



Cloudera Open Data Lakehouse

NDA & SAFE HARBOR COMPLIANCE REQUIRED



The information in this document is proprietary to Cloudera. No part of this document may be reproduced or disclosed without the express prior written permission of Cloudera.

The information in this document is our currently intended developments and functionalities of Cloudera products which may change without notice at Cloudera's discretion. Cloudera makes no commitments about any future developments or functionality in any Cloudera product. The development, release, and timing of release of any software features or functionality described in this document remains at the discretion of Cloudera and you should not rely on any statements about development plans or anticipated functionality in this document when making any purchasing decisions.

Agenda

5	Kick off	<ul style="list-style-type: none">- Agenda- Goals / Objectives- Expectations
40	Intro to Cloudera Open Data Lakehouse	<ul style="list-style-type: none">- What is Cloudera Open Data Lakehouse?- Why Iceberg for Open Data Lakehouse?- Features, value, benefits- Roadmap
40	Iceberg Architecture Deep Dive	<ul style="list-style-type: none">- Architecture- Snapshots- Partition Evolution- Default Storage Location- Table Mutations (COW / MOR)- Best Practices
10	Q & A	
10	Break	
10	Hands on labs	Logistics
20	- Key feature demonstrations	<ul style="list-style-type: none">- Performance- Migration from Hive to Iceberg table format- Schema / Partition Evolution- Time Travel- SDX integration (FGAC, lineage)

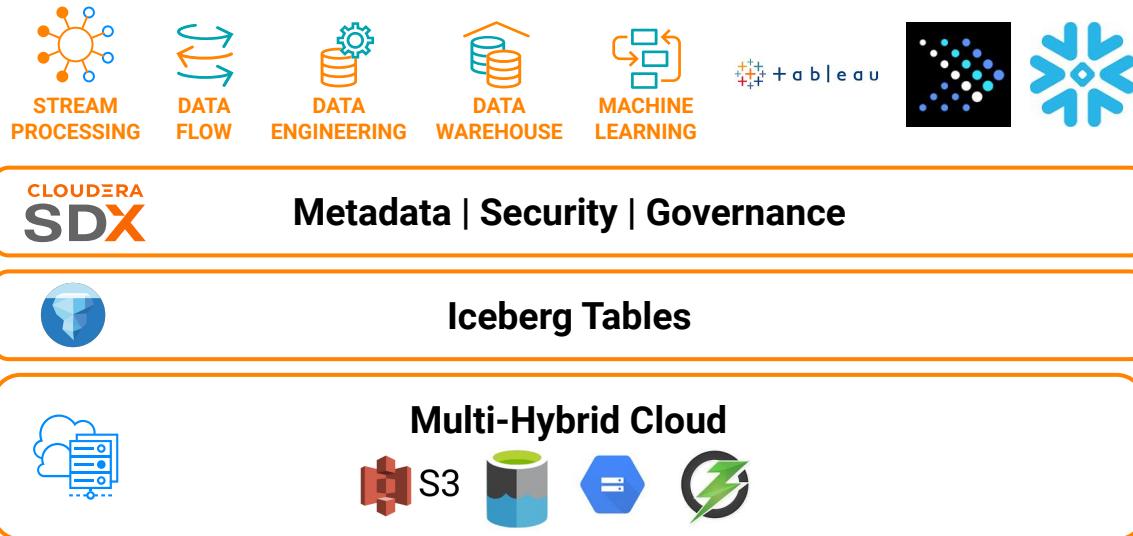
Agenda

	- Ingestion (DiM)	- Ingest data from s3 into an existing Iceberg Table - Ingest CDC data into an existing Iceberg Table to show how we can support Type 2 Slowly changing dimension tables
20	- Processing (CDE, CDW)	- Migrate existing tables to Iceberg - Creating Iceberg Tables - Load data into an Icebeg table - Schema Evolution - Partition Evolution - Time Travel
20	- Consumption (CDW, CML)	- Query Iceberg Tables - Setup simple Data Viz Dataset - Create simple Data Viz Dashboard - Create simple CML data processing python code to read data from Iceberg into a DataFrame for further processing
10	- Recap	Recap & Observability
	Q & A	

Cloudera Open Data Lakehouse BI, AI, and Beyond

Platform architecture delivers secure interoperability to any infrastructure cost-effectively

- Multi-function analytics for Streaming, Data Engineering, BI, & AI/ML
- Choose your engine: Cloudera integrated data services or 3rd party technologies
- SDX: Common security and governance with data lineage
- One Dataset: With all Compute engines. No data duplication or movement
- Deployment freedom with Multi-Hybrid Cloud on CPU & GPU Containers



