Apache Iceberg 調査報告書

Apache Iceberg と Iceberg REST Catalog について、一次情報の裏取りを前提にまとめた技術調査

dobachi

2026-07-21

目次

1 Apache Iceberg 調査報告書

> ⚠ この文書の位置づけ
>
> **これは著者による個人的な調査メモ（手記）であり、本番環境での利用を想定して書かれたものではありません。**
>
> - 特定時点（2026 年 7 月）のスナップショットであり、**内容は急速に陳腐化します**
> - 網羅性を保証するものではなく、著者の関心に沿って調べた範囲に限られます
> - **本番システムの設計・構築の根拠として、そのまま用いないでください。**判断に使う数値・バージョン・仕様は、必ず引用元の一次情報で確認してください
> - 記述の誤りによって生じた結果について、著者は責任を負いません
>
> 技術的な議論のたたき台や、調べ始める際の地図として使っていただくことを想定しています。

Apache Iceberg と Iceberg REST Catalog について、**一次情報の裏取りを前提に**調べた記録です。仕様やエコシステムの現況を把握するための地図として書いています。

Version: 2026-07-21 / 調査基準日: 2026-07-17 / 対象: Apache Iceberg 1.11.0（2026-05-19 リリース）

> Version はこのページを更新した日付、調査基準日は内容を調べた時点です。Iceberg 周辺は変化が速いため、**調査基準日から離れるほど内容は古くなります**。

本報告書の主張の一部は、Apache Polaris と PyIceberg を用いた実環境で検証しています。検証環境と実験の記録は姉妹リポジトリ **Iceberg REST Lab** に公開しています。

1.1 目次

#	ドキュメント	内容
01	アーキテクチャと中核概念	三層メタデータ構造、スナップショット、楽観的並行制御、クエリプランニング
02	テーブル仕様 v1/v2/v3	各バージョンの差分、スキーマ/パーティション進化、CoW/MoR、deletion vector、row lineage、**v4 の現況**

#	ドキュメント	内容
03	REST Catalog 仕様	エンドポイント、コミットプロトコル、認証、credential vending、server-side scan planning
04	カタログ実装の比較	Polaris / Nessie / Lakekeeper / Gravitino / Glue / S3 Tables / Unity Catalog ほか、選定指針
05	エンジン対応状況	Spark / Flink / Trino / Presto / Snowflake / Databricks / Dremio ほかの対応マトリクス
06	運用・メンテナンス・性能	必須メンテナンス、テーブルプロパティ、アンチパターン、性能劣化の機序
07	エコシステムと業界動向	Delta / Hudi / Paimon 比較、「標準になったのか」の検証、**使うべきでない場面**
08	選定ガイド	**要求から技術選択への決定チャート**（フォーマット→カタログ→エンジンの 3 段階）
90	用語集	本報告書で使う用語の一覧（**各項目に出典を明記**）
99	調査方法論と未確認事項	証跡の扱い方、踏んだ罠、未確認事項の一覧

1.2 先に知っておくと良い結論

報告書全体から、実務判断に効く事実を抜き出したものです。詳細は各ドキュメントを参照してください。

1.2.1 仕様

- **v4 は仕様として未採択ですが、Java 実装ではすでに v4 テーブルを作れます。** ここは取り違えやすく、取り違えると判断を誤ります。仕様本文は「Version 4 is under active development and **has not been formally adopted**」と明記していますが、**Iceberg Java は 1.10.0（2025-09-11）以降 `format-version=4` を設定できます**（`SUPPORTED_TABLE_FORMAT_VERSION = 4`）。一方 **PyIceberg 0.11.1 は受け付けません**（`TableVersion = Literal[1, 2, 3]`）。既定値はいずれも 2。→ 02
- **デフォルトは今も v2 + copy-on-write。** `format-version` の既定値は 2、`write.{delete,update,merge}.mode` の既定値は `copy-on-write` です。v3 の恩恵（deletion vector 等）は明示的なオプトインなしには得られません。→ 02
- **v3 の「GA」は Apache の宣言ではありません。** Apache 側の公式ステータスは「complete and adopted by the community」までで、「GA」はベンダー用語です。→ 02

1.2.2 エンジン

- **v3 対応はベンダーごとにバラバラです。**Snowflake GA（2026-05-07）、Databricks GA（2026-05-28）、**Dremio は今も Preview**。Trino は**公式 docs 上まだ experimental**（PR はマージ済みだが「マージ済み ≠ リリース済み ≠ サポート対象」）。→ 05
- **PyIceberg は v3 テーブルを書けません。**読めますが書けません。equality delete は**読むこともできません**。merge-on-read を指定しても警告を出して copy-on-write に落ちます。→ 05

1.2.3 運用

- **公式は compaction を “Optional”、`expire_snapshots` と metadata 掃除を “Recommended” に分類しています。**一般的な認識と逆です。しかも **`expire_snapshots` を回さない限り compaction は容量を一切解放しません**（むしろ増えます）。→ 06
- **`write.metadata.delete-after-commit.enabled` はテーブル作成時に決めるべき設定です。**既定は false で、`previous-versions-max`（既定 100）を溢れた metadata JSON は追跡対象から外れ、**後から有効化しても救済できません**。→ 06
- **S3 Tables はタグやブランチを 1 つ作るだけで自動スナップショット管理がテーブル全体で停止します**（fail-closed）。`history.expire.*` の設定でも同様。しかも失敗は課金増加としてしか現れません。→ 06

1.2.4 エコシステム

- **「Iceberg がデファクト標準になった」は不正確です。**カタログの相互接続層としては REST Catalog 仕様が共通語になりつつある一方、フォーマット単体では決着していません（Fabric のネイティブは Delta、Confluent/Fivetran は両対応、Paimon の開発速度は Iceberg の 1.63 倍〔人間コミット換算〕、DuckLake v1.0 が 2026 年 4 月に登場）。→ 07
- **Databricks が Delta 5.0 に Iceberg v4 のメタデータ構造を採用する提案を出しています。**実現すれば「勝敗」ではなく統合になります。ただし Delta 5.0 も Iceberg v4 も未リリースです。→ 07

1.3 本報告書の証跡方針

> **この報告書は人間と AI の共同作業で作成しています。**調査・執筆・検証を人間と AI が分担しており、**人間が書いた部分と AI が書いた部分が混在しています。**正確性を高めるため一次情報での裏取りと公開前のファクトチェックを行い、その過程で見つかった誤りは修正しましたが（fact-check-report.md に記録）、**すべての主張を検証したわけではなく、誤りが残っている可能性があります。**重要な判断に使う数値・バージョン・仕様の記述は、引用元の一次情報で再確認してください。

誤りを減らすため、**同じ問いを複数回・独立に調べ、結果が食い違った点は改めて一次情報に当たって決着させています**。方針は以下の通りです。

1. **一次情報に直接あたって確認する。**記憶や推測でバージョン番号・プロパティ名・API シグネチャを書きません。
2. **優先順位を固定する。**公式ドキュメント > リポジトリの実ファイル/ソースコード > 公式ブログ・プレスリリース > 技術ブログ。矛盾したら上位を採用し、**矛盾の存在自体を記録します**。
3. **「未確認」と「非対応」を区別する。**「ソースが見つからなかった」と「公式が非対応と明記している」は全く別の事実です。本文中でも区別しています。
4. **「マージ済み ≠ リリース済み ≠ サポート対象」を区別する。**
5. **出典 URL を付ける。**日付が判明するものは日付も添えます。

調査中に判明した具体的な罠(リリースノートは遡及更新されない、プレスリリースと docs が食い違う、など)は 99-methodology.md に方法論としてまとめています。**この報告書を更新する人、および同種の調査を行う人はそちらを先に読むことを勧めます。**

1.3.1 実機で検証した範囲

Apache Polaris 1.6.0 + PyIceberg 0.11.1 の実環境に対して、以下を実際に確認済みです。

主張	検証結果
新規テーブルの既定は format-version=2	○ 実測で 2
`write.delete.mode` の既定は copy-on-write	○ 実測で copy-on-write
PyIceberg は V3 を書けない	○ `NotImplementedError` を確認
MoR 指定は警告を出して CoW に落ちる	○ `Merge on read is not yet supported, falling back to copy-on-write` を捕捉
`MaintenanceTable` は `expire_snapshots` のみ	○ compaction / orphan 削除の不在を確認
`s3.path-style-access` は PyIceberg に存在しない	○定数の不在を確認
rename しても field ID は変わらない	○ 実測で一致
credential vending の prefix スコープ	○テーブル単位の短命資格情報が払い出されることを確認
409 CommitFailedException とそのメッセージ	○ 意図的に再現
サーバが `overrides` で `prefix` を配る	○ `prefix = lab_catalog` を確認
`namespace-separator` の既定は `0x1F`	○ `%1F` を確認

1.3.2 既知の限界

- **調査基準日時点のスナップショットです。**Iceberg 周辺は変化が速く、特にベンダーの対応状況は数ヶ月で変わります。バージョン番号や GA/Preview の別を引用する際は再確認してください。
- **実機検証は上記の範囲に限られます。**それ以外はドキュメントとソースコードの読解に基づきます。特に **Spark / Flink / Trino / Snowflake / Databricks / Dremio の対応状況は未検証**で、各社のドキュメントの読解に依拠しています。
- **未確認事項が残っています。**99-methodology.md に一覧化しています。

1.4 ライセンス

本報告書は CC BY 4.0 のもとで公開します。引用元の各プロジェクト・製品のドキュメントの権利は、それぞれの権利者に帰属します。

2 01. アーキテクチャと中核概念

調査基準日: 2026-07-17 / **一次情報:** Apache Iceberg Table Spec（**生ソース:** `format/spec.md`）

> **出典についての注記**: `iceberg.apache.org/spec/` は JavaScript でページを組み立てる方式のため、本文をそのまま取り込めません。そこで本報告書は、同じ内容の生ソース（`apache/iceberg` の main ブランチにある `format/spec.md`、2137 行）を出典としています。以降のドキュメントも同じ方針です。詳細は 99-methodology.md を参照。

2.1 Iceberg とは何か（そして何でないか）

Iceberg は**テーブルフォーマット**です。ファイルフォーマット（Parquet/ORC/Avro）でもストレージでもエンジンでもありません。「オブジェクトストレージ上に散らばったファイル群を、1 つのテーブルとして正しく扱うためのメタデータの規約」がその実体です。

Hive テーブルとの根本的な違いは、仕様の Overview に明記されています。

> This table format tracks **individual data files** in a table instead of directories.

ディレクトリではなくファイル単位で追跡する --- ここからほぼすべての設計上の帰結が導かれます。なぜそれが効くのか、順に見ていきます。

Hive 形式のテーブルは「このディレクトリの下にあるファイルが、このパーティションのデータ」という約束でデータの所在を表します。どのファイルがテーブルに属するかは、クエリのたびにディレクトリを一覧して確かめます。ところが S3 のようなオブジェクトストレージに本当のディレクトリは無く、「ディレクトリの一覧」は接頭辞を指定した LIST 操作にすぎません。この操作はファイル数に比例して遅く、ページングを伴います。かつては結果がすぐ反映されないことによる取りこぼしも起こり得ました。テーブルが大きくなるほど、この「毎回数える」コストが重くのしかかります。

Iceberg は、どのファイルがテーブルに属するかをメタデータ（マニフェスト）に明示的に書き留めます。クエリのプランニングはストレージを一覧する代わりに、このメタデータを読むだけで済みます。ここから 3 つの利点が生まれます。第一に**正確さ**--- 一覧の網羅性や反映タイミングに左右されません。第二に**速さ**--- ファイル数に比例した一覧走査が要りません（後述のクエリプランニングの多段プルーニングにつながります）。第三に、**ファイルの所在が「置き場所」から切り離されます**。あるファイルがどのパーティションに属するかは、ディレクトリのパスではなくメタデータに記録された値で決まります。そのためファイルをどこに置いても、パーティションの切り方を後から変えても、データを動かす必要も、クエリを書き換える必要もありません。クエリが参照するのはディレクトリのパスではなく論理的な列だからです。

2.2 三層のメタデータ構造

```
catalog                          … テーブル → 現 metadata ファイルへのポインタ
  └── v<N>.metadata.json           … テーブル状態（JSON）
      └── snapshot                 … metadata.json 内に埋め込まれたリスト
          └── manifest list        … snap-*.avro（1 スナップショットに 1 つ）
              └── manifest         … *-m0.avro（データ用/削除用は分離）
                  └── data file / delete file（Parquet/ORC/Avro/Puffin）
```

層	ファイル名例	フォーマット	役割
catalog	（外部）	メタストア/REST/DB	現 metadata ファイルへのポインタを保持。**atomic swap の実行主体**
metadata file	`v<V>.metadata.json` または `<V>-<uuid>.metadata.json`	**JSON**	schema / spec / sort-order / snapshot 一覧 / properties
manifest list	`snap-<snapshot-id>-<n>.avro`	**Avro**	1 スナップショット分の manifest 一覧＋パーティション要約統計
manifest	`<uuid>-m<n>.avro`	**Avro**	data file または delete file の一覧＋パーティション値＋列統計
data file	`*.parquet` 等	Parquet / ORC / Avro	実データ
delete file / DV	`*-deletes.parquet` / `*.puffin`	Parquet/ORC/Avro / **Puffin**	行レベル削除

仕様が明示する重要な性質:

- 「The data in a snapshot is the **union of all files in its manifests**.」
- 「Manifest files are **reused across snapshots** to avoid rewriting metadata that is slow-changing.」--- マニフェストは変化の遅いメタデータを再利用する仕組みです。
- 「Manifests can track data files with **any subset of a table and are not associated with partitions**.」--- **マニフェストはパーティションに紐づきません。**これは頻出する誤解です。

2.2.1 マニフェストの制約

- 「A manifest may store either data files or delete files, **but not both** because manifests that contain delete files are **scanned first** during job planning.」--- delete を先に読む必要があるため、分離されています。
- 「A manifest stores files for a **single partition spec**.」--- manifest の Avro スキーマがパーティションスペックに依存するため、スペックが変わればファイルは別の manifest に書かれます。
- 「All columns must be written to data files even if they introduce redundancy with metadata（例: identity パーティション列）… provides a **backup in case of corruption or bugs in the metadata layer**.」--- メタデータ層のバグに対する保険として、冗長でも全列を書きます。

2.2.2 具体例: 1 回の追記で何が起きるか

抽象的なので、小さなテーブルに一度だけ追記した場合を追ってみます(以下のファイル名は説明用の例です)。

`db.events`(列は `event_id`, `region`, `ts`。`ts` の日付でパーティション)に、7 月 1 日と 7 月 2 日にまたがる数行を 1 回 `INSERT` したとします。生成されるファイルはこうなります。

```
s3://warehouse/db/events/
├── metadata/
│   ├── 00001-8a1f….metadata.json      ← 追記後のテーブル状態。ここに snapshot が 1 つ
│   ├── snap-73…-1-b2c….avro            ← manifest list(このスナップショット用に 1 つ)
│   └── 0f9…-m0.avro                    ← manifest(下の 2 ファイルを列挙)
└── data/
    ├── ts_day=2026-07-01/00000-….parquet   ← 7/1 の行
    └── ts_day=2026-07-02/00000-….parquet   ← 7/2 の行
```

このテーブルを読むエンジンは、次の順にたどります。

1. **カタログ**に「`db.events` の現在の metadata はどれか」を尋ね、`00001-8a1f….metadata.json` を得る。
2. その **metadata.json** を読み、現在のスナップショットが指す **manifest list**(`snap-73…-1-b2c….avro`)にたどり着く。
3. **manifest list** から、このスナップショットに属する **manifest**(`0f9…-m0.avro`)を得る。ここには各 manifest のパーティション要約統計も入っている。
4. **manifest** を読み、実際の**データファイル** 2 つと、それぞれの列統計・パーティション値を得る。

肝心なのは、**どのファイルがテーブルに属するかを、一度もストレージを一覧せずに確定できる**ことです。`data/` の下を `ls` して回るのではなく、メタデータをたどるだけで「読むべきは 2 ファイル」と分かります。2 回目の追記をすると、新しい metadata.json・manifest list・manifest が増え、カタログのポインタがそちらへ差し替わります(ACID と楽観的並行制御)。古いスナップショットのファイルはそのまま残るので、タイムトラベルで過去の状態も読めます。

2.3 ACID と楽観的並行制御

仕様の記述:

> An **atomic swap** of one table metadata file for another provides the basis for **serializable isolation**.
>
> Readers use the snapshot that was current when they load the table metadata and are **not affected by changes until they refresh**.
>
> Writers create table metadata files **optimistically**, assuming that the current version will not be changed before the writer's commit.

そして最も重要な一文:

> **The conditions required by a write to successfully commit determines the isolation level. Writers can select what to validate and can make different isolation guarantees.**

分離レベルは固定ではなく、書き込み側の検証条件で決まります。これは実務上きわめて重要な性質で、`write.*.isolation-level`（既定 `serializable`、`snapshot` に緩和可能）という設定が存在する理由でもあります。

設計目標として「**Readers will not acquire locks.**」が明記されています。読み手はロックを取りません。

2.3.1 コミットの実装

Metastore Tables（推奨）:

1. 現メタデータを基に新メタデータを作成
2. `<V+1>-<random-uuid>.metadata.json` に書き出し
3. メタストアに **check-and-put** でポインタ swap を要求 → 成功なら commit 成立、失敗なら 1 に戻る

File System Tables（**使わないこと**）: atomic rename を使う方式ですが、仕様が明示的に deprecated とし、さらに「The scheme is **unsafe** in object stores and local file systems.」と述べ、**v4 で削除予定**としています。カタログ（metastore / REST）を使ってください。

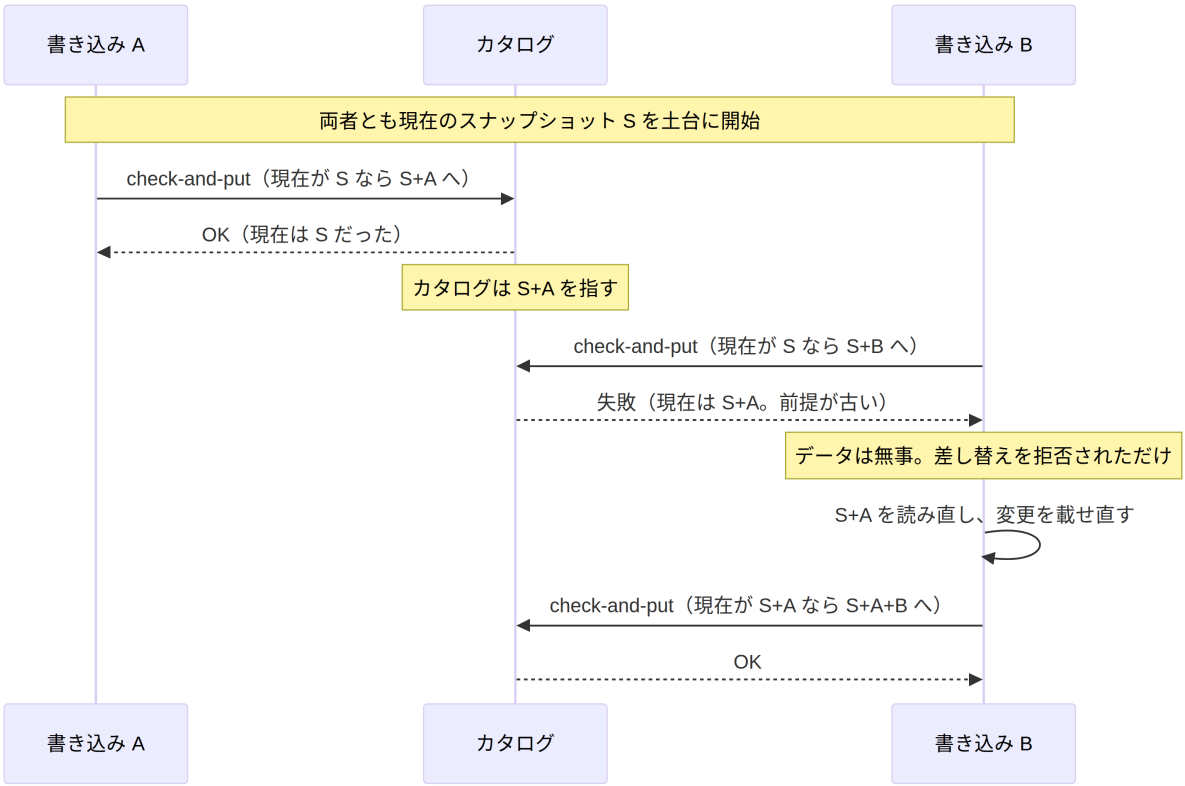
> **重要**: 仕様は commit の atomic 操作自体を標準化していません。「The atomic operation used to commit metadata depends on how tables are tracked and **is not standardized by this spec**.」--- **カタログ実装依存**です。これが REST Catalog 仕様（03）が重要になる理由の一つです。

2.3.2 競合解決とリトライ

操作	検証条件
Append	「have no requirements and can **always** be applied.」--- 常に適用可能
Replace	削除予定ファイルがまだテーブルにあることを検証（compaction 等）
Delete	削除対象ファイルがまだテーブルにあることを検証。ただし**式ベースの削除（`where timestamp < X`）は常に適用可能**
スキーマ/スペック更新	base 版と current 版でスキーマが変わっていないことを検証

2.3.3 具体例: 2 つの書き込みが同時に来たら

先ほどの `db.events` に、A と B が同時に追記しようとした場面を追ってみます。両者とも、いまカタログが指すスナップショット（仮に S とします）を土台にして書き始めます。



1. **A が先にコミット**: A は S に自分の追加ファイルを載せた新しい metadata を書き、カタログへ「**現在が S なら、これに差し替えて**」と check-and-put を要求する。カタログの現在は S なので成功。カタログは A のスナップショット（S+A）を指す。
2. **B のコミットは弾かれる**: B も S を土台に metadata を書き、「**現在が S なら差し替えて**」と要求する。しかしカタログの現在はもう S+A。**前提が崩れているので check-and-put は失敗する**（B が書いたデータが壊れるわけではなく、ポインタの差し替えが拒否されるだけ）。
3. **B はリトライ**: B はテーブルを読み直し、最新の S+A を土台に自分の変更を載せ直して再びコミットする。今度は現在が S+A なので成功し、カタログは S+A+B を指す。

ここで効いているのが、すぐ上の「競合解決とリトライ」の表です。A も B も **append**（ファイルを足すだけ）なので要求は「have no requirements」--- **中身の衝突は起きません**。B が失敗したのは中身がぶつかったからではなく、単に「土台にしたスナップショットが古くなった」からで、載せ直せば必ず通ります。**待ちもロックも発生せず、負けた側が土台を更新してやり直すだけ**です。

これが **delete** だと話が変わります。B が「`event_id = 5` の行を含むファイルを消す」つもりでいても、その間に A が compaction でそのファイルを別のファイルへ書き換えていたら、B の前提（消す対象のファイルがまだある）は崩れ、リトライが必要になります。**何を検証するか＝どこまで厳しく衝突を見るかを、書き込み側が選べる**。これが「分離レベルは書き込み側の検証条件で決まる」の具体的な中身です。

2.3.4 シーケンス番号と継承

コミット成功ごとに単調増加します。設計の妙は**継承(inheritance)機構**にあります。

> Inheriting the sequence number from manifest metadata allows **writing a new manifest once and reusing it in commit retries**. To change a sequence number for a retry, **only the manifest list must be rewritten**.

新規エントリは `null` で書かれ、読み取り時に manifest list の値へ置き換わります。おかげで**リトライのコストは manifest list の書き直しだけで済みます**。継承が効くのは status=1 (ADDED) のときに限られ、EXISTING/DELETED では明示値が必須です。

`sequence_number`（データシーケンス番号 = 内容の相対的な古さ）と `file_sequence_number`（追加時のスナップショット番号）は別物です。仕様は「**The file sequence number can't be used for pruning delete files**」と明記しています。

2.4 クエリプランニングの多段プルーニング

仕様の設計目標:

> **Speed** --- Operations will use **O(1) remote calls** to plan the files for a scan and **not O(n)** where n grows with the size of the table, like the number of partitions or files.

これを支えるのが多段プルーニングです。

段 1: manifest レベルのスキップ manifest list の各エントリは `field_summary`（`contains_null` / `contains_nan` / `lower_bound` / `upper_bound`）と各種カウントを持ちます。仕様:「Manifests that contain no matching files, determined using either **file counts or partition summaries**, may be skipped.」--- **manifest ファイル自体を開かずに除外**します。

段 2: manifest 内のパーティションプルーニング スキャン述語を **inclusive projection** でパーティション述語に変換します。「Conversion uses the partition spec that was **used to write the manifest file** regardless of the current partition spec.」--- これがパーティション進化に対応できる鍵です。

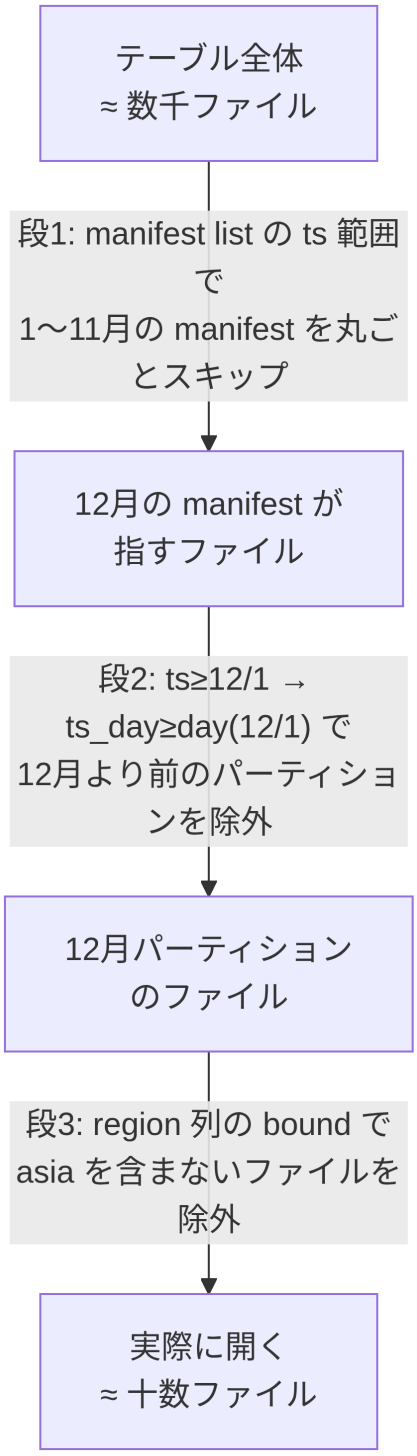
段 3: 列統計によるファイルプルーニング「Scan predicates are also used to filter data and delete files using **column bounds and counts** that are stored by field id in manifests.」

段 4: delete ファイルのプルーニング

> **The same filter logic can be used for both data and delete files** because both store metrics of the rows either inserted or deleted. If metrics show that a delete file has no rows that match a scan predicate, it may be **ignored just as a data file would be**.

