What is an open table format? & Why to use one?

with Apache Iceberg

Joseph Machado

Introduction

- Name: Joseph Machado
- Work experience: Data engineer
- Specialty: Data Platform & Pipelines
- Blogs at: Startdataengineering

Table of Contents

Questions to address

- What is an open table format(OTF)? & How is it different from file formats like Parquet or ORC?
- What are the pros and cons of using an open table format? & How does it work?
- Is it just a pointer to some metadata files and helps you sift through the data quickly?

By the end of this presentation, you will know what OTFs are, why you use them, and how they work.

TL;DR

Open table formats are wrappers around your data store & use a series of files to

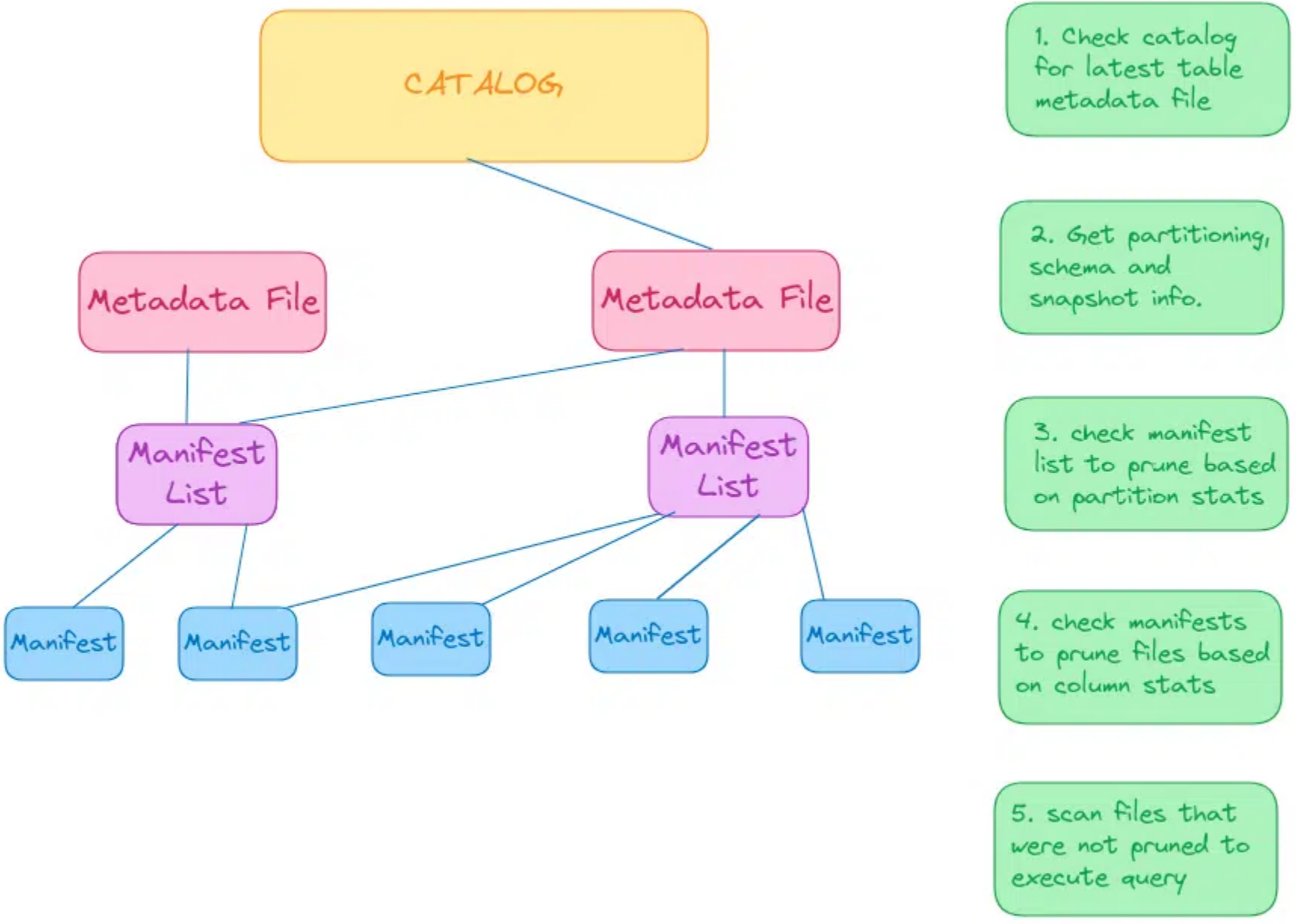
1. Track schema/partition (DDL) changes on your table.
2. Track the table's data files & their column statistics.
3. Track all the Inserts/Updates/Deletes (DML) on your table.