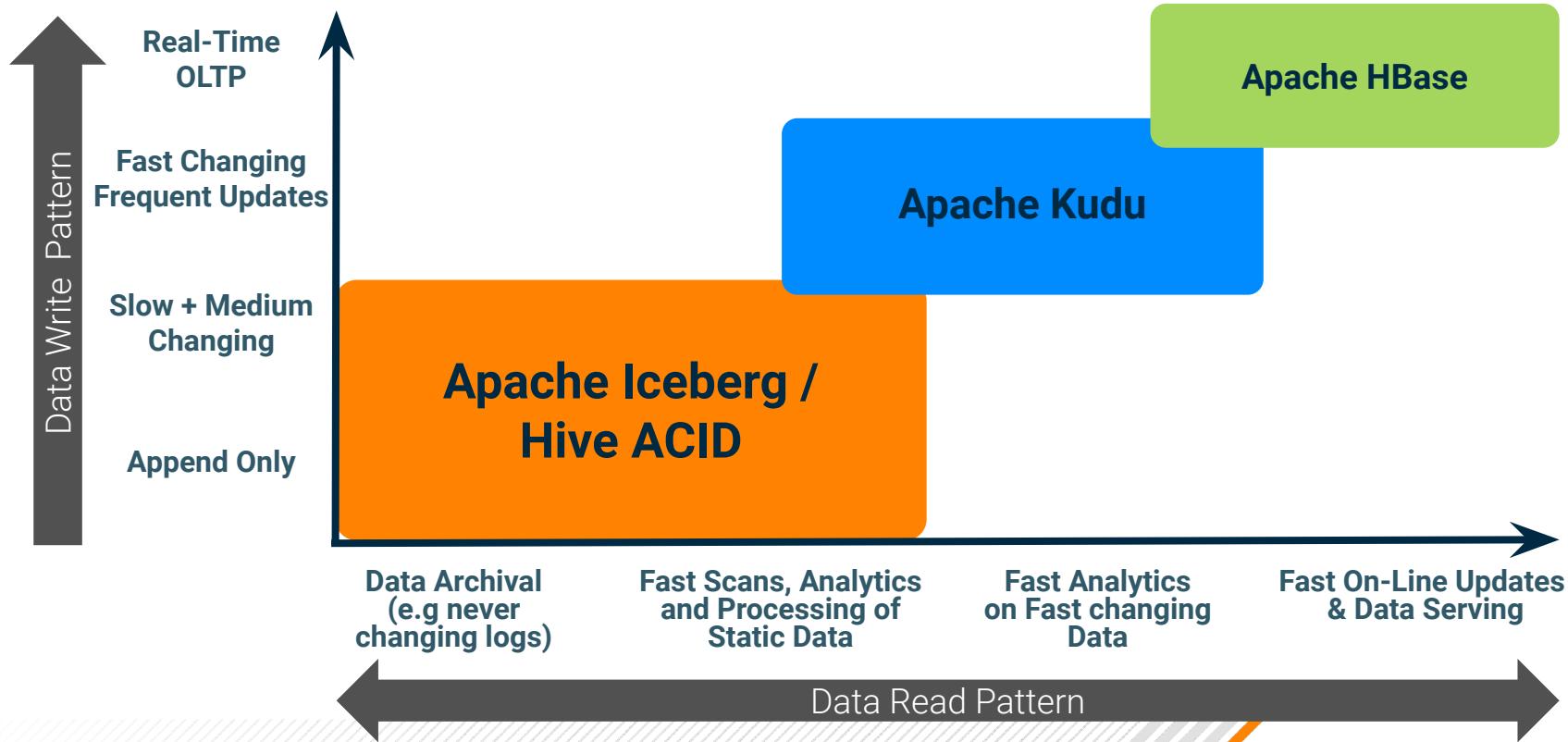
APACHE ICEBERG

A Flexible, Performant & Scalable Table Format

- Donated by **Netflix** to the Apache Foundation in 2018
- Flexibility
 - Hidden partitioning
 - Full schema evolution
- Data Warehouse Operations
 - ACID Transactions
 - Time travel and rollback
- Supports best in class SQL performance
 - High performance at Petabyte scale

