Shwetank Singh
GritSetGrow - GSGLearn.com

DATA AND AI

STORAGE FORMATS & COMPRESSION

PARQUET/ORC/AVRO, ENCODINGS, CODECS

www.gsglearn.com

Storage Formats & Compression --- Parquet - ORC - Avro (Encodings, Codecs, Pushdown, Tuning)

> ***Goal:*** *A complete, hands-on guide to choosing, writing, and tuning Parquet/ORC/Avro for analytics and streaming. Includes internals, encodings, codecs, schema evolution, pushdown, timestamps, tools, and engine-specific knobs (Spark/Trino/Hive/Flink).*

Table of Contents

---

1) Big Picture & When to Use What

- **Parquet** (columnar): default for analytics/lakehouse. Great column pruning, rich stats, wide engine support.
- **ORC** (columnar): deep Hive/Presto/Spark integration; strong stats & indexes (row index stride, bloom filters).
- **Avro** (row-oriented): compact row payload for events/CDC; tightly integrated with schema registries; also useful as a durable interchange format.

Scenario	Parquet	ORC	Avro
Interactive analytics / BI	✅	✅	❌
Wide tables, column pruning	✅	✅	❌
Streaming payloads (Kafka/Event Hubs)	⚠ heavy	⚠ heavy	✅
Long-term lake storage	✅	✅	⚠
Schema evolution friendliness	✅	✅	✅ (via registry)

---

2) Internals & Layout

Parquet

- File → **Row Groups** (e.g., 128--512MB) → **Column Chunks** → **Pages**.
- Per-page stats: min/max, null count, num values.

ORC

- File → **Stripes** (e.g., 64--256MB) → **Streams** per column (data, index, dictionary).
- **Row index stride** (e.g., every 10k rows) enables fine-grained skipping.

Avro

- File → blocks of records; sync markers for resync.
- Self-describing schema embedded in file header.

Sizing Rules

- File size: **256--1024MB**.
- Row group/stripe target: **~128MB**.
- Keep **1--8** groups per file to balance parallelism vs metadata.

---

3) Encodings

- **Dictionary**: map repeated values to IDs. Best for low/med-cardinality strings/ints.
- **RLE (Run Length Encoding)**: compress runs (e.g., booleans/status codes, sorted keys).
- **Bit-packing**: store small ranges in few bits.
- **Delta**: store differences (timestamps, monotonic IDs). Parquet supports **DELTA_BINARY_PACKED** for ints and **DELTA_LENGTH_BYTE_ARRAY** for strings.
- **Plain**: raw values; fallback for high cardinality.

Tips

- Sort/cluster by categorical columns to **amplify dictionary/RLE**.
- For time columns, **delta** + **Zstd** is a strong combo.
- Dictionaries can overflow; engines fall back to plain automatically.

---

4) Compression Codecs

Codec	Speed (comp/dec)	Ratio	Where it shines
Snappy	Fast/Fast	Medium	Safe default; low CPU.
Zstd	Med/Fast	High	Best size/speed tradeoff; tunable levels.
Gzip/Zlib	Slow/Med	High	Cold data; CPU heavy for ingest.
LZ4	Very Fast/Very Fast	Low-Med	Low latency (streaming).
Brotli	Slow/Fast	Very High	Rare in engines; great size if supported.

Heuristics

- Analytics (scan-heavy): **Parquet/ORC + Zstd** (`level 3--6`).
- Streaming/event: **Avro + Snappy/LZ4**.
- CPU-bound cluster: prefer **Snappy**.

---

5) Predicate Pushdown, Statistics & Bloom Filters

- **Column pruning**: read only referenced columns.
- **Predicate pushdown**: skip row groups/stripes/pages outside min/max.
- **Bloom filters** (optional): fast membership checks on selective columns (ORC native; Parquet via implementations/platforms).
- **Row index stride** (ORC): skip within stripes at smaller granularity.

Caveats

- Strings with different collations, NaN semantics, or mixed case can hurt pruning. Normalize.
- Don't store numbers/dates as strings; you lose pushdown.

