberg table will cause parquet files supporting the table to be created in the data lake. As more files get added to the table, there may be a need to optimize the Iceberg table in Dremio due to suboptimal file sizes and an evolving partition scheme. The `OPTIMIZE` command supports the `bin_pack` clause, enabling users to bin pack files (i.e. reduce the number of data files by putting as much data as possible together) in partitions they actively write to.

As an example, the following SQL command will pick up all files of size less than 100MB or greater than 1GB and rewrite them to file sizes of 256 MB. It will need at least five files satisfying the criteria to make the SQL operation successful.

```
OPTIMIZE TABLE appazure.demotable REWRITE DATA USING BIN_PACK (MIN_FILE_SIZE_MB=100,
MAX_FILE_SIZE_MB=1000, TARGET_FILE_SIZE_MB=256, MIN_INPUT_FILES = 5)
```

Time Travel

Snapshot-Based

Each change made to an Iceberg table creates a snapshot of that table. A snapshot is a timestamped version of a table. Users can get historical versions of the data by using time travel queries that contain the `AT SNAPSHOT` clause to retrieve snapshots.

For example, let's first consider that we have some temperature data (see the `data` column) for a particular timestamp in our Iceberg table `demotable`.

```
select * FROM APPAZURE.demotable where ts_year = 2024 and ts_month = 1 and ts_day = 5 and ts_hour = 18
and ts = '2024-01-05 18:06:49.000' limit 5
```

Query1 | Add Column | Group By | Join | Filter Columns | 7 Columns | Job: Run | Rows: 5 | <1s

	ts_month	ts_day	ts_hour	ts	metric	data
2024	1	5	18	2024-01-05 18:06:49.000	'Temperature'	107.85
2024	1	5	18	2024-01-05 18:06:49.000	'Temperature'	67.34
2024	1	5	18	2024-01-05 18:06:49.000	'Temperature'	110.26
2024	1	5	18	2024-01-05 18:06:49.000	'Temperature'	90.65
2024	1	5	18	2024-01-05 18:06:49.000	'Temperature'	74.35

We can now update the data for this specific timestamp, making all `data` values 0.

```
update appazure.demotable set data = 0 where ts_year = 2024 and ts_month = 1 and ts_day = 5 and ts_hour = 18
and ts = '2024-01-05 18:06:49.000'

```

Query1 | Add Column | Group By | Join | Filter Columns | 1 Column | Job: Run | Rows: 5,000 | 3s

Rows Updated
1808

Now we see that in the current, most recent snapshot, the `data` field has been updated as per our `update` statement:

```
select * from appazure.demotable where ts_year = 2024 and ts_month = 1 and ts_day = 5 and ts_hour = 18
and ts = '2024-01-05 18:06:49.000'

```

Query1 | Add Column | Group By | Join | Filter Columns | 7 Columns | Job: Run | Rows: 1,808 | 1s

	ts_month	ts_day	ts_hour	ts	metric	data
2024	1	5	18	2024-01-05 18:06:49.000	'Temperature'	0.00
2024	1	5	18	2024-01-05 18:06:49.000	'Temperature'	0.00
2024	1	5	18	2024-01-05 18:06:49.000	'Temperature'	0.00
2024	1	5	18	2024-01-05 18:06:49.000	'Temperature'	0.00
2024	1	5	18	2024-01-05 18:06:49.000	'Temperature'	0.00
2024	1	5	18	2024-01-05 18:06:49.000	'Temperature'	0.00
2024	1	5	18	2024-01-05 18:06:49.000	'Temperature'	0.00
2024	1	5	18	2024-01-05 18:06:49.000	'Temperature'	0.00
2024	1	5	18	2024-01-05 18:06:49.000	'Temperature'	0.00
2024	1	5	18	2024-01-05 18:06:49.000	'Temperature'	0.00

Since Iceberg supports time travel, users can still query the data from a point in time before the update statement was executed. Doing this requires us first to query the history information for our Iceberg table, which will give us a list of all snapshots created for the table, when they were created, and the `snapshot_id` of each snapshot. We can use the following query to retrieve the snapshot history:

```
SELECT * FROM TABLE(TABLE_HISTORY('appazure.demotable'))
```

```
1 select * from table(table_history('appazure.demotable'))
2
```

Query1

Add Column | Group By | Join | Filter Columns | 4 Columns | Job: Run | Rows: 6 | <1s

made_current_at	snapshot_id	parent_id	is_current_ancestor
2024-01-05 12:35:10.819	1262466556998718225	null	true
2024-01-05 12:38:16.716	5931800812968202180	1262466556998718225	true
2024-01-10 07:49:35.397	1596887186391396357	5931800812968202180	false
2024-01-11 05:20:54.313	2164370265865781557	1596887186391396357	false
2024-01-11 05:24:41.725	5931800812968202180	1262466556998718225	true
2024-01-11 05:34:01.965	6479928264204733780	5931800812968202180	true