具体例(仕様より):「if `file_a` has rows with `id` between 1 and 10 and a delete file contains rows with `id` between 1 and 4, a scan for `id = 9` **may ignore the delete file**」--- **delete ファイルもプルーニング対象**です。MoR の性能を理解する上で重要な点です。

段 1〜3 が実際にどう効くか、先ほどの `db.events` が 1 年分・数千ファイルに育った状態で、`WHERE region = 'asia' AND ts >= '2026-12-01'` を投げた場合で追ってみます(ファイル数は説明用の概数です)。



- **段 1（manifest スキップ）**: manifest list の要約統計を見て、`ts` の範囲が 12/1 より前で閉じている manifest を丸ごと飛ばす。1〜11 月ぶんの manifest がここで落ち、その中の数千ファイルは**そもそも見に行かない**。
- **段 2（パーティションプルーニング）**: 残った manifest 内で、`ts >= '2026-12-01'` を inclusive projection で `ts_day >= day('2026-12-01')` に変換し、12 月より前のパーティションのファイルを除外する。

- **段 3（列統計でのファイル除外）**：残ったファイルごとに `region` 列の lower/upper bound を見て、`asia` を含みえないファイル（例: `region` が `europe` しか入っていないファイル）を、開かずに落とす。

こうして数千ファイルが、実際に開く十数ファイルまで絞られます。**どの段でもデータファイルの中身は読まず、メタデータに記録された範囲情報だけで判断している**のが要点です。これが「O(1) remote calls でプランする」の中身であり、テーブルが何倍に育ってもプランニングのコストが比例して増えない理由です。

2.4.1 inclusive projection とは

パーティションの厄介なところは、**生の列そのものではなく、その列を変換した値でデータを仕分ける**点にあります。`db.events` は `ts` の日付（`day(ts)` = `ts_day`）でパーティションされていますが、ユーザーが書くクエリの条件は生の `ts` に対する `ts > X` です。ここで「`ts_day` に一言も触れていないクエリから、どうやって `ts_day` を使って不要なパーティションを飛ばすのか」という問題が生じます。この橋渡しをするのが inclusive projection です。

> if a scan predicate matches a row, then the partition predicate must match that row's partition. This is called inclusive because **rows that do not match the scan predicate may be included in the scan** by the partition predicate.

inclusive projection は、生の列への述語を**パーティション値への述語に変換する規則**で、次の保証を持ちます--- **ある行が元の述語を満たすなら、その行が属するパーティションは必ず変換後の述語を満たす**。裏返せば（対偶）、**変換後の述語を満たさないパーティションには、元の述語を満たす行が 1 つも無い**。だからそのパーティションは丸ごと飛ばして安全です。これが段 2 のプルーニングの根拠です。

具体的に見ます。`ts > '2026-03-15 10:00'` という条件を、`ts_day = day(ts)` のパーティションに投影します。この条件を満たす行は必ず 2026-03-15 以降の日付にあるので、変換後の述語は `ts_day >= day('2026-03-15')`、つまり **`ts_day >= 2026-03-15`** になります。

ここで**変換がわざと緩い**ことに注目してください。2026-03-15 のパーティションには、00:00〜10:00 の（`ts > 10:00` を満たさない）行も混じっています。それでもこのパーティションは「含める」側に倒します。逆に、03-15 より前のパーティションに `ts > '…10:00'` を満たす行は絶対に無いので、そこは確実に飛ばせます。

この非対称さが肝です。**false positive（実際には該当行が無いパーティションやファイルを読んでしまう）は許容**します--- 無駄な読み込みは起きますが、答えは正しいままです。一方 **false negative（該当行があるパーティションを飛ばしてしまう）は絶対に許されません**--- 答えが欠けてしまうからです。inclusive projection は「取りこぼしゼロ、その代わり多少の空振りは許す」という設計で、だからこそプルーニングに安全に使えます。緩い側（inclusive）に倒すのはそのためです。

しかも、この変換をエンジンが**クエリの述語から自動で導く**ので、ユーザーは `ts_day` を書く必要がありません。Hive なら `WHERE ts > X AND ts_day >= '2026-03-15'` のように論理条件とパーティション条件の両方を書く必要がありましたが、Iceberg では `WHERE ts > X` だけで済みます。これが hidden partitioning（パーティション列がクエリから「隠れている」）の実体です。パーティションの切り方を後から変えても（パーティション進化）クエリを書き換えずに済むのも、この自動変換のおかげです。

対になるのが **strict projection** です。

> a partition predicate that will match a file **if all of the rows in the file must match** the scan predicate

strict projection は inclusive の逆で、**そのファイルの全行が確実に述語を満たすときだけ**マッチする、厳しい側に倒した変換です。用途は **residual predicate（残余述語）の計算**です。あるファイルのパーティションが strict projection にマッチするなら、そのファイルの行はパーティション由来の条件を全て満たすと分かっているので、**その部分は行ごとに再チェックしなくてよい**。残った条件（residual）だけを各行に当てれば済み、無駄な評価が減ります。inclusive が「読むファイルを絞る」ため、strict が「読んだファイルの中で何を再評価すべきかを削る」ため、と役割が対になっています。

2.4.2 未知の transform

「The inclusive projection for an unknown partition transform is **true**」--- フィルタリングでは無視されます。ただし「the result of an unknown transform is **still used when testing whether partition values are equal**」（delete 適用のパーティション一致判定には使われる）点に注意。

2.4.3 スキャンの前提条件

> for any snapshot, all file paths marked with "ADDED" or "EXISTING" **may appear at most once** across all manifest files in the snapshot. If a file path appears more than once, the results of the scan are **undefined**.

2.5 Puffin と統計情報

Puffin は統計とインデックスのための blob コンテナ形式です。現在 2 つの blob 型が定義されています。

- **`apache-datasketches-theta-v1`**: Apache DataSketches の compact Theta sketch。blob metadata の properties に `ndv`（distinct 値数の推定）を含みうる。
- **`deletion-vector-v1`**: v3 の deletion vector（02 参照）。

統計は情報提供目的（informational）です。

> Statistics are informational. A reader can **choose to ignore** statistics information. **Statistics support is not required to read the table correctly.**

パーティション統計も同様に「not required for reading or planning」です。**正しさには不要で、速度のためだけに存在します。**ただし読み手が有効な統計ファイルとして扱うには「**must be registered in the table metadata file** to be considered as a valid statistics file for the reader」という登録が求められます。

2.6 メタデータテーブル

db.table.history のようにテーブル名の後にメタデータテーブル名を付けて参照します。

テーブル	用途
history	テーブル履歴。is_current_ancestor で**ロールバックされたコミットを検出**できる
metadata_log_entries	metadata.json の変遷
snapshots	有効なスナップショット一覧
entries	現在の manifest エントリ(data/delete 両方)。readable_metrics 付き
files / data_files / delete_files	現在のファイル一覧と統計
manifests	現在の manifest 一覧と partition_summaries
partitions	現在のパーティションとレコード数
position_deletes	position delete の中身
refs	branch / tag 一覧
all_*（all_files, all_entries, all_manifests ほか）	**全スナップショット横断**。orphan file の調査に有用だがコスト高

実務上の注意:

- **partitions テーブルは delete を適用しません。**「delete files are not applied, and so in some cases partitions may be shown even though all their data rows are marked deleted by delete files.」--- 全行削除済みのパーティションも表示されます。
- manifests の contains_nan は **v1 テーブルでは null になりえます**（populate されないため)。
- partitions テーブルは非パーティションテーブルでは partition と spec_id 列を持ちません。

2.7 タイムトラベルと 2 つの履歴

まず基本から。Iceberg はテーブルに変更を加えるたびに、その時点の状態を 1 枚の**スナップショット**として保存します(テーブルの写真を撮るイメージです)。**タイムトラベル**とは、最新ではなく過去のスナップショットを読むこと---「昨日の 15 時のテーブルを見せて」ができる機能です。過去を指定する方法は 2 通りあります。

- **スナップショット ID（バージョン)で指定**:「この ID の状態を見せて」。素直で、そのスナップショットをそのまま読むだけです(Spark なら VERSION AS OF <id>)。
- **時刻で指定**:「この時刻の状態を見せて」(Spark なら TIMESTAMP AS OF '<時刻>')。

ややこしいのは時刻指定のほうです。その理由が「2 つの履歴」にあります。

Iceberg は履歴を 2 つの別々の形で持っていて、両者は食い違うことがあります。

- **系譜(parent-child lineage)** : snapshots に記録される、スナップショットの親子関係です。コミットのたびに「直前の current を親として新しいスナップショットができる」という、**どう作られたかの系図**です。

- **current の履歴（`snapshot-log`）**：「いつ、どのスナップショットが current になったか」を時刻つきで並べた**時系列のログ**です。各エントリは（時刻, snapshot-id）の対になっています。

  > These two histories **might indicate different snapshot IDs for a specific timestamp**.

ふだんはこの 2 つは一致します。通常のコミットは、新しいスナップショットの親を直前の current にしつつ、同時に snapshot-log にも新しいエントリを足すので、系図をたどっても時系列をたどっても同じ順序になるからです。ずれるのは、**current を「任意の既存スナップショット」へ動かす操作**をしたときです。仕様の言葉では:

> updating the `current-snapshot-id` can be used to set the snapshot of a table to **any arbitrary snapshot**, which might have a lineage derived from a table branch or **no lineage at all**

`rollback_to_snapshot` や `set_current_snapshot`、cherry-pick がこれに当たります。これらは新しいスナップショットを作らず、current のポインタを既存のスナップショットへ移すだけです。すると**snapshot-log には「その時刻に current が X になった」と記録される一方、X の親子系譜は元のまま**なので、両者がずれます。

具体例で見ます。10:00・11:00・12:00 に普通に 3 回コミットし、その後 12:30 に S1 へロールバックしたとします。

系譜は、ロールバックしても変わりません。

一方 current の履歴（snapshot-log）は、ロールバックで末尾に 1 行増えます。

時刻	current になった snapshot
10:00	S1
11:00	S2
12:00	S3
12:30	**S1（ロールバック）**

ここで「**12:15 時点の状態**」を時刻指定で読むと、何が返るべきでしょうか。答えは **S3** です。12:15 の時点で、テーブルは実際に S3 だったからです。これは snapshot-log を見れば分かります（12:15 以前で最も新しいエントリは 12:00 → S3）。ところが、現在の current（S1）から親子系譜をさかのぼっても、S1 の祖先に S3 は出てきません。**時刻指定のタイムトラベルは、系譜ではなく snapshot-log を使わないと正しく答えられない** --- これがこの節の要点です。

では、食い違うと何が問題なのでしょうか。まず、これは**データの破損ではありません**。「いまが何か（current）」と「各スナップショットがどう連なっているか（系譜）」はもともと別物で、ロールバックや cherry-pick はその 2 つをずらすだけです。実際に困りうる現象は主に 2 つあります。

1. **時刻指定のタイムトラベルが「あとで取り消した状態」を返すことがある。**上の例で 12:30 に S1 へ戻したあとでも、`TIMESTAMP AS OF '12:15'` は S3 を返します。これは正しい挙動なのですが（12:15 の時点では確かに S3 だった）、「S1 に戻したのに、なぜ時刻指定だと S3 が出るのか」と直感に反して見えることがあります。**時刻指定は『その時点で current だったもの』を返す**、と理解すれば筋が通ります。
2. **実装がここを取り違えると、時刻指定が間違った答えを返す。**時刻指定を系譜のさかのぼりで実装してしまうと、ロールバック後に正しいスナップショットを見つけられません。仕様が次のように snapshot-log の使用を指示しているのは、この誤りを防ぐためです。

   仕様の指示:「When processing point in time queries implementations **should use "snapshot-log" metadata**」

該当するエントリが無い、または snapshot-log が未 populate（optional フィールドなので空のこともあります）なら、推測で答えず「informative error message」を出すべきとされています。

> **git を使っている人向けの補足**（知らなければ読み飛ばして構いません）：この 2 つの履歴は、git の **reflog** と **commit グラフ**の関係にそのまま対応します。snapshot-log は reflog（「HEAD がいつどこを指したか」の時系列）、系譜は commit の親子グラフです。`git reset --hard <古いコミット>` で HEAD を戻しても commit グラフ自体は変わらないのと同じで、Iceberg のロールバックも current を動かすだけで系譜は変えません。「ある時刻に何を指していたか」は reflog（＝ snapshot-log）、「そのコミットの祖先は何か」はグラフ（＝系譜）で答える、という役割分担も同じです。

なおメタデータテーブルの `history` に `is_current_ancestor` 列があるのは、まさにこの区別のためです。ロールバック後、S2 と S3 は現在の current（S1）の祖先ではなくなるので `is_current_ancestor = false` になり、「時系列には現れるが、いまの系譜からは外れたスナップショット」を見分けられます。

2.8 branch / tag と WAP

branch（ブランチ）と **tag（タグ）**は、どちらも「あるスナップショットを指す、名前つきのしおり」です。git のブランチ・タグと同じ発想で、用語もそこから借りています。

- **tag**: 特定のスナップショットに貼る**動かない名札**です。「月末時点の状態」に `EOM-2026-07` というタグを付けておけば、その後どれだけ書き込んでも、そのタグはずっと同じ状態を指し続けます。監査や、あとで参照したい状態の保存に使います。
- **branch**: **本流（`main`）とは別に書き込める枝**です。`main` を汚さずに枝の上で変更を積み、問題なければ後で `main` に取り込みます。

この「別の枝で作業して、良ければ本流に取り込む」流れを、データの品質検証に応用したのが後述の WAP です。

内部的には、branch と tag（まとめて **snapshot reference** と呼びます）は次の情報を持ちます: `snapshot-id` / `type`（`tag` | `branch`）/ `min-snapshots-to-keep`（branch のみ）/ `max-snapshot-age-ms`（branch のみ）/ `max-ref-age-ms`。

- 「There is always a `main` branch reference pointing to the `current-snapshot-id` even if the `refs` map is null.」
- **「The `main` branch never expires.」**

つまり main という branch は常に存在し（refs が空でも暗黙にある）、期限切れで消えることはありません。

2.8.1 スナップショット保持ポリシーのアルゴリズム

要点は「**どこかの branch / tag から参照されているスナップショットは expire（期限切れ削除）されない**」です。以下はその具体化です。

1. 保持集合を空で開始
2. main 以外の ref で、参照先が max-ref-age-ms より古いものを削除
3. 各 branch / tag について参照先スナップショットを保持集合に追加
4. 各 branch について、以下の**両方**を満たすまで祖先を追加: (a) max-snapshot-age-ms より古い **AND** (b) branch の最初の min-snapshots-to-keep 個に含まれない
5. 保持集合に無いスナップショットを expire

> **運用上の罠**: タグを 1 つ付けただけで、そのタグが参照するスナップショットとその依存ファイルは無期限に残ります。Amazon S3 Tables ではこれがさらに深刻な挙動（テーブル全体で自動メンテナンスが停止）になります→ 06

2.8.2 Write-Audit-Publish（WAP）

WAP は Write（書く）→ Audit（検証する）→ Publish（公開する）の略で、**「検証してから公開する」ための仕組み**です。

ふつうにテーブルへ書き込むと、その結果はすぐに全員から見えてしまいます。もし壊れたデータを書けば、そのまま下流のクエリに流れます。WAP はこれを避けます --- **まず監査用のブランチに書き込み、そのブランチの上でデータ品質チェックを行い、問題なければ main に取り込んで（publish）はじめて全員に見えるようにする**、という段取りです。バッチごとに「ステージング→ 検証 → 本番反映」のゲートを設けるイメージです。

下は Spark での典型的な流れです。

```
-- 1. WAP を有効化
ALTER TABLE db.table SET TBLPROPERTIES ('write.wap.enabled'='true');
-- 2. 監査ブランチを作成（7日保持）
ALTER TABLE db.table CREATE BRANCH `audit-branch` AS OF VERSION 3 RETAIN 7 DAYS;
-- 3. ブランチへ書き込み（main の履歴とは独立）
SET spark.wap.branch = audit-branch;
INSERT INTO prod.db.table VALUES (3, 'c');
-- 4. 検証ワークフローで audit-branch の状態を検証
-- 5. 検証後、main を audit-branch の head まで fast-forward
CALL catalog_name.system.fast_forward('prod.db.table', 'main', 'audit-branch');
```

snapshot summary の WAP 関連フィールド: wap.id（ステージされたスナップショットの WAP ID）、published-wap-id、source-snapshot-id（cherry-pick 元）。

> **重要な制約**:「the schema tracked for a table is valid across all branches」--- **スキーマはブランチ間で共有されます。**ブランチごとに独立したスキーマは持てません。

2.8.3 rollback 系プロシージャ

過去のスナップショットを操作するための代表的なプロシージャです(いずれも Spark の例)。

プロシージャ	用途
`rollback_to_snapshot('db.sample', 1)`	特定スナップショットへロールバック
`rollback_to_timestamp('db.sample', TIMESTAMP '...')`	特定時刻へロールバック
`set_current_snapshot('db.sample', 1)`	現スナップショットを任意に設定(**祖先でなくてもよい**)
`cherrypick_snapshot('my_table', 1)`	既存スナップショットの変更を現在の状態に適用
`publish_changes('my_table', 'wap_id_1')`	WAP ID からスナップショットを publish。同じ wap_id が複数あると**失敗**
`fast_forward('my_table', 'main', 'audit-branch')`	ブランチを別ブランチの head へ早送り

2.9 出典

- Iceberg Table Spec: https://iceberg.apache.org/spec/ / 生ソース https://github.com/apache/iceberg/blob/main/format/spec.md
- Puffin Spec: https://iceberg.apache.org/puffin-spec/ / https://github.com/apache/iceberg/blob/main/format/puffin-spec.md
- Spark Queries（メタデータテーブル、タイムトラベル）: https://github.com/apache/iceberg/blob/main/docs/docs/spark-queries.md
- Spark Procedures（rollback、cherrypick、fast_forward）: https://github.com/apache/iceberg/blob/main/docs/docs/spark-procedures.md
- Branching and Tagging（WAP）: https://github.com/apache/iceberg/blob/main/docs/docs/branching.md

3 02. テーブル仕様 v1 / v2 / v3（と v4 の現況）

調査基準日: 2026-07-17 / **一次情報:** `format/spec.md`（main、2137 行を全文精査）

まず「フォーマットバージョン」とは何か。 Iceberg のテーブルには v1・v2・v3… というバージョンがあり、これは**テーブルのメタデータがどの決まりに従って書かれているか**を表します。ソフトウェアのファイル形式にバージョンがあるのと同じで、新しいバージョンほど新しい機能（行単位の削除、新しいデータ型など）を扱えます。逆に、v2 までしか理解しないツールは v3 の新機能を正しく読めません。だからバージョンを上げるかどうかは、**使いたい機能と、読み書きするツールの対応状況しだい**で決まります。

大きな流れはこうです。**v1** はファイルを並べるだけの素朴な形式、**v2** で「一部の行だけを消す・更新する」が効率的にできるようになり、**v3** で新しいデータ型や高度な機能が加わりました。**v4** はまだ策定中です。以下で 1 つずつ見ていきます。

3.1 最初に: バージョンの現況

仕様書の冒頭が、すべてを簡潔に述べています。

> Versions 1, 2 and 3 of the Iceberg spec are **complete and adopted by the community**.
> **Version 4 is under active development and has not been formally adopted.**

バージョン	名称	ステータス
v1	Analytic Data Tables	採択済み
v2	Row-level Deletes	採択済み。**現在のデフォルト**
v3	Extended Types and Capabilities	**採択済み**（1.8.0〜段階的に実装）
v4	Metadata Structure and Representation	**仕様は未採択**。ただし **Java 実装は 1.10.0+ で `format-version=4` を受け付ける**（PyIceberg は不可）。既定は 2

3.1.1 バージョニングの原則

> The format version number is incremented when new features are added that will break **forward-compatibility** --- that is, when **older readers would not read newer table features correctly**.

前方互換性が壊れるときにだけ上がります。「新機能が入ったから上げる」わけではありません。

3.2 v1: Analytic Data Tables

イミュータブルなファイル形式(Parquet, Avro, ORC)で大規模分析テーブルを管理する、最初の仕様です。データファイルは一度書いたら変更しない前提で、大量データの分析に向いています。その代わり、**行単位の削除・更新という仕組みがまだ無く**、一部の行を消すだけでもファイルを丸ごと書き直す必要がありました。これを解決したのが v2 です。

「All version 1 data and metadata files are valid after upgrading a table to version 2.」--- **v1 → v2 アップグレードでメタデータツリーの書き換えは不要**です。

3.3 v2: Row-level Deletes

v2 の目玉は、名前のとおり **行単位の削除(row-level delete)** です。「削除された行」を別の小さなファイル(delete file)に記録しておき、読み取り時に元データと突き合わせて除外する --- という仕組みを導入しました。おかげで、数行消すために巨大なデータファイルを書き直さずに済みます。`MERGE` / `UPDATE` / `DELETE` が現実的なコストで回るのは、これがあるからです。

仕様レベルの細かな変更点は次の通りです(Appendix E より):

- **delete file の追加**(position / equality)→ファイル書き換えなしの行単位削除・更新
- **シーケンス番号**の導入
- `data_file.content` 追加(0=data, 1=position deletes, 2=equality deletes)、`equality_ids` 追加
- manifest list `content` 追加(0=data, 1=deletes)
- **必須化**: `table-uuid`, `current-schema-id`, `schemas`, `partition-specs`, `default-spec-id`, `last-partition-id`, `sort-orders`, `default-sort-order-id`, snapshot の `manifest-list`, manifest list の各カウント類
- **削除**: `block_size_in_bytes`(**v1 reader 互換を破ると明記**), `file_ordinal`, `sort_columns`
- `manifest_entry.snapshot_id` が optional 化(継承サポートのため)
- 「Manifest list files are **required in v2**」

3.4 v3: Extended Types and Capabilities

v3 は、v2 までで扱えなかった **新しいデータ型** と、いくつかの**高度な機能**を加えたバージョンです。半構造化データ(variant)、地理データ(geometry / geography)、ナノ秒精度の時刻といった型に加え、列のデフォルト値、行の来歴を追う row lineage、より効率的な削除方式の deletion vector などが入りました。仕様が挙げる新機能は次の通りです:

- New data types: **nanosecond timestamp(tz)**, **unknown**, **variant**, **geometry**, **geography**
- **Default value support** for columns
- **Multi-argument transforms** for partitioning and sorting
- **Row Lineage** tracking

- **Binary deletion vectors**
- **Table encryption keys**

3.4.1 新しい型

型	説明
`unknown`	「Default / null column type used when a more specific type is not known」。**optional かつ null default 必須、データファイルに格納しない**。任意の型へプロモーション可能
`timestamp_ns` / `timestamptz_ns`	ナノ秒精度。Avro は `timestamp-nanos`（**Iceberg 独自規約、Avro spec には無い**と明記）、Parquet は `TIMESTAMP_NANOS`
`variant`	半構造化データ。**エンコーディングは Parquet プロジェクト側で定義**（VariantEncoding.md）、現状 **V1 のみ**。Avro では**シュレッディング非対応**
`geometry(C)`	OGC Simple feature access。エッジ補間は**常に線形/平面(Cartesian)**。CRS パラメータ C、既定 `OGC:CRS84`
`geography(C, A)`	CRS C ＋エッジ補間アルゴリズム A（`spherical`(既定) / `vincenty` / `thomas` / `andoyer` / `karney`）

CRS の重要な制約:

> CRS value **must not contain inlined PROJJSON definitions** and implementations **must not parse** the contents of the CRS as PROJJSON. PROJJSON definitions are very verbose, hence inlining them as part of schema would cause **significant performance degradation**.

`projjson:<property-name>` 形式でテーブルプロパティを参照します。

3.4.2 default values

- **`initial-default`**: フィールド追加**以前**に書かれた全レコードの値を埋めます →**データファイルを書き換えずに SQL の DEFAULT 挙動を実現**します
- **`write-default`**: フィールド追加**以後**、writer が値を供給しない場合に埋めます

規則:

- 「The `initial-default` is set only when a field is added to an existing schema.」
- 「If the write default for a **required** field is not set, the writer **must fail**.」
- **`unknown`, `variant`, `geometry`, `geography` は全て null default 必須**（非 null は invalid）
- ネスト struct の default は「null または**フィールド値を持たない非 null struct（{}）**」のみ

前方互換性の注意:「Old readers will **fail to read required fields** which are populated by `initial-default` because that default is not supported.」

実装状況の注意: Spark は default values の**書き込みを明示的に拒否**します（05 参照）。読み取り適用は 1.8.0 から対応。

3.4.3 multi-argument transforms

Partition Field / Sort Field JSON に **`source-ids`**（JSON int リスト）が追加されました。

- 「For partition fields with a transform with a **single argument, only `source-id` is written**. In case of a **multi-argument transform, only `source-ids` is written**.」--- 排他的です。
- v3 では `source-id` / `source-ids` **両方 optional**（v1/v2 では `source-id` が required）。

注記: 仕様の transform 一覧表には現時点で**複数引数の具体的な transform は載っていません**。仕組み（`source-ids`）のみが定義された状態です。将来の transform 追加のための基盤整備と読めます。

3.4.4 row lineage

> In v3 and later, an Iceberg table **must** track row lineage fields for all newly created rows.

v3 では必須で、オプトインではありません。 1.10.0 のリリースノート「remove update to enable row lineage as it is **always on for V3 table**」がこれを裏付けます。

2 つの行レベルフィールド:

- **`_row_id`**（field id 2147483540）：テーブル内で一意な long。行が最初に追加された時に**継承で割り当て**
- **`_last_updated_sequence_number`**（field id 2147483539）：最後に更新したコミットのシーケンス番号

なぜ継承なのか:

> the commit sequence number and starting row ID are **not assigned until the snapshot is successfully committed**. Inheritance is used to allow writing data and manifest files before values are known so that it is **not necessary to rewrite data and manifest files when an optimistic commit is retried**.

割り当ての連鎖: table `next-row-id` → snapshot `first-row-id` → manifest `first_row_id` → data file `first_row_id` → 行の `_row_id = first_row_id + _pos`

3.4.4.1 row lineage の重大な制約: equality delete では追跡されない

> engines using equality deletes **avoid reading existing data before writing changes** and can't provide the original row ID for the new rows. These updates are always treated as if the existing row was **completely removed and a unique new row was added**.