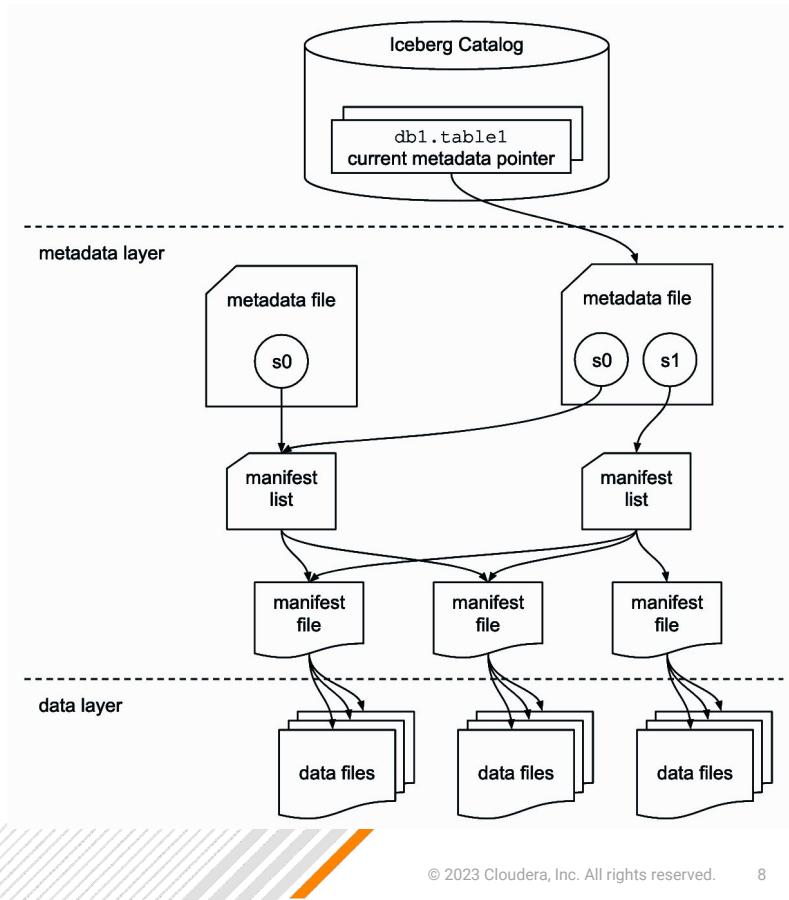


Where Apache Iceberg fits in the CDP Ecosystem

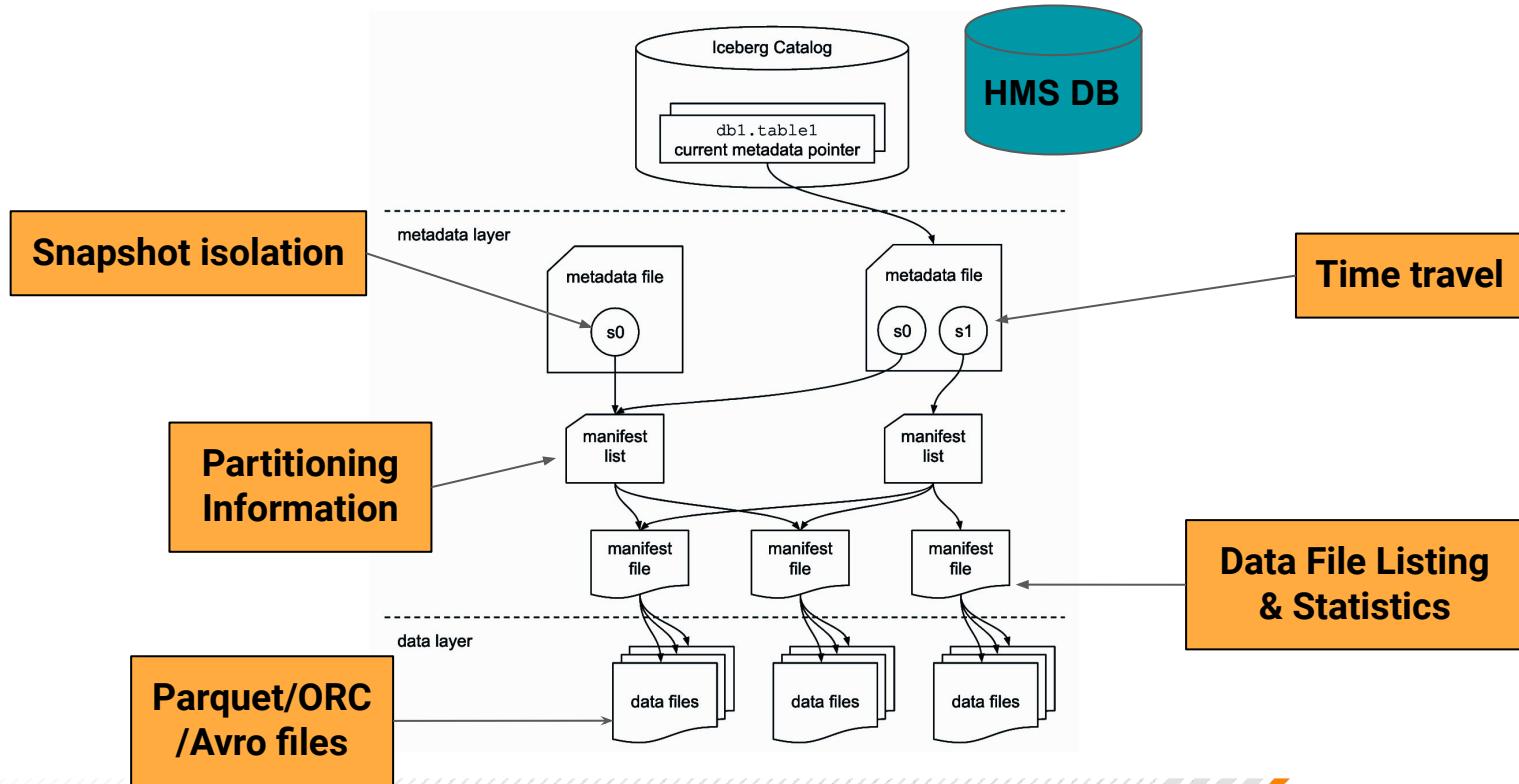


What is Iceberg Table Format?

- Iceberg is a new Big Data scale **Table Format** that specifies how a table's files are organized in a file system or object store
- Iceberg offers advanced features and performance improvements over the default table format
- 3 file formats supported - Parquet, Avro, and ORC
 - Supported by Impala, Hive, Spark, Flink, etc



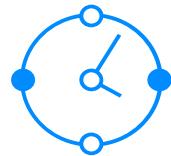
How Does Iceberg Format Solve the Issues



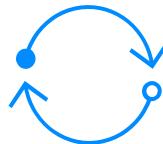
KEY FEATURES



ACID
Transaction



Time Travel /
Table Rollback



In-place Table
Evolution

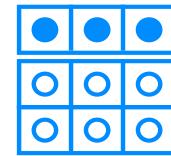
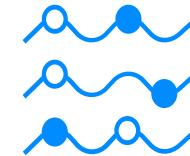
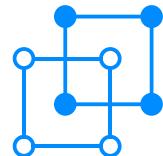


Table
Maintenance



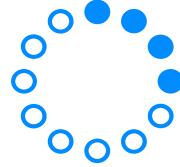
Streaming &
Batch Ingestion



Iceberg
Replication



SDX
Integration



Ease of
Adoption



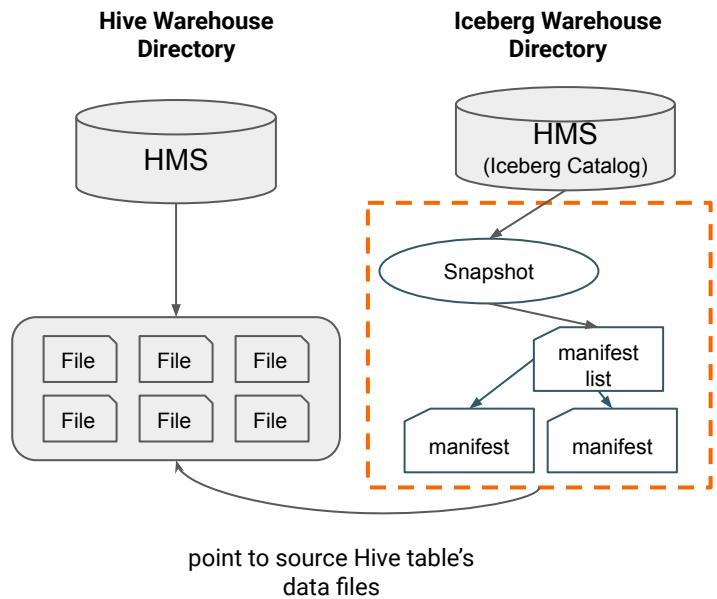
Multi-Hybrid
Cloud



Performance
/ Scalability

Table Migration

Table Migration In-Place



Avoids rewriting data files, just **re-write** the metadata

- **Hive table migration:**

```
ALTER TABLE tbl SET TBLPROPERTIES  
('storage_handler'='org.apache.iceberg.mr.hive.Hive  
IcebergStorageHandler')
```

