Data Lakes in ClickHouse

DeltaLake, Iceberg, Hudi

What are Deltalake, Iceberg or Hudi?

- Storage layers, independent from underlying storage (AWS S3, HDFS, Local). Provide:
  - Transactions
  - Partitioning
  - Data mutation
  - Data multiversion, quick rollback to previous versions
  - Advanced metadata with statistics
- Differ in transaction models, multiversion and data streaming support
- Engines like spark, presto, hive can work with the data
- Use .parquet format by default

How we support them in ClickHouse?

- Table function:
  - SELECT * FROM deltaLake(...), hudi(...), iceberg(...)
- Table engine:
  - CREATE TABLE data_lake ENGINE = DeltaLake/Hudi/Iceberg(...);
  - SELECT * FROM data_lake;
- Currently work only on top of s3:
  - deltaLake(<s3_configuration>)
  - hudi(<s3_configuration>)
  - iceberg(<s3_configuration>)
- Configuration - the same as for S3 engine.

Let’s compare performance!

- Dataset: UK Price dataset (27 million rows)
- Loaded via Spark in DeltaLake and Iceberg formats
- Underlying format .parquet
- Backended by S3
- Comparing only IO performance
  - Read all (SELECT *)
  - Read minimum (SELECT count())

Let’s compare: plain S3

```
ip-172-31-4-213.eu-west-1.compute.internal :) select count() from  s3('http://s3.eu-west-1.amazonaws.com/kssenii-public/house.parquet')

SELECT count()
FROM s3('http://s3.eu-west-1.amazonaws.com/kssenii-public/house.parquet')

Query id: f061a5c9-3d95-47d5-b7a8-9bd74c72e604

┌─count()─┐
│ 27734966 │
└─────────┘

1 row in set. Elapsed: 0.987 sec. Processed 27.73 million rows, 55.47 MB (28.09 million rows/s., 56.18 MB/s.)
```

```
ip-172-31-4-213.eu-west-1.compute.internal :) select * from  s3('http://s3.eu-west-1.amazonaws.com/kssenii-public/house.parquet') format Null

SELECT *
FROM s3('http://s3.eu-west-1.amazonaws.com/kssenii-public/house.parquet')
FORMAT `Null`

Query id: 3341851e-5af3-4506-99e6-3767d14629f5

Ok.

0 rows in set. Elapsed: 6.700 sec. Processed 27.73 million rows, 5.60 GB (4.14 million rows/s., 836.09 MB/s.)
```


Let’s compare: Iceberg

```
ip-172-31-4-213.eu-west-1.compute.internal :) select count() from iceberg('http://s3.eu-west-1.amazonaws.com/kssenii-public/iceberg_house_result/default/iceberg_table/')

SELECT count()
FROM iceberg('http://s3.eu-west-1.amazonaws.com/kssenii-public/iceberg_house_result/default/iceberg_table/')

Query id: bf771930-9e61-47f9-b516-94bd6e27d0d3

┌─count()─┐
│ 27734966 │
└─────────┘

1 row in set. Elapsed: 0.536 sec. Processed 27.73 million rows, 138.67 MB (51.76 million rows/s., 258.78 MB/s.)
```

```
ip-172-31-4-213.eu-west-1.compute.internal :) select * from iceberg('http://s3.eu-west-1.amazonaws.com/kssenii-public/iceberg_house_result/default/iceberg_table/') format Null

SELECT *
FROM iceberg('http://s3.eu-west-1.amazonaws.com/kssenii-public/iceberg_house_result/default/iceberg_table/')
FORMAT `Null`

Query id: 3e3e7c06-ed3b-4ec6-841d-d1e8caf13b1f

Ok.

0 rows in set. Elapsed: 5.927 sec. Processed 27.73 million rows, 5.74 GB (4.68 million rows/s., 968.64 MB/s.)
```


Let’s compare: DeltaLake

```
ip-172-31-4-213.eu-west-1.compute.internal :) select count() from deltaLake('http://s3.eu-west-1.amazonaws.com/kssenii-public/delta_house_result')

SELECT count()
FROM deltaLake('http://s3.eu-west-1.amazonaws.com/kssenii-public/delta_house_result')

Query id: b15c4aa9-ef49-4200-88b6-13e039060bf2

┌─count()─┐
│ 27734966 │
└─────────┘

1 row in set. Elapsed: 0.418 sec. Processed 27.73 million rows, 83.20 MB (66.32 million rows/s., 198.95 MB/s.)
```

```
ip-172-31-4-213.eu-west-1.compute.internal :) select * from deltaLake('http://s3.eu-west-1.amazonaws.com/kssenii-public/delta_house_result') format Null

SELECT *
FROM deltaLake('http://s3.eu-west-1.amazonaws.com/kssenii-public/delta_house_result')
FORMAT `Null`

Query id: a80330e0-5c10-4089-8499-e48df1283ed0

Ok.

0 rows in set. Elapsed: 5.179 sec. Processed 27.73 million rows, 5.69 GB (5.36 million rows/s., 1.10 GB/s.)
```


What's next

- Use checkpoints to speed up reading of metadata (DeltaLake)
- Support querying specific snapshots/versions (Iceberg, Delta)
- Support for partitioning (optimize reading by splitting metadata and data reading per partition)
- Use the advanced metadata to speed up queries (statistics)
- Support Iceberg V2 (unlike V1 supports row level deletes and updates)
- Support local and hdfs storage as underlying storage apart from s3

Thank you for attention