CDC 用途で row lineage を当てにするなら、equality delete を使うエンジン（典型的には Flink の upsert）では機能しません。 これは v3 の目玉機能の実用上最大の穴です。

3.4.4.2 アップグレード時の非対称性

v3 化で `next-row-id` は 0 に初期化され、既存スナップショットは変更されず `first-row-id` は未設定のままです。→ **アップグレード前のスナップショットには row ID が無く、`_row_id` は全行 null として読まれます。**後付けできません。

行を別ファイルに移動する場合（compaction 等）の規則:

1. 既存の非 null `_row_id` をコピー
2. 変更していれば `_last_updated_sequence_number` を null に
3. 変更していなければ既存値をコピー

3.4.5 table encryption keys

`encryption-keys`（`key-id` / `encrypted-key-metadata`(base64) / `encrypted-by-id` / `properties`）を table metadata に持ち、snapshot は `key-id` で参照します。1.10.0 で仕様に追加されました。

3.5 Copy-on-Write と Merge-on-Read

行を更新・削除するとき、Iceberg には **2 つの書き込み方**があります。

- **Copy-on-Write（CoW）**: 変更が及ぶデータファイルを**丸ごと書き直す**方式。書き込みは重いですが、読み取りは速い（読む側は特別なことをしなくてよい）。
- **Merge-on-Read（MoR）**: 元のデータファイルはそのままに、「この行は消えた」という情報を別ファイルに**追記するだけ**の方式。書き込みは軽いですが、読み取り時に元データと差分を突き合わせる手間がかかります。

どちらを使うかはテーブルプロパティで選びます。既定は CoW です。

3.5.1 デフォルトは CoW（重要）

プロパティ	デフォルト	説明
`write.delete.mode`	**copy-on-write**	copy-on-write または merge-on-read（**v2 以上**）
`write.update.mode`	**copy-on-write**	同上
`write.merge.mode`	**copy-on-write**	同上
`write.delete.granularity`	**partition**	partition または file
`write.delete.isolation-level`	serializable	serializable または snapshot
`format-version`	**2**	「Defaults to 2 since version 1.4.0.」

Iceberg のテーブルはデフォルトで format-version=2 かつ CoW です。MoR を使うには明示設定が必要です。そして **v3 にしても DV は自動では使われません**（`write.delete.mode` の既定が CoW のままのため）。見落とされやすい点です。

なお仕様自体は CoW/MoR という用語で書き分けておらず、これはエンジン側（Spark 等）の書き込み戦略の設定です。

3.6 delete の 3 形式

行を削除したとき、その「削除」をどう記録するかに 3 つの方式があります。いずれも**元のデータファイルはいじらず、削除の情報を別に持つ**（= MoR）点は同じで、記録の仕方が異なります。

> Row-level deletes are added by **v2** and are **not supported in v1**. Deletion vectors are added in **v3** and are **not supported in v2** or earlier. **Position delete files must not be added to v3 tables**, but existing position delete files are valid.

形式	導入	識別方法	状態
Position delete file	v2	データファイルパス + 行位置	**v3 で deprecated**、v3 テーブルへの新規追加は禁止
Equality delete file	v2	1 つ以上の列値（例 `id = 5`）	現役
Deletion vector (DV)	**v3**	単一データファイル内の位置ビットマップ	v3 の推奨形式

3.6.1 Position delete file

`file_position_delete` 構造体: `file_path`（string）/ `pos`（long, 0 始まり）/ `row`（optional struct）

- 「The rows in the delete file **must be sorted by `file_path` then `pos`**」--- `file_path` ソートは列指向でのフィルタプッシュダウン、`pos` ソートは**削除情報をメモリに保持せずスキャンできるようにする**ため。
- `row` 列は「present なら required」（統計の正確性を担保するため）。ただし「no implementation currently writes it … **deprecated in version 1.11.0**」。

3.6.2 Equality delete file

- 各行が 1 つの等値述語を生成し、複数列は `AND` として扱われます。**`null` は `col IS NULL` と等価にマッチ**します。
- **列が後で drop されても、equality delete の適用時には使い続けなければなりません。**

3.6.3 Deletion vector (DV)

Puffin の **`deletion-vector-v1`** blob 型で格納されます。

- **Roaring bitmap** ベース。64-bit 位置をサポートしつつ「optimized for cases where most positions fit in 32 bits」--- 上位 4 バイトを "key"、下位 4 バイトをサブ位置とし、key ごとに 32-bit Roaring bitmap を保持。
- **スナップショット内で 1 データファイルにつき DV は最大 1 つ**。
- 「If a DV is written for a data file, it must **replace all previously written position delete files** so that when a DV is present, **readers can safely ignore matching position delete files**.」
- manifest 側は `file_path` / `content_offset` / `content_size_in_bytes` で DV blob を個別追跡。**Puffin footer の値と厳密一致必須**。
- **1 つの Puffin ファイルに複数の DV を格納可能**、参照先データファイルに制限なし。
- データファイル削除時は delete manifest から対応 DV も削除必須。ただし **Puffin ファイル自体の書き直しは不要**。

シリアライズ形式:

- 4 バイト big-endian: ベクトル＋マジックバイトの長さ
- 4 バイトマジック: **`D1 D3 39 64`**
- Roaring bitmap "portable" format（**little-endian**）
- 4 バイト big-endian: CRC-32
- 「Big endian values were chosen for **compatibility with existing deletion vectors in Delta tables**.」--- **Delta 互換を意識した設計**です
- `properties` に `referenced-data-file` と `cardinality` 必須、`compression-codec` は省略必須（非圧縮）
- 「`snapshot-id` and `sequence-number` must be set to **-1**」（Puffin 作成時点で未確定のため）

3.6.4 delete 適用のスコープ規則（実務上最重要）

形式	適用条件
DV	`file_path` == `referenced_data_file` **かつ** データシーケンス番号 **≤** DV のそれ **かつ**パーティション（spec と値の両方）が一致
Position delete	同様 ＋「**適用すべき DV が存在しないこと**」
Equality delete	データシーケンス番号が**厳密に小さい（strictly less than）** ＋ パーティション一致**または delete 側の spec が unpartitioned**

2 つの特例:

1. 「Equality delete files stored with an **unpartitioned spec are applied as global deletes**.」--- **グローバル equality delete は全データファイルに対して照合が必要**になります。MoR の最悪ケースです。
2. 「Position deletes (vectors and files) **must be applied to data files from the same commit**, when the data and delete file data sequence numbers are **equal**.」--- 同一コミット内で追加した行を削除できるようにするため。

→ **position delete は ≤、equality delete は<** という非対称性が肝です。

3.6.5 DV が改善したこと

1. **実行時効率**: 仕様が「**deletion vector は単一データファイルに対する delete のバイナリ表現であり、実行時に position delete ファイルより効率的**」と明言。
2. **不変条件による予測可能性 --- これが最大の改善**:「1 データファイルにつき DV は高々 1 つ」。v2 の position delete は 1 データファイルに対して**無制限に**積み上がりえたため、読み取りコストが「これまで何回 delete したか」に依存して**単調悪化**しました。v3 ではこれが構造的に不可能になり、**読み取りコストが delete 回数に依存しなくなります**。定量的改善ではなく漸近的な性質の変化です。
3. **メンテナンス負荷の消滅**: 上記の帰結として、v2 で必須だった「delete ファイルの minor compaction」というタスク自体が不要になります。
4. **直接アクセス**: `content_offset` / `content_size_in_bytes` により、Puffin ファイル全体を読まずに必要な DV blob だけレンジ取得できます。

ただし残る問題:「writer は削除された DV を含む Puffin ファイルを書き直す必要はない」ため、**参照されない DV blob を含む Puffin ファイルが物理的に残りえます**。`remove_orphan_files` の領分です。DV は運用を楽にしますが、ゼロにはしません。

3.7 スキーマ進化

3.7.1 field ID ベース

> Columns in Iceberg data files are **selected by field id**. The table schema's column names and order may change after a data file is written, and projection **must** be done using field ids.

Rename は field ID を変えません。「Renaming an existing field must change the name, **but not the field ID**.」

3.7.2 型プロモーション

元の型	v1, v2	**v3+**
`unknown`	---	**任意の型へ**
`int`	`long`	`long`
`date`	---	**`timestamp`, `timestamp_ns`** （`timestamptz` 系へは**不可**）
`float`	`double`	`double`
`decimal(P,S)`	`decimal(P',S)` P'>P	同左（精度拡大のみ）

実務上の落とし穴:

> Iceberg's Avro manifest format **does not store the type of lower and upper bounds**, and type promotion does not rewrite existing bounds.

bounds のバイト長から書き込み時の型を推論する必要があります（`long` 4 バイト → 元は `int`、`double` 4 バイト → 元は `float` 等）。仕様に推論テーブルが定義されています。

禁止事項: struct のフィールドをネスト struct にグルーピングする／その逆（`struct<a,b,c>` ↔ `struct<a,struct<b,c>>`）は**不可**。プリミティブ↔ struct も不可。

bucket transform との相互作用:

> `bucket[N]` produces different hash values for `34` and `"34"` (2017239379 != -427558391) but the same value for `34` and `34L`

パーティションソース列は、transform 結果が変わるプロモーションが禁止されます。

3.7.3 列プロジェクションの解決順序

データファイルに field id が無い場合:

1. identity transform のパーティションメタデータから値を返す（**Hive テーブルのメタデータのみの移行**を可能にする）
2. `schema.name-mapping.default` で field id をマップ
3. `initial-default` があればそれを返す
4. それ以外は `null`

3.8 パーティション進化と transform

3.8.1 partition transform

transform	説明	ソース型	結果型
`identity`	そのまま	`geometry`/`geography`/`variant`**以外**の任意	ソース型
`bucket[N]`	ハッシュ mod N	int, long, decimal, date, time, timestamp(tz), timestamp(tz)_ns, string, uuid, fixed, binary	`int`
`truncate[W]`	幅 W で切り詰め	int, long, decimal, string, binary	ソース型
`year`	1970 年からの年数	date, timestamp(tz), timestamp(tz)_ns	`int`
`month`	1970-01-01 からの月数	同上	`int`
`day`	1970-01-01 からの日数	同上	**`date`**

transform	説明	ソース型	結果型
`hour`	1970-01-01 00:00:00 からの時間数	timestamp(tz), timestamp(tz)_ns	`int`
`void`	常に `null`	任意	ソース型 or `int`

細部(間違えやすい点):

- **全 transform は `null` 入力に対し `null` を返す**こと
- `day` の結果型は `date`。ただし「Readers must also **accept `int` values** for the `day` transform」(後方互換)
- **bucket のハッシュは 32-bit Murmur3、x86 variant、seed=0**。`bucket_N(x) = (murmur3_x86_32_hash(x) & Integer.MAX_VALUE) % N`(符号ビットを捨てて正値化)
- `truncate` の負値:「remainders must be positive」→ `v - (((v % W) + W) % W)`。`W=10` で `-1` → `-10`
- `truncate` の decimal は**列のスケールを使って** W を適用(`W=50, s=2`: `10.65` → `10.50`)。string は**コードポイント**単位、binary は**バイト**単位
- `void` は v1 テーブルでフィールドを実質的に drop するために使う

3.8.2 進化の規則

- スペック変更は新しい spec ID を生成し、テーブルの spec 一覧に追加されます。**既存データは書き換えません。**
- 「changes should not cause **partition field IDs to change** because the partition field IDs are used as the partition tuple field IDs in manifest files.」
- **v2 では partition field ID を明示的に追跡必須。v1 では追跡されず** 1000 から順に割り当てられていました。これが「複数スペックの manifest を跨いでメタデータテーブルを読むと、同じ ID のフィールドが異なる型を持ちうる」問題を生みました。
- **v1 テーブルでの推奨ルール**: ①フィールドを並べ替えない ② drop せず `void` transform に置換する ③末尾にのみ追加する
- 未知の transform: v1/v2 では reader は無視すべき、**v3 では無視が必須**。writer は未知 transform を含む spec でのコミットは**禁止**

3.9 sort order

- 構成要素: source column id(s) / transform / direction(`asc`|`desc`) / null order(`nulls-first`|`nulls-last`)
- **order id `0` は unsorted 用に予約**
- 浮動小数点の順序: `-NaN < -Infinity < -value < -0 < 0 < value < Infinity < NaN`
- 「Writers **should** use this default sort order to sort the data on write, but are **not required to** if the default order is prohibitively expensive, **as it would be for streaming writes**.」
- **position delete file は sort order id を null にすること**、reader は position delete file の sort order id を**無視しなければならない**

3.10 v3 の策定・リリース状況

3.10.1 Apache 側の公式ステータス

「v3 が GA」という宣言は Apache の公式リリースノート・仕様のいずれにも存在しません。公式ステータスは仕様冒頭の「complete and adopted by the community」という表現に留まります。**「GA」はベンダー側の製品提供状況を指す用語です。**

3.10.2 参照実装（Java）のリリース経緯

リリース	日付	v3 関連の主な内容
1.8.0	2025-02-13	**Deletion vectors を table spec に追加**、**Variant Type 追加**、`EnableRowLineage` メタデータ更新、snapshot に `added-rows` 追加。API: **UnknownType 追加**
1.9.0	2025-04-28	equality delete と row lineage の挙動定義、**V3 での source-id 使用許可**。Core: **全 v3 テーブルで row lineage を有効化**、**geometry / geography 型サポート追加**、Variant reader/writer 実装
1.10.0	2025-09-11	孤立 DV 防止の write 要件明確化、**encryption keys 追加**、default values の struct フィールド衝突回避。REST: **「row lineage is always on for V3 table」のためオプトイン更新を削除**
1.11.0	**2026-05-19**	**SQL UDF 仕様の導入**、snapshot に `added-rows` を復活、**position delete files with row data を deprecated 化**。API: **V4 manifest サポートの基盤型を導入**、**content stats のクラス導入**。Java 11 サポート終了

v3 の機能は 1.8.0（2025 年 2 月）から段階的に投入され、1.9.0 で geometry/geography と row lineage の全 v3 テーブル有効化、1.10.0 で encryption keys まで到達しました。

3.10.3 ベンダーの GA 状況

ベンダー	v3 の状態	日付	備考
Snowflake	**GA**	2026-05-07	新規 managed テーブルの既定は**依然 v2**（v3 はオプトイン）
Databricks	**GA**	2026-05-28	2026-04-09 に Public Preview → 5/28 GA。**defaults / unknown 型 / ns timestamp / multi-arg transforms は明示的に非対応**
Dremio	**Preview**	---	プレスリリース（2026-04-06）は “at GA” と書くが、**docs は今も “Iceberg v3 Preview” と明記**。Cloud のみで Software 側には記述なし

詳細は 05-engine-support.md を参照。

3.10.4 デフォルトは依然 v2

`format-version` プロパティのデフォルトは **2**（「Defaults to 2 since version 1.4.0.」）。v3 を使うには明示的な指定/アップグレードが必要です。

3.11 v4: Metadata Structure and Representation — 未採択、ただし実装は先行している

3.11.1 現況 —「未採択」と「設定できない」は別物です

仕様本文は「**Version 4 is under active development and has not been formally adopted.**」と明記しています。**仕様としては未採択です。**

しかし、参照実装は既に v4 を受け付けます。ここを取り違えないでください。

実装	format-version=4	根拠（実測）
Iceberg Java 1.10.0+（2025-09-11〜）	**設定できる**	TableMetadata.java: SUPPORTED_TABLE_FORMAT_VERSION = 4。バリデーションは formatVersion <= SUPPORTED_TABLE_FORMAT_VERSION なので v4 は通過する
Iceberg Java 1.9.0 以前	設定できない	同定数が 3
PyIceberg 0.11.1	**設定できない**	typedef.py: TableVersion = Literal[1, 2, 3]。metadata.py: SUPPORTED_TABLE_FORMAT_VERSION = 2

リリースタグごとの実測値:

タグ	SUPPORTED_TABLE_FORMAT_VERSION
apache-iceberg-1.8.0	3
apache-iceberg-1.9.0	3
apache-iceberg-1.10.0（2025-09-11）	**4**
apache-iceberg-1.11.0（2026-05-19）	4

さらに main の TableMetadata.java には **v4 機能のゲートがコードとして存在**します:

```
static final int DEFAULT_TABLE_FORMAT_VERSION = 2;
static final int SUPPORTED_TABLE_FORMAT_VERSION = 4;
static final int MIN_FORMAT_VERSION_ROW_LINEAGE = 3;
static final int MIN_FORMAT_VERSION_PARQUET_MANIFESTS = 4;   // ← v4 機能
static final int MIN_FORMAT_VERSION_OPTIONAL_LOCATION = 4;   // ← v4 機能（相対パス）
```

関連コミット: Core: Add basic classes for writing table format-version 4 (#13123)（2025-06-04）、Core, Parquet: Allow for Writing Parquet/Avro Manifests in V4 (#15634)（2026-05-22）、Core: v4 table metadata location should be optional (#16572)（2026-06-01）。

正確な言い方:「v4 は**仕様として未採択**だが、**Java 実装では 1.10.0 以降 format-version=4 の設定自体は可能**。ただし仕様が流動的なため本番利用は想定されていない。**既定値は依然として 2**。

> **この区別が重要な理由**:「未採択だから触れない」と「実装が受け付けない」は違います。前者は**あなたの判断**の問題ですが、後者は**物理的な制約**です。Java では前者しか存在しません。**仕様が固まる前の v4 テーブルを作れてしまう**ということでもあり、それはそれでリスクです。

3.11.2 なぜ「公開された」と誤解されやすいか

紛らわしい材料が揃っています。

1. **仕様書に既に v4 の記述が入っている。**main の `format/spec.md` には「Version 4: Metadata Structure and Representation」という節があり、相対パス対応や `content_stats` が書かれています。**未採択のドラフトが同じファイルに同居している**状態です。
2. **参照実装に v4 の足場が入り始めた。**1.11.0（2026-05-19）に「Introduce foundational types for V4 manifest support」「Introduce classes for content stats」が入っています。**コードが入った ≠ 仕様が採択された。**
3. **Iceberg Summit 2026 で大きく取り上げられた。**Adaptive Metadata Tree、single-file commit、column families、拡張可能な列統計といった提案が語られ、ブログが大量に出ました。**提案の議論 ≠ 採択。**
4. **v3 の「GA」ラッシュと混ざりやすい。**

3.11.3 公式マイルストーンの実態

Iceberg V4 Spec マイルストーン #58 に載っているのは **2 件だけ**（Open 2 / Closed 0、**期日なし**）：

1. **Column Stats Improvements**（#13153、2025-05-26）
2. **Relative Path Support In Table Spec**（#13141、2025-05-23）

Summit やブログで語られる Adaptive Metadata Trees・column families・single-file commits は、公式マイルストーン上にはまだ存在しません。

3.11.4 仕様に現時点で書かれている v4 の内容

- **相対パス（relative locations）**：「enabling tables to be **relocated without rewriting metadata files**」。table metadata の `location` が **optional** 化。
- **`content_stats`（型付き列統計）**：v3 までの `map<int, binary>` 形式を**型付きの構造体**に置き換え。新規メトリクスとして **`tight_bounds`**（true なら bounds が実際の min/max と厳密一致）と **`avg_value_size_in_bytes`**。
- **File System Tables 方式の削除**:「will be removed in version 4 of this spec」

3.11.5 実務上の結論

v4 を前提にした設計判断は時期尚早です。公式マイルストーンは 2 提案のみ、期日なし。ブログ等で語られるタイムライン（2027 年前後）は**公式には裏取りできません。**

ただし「作れないから安全」ではありません。上記のとおり Iceberg Java 1.10.0+ は `format-version=4` を受け付けます。**仕様が採択前に変更されれば、作ってしまった v4 テーブルが読めなくなりかねません。**明示的に避ける判断が必要です。

ただし、v4 の方向性は業界動向として重要です。**Databricks が Delta 5.0 に Iceberg v4 の adaptive metadata tree 構造を採用する提案を公式ブログで出しています** → 07-ecosystem.md。

3.12 実務者向けサマリ

1. **デフォルトは v2 + copy-on-write。**v3 の恩恵（DV、row lineage 等）を得るには**明示的なアップグレードと設定変更の両方**が必要です。v3 にしただけでは DV は使われません。
2. **v3 で position delete file は事実上終了。**v3 テーブルへの新規追加は**禁止**。v2 から上げた場合、既存 position delete file は有効ですが、対象データファイルの DV を作る際に**マージが必須**です。
3. **row lineage は v3 で必須かつ後付け不可。**アップグレード前のスナップショットには row ID が無く `_row_id` は null。**CDC 目的なら、equality delete を使うエンジン（Flink upsert 等）では追跡されない**点が最大の制約です。
4. **v4 は仕様未採択だが、Java 実装は受け付ける。**公式マイルストーンは 2 提案のみ。**Iceberg Java 1.10.0+ で `format-version=4` は設定でき、PyIceberg では設定できません。**「未採択」と「設定不可」を取り違えないこと。v4 前提の計画は時期尚早ですが、理由は「作れない」ではなく「**仕様が変わりうる**」です。
5. **File System Tables（HDFS の atomic rename 方式）は使わない。**仕様が deprecated +「オブジェクトストアとローカルファイルシステムでは **unsafe**」+ v4 で削除予定と宣言しています。
6. **統計は「あれば速くなる」もので、正しさには不要。**Puffin の theta sketch（NDV）もパーティション統計も informational です。

3.13 出典

- Iceberg Table Spec: https://iceberg.apache.org/spec/ / 生ソース https://github.com/apache/iceberg/blob/main/format/spec.md
- Appendix E（バージョン差分）: https://iceberg.apache.org/spec/#appendix-e-format-version-changes
- Puffin Spec: https://iceberg.apache.org/puffin-spec/
- 公式リリースノート: https://iceberg.apache.org/releases/ / https://github.com/apache/iceberg/blob/main/site/docs/releases.md
- Configuration（CoW/MoR、format-version 既定値）: https://github.com/apache/iceberg/blob/main/docs/docs/configuration.md
- Iceberg V4 Spec マイルストーン: https://github.com/apache/iceberg/milestone/58
- Snowflake Iceberg v3 GA（2026-05-07）: https://docs.snowflake.com/en/release-notes/2026/other/2026-05-07-iceberg-v3-ga
- Databricks Iceberg v3 GA（2026-05-28）: https://www.databricks.com/blog/unity-catalog-and-next-era-apache-icebergtm
- Iceberg Summit 2026 recap（v4、二次情報）: https://www.snowflake.com/en/blog/engineering/iceberg-summit-2026-recap-v4-spec/

4 03. Iceberg REST Catalog (IRC) 仕様

調査基準日: 2026-07-17 / **一次情報:** `open-api/rest-catalog-open-api.yaml`（main、5822 行を精査）

> **出典の注記**: https://iceberg.apache.org/rest-catalog-spec/ は Swagger UI がクライアント側で描画する方式のため、仕様本文をそのまま取り込めません。そこで本ドキュメントは、同ページが読み込む大元の YAML を出典としています。

4.1 まず知るべきこと: 仕様にバージョンが無い

```yaml
openapi: 3.1.1
info:
  title: Apache Iceberg REST Catalog API
  version: 0.0.1     # ← ずっとこのまま
```

`info.version` は `0.0.1` のままです。IRC 仕様には意味のあるセマンティックバージョンが付いていません。

これは実務上重い意味を持ちます。「このカタログは IRC v1.2 準拠」といった表現は成立せず、**機能の有無は Iceberg のリリース番号と、実行時の `GET /v1/config` の `endpoints` フィールドで判断するしかありません**。カタログ実装を比較する際に「仕様準拠度」を単一の数値で語れないのはこのためです。

4.2 1. なぜ REST Catalog が生まれたか

公式の動機説明は、用語集 `site/docs/terms.md` の「Decoupling Using the REST Catalog」節が唯一です。

> The REST catalog was introduced in the **Iceberg 0.14.0** release and provides greater control over how Iceberg catalogs are implemented. Instead of using technology-specific logic contained in the catalog clients, **the implementation details of a REST catalog lives on the catalog server**. … The server-side logic can be written in **any language** and use any custom technology, as long as the API follows the Iceberg REST Open API specification.

> A great benefit of the REST catalog is that it allows you to use **a single client to talk to any catalog backend**. This increased flexibility makes it easier to make custom catalogs compatible with engines like Athena or Starburst **without requiring the inclusion of a Jar into the classpath**.

