Building an Apache Iceberg Table from Parquet Using Daft, Pylceberg, SQLite Catalog, and DuckDB

What is Apache Iceberg?

Apache Iceberg is an open table format for huge analytic datasets.
It brings **ACID transactions**, **schema evolution**, and **time travel** to data stored in Parquet, ORC, or Avro files.

Key benefits:

- Versioned, snapshot-based tables.
- Separation of metadata and data.
- Compatible with engines like Spark, Flink, Trino, and DuckDB.

Purpose of the Project

To build an **Apache Iceberg table** using a **local SQLite catalog**, by converting a Parquet dataset into partitioned Parquet files, registering them as an Iceberg table, and then reading/querying that table programmatically and via DuckDB.

Technologies Used:

Python

Implementation language

VS Code

Development environment

Daft

Distributed DataFrame library to handle Parquet I/O and partitioning

Pylceberg

Python client for Apache Iceberg metadata management

SQLite Catalog

Stores Iceberg table metadata locally

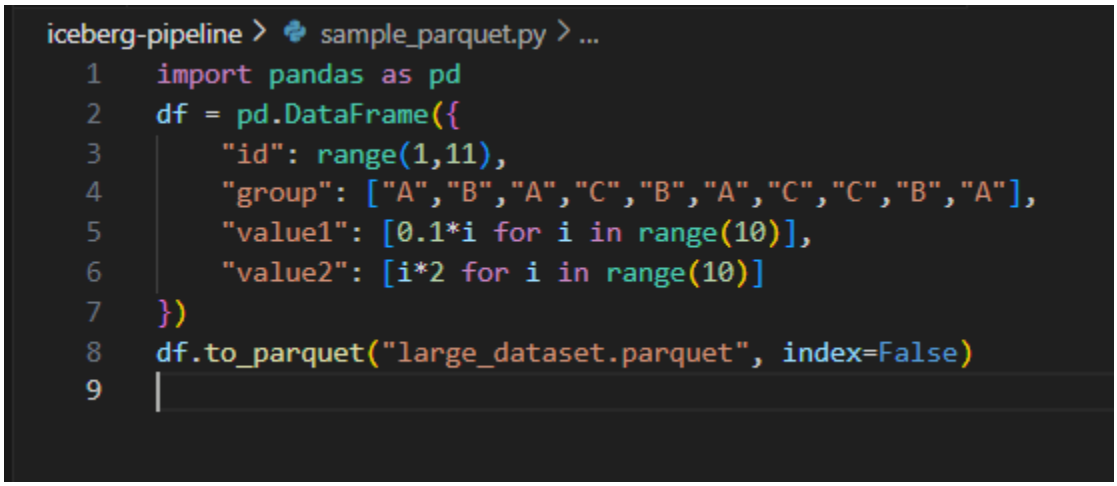
DuckDB

In-process SQL engine to query Iceberg tables

PyArrow

Parquet + Arrow schema manipulation

1. Create a Sample Parquet and run it



2. Create three files: main.py, read_history.py, query_iceberg_duckdb.py

   main.py

```
import os
import glob
import shutil
import pyarrow.parquet as pq
import pyarrow as pa
import daft  # type: ignore
from pyiceberg.catalog import load_catalog
from pyiceberg.schema import Schema, NestedField, IntegerType, StringType, FloatType
from pyiceberg.partitioning import PartitionSpec, PartitionField, INITIAL_PARTITION_SPEC_ID, PARTITION_FIELD_ID_START
from pyiceberg.transforms import IdentityTransform
```

   os, glob, shutil — filesystem helpers (create dirs, find files, remove dirs).
   pyarrow.parquet as pq and pyarrow as pa — read parquet files, inspect Arrow schemas, cast tables.
   daft — used to read the input Parquet and write partitioned Parquet files.
   pyiceberg.* modules — create catalogs, define Iceberg schema, partition specs, transforms.

```
daft.context.set_runner_native()
```

   Ensures Daft runs with its native in-process runner (needed so `daft.read_parquet` works predictably on local machine).

```python
class IcebergPipeline:
    def __init__(self, parquet_input="large_dataset.parquet",
                 output_dir="./output_partitioned_parquet",
                 catalog_config=None,
                 namespace="default",
                 table_name="my_iceberg_table",
                 partition_field_name="group"
                ):
        self.parquet_input = parquet_input
        self.output_dir = output_dir
        self.namespace = namespace
        self.table_name = table_name
        self.partition_field_name = partition_field_name
        if catalog_config is None:
            self.catalog_config = {
                "catalog_type": "sqlite",
                "uri": "sqlite:///catalog.db",
                "warehouse": "iceberg_warehouse"
            }
        else:
            self.catalog_config = catalog_config
```

