Shwetank Singh
GritSetGrow - GSGLearn.com

DATA AND AI

LAKEHOUSE TABLE FORMATS (DELTA/ICEBERG/HUDI)

FEATURES, TIME TRAVEL, COMPACTION

www.gsglearn.com

Lakehouse Table Formats --- Delta vs Iceberg vs Hudi (Features - Time Travel - Compaction)

> ***Goal:*** *Give you a practical, side-by-side guide to the three leading lakehouse table formats---**Delta Lake**, **Apache Iceberg**, and **Apache Hudi**---with a focus on **features**, **time travel**, and **compaction/garbage collection**. Includes how-to snippets, operational runbooks, and decision tips.*

> ***Scope:*** *ANSI-first ideas with examples in Spark SQL where possible, plus system procedures for Iceberg and common config keys for Hudi. Exact syntax may vary by engine/catalog.*

---

1) Mental Model

All three formats layer **ACID transactions** and **table metadata** on top of object storage (e.g., S3/ABFS/GS). They differ mainly in how they handle **metadata**, **evolution**, **upserts/streaming**, and **ops**.

- **Delta Lake**: Transaction log (`_delta_log` JSON/Parquet) + data files. Optimistic concurrency. Strong Spark/Databricks ecosystem, growing engines support.
- **Apache Iceberg**: Snapshot-based metadata tree (manifests + metadata.json). Hidden partitioning, partition spec evolution, broad multi-engine support (Spark, Flink, Trino/Presto, Snowflake, Dremio, etc.).
- **Apache Hudi**: Commit timeline (instants) + record-level indexing. First-class **upserts** and streaming ingestion; **Copy-on-Write (COW)** and **Merge-on-Read (MOR)** table types.

---

2) Feature Matrix (quick reference)

Capability	Delta Lake	Apache Iceberg	Apache Hudi
ACID transactions	Yes (optimistic)	Yes (snapshot isolation)	Yes (timeline)
Schema evolution	Add cols, rename, type widen; constraints support	Rich evolution incl. partition spec; hidden partitioning	Add cols, evolve keys (with care), schema on write
Partition evolution	Limited (static spec; Z-order for clustering on platform)	**Yes**, change partition spec over time	Static partitions; clustering & layout via **clustering**
Time travel (read)	`VERSION AS OF` / `TIMESTAMP AS OF`	Read by **snapshot id** or timestamp via	Read at **instant** (commit timestamp)

		engine-specific syntax	
Rollback (write)	Restore to version; `VACUUM` governs files	`rollback_to_snapshot`, branches/tags	Rollback by committing revert, or point-in-time restore
Upserts/MERGE	Native `MERGE INTO`	`MERGE INTO` supported (engine-dependent)	**Core strength**; upsert by key with index
Streaming	Structured Streaming native	Flink/Spark streaming connectors	**First-class** streaming ingestion
Compaction	`OPTIMIZE` (+ optional Z-order on some platforms)	`rewrite_data_files` / `rewrite_manifests`	**Compaction (MOR)** + **Clustering (COW/MOR)** + **Clean**
GC / Cleanup	`VACUUM` (retention window)	`expire_snapshots` + `remove_orphan_files`	**Cleaner** (retain commits), **Archival**
Engines	Strong Spark; plus Trino/Presto, Synapse, others	Strong multi-engine	Spark/Flink/Presto/Trino (varies by feature)

> *Note: Some capabilities (e.g., Z-order, advanced OPTIMIZE UI) can be **platform-specific** even when the format is open.*

---

3) Time Travel --- Concepts & How-To

Time travel lets you **query past table states** for debugging, audits, and reproducibility.