課題	REST Catalog による解決
カタログ実装の乱立（Hive Metastore, JDBC, Nessie, Glue, DynamoDB, Hadoop + ベンダー独自）	単一の OpenAPI 契約に統一。「a single client to talk to any catalog backend」
クライアント側にテクノロジ固有ロジックが分散（コミットロジック・ロック・リトライがクライアント JAR に埋め込まれる）	「implementation details … lives on the catalog server」。サーバ側は任意の言語で実装可
JAR クラスパス依存	「without requiring the inclusion of a Jar into the classpath」
多言語対応の困難（Thrift ベースの HMS は実質 JVM 前提）	HTTP/JSON なので Python / Rust / Go から素直に実装可能

カタログの最重要責務についての公式定義:

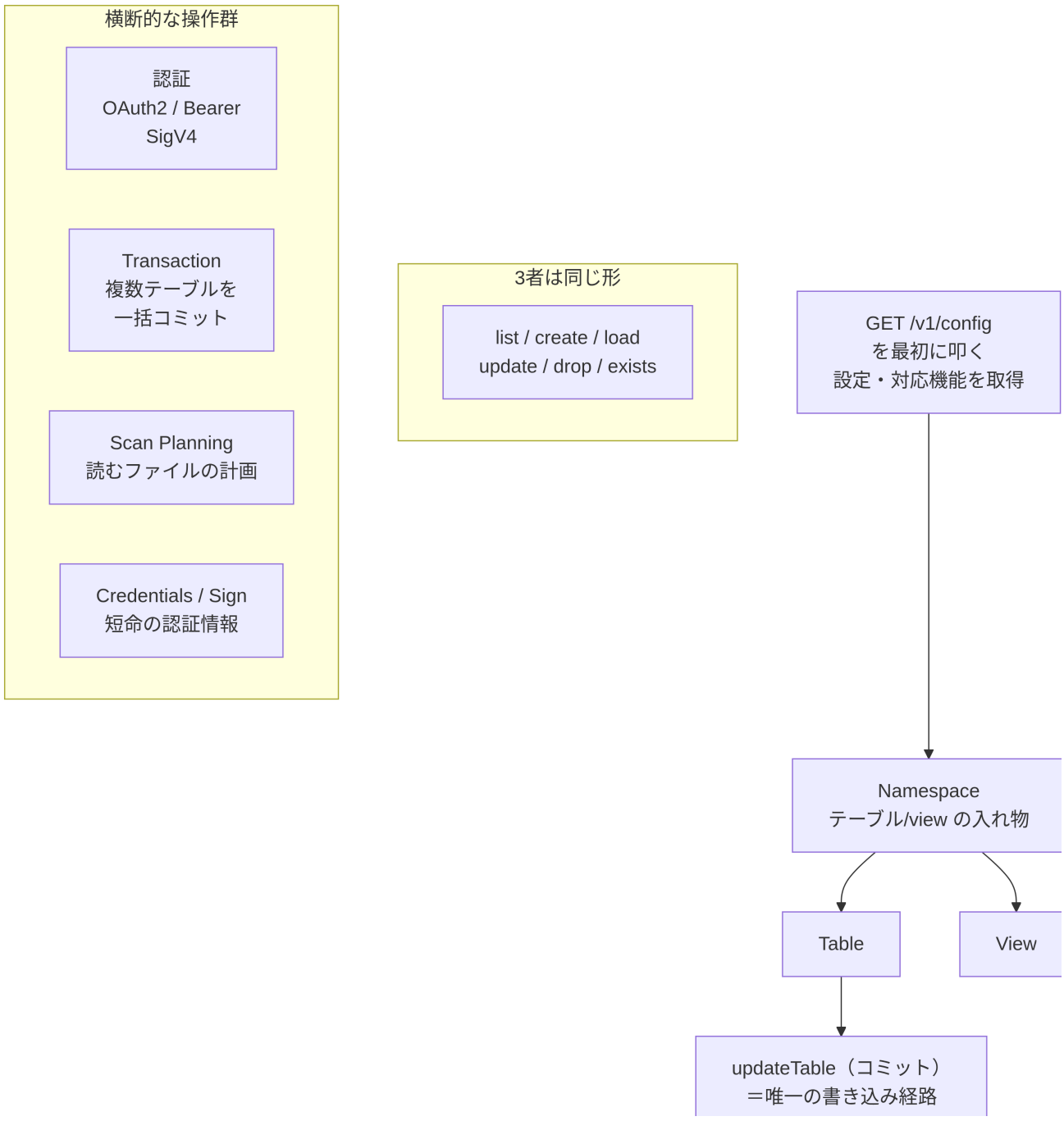
> The most important responsibility of a catalog is **tracking a table's current metadata**, which is provided by the catalog when you load a table.

これは 01-architecture.md で見た「仕様は commit の atomic 操作を標準化していない（カタログ実装依存）」という事実と表裏です。**カタログこそが ACID の実行主体**であり、その実装が乱立していたことが問題でした。

> **未確認事項**:「Hive Metastore は Thrift で JVM を要求する」「HMS はロッキングの問題を抱えていた」といった具体的な HMS 批判は、**Apache Iceberg の公式一次情報には見つかりませんでした**。これらはベンダーブログ由来の説明です。公式は Hive Thrift service を「批判」ではなく「類似のアーキテクチャ」として引き合いに出しているのみです。

4.3 2. エンドポイント一覧（全 35 オペレーション）

IRC の API は、大きく **リソース階層（namespace の中に table と view がある）への CRUD** と、それを支える**横断的な操作群**（設定取得・認証・トランザクション・スキャンプランニング・認証情報の払い出し）でできています。namespace / table / view はどれも **list（一覧）・create（作成）・load（読み込み）・update（更新）・drop（削除）・exists（存在確認）**という同じ形をしているので、1 つ分かれば残りも読めます。



4.3.1 Configuration

クライアントが**最初に叩くべき**エンドポイントです。サーバの設定と対応機能（ケイパビリティ）を受け取ります（詳細は §6）。

パス	メソッド	operationId	用途
/v1/config	GET	getConfig	カタログ設定取得。**prefix を取らない唯一のパス**。?warehouse= 対応

4.3.2 OAuth2（非推奨）

トークン取得用ですが、セキュリティ上の理由で**非推奨・削除予定**です。本番では外部の IdP を使います（§4）。

パス	メソッド	operationId	用途
/v1/oauth/tokens	POST	getToken	**DEPRECATED for REMOVAL**（deprecated: true）→ §4

4.3.3 Namespace

テーブルや view をまとめる入れ物（データベースのスキーマに相当）を操作します。

パス	メソッド	operationId
/v1/{prefix}/namespaces	GET	listNamespaces（?parent= で階層指定、ページネーション対応）
/v1/{prefix}/namespaces	POST	createNamespace
/v1/{prefix}/namespaces/{namespace}	GET	loadNamespaceMetadata
/v1/{prefix}/namespaces/{namespace}	HEAD	namespaceExists
/v1/{prefix}/namespaces/{namespace}	DELETE	dropNamespace
/v1/{prefix}/namespaces/{namespace}/properties	POST	updateProperties

namespace の区切り文字（見落としやすい）：

> Multipart namespace parts must be separated by the namespace separator as indicated via the /config override `namespace-separator`, which **defaults to the unit separator `0x1F` byte (url encoded `%1F`)**. To be compatible with older clients, servers must use **both** the advertised separator and `0x1F` as valid separators when decoding namespaces.

例: `accounting.tax` → `GET /v1/{prefix}/namespaces/accounting%1Ftax`

4.3.4 Table

テーブルの CRUD です。中でも `updateTable` が**唯一の書き込み経路**で、追記も削除もコミットはすべてここを通ります（§3）。

パス	メソッド	operationId	用途
`/v1/{prefix}/namespaces/{namespace}/tables`	GET	`listTables`	
`/v1/{prefix}/namespaces/{namespace}/tables`	POST	`createTable`	`stage-create: true` でステージング
`/v1/{prefix}/namespaces/{namespace}/tables/{table}`	GET	`loadTable`	`LoadTableResult` を返す
`/v1/{prefix}/namespaces/{namespace}/tables/{table}`	POST	`updateTable`	**コミット** → §3
`/v1/{prefix}/namespaces/{namespace}/tables/{table}`	DELETE	`dropTable`	`?purgeRequested=`
`/v1/{prefix}/namespaces/{namespace}/tables/{table}`	HEAD	`tableExists`	
`/v1/{prefix}/namespaces/{namespace}/register`	POST	`registerTable`	既存 metadata file location で登録
`/v1/{prefix}/namespaces/{namespace}/tables/{table}/unregister`	POST	`unregisterTable`	データを消さずに登録解除
`/v1/{prefix}/tables/rename`	POST	`renameTable`	namespace 跨ぎ可
`/v1/{prefix}/namespaces/{namespace}/tables/{table}/metrics`	POST	`reportMetrics`	

4.3.5 View

SQL の view を操作します。テーブルと同じ CRUD の形です。

パス	メソッド	operationId
/v1/{prefix}/namespaces/{namespace}/views	GET / POST	listViews / createView
/v1/{prefix}/namespaces/{namespace}/views/{view}	GET / POST / DELETE / HEAD	loadView / replaceView / dropView / viewExists
/v1/{prefix}/views/rename	POST	renameView
/v1/{prefix}/namespaces/{namespace}/register-view	POST	registerView（**1.11.0 で実装**、PR #14870）

4.3.6 Function（新しい領域）

SQL の関数（UDF）を扱う新しい領域です。今のところ read 系（一覧・読み込み）のみ。

パス	メソッド	operationId
/v1/{prefix}/namespaces/{namespace}/functions	GET	listFunctions
/v1/{prefix}/namespaces/{namespace}/functions/{function}	GET	loadFunction

1.11.0 で「Introduce SQL UDF specification」（PR #14117）が入り、site/docs/udf-spec.md が存在します。**両者を結びつける明示的な公式記述は未確認**ですが、対応するものと考えられます。

4.3.7 Transaction

複数のテーブルへの変更を、**1 つの単位でまとめてコミット**します。

パス	メソッド	operationId	用途
/v1/{prefix}/transactions/commit	POST	commitTransaction	**複数テーブルのアトミックコミット**。CommitTransactionRequest { table-changes: [CommitTableRequest] }

4.3.8 Scan Planning

「どのファイルを読むか」の計画をサーバ側で行う仕組みです（§7）。

パス	メソッド	operationId
/v1/{prefix}/namespaces/{namespace}/tables/{table}/plan	POST	planTableScan
/v1/{prefix}/namespaces/{namespace}/tables/{table}/plan/{plan-id}	GET	fetchPlanningResult
/v1/{prefix}/namespaces/{namespace}/tables/{table}/plan/{plan-id}	DELETE	cancelPlanning
/v1/{prefix}/namespaces/{namespace}/tables/{table}/tasks	POST	fetchScanTasks

→ §7

4.3.9 Credentials / Remote Signing

ストレージへアクセスするための**短命の認証情報を払い出す/署名する**エンドポイントです(§5)。

パス	メソッド	operationId
/v1/{prefix}/namespaces/{namespace}/tables/{table}/credentials	GET	loadCredentials（?planId=, ?referenced-by=）
/v1/{prefix}/namespaces/{namespace}/tables/{table}/sign	POST	signRequest（**S3 remote signing**）

→ §5

4.3.10 横断パラメータ

特定のグループに属さず、多くのエンドポイントで共通して使うパラメータです。

パラメータ	位置	内容
prefix	path	「An optional prefix in the path」→ §6
X-Iceberg-Access-Delegation	header	vended-credentials / remote-signing（カンマ区切り）
Idempotency-Key	header	UUID（36 文字固定）。「Optional client-provided idempotency key for **safe request retries**」

パラメータ	位置	内容
`pageToken` / `pageSize`	query	ページネーション
`referenced-by`	query	view 経由アクセス時の参照チェーン

4.4 3. コミットプロトコル

4.4.1 仕様本文

> Commits have two parts, **requirements** and **updates**. Requirements are assertions that will be validated before attempting to make and commit changes. … **Server implementations are required to fail with a 400 status code if any unknown updates or requirements are received.**

「unknown requirement は 400 で落とす」という規約が楽観ロックの安全性の要です。サーバが知らない assertion を黙って無視すると、クライアントが期待した検証が行われないままコミットが通ってしまいます。

4.4.2 requirements 一覧

type	追加フィールド	意味
`assert-create`	---	テーブルが未存在であること（create transaction 用）
`assert-table-uuid`	`uuid`	table UUID の一致
`assert-ref-snapshot-id`	`ref`, `snapshot-id`（nullable）	指定 ref が指定スナップショットを参照。**`null` なら「ref がまだ存在しないこと」の表明**
`assert-last-assigned-field-id`	`last-assigned-field-id`	最後に割り当てられた列 ID の一致
`assert-current-schema-id`	`current-schema-id`	現在のスキーマ ID の一致
`assert-last-assigned-partition-id`	`last-assigned-partition-id`	最後のパーティション ID の一致
`assert-default-spec-id`	`default-spec-id`	デフォルト spec ID の一致
`assert-default-sort-order-id`	`default-sort-order-id`	デフォルト sort order ID の一致

4.4.3 updates 一覧（23 種）

`AssignUUIDUpdate`, `UpgradeFormatVersionUpdate`, `AddSchemaUpdate`, `SetCurrentSchemaUpdate`, `AddPartitionSpecUpdate`, `SetDefaultSpecUpdate`, `AddSortOrderUpdate`, `SetDefaultSortOrderUpdate`, **`AddSnapshotUpdate`**, **`SetSnapshotRefUpdate`**,

RemoveSnapshotsUpdate, RemoveSnapshotRefUpdate, SetLocationUpdate, SetPropertiesUpdate, RemovePropertiesUpdate, SetStatisticsUpdate, RemoveStatisticsUpdate, SetPartitionStatisticsUpdate, RemovePartitionStatisticsUpdate, RemovePartitionSpecsUpdate, RemoveSchemasUpdate, **AddEncryptionKeyUpdate**, **RemoveEncryptionKeyUpdate**

4.4.4 具体例（append コミット）

```
POST /v1/prod/namespaces/accounting%1Ftax/tables/paid HTTP/1.1
Authorization: Bearer eyJhbGciOi...
Content-Type: application/json
Idempotency-Key: 017F22E2-79B0-7CC3-98C4-DC0C0C07398F
```

```
{
  "identifier": { "namespace": ["accounting", "tax"], "name": "paid" },
  "requirements": [
    { "type": "assert-table-uuid",
      "uuid": "9c12d441-03fe-4693-9a96-a0705ddf69c1" },
    { "type": "assert-ref-snapshot-id",
      "ref": "main",
      "snapshot-id": 3051729675574597004 }
  ],
  "updates": [
    { "action": "add-snapshot",
      "snapshot": {
        "snapshot-id": 3055729675574597004,
        "parent-snapshot-id": 3051729675574597004,
        "sequence-number": 34,
        "timestamp-ms": 1785000000000,
        "manifest-list": "s3://bucket/warehouse/.../metadata/snap-3055729675574597004-1-.avro",
        "summary": { "operation": "append", "added-data-files": "4", "added-records": "10000" },
        "schema-id": 0
      }
    },
    { "action": "set-snapshot-ref",
      "ref-name": "main", "type": "branch",
      "snapshot-id": 3055729675574597004 }
  ]
}
```

成功レスポンス（200, `CommitTableResponse`。`metadata-location` と `metadata` が required）：

```
{
  "metadata-location": "s3://bucket/warehouse/.../metadata/00042-9c12d441-....metadata.json",
  "metadata": { "format-version": 2, "table-uuid": "9c12d441-...", "...": "(TableMetadata 全体)" }
}
```

4.4.5 409 CommitFailedException

> **Conflict - CommitFailedException, one or more requirements failed. The client may retry.**


```
{
  "error": {
    "message": "Requirement failed: branch main has changed: expected id 3051729675574597004 != 3055729675
    "type": "CommitFailedException",
    "code": 409
  }
}
```


409 はリトライ可能と仕様が明示しています。テーブルを再ロード →最新の `snapshot-id` を取得 → requirements を作り直して再コミット、という古典的な CAS リトライループです。

4.4.6 500 CommitStateUnknownException — 最重要の落とし穴

> An unknown server-side problem occurred; **the commit state is unknown.**

409 と 500 の決定的な違い:

	意味	対処
409 **500 CommitStateUnknown**	コミットは**確実に失敗した** コミットが成功したか失敗したか**わからない**	安全にリトライできる **単純にリトライするとスナップショットの二重適用が起こりえます**

これが `Idempotency-Key` ヘッダが存在する理由に直結します(**因果を明示する公式記述は未確認**ですが、`Idempotency-Key` の「safe request retries」および `ServiceUnavailableResponse` の「request could have been partially processed … a **non idempotent request should only be retried by the client when the Retry-After header is present**」という記述と整合します)。

4.5 4. 認証

4.5.1 仕様上の securitySchemes


```
security:
  - OAuth2: [catalog]
  - BearerAuth: []
```

> This scheme is used for OAuth2 authorization. For unauthorized requests, services should return an appropriate 401 or 403 response. **Implementations must not return altered success (200) responses when a request is unauthenticated or unauthorized.**

未認可時に「空の結果を 200 で返す」ような挙動を明確に禁止しています（情報漏洩防止）。

SigV4 は OpenAPI の securitySchemes に含まれていません。仕様レベルではなく**クライアント設定プロパティ**として定義されています。

4.5.2 クライアント認証プロパティ

`docs/docs/catalog-properties.md` より：

> The following catalog properties configure authentication for the REST catalog. They support **Basic, OAuth2, SigV4, and Google** authentication.

Property	Default	説明
`rest.auth.type`	`none`	`none` / `basic` / `oauth2` / `sigv4` / `google`
`rest.auth.basic.username` / `.password`	null	Basic 認証用
`rest.auth.sigv4.delegate-auth-type`	`oauth2`	**SigV4 署名後に委譲する認証方式**

OAuth2

Property	Default	説明
`token`	null	Bearer token。`token` か `credential` のいずれかが必須
`credential`	null	`client_id:client_secret` 形式
`oauth2-server-uri`	`v1/oauth/tokens`	**REST カタログが OAuth2 認証サーバでない場合は必須**
`token-expires-in-ms`	3600000（1 時間）	
`token-refresh-enabled`	true	
`token-exchange-enabled`	true	無効化すると client credential flow にフォールバック
`scope`	`catalog`	

SigV4

Property	Default
`rest.signing-region`	null
`rest.signing-name`	`execute-api`

Property	Default
`rest.access-key-id` / `rest.secret-access-key` / `rest.session-token`	null
`client.credentials-provider`	null

> **重要**: `rest.auth.sigv4.delegate-auth-type`（既定 `oauth2`）が示す通り、**SigV4 は排他的ではなく合成可能**です。AWS Glue の IRC エンドポイントのように「SigV4 で署名した上で、さらに OAuth2 トークンを載せる」構成が想定されています。

4.5.3 `/v1/oauth/tokens` の非推奨

仕様本文（原文）：

> summary: “Get a token using an OAuth2 flow (**DEPRECATED for REMOVAL**)”
>
> The `oauth/tokens` endpoint is **DEPRECATED for REMOVAL**. It is not recommended to implement this endpoint, unless you are fully aware of the potential security implications.
>
> All clients are encouraged to explicitly set the configuration property `oauth2-server-uri` to the correct OAuth endpoint.
>
> **Deprecated since Iceberg (Java) 1.6.0. The endpoint and related types will be removed from this spec in Iceberg (Java) 2.0.**

非推奨の理由（Issue #10537 “Security improvements in the Iceberg REST specification”）：

1. **クレデンシャル漏洩リスク** --- 正しい OAuth エンドポイントを設定していないと、**平文クレデンシャルを誤ったサービスに送ってしまう**
2. **中間者化（MITM）** --- 素朴な実装がトークンリクエストをプロキシしてしまい、OAuth 2.0 仕様に反する中継者になる
3. **認可方式の硬直化** --- OAuth 一択を強制し、SAML など他の方式を選べなくする

推奨される方向性は、標準的な OAuth2/OIDC プロバイダ（Okta, Keycloak, Authelia 等）と直接連携し、**IRC サーバから認可サーバの責務を切り離す**ことです。

バージョン	日付	出来事
1.6.0	2024-07-23	非推奨化（PR #10603）
2.0	未定	仕様から**削除**予定

> **齟齬の注記**: Issue #10537 のマイルストーンは「M1 (v1.7.0): Deprecate」「M5 (v1.9.0 or v2.0): Remove」とありますが、**リリースノートと OpenAPI 本文の双方が 1.6.0 での非推奨を示しています**。Issue は提案段階の計画、YAML/リリースノートは実際に起きたこと、と解釈すべきです。2026 年 7 月時点で Iceberg 2.0 (Java) は未リリースで、エンドポイントは `deprecated: true` のまま残存しています。

4.6 5. Credential Vending / Remote Signing

4.6.1 X-Iceberg-Access-Delegation ヘッダ

```yaml
name: X-Iceberg-Access-Delegation
in: header
description: >
  Optional signal to the server that the client supports delegated access
  via a comma-separated list of access mechanisms.  The server may choose
  to supply access via any or none of the requested mechanisms.
schema:
  type: array
  items: { type: string, enum: [vended-credentials, remote-signing] }
example: "vended-credentials,remote-signing"
```

これはシグナルであって要求ではありません。「The server may choose to supply access via **any or none** of the requested mechanisms」--- サーバは要求されたメカニズムのいずれかを提供することも、**どれも提供しない**ことも選べます。そのためクライアントは、ヘッダを送っても credential が返ってこないケースを常に想定しておく必要があります。

4.6.2 LoadTableResult の規約

> ## **Storage Credentials** Credentials for ADLS / GCS / S3 / … are provided through the `storage-credentials` field. **Clients must first check whether the respective credentials exist in the `storage-credentials` field before checking the `config` for credentials.**

優先順位ルール: `storage-credentials` が新しい正式ルートで、`config` 内の `s3.access-key-id` 等はレガシー互換のフォールバックです。

4.6.3 StorageCredential と longest-prefix-wins

```yaml
StorageCredential:
  required: [prefix, config]
  properties:
    prefix:
      description: Indicates a storage location prefix where the credential is relevant.
        **Clients should choose the most specific prefix (by selecting the longest prefix)
        if several credentials of the same type are available.**
    config:
      additionalProperties: { type: string }
```

longest-prefix-wins ルールが明記されています。1 つのテーブルが複数バケット/コンテナに跨る場合（例: データとメタデータが別ロケーション）に、それぞれ異なるスコープの credential を配ることを想定した設計です。

4.6.4 やりとりの実例

リクエスト:

```
GET /v1/prod/namespaces/accounting%1Ftax/tables/paid HTTP/1.1
Authorization: Bearer eyJhbGciOi...
X-Iceberg-Access-Delegation: vended-credentials,remote-signing
```

レスポンス（vended-credentials 選択時）:

```
{
  "metadata-location": "s3://bucket/warehouse/.../metadata/00042-....metadata.json",
  "metadata": { "format-version": 2, "...": "..." },
  "config": { "client.region": "us-east-1" },
  "storage-credentials": [
    {
      "prefix": "s3://bucket/warehouse/accounting/tax/paid",
      "config": {
        "s3.access-key-id": "ASIAIOSFODNN7EXAMPLE",
        "s3.secret-access-key": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
        "s3.session-token": "IQoJb3JpZ2luX2VjE...",
        "s3.session-token-expires-at-ms": "1785000000000"
      }
    }
  ]
}
```

レスポンス（remote-signing 選択時）:

```
{
  "metadata-location": "s3://bucket/.../metadata/00042-....metadata.json",
  "metadata": { "...": "..." },
  "config": {
    "s3.remote-signing-enabled": "true",
    "signer.endpoint": "v1/prod/namespaces/accounting%1Ftax/tables/paid/sign",
    "signer.uri": "https://catalog.example.com"
  }
}
```

この場合、クライアントは S3 への各リクエストについて `POST .../sign` を呼び、サーバが署名した URL/ヘッダを受け取って S3 にアクセスします。**クライアントは AWS クレデンシャルを一切保持しません。**

4.6.5 なぜセキュリティ上重要か

従来(vending なし)	credential vending / remote signing あり
エンジン(Spark executor 等)に**長命かつ広範な**ストレージクレデンシャルを配布する必要がある カタログでテーブルレベルの ACL を設定しても、**エンジンがストレージに直接アクセスすればバイパス可能** クレデンシャル漏洩でバケット全体が危険	カタログが**テーブル単位・prefix 単位・短命**の credential を都度払い出す ストレージアクセス自体がカタログの認可を経由するため、**バイパス不能** prefix スコープ + 期限付き。remote signing ならクライアントは credential を**そもそも持たない**

仕様が裏付ける設計上の根拠:

- `StorageCredential.prefix` + longest-prefix-wins →**最小権限をロケーション粒度で実現**
- `s3.session-token` の存在 → **短命の STS セッション credential** が前提
- `loadCredentials` の `?planId=` → **scan planning で払い出したプラン専用に credential をスコープできる**
- `loadCredentials` の `?referenced-by=` → **view 経由アクセスの参照チェーンをサーバが把握**でき、「元テーブルへの直接権限を与えずに view 経由でのみアクセス許可」を実装可能

未確認:「カタログ ACL のバイパス防止」という動機付けは仕様設計から論理的に導かれますが、公式がこの言葉で明示的に説明する箇所は見つかりませんでした。

4.6.6 バージョン履歴

バージョン	内容	PR
1.7.0（2024-11-08）	`loadCredentials` エンドポイント追加	#11281
1.7.0	`storage-credentials` フィールドの標準化	#10722
1.9.0（2025-04-28）	AWS/GCP: 複数の storage credential prefix をサポート	#12799, #12881
1.11.0（2026-05-19）	AWS/GCP: held storage credentials の定期リフレッシュ	#15678, #15696

S3 remote signing（`/sign`)はさらに古く 1.3 系から存在し、**vending より先にありました**。

4.7 6. GET /v1/config — ケイパビリティネゴシエーション

4.7.1 適用順序(決定的に重要)

> **All REST clients should first call this route** to get catalog configuration properties from the server…

- **defaults** - properties that should be used as default configuration; applied **before** client configuration
- **overrides** - properties that should be used to override client configuration; applied **after** defaults and client configuration

```
defaults  →  クライアント設定  →  overrides     （後勝ち）
 ↑弱い                              ↑最強
```

- **defaults**: サーバの提案。クライアントが上書きできる。
- **overrides**: サーバの命令。クライアントは上書きできない。→**マルチテナントで運用者がテナントに強制したい値（warehouse、prefix、FileIO 実装クラス等）を置く場所。**

仕様の例:「An override might be used to set the **warehouse location, which is stored on the server rather than in client configuration**.」

4.7.2 endpoints フィールドと後方互換設計

> The catalog configuration also holds an optional `endpoints` field… **If a server does not send the `endpoints` field, a default set of endpoints is assumed**

形式は"<HTTP verb> <resource path>"（空白 1 つで区切る）。

省略時に仮定されるデフォルトセット（14 個）:

```
GET    /v1/{prefix}/namespaces
POST   /v1/{prefix}/namespaces
GET    /v1/{prefix}/namespaces/{namespace}
DELETE /v1/{prefix}/namespaces/{namespace}
POST   /v1/{prefix}/namespaces/{namespace}/properties
GET    /v1/{prefix}/namespaces/{namespace}/tables
POST   /v1/{prefix}/namespaces/{namespace}/tables
GET    /v1/{prefix}/namespaces/{namespace}/tables/{table}
POST   /v1/{prefix}/namespaces/{namespace}/tables/{table}
DELETE /v1/{prefix}/namespaces/{namespace}/tables/{table}
POST   /v1/{prefix}/namespaces/{namespace}/register
POST   /v1/{prefix}/namespaces/{namespace}/tables/{table}/metrics
POST   /v1/{prefix}/tables/rename
POST   /v1/{prefix}/transactions/commit
```

この後方互換設計が肝です。`endpoints` を送らない古いサーバは「view も scan planning も credential vending も**サポートしていない**」と正しく解釈されます。逆に言えば、**view 系エンドポイントを 1 つでも使いたいサーバは `endpoints` を必ず送る必要があります**（デフォルトセットに view が含まれないため）。

4.7.3 レスポンス例（仕様の example そのまま）

```
{
  "overrides": { "warehouse": "s3://bucket/warehouse/" },
  "defaults": { "clients": "4" },
  "idempotency-key-lifetime": "PT30M",
  "endpoints": [
    "GET /v1/{prefix}/namespaces/{namespace}",
    "GET /v1/{prefix}/namespaces",
    "POST /v1/{prefix}/namespaces",
    "GET /v1/{prefix}/namespaces/{namespace}/tables/{table}",
    "GET /v1/{prefix}/namespaces/{namespace}/views/{view}"
  ]
}
```

CatalogConfig の **required は defaults と overrides のみ**。endpoints と idempotency-key-lifetime は optional です。