- **Spark 3:**

- a. Import Hive tables into Iceberg

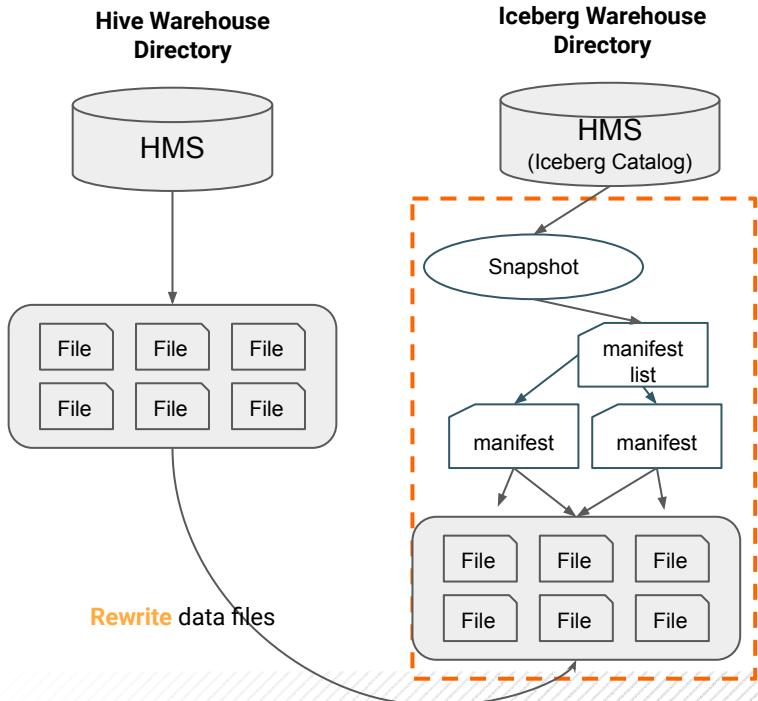
```
spark.sql("CALL  
<catalog>.system.snapshot('<src>', '<dest>')")
```

- b. Migrate Hive tables to Iceberg tables

```
spark.sql("CALL  
<catalog>.system.migrate('<src>')")
```

Table Migration **CTAS / RTAS**

Data files **recreated** in addition to creation of Iceberg tables and corresponding metadata



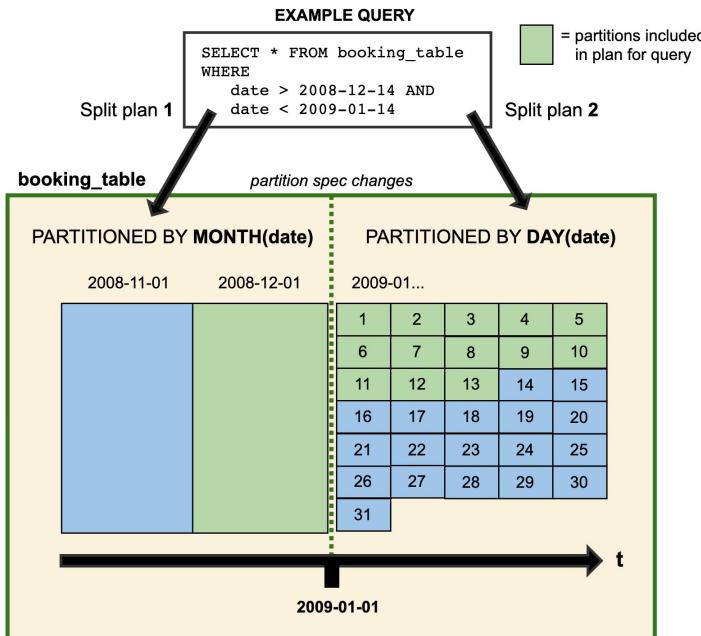
```
CREATE TABLE ctas PARTITIONED BY(z) STORED  
BY ICEBERG AS SELECT x, y FROM t;
```

```
spark.sql("CREATE TABLE ctas PARTITIONED  
BY(z) USING iceberg AS (SELECT x, y FROM  
t)")
```

```
spark.sql("REPLACE TABLE rtas PARTITIONED  
BY(z) USING iceberg AS (SELECT x, y FROM  
t)")
```

Partition Evolution

Iceberg Partition Evolution



```
ALTER TABLE t1 SET PARTITION SPEC (hour(ts))  
ALTER TABLE t1 ADD PARTITION FIELD (month(ts))
```

- Existing big data solution doesn't support **in-place** partition evolution. Entire table must be completely rewritten with new partition column
- With Iceberg's **hidden partition**, a separation between physical and logical, users are not required to maintain partition columns.
- Iceberg tables can **evolve partition** schemas over time as data volume changes.
- **Benefits:**
 - No costly table rewrites or table migration
 - No query rewrites
 - Reduce downtime and improve SLA

PARTITION EVOLUTION

From less to more granular

Partitioned by year:

- 2015
- 2016
- 2017
- 2018
- 2019
- 2020
- 2021
- 2022
- 2023



Partitioned by year, month:

- 2015
 - 2016
 - 2017
 - 2018
 - 2019
 - 2020
 - 2021
 - 2022
 - 2023
- June
 - July
 - August
 - September
 - October
 - November
 - December



Partitioned by year, month, day:

- 2015
 - 2016
 - 2017
 - 2018
 - 2019
 - 2020
 - 2021
 - 2022
 - 2023
- June
 - July
 - August
 - September
 - October
 - November
 - December
- 20
 - 21
 - 22
 - 23
 - 24
 - 25

Table structure, partitioned by year:

```
CREATE TABLE transaction (id INT,  
time TIMESTAMP, ...)  
PARTITIONED BY YEARS(time);
```

Adding new partitions. First, break down by month, then by day:

```
ALTER TABLE transaction ADD  
PARTITION FIELD MONTH(time)
```

```
ALTER TABLE transaction ADD  
PARTITION FIELD DAY(time)
```

Time Travel

TIME TRAVEL

Two rollback and time travel options: snapshot-id and as-of-timestamp

TIME TRAVEL



Audit data changes (BMS)

- Creates automatic snapshots of past data
- History of all operations are recorded for audits

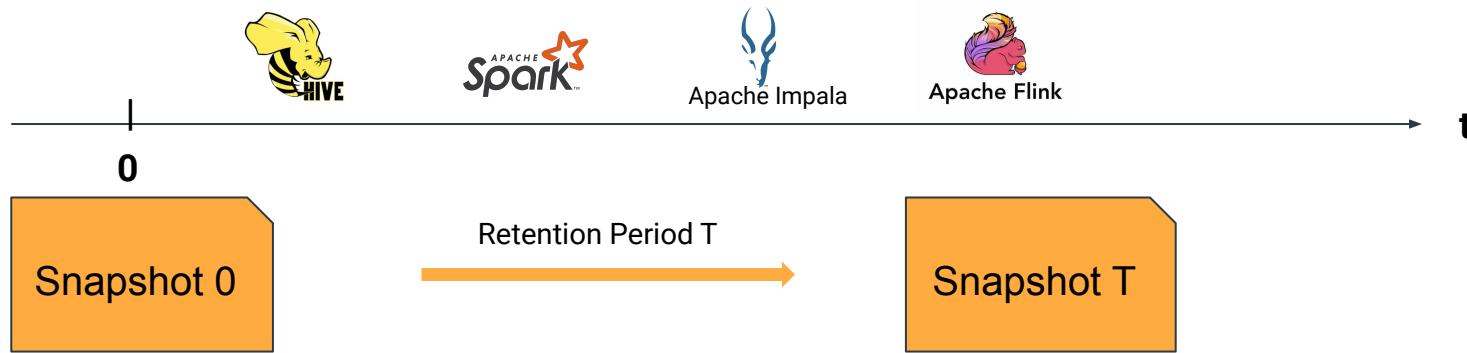
Data Reproducibility

- Retrain ML models with past data
- Generate financial reports faithfully, on past data