3.1 Delta Lake --- query past versions

Read past version

```
-- By version
SELECT * FROM sales VERSION AS OF 123;
-- By timestamp
SELECT * FROM sales TIMESTAMP AS OF '2024-06-30T23:59:59Z';
```

Restore table (write rollback)

```
RESTORE TABLE sales TO VERSION AS OF 120;  -- or TO TIMESTAMP AS OF '2024-06-01'
```

Retention & safety

- Data files needed by time travel are kept until `VACUUM` removes them.
- Default retention is conservative (e.g., 7 days). Avoid disabling safety checks unless you fully understand implications.

3.2 Apache Iceberg --- snapshots, branches & tags

List snapshots (engine/catalog procedure pattern)

```
-- Show snapshots
CALL catalog.system.snapshots('db.sales');
```

Read a snapshot (engine-specific)

- Engines provide one or more of:
    - *Version/Time as of* functions
    - Table references (snapshot id / tag / branch)
    - Session properties

```
-- Example patterns (adjust to your engine)
SELECT * FROM db.sales /* VERSION AS OF 123456789 */;
-- or
SELECT * FROM db.sales /* TIMESTAMP AS OF '2024-06-30 23:59:59' */;
-- or
CALL catalog.system.set_current_snapshot('db.sales', 123456789);
```

Rollback / references

```
-- Roll back table pointer to a snapshot
CALL catalog.system.rollback_to_snapshot('db.sales', 123456789);
-- Manage branches/tags for long-lived references
CALL catalog.system.create_branch('db.sales', 'prod');
CALL catalog.system.create_tag('db.sales', 'release_2024_06');
```

Retention: governed by `expire_snapshots` and `remove_orphan_files` (see §5.2).

3.3 Apache Hudi --- instants on the timeline

Read as of a commit instant

```
-- Spark DataFrame options (typical)
SELECT * FROM hudi_sales /* AS OF INSTANT '20240630235959' */;  -- illustrative
-- With DataFrame API:
-- .option("as.of.instant", "20240630235959") on the reader
```

Restore/roll back

- Use the timeline to **rollback** failed writes or **restore** to a previous instant via CLI/DeltaStreamer configs.
  Retention controlled by cleaner/archival configs (see §5.3).

---

4) Compaction & Small Files --- Concepts

Object stores like S3/ABFS favor **large files** (e.g., 128--1024 MB). Many streaming/micro-batch jobs create **lots of small files**, which hurts scan performance and metadata overhead. Compaction strategies:

- **Bin-packing / file rewrite**: combine small files into larger ones.
- **Clustering (layout optimization)**: rewrite files sorted by key(s) to improve pruning/skipping.
- **Manifest rewrite** (Iceberg): reduce metadata overhead by coalescing manifests.
- **Vacuum/Clean**: remove unreachable or old versions according to retention.

---

5) Operations Runbook --- By Format

5.1 Delta Lake

Compact data files

```
-- Combine many small files into fewer large ones
OPTIMIZE sales;  -- optional WHERE predicate for partition(s)
-- On some platforms, order data for skipping
OPTIMIZE sales ZORDER BY (customer_id, order_date);
```

Vacuum (garbage collect old files)

```
-- Remove files older than retention (e.g., 7 days)
VACUUM sales RETAIN 168 HOURS;  -- 7 days
```

Typical schedule

- Daily/weekly `OPTIMIZE` on busy partitions (or after large backfills).
- `VACUUM` after sufficient retention to preserve time-travel needs.
  Cautions
- Do **not** set retention too low; you can irreversibly break time travel and restore.
- Z-order availability/behavior can be platform-specific.

5.2 Apache Iceberg

Rewrite data files (bin-packing)

```
-- Coalesce small files; choose strategy & target size
CALL catalog.system.rewrite_data_files(
  table => 'db.sales',
  options => map('min-input-files','5','target-file-size-bytes','536870912')
);
```

Rewrite manifests

```
CALL catalog.system.rewrite_manifests('db.sales');
```

Expire snapshots (GC)

```
CALL catalog.system.expire_snapshots(
  'db.sales',
```

```
  older_than => TIMESTAMP '2024-06-30 23:59:59',
  retain_last => 3
);
```

Remove orphan files

```
CALL catalog.system.remove_orphan_files(
  'db.sales', older_than => TIMESTAMP '2024-06-30 23:59:59'
);
```

Typical schedule

- Frequent `rewrite_data_files` for high-ingest tables.
- Periodic `rewrite_manifests` to curb metadata growth.
- `expire_snapshots` + `remove_orphan_files` based on audit/rollback needs.