Storing a chronological series of files with all the DDL and DML statements applied to your table & index of the data file locations enables

1. **Schema & Partition Evolution**
2. **Travelling back in time to a previous table state**
3. **Creating table branches & tagging table state (similar to git)**
4. **Handling multiple reads & writes concurrently**

How do OTFs work?

- OTFs are implemented as libraries that handle table management using metadata files. E.g., Apache Iceberg Java library
- Apache Iceberg uses
  - `Metadata` files to keep track of data/partition schema changes.
  - `Manifest_list` contains partition statistics.
  - `Manifest` files contain column-level statistics and links to the actual data.



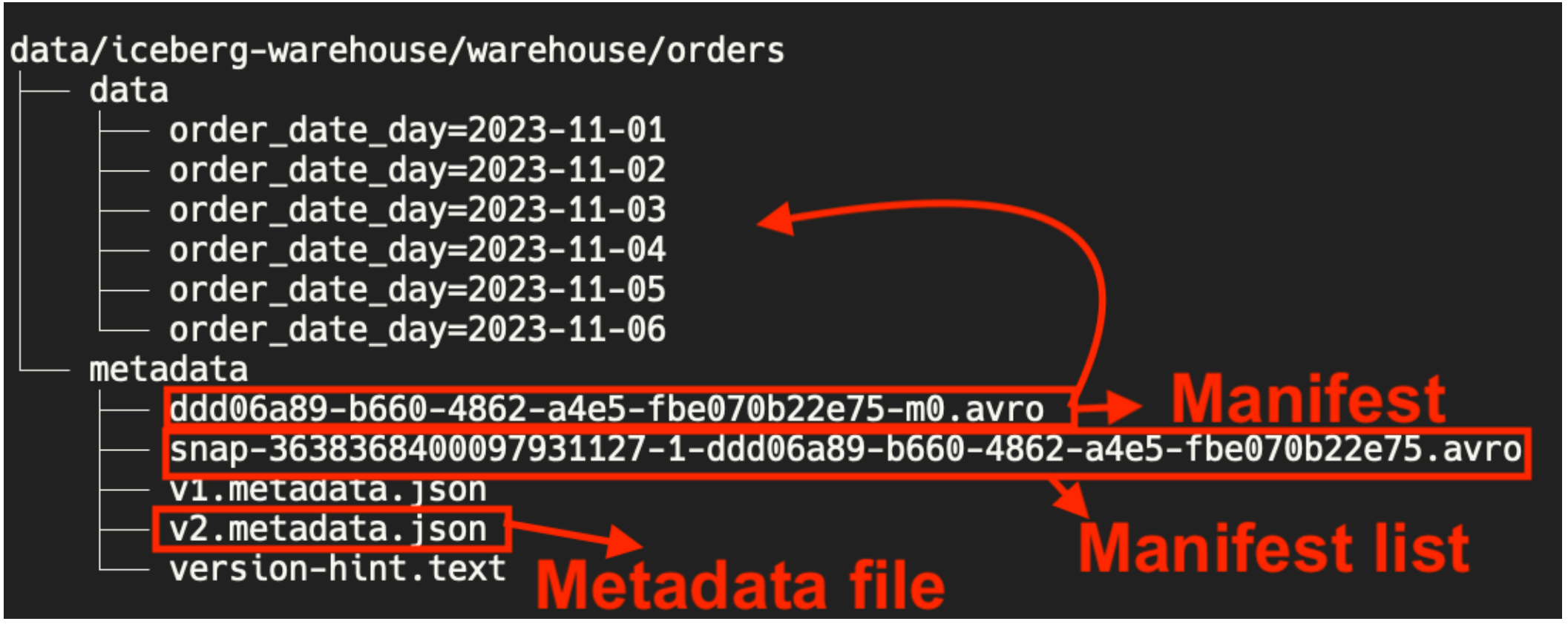
Ref: Dremio blog

```
CREATE SCHEMA warehouse;
-- Use Apache Iceberg table format
DROP TABLE IF EXISTS local.warehouse.orders;
CREATE TABLE local.warehouse.orders (
    order_id string,
    cust_id INT,
    order_status string,
    order_date timestamp
) USING iceberg
PARTITIONED BY (date(order_date));

INSERT INTO local.warehouse.orders VALUES
('e481f51cbdc54678b7cc49136f2d6af7',69,'delivered',CAST('2023-11-01 09:56:3
('e481f51cbdc54678b7cc49136f2d6af7',87,'delivered',CAST('2023-11-01 10:56:3
('e481f51cbdc54678b7cc49136f2d6af7',125,'delivered',CAST('2023-11-01 11:56:
('53cdb2fc8bc7dce0b6741e2150273451',17,'delivered',CAST('2023-11-02 11:56:3
('53cdb2fc8bc7dce0b6741e2150273451',19,'on_route',CAST('2023-11-02 12:56:33
('47770eb9100c2d0c44946d9cf07ec65d',26,'on_route',CAST('2023-11-03 12:56:33
```

Code: Github Iceberg Features

Files generated as part of our Orders Iceberg table



ALTER data and partitions without reprocessing

Data Schema & Partition Evolution

- Apache Iceberg enables you to **change your table's data schema or partition schema without reprocessing the existing data**.
- Metadata files track schema & partition changes over time.
- Processing systems use metadata to understand current and past schemas and apply the appropriate schema.

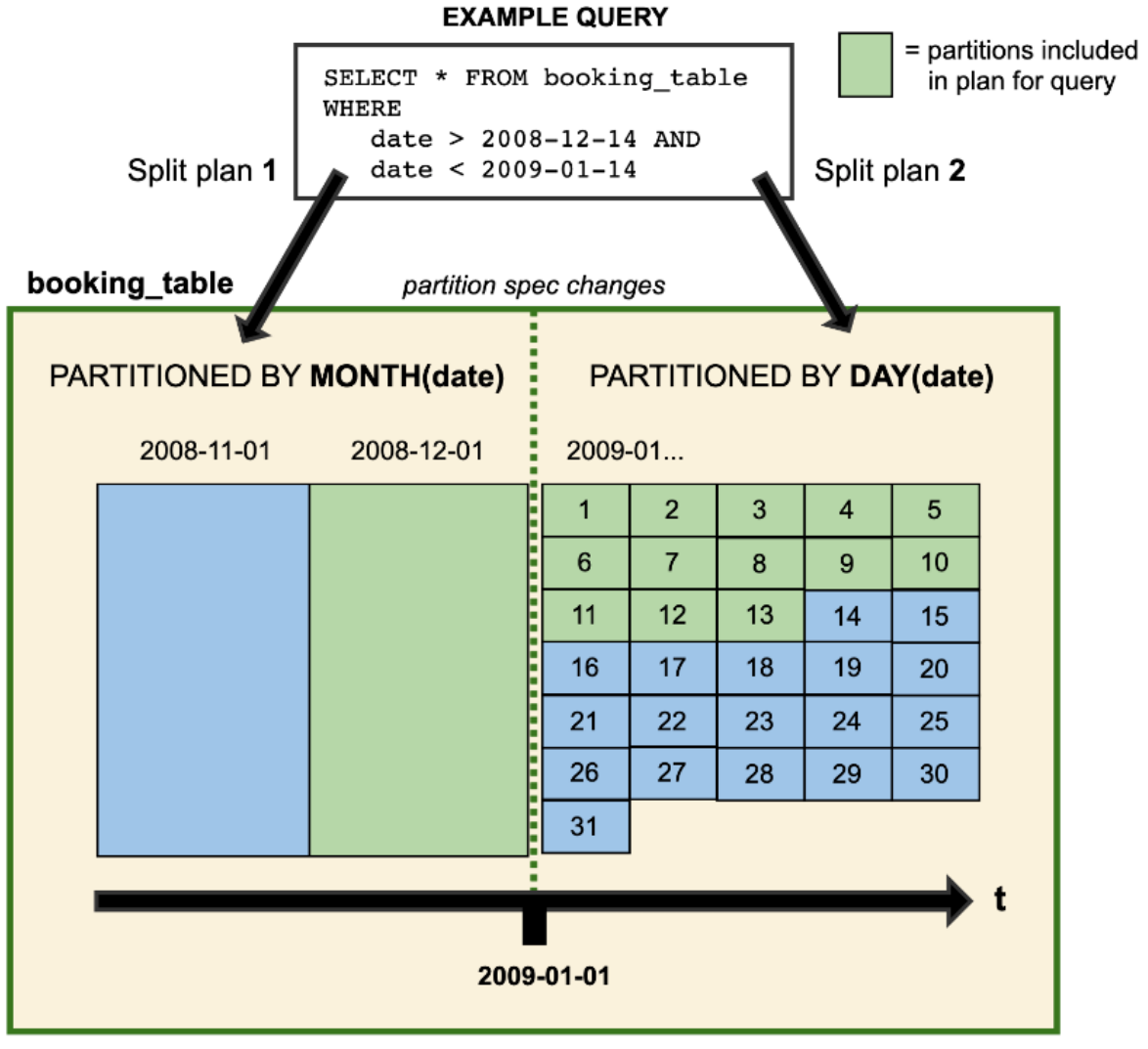
For example:

```
-- schema evolution
ALTER TABLE local.warehouse.orders ALTER COLUMN cust_id TYPE bigint;
ALTER TABLE local.warehouse.orders DROP COLUMN order_status;

-- parititon evolution
ALTER TABLE local.warehouse.orders ADD PARTITION FIELD cust_id;
INSERT INTO local.warehouse.orders VALUES
(
    'e481f51cbdc54678b7cc49136f2d6af7',
    69,
    CAST('2023-11-14 09:56:33' AS TIMESTAMP)
),
(
    'e481f51cbdc54678b7cc49136f2d6af7',
    87,
    CAST('2023-11-14 10:56:33' AS TIMESTAMP)
);

-- check snapshots
```

Reading data which has undergone partition evolution



Hidden partition

- Typically, you would have had to create a separate column to add a partition.
- Apache Iceberg lets you define(& enable the use of) partitions based on column transformations of another column. E.g., you can partition by date on a timestamp column.

```
-- The partition is done on a transformation of an existing col
CREATE TABLE local.warehouse.orders (
    order_id string,
    cust_id INT,
    order_status string,
    order_date timestamp
) USING iceberg
PARTITIONED BY (date(order_date));

-- The below query automatically uses the partition to
-- prune data files to scan.
SELECT cust_id, order_date
FROM local.warehouse.orders
WHERE order_date
BETWEEN '2023-11-01 12:45:33' AND '2023-11-03 12:45:33';
```

The `day` transform function is applied to the `order_date` column for a partition.

See previous point-in-time table state, aka time travel

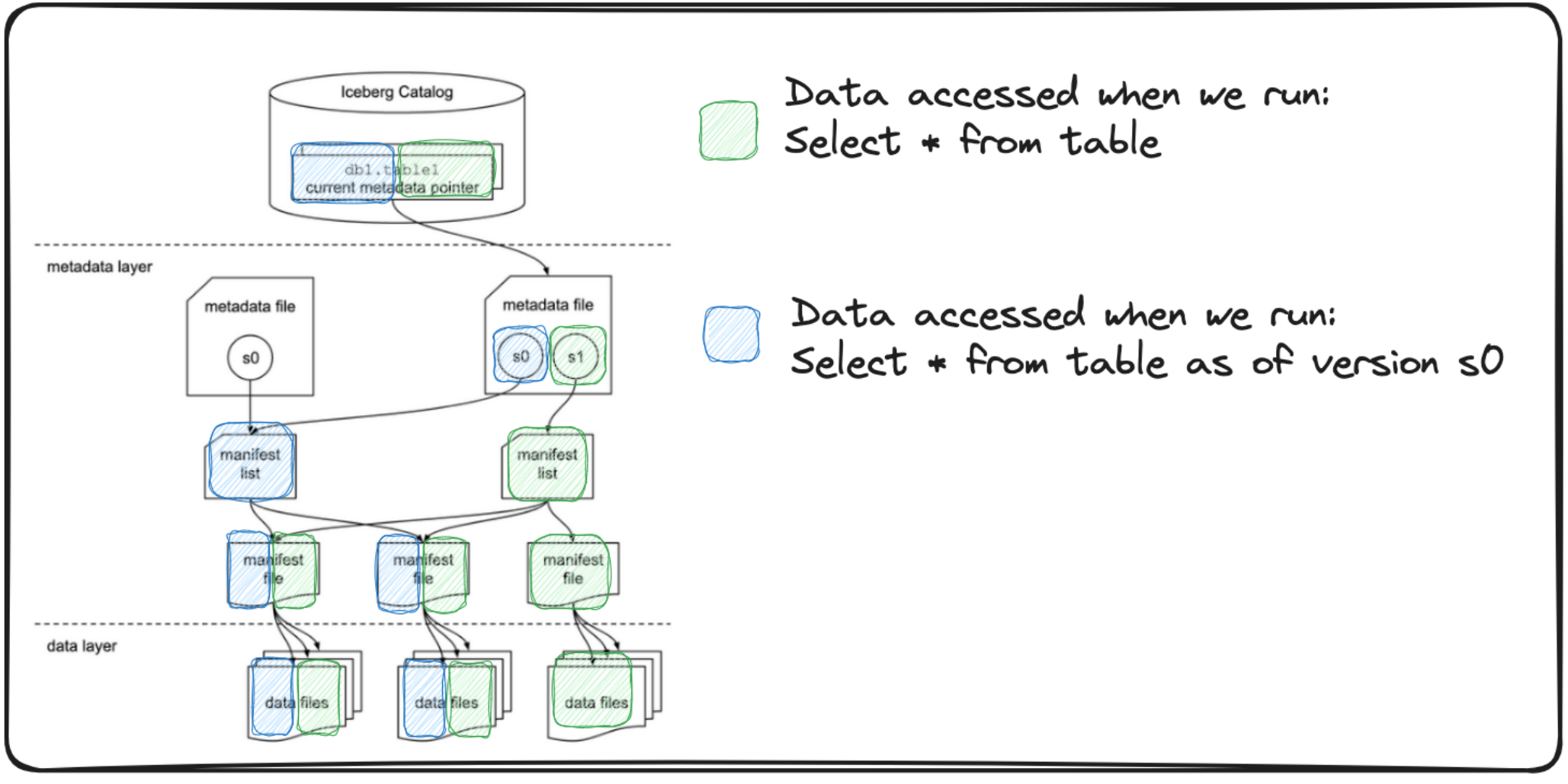
- We can see our table at a previous point-in-time table state(aka time travel). For example, If you want to see what a table looked like four days ago, you can do this using time travel.
- Time travel is possible because the metadata files keep track of all the DML and DDL operations that ran on the table.
- Apache Iceberg keeps links to the deleted data; this enables the user to see deleted data.

```
-- get the time of the first data snapshot
select min(committed_at) as min_committed_at
from local.warehouse.orders.snapshots;
-- e.g. 2023-11-21 12:03:08.833

-- Query data as of the oldest committed_at (min_committed_at from the abov
SELECT *
FROM local.warehouse.orders TIMESTAMP AS OF '2023-11-21 12:03:08.833';
-- 15 rows

-- Query without time travel, and you will see all the rows
SELECT * FROM local.warehouse.orders;
-- 17 rows
```

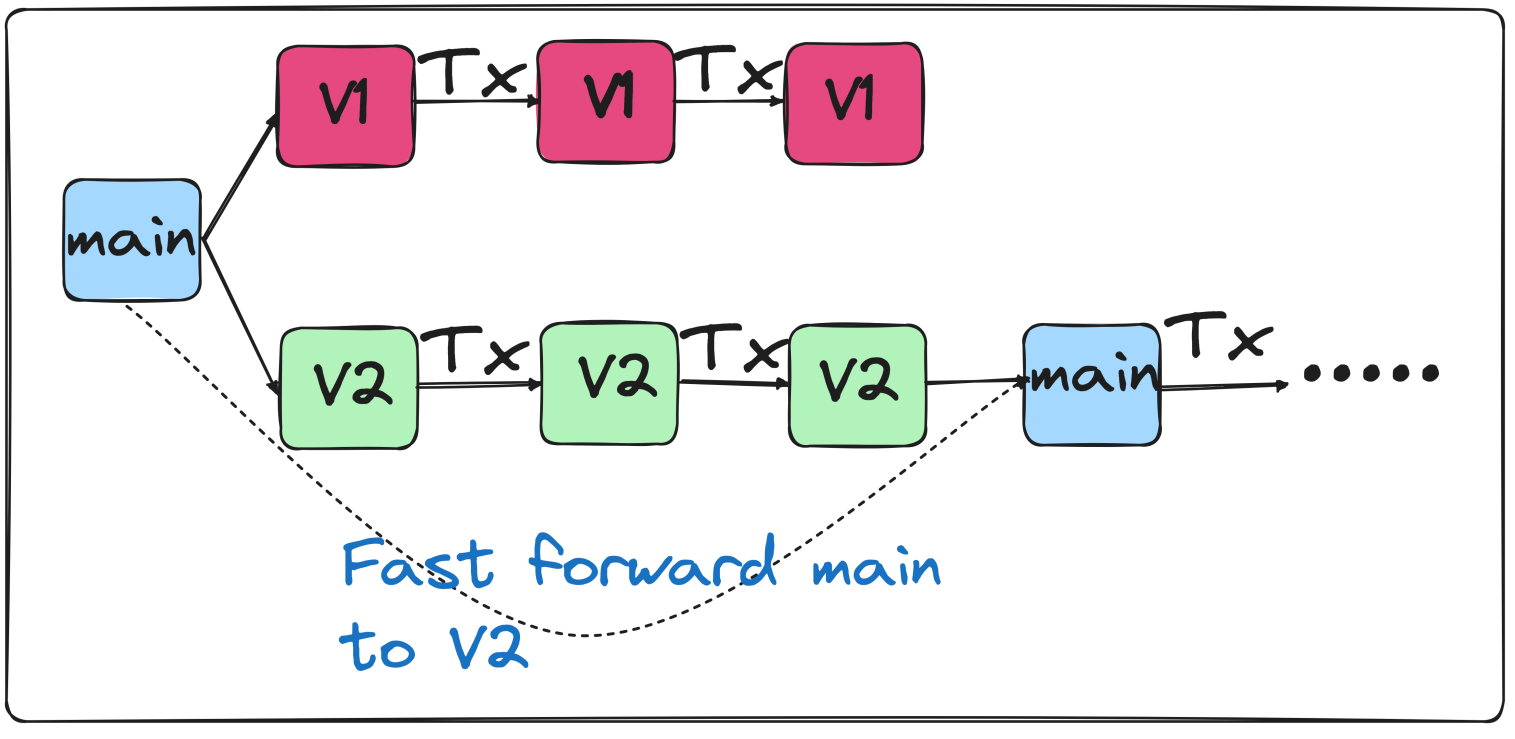
Note: For Apache Iceberg, as your data size grows, it's recommended to clean up old snapshots.



Apache Iceberg Time Travel

Git like branches & tags for your tables

- Apache Iceberg enables the creation of table branches by managing isolated metadata files per branch.
- For example, assume you have to run a pipeline in production for a week to validate that the changes you made are valid. You can do this using branches as shown below:



Create a simple aggregate table that contains aggregated order data.

```
DROP TABLE IF EXISTS local.warehouse.orders_agg;
CREATE TABLE local.warehouse.orders_agg(
    order_date date,
    num_orders int
)  USING iceberg;

INSERT INTO local.warehouse.orders_agg
SELECT date(order_date) as order_date,
count(order_id) as num_orders
from local.warehouse.orders
WHERE date(order_date) = '2023-11-02'
GROUP BY 1;
```

Create two branches that are both stored for ten days.

```
ALTER TABLE local.warehouse.orders_agg
CREATE BRANCH `branch-v1` RETAIN 10 DAYS;
ALTER TABLE local.warehouse.orders_agg
CREATE BRANCH `branch-v2` RETAIN 10 DAYS;

-- Use different logic for each of the branches

-- inserting into branch v1
INSERT INTO local.warehouse.orders_agg.`branch_branch-v1`
SELECT date(order_date) as order_date
, count(order_id) as num_orders
from local.warehouse.orders
WHERE date(order_date) = '2023-11-03'
GROUP BY 1;

INSERT INTO local.warehouse.orders_agg.`branch_branch-v1`
SELECT date(order_date) as order_date
, count(order_id) as num_orders
from local.warehouse.orders
```

We noticed that `branch-v2` has the correct logic, so we fast-forward the main branch to `branch-v2`. The main branch will now have the accurate data for the past two days.

```
select *
from local.warehouse.orders_agg
order by order_date;

-- Push the main branch to branch v2's state
CALL local.system.fast_forward('warehouse.orders_agg', 'main', 'branch-v2')

select * from local.warehouse.orders_agg order by order_date;
```

Handle multiple reads & writes concurrently

- In traditional OLAP systems (e.g., HIVE) or when creating raw data on cloud storage, if multiple processes read/write to the same table without proper safeguards, there may be inconsistent data reads, or data may get overwritten during writes.
- Apache Iceberg atomically updates its metadata, which forces writers to “commit” their changes one at a time (if multiple writers collide, there will be a retry for the failed writer).
- When reading data, Apache Iceberg uses the most recent snapshot (using a metadata file) to ensure that no in-process data operations impact reads.

- Since Apache Iceberg is OSS, we can use any system that implements the table format to be able to read and write.
- E.g., We can use DuckDB to read our data:

```
INSTALL iceberg;
LOAD iceberg;

-- Count orders by date
WITH orders as (
    SELECT *
    FROM
    iceberg_scan(
        'data/iceberg-warehouse/warehouse/orders',
         ALLOW_MOVED_PATHS=true
         )
    )
select strftime(order_date, '%Y-%m-%d') as order_date
, count(distinct order_id) as num_orders
from orders
group by strftime(order_date, '%Y-%m-%d')
order by 1 desc;
```

Recap

Open table formats are wrappers around your data store & use a series of files to

1. Track schema/partition (DDL) changes on your table.
2. Track the table's data files & their column statistics.
3. Track all the Inserts/Updates/Deletes (DML) on your table.

Storing a chronological series of files with all the DDL and DML statements applied to your table & index of the data file locations enables

1. **Schema & Partition Evolution**
2. **Travelling back in time to a previous table state**
3. **Creating table branches & tagging table state (similar to git)**
4. **Handling multiple reads & writes concurrently**

Available now

Public preview Apache Iceberg tables on Snowflake !!!