4.7.4 idempotency-key-lifetime

> **Presence of this field indicates the server supports Idempotency-Key semantics for mutation endpoints. If absent, clients MUST assume idempotency is not supported.**

endpoints と同じ「**フィールドの存在＝ケイパビリティの宣言**」パターンです。

4.7.5 prefix とマルチテナント

仕様上の定義は驚くほど素っ気ないものです（原文全文）：

```
prefix:
  name: prefix
  in: path
  schema: { type: string }
  required: true
  description: An optional prefix in the path
```

確認できた事実:

- prefix は path parameter で required: true（パス上は必ず何かが入る）だが description は “An optional prefix”
- /v1/config **だけ**が prefix を取らない。これが「まず config を叩け」の理由でもある
- /v1/config は ?warehouse= を受け取り、404 は「Warehouse provided in the warehouse query parameter is not found」

典型的なマルチテナントのフロー（仕様の構造から強く支持される推測）：

1. クライアントが GET /v1/config?warehouse=my_warehouse を呼ぶ

2. サーバが `overrides` に `prefix` を入れて返す（例: `{"overrides": {"prefix": "ws/tenant-a/wh-123"}}`）
3. 以降クライアントは全リクエストを`/v1/ws/tenant-a/wh-123/namespaces/...` に送る

   未確認:「prefix はマルチテナント用である」と明言する公式記述は OpenAPI 内に見つかりませんでした。**仕様は意図的に prefix の意味論をサーバ実装に委ねています。**`prefix` に `/` を含められるかも仕様は明示していません。

4.8 7. サーバサイド Scan Planning

4.8.1 結論

問い	回答
仕様に入ったか？	**入っている。1.7.0（2024-11-08）で OpenAPI に追加**（PR #9695、dev ML の [VOTE] を経て正式マージ）
実装はあるか？	**ある。1.11.0（2026-05-19）でクライアント実装が完成**（PR #13400 + Spark 統合 #14963）

仕様追加（1.7.0）から実用実装（1.11.0）まで約 1 年半かかっており、この機能が長く「仕様はあるが Java クライアント実装がない」状態だったことが読み取れます。

バージョン	日付	内容	PR
1.7.0	2024-11-08	OpenAPI: Scan Planning Endpoints 追加（**仕様のみ**）	#9695
1.10.0	2025-09-11	REST: request/response models and parsers	#13004
1.11.0	2026-05-19	**REST Scan Planning Task Implementation**	#13400
1.11.0	2026-05-19	**Spark: Enable remote scan planning**	#14963

4.8.2 なぜ重要か

1.11.0 のリリースブログが位置づけを明確に述べています。

1.11.0 は REST カタログプロトコルが導入されて以来、**最も重要な前進**を表す。**Remote scan planning**（PR #13400）はカタログサーバがテーブルスキャンをプランし、file scan task を直接ストリームバックすることを可能にする。**以前は、全てのクライアントがどのファイルを読むか判断するために manifest list と manifest を自分で取得しなければならなかった。server-side planning により、クライアントは関連する scan task のみを受け取り、driver のメモリ圧迫を軽減し、クエリエンジンに透過的なサーバサイド最適化を可能にする。**

これは 06-operations.md で扱う「メタデータが肥大するとクライアント側 driver が死ぬ」という構造的問題への**プロトコルレベルの解**です。従来は運用（`expire_snapshots` 等）で対処するしかありませんでした。

1.11.0 はさらに以下も追加しています:

- **incremental scans** への拡張（#14661、Structured Streaming 向け）
- **メタデータテーブル**（`history`, `snapshots` 等）への拡張（#14881）
- **per-table override**（#15572）--- 個別テーブルがカタログレベルの mode をオプトアウト可能
- **`scan-planning-mode` を `LoadTableResult` に**（#14867）--- クライアントが probing せずに事前に知れる
- **scan planning レスポンスにストレージ認証情報**（#15524）

4.8.3 ステートマシン

```
POST /plan
    ├── 200 { status: "completed", plan-tasks: [...], file-scan-tasks: [...] }  → 同期完了
    ├── 200 { status: "submitted", plan-id: "..." }                             → 非同期。↓へ
    └── 200 { status: "failed", error: {...} }                                  → 失敗

   GET /plan/{plan-id}   （ポーリング）
       ├── { status: "submitted" }   → 再ポーリング
       ├── { status: "completed", plan-tasks: [...], file-scan-tasks: [...] }
       ├── { status: "cancelled" }
       └── { status: "failed", error: {...} }

   POST /tasks  { "plan-task": "<opaque>" }   → 各 plan-task を file scan tasks に展開
   DELETE /plan/{plan-id}                     → 不要になったら明示的にキャンセル
```

仕様の規定:

- **「A "cancelled" status is considered invalid for this endpoint」**（`POST /plan` のレスポンスとして）
- 「Responses that include a `plan-id` indicate that the service is **holding state or performing work for the client**」
- **「Cancellation is not necessary after fetchScanTasks has been used to fetch scan tasks for each plan task.」**
- 「Requests that include both point in time config properties and incremental config properties are **invalid**」

4.8.4 PlanTableScanRequest の主なフィールド

フィールド	説明
`snapshot-id`	point-in-time scan 用
`select`	選択するスキーマフィールド
`filter`	フィルタ式
`min-rows-requested`	**ヒントであって強制ではない**（「It is not required for the server to return that many rows … The server can also return more rows than requested」）
`case-sensitive`	既定 `true`
`use-snapshot-schema`	既定 `false`。「**When time travelling, the snapshot schema should be used (true). When scanning a branch, the table schema should be used (false).**」
`start-snapshot-id` / `end-snapshot-id`	incremental scan 用（start は**排他的**、end は**包含的**）
`stats-fields`	列統計を返してほしいフィールド

`PlanTask` の定義:「**An opaque string** provided by the REST server that represents a unit of work… This allows clients to fetch tasks across multiple requests to accommodate large result sets.」

`FileScanTask.residual-filter`:「**If the residual is not present, the client must produce the residual or use the original filter.**」

4.8.5 scan-planning-mode — サーバがモードを強制できる

- `scan-planning-mode`: Communicates to clients the supported planning mode. **Clients should use this value to fail fast if the supported scanning mode is not available on the client.** Valid values:
  - `client`: Clients **MUST** use client-side scan planning
  - `server`: Clients **MUST** use server-side scan planning via the `planTableScan` endpoint

これは強い規定です。`server` を返すカタログに対しては、クライアントは manifest を自分で読むことを**許されません。**この規定があってはじめて、管理型カタログがガバナンス（行/列レベルセキュリティ、監査）をスキャンプランニングに埋め込み、**クライアントによるメタデータの直接読み取りを禁止する**ユースケースが成立します。

4.8.6 エコシステムの実装状況

確認済み: Iceberg Java 1.11.0（クライアント実装 + Spark 統合）

未確認 / 二次情報:

- DuckDB (`duckdb-iceberg`) は 2026 年 5 月時点で未対応（Issue #977 がオープン）

- **PyIceberg は 0.11.0 で REST scan planning に対応（同期のみ**、async は将来）→ 05-engine-support.md

---

4.9 8. エラーレスポンス

4.9.1 IcebergErrorResponse — 必ず error でラップされる

全ての非 2xx レスポンスは以下の形になります:

```
{
  "error": {
    "message": "The given namespace does not exist",
    "type": "NoSuchNamespaceException",
    "code": 404
  }
}
```

ErrorModel の required は message / type / code。stack（string 配列）は optional（本番では返さないのが通例）。

> **よくある実装ミス**: ErrorModel を裸で返すこと。仕様上は必ず {"error": {...}} でラップします。

4.9.2 主要ステータスコード

コード	型の例	意味	リトライ
400	BadRequestErrorResponse	不正なリクエスト。**unknown updates/requirements は 400 必須**	不可
401	UnauthorizedResponse	未認証	再認証後
403	NotAuthorizedException	権限不足	不可
404	NoSuchTableException 等	対象が存在しない	不可
406	UnsupportedOperationException	「The server does not support this operation」。planTableScan 等で使用	不可
409	CommitFailedException / AlreadyExistsException	**2 つの意味を持つ（下記）**	文脈による

コード	型の例	意味	リトライ
419	`AuthenticationTimeoutException`	トークン期限切れ。ただし「**401 SHOULD be preferred over this response type on token expiry**」	可
500	`CommitStateUnknownException`	**コミット状態不明**	**危険**
503	`SlowDownException`	「request could have been **partially processed**」「a non idempotent request should only be retried **when the Retry-After header is present**」	条件付き

4.9.3 409 の 2 つの意味に注意

文脈	意味	対処
`updateTable`, `commitTransaction`, `replaceView`	**CommitFailedException** --- 楽観ロック競合	テーブル再ロード → requirements 再構築 → リトライ
`createTable`, `createNamespace`, `renameTable`, `registerTable`	**AlreadyExistsException** --- 識別子が既に存在	**リトライ不可**

4.9.4 429 は存在しない

仕様には 429 Too Many Requests が定義されていません。レート制限は **503 + `Retry-After`**（`SlowDownException`）で表現するのが仕様の意図です。実装によっては 429 を返す可能性がありますが、仕様準拠クライアントは 503 を期待します。

4.10 補足: REST Compatibility Kit (RCK)

`open-api/README.md` より:

> The REST Compatibility Kit (RCK) is a Technology Compatibility Kit (TCK) implementation for the Iceberg REST Specification. It comprises **a series of tests based on the Java reference implementation of the REST Catalog that can be executed against any REST server that implements the spec.**

> The RCK accommodates diverse implementations through configurable test behaviors, acknowledging that **"not all behaviors are strictly defined by the REST Specification."**

1.7.0 で追加されました（PR #10908）。

この「not all behaviors are strictly defined」という Apache 自身の認識は重要で、`prefix` の意味論のように、**仕様が意図的に未定義のまま残している領域が存在する**ことを裏付けています。カタログ実装ごとの挙動差が生じる根本原因はここにあります。

4.11 未確認事項

1. **Hive Metastore の具体的批判**（Thrift/JVM 必須、ロッキング問題）--- Apache の一次情報には見つからず。ベンダーブログ由来。
2. **Hadoop catalog の限界を REST catalog の動機として語る公式記述**--- 未発見。
3. **`prefix` = マルチテナント用という明示** --- 構造から導かれる強い推測に留まる。
4. **`prefix` に `/` を含められるか** --- 仕様が明示せず。
5. **`Idempotency-Key` が `CommitStateUnknownException` 対策として導入された**という因果--- 記述同士は整合するが明示的な因果の記述は未発見。
6. **credential vending の「カタログ ACL バイパス防止」動機** --- 公式がこの言葉で説明する箇所は未発見。
7. **function 系エンドポイントと SQL UDF spec（#14117）の関係** --- 両者を結ぶ明示的記述は未確認。
8. **iceberg-rust の scan planning 対応状況** --- 未確認。

4.12 出典

一次情報（Apache Iceberg 公式）

- OpenAPI 仕様本体: https://github.com/apache/iceberg/blob/main/open-api/rest-catalog-open-api.yaml
- Swagger UI: https://iceberg.apache.org/rest-catalog-spec/
- open-api README (RCK): https://github.com/apache/iceberg/blob/main/open-api/README.md
- 用語集 / REST catalog の背景: https://iceberg.apache.org/terms/
- カタログプロパティ（認証）: https://github.com/apache/iceberg/blob/main/docs/docs/catalog-properties.md
- リリースノート: https://iceberg.apache.org/releases/
- Issue #10537（OAuth セキュリティ改善）: https://github.com/apache/iceberg/issues/10537
- PR #9695（Scan Planning 仕様）/ #13400（実装）/ #14963（Spark 統合）
- PR #11281（loadCredentials）/ #10722（storage-credentials 標準化）
- dev ML [VOTE] Scan Planning APIs: https://www.mail-archive.com/dev@iceberg.apache.org/msg07629

- dev ML [DISCUSS] REST: Scan Planning mode: http://www.mail-archive.com/dev@iceberg.apache.org/msg12158.html

5 04. カタログ実装の比較

調査基準日: 2026-07-17 / 健全性シグナル（コミット数・contributor 数）は GitHub API で実測

03-rest-catalog.md で見た通り、**IRC 仕様にはバージョン番号がなく**（`info.version: 0.0.1` のまま）、かつ Apache 自身が「**not all behaviors are strictly defined by the REST Specification**」と認めています。したがって「仕様準拠度」を単一の数値で語ることはできません。本ドキュメントでは機能単位で比較します。

5.1 比較表

5.1.1 一覧

提供形態とライセンス、credential vending と view の対応です。

実装	提供形態	言語	ライセンス	vending	View
Apache Polaris	OSS（マネージド版あり）	Java	Apache-2.0	○ S3/GCS/Azure	○
Project Nessie	OSS	Java	Apache-2.0	○ signing + assume role	○
Lakekeeper	OSS（商用版あり）	**Rust**	Apache-2.0	○**最も充実**（S3 signing+STS, ADLS SAS, GCS）	○
Apache Gravitino	OSS	Java	Apache-2.0	○ S3/GCS/ADLS/OSS	○（1.3.0 で Logical view）
Unity Catalog OSS	OSS	Java/Scala	Apache-2.0	---	×
Databricks Unity Catalog	マネージド	---	商用	○（Foreign は不可）	○（MV は preview）
AWS Glue Data Catalog	マネージド	---	商用	○ Lake Formation 経由	×
Amazon S3 Tables	マネージド	---	商用	※ ネイティブ endpoint では**文書化なし**	×

実装	提供形態	言語	ライセンス	vending	View
Cloudflare R2 Data Catalog	マネージド	---	商用	○（token 権限を継承）	不明
Snowflake Open Catalog	マネージド	---	商用	○	○
Hive Metastore	OSS	Java	Apache-2.0	×	限定
Hadoop catalog	OSS	Java	Apache-2.0	×	---
JDBC catalog	OSS	Java	Apache-2.0	×	○
Apache XTable	OSS	Java	Apache-2.0	---	---

5.1.2 仕様準拠・認証・権限管理

実装	REST 準拠	認証	RBAC / マルチテナント
Apache Polaris	高（IRC の本命）	OAuth2 client_credentials, OIDC	○ catalog/principal role の 2 層
Project Nessie	**experimental**、branch は独自 URI 記法	OIDC/Keycloak, OAuth2	CEL 式ルール
Lakekeeper	高（ただし準拠バージョンの公式明記なし）	OIDC/OAuth2, K8s SA。**Kerberos なし**	○ OpenFGA、project → warehouse
Apache Gravitino	中（**マルチテーブル tx・view 登録は非対応と明記**）	Simple/Basic/OAuth2/**Kerberos**	○ Ranger 連携、metalake 階層
Unity Catalog OSS	**読み取り専用・UniForm 経由のみ**	OAuth2/OIDC	3 層権限
Databricks Unity Catalog	Managed Iceberg のみ**読み書き可**、Foreign/UniForm は読取のみ	OAuth2/PAT	UC
AWS Glue Data Catalog	中（**view API・RenameTable なし**、単一階層 namespace、独自 prefix `/catalogs/{c}`）	**SigV4**	○ **最も強力**（LF 行/列レベル）
Amazon S3 Tables	中（**CTAS 不可**＝stage-create なし、**view 非対応**、purge=false 不可）	SigV4（**OAuth 不可**）	IAM/リソースポリシー

実装	REST 準拠	認証	RBAC / マルチテナント
Cloudflare R2 Data Catalog	要検証	**Bearer API token**	token スコープのみ(粗い)
Snowflake Open Catalog	Polaris 準拠	Polaris 準拠	Polaris RBAC + Horizon
Hive Metastore	REST でない(Thrift)	Kerberos	弱
Hadoop catalog	REST でない	---	×
JDBC catalog	REST でない	DB 認証	×
Apache XTable	**カタログではない**（メタデータ変換器）	---	---

5.1.3 成熟度と現況

実装	成熟度・現況
Apache Polaris	**ASF TLP（2026-02）**、contributor 156、年間 2,100 commits（**うち 35%はボット、人間 1,356**）
Project Nessie	**実質 1 人保守**（年間の人間コミット 245 件、うち 212 が 1 人。全体の 85%はボット）。Dremio 主導だが戦略的地位は低下
Lakekeeper	**v0.13.1 / pre-1.0**、star 1.4k、**実質コア 2 名、公表採用事例なし**
Apache Gravitino	**TLP（2025-06）**、v1.3.0、star 3.1k、**Uber/Pinterest 採用**
Unity Catalog OSS	LF AI&Data **sandbox**、v0.5.0、API 不安定と明記
Databricks Unity Catalog	**GA**（Iceberg v3 も GA）
AWS Glue Data Catalog	GA
Amazon S3 Tables	GA（REST は 2025-03〜）、**自動 compaction 標準 ON**
Cloudflare R2 Data Catalog	**open beta（GA 未達）**
Snowflake Open Catalog	GA。**ただし新規顧客に実質クローズ**（下記）
Hive Metastore	枯れているが漸減。**公式な非推奨宣言は存在しない**
Hadoop catalog	**S3 で危険**。2.0 での deprecate 提案あり
JDBC catalog	安定。**Hadoop catalog の推奨代替**
Apache XTable	**incubating のまま**。0.3.0-incubating (2025-06)から **13 ヶ月リリースなし**

5.2 主要実装の詳細

5.2.1 Apache Polaris — TLP 卒業済み（最大の変化）

2026 年 2 月に Apache Incubator を卒業し Top-Level Project 化しました。（正確な卒業日は公式ソース間で食い違っており、本報告書は単一の日付には断定しません。下記参照。）

リリース履歴が卒業タイミングと一致しています（`-incubating` サフィックスが 1.3.0 で終わり、1.4.0 以降が TLP リリース）：

バージョン	日付
1.6.0	2026-07-09
1.5.0	2026-05-18
1.4.1 / 1.4.0	2026-05-01 / 2026-04-21
1.3.0-incubating	2026-01-16
1.0.0-incubating	2025-07-11

健全性の実測: star 2,014 / fork 485 / **全期間 contributor 156** / 直近 365 日 **2,100 commits（うち 744 = 35.4% は renovate ボット。人間は 1,356）** / 直近 365 日のユニーク作者 106 / 最終 push 2026-07-16。

ボットを除いても人間 1,356 コミット・106 人が関与しており、**開発は実際に活発**です（Nessie とは対照的）。

Generic Table: 非 Iceberg テーブル（Delta, CSV 等）を扱う **beta** 機能。**Iceberg REST とは別エンドポイント**`/polaris/v1/{prefix}/namespaces/{ns}/generic-tables`。制約が多く、Schema/Partition spec を持たない・コミット調整をしない・**更新不可（drop & recreate）・credential vending 非対応**。

> **要検証**: 1.5 で `Polaris-Generic-Table-Access-Delegation` ヘッダによる vending 拡張が入ったとの Snowflake の記述がありますが、公式 docs の「vending 非対応」という制約記述と食い違います。

卒業日は公式ソース 3 つが 3 つとも食い違っています（実測）：

ソース	日付
Polaris 公式ブログ	**2026-02-19**
Incubator ステータスページ	**2026-02-15**
Incubator プロジェクト一覧	**2026-02-18**

後者 2 つは**同じ incubator.apache.org 内での自己矛盾**です。02-18 が ASF 理事会の議決日、02-19 がプロジェクト側の告知日である可能性が高いと思われますが、**board minutes を確認していないため断定できません**。本報告書は日付を 1 つに断定せず、食い違いの存在を事実として記します。

なおステータスページは「2026-02-15 Graduation as TLP.」と**確定形で記載**しており、projected（予定）の但し書きは**ありません**（ページ全文を検索して 0 件）。

5.2.2 Project Nessie — 「Polaris に統合されて廃止」は実現していない

ここが最も誤解されやすい点です。

2024 年の報道（BigDATAwire、Dremio newsroom）は Dremio CMO の「We will treat Polaris as our catalog and we will merge Nessie into Polaris」を引用し、**Nessie は retire される**と伝えました。しかし 2026 年 7 月の実測は逆です:

- 最新リリース **0.108.2（2026-07-17）**、直近も継続パッチ
- **archived: false**、最終 push は測定日当日
- Nessie 公式サイトに Polaris への統合・廃止の記載は**一切なし**

> **ただし「活発」という評価語は使えません。**直近 365 日のコミット 1,652 のうち **85.2%（1,407 件）が renovate ボット**で、**人間のコミットは 245 件**、しかも実質 1 人に集中しています（snazy が 212、残り全員で 33）。ユニーク作者は **22 人**（Iceberg 270 の 1/12）。**リリース頻度が高いのは、依存更新を自動リリースしているためです。**→ 07-ecosystem.md

Nessie 自身の公式アナウンスの表現は retire ではなく「Nessie の機能（カタログレベル versioning、Git-like セマンティクス、マルチテーブルトランザクション）を Polaris に**コントリビュートする意図**」であり、両者は併存前提です。

結論:「統合の噂」は Dremio 側の発言に基づく報道であり、**2026 年 7 月時点で Nessie は独立プロジェクトとして存続しています**（archived ではなく、リリースも継続）。

しかし「活発」とは言えません。人間のコミットは年間 245 件・実質 1 人であり、**bus factor は 1 に近い**。「retire された」は誤りですが、「健在」と言うのも過大評価です。**正確には「形式的には存続、実質的には少数保守モード」**です。Dremio の戦略的重心が Polaris にある以上、長期的な先細りリスクは現実的です。

Iceberg REST 対応（guides/iceberg-rest）: 実装済みですが**公式に「experimental」と明記**されています。エンドポイントは `/iceberg`、ブランチは URI に `{branch}|{warehouse}` 形式で埋め込みます（例 `http://127.0.0.1:19120/iceberg/experiments|sales`）--- **これは REST 仕様の標準的な prefix 用法ではない独自拡張**です。S3 request signing と credential vending（assume role）は両方対応。制約: テーブルの base location は Nessie が強制し、変更は無視されます。

5.2.3 Lakekeeper

Rust 製で JVM 不要、credential vending が最も充実しています。技術的な出来は良いのですが、**本番マルチテナントには時期尚早**と判断します:

- **pre-1.0（v0.13.1、2026-06-30）**
- **コミット実績が実質 2 名に集中**（bus factor ≈ 2）
- **公式に名前の出た本番採用事例がゼロ**
- **REST 仕様の準拠バージョンとエンドポイント単位の適合表が、公式 docs にもリポジトリにも存在しない**

5.2.4 Apache Gravitino

TLP 卒業(2025-06-03)、v1.3.0、star 3.1k、**Uber / Pinterest の採用実績**あり。**Kerberos 対応**が特徴で、既存 Hadoop 資産との統合が必要な場合の選択肢になります。

ただし Iceberg REST service の docs が**マルチテーブルトランザクションと view 登録を非対応と明記**しています。

5.2.5 Hadoop catalog は本番使用不可

Javadoc が「atomic rename を要求」「renameTable は未サポート」と明記し、dev@ML で Ryan Blue が「**HDFS 以外(ローカル FS、S3、一般的なオブジェクトストア)では安全でない**」として 2.0 での deprecate を提案、概ね合意済みです。

これは 01-architecture.md で見た仕様側の記述(File System Tables 方式は deprecated、オブジェクトストアでは unsafe、v4 で削除予定)と一致します。**ローカルの単一ライタ検証以外では使わないでください。**推奨代替は **JDBC catalog** です。

5.2.6 Apache XTable はカタログの代替ではない

Hudi/Delta/Iceberg 間の**メタデータ変換層**です。CoW/read-optimized のみ対応(Hudi log file・Delta deletion vector は未反映)。

実測で明確に停滞しています:

- 最新リリース **0.3.0-incubating = 2025-06-04**(**13 ヶ月以上リリースなし**)
- コミット: 直近 90 日 **10 件** / 365 日 **42 件**(うち 6 件はボット。人間は 36 件。Iceberg は人間 1,387 件)
- Incubation 入りは 2024-02-11、**2026 年 7 月時点でまだ incubating**(約 2 年 5 ヶ月)

ベンダー中立の相互運用を担う唯一のプロジェクトがこの状態である点は、ロックインの観点でリスクとして認識すべきです→ 07-ecosystem.md

5.2.7 Snowflake Open Catalog は新規顧客に実質クローズ

公式 docs が明言しています(逐語):

> Customers who **haven't previously created a Snowflake Open Catalog account can't sign up for their first Open Catalog account**.
>
> **New customers should use Snowflake Horizon Catalog** for Apache Iceberg™ tables and multi-engine interoperability with Iceberg.
>
> **Existing** Snowflake Open Catalog customers who already have at least one Open Catalog account **can continue using Open Catalog and can create additional Open Catalog accounts** if necessary.