Rollback: Eliminate Data Errors

- Data pipeline debugging
- Rollback to any point to find source of bad data
- Root cause analysis
- Update, replace, or delete bad data

Time Travel & Table Rollback



- **Time Travel** enables reproducible queries that use exactly the same table snapshot, or lets users easily examine changes
 - `SELECT * FROM table FOR SYSTEM_TIME AS OF '2021-08-09 10:35:57';`
 - `SELECT * FROM table FOR SYSTEM_VERSION AS OF 1234567;`
 - `ALTER TABLE table EXECUTE ROLLBACK ('2021-08-09 10:35:57')`
 - `ALTER TABLE table EXECUTE ROLLBACK (1234567)`
- **Rollback** allows users to quickly correct problems by resetting tables to a good state

INSERT OVERWRITE VS. MERGE INTO

With Iceberg v2 format support (row-level deletes / updates), for **slow-changing tables' change data capture (CDC)**, **MERGE INTO** is recommended instead of **INSERT OVERWRITE** as **MERGE INTO** can replace only the affected data files instead of partitions.

SQL MERGE FOR CDC IN SCD TABLES

	SQL Examples
HIVE	<pre>MERGE INTO <target table> AS T USING <source expression/table> AS S ON <boolean expression1> WHEN MATCHED [AND <boolean expression2>] THEN UPDATE SET <set clause list> WHEN MATCHED [AND <boolean expression3>] THEN DELETE WHEN NOT MATCHED [AND <boolean expression4>] THEN INSERT VALUES <value list></pre>
SPARK	<pre>MERGE INTO <target table> t USING <source expression/table> s ON <boolean expression1> WHEN MATCHED [AND <boolean expression2>] THEN UPDATE SET <set clause list> WHEN MATCHED [AND <boolean expression3>] THEN DELETE WHEN NOT MATCHED [AND <boolean expression4>] THEN INSERT VALUES <value list></pre>

Row-Level Changes on the Lakehouse: COW vs MOR

Iceberg table has two different table formats v1 & v2.

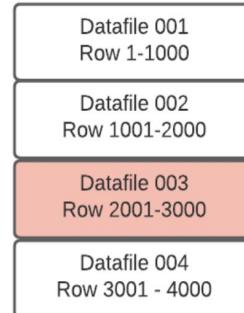
- v1 format — defaults **copy-on-write**
- v2 format — **copy-on-write or merge-on-read.**

Copy-on-Write (COW)

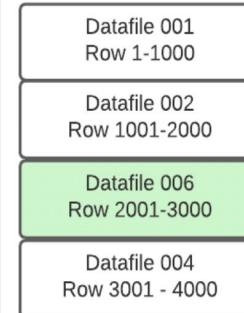
- With COW, when a change is made to delete or update a particular row or rows, the datafiles with those rows are duplicated, but the new version has the updated rows.
- This makes writes slower depending on how many data files must be re-written.
- If updating a large number of rows, COW is ideal.
- However, if updating just a few rows you still have to rewrite the entire data file, making small or frequent changes expensive.

[DELETE FROM table where id = 2585](#)

Before Update



After Update



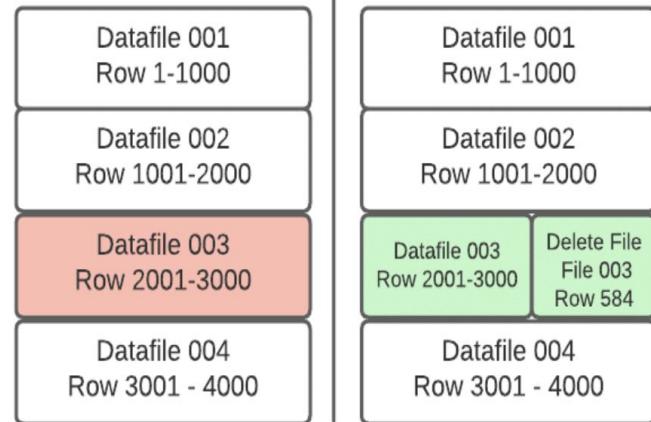
Summary of COW (Copy-On-Write)	
PROS	CONS
Fastest reads	Expensive writes
Good for infrequent updates/deletes	Bad for frequent updates/deletes

Merge-on-Read (MOR)

- With merge-on-read, **the file is not rewritten, instead the changes are written to a new file.**
- Then when the data is read, **the changes are applied or merged to the original data file** to form the new state of the data during processing.
- This makes writing the changes much quicker, but also means more work must be done when the data is read.
- If you **delete a row**, it gets added to **a delete file** and reconciled on each subsequent read till the files undergo compaction.
- If you **update a row**, that row is tracked **via a delete file** so future reads ignore it from the old data file and the updated row is added to a new data file.
- Again, once **compaction** is run, **all the data will be in fewer data files and the delete files will no longer be needed.**

DELETE FROM table where id = 2585

Before Update



After Update

Types of Delete Files

Position Deletes:

When Deleting a record, the file is read to determine its position in the file and the position of all deleted records (within files of a certain partition) are tracked in a delete file.

```
001.parquet 0  
001.parquet 5  
001.parquet 20  
006.parquet 5  
006.parquet 9  
006.parquet 100  
009.parquet 8  
009.parquet 10
```

Equality Deletes:

Instead of reading the file to identify the position of the deleted record, equality deletes just track specified filter predicates from the query and then matches them during the read. Faster during write but slower during read than position deletes.

i	id
5	
7	
38	
50	
90	

ROW-LEVEL MUTATION

	Change actions		Speed		
Mutation Strategy	DELETIONS	UPDATES	Read	Write	Recommendations
COW	The entire file is rewritten excluding the deleted rows	The whole file is rewritten with the updated rows	Fastest	Slowest	Recommended where changes to data are low frequency and reads performance is significantly more important
MOR (Equality deletes)	New files are created containing the row conditions to be deleted	New files are created just with the updated rows	Slow	Fastest	Requires frequent complications depending on how frequent are the updates/deletes.
MOR (Position deletes)	New files are created containing the row positions to be deleted	New files are created just with the updated rows	Fast	Fast	Requires frequent complications depending on how frequent are the updates/deletes.