Stores config in instance variables. Defaults to a SQLite catalog (sqlite:///catalog.db) and a local warehouse directory iceberg_warehouse. This matches the article's lightweight development approach using SQLite as a catalog.

```python
def get_catalog(self):
    config = self.catalog_config.copy()
    catalog_type = config.pop("catalog_type", "sqlite")
    return load_catalog(catalog_type, **config)
```

load_catalog from PyIceberg creates/loads a catalog instance. We pop catalog_type then pass the rest as kwargs. For the default config this will call load_catalog("sqlite", uri="sqlite:///catalog.db", warehouse="iceberg_warehouse").

```python
def infer_schema(self):
    sample_files = glob.glob(os.path.join(self.output_dir, "**", "*.parquet"), recursive=True)
    if not sample_files:
        raise ValueError("No parquet files found.")
    sample_schema = pq.read_schema(sample_files[0])
    fields = []
    fid = 1
    for field in sample_schema:
        if pa.types.is_integer(field.type):
            t = IntegerType
        elif pa.types.is_floating(field.type):
            t = FloatType
        elif pa.types.is_string(field.type):
            t = StringType
        else:
            t = StringType
        fields.append(NestedField(fid, field.name, t(), required=not field.nullable))
        fid += 1
    return Schema(*fields)
```

Finds one of the partitioned Parquet files, reads its Arrow schema, maps Arrow types to Iceberg types (IntegerType, FloatType, StringType) and constructs a pyiceberg.schema.Schema with NestedField objects. fid is a field id counter — Iceberg schema fields require ids.

```python
def run_pipeline(self):
    # Clear the output directory to avoid duplicate processing
    if os.path.exists(self.output_dir):
        shutil.rmtree(self.output_dir)
    os.makedirs(self.output_dir, exist_ok=True)
```

Remove old outputs so we don’t append duplicates, then recreate directory.

```python
    # Use Daft to read the input Parquet file and write partitioned files by the specified column.
    df = daft.read_parquet(self.parquet_input)
    df.write_parquet(self.output_dir, partition_cols=[self.partition_field_name])
    print(f"Partitioned Parquet files written to: {self.output_dir}")
```

Read the input Parquet into a Daft dataframe and write partitioned Parquet files partitioned by group (or whatever partition_field_name you set). This creates a directory tree under self.output_dir with files grouped by partition value.

```python
# Load or create the catalog
catalog = self.get_catalog()
os.makedirs(self.catalog_config["warehouse"], exist_ok=True)

schema = self.infer_schema()
```

Create/load the SQLite catalog and ensure warehouse directory exists, then infer the schema from the generated partitioned Parquet.

```
# Determine the field id for the partition field
pfid = None
for field in schema.fields:
    if field.name == self.partition_field_name:
        pfid = field.field_id
        break
if pfid is None:
    raise ValueError(f"Partition field '{self.partition_field_name}' not found in inferred schema.")

partition_field = PartitionField(
    name=self.partition_field_name,
    source_id=pfid,
    field_id=PARTITION_FIELD_ID_START,
    transform=IdentityTransform()
)
spec = PartitionSpec(spec_id=INITIAL_PARTITION_SPEC_ID, fields=(partition_field,))
```

Finds the field_id assigned earlier for the partition column and constructs a PartitionField that uses an IdentityTransform (no bucketing/truncation) — same as the article — and creates a PartitionSpec.

```
# Create namespace and table in the catalog
try:
    catalog.create_namespace(self.namespace)
    print(f"Namespace '{self.namespace}' created.")
except Exception as e:
    print(f"Namespace '{self.namespace}' may already exist: {e}")

table_identifier = f"{self.namespace}.{self.table_name}"
table = catalog.create_table(table_identifier, schema=schema, partition_spec=spec)
print(f"Iceberg table {table_identifier} created with partition spec on '{self.partition_field_name}'."

parquet_files = glob.glob(os.path.join(self.output_dir, "**", "*.parquet"), recursive=True)
print(f"Found {len(parquet_files)} Parquet files to register.")
```

