

# TABLE FORMAT NOTES

---

- Wire format
- Iceberg

Gronet A.



- Un buon Tablet format deve essere specifico, deve essere portatile e documentabile. Deve soddisfare:
1. ATOMIC CHANGES o che cambiate a nulla
  2. SCHEMA EVOLUTION senza conseguenze non volute
  3. EFFICIENT ACCESS come predici a proiezione in profondità.
  4. HIDDEN LAYOUT non c'è bisogno di sapere la struttura dei dati; (optional)
  5. LAYOUT EVOLUTION possibilità di cambiare la struttura nel tempo (optional)

## NIVE TABLES

L'idea è di organizzare i dati in struttura a catinelle, in cui le partizioni diventano catinelle con valori.

```
date=20180513/
|- hour=18/
|  |- ...
|- hour=19/
|  |- 000000_0
|  |- ...
|  |- 000031_0
|- hour=20/
|  |- ...
|- ...
```

In questo modo è possibile usare queste colonne come filtri.

Ad esempio, se faccio `date='20180513'` leggeremo solo quelle catinelle.

Ogni DB Hive ha un metastore in cui tiene traccia di: 1. posizioni 2. informazioni sullo schema 3. statistiche sulla tabella.

Esso permette di filtrare tramite valori di posizione, usare database esterni SQL, solo il file system traccia i file.

## HIVE ACID LAYOUT

Fornisce update atomici e isolati. Le transazioni sono salvate nel metastore. Usa lo stesso partition/partition layout.

## DESIGN PROBLEMS

- Lo stato delle tabelle è memorizzato in 2 posti: Metastore e filesystem
- Bucketing è definito dall'implementazione Hive nell'HDFS.
- L'unica operazione atomica del Non-Acid layout è l'aggiunta di partitioni.
- Richiede lo spostamento atomico degli oggetti nel filesystem.
- Richiede la lista delle diretive per pianificare i job
- I valori delle partitioni sono salvati come stringhe: - Richiede Escape - will salvati come `_HIVE_DEFAULT_PARTITION`
- Le statistiche diventano obsolete e devono essere rigenerate manualmente
- Molte varianti di layout non documentati
- Le definizioni di Bucket sono affidate a Hive e HDFS
- L'utente deve conoscere il layout fisico
- Schema Evolution dipende dal formato del file
- I tipi cambiano anch'essi a seconda dei formati.

## ICEBERG TABLES

L'idea è quella di tenere traccia di ogni file in una tabella nel corso del tempo:

- Uno snapshot è una lista completa di file in una tabella
- Ogni scrittura e commit creano un nuovo snapshot.

I benefici dello snapshot sono diversi:

- Isolamento dello snapshot senza lock: - Readers usano lo snapshot corrente - Writer crea nuovi snapshot
- Ogni cambio alla lista dei file è un'operazione atomica: - Aggiunta ed rimozione - Merge o riscrittura file

## ICEBERG METADATA

Implementa il monitoraggio basato su snapshot: - Aggiunge table schema, posizioni e tracce  
- traccia i vecchi snapshot per garbage collection

- Ogni file dei metadati è immutabile

- Metadati di riga sempre in ordine, lo storico è lineare

- Lo snapshot corrente può essere rolled back.

## MANIFEST FILES

Gli snapshot sono diritti in 1 o più manifest files: - Un manifest salva i file sulle varie posizioni

- Una tuta di dati di partizione sono salvati per ogni file
- Riutilizzati per evitare altri volumi di scrittura.

Un manifest file contiene:

- Info sul file: Location, formato, iceberg tracking data
- Valori per filtrare: valori partizione, lower e upper bounds per colonna
- Metadati per ottimizzare: FILE LEVEL - count e size - COLUMN LEVEL - count, null, size

## Commits

Prima di committere un writer deve:

- Segnare la versione dei metadati anziché
- Creare un nuovo file di metadati e 2 manifest
- Cambiare la versione con la nuova atomicamente

Questo permette di avere uno storico lineare. Lo swap è implementato da un metastore custom ed un rename in HDFS.

Quando ci sono conflitti si usa un writer ottimistico. Si assume che nessun altro stia scrivendo e in caso di conflitti si sovrasta con gli ultimi metadati.

## BENEFICI

- Non ci sono operazioni onerose ai File System
- Lettura e scrittura sono isolate ed atomiche.
- Scan più veloci: O(1) manifest per la lettura, non O(h) partition list
- Full Schema Evolution - Null partitioning
- Sopporta per i tipi - Sopporta per diversi file format

METADATA → immutable! OR

Feature of Delta → Non è Vacuum

multiple table → No conflict

few reads → Merge on read

non è untable format → parquet

SQL covering → info in metadata

transaction log - JSON - version

metadata/index/shema - ?