The data above shows that the snapshot_id “5931800812968202180” represents the state just before our update statement was executed. By introducing the `AT SNAPSHOT '<snapshot_id>'` clause into the query against our Iceberg table, we can see the data pertaining to that point in time before the update.

```
6
7 select * from appazure.demotable at snapshot '5931800812968202180' where ts_year = 2024 and ts_month = 1 and ts_day = 5 and ts_hour = 18
8 and ts = '2024-01-05 18:06:49.000'
```

Query1

Add Column | Group By | Join | Filter Columns | 7 Columns | Job: Run | Rows: 1,808 | <1s

	ts_month	ts_day	ts_hour	ts	metric	data
2024	1	5	18	2024-01-05 18:06:49.000	'Temperature'	107.85
2024	1	5	18	2024-01-05 18:06:49.000	'Temperature'	67.34
2024	1	5	18	2024-01-05 18:06:49.000	'Temperature'	110.26
2024	1	5	18	2024-01-05 18:06:49.000	'Temperature'	90.65
2024	1	5	18	2024-01-05 18:06:49.000	'Temperature'	74.35
2024	1	5	18	2024-01-05 18:06:49.000	'Temperature'	74.61
2024	1	5	18	2024-01-05 18:06:49.000	'Temperature'	69.28

Timestamp-Based

Iceberg tables also support timestamp-based references where users can select historical versions of the data by using time travel queries that contain the `AT <timestamp>` clause. A query using this clause will use the most recent Iceberg snapshot as of the provided timestamp.

Rollback

Iceberg allows rolling back a table to any previous snapshot using either a timestamp or a snapshot ID together with the `ROLLBACK TABLE` command.

Rollback to a Previous Snapshot

When you roll back an Iceberg table using a snapshot, you create a new snapshot containing the same state as the referenced snapshot. Here, we are rolling back our `demotable` Iceberg table to the snapshot before we made the updates in the previous section.

```
ROLLBACK TABLE appazure.demotable TO SNAPSHOT '5931800812968202180'
```

Query1

Add Column | Group By | Join | Filter Columns | 2 Columns | Job: Run | Rows: - | <1s

ok	summary
true	Table [appazure.demotable] rollbacked

By issuing the same original query as in the previous section, we can see our data has reverted to how it looked before it was updated.

```
select * from appazure.demotable  where ts_year = 2024 and ts_month = 1 and ts_day = 5 and ts_hour = 18
and ts = '2024-01-05 18:06:49.000'
```

Query1

Add Column | Group By | Join | Filter Columns | 7 Columns | Job: Run | Rows: 1,808 | <1s

ts_month	ts_day	ts_hour	ts	metric	data
2024	1	5	18	2024-01-05 18:06:49.000	'Temperature'	107.85
2024	1	5	18	2024-01-05 18:06:49.000	'Temperature'	67.34
2024	1	5	18	2024-01-05 18:06:49.000	'Temperature'	110.26
2024	1	5	18	2024-01-05 18:06:49.000	'Temperature'	90.65
2024	1	5	18	2024-01-05 18:06:49.000	'Temperature'	74.35
2024	1	5	18	2024-01-05 18:06:49.000	'Temperature'	74.61
2024	1	5	18	2024-01-05 18:06:49.000	'Temperature'	69.28
2024	1	5	18	2024-01-05 18:06:49.000	'Temperature'	102.05