---

6) Schema & Evolution (Logical Types, Registry)

- Use **logical types**: `DATE` , `TIMESTAMP` , `DECIMAL(precision,scale)` rather than strings.
- **Additive** changes (new nullable fields, widened types) are safest; avoid narrowing.
- For **events**, use **Avro + Schema Registry** with **FULL/BACKWARD_TRANSITIVE** compatibility.

Avro (Confluent CLI examples)

```
# Set subject compatibility
confluent kafka subject update --subject my-topic-value --compatibility
FULL_TRANSITIVE
# Validate a candidate schema
confluent schema validate --subject my-topic-value --schema schemas/order_v5.avsc
```

---

7) Timestamps, Timezones, Decimals & Null/NaN

- **Timestamps**: prefer UTC `TIMESTAMP` with explicit timezone handling. Avoid legacy Parquet `INT96` if possible; use `TIMESTAMP_MILLIS/MICROS`.
- **Decimals**: use proper `DECIMAL(p,s)`; store money amounts as decimals rather than floats.
- **Null/NaN**: stats may exclude NaN; be deliberate about null ordering (`NULLS FIRST/LAST`).

---

8) Engine-Specific Settings

Spark (SQL & PySpark)

```
-- Session defaults
SET spark.sql.parquet.compression.codec = zstd;
SET spark.sql.orc.compression.codec     = zstd;
SET spark.sql.avro.compression.codec    = snappy;

-- Parquet row group & page sizing
SET parquet.block.size = 134217728;  -- 128MB
SET parquet.page.size  = 1048576;    -- 1MB
```

```
# PySpark write examples
(df
  .repartition(200, "dt")
  .write
  .option("maxRecordsPerFile", 5_000_000)
  .mode("append")
  .parquet("abfss://lake/curated/sales"))

(df.write.format("orc").option("orc.compress","ZSTD").save(".../orc_out"))

(df.write.format("avro").option("compression","snappy").save(".../avro_out"))
```

Trino / Presto

```
-- Session properties
SET SESSION parquet_optimized_writer_enabled = true;
SET SESSION parquet_writer_block_size = '134217728B';
SET SESSION hive.compression_codec = 'ZSTD';
```

```
-- Verify pushdown & stats used
EXPLAIN SELECT * FROM sales WHERE dt >= DATE '2025-08-01' AND store_id = 42;
```

Hive

```
SET hive.exec.orc.default.stripe.size = 268435456;  -- 256MB
SET hive.exec.orc.compress = ZSTD;
CREATE TABLE sales_orc (...)
STORED AS ORC TBLPROPERTIES (
  'orc.compress'='ZSTD',
  'orc.create.index'='true'
);
```

Flink (Table API)

```
CREATE TABLE sales_parquet (
  dt DATE,
  store_id INT,
  amount DECIMAL(12,2)
) WITH (
  'connector'='filesystem',
  'path'='s3://bucket/sales',
  'format'='parquet',
  'parquet.compression'='ZSTD'
);
```

---

9) Lakehouse Interop (Delta/Iceberg/Hudi)

- **Delta**: stores data in **Parquet**; compaction via `OPTIMIZE`, GC via `VACUUM`; Z-order (platform-specific) improves skipping.
- **Iceberg**: Parquet/ORC/Avro data files; snapshot metadata governs pruning; maintenance via `rewrite_data_files`, `rewrite_manifests`.
- **Hudi**: Parquet base + delta logs (MOR) or Parquet only (COW); compaction/clustering policies control file sizes/layout.

---

10) Streaming & CDC Payloads (Avro focus)

- **Avro + Registry** gives producer/consumer decoupling and compact wire format.
- Use **subject versioning**; keep **defaults** for new optional fields.
- For CDC (Debezium): fields like `op` (c/u/d), `ts_ms`, and before/after payloads --- consider normalizing to Parquet for analytics downstream.