5.3 Apache Hudi

Table types

- **COW**: writes create new Parquet files; reads are simple; compaction not needed but **clustering** helps layout.
- **MOR**: delta logs appended; **compaction** merges logs into base files to keep reads fast.

Inline/async compaction (MOR) --- key configs

```
hoodie.table.type = MERGE_ON_READ
hoodie.compact.inline = true
hoodie.compact.inline.max.delta.commits = 10     # after N delta commits, compact
hoodie.parquet.small.file.limit = 134217728      # 128 MB; avoid tiny files
```

Clustering (file layout optimization) --- both COW & MOR

```
hoodie.clustering.inline = true
hoodie.clustering.plan.strategy.class =
org.apache.hudi.client.clustering.plan.strategy.SparkRecentFileSliceClusteringPlanStrate

hoodie.clustering.execution.strategy.class =
org.apache.hudi.client.clustering.run.strategy.SparkSortAndSizeExecutionStrategy
hoodie.clustering.sort.columns = order_date, customer_id
hoodie.clustering.max.bytes.per.group = 536870912  # 512 MB targets
```

Cleaner & archival (GC)

```
hoodie.cleaner.policy = KEEP_LATEST_COMMITS
hoodie.cleaner.commits.retained = 20  # keep enough for time travel/rollback
hoodie.keep.min.commits = 30          # compaction/archival watermarks
hoodie.keep.max.commits = 60
```

Operational tips

- Schedule **compaction** off-peak for MOR; monitor read latency improvements.
- Tune **index** (Bloom/Global Bloom/HBase/Record-level) to balance upsert speed vs cost.

---

6) How They Evolve Data

- **Schema evolution**
  - *Delta*: add/rename columns; enforce constraints; requires `mergeSchema` or equivalent flags for writes that add columns.
  - *Iceberg*: robust schema + **partition spec evolution** without rewriting old data; hidden partitioning avoids user-side partition columns.
  - *Hudi*: supports schema changes; watch key evolution and payload classes when upserting.
- **Partitioning / Layout**
  - *Delta*: static partition columns; platforms add **Z-order** clustering for pruning.
  - *Iceberg*: hidden transforms (e.g., `bucket(id, 16)`, `truncate(col, 4)`, `day(ts)`), and can evolve the spec.
  - *Hudi*: partition path fields; **clustering** to re-sort/size files as data grows.

---

7) Choosing Guide (when to prefer which)

- **You need heavy upserts, fast incremental pulls, near-real-time ingestion** → **Hudi** (MOR), or **Delta** with strong Structured Streaming.
- **You need broad multi-engine analytics, partition spec evolution, and clean metadata at scale** → **Iceberg**.
- **You already run Databricks / Spark-centric workloads and want simple time travel & merges** → **Delta** is ergonomic.
- **Mixed workloads**: Delta & Iceberg both support MERGE/time travel; pick based on your catalog/engine ecosystem and governance needs.

---

8) Anti-Patterns & Safety Checks

- **Aggressive GC**: Running `VACUUM` / `expire_snapshots` / `clean` with very short retention can **destroy** the ability to audit/rollback.
- **Exploding small files**: Streaming with tiny batch sizes → schedule compaction and set target file sizes.
- **Non-deterministic partition transforms**: Avoid transforms that can't be reproduced across engines.

- **Blind schema overwrite**: Require schema compatibility checks in CI before promoting writes.