ROW-LEVEL MUTATION



Icetip

Understand the use case needs - how data will be ingested, processed and consumed

Use case	Description	Requirements	Strategy
Batch processing	Process large chunks of files	Data can be consumed minutes/hours after processed	COW
Streaming analytics	(Near) real-time ingestion and processing	Data should be available immediately after ingestions	MOR, and run compactions to optimize reads; or consider Hbase or Kudu as better alternatives
Machine Learning	Run advanced analytics on static/batch data	Train ML models on vast datasets, creating new ones	COW
Real-time Business Intelligence	BI analytics for reporting and dashboards	Build snowflake or star schemas in 3NF	MOR, and run compactions to optimize reads

Table Maintenance with CDE / CML (Spark 3 SQL)

Table Maint.	Examples
Snapshot Management	<pre>// Remove snapshots older than one day, but retain the last 100 snapshots spark.sql("CALL <catalog>.system.expire_snapshots('db.sample', TIMESTAMP '2021-06-30 00:00:00.000', 100)") // Rollback a table to specific snapshot ID spark.sql("CALL <catalog>.system.rollback_to_snapshot('db.sample', 12345)") // Rollback a table to one day spark.sql("CALL <catalog>.system.rollback_to_timestamp('db.sample', TIMESTAMP '2021-06-30 00:00:00.000')")</pre>
Compaction	<pre>// Rewrite the data files in db.sample using the default rewrite // algorithm of bin-packing to combine small files and also split large // files according to the default write size of the table spark.sql("CALL <catalog>.system.rewrite_data_files('db.sample')")</pre>

Expiring old snapshots removes them from metadata, so they are no longer available for time travel operations. Data files are not deleted until they are no longer referenced by a snapshot that may be used for time travel. Regularly expiring snapshots deletes unused data files.