Spark Structured Streaming (Avro payload in Kafka)

```
from pyspark.sql.functions import col, from_avro
orders_s = (spark.readStream
  .format("kafka")
```

```
  .option("kafka.bootstrap.servers", BOOTSTRAP)
  .option("subscribe","orders")
  .load())

orders = orders_s.select(from_avro(col("value"),
avro_schema_json).alias("o")).select("o.*")

(orders.writeStream
  .format("delta")  # or parquet sink
  .option("checkpointLocation","abfss://.../chk")
  .start("abfss://.../bronze/orders"))
```

---

11) Diagnostics & CLI Tools

- **parquet-tools**: `meta`, `head`, `dump`, `rowcount`.
- **orc-tools**: `meta`, `json-print`, `rowindex`.
- **avro-tools**: `getschema`, `tojson`.

```
parquet-tools meta s3://bucket/path/file.parquet
orc-tools meta s3://bucket/path/file.orc
avro-tools getschema file.avro
```

Inspect **encodings**, **row group sizes**, **min/max**, **null counts** to validate pushdown readiness.

---

12) Tuning Playbooks & Code Snippets

A) Cold backfill to Parquet (Spark)

```
spark.conf.set("spark.sql.parquet.compression.codec","zstd")
spark.conf.set("parquet.block.size", 134_217_728)
(
  df.repartition(200, "dt")                 # by partition key
    .sortWithinPartitions("dt","store_id")  # improve RLE/dictionary
    .write.option("maxRecordsPerFile", 5_000_000)
    .mode("append")
    .parquet("abfss://lake/gld/sales")
)
```

B) Convert Avro bronze → Parquet silver (Spark SQL)

```
CREATE TABLE bronze.orders_avro USING AVRO LOCATION 'abfss://.../bronze/orders';
CREATE TABLE silver.orders_parquet USING PARQUET LOCATION 'abfss://.../silver/orders';
INSERT INTO silver.orders_parquet SELECT * FROM bronze.orders_avro;
```

C) ORC with Bloom filters (Hive)

```
ALTER TABLE t SET TBLPROPERTIES (
  'orc.bloom.filter.columns'='customer_id,sku_id',
  'orc.bloom.filter.fpp'='0.05'
);
```

D) Validate predicate pushdown (Trino)

```
EXPLAIN ANALYZE SELECT * FROM sales WHERE dt = DATE '2025-08-01' AND store_id = 101;
-- Look for "dynamicFilter", "predicatePushdown" and reduced bytes read
```

---

13) Benchmark Micro-Harness

> *Quick way to compare codecs/encodings on your data (Spark)*

```
from time import perf_counter

fmt = "parquet"; path = "abfss://lake/tmp/bench"; codecs = ["snappy","zstd"]
for c in codecs:
    spark.conf.set("spark.sql.parquet.compression.codec", c)
    t0 = perf_counter()
    (df.write.mode("overwrite")
       .option("maxRecordsPerFile", 5_000_000)
       .parquet(f"{path}/{c}"))
    write_s = perf_counter()-t0
    read_t0 = perf_counter()
    spark.read.parquet(f"{path}/{c}").select("dt","amount").where("dt >= '2025-08-
01'").count()
    read_s = perf_counter()-read_t0
    print({"codec": c, "write_s": write_s, "read_s": read_s})
```

---

14) Best Practices & Anti-Patterns

Do

- Use **columnar** (Parquet/ORC) for analytics; **Avro** for events.
- Choose **Zstd** for analytics unless CPU is scarce.
- Keep files **256--1024MB**; row group/stripe ≈ **128MB**.
- Normalize types (DATE/DECIMAL), enable clustering/sorting for better encodings.
- Validate pushdown using EXPLAIN/tools; monitor **bytes read** vs **table size**.

Avoid

- Stringifying numbers/dates; you lose pushdown & compression.
- Over-partitioning; leads to millions of tiny files.
- Over-compressing hot tables (e.g., Gzip level 9) → CPU bottlenecks.
- Legacy Parquet **INT96** timestamps (unless compatibility demands it).

---

15) Cheat Sheet

- **Analytics default**: Parquet + Zstd (3--6), 256--1024MB files, 128MB row groups.
- **Events default**: Avro + Snappy, governed by a Schema Registry with **FULL_TRANSITIVE** compatibility.
- Enable **predicate pushdown** & consider **bloom filters** on selective columns.
- In Delta/Iceberg/Hudi, schedule **compaction**/rewrite to maintain large files and good layout.

Thank you

Shwetank Singh
GritSetGrow - GSGLearn.com