正式な deprecation ではありません(docs に deprecation/EOL の語はなく、既存顧客の継続利用と追加アカウント作成を明示的に許可)。しかし新規サインアップは**できません**。

決着つかず: 課金について docs は「Open Catalog is currently free to use with general availability. **Billing will begin in the first half of 2026**」と将来形のままです。現在は 2026 年 7 月で上半期は既に経過しており、**docs の更新漏れか課金開始の遅延か判別できません**。

5.2.8 AWS: Glue と S3 Tables は別エンドポイント

AWS は使い分けを案内しています。

- **細粒度ガバナンスが要る** → Glue endpoint（`glue.<region>.amazonaws.com/iceberg`）+ Lake Formation
- **単一バケットの素の読み書き** → S3 Tables ネイティブ（`s3tables.<region>.amazonaws.com/iceberg`）

要検証: S3 Tables ネイティブ側の credential vending は AWS 公式ドキュメントに記述がなく、re:Post に失敗報告もあります。採用前に必ず PoC で検証してください。

S3 Tables の自動メンテナンスには重大な落とし穴があります → 06-operations.md

5.3 ローカル構築のしやすさ（実際の compose ファイルを取得して評価）

結論: Apache Polaris が最有力。次点で Lakekeeper。

5.3.1 1 位: Apache Polaris — 4 サービス・自動ブートストラップ・外部 DB 不要

公式 quickstart compose の実内容:

サービス	イメージ	役割
`polaris`	`apache/polaris:latest`	本体（8181=API, 8182=管理）。**永続化は in-memory 既定 → Postgres 不要**
`rustfs`	`rustfs/rustfs:1.0.0-alpha.81`	S3 互換ストア（9000/9001）
`bucket-setup`	`amazon/aws-cli:2.35.21`	バケット作成の使い捨て
`polaris-setup`	`alpine/curl:8.21.0`	catalog/principal/role/grant を curl で自動作成

```
curl -s https://raw.githubusercontent.com/apache/polaris/refs/heads/main/site/content/guides/quickstart/doc
  | docker compose -p polaris-quickstart -f - up -d
docker compose -p polaris-quickstart logs | grep -i client   # 資格情報を拾う
```

`docker compose up` だけで catalog・principal・role・権限まで自動生成し、最後に **Spark の接続コマンドと発行済み認証情報を丸ごと標準出力に表示**します。公式の注記:「**This setup is not intended for production use.**」

決定的な強みは site/content/guides/ 配下に用途別 compose が完備されていることです(実測で存在確認):

```
.it-setup, assets, ceph, cockroachdb, flink, jdbc, keycloak,
minio, ozone, quickstart, rustfs, spark, telemetry, trino
```

Keycloak を足して OIDC、jdbc で Postgres 永続化、Trino/Flink でマルチエンジン、と段階的にラボを拡張できます。

> **注意点**: - guides/quickstart は rustfs:1.0.0-alpha.81、guides/rustfs は rustfs:1.0.0-beta.8 と**別バージョンを使っています**。- apache/polaris:latest はタグ固定されておらず、1.6.0 相当かは**未確認**。再現性重視なら固定してください。- getting-started/eclipselink/ は**撤去済み(404)**。永続化ガイドは jdbc(PostgreSQL)に置換されました。古い記事の手順は動きません。

認証の要点(公式 assets/polaris/obtain-token.sh より):

```
TOKEN=$(curl --fail-with-body -s \
  http://polaris:8181/api/catalog/v1/oauth/tokens \
  --user ${CLIENT_ID}:${CLIENT_SECRET} \
  -H "Polaris-Realm: $realm" \
  -d grant_type=client_credentials \
  -d scope=PRINCIPAL_ROLE:ALL | jq -r .access_token)
```

- OAuth2 **client_credentials**、scope=PRINCIPAL_ROLE:ALL、realm は Polaris-Realm ヘッダ
- ブートストラップ: POLARIS_BOOTSTRAP_CREDENTIALS: POLARIS, ${ROOT_CLIENT_ID:-root}, ${ROOT_CLIENT_SECRET:-s3cr3t}(**書式は realm,clientId,clientSecret**)
- ※ **/v1/oauth/tokens は削除予定**(03-rest-catalog.md 参照)。Polaris も「already deprecated for removal due to security concerns」とし、外部 IdP を構成した realm では **HTTP 501** を返します。

5.3.2 2 位: Lakekeeper — Rust で JVM 不要、Trino/StarRocks 同梱

examples/minimal/docker-compose.yaml: quay.io/lakekeeper/catalog + Postgres 17 + SeaweedFS + Jupyter/PySpark + Trino 476 + StarRocks。依存は **Postgres + S3 ストアのみ**。Polaris より 1 歩重い(Postgres 必須)ですが、Trino/StarRocks が最初から入っているのは魅力です。

> compose 既定は latest-main タグ = **開発ブランチ追従**で、リリース版 v0.13.1 とは別物です。

5.3.3 3 位: Project Nessie — 最小構成は圧倒的に軽い

docker/in_memory/docker-compose.yml は **1 サービスのみ**。Git-like ブランチを試すだけなら最速です。ただし REST は experimental。

5.3.4 apache/iceberg-rest-fixture

タグ（Docker Hub tags API 実測）：latest(2026-04-29), 1.10.1(2025-12-22), 1.10.0, 1.9.1, 1.9.0, 1.8.1

> latest(2026-04-29) がバージョン付き最新 1.10.1(2025-12-22) より**新しい** ＝中身が一致しない可能性。タグ固定必須。

「本番非推奨」の正確な扱い --- ここは訂正が必要です。しばしば「公式が本番使用を禁止している」と言われますが、**README を直接確認した限り、そのような明示的な免責文は存在しません（未確認）**。よく引用される「integration tests will delete data…」は **RCK（統合テスト実行）側の警告**であり、fixture イメージへの本番禁止表明とは別物です。

ただし**テスト用途であることを示す状態証拠は強固**です:

- README タイトル「Iceberg REST Catalog Adapter **Test Fixture**」
- ソース配置が open-api/src/**testFixtures**/java/...
- **既定がインメモリ SQLite JdbcCatalog**（再起動で全メタデータ消失）
- イメージ名自体が "fixture"

正確な言い方は「公式が本番禁止と明記している」ではなく「**命名・ソース配置・既定値のすべてがテスト用途を示す**」です。

環境変数の変換規則（README + RCKUtils.java の実装で二重検証）:

CATALOG_ 除去 → "__" を "-" へ → 残る "_" を "." へ → 小文字化

順序が本質的です（__→- を先に処理するため CATALOG_IO__IMPL → io-impl。逆順なら io.impl になり壊れます）。

環境変数	プロパティ
CATALOG_WAREHOUSE	warehouse
CATALOG_IO__IMPL	io-impl
CATALOG_S3_ENDPOINT	s3.endpoint
CATALOG_S3_PATH__STYLE__ACCESS	s3.path-style-access

5.3.5 その他

- **Unity Catalog OSS**: 2 サービスで起動は最も手軽ですが、**Iceberg REST が読み取り専用**なのでラボの題材には不適。
- **Apache Gravitino**: compose は別リポジトリ gravitino-playground。約 11 サービス、**2 CPU / 8GB RAM / 25GB disk**、起動 3〜5 分。重い。
- **クラウド系（Glue / S3 Tables / R2）**: R2 と Glue の/iceberg エンドポイントにはエミュレータが見つかりません。S3 Tables は LocalStack が対応しますが **Ultimate ティア（有償）**。

5.4 選定指針

5.4.1 シナリオ A: セルフホストで REST カタログを試したい → Apache Polaris

quickstart が 4 サービス・外部 DB 不要・完全自動ブートストラップで、接続情報まで出力してくれます。TLP 卒業済みで仕様のリファレンス実装的な位置づけであり、**学んだことがそのまま本番知識になります**。用途別 compose ガイドが揃い、段階的にラボを育てられます。

推奨の進め方: Polaris quickstart → `guides/jdbc` で永続化 → `guides/keycloak` で OIDC → `guides/trino` でマルチエンジン。

対抗: Rust 単一バイナリの軽さと vending の充実を重視するなら Lakekeeper。Git-like ブランチが主目的なら Nessie の in_memory 1 サービス。

> 本報告書の検証環境もこの構成を採用しています。

5.4.2 シナリオ B: 本番マルチテナント → Apache Polaris

TLP ガバナンス、contributor 156 名、catalog role / principal role の 2 層 RBAC、realm によるマルチテナント分離、Postgres/CockroachDB バックエンド、Helm chart 提供。**ベンダー中立性が最重要ならここ。**

- **Kerberos や既存 Hadoop 資産との統合が必須** → **Gravitino**（Kerberos 対応 + Uber/Pinterest の本番実績 + ASF TLP）。ただし IRC がマルチテーブル tx・view 登録を非対応と明記している点を許容できるか確認を。
- **行・列レベルのガバナンスが必須で AWS 中心** → **Glue + Lake Formation**。細粒度権限はこの中で最強。
- **Lakekeeper は時期尚早**（pre-1.0、bus factor ≈ 2、公表採用事例ゼロ）。リスクを取れるなら選択肢。

5.4.3 シナリオ C: クラウドマネージドで済ませたい

状況	推奨	注意
AWS 完結・運用ゼロ最優先	**S3 Tables**	自動 compaction/snapshot 管理が標準 ON なのが最大の価値。ただし **CTAS 不可・view 非対応** を設計前に織り込むこと。さらに自動メンテナンスの fail-closed 問題を必ず理解すること
複数エンジン・複数アカウントの統合ガバナンス	**Glue Iceberg REST + Lake Formation**	2025-11 のカタログフェデレーションで Polaris/Unity も Glue 配下に束ねられる
OSS Polaris と同じ API のまま管理サービス化	**Snowflake Horizon Catalog**	**Open Catalog は新規顧客に実質クローズ**。新規は Horizon へ誘導される

状況	推奨	注意
Databricks 中心	**UC 管理版**	Managed Iceberg なら IRC 経由で**読み書き両方 GA**、v3 も GA。ただし **Foreign Iceberg / UniForm 経由は読み取り専用かつ vending 不可**という非対称性に注意
データ持ち出し料金（エグレス）が支配的な多クラウド	**R2 Data Catalog**（エグレス無料）	**依然 open beta で GA 未達**、RBAC は token スコープのみ。本番採用は慎重に

5.5 未確認事項

1. **Lakekeeper の REST 仕様準拠バージョンとエンドポイント単位の適合表** --- 公式 docs にもリポジトリにも記載なし。「マルチテーブルコミット含む完全準拠」はブログ由来で未確認。
2. **Lakekeeper の本番採用事例** --- 公式情報源にゼロ。
3. **Polaris 1.5 の Generic Table vending 拡張** --- Snowflake ブログの記述と公式 docs の「vending 非対応」が食い違う。
4. **S3 Tables ネイティブエンドポイントの vending 有無** --- AWS 公式に記述なし、コミュニティでは失敗報告あり。
5. **Glue REST の view / RenameTable 対応** --- サポート一覧に無いが、明示的な非対応宣言でもない。
6. **Glue の vending ヘッダ機構**（`X-Iceberg-Access-Delegation`）--- AWS 公式ドキュメント上に記述なし。
7. **R2 Data Catalog の view 対応と GA 時期** --- 情報なし。
8. **Gravitino の remote signing 対応**（vended credentials のみ文書化）と詳細 RBAC モデル。
9. **Hive Metastore の非推奨化** --- **Iceberg 公式に非推奨宣言は存在しません。**「REST が新規推奨」もセカンダリブログ由来で、Apache Iceberg 公式はカタログの推奨ランキングを公開していません。
10. **Nessie の長期的な将来** --- 2024 年の Dremio 発言（retire 予定）と 2026 年の実態（活発に開発中）が矛盾しており、公式の最新見解が見当たりません。
11. **Polaris の TLP 卒業日** --- Incubator ページ 2026-02-15 vs 公式ブログ 2026-02-19。
12. **`apache/polaris:latest` の実体バージョン**、および `iceberg-rest-fixture` の `latest` と `1.10.1` の中身の異同。
13. **Nessie の公式 docker compose 例**（`/guides/docker/` は `docker run` のみ）。

5.6 出典

- Polaris TLP 卒業ブログ（2026-02-19）· ASF news · Incubator status

- Polaris quickstart compose · Generic Table · obtain-token.sh
- Nessie Iceberg REST guide · Nessie/Polaris 公式見解 · BigDATAwire（retire 報道）
- Lakekeeper compose · Lakekeeper storage docs
- Gravitino TLP · Gravitino Iceberg REST · gravitino-playground
- UC UniForm · Databricks Iceberg 外部アクセス
- Glue Iceberg REST · Glue REST API 仕様
- S3 Tables OSS 連携
- R2 Data Catalog · Snowflake Open Catalog overview
- XTable incubator status
- HadoopCatalog javadoc · dev@ deprecation スレッド
- iceberg-rest-fixture tags API · RCKUtils.java

6 05. エンジン対応状況

調査基準日: 2026-07-17 / Iceberg 1.11.0（2026-05-19）基準

6.1 読み方（重要）

対応状況の判定は **3 値**で区別しています。「非対応」と「未確認」を混ぜると、実際には使える機能を諦めたり、逆に無い機能を当てにしたりするためです。

記号	意味
【確認】	公式ソースで対応と確認
【明示的非対応】	**公式ソースが非対応と明記**
【未確認】	**ソースが見つからなかった。「非対応」ではない**

さらに「**マージ済み ≠ リリース済み ≠サポート対象**」を厳密に区別します。この罠は Trino・Presto・Spark で繰り返し現れました。

6.2 サマリ表

エンジン	読み	書き	REST catalog	format v3
Spark	GA	GA（MERGE/UPDATE/DELETE、パーティション/スキーマ進化）	クライアント GA	最も進んでいるが **サブ機能ごとに差**（下記）
Flink	GA	GA だが **INSERT/UPSERT のみ。MERGE/UPDATE/DELETE の SQL なし**	クライアント GA	部分的
Trino 482	GA（v1/v2）	GA（MERGE あり）	クライアント GA	**experimental**（docs 明記）

エンジン	読み	書き	REST catalog	format v3
Presto 0.298.1	GA（v1/v2）	GA だが **MERGE 非対応**（Java）、**C++ は INSERT のみ**	クライアント GA	部分的、大半がトランク上で未リリース
PyIceberg 0.11.1	GA	GA だが**制約が重い**（下記）	クライアント GA	**読めるが書けない**
Snowflake	GA	GA	**サーバ + クライアント**	**GA**（2026-05-07）
Databricks	GA	GA（**managed のみ**）	**サーバ+ クライアント**	**GA**（2026-05-28）
Dremio	GA	GA（フル DML）	**サーバ + クライアント**（Polaris ベース）	**Preview**
StarRocks 4.0.1	GA	**INSERT 系のみ。DELETE/UPDATE/MERGE なし**	クライアント	**なし**（tracker #63819 未アサイン）
Doris 4.1.1	GA（**v3 読みは 4.1.0〜**）	INSERT/CTAS は GA、**UPDATE/DELETE/MERGE は experimental**	クライアント（3.1.0〜）	部分的/preview
Impala 4.5.0	GA（v1/v2）	INSERT/DELETE/UPDATE/MERGE（**position delete のみ、Parquet のみ、MoR のみ**）	**未リリース**（5.0 目標）	**なし**
Hive 4.2.0	GA	GA（フル DML、Tez 必須）	**クライアント + サーバ**（HMS REST Catalog、4.2.0）	部分的/preview（**row lineage は既知のギャップ**）
DuckDB	GA	1.4〜（CREATE/INSERT/COPY）	クライアント	未確認
ClickHouse	GA（catalog 経路は experimental）	**experimental・フラグ制**（INSERT/CREATE のみ）	クライアント（experimental）	未確認
BigQuery	GA	GA（managed）／**BigLake external は読み取り専用**	BigLake Metastore（IRC）**GA**	未確認
Athena	GA	GA（MERGE、OPTIMIZE、VACUUM）	Glue 経由	DV + row lineage（2025-11 発表）

エンジン	読み	書き	REST catalog	format v3
Redshift	GA（Spectrum/Glue/S3 Tables）	**未確認**（AWS docs に肯定も否定もなし）	Glue/S3 Tables 経由	未確認

6.3 ベンダー 3 社の v3 対応 — 食い違いを一次情報で決着

先行の調査結果の間で主張が食い違ったため、改めて一次情報で検証しました。**結果として、食い違いの大半は「時系列の異なる時点を切り取った」ことによるもの**でした。

6.3.1 Databricks —【確定】GA

両方の主張が正しく、見ていた時点が違っただけでした。

日付	出来事
2026-04-09	Public Preview 発表 ---「Today, Databricks's support of Iceberg v3 enters **Public Preview**」
2026-05-28	**GA 発表** ---「**Iceberg v3 (GA)**: Native support for deletion vectors, row tracking, and the new VARIANT type across managed, foreign, and UniForm-enabled tables.」
2026-06-17	docs 最終更新。**Preview/Beta バナーなし**（Databricks は Preview 機能に明示ラベルを付ける慣習があり、その不在は GA と整合）

サブ機能:

- 対応: deletion vectors / VARIANT / row lineage
- **【明示的非対応】**:「Defaults, including write defaults and initial defaults, aren't supported」、**unknown 型**、**ナノ秒精度 timestamp**、**multi-argument transforms**

書き込みの非対称性（重要）:

- **Managed Iceberg のみ書き込み可**（外部エンジン Spark/Flink/Trino/Kafka からも可）
- **Foreign Iceberg は Databricks 内で読み取り専用**、**credential vending 非対応**、手動 `REFRESH FOREIGN TABLE` が必要
- **パーティション進化は外部エンジン経由でのみ GA。Databricks SQL 経由では非対応**
- **v2 の position/equality delete は非対応**（v3 の DV は対応）--- 注目すべき相互運用ギャップ

未解決の食い違い: foreign Iceberg の credential vending は 2026-05-28 のブログが GA と述べる一方、REST API docs ページは非対応としています。docs の遅れの可能性がありますが未解決です。

6.3.2 Snowflake —【確定】GA、外部エンジンからの v3 書き込みも対応済み

リリースノートと現行 docs の「矛盾」は、矛盾ではなく時系列でした。

日付	出来事
2026-02-06	外部エンジンからの **read** が GA
2026-03-04	Iceberg v3 サポート **Preview**
2026-03-16	外部エンジンからの **write** が Preview
2026-05-07	**v3 が GA** ---「Writes from external engines to Snowflake-managed Iceberg v3 tables through Horizon Catalog **aren't supported yet**」←**この時点では write 自体がまだ Preview だったので正しい記述**
2026-05-26	**外部エンジン write が GA** ---「read and write to Snowflake-managed Iceberg **v2 and v3** tables」← **ここで v3 制約が解消**

現行 docs（最終根拠）：

> You can **read and write** to Snowflake-managed Iceberg **v2 and v3** tables from external engines through the Horizon Iceberg REST Catalog API
>
> **教訓**: **Snowflake のリリースノートを現況の根拠に使わないこと。**仕様上、後から更新されることはなく、その時点の記述がそのまま残り続けます。→ 99-methodology.md

IRC: サーバ（Horizon Catalog、`https://<account>.snowflakecomputing.com/polaris/api/catalog`）とクライアントの**両方向 GA**。ただしサーバ側が公開するのは **Snowflake-managed Iceberg テーブルのみ**です。

【明示的非対応】: nested VARIANT、multi-arg transforms、table encryption keys、UNKNOWN 型、**in-place の v2 → v3 アップグレード**、externally-managed v3 での append-only stream

新規 managed テーブルの既定は依然 v2（v3 はオプトイン）。

6.3.3 Dremio —【確定】Preview

ここだけ本当の誤りがありました。プレスリリースの表現を採った調査結果が、docs と突き合わせられていませんでした。

ソース	主張
プレスリリース（2026-04-06）	副題で「ships the latest V3 spec **at GA**」、本文「V3 support is **now available in Dremio Cloud**」「full read and write support」
docs（現時点）	見出しに **"Iceberg v3 Preview"** と明記。「Although **v2 remains the default format version in Dremio**…」

docs 優先ルールにより Preview を採用します。

Cloud と Software の差: Software（current/26.x）側の docs には **v3 の記述が一切ありません**。プレスリリースも Cloud に限定。→ **v3 は Cloud のみ（Preview）、Software は未提供/未文書化**。

IRC: 両方向 GA。Dremio 自身のカタログは **Apache Polaris ベース**で、`{DREMIO_ADDRESS}:8181/api/catalog` に IRC エンドポイントを公開します。

6.4 Apache Spark

6.4.1 サポート対象バージョン（Iceberg 1.11.0）

Spark	状態	初回サポート	最終サポート
3.4	**End of Life**	1.3.0	**1.11.0（最後）**
3.5	Maintained	1.4.0	1.11.0
4.0	Maintained	1.10.0	1.11.0
4.1	Maintained	**1.11.0**	1.11.0

6.4.2 v3 サブ機能ごとの対応

サブ機能	判定	根拠
DV 書き込み	**【確認】** 1.8.0〜	PR #11561 “Spark: Write DVs for V3 MoR tables”（2024-12-06 merged）
DV 読み取り	**【確認】** 1.8.0〜	Core #11481。Spark 側は #11657、1.9.0 で「Rewrite V2 deletes to V3 DVs」(#12250)
DV の v3 デフォルト挙動	**【確認】**	**`write.delete.mode` の既定は `copy-on-write`。v3 でも DV は自動では使われず、MoR を明示選択した場合のみ**
row lineage 読み取り	**【確認】** 1.9.0〜	#12836（`_row_id` / `_last_updated_sequence_number` readers）
row lineage 生成（通常書き込み）	**【確認】** 1.10.0〜	#13310 “Spark 4.0: Row Lineage support”。**Spark 3.5 と 4.0/4.1 で実装方式が異なる**（3.5 は Scala rule `RewriteOperationForRowLineage.scala`、4.0/4.1 は `ExtractRowLineage.java`）

サブ機能	判定	根拠
row lineage の compaction 保存	**【確認】** 1.10.0〜	#13555。ただし PR タイトルが **“Spark 4.0” 限定。3.5 での保存は未確認**
VARIANT 読み取り	**【確認】** 1.10.0〜	#13219。型表は `variant | variant | Spark 4.0+`
VARIANT shredding 書き込み	**【確認】** 1.11.0〜、**Spark 4.1 のみ**	#14297。変更ファイルは **`spark/v4.1/` のみ**。型表の「Spark 4.0+」とは食い違うので注意
default values 読み取り適用	**【確認】** 1.8.0〜	#11803
default values 書き込み/DDL 設定	**【明示的非対応】**	下記
geometry / geography	**【明示的非対応】(1.11.0) / 1.12.0 でマージ済み・未リリース**	下記
nanosecond timestamps	**【明示的非対応】**	型表が `Not supported`。コードも `TIMESTAMP_NANO` の case が無く `UnsupportedOperationException` に落ちる
multi-argument transforms	**【明示的非対応】**	`Spark3Util.java: checkArgument("zorder".equals(transform.na || transform.references().length == 1, "Cannot convert transform with more than one column reference: %s")` --- **zorder 以外の複数カラム参照を明示的に拒否**

6.4.3 罠 1: default values は「サポート追加」に見えて書けない

1.11.0 のリリースノートに「Add schema conversion support for default values (#14407)」とあります。しかし:

- 実タイトルは **“Spark 4.0: Add schema conversion support for default values”**、変更は `TypeToSparkType.java` のみ = **スキーマ変換だけ**
- 1.11.0 タグの `Spark3Util.java` に以下が**実在**します:

```
throw new UnsupportedOperationException(
    "Cannot add column %s since setting default values in Spark is currently unsupported");
```

- 1.10.0 に「Throw unsupported exception for ADD COLUMN with default value (#13464)」

→ リリースノートの文言だけを読むと誤読を招きます。DDL での default 設定は依然として不可です。

6.4.4 罠 2: geometry/geography はマージ済みだが使えない

- 1.11.0 タグの `spark-getting-started.md`: `geometry | | Not supported`
- 一方 `main` の `spark/v4.1/.../TypeToSparkType.java` は `case GEOMETRY:` を持つ
- PR **#17073** "Spark 4.1: Read and write geometry and geography values in Parquet"、merged **2026-07-08**、**milestone: Iceberg 1.12.0**

→ **マージ済みだが未リリース。今日時点で使えません。**「マージ済み≠ リリース済み」の典型例です。

6.4.5 発見: Spark の公式 docs が v3 に追いついていない

箇所	問題
`spark-writes.md`	**1.11.0 タグでも DV / row lineage / variant / format-version の言及が 0 件。** Spark の主要書き込みドキュメントが v3 に完全に追いついていない
`spark-ddl.md`	default values への言及なし（実際は非対応なのに、その旨の記載もない）。transform 一覧は単一引数のみ
`spark-queries.md`	`_row_id` / `_last_updated_sequence_number` メタデータカラムの記載なし（1.9.0 で reader 実装済みなのに）

v3 の実装状況はリリースノート・PR・ソースコードのレベルでは十分に追跡可能ですが、ユーザー向け docs はほぼ v2 のまま放置されています。これ自体が本調査の発見の一つです。

6.5 Apache Flink

6.5.1 サポート対象バージョン（Iceberg 1.11.0）

Flink	状態	初回	最終
1.19	**End of Life**	1.6.0	1.10.2
1.20	Maintained	1.7.0	1.11.0
2.0	Maintained	1.10.0	1.11.0
2.1	Maintained	**1.11.0**	1.11.0

6.5.2 書き込みは Spark より狭い

INSERT INTO / INSERT OVERWRITE / UPSERT のみ。MERGE / UPDATE / DELETE の SQL はありません。

UPSERT は equality delete ベース（`write.upsert.enabled`、v2+）。exactly-once はチェックポイントで担保。`DynamicIcebergSink` は 1.11.0 時点で **experimental**。

6.5.3 v3 サブ機能

サブ機能	判定	根拠
DV 書き込み	**【確認】** 1.11.0〜、**Flink 2.1**	#14197。変更ファイルは `flink/v2.1/.../sink/`
DV（equality delete 変換）	**【確認】**	`flink-writes.md`: 「`convertEqualityDeletes()` converts equality deletes to deletion vectors (**requires equality fields and table format version** >= **3**)」
DV 読み取り	**【未確認】**	Flink 固有の DV 読み取りを述べる公式ソースなし。Core #11481 経由で機能する可能性が高いが **Flink レベルの証跡なし**
row lineage 読み取り	**【確認】** 1.11.0〜	#14148
row lineage の compaction 保存	**【確認】** 1.11.0〜	#14149 “Preserve row lineage in RewriteDataFiles”
row lineage 生成（通常書き込み）	**【未確認】**	Flink sink が生成すると述べる公式ソースなし
VARIANT	**公式ソース同士が矛盾**	下記
default values 読み取り適用	**【確認】** 1.8.0〜	#12072。`defaultReader()` が `field.initialDefault()` を参照
default values 書き込み/DDL	**【未確認】**	
geometry / geography	**【明示的非対応】**	型表が `Not supported`。GitHub code search `GeometryType path:flink` → **0 件**（Spark は 8 件）。1.12.0 向け PR も見つからず
nanosecond timestamps	**【確認】** 1.9.0〜（1.11.0 強化）	型表が `timestamp(9)` / `timestamp_ltz(9)`（Note 欄空＝対応）。#12470、#15475
multi-argument transforms	**【未確認】**	

6.5.4 VARIANT — 公式ソース同士の矛盾

ソース	主張
`flink.md` 型表(**1.11.0 タグ・main 双方**)	`variant | | Not supported`
1.11.0 リリースノート	「Support Variant to Flink 2.1 (#15265)」(2026-02-26 merged)
1.11.0 タグの実コード	`flink/v2.1/.../TypeToFlinkType.java` に variant 参照 **3 件を実測**。テストも存在

→ **実装は Flink 2.1 に入っており、型対応表が未更新(ドキュメントの遅れ)と判断します。**ただし型表という公式ソースが「Not supported」と書いている事実は残ります。

6.5.5 v3 アップグレード時の制約

`flink-writes.md`(**【確認】**):

> **Table Format Version upgrade**: Dynamic sink does not support upgrading a table with dynamic records. **The job should not be running while the V2 to V3 upgrade is in progress.**

6.5.6 ナノ秒 timestamp の非対称

Flink は対応、Spark は非対応という非対称があります。Spark はコード上も `UnsupportedOperationException` に落ちます。

6.5.7 Flink docs の不整合

- `flink-ddl.md` の `format-version` の例が**すべて `'2'`**。v3 テーブル作成例なし
- `flink-writes.md` の UPSERT 節は「The table must use **v2 table format**」と v2 前提のまま。**同ファイル内で DV(v3 以上)に言及しているのに未整合**

6.6 Trino(482、2026-06-25)

6.6.1 v3 は experimental — ブログを信じないこと

docs(master、`connector/iceberg.md`)が明記:

> **Support for format version 3 is experimental.**
>
> **Version 3 support is experimental; row-level updates, deletes, and OPTIMIZE are not supported.** Tables with v3 features such as **column default values and encryption are not supported**. Version 3 is required for tables containing VARIANT columns.