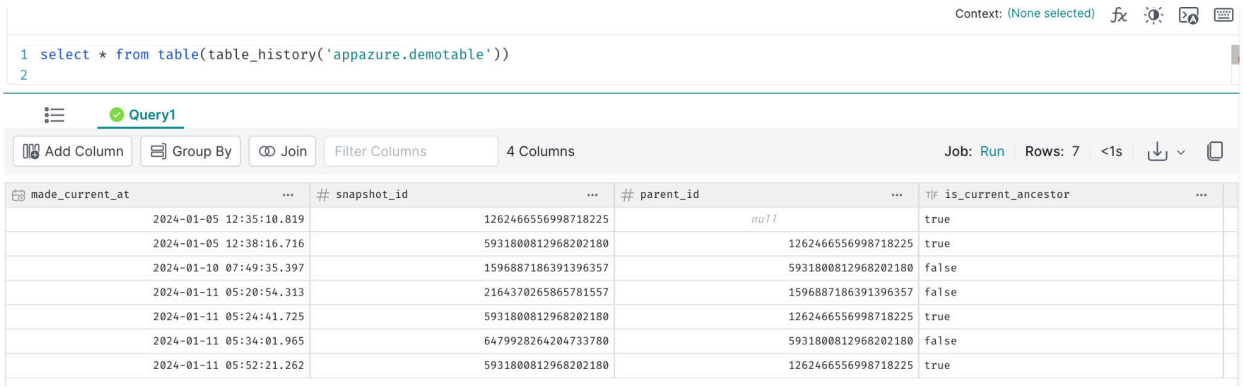
Rollback to a Specific Point In Time

When you roll back an Iceberg table using a timestamp, you are reverting the table to a previous snapshot based on your specified time. Specifically, if the timestamp matches a snapshot's timestamp exactly, the Iceberg table is rolled back to that snapshot. Otherwise, the table is rolled back to the last snapshot before the specified timestamp.

Vacuum

As discussed previously, Dremio creates a snapshot for any DML operation applied to the table. At some point, we may decide that older snapshots are no longer needed. You can use the `VACUUM TABLE` command to remove snapshots you no longer need and the files (data files, the manifest file, the manifest list file, and partition stats files) associated only with them.

To demonstrate this, as a first step, we can remind ourselves of the snapshots associated with our `demotable`.



made_current_at	snapshot_id	parent_id	is_current_ancestor
2024-01-05 12:35:10.819	1262466556998718225	null	true
2024-01-05 12:38:16.716	5931800812968202180	1262466556998718225	true
2024-01-10 07:49:35.397	1596887186391396357	5931800812968202180	false
2024-01-11 05:20:54.313	2164370265865781557	1596887186391396357	false
2024-01-11 05:24:41.725	5931800812968202180	1262466556998718225	true
2024-01-11 05:34:01.965	6479928264204733780	5931800812968202180	false
2024-01-11 05:52:21.262	5931800812968202180	1262466556998718225	true

We can see above that the most recent snapshot is taken as of '2024-01-11 05:52:21.262'.
Using the following statement, we can keep the latest snapshot and eliminate all previous ones.

```
VACUUM TABLE appazure.demotable
EXPIRE SNAPSHOTS OLDER_THAN '2024-01-11 05:52:21.262' RETAIN_LAST 1
```

deleted_data_files_cou…	deleted_position_delet…	deleted_equality_delet…	deleted_manifest_files…	deleted_manifest_lists…	deleted_partition_stat…
3	0	0	5	4	0

Organizations may keep as many historic snapshots as desired, per their policy.

> ⚠ NOTICE
> VACUUM effectively purges the data associated with earlier snapshots. It cannot be regained with a ROLLBACK. So users can only do time travel on the remaining snapshots.

Schema Evolution

Dremio allows altering a table's definition or schema using the [ALTER TABLE] command.
As an example, we can add a partition scheme on an existing column in our `demotable` using the following statement:

```
ALTER TABLE appazure.demotable ADD PARTITION FIELD ts_year
```

👍TIP
Run the `OPTIMIZE` command after adding a partition

Concurrency

Iceberg supports concurrent reads. Readers always use a consistent snapshot of the table without holding a lock.

Iceberg supports multiple concurrent writes wherein each writer assumes that no other writers are operating and writes out new table metadata for an operation. Then, the writer attempts to commit by atomically swapping the new table metadata file for the existing metadata file.

If the file swap fails because another writer has committed, the failed writer retries by writing a new metadata tree based on the new current table state.

Metadata Refresh

For Iceberg Tables created by Dremio, there is no need to do a metadata refresh separately. When data is added or updated in an Iceberg table, the table's metadata is updated with DML operation, so no separate refresh is needed.

Suppose all or a majority of your datasets are Iceberg tables. In that case, there may be no need to set up an isolated engine for metadata refresh since that engine will likely be underutilized.

Other Considerations

There are two approaches to handling deletes and updates in a Data Lakehouse: copy-on-write (COW) and merge-on-read (MOR).

Like with almost everything in computing, there isn't a one-size-fits-all approach; each strategy has trade-offs that make it the better choice in certain situations. The considerations largely come down to latency on the read versus write side.

Copy-On-Write (COW) – Best for tables with frequent reads, infrequent writes/updates, or extensive batch updates

Merge-On-Read (MOR) – Best for tables with frequent writes/updates

This blog post gives an excellent summary of the differences between the two strategies.

> ⚠ NOTICE
> Dremio supports only the copy-on-write storage mechanism and reads only the latest data files for each Iceberg v2 table against which you run SQL commands. Dremio does not support Iceberg v2 tables that have merge-on-read manifests.

Conclusion

Data lakehouses built using Iceberg provide the capacity to store and analyze huge volumes of data. Parallely, it gives flexibility for easy data manipulation and features like time travel and schema evolution. With the help of an example, this document demonstrated how users can build a data lakehouse using Dremio and Iceberg.