---

9) Example Playbooks

9.1 Backfill a month safely (Delta/Iceberg)

1. **Pause** downstream jobs or write to a **branch** (Iceberg) or new table version (Delta) for validation.
2. Load/backfill the month with reasonably large target files (256--512 MB).
3. Run **compaction** (Delta `OPTIMIZE`; Iceberg `rewrite_data_files`).
4. Validate counts/checksums vs source.
5. **Switch pointer** (Iceberg rollback/branch commit or promote Delta version).
6. Schedule GC only after audit sign-off.

9.2 Weekly maintenance window

- **Delta**: `OPTIMIZE` hot partitions → `VACUUM` with 7--30d retention.
- **Iceberg**: `rewrite_data_files` → `rewrite_manifests` → `expire_snapshots` (retain N).
- **Hudi**: for MOR, compaction + clean; for COW, clustering + clean.

9.3 Point-in-time investigation

- **Delta**: run `DESCRIBE HISTORY table` → query `VERSION AS OF` the target change.
- **Iceberg**: list `snapshots()` → read the snapshot/tag.
- **Hudi**: inspect timeline → query with `as.of.instant`.

---

10) Quick Syntax Cheat Sheet

Delta Lake

```
-- Time travel
SELECT * FROM t VERSION AS OF 42;
SELECT * FROM t TIMESTAMP AS OF '2024-06-30 23:59:59';
-- Compaction
OPTIMIZE t;  -- optionally ZORDER BY (k1, k2)
-- GC
VACUUM t RETAIN 168 HOURS;
```

Iceberg

```
-- Maintenance
CALL catalog.system.rewrite_data_files('db.t');
CALL catalog.system.rewrite_manifests('db.t');
```

```
CALL catalog.system.expire_snapshots('db.t', older_than => TIMESTAMP '2024-06-30
23:59:59', retain_last => 3);
CALL catalog.system.remove_orphan_files('db.t', older_than => TIMESTAMP '2024-06-30
23:59:59');
```

Hudi (configs)

```
# MOR compaction
hoodie.compact.inline=true
hoodie.compact.inline.max.delta.commits=10
# Cleaner/archival
hoodie.cleaner.policy=KEEP_LATEST_COMMITS
hoodie.cleaner.commits.retained=20
hoodie.keep.min.commits=30
hoodie.keep.max.commits=60
```

---

11) Frequently Asked

Q: Can I mix engines on the same table?
A: Iceberg is most mature for multi-engine. Delta/Hudi support is improving, but ensure your readers/writers use compatible versions and the **same catalog** semantics.

Q: What retention is safe?
A: Common is **7--30 days** depending on audit/SLA. Keep longer for regulated datasets. Always align GC to your **time travel needs** and backup cadence.

Q: Do I need both compaction and clustering?
A: Compaction fixes file size; clustering improves data skipping. Iceberg's bin-packing and partition spec evolution partially cover both; Hudi exposes clustering explicitly; Delta offers layout helpers on some platforms.

Q: How big should files be?
A: Typical targets: **256--1024 MB** Parquet depending on workload and engine parallelism.

---

12) Checklist Before You Go Live

- Define **primary keys/natural keys** for upserts (Hudi/Delta MERGE, Iceberg MERGE).
- Choose **partition spec** (or hidden transforms) aligned to top queries.
- Set **target file size** and compaction cadence.
- Decide **retention** for time travel & GC; document it.
- Wire **data quality** checks pre/post compaction.
- Capture **lineage** and **history** (Delta `DESCRIBE HISTORY`; Iceberg snapshots; Hudi timeline).
- Add **CI tests** for schema compatibility.

TL;DR

- **Delta**: simple time travel & merges, Spark-centric; compact with `OPTIMIZE`, clean with `VACUUM`.
- **Iceberg**: powerful snapshot metadata, partition spec evolution, multi-engine; `rewrite_*` + `expire_snapshots`.
- **Hudi**: streaming & upserts powerhouse; MOR needs **compaction**; cleaner/archival manage GC; **clustering** for layout.

Thank you

Shwetank Singh
GritSetGrow - GSGLearn.com