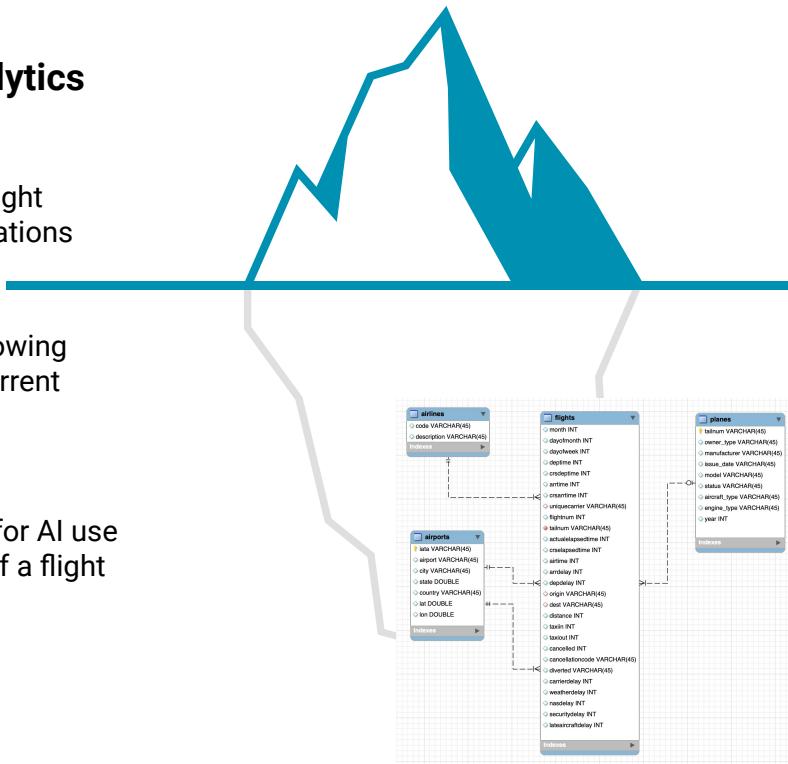
Lakehouse Hands-on Lab

Lab Overview

Try it out and experience the benefits of Iceberg first-hand

Airline Logistics Analytics Data Lakehouse

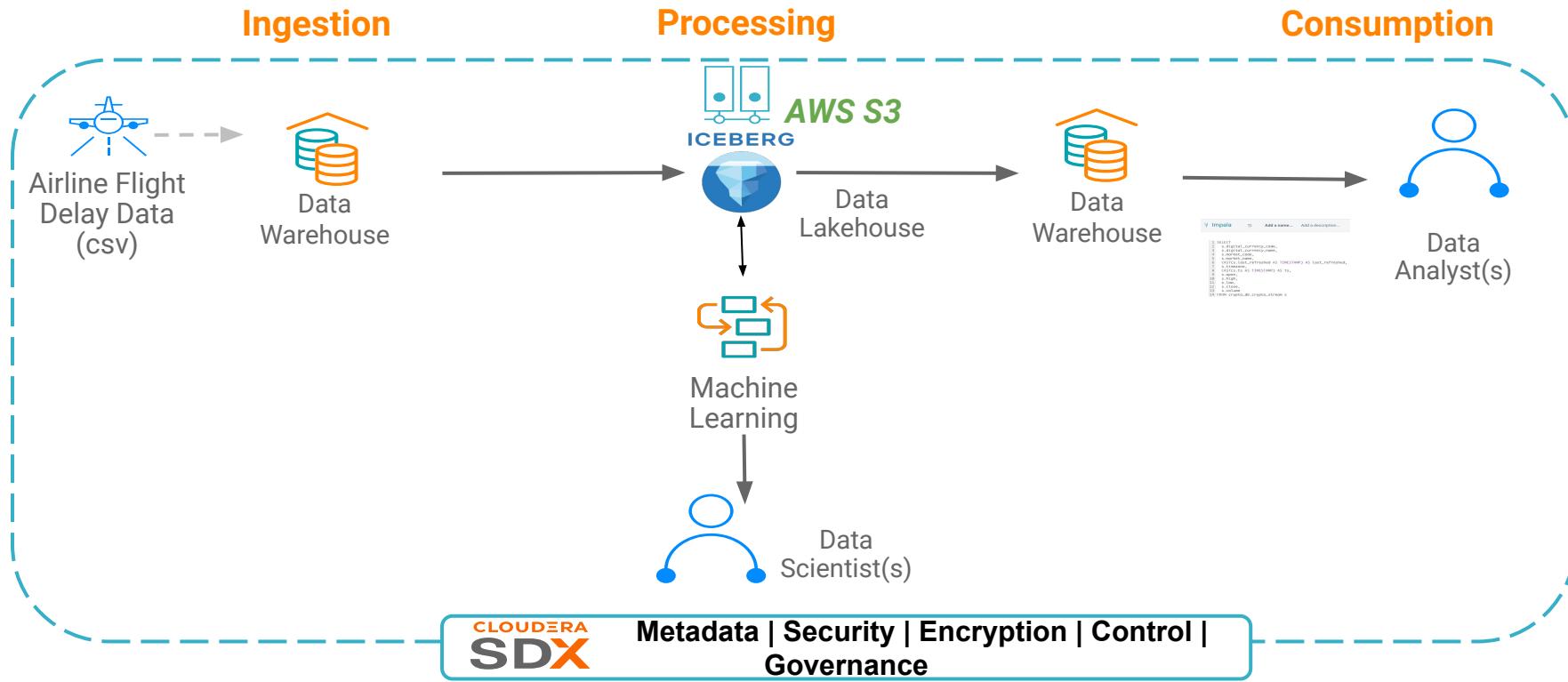
- Concerns with Flight Delays & Cancellations
 - Data volumes growing fast, stressing current solution(s)
 - Need to prepare for AI use cases to predict if a flight will cancel



What to Expect in the Labs

1. Improved performance
 2. Simplified management - schema/partition evolution
 3. Multi-function analytics (SQL, ML/AI) against ***single copy*** of data
 4. ACID - Insert data (update, delete and merge)
 5. Time Travel
 6. Security & Lineage - behind the scenes