一方、v3 の PR は**すべてマージ済み**です（API で `merged_at` を検証）：

PR	内容	merged
#27786	v3 read 有効化	2026-01-10
#27837	column default values	2026-01-16
#27788	deletion vector	2026-01-16
#27836	row lineage	2026-03-19
#27753	VARIANT	2026-03-30
#27893	geometry/geography	2026-06-03

コードは landed しているが、docs は今も v3 を experimental とし、row-level updates/deletes/OPTIMIZE を非対応と宣言し、default values も #27837 がマージされているにもかかわらず非対応としています。

→ **マージ済み PR のリストを「サポート済み」と読まないこと。docs が authoritative です。**

v1/v2 の読み書きは GA（MERGE、パーティション進化、スキーマ進化あり）。`format_version` の既定は 2。REST カタログはクライアントとして GA（`iceberg.catalog.type=rest`）。482 で prefixed-path storage credentials と view の `location` を追加。

> **注意**: 482 の「data loss … when deletion vectors are enabled」という注記は **Delta Lake コネクタの DV** を指す可能性があり、Iceberg の証拠として引用すべきではありません（未解決）。

6.7 Presto（PrestoDB 0.298.1、2026-06-17）— Trino とは別物

Trino と混同しないこと。機能差が実在します。

- **読み**: GA。docs は v1/v2 に限定（「SELECT table operations are supported for Iceberg format version 1 and version 2」）
- **書き**: v1/v2 で GA だが **MERGE は "No"（Presto Java）**。`format-version` は「either 1 or 2」で **v3 は legal な値ですらない**
- **Presto C++ は INSERT のみ**（DML なし）、Presto Java はフル DML
- **row lineage は読み取り専用**

v3: 部分的で、大半が未リリース。docs が認めるのは **default column values のみ**（metadata-only、`initial-default` を読み取り時に注入）。残りは prestodb/presto#27198（**Open**、2026-06-23 更新）で進行中:

- **M1（2026 年 6 月）**: API を 1.10.0 へ、DV **read**（Velox マージ済み、Prestissimo 保留）、default-value read マージ済み
- **M2（目標 2026 年 9 月）**: multi-arg transforms、**VARIANT read**、UNKNOWN 型、ナノ秒 timestamp → **VARIANT は今日の Presto では使えません**

トランクにはマージ済みだが**どのリリースにも入っていない**もの: #28058/#27943（DV + UPDATE/MERGE、2026-06-30/06-24）、#28116（field-id プロトコル + Velox bridge、2026-07-08）、#27197（native row lineage read、2026-07-02）

6.8 PyIceberg

最新: 0.11.1（2026-03-03）（PyPI の `upload_time_iso_8601` で実測）

Version	Upload (UTC)
0.11.1	**2026-03-03**
0.11.0	2026-02-10
0.10.0	2025-09-11
0.9.1	2025-04-30

`requires_python`: `<4.0.0,>=3.10.0`

6.8.1 0.11.0 の主要変更

- **ORC read サポート**
- **Sort order evolution**
- **REST scan planning**（サーバサイド scan planning、**同期のみ**。async は将来）
- **ConfigResponse の supported endpoints ネゴシエーション**（未対応操作で明快なエラー）
- スナップショットのロールバック、Entra ID auth manager
- **Breaking: Python 3.9 サポート打ち切り**

6.8.2 制約 — ここが重要（released 0.11.1 のソースを直接読んで確定）

PyIceberg のドキュメントサイトにはバージョンセレクタがなく、`main` 追従の疑いがあります。そのため以下はリリースタグ `pyiceberg-0.11.1` のソースを直接参照して照合した結果です。

6.8.2.1 1. format-version 3 は読めるが書けない【明示的非対応】

```
# table/metadata.py TableMetadataV3.model_dump_json
raise NotImplementedError("Writing V3 is not yet supported, see: https://github.com/apache/iceberg-python/i
```

- `TableMetadataV3` は**実在**し、`format_version: Literal[3]`、`next_row_id` を持つ。**パース（読み取り）は可能**
- しかし**書き込みは明示的に禁止**
- 追跡 issue apache/iceberg-python#1551 は **2026-07-17 時点で Open**（マイルストーン/紐付け PR なし）
- `TableProperties.DEFAULT_FORMAT_VERSION = 2`

→ **deletion vector / row lineage といった v3 機能は利用不可。v3 前提の設計は不可能です。**

6.8.2.2 2. equality delete は書けないだけでなく読めない【明示的非対応】

```
# table/__init__.py L2114
raise ValueError("PyIceberg does not yet support equality deletes: https://github.com/apache/iceberg/issues
# L1886 (REST scan planning 経路)
raise NotImplementedError(f"PyIceberg does not yet support equality deletes: {delete_file.file_path}")
```

Spark や Flink が equality delete を書いたテーブルは PyIceberg で開けません。「書けない」だけでない点が重要です。Flink の UPSERT は equality delete ベースなので、この組み合わせは実害が出ます。

6.8.2.3 3. merge-on-read は警告を出して copy-on-write に落ちる【明示的非対応】

```
# table/__init__.py L686
warnings.warn("Merge on read is not yet supported, falling back to copy-on-write", stacklevel=2)
```

テーブルプロパティ `write.delete.mode=merge-on-read` を設定していても**警告を出して CoW で実行**します。position delete の書き出しは行われません。

6.8.2.4 4. compaction / orphan files 削除は非対応

`pyiceberg/table/maintenance.py` の `MaintenanceTable` は **`expire_snapshots` のみ**。他のメソッドは存在しません。

6.8.2.5 5. MERGE INTO 相当はない

`upsert()` が近いですが SQL の MERGE INTO そのものではありません。

```
def upsert(self, df, join_cols=None, when_matched_update_all=True,
           when_not_matched_insert_all=True, case_sensitive=True,
           branch=MAIN_BRANCH, snapshot_properties=EMPTY_DICT) -> UpsertResult
```

真偽値 2 つのみで、**条件付き句や `WHEN MATCHED THEN DELETE` はありません**。

6.8.3 対応している操作

操作	可否
`create_namespace` / `drop_namespace` / `list_namespaces` / `namespace_exists`	○

操作	可否
create_table / create_table_if_not_exists / create_table_transaction	○
register_table / load_table / table_exists / rename_table / drop_table / purge_table	○
append（既定 fast append）	○
overwrite（overwrite_filter 可） / dynamic_partition_overwrite	○
delete（delete_filter）	○（**CoW のみ**）
upsert	○
add_files（check_duplicate_files=True 既定）	○
update_schema（add/rename/update/delete/move/union_by_name）	○
パーティション進化 update_spec	○
update_sort_order	○（0.11.0〜）
manage_snapshots（create/remove tag・branch）	○
タイムトラベル、ロールバック	○
scan → to_arrow / to_pandas / to_duckdb / to_polars / to_ray / to_daft / to_bodo	○
inspect.*（snapshots/partitions/entries/refs/manifests/history/files/…）	○
table.maintenance.expire_snapshots()	○

6.8.4 extras（PyPI の provides_extra で実測、全 24 件）

```
pyarrow, pandas, duckdb, ray, bodo, daft, polars, snappy,
hive, hive-kerberos, s3fs, glue, adlfs, dynamodb, bigquery,
sql-postgres, sql-sqlite, gcsfs, rest-sigv4, hf,
pyiceberg-core, datafusion, gcp-auth, entra-auth
```

> rest や minio という extras は**存在しません**。REST カタログは追加 extras 不要（コア機能）です。

6.8.5 設定の落とし穴

s3.path-style-access は PyIceberg に存在しません。

released 0.11.1 のソース全体を grep しても該当キーはなく、S3_FORCE_VIRTUAL_ADDRESSING = "s3.force-virtual-addressing" のみ実在します。s3.path-style-access は **Java 版 Iceberg / Spark 側のキー**であり、PyIceberg に書いても**エラーにならず黙って無視されます**（プロパティは単なる dict 参照のため）。

さらに重要なのは、**MinIO 等で path-style を使いたい場合、何も設定しなくてよい**という点です:

- _initialize_s3_fs は**プロパティが明示指定された時のみ** force_virtual_addressing を PyArrow へ渡す

- PyArrow の既定は `False` で、意味は「If false, then virtual addressing is only enabled if `endpoint_override` is empty」
- → **`s3.endpoint` を設定した時点で自動的に path-style になります**

実在する S3 キー（`pyiceberg/io/__init__.py` から実測）：

```
s3.endpoint             s3.access-key-id       s3.secret-access-key
s3.session-token        s3.region              s3.resolve-region
s3.profile-name         s3.anonymous           s3.proxy-uri
s3.connect-timeout      s3.request-timeout     s3.signer
s3.signer.uri           s3.signer.endpoint     s3.role-arn
s3.role-session-name    s3.force-virtual-addressing   s3.retry-strategy-impl
```

`s3.region` は実質必須です。未指定だと `_cached_resolve_s3_region(bucket=...)` でバケットのリージョン解決を試み、S3 以外（MinIO/RustFS）では失敗/遅延の原因になります。

FileIO の優先順位（`SCHEMA_TO_FILE_IO` より）：

scheme	優先順
`s3` / `s3a` / `s3n`	**PyArrowFileIO → FsspecFileIO**
`oss`, `gs`, `hdfs`, `viewfs`	PyArrowFileIO のみ
`abfs(s)`, `wasb(s)`	FsspecFileIO → PyArrowFileIO

> **落とし穴**: `s3` は PyArrow が優先です。`pip install pyiceberg[s3fs]` だけ入れて pyarrow も入っていると、**s3fs ではなく PyArrow が使われます**。明示するには `py-io-impl` を指定してください。

`X-Iceberg-Access-Delegation` は自動送信されます（`rest/__init__.py` L773: `session.headers.setdefault("X-Iceberg-Access-Delegation", "vended-credentials")`）。手動で付ける必要はありません。

`scope` の既定は `catalog`。Polaris では `PRINCIPAL_ROLE:ALL` の明示が必須です。

6.9 その他のエンジン（要点のみ）

6.9.1 StarRocks（4.0.1、2025-11-17）

読みは GA（v2.4+）、書きは **INSERT 系のみ**（CTAS/INSERT/OVERWRITE、v3.1+）。**DELETE/UPDATE/MERGE なし**。v3 は**なし**（tracker #63819、未アサイン）。

6.9.2 Doris（4.1.1）

V3 read は 4.1.0〜。INSERT/CTAS は GA、**UPDATE/DELETE/MERGE は 4.1.0 から experimental**（「POC before production」）。DV read あり、row lineage は experimental。

6.9.3 Impala（4.5.0、2025-03-03）

INSERT、DELETE（4.3）、UPDATE（4.4）、**MERGE（4.5）**。ただし **position delete のみ、Parquet のみ、MoR のみ**。**REST catalog はどのリリースにも入っておらず**、未リリースの 5.0 が目標（IMPALA-15144 In Progress）。v3 は**なし**。**16 ヶ月リリースが止まっています**。

6.9.4 Hive（4.2.0、2025-11-23）— 死んでいない

Attic に入っていません（attic.apache.org を実際に取得して確認。Hive は不在。なお HiveMind は Attic にあるが別プロジェクト）。

リリース: 4.0.0（2024-03）→ 4.1.0（2025-07-31）→ 4.2.0（2025-11-23）。`[DISCUSS] Next Release (4.3.0)` スレッド（2026-01-19〜2026-05-20）で **Iceberg V3 が最優先スコープ**（「Hive should be one of the core engines supporting Iceberg v3」）。

この中で唯一 IRC サーバを持つエンジンです（HMS REST Catalog、4.2.0）。4.2.0 で auto-compaction と Z-ordering を追加。

ただし勢いはありません: dev リストの流量は細く（2026 年 7 月は 1 メッセージ）、4.3.0 は~4 月目標を過ぎています。**going concern だが遅い**、というのが実態です。「Hive は死んだ」という言説を支持する一次情報は見つかりませんでした。

6.9.5 DuckDB

書きは **1.4（2025-09-16）から**（CREATE/INSERT/COPY）。UPDATE/DELETE/MERGE は**未確認**。カタログ接続は S3 Tables、R2、Polaris、Lakekeeper、Unity に対応。scan planning は**未対応**（Issue #977 オープン）。

6.9.6 ClickHouse

書きは **experimental・フラグ制**（`allow_experimental_insert_into_iceberg`）、INSERT/CREATE のみ、DML なし。カタログ経路も experimental（`allow_experimental_database_iceberg`）。

> **注意**: DuckDB / ClickHouse の**ネイティブ VARIANT 型と Iceberg v3 VARIANT を混同しないこと。**両者を結びつけるソースはありません。

6.9.7 BigQuery

2026-04-20 に **BigLake を「Lakehouse for Apache Iceberg」へ改称**。Managed Iceberg tables **GA**、BigLake Metastore（Iceberg REST カタログ）**GA**。BigLake external tables は**設計上読み取り専用**。

6.9.8 Athena

engine v3 で GA。INSERT/UPDATE/DELETE/**MERGE**、OPTIMIZE、VACUUM。DV + row lineage は 2025-11 の AWS 発表による。

6.9.9 Redshift

読みは GA（Spectrum/Glue/S3 Tables 経由）。**書き込みは【未確認】**--- AWS のドキュメントに肯定も否定も見つかりませんでした。**これは本調査で最大の真の未知です。**

6.10 未確認事項

- **Flink の row lineage 生成（通常書き込みパス）/ DV 読み取り / multi-arg transforms / default values 書き込み** --- 一次情報を探した限り**本当に見つかりませんでした**
- **Spark 3.5 での row lineage compaction 保存**（PR タイトルが "Spark 4.0" 限定）
- **Trino / Presto の CoW vs MoR 設定**
- **Redshift の Iceberg 書き込み可否**
- **Databricks の新規 managed テーブルの既定 format version**
- **Snowflake の externally-managed テーブルにおける MERGE の制限**（列挙された DML 制約以外）
- **Dremio の並行ライタ / 外部エンジンとの競合セマンティクス**
- **iceberg-rust の対応状況**

6.11 出典

Apache Iceberg

- Multi-engine support: https://raw.githubusercontent.com/apache/iceberg/main/site/docs/multi-engine-support.md
- Releases: https://iceberg.apache.org/releases/ / https://github.com/apache/iceberg/releases/tag/apache-iceberg-1.11.0
- Spark docs（生ソース）: `https://raw.githubusercontent.com/apache/iceberg/main/docs/docs/spark-{writes,ddl,configuration,queries,procedures}.md`
- Flink docs（生ソース）: `https://raw.githubusercontent.com/apache/iceberg/main/docs/docs/flink-{writes,ddl,configuration}.md`
- PR #11561（Spark DV write）、#13310（Spark 4.0 row lineage）、#14297（VARIANT shredding）、#17073（geometry、1.12.0）
- PR #14197（Flink DV write）、#14148（Flink row lineage read）、#15265（Flink VARIANT）

PyIceberg

- PyPI JSON API: https://pypi.org/pypi/pyiceberg/json
- 0.11.0 release blog: https://iceberg.apache.org/blog/apache-iceberg-python-0.11.0-release/
- Configuration: https://py.iceberg.apache.org/configuration/
- released tag `pyiceberg-0.11.1` ソース（`io/__init__.py`, `io/pyarrow.py`, `table/__init__.py`, `table/metadata.py`, `table/maintenance.py`, `catalog/rest/__init__.py`）

- Issue #1551（V3 write）: https://github.com/apache/iceberg-python/issues/1551

エンジン各社

- Trino: https://trino.io/docs/current/connector/iceberg.html / https://raw.githubusercontent.com/trinodb/trino/master/docs/src/main/sphinx/connector/iceberg.md
- Presto: https://prestodb.io/docs/current/connector/iceberg.html / https://github.com/prestodb/presto/issues/27198
- Snowflake: https://docs.snowflake.com/en/user-guide/tables-iceberg-access-using-external-query-engine-snowflake-horizon / release notes 2026-05-07, 2026-05-26
- Databricks: https://docs.databricks.com/aws/en/iceberg/iceberg-v3 / https://www.databricks.com/blog/catalog-and-next-era-apache-icebergtm
- Dremio: https://docs.dremio.com/dremio-cloud/developer/data-formats/iceberg/

7 06. 運用・メンテナンス・性能

調査基準日: 2026-07-17 / Iceberg 1.11.0 基準。プロパティ名・デフォルト値・シグネチャはすべて `main` の原文から逐語確認

7.1 公式の分類: “Recommended” と “Optional”

最初にこれを知ってください。一般的な認識と逆です。

`maintenance.md` の分類:

分類	作業
Recommended Maintenance	**Expire Snapshots / Remove old metadata files / Delete orphan files**
Optional Maintenance	**Compact data files / Rewrite manifests**

公式は compaction を「オプション」、スナップショット期限切れと metadata 掃除を「推奨(必須級)」に位置づけています。

そしてこの因果関係が決定的です(`maintenance.md` の`!!! info`):

> **データファイルは、タイムトラベルやロールバックに使われうるスナップショットから参照されなくなるまで削除されない。定期的なスナップショット期限切れが未使用データファイルを削除する。**

→ **`expire_snapshots` を回さない限り、compaction は容量を一切解放しません。むしろ増えます**(新しいファイルを書いた上で、古いファイルもスナップショットに参照されたまま残るため)。運用上もっとも誤解されやすい点です。

7.2 A. メンテナンス作業

7.2.1 A-1. compaction / `rewrite_data_files`

なぜ必要か(公式の理由):「Iceberg はテーブル内の各データファイルを追跡する。データファイルが増えるほど manifest に格納されるメタデータが増え、小さなデータファイルは**ファイルオープンコストにより不要な量のメタデータと非効率なクエリ**を引き起こす」

シグネチャ:

引数	必須	型	説明
table	必須	string	
strategy		string	binpack または sort。**デフォルトは binpack**
sort_order		string	Zorder は zorder(c1,c2,c3) 形式。それ以外は (ColumnName SortDirection NullOrder) のカンマ区切り
options		map<string,string>	
where		string	**フィルタに合致するデータを含みうる全ファイルが書き換え対象に選ばれる**点に注意

3 戦略の違い:

- **bin-pack**（デフォルト）: 小ファイルを結合し、大ファイルは分割する。**行の並べ替えを伴わない**ため最も安価。
- **sort**: 指定した sort order で全データをソートして書き直す。専用オプション compression-factor（既定 1.0）、shuffle-partitions-per-file（既定 1）。
- **z-order**: strategy => 'sort' かつ sort_order => 'zorder(c1,c2)' として指定します（**独立した strategy 名ではありません**。実装上重要）。専用オプション var-length-contribution（既定 8）、max-output-size（既定 2147483647）。

主要オプションとデフォルト値:

オプション	デフォルト	意味
target-file-size-bytes	536870912（512MB）	write.target-file-size-bytes の既定値
min-file-size-bytes	**目標サイズの 75%**	これ未満は他条件に関わらず書き換え候補
max-file-size-bytes	**目標サイズの 180%**	これ超過は他条件に関わらず書き換え候補
min-input-files	5	この数以上のファイルを持つグループは書き換え（最低 2 ファイル必要）
max-concurrent-file-group-rewrites	5	
partial-progress.enabled	false	全体完了前に部分コミットを許可
partial-progress.max-commits	10	
partial-progress.max-failed-commits	max-commits の値	

オプション	デフォルト	意味
use-starting-sequence-number	true	
rewrite-job-order	none	bytes-asc/bytes-desc/files-asc/files-desc/none
rewrite-all	false	全ファイル強制書き換え
max-file-group-size-bytes	107374182400（100GB）	巨大パーティションをリソース制約内で分割処理するため
delete-file-threshold	2147483647	
delete-ratio-threshold	0.3	
output-spec-id	現在の spec id	
remove-dangling-deletes	false	**追加コミットが 1 回発生**
max-files-to-rewrite	null	

> **min-file-size-bytes**=75% / **max-file-size-bytes**=180% **は、実質的に「384MB〜922MB の範囲外のファイルは書き換え候補」という公式のヒステリシス帯**を意味します。**これが「望ましいファイルサイズ帯」について公式から導ける最も具体的な指針です。**

remove-dangling-deletes の公式注記: dangling delete はデータシーケンス番号のみに基づいて削除され、**グローバル equality delete、無効な equality delete、生存データファイルに合致しなくなった position delete を含む position delete ファイルには適用されません**。

7.2.2 A-2. rewrite_manifests

compaction とは目的が異なります。これはデータではなく**インデックスの再編成**です。

> Iceberg は manifest list と manifest 内のメタデータを使ってクエリプランニングを高速化し、不要なデータファイルを枝刈りする。**メタデータツリーはテーブルのデータに対するインデックスとして機能する。**
>
> manifest は追加された順に自動的にコンパクションされるため、**書き込みパターンが読み取りフィルタと整合している場合**にクエリが速くなる（例: 時間範囲クエリに対する時間到着順の書き込み）。**テーブルの書き込みパターンがクエリパターンと整合しない場合、**rewriteManifests でメタデータを書き直してデータファイルを manifest に再グループ化できる。

→必要になるのは「書き込み順とクエリ述語がずれているとき」です。

引数	必須	型	説明
table	必須	string	
use_caching		boolean	**既定 false**。有効化すると executor のメモリ使用量が増える
spec_id		int	

引数	必須	型	説明
sort_by		array	**頻繁にクエリされる transform を選ぶと不要な manifest をスキップして planning 時間を削減できる**

公式注記:「この procedure は対象テーブルを参照する**全てのキャッシュ済み Spark プランを無効化する**」

7.2.3 A-3. expire_snapshots

なぜ必要か:「Iceberg の各 write/update/delete/upsert/compaction は新しいスナップショットを生成し、**スナップショット分離とタイムトラベルのために古いデータとメタデータを残し続ける**」

放置時:「スナップショットは expire されるまで蓄積し続ける」→ストレージ課金の無限増加とメタデータサイズ増大。

この procedure は、未期限のスナップショットがまだ必要としているファイルを削除することはありません。

引数	必須	型	デフォルト
table	必須	string	
older_than		timestamp	**5 日前**
retain_last		int	**1**
max_concurrent_deletes		int	**デフォルトではスレッドプールを使わない**
stream_results		boolean	**大容量時の Spark driver OOM を防ぐため true 推奨**
snapshot_ids		array of long	
clean_expired_metadata		boolean	参照されなくなった partition spec / schema も整理

運用上の罠:「**ブランチやタグから参照されているスナップショットは削除されません。**デフォルトではブランチとタグは決して expire しませんが、history.expire.max-ref-age-ms で変更できます。**main ブランチは決して expire しません。**」

→タグを 1 つ付けただけで、そのスナップショットと依存ファイルが無期限に残ります。Amazon S3 Tables ではこれがさらに深刻な挙動になります(C-1)。

7.2.4 A-4. remove_orphan_files ※最も危険な操作

なぜ必要か:「Spark やその他の分散処理エンジンでは、**タスクやジョブの失敗によりテーブルメタデータから参照されないファイルが残る**ことがあり、また通常のスナップショット期限切れではファイルが不要と判断できず削除できない場合がある」

Iceberg の書き込みは「先にデータファイルを書く →最後にメタデータをコミットする」順序なので、**コミット前に落ちたジョブが書いたデータファイルは、どのメタデータからも参照されないまま物理的に残ります。**これは設計上不可避であり、だからこそ専用の掃除機構が必要です。

公式が明記する危険性（逐語）:

> **書き込みが完了するのに要すると見込まれる時間より短い保持間隔で orphan ファイルを削除するのは危険である。進行中のファイルが orphan とみなされて削除されると、テーブルを破壊する可能性があるからである。デフォルトの間隔は 3 日である。**
>
> **Iceberg はどのファイルを削除すべきか判断する際にパスの文字列表現を使う。一部のファイルシステムではパスが時間とともに変化しうるが、それでも同じファイルを表す。例えば HDFS クラスタの authority を変更した場合、作成時に使われた古いパス URL はどれも現在のリスティングに現れるものと一致しない。これは RemoveOrphanFiles を実行した際にデータ損失につながる。**

実務結論:

- **必ず `dry_run => true` で先に確認する**（公式の最初の例がこれです）
- **`older_than` をデフォルトの 3 日より短くしない。**特に長時間バッチや Flink の長時間チェックポイントがある環境では、最長書き込み時間より確実に長く取る
- **`prefix_mismatch_mode` のデフォルトは `ERROR`（例外送出）であり、これは安全側の設計です。**これを `DELETE` にするのは、スキーム/authority の不一致が本当に同一ファイルを指すと確証がある場合のみ。**安易な `DELETE` 指定はまさに上記のデータ損失シナリオそのものです**
- `gc.enabled`（既定 true）を false にすると GC 操作自体が禁止されます。外部でファイルを共有している場合の防御策になります

引数	デフォルト	説明
`older_than`	**3 日前**	
`dry_run`	**false**	
`location`	テーブルの location	
`stream_results`		**有効時、出力は最大 20,000 パスのサンプルを含む**
`equal_schemes`	**`map('s3a,s3n','s3')`**	等価とみなすスキーム
`equal_authorities`		
`prefix_mismatch_mode`	**ERROR**	ERROR（例外）/ IGNORE / DELETE
`prefix_listing`	false	

1.11.0 の関連改善:「Unique table locations（新カタログプロパティによりテーブルストレージパスに UUID を付与）は、**`DeleteOrphanFiles` がリネームされたテーブルのファイルを削除しうるデータ損失シナリオを防ぐ**」--- orphan 削除のデータ損失リスクが実在の問題として認識され、構造的に対処された証左です。

7.2.5 A-5. `rewrite_position_delete_files` と v3 deletion vector

2 つの目的:

- **Minor Compaction**: 小さな position delete ファイルを大きなものに結合

- **Remove Dangling Deletes**:「`rewrite_data_files` の後、書き換えられたデータファイルを指す position delete レコードは常に削除対象としてマークされるとは限らず、テーブルの生存スナップショットメタデータに追跡され続けることがある。**これが 'dangling delete' 問題として知られる**」

オプションは `rewrite_data_files` とほぼ同じですが、**`target-file-size-bytes` の既定が 67108864 (64MB)**（データファイルの 512MB ではない）。

7.2.5.1 v3 でどう変わるか

02-table-spec.md で見た通り、v3 では:

- position delete ファイルの新規追加が**禁止**
- 「1 データファイル = 高々 1 つの DV」が spec レベルで保証
- DV は既存 position delete を置き換える

→ **`rewrite_position_delete_files` の存在意義が構造的に縮小します。**小さな delete ファイルが際限なく積み上がるという v2 の主要な病理が消え、「minor compaction」の対象そのものが生まれません。

ただし完全には消えません:

1. **Puffin ファイル自体の書き直しは spec 上不要**なので、「参照されない DV blob を含む Puffin ファイル」が残りえます → `remove_orphan_files` の領分
2. **v2 テーブルには依然として必要**です。v2 → v3 アップグレード後も既存の position delete ファイルは有効なまま残るため、移行期は両方の考慮が必要

   【未確認】: v3 の DV に対する専用の compaction procedure（DV 版 `rewrite_position_delete_files` に相当）は `spark-procedures.md` に**見つかりませんでした**。`rewrite_data_files` の `remove-dangling-deletes` は「position and equality deletes」を対象とする記述で、DV に対する挙動は docs から読み取れません。

7.2.6 A-6. 古い metadata ファイルの削除 — 後戻りできない設定

これは公式分類で "Recommended" です。

> 各 metadata ファイルは `metadata-log` フィールドで古い metadata ファイルを追跡する。追跡される metadata ファイル数は `write.metadata.previous-versions-max` で定義される。
>
> `write.metadata.delete-after-commit.enabled=true` を設定すると、**追跡されている (tracked)** metadata ファイルを保持しつつ、新しいものが作られるたびに最古のものを削除する。**これは metadata log に追跡されている metadata ファイルのみを削除し、orphan 化した metadata ファイルは削除しない。**

公式の具体例が罠の本質を示しています（逐語）:

> `write.metadata.delete-after-commit.enabled=false` かつ `write.metadata.previous-versions-max=10` で 100 コミット後、**10 個の tracked metadata ファイルと 90 個の orphaned metadata ファイル**を持つことになる。**これら 90 個の orphaned metadata ファイルは、後から `write.metadata.delete-after-commit.enabled=true` に設定しても削除できない。既に untracked だからである。orphan file deletion procedure でしかクリーンできない。**

→ **write.metadata.delete-after-commit.enabled は既定 false であり、テーブル作成時に有効化しておかないと後戻りできません。**ストリーミング/高頻度コミットのテーブルでは作成時点で必ず有効化してください。既に肥大したテーブルでは remove_orphan_files（= A-4 の危険な操作）が唯一の手段になります。

7.3 B. 重要なテーブルプロパティ

すべて docs/docs/configuration.md（main）から逐語確認。

7.3.1 Write properties

プロパティ	デフォルト	説明
write.format.default	parquet	parquet / avro / orc
write.target-file-size-bytes	**536870912（512MB）**	
write.delete.target-file-size-bytes	**67108864（64MB）**	
write.parquet.row-group-size-bytes	134217728（128MB）	
write.parquet.compression-codec	**zstd**	zstd / brotli / lz4 / gzip / snappy / uncompressed
write.distribution-mode	**未設定**（エンジン依存）	none / hash / range
write.delete.distribution-mode 等	未設定	delete/update/merge 用に**独立に存在**
write.metadata.metrics.default	**truncate(16)**	none / counts / truncate(length) / full
write.metadata.metrics.max-inferred-column-defaults	**100**	**メトリクスを収集する最大列数**
write.metadata.delete-after-commit.enabled	**false**	← A-6 参照
write.metadata.previous-versions-max	100	
write.object-storage.enabled	false	ファイルパスにハッシュ要素を加える
write.delete.mode / write.update.mode / write.merge.mode	**copy-on-write**	v2 以降で merge-on-read 可
write.*.isolation-level	serializable	serializable / snapshot
write.delete.granularity	partition	partition / file
write.spark.fanout.enabled	false	**メモリ使用量が増える**

metrics モードの定義:

- `none`: 永続化しない
- `counts`: カウントのみ
- `truncate(length)`: カウント + 切り詰めた境界値。**切り詰めは string と binary にのみ適用**
- `full`: 完全な境界値

7.3.2 Table behavior properties

プロパティ	デフォルト
`commit.retry.num-retries`	**4**
`commit.retry.min-wait-ms`	100
`commit.retry.max-wait-ms`	60000（1 分）
`commit.retry.total-timeout-ms`	**1800000（30 分）**
`commit.status-check.num-retries`	3
`commit.manifest.target-size-bytes`	8388608（8MB）
`commit.manifest.min-count-to-merge`	100
`commit.manifest-merge.enabled`	true
`history.expire.max-snapshot-age-ms`	**432000000（5 日）**
`history.expire.min-snapshots-to-keep`	1
`history.expire.max-ref-age-ms`	**`Long.MAX_VALUE`（無期限）。`main` は決して expire しない**
`gc.enabled`	true

7.3.3 Read properties

プロパティ	デフォルト
`format-version`	**2**（1.4.0 以降）
`read.split.target-size`	134217728（128MB）
`read.split.open-file-cost`	4194304（4MB）
`read.parquet.vectorization.enabled`	true
`read.orc.vectorization.enabled`	**false**

矛盾の指摘: `configuration.md` は `write.distribution-mode` を「not set, see engines for specific defaults」とする一方、`spark-writes.md` は「**Iceberg 1.2.0 以降、`hash` が新しいデフォルト**」と記述します。

これは矛盾ではなく**階層の違い**です --- テーブルプロパティとしては未設定で、**Spark エンジンが実行時に `hash` を要求する**という構造です。したがって**他エンジン（Flink 等）では挙動が異なりえます**。また「**Spark は 3.5.0 より前では CTAS/RTAS で distribution mode を尊重しない**」という重要な例外も明記されています。

7.4 C. アンチパターン（仕様レベルの発生機序）

7.4.1 1. Over-partitioning

なぜ小ファイルが生まれるのか（仕様上の根拠）：

`spark-writes.md` が明記する通り「**ファイルは Iceberg のパーティション境界をまたげない**（a file cannot span an Iceberg partition boundary）」--- これが over-partitioning が**必ず**小ファイルを生む理由です。パーティションを細かくすると、各パーティションのデータ量が 512MB を大きく下回り、パーティション境界がそのままファイル境界になります。

さらに `write.distribution-mode=hash`（Spark 既定）ではパーティション値でハッシュ分散されるため、パーティション数が多いほどタスクあたりのデータが薄くなり、**1 パーティション ×1 タスクごとに小ファイルが 1 つ**生まれます。

> **公式な「パーティションあたり推奨サイズ」の数値は、`partitioning.md`・`performance.md`・`configuration.md` を確認した範囲で見つかりませんでした（未確認）。**「1 パーティション最低 1GB」等の一般に流布する目安には公式の裏付けが確認できません。

7.4.2 2. 高頻度小コミットによるメタデータ肥大

なぜ起きるか:「**Iceberg はテーブルメタデータを JSON ファイルで追跡する。テーブルへの各変更は、原子性を提供するために新しい metadata ファイルを生成する**」

つまり**コミット 1 回 = metadata JSON 1 個 + スナップショット 1 個 + manifest list 1 個**が確定的に生成されます。これは最適化ではなく**原子性のための構造的要件**です。

`write.metadata.delete-after-commit.enabled` が既定 false なので、**何も設定しなければ metadata JSON は無限に蓄積し、`previous-versions-max=100` を超えた分は untracked（＝後から自動削除不能）になります**（A-6）。

緩和策の `commit.manifest-merge.enabled`（既定 true）は manifest レベルの話であり、metadata JSON とスナップショットの蓄積は別途対処が必要です。

7.4.3 3. 複数ライタ競合と commit retry

なぜ起きるか: 01-architecture.md の通り、Iceberg は**楽観的並行制御**です。コミットは「現在のスナップショットを読む→ 新スナップショットを作る → アトミックにスワップ」という CAS 的操作で、他のライタが先にスワップすると失敗し `commit.retry.*` に従いリトライします（既定 4 回、100ms〜60 秒のバックオフ、合計 30 分）。

実務上の帰結: compaction ジョブ自体が最も競合を起こしやすいライタです。大きなパーティションを書き換える compaction は長時間データファイルを保持し、その間の並行書き込みと衝突します。

対策:

- `partial-progress.enabled=true` + `partial-progress.max-commits` で小さく刻んでコミットし、1 コミットあたりの競合窓を狭める
- `max-file-group-size-bytes`（既定 100GB）を下げ、`where` 句で書き込みが起きていないパーティションに限定する

`write.*.isolation-level` を `serializable` から `snapshot` に下げると競合検出が緩和されますが、**これは分離レベルを落とす判断**であり、正しさへの影響を理解した上でのみ行ってください。

7.4.4 4. Orphan files

2 つの独立した機序があります:

- **タスク/ジョブ失敗**: 書き込み順序(データ先、メタデータ後)から構造的に不可避(A-4)
- **untracked metadata**: `previous-versions-max` を溢れた metadata JSON(A-6)

7.4.5 5. メタデータ JSON 肥大

上記 2 と 4 の合流点です。**設計時に決めるべき設定であり、運用開始後の是正はコストが高い**(唯一の手段が A-4 の危険な操作)というのが結論です。

7.5 D. 性能

7.5.1 D-1. 多段プルーニングと劣化条件

プルーニングの仕組み自体は 01-architecture.md を参照してください。ここでは**劣化する条件**を扱います。

`performance.md`:「プランニング時に上限・下限でデータファイルをフィルタすることにより、Iceberg はクラスタ化されたデータを使ってタスクを実行せずにスプリットを排除する。**場合によってはこれは 10 倍の性能改善になる**」(出典: Strata NY 2018)

7.5.1.1 劣化条件(仕様から導かれるもの)

1. **`field_summary` の範囲が広がる(段 1 の失効)**: manifest 内のデータファイルが多数のパーティション値にまたがると lower/upper bound が広範囲になり、どの述語にも「合致しうる」と判定されて manifest スキップが効かなくなります。**これがまさに `rewrite_manifests` の `sort_by` が対処する問題**です。書き込み順とクエリ述語がずれているテーブルで発生します。
2. **列メトリクスが存在しない(段 3 の失効)**: **`write.metadata.metrics.max-inferred-column-defaults` が 100 なので、101 列目以降の列にはデフォルトでメトリクスが収集されません**(pre-order 走査順)。ワイドテーブルで後方の列に述語をかけても bounds プルーニングが効かず全ファイル読み込みになります。`write.metadata.metrics.column.<col>` で個別指定が必要です。
3. **`truncate(16)` による境界値の粗さ**: デフォルトは string/binary の境界値を 16 文字で切り詰めます。**長い共通プレフィックスを持つ文字列列(URL、UUID 文字列等)では境界値が実質同一になり、プルーニングが効きません。**
4. **未知の partition transform**: inclusive projection が `true` になるため、そのフィールドでの枝刈りが完全に無効化されます。
5. **小ファイル大量発生**: manifest エントリ数に比例してプランニングコストが増大します。

7.5.2 D-2. メタデータ肥大とサーバサイド scan planning

performance.md の設計目標は「**マルチペタバイトのテーブルでも、テーブルメタデータをふるいにかける分散 SQL エンジンを必要とせず、単一ノードから読める**」です。

しかしこの前提は**メタデータが単一ノードのメモリ・I/O 帯域に収まる限り**で成立します。スナップショットが expire されず manifest が肥大すると前提が崩れ、Spark driver が全 manifest を読む際のボトルネック（および OOM）になります。

`expire_snapshots` / `remove_orphan_files` の `stream_results` オプションが「Spark driver の OOM を防ぐため true 推奨」と明記されているのは、この問題が実在することの公式な証左です。

server-side scan planning はこれへのプロトコルレベルの解です。仕様・実装ともに存在します（03-rest-catalog.md 参照）：

- 仕様: 1.7.0（2024-11-08、PR #9695）
- 実装: **1.11.0（2026-05-19、PR #13400 + Spark 統合 #14963）**

1.11.0 リリースブログ:

> **以前は、全てのクライアントがどのファイルを読むか判断するために manifest list と manifest を自分で取得しなければならなかった。server-side planning により、クライアントは関連する scan task のみを受け取り、driver のメモリ圧迫を軽減し、クエリエンジンに透過的なサーバサイド最適化を可能にする。**

ただし**カタログサーバ側の実装が必須**であり、その最適化の質はカタログ実装に依存します。

> **【未確認】**: Spark/Flink 等の各エンジン統合層がどのバージョンで remote scan planning を実際に使うようになるかについて、公式 docs 上の明確な記述は確認できませんでした。「Spark（Iceberg 1.11+）が Scan API をサポート」とする第三者ブログがありますが、**公式 docs で裏付けが取れなかったため採用しません**。

7.5.3 D-3. ファイルサイズの指針（出典のある数値のみ）

数値	値	出典
データファイル目標サイズ	**512MB**	`write.target-file-size-bytes` 既定
delete ファイル目標サイズ	**64MB**	`write.delete.target-file-size-bytes` 既定
Parquet row group	128MB	`write.parquet.row-group-size-bytes` 既定
読み取りスプリット目標	128MB	`read.split.target-size` 既定
ファイルオープン推定コスト	4MB	`read.split.open-file-cost` 既定
compaction 書き換え下限	**目標サイズの 75%**	`min-file-size-bytes` 既定
compaction 書き換え上限	**目標サイズの 180%**	`max-file-size-bytes` 既定
compaction 発火ファイル数	5	`min-input-files` 既定
manifest マージ目標サイズ	8MB	`commit.manifest.target-size-bytes` 既定
manifest マージ発火数	100	`commit.manifest.min-count-to-merge` 既定

数値	値	出典
compaction 例での明示値	500MB	`maintenance.md` の Java 例
S3 Tables compaction 目標	512MB（設定範囲 64--512MB）	AWS S3 Tables docs

7.5.3.1 公式な推奨値が「存在しない」もの（未確認）

以下は `partitioning.md`、`performance.md`、`configuration.md` を確認しましたが**見つかりませんでした**:

- **1 パーティションあたりの推奨データ量**
- **1 テーブルあたりの推奨パーティション数上限**
- **manifest あたりの推奨エントリ数**
- **スナップショット保持数の推奨値**（デフォルト以外）

出典のない数値目安は書かない方針のため、ここは「公式な推奨値は見つからなかった」と記します。代わりに使える公式由来の指針は上記の**「384MB〜922MB のヒステリシス帯」**です。

7.5.3.2 Spark 固有の重要な制約

`spark-writes.md`（逐語）:

> Spark で Iceberg にデータを書く際、**Spark は Spark タスクより大きいファイルを書けず、ファイルは Iceberg のパーティション境界をまたげない**点に注意することが重要である。これは、Iceberg は常に `write.target-file-size-bytes` に成長した時点でファイルをロールオーバーするが、**Spark タスクが十分に大きくない限りそれは起こらない**ことを意味する。**ディスク上に作られるファイルサイズは Spark タスクよりずっと小さくなる。**

→ **`write.target-file-size-bytes` は「上限（ロールオーバー閾値）」であって「保証値」ではありません。**512MB に設定しても、Spark タスクが小さければ小さいファイルしかできません。**設定だけで解決しない、実務で最も頻繁に踏まれる誤解です。**

7.5.4 D-4. `write.distribution-mode` の選択

設計上の背景（`spark-writes.md` 逐語）:

> Iceberg のデフォルト Spark writer は、**各 Spark タスク内のデータがパーティション値でクラスタ化されていることを要求する。この分散は、書き込み中に開いたまま保持されるファイルハンドルの数を最小化するために必要である。**Iceberg 1.2.0 以降、デフォルトで Iceberg は Spark にこの分散に合うようデータを事前ソートすることも要求する。

モード	シャッフル	ファイル数への影響	コスト	位置づけ
`none`	なし	**最悪**。手動ソートしないとタスク × パーティションの組み合わせ数だけファイルが生成される	最安	「Iceberg の**以前の**デフォルト」
`hash`	ハッシュ交換	良好。1 パーティションのデータが特定タスクに集約	中	「**新しいデフォルト**」(1.2.0〜)
`range`	レンジ交換(2 段階)	良好 + グローバルソート	**最高**	「**テーブルが sort-order 付きで作られた場合にデフォルトで使われる**」

実務判断のポイント:

1. **`none` は「速い」のではなく「Spark に仕事をさせない」だけです。**データが既にパーティション値でクラスタ化されている場合にのみ最適。そうでなければ**小ファイルの温床**になり、**シャッフルコストを払わないぶん、compaction で払う**ことになります。
2. **`none` + fanout writer の罠**:「ソートは `write.spark.fanout.enabled` で回避できるが、**これは各書き込みタスクが完了するまで全ファイルハンドルが開いたままになる**」。**「ソートを避ける」代償はメモリです。**パーティション数が多いと executor の OOM に直結します。
3. **`range` は sort-order 付きテーブルの暗黙のデフォルト**。「より高価だが、ソート列がクエリで使われる場合、グローバル順序は読み取り性能に有益になりうる」。**D-1 の段 3（列 bounds プルーニング）を効かせたいなら、書き込みコストを払って読み取りの枝刈り効率を買う取引**です。
4. **AQE との相互作用**: `hash` と `range` の両方で「**Spark の Adaptive Query planning によりタスクのさらなる分割と結合が起こりうる**」。distribution-mode だけでファイル数は決まりません。
5. **delete/update/merge には別プロパティ**が独立に存在し、**すべて既定未設定**です。MoR で delete を多用する場合、これらが未設定のままだと delete ファイルの分散が制御されません。

7.5.5 D-5. MoR と CoW のトレードオフ

観点	CoW（デフォルト）	MoR（v2 position delete）	MoR（v3 deletion vector）
書き込みコスト	**高**。ファイル全体を書き直す	**低**。delete ファイルを追加するだけ	**低**。DV を書くだけ
読み取りコスト	**ゼロ**	**高**。合致する delete ファイル群を読みマージ	**中**。1 データファイルにつき高々 1 つなので照合対象が確定的
メタデータ増加	スナップショット分のみ	**delete ファイルが際限なく蓄積**しうる	**蓄積しない**（1:1 の上限が spec で保証）

観点	CoW（デフォルト）	MoR（v2 position delete）	MoR（v3 deletion vector）
追加メンテナンス	compaction のみ	**`rewrite_position_delete_files` が事実上必須**	構造的に不要
dangling delete	なし	**あり**	緩和（ただし Puffin は残りうる）

equality delete が最も危険:「delete ファイルの partition spec が unpartitioned」の場合にも適用対象になるため、**グローバル equality delete は全データファイルに対して照合が必要**になります。MoR の最悪ケースです。しかも `remove-dangling-deletes` は公式注記の通り**グローバル equality delete には適用されない**ため、自動的な掃除も効きません。

DV の最大の改善は「不変条件による予測可能性」です。v2 では読み取りコストが「これまで何回 delete したか」に依存して**単調悪化**しました。v3 ではこれが構造的に不可能になります。定量的改善ではなく**漸近的な性質の変化**です→ 02-table-spec.md

v3 採用の前提: `format-version` の既定は 2 のままで、**v3 にしても `write.delete.mode` の既定が CoW なので DV は自動では使われません。**エンジン側の v3 対応も前提です → 05-engine-support.md

---

7.6 E. マネージドサービスの自動メンテナンス

7.6.1 C-1. Amazon S3 Tables

「以下のオプションは**テーブルバケット内の全テーブルでデフォルトで有効**である」

操作	プロパティ	バケット単位	テーブル単位	**デフォルト**	最小
Compaction	`targetFileSizeMB`	不可	可	**512MB**	64MB
Snapshot management	`minimumSnapshots`	不可	可	**1**	1
Snapshot management	`maximumSnapshotAge`	不可	可	**120 時間（5 日）**	1 時間
Unreferenced file removal	`unreferencedDays`	**可**	不可	**3 日**	1 日
Unreferenced file removal	`nonCurrentDays`	**可**	不可	**10 日**	1 日

compaction 戦略（4 種）: Auto（既定。sort order があれば `sort`、なければ `binpack`）/ Binpack / Sort / Z-order。「**`z-order` と `sort` は `binpack` より高いコストを発生させうる**」

「compaction はオブジェクトを結合する際、**テーブルの行レベル delete の効果も適用する**」

7.6.1.1 ※ 最重要の落とし穴: Snapshot management が fail-closed する

公式逐語:

> Snapshot management は、`metadata.json` ファイルまたは `ALTER TABLE SET TBLPROPERTIES` で設定した retention 値を**サポートしない**。以下のいずれかの条件が存在する場合、**snapshot management はテーブル全体で失敗し、Amazon S3 はスナップショットを一切 expire も削除もしない**:
>
> - **ユーザー定義のタグまたはブランチ** --- テーブルにユーザー定義のタグやブランチが存在する場合、snapshot management はテーブル全体で失敗する。**これは、そのタグやブランチの retention 期間が短くても適用される。**
> - **Iceberg snapshot retention テーブルプロパティ** --- `history.expire.max-snapshot-age-ms` または `history.expire.min-snapshots-to-keep` がテーブルプロパティとして設定されている場合、**設定値に関わらず** snapshot management はテーブル全体で失敗する。

これは極めて重大です。 Iceberg のベストプラクティスに従って `history.expire.*` を設定する、あるいはリリースタグを打つといった**ごく自然な運用が、自動メンテナンスをサイレントに全面停止させます。**しかも失敗は**ストレージ課金の増加としてしか現れません**。

→ **`GetTableMaintenanceJobStatus` API による能動的な監視が必須**です(失敗時は `FAILED` ステータスと原因メッセージが返ります)。CloudTrail でも追跡可能です。

7.6.1.2 その他の制約

- **削除は復旧不能**:「**noncurrent オブジェクトの削除は永久的で、これらのオブジェクトを復旧する方法はない**」「閲覧・復旧には **AWS Support への問い合わせが必要**」
- **外部参照は保護にならない**:「**テーブル外部からのこれらオブジェクトへの参照は、snapshot management による削除を妨げない**」
- **形式・型の制約**: compaction は Parquet/Avro/ORC 対応だが**既定で Parquet を書く**。**データ型 `Fixed` 非対応。圧縮 `brotli`、`lz4` 非対応。Parquet の row-group サイズ 128MB が強制される**
- **楽観的並行の競合は消えない**:「楽観的並行では、**ユーザーと compaction のトランザクションが競合してトランザクションが失敗しうる**」「compaction ジョブは失敗時にリトライする。**パイプライン側でもリトライロジックを使うことを推奨する**」
- **【未確認】**`rewrite_manifests` に相当する自動操作は、確認した範囲の docs に存在しません。提供されるのは 3 種のみです。D-1 の manifest レイアウト起因の劣化は S3 Tables では自動的に解決されません

7.6.2 C-2. Databricks Predictive Optimization

自動実行: OPTIMIZE / VACUUM / ANALYZE **対象**: **Unity Catalog managed tables（Delta Lake および Iceberg）**

限界（公式に「サポートしない」と記載）:

- **外部テーブル** --- 対象外
- **OpenSharing recipient としてロードされたテーブル** --- 対象外

- **「Predictive Optimization が実行する OPTIMIZE は ZORDER 操作を実行しない」** --- S3 Tables が z-order 戦略を提供するのと対照的

保持期間: VACUUM は `delta.deletedFileRetentionDuration` を尊重し、**既定の窓は 7 日**。

有効化: Premium プラン以上。「**2024 年 11 月 11 日以降に作成されたアカウントではデフォルトで有効**」。既存アカウントへの段階的ロールアウトは **2026 年 8 月完了予定**（調査時点で進行中）。

> **【未確認】**: `delta.deletedFileRetentionDuration` は Delta Lake のプロパティです。**UC managed Iceberg テーブルに対する VACUUM の保持期間が同じプロパティで制御されるのか、Iceberg の `history.expire.*` が尊重されるのかは、参照した docs から判別できませんでした。** Iceberg テーブルを Predictive Optimization に委ねる場合、この点は事前検証が必要です。

7.6.3 C-3. マネージド自動メンテナンスの共通の限界

1. **「肩代わり」されるのは compaction と GC のみで、設計判断は肩代わりされません。** パーティション設計、sort order の選択、`write.metadata.delete-after-commit.enabled` の初期設定（A-6 の後戻り不能問題）はユーザーの責任のまま残ります。S3 Tables の `sort/z-order` が「`sort_order` の定義が必須」であることは、この境界を明確に示しています --- **サービスは実行を代行するが、何でソートすべきかは決めてくれません。**
2. **楽観的並行の競合はマネージドでも消えません。** AWS が明示的に「パイプライン側でもリトライロジックを使うことを推奨」と書く通り、これは仕様に由来する本質的性質であり、サービス層で隠蔽できません。むしろ**自動 compaction が追加のライタとして競合を増やす**側面があります。
3. **自動化と Iceberg 標準機能が衝突しうる。** S3 Tables の snapshot management が fail-closed する仕様はその最も鋭い例です。**「Iceberg のベストプラクティスに従うこと」と「マネージドサービスの自動メンテナンスを機能させること」が両立しないケースが実在します。**
4. **監視は依然として必要。**「サイレント失敗」を検出する手段を運用に組み込まない限り、課金増加として顕在化するまで気づけません。
5. **【未確認】** `rewrite_manifests` 相当は、確認した範囲ではどちらのサービスも自動提供していません。

7.7 出典

Apache Iceberg 公式（生ソース）

- Spark Procedures: https://raw.githubusercontent.com/apache/iceberg/main/docs/docs/spark-procedures.md
- Configuration: https://raw.githubusercontent.com/apache/iceberg/main/docs/docs/configuration.md
- Maintenance: https://raw.githubusercontent.com/apache/iceberg/main/docs/docs/maintenance.md
- Performance: https://raw.githubusercontent.com/apache/iceberg/main/docs/docs/performance.md

- Spark Writes: https://raw.githubusercontent.com/apache/iceberg/main/docs/docs/spark-writes.md
- Table Spec: https://raw.githubusercontent.com/apache/iceberg/main/format/spec.md
- 1.11.0 リリースブログ: https://raw.githubusercontent.com/apache/iceberg/main/site/docs/blog/posts/2005-19-iceberg-1.11.0-release.md
- Remote scan planning: PR #13400

AWS

- Maintenance for tables (S3 Tables)
- Considerations and limitations
- PutTableMaintenanceConfiguration API

Databricks

- Predictive optimization

採用しなかった情報:「Spark（Iceberg 1.11+）が Scan API をサポート」等の第三者ブログ（Medium/Dremio、DEV Community、Substack）の記述は、公式 docs で裏付けが取れなかったため採用していません。

8 07. エコシステムと業界動向

調査基準日: 2026-07-17 / リリース日・コミット数・contributor 数は GitHub API および ASF 配布サイト（dlcdn.apache.org）で実測

> **測定方法の注記**: contributor 数は過去 365 日のコミットの GitHub ログインをユニーク集計したものです。**GitHub アカウント紐付けのないコミットは脱落する**ため、