Creates a namespace in the catalog if not present, then creates the Iceberg table using the schema & partition spec.

```python
# Define the forced schema for reading files.
read_schema = pa.schema([
    pa.field("id", pa.int32(), nullable=False),
    pa.field("group", pa.string(), nullable=False),
    pa.field("value1", pa.float32(), nullable=False),
    pa.field("value2", pa.int32(), nullable=False)
])

for file_path in parquet_files:
    try:
        pf = pq.ParquetFile(file_path)
        arrow_table = pf.read()
        gf = arrow_table.schema.field(self.partition_field_name)
        if pa.types.is_dictionary(gf.type):
            idx = arrow_table.schema.get_field_index(self.partition_field_name)
            arrow_table = arrow_table.set_column(
                idx,
                self.partition_field_name,
                arrow_table[self.partition_field_name].dictionary_decode()
            )
        arrow_table = arrow_table.cast(read_schema)
    except Exception as e:
        print(f"Error reading or casting file {file_path}: {e}")
        continue
    table.append(arrow_table)
    print(f"Appended data from file: {file_path} (rows: {arrow_table.num_rows}, size: {os.path.getsize(file_path)} bytes)")
```

For each parquet file: read it into an Arrow table, check if the partition field is dictionary-encoded (a common Parquet optimization) and decode it to regular strings so Iceberg can use it, cast to a forced read_schema to ensure consistent types, then append the Arrow table to the Iceberg table (table.append(arrow_table)). Appending creates snapshot metadata files that Iceberg keeps in the table metadata directory.

```python
        print("\n--- Iceberg Table Metadata ---")
        print("Schema:")
        print(table.schema())
        self.table = table
        self.catalog = catalog

    def print_snapshot_history(self):
        if not hasattr(self, "table"):
            print("Table not loaded. Run run_pipeline() first.")
            return
        print("\nSnapshot History:")
        for snap in self.table.history():
            print(snap)

if __name__ == "__main__":
    pipeline = IcebergPipeline()
    pipeline.run_pipeline()
    pipeline.print_snapshot_history()
```

read_history.py

After append, prints the Iceberg table schema and lets you inspect snapshot history. loads the same SQLite catalog used in main.py, loads the created table, prints the metadata file path and snapshot history so you can see the created metadata JSON files and recorded snapshots

query_iceberg_duckdb.py

iceberg-pipeline > query_iceberg_duckdb.py > ...

```python
import sys
import duckdb
from pyiceberg.catalog import load_catalog

def main():
    config = {"uri": "sqlite:///catalog.db", "warehouse": "iceberg_warehouse"}
    catalog = load_catalog("sqlite", **config)
    table_identifier = "default.my_iceberg_table"
    try:
        table = catalog.load_table(table_identifier)
    except Exception as e:
        print(f"Failed to load table '{table_identifier}': {e}")
        sys.exit(1)

    metadata_file = table.metadata_location
    if not metadata_file:
        print("No metadata file found. Ensure that data was appended to the table.")
        sys.exit(1)
    print("Latest metadata file:", metadata_file)

    metadata_file_duckdb = metadata_file.replace("\\", "/")
    print("Using metadata file for scan:", metadata_file_duckdb)

    con = duckdb.connect(database=":memory:")
    con.execute("INSTALL iceberg;")
    con.execute("LOAD iceberg;")
    query = f"""
        SELECT *
        FROM iceberg_scan('{metadata_file_duckdb}') limit 10;
    """
    try:
        df = con.execute(query).fetchdf()
        print("Iceberg Table Query Result:")
        print(df)
    except Exception as e:
        print("Error executing query:", e)
        sys.exit(1)

if __name__ == "__main__":
    main()
```

loads the table via pyiceberg, takes the latest metadata_location (a JSON file under the warehouse), and calls DuckDB's iceberg_scan to query the snapshot by pointing DuckDB at the metadata file. Remember to install DuckDB (we already added it to pip install).

Outputs:

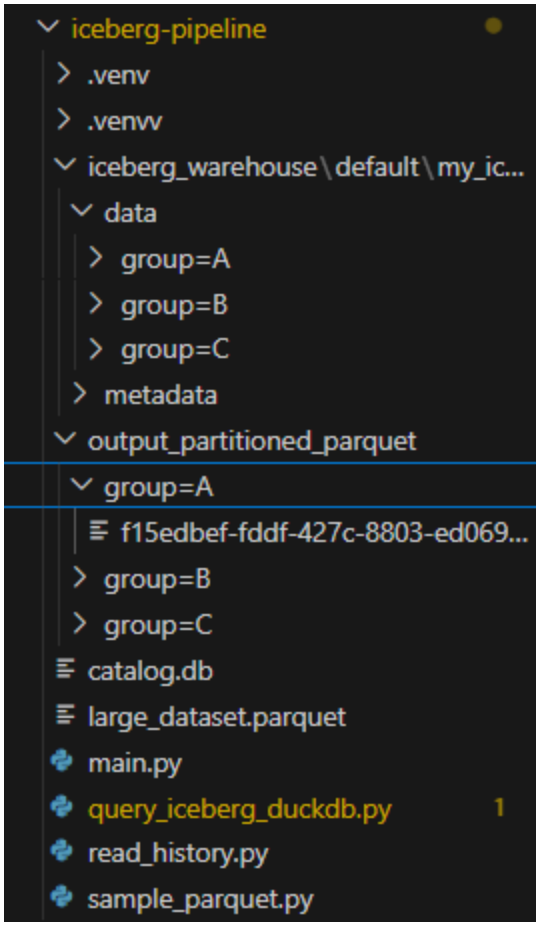
```
(.venv) E:\XCaliber\iceberg-pipeline>python main.py
E:\XCaliber\iceberg-pipeline\main.py:13: DeprecationWarning: This method is deprecated and will be removed in v0.7.0. Use daft.set_runner_native instead.
  daft.context.set_runner_native()
Partitioned Parquet files written to: ./output_partitioned_parquet
Namespace 'default' created.
Iceberg table default.my_iceberg_table created with partition spec on 'group'.
Found 3 Parquet files to register.
Appended data from file: ./output_partitioned_parquet\group=B\f3646676-aae4-485c-aa2a-37e460b4a7eb-0.parquet (rows: 3, size: 1009 bytes)
Appended data from file: ./output_partitioned_parquet\group=A\f15edbef-fddf-427c-8803-ed0696a872a3-0.parquet (rows: 4, size: 1021 bytes)
Appended data from file: ./output_partitioned_parquet\group=C\c71bdad3-d644-4f4f-8c3f-70418e650743-0.parquet (rows: 3, size: 1009 bytes)

--- Iceberg Table Metadata ---
Schema:
table {
  1: id: optional int
  2: group: optional string
  3: value1: optional float
  4: value2: optional int
}

Snapshot History:
snapshot_id=3456922592047266445 timestamp_ms=1762768388714
snapshot_id=4420365441817198852 timestamp_ms=1762768389044
snapshot_id=5409131895043732158 timestamp_ms=1762768389318
```

```
(.venv) E:\XCaliber\iceberg-pipeline>python read_history.py
Table Metadata Location:
Snapshot History:
snapshot_id=3456922592047266445 timestamp_ms=1762768388714
snapshot_id=4420365441817198852 timestamp_ms=1762768389044
Snapshot History:
snapshot_id=3456922592047266445 timestamp_ms=1762768388714
snapshot_id=4420365441817198852 timestamp_ms=1762768389044
snapshot_id=5409131895043732158 timestamp_ms=1762768389318
Snapshot History:
snapshot_id=3456922592047266445 timestamp_ms=1762768388714
snapshot_id=4420365441817198852 timestamp_ms=1762768389044
snapshot_id=3456922592047266445 timestamp_ms=1762768388714
snapshot_id=4420365441817198852 timestamp_ms=1762768389044
snapshot_id=5409131895043732158 timestamp_ms=1762768389318
```

```
(.venv) E:\XCaliber\iceberg-pipeline>python query_iceberg_duckdb.py
Latest metadata file: iceberg_warehouse/default/my_iceberg_table/metadata/00003-c715acc5-4776-487d-baa6-d3e9a298b2c3.metadata.json
(.venv) E:\XCaliber\iceberg-pipeline>python query_iceberg_duckdb.py
Latest metadata file: iceberg_warehouse/default/my_iceberg_table/metadata/00003-c715acc5-4776-487d-baa6-d3e9a298b2c3.metadata.json
Latest metadata file: iceberg_warehouse/default/my_iceberg_table/metadata/00003-c715acc5-4776-487d-baa6-d3e9a298b2c3.metadata.json
Using metadata file for scan: iceberg_warehouse/default/my_iceberg_table/metadata/00003-c715acc5-4776-487d-baa6-d3e9a298b2c3.metadata.json
Iceberg Table Query Result:
   id group  value1  value2
Using metadata file for scan: iceberg_warehouse/default/my_iceberg_table/metadata/00003-c715acc5-4776-487d-baa6-d3e9a298b2c3.metadata.json
Iceberg Table Query Result:
   id group  value1  value2
Iceberg Table Query Result:
   id group  value1  value2
0   4     C     0.3       6
1   7     C     0.6      12
2   8     C     0.7      14
3   1     A     0.0       0
4   3     A     0.2       4
5   6     A     0.5      10
6  10     A     0.9      18
7   2     B     0.1       2
8   5     B     0.4       8
9   9     B     0.8      16
```


iceberg-pipeline
.venv
.venvv
iceberg_warehouse\default\my_ic...
data
group=A
group=B
group=C
metadata
output_partitioned_parquet
group=A
f15edbef-fddf-427c-8803-ed069...
group=B
group=C
catalog.db
large_dataset.parquet
main.py
query_iceberg_duckdb.py
1
read_history.py
sample_parquet.py