Airline Logistics Details

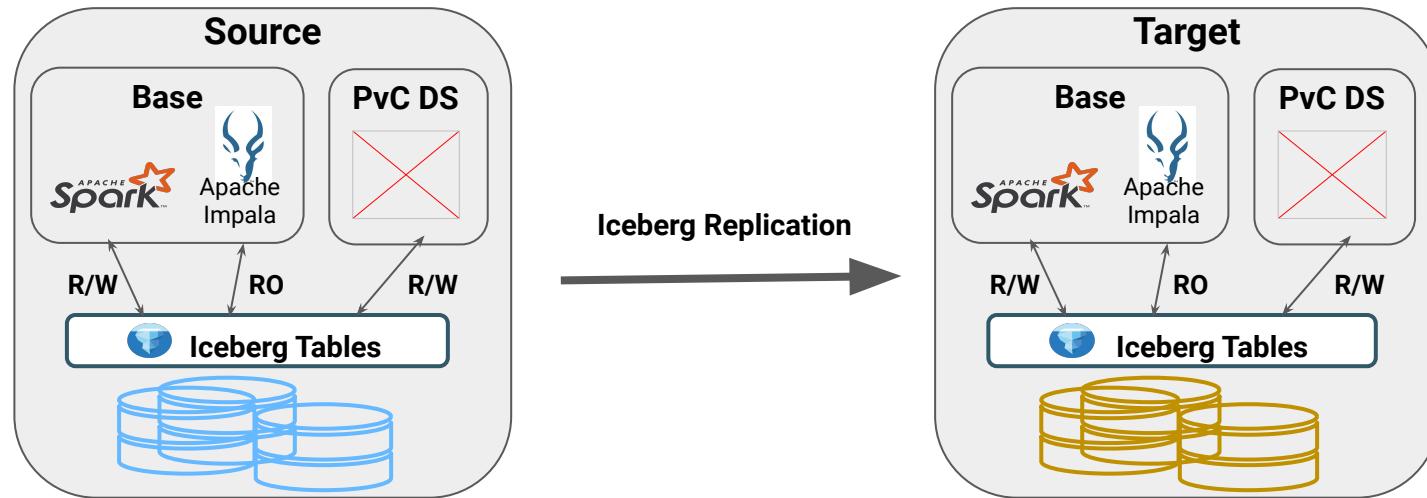


Q&A

Thank You

Iceberg Replication

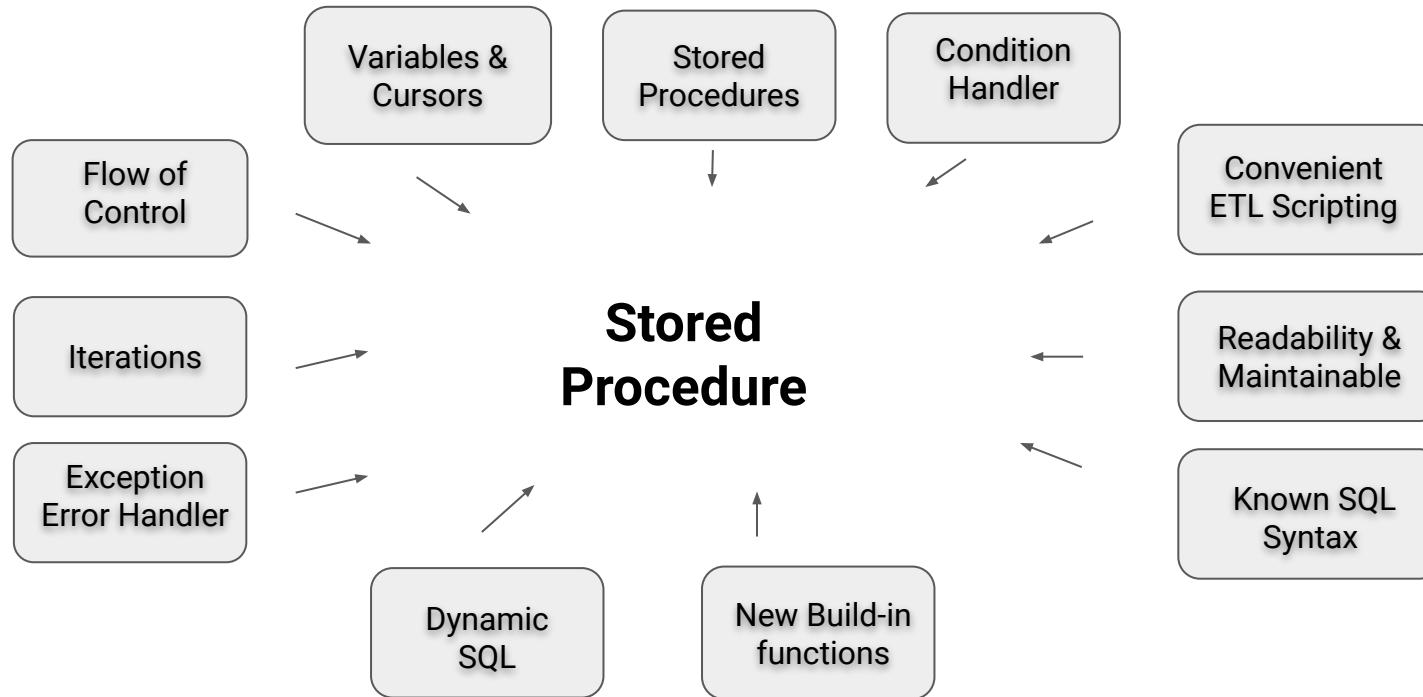
Iceberg Replication



- Bootstrap and Incremental Replication Support
- Iceberg Version V1/V2 support
- Table level replication scope with inclusion and exclusion filters

CDW Stored Procedure - HPL/SQL

CDW Stored Procedures - HPL/SQL

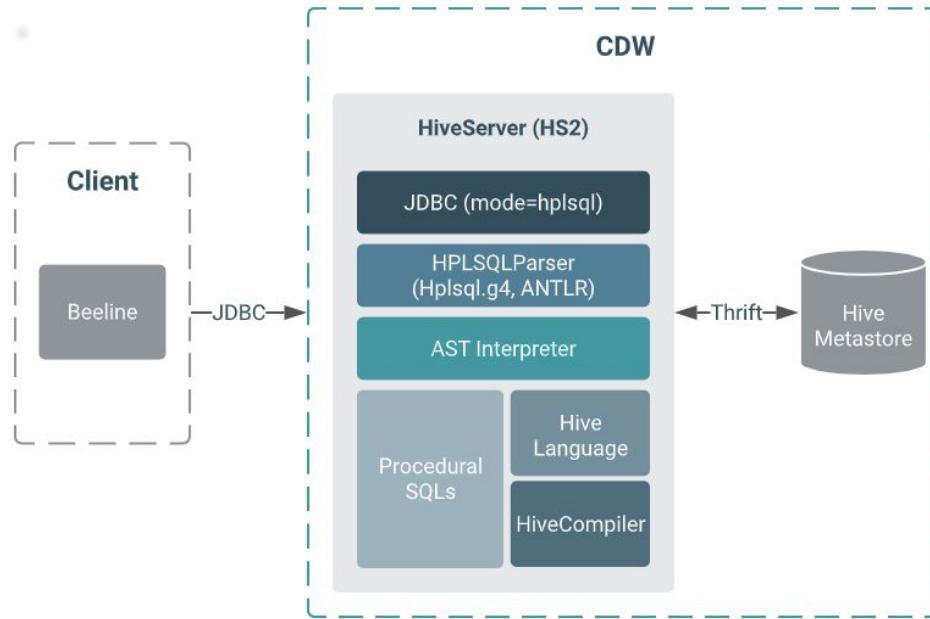


Stored Procedure Architecture

CDW Stored Procedure has been re-architected from a command line tool to an **integrated part of HiveServer (HS2)**.

The **interpreter** executes the abstract syntax tree (AST) from the parser.

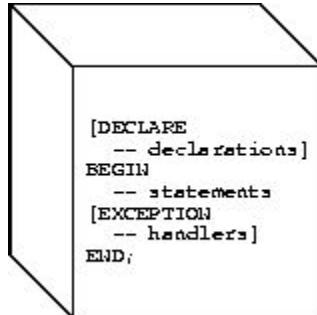
Hive metastore securely stores the function and procedure code permanently.



The procedure is loaded and cached on demand to the interpreter's memory when needed.

Procedural Language & SQL

CDW Stored Procedure is (partial) compatible with Oracle PL/SQL, IBM DB2, MySQL and Teradata.



```
create procedure hello(name STRING)  
begin  
    print 'Hello ' || name;  
end;  
  
call hello ('World!');
```

```
declare row_cnt      int = 0;  
declare rows         flights_csv%rowtype;  
declare cur cursor for SELECT * FROM flights_csv;  
  
OPEN cur;  
FETCH cur INTO rows;  
WHILE SQLCODE=0 THEN  
    row_cnt = row_cnt + 1;  
    FETCH cur INTO rows;  
END WHILE;  
CLOSE cur;  
  
PRINT 'row count ' || row_cnt;
```

The screenshot shows the Cloudera Manager interface with a focus on a HPL/SQL editor window. The left sidebar lists various datasets: airlines_csv, airlines_orc, airports_csv, airports_experiences, airports_orc, dept, flights_csv, flights_orc, planes_csv, planes_orc, unique_tickets_csv, and unique_tickets_orc. The main window title is "HPL/SQL". It includes a search bar at the top right and buttons for "Add a name..." and "Add a description...". The code area contains the following HPL/SQL script:

```
1 drop table if exists airports_experiences;
2 create table airports_experiences(iata string, delay_top_fli
3 begin
4 declare c_iata string;
5 declare c_anz int;
6 declare cur cursor as select origin, count(*) anz
7   from flights_orc
8   group by origin
9   order by anz desc
10  limit 100;
11 open cur;
12 FETCH cur INTO c_iata, c_anz;
13 WHILE SQLCODE=0 THEN
14   DBMS_OUTPUT.PUT_LINE( 'airport:' || c_iata || ' Anzahl'
15   CALL airport_experience.generate( c_iata);
16   FETCH cur INTO c_iata, c_anz;
17 END WHILE;
18 CLOSE cur;
19 END;
20 /
21 select * from airports_experiences;
22 drop table if exists airports_orc;
23 create table airports_orc as select * from airports_csv;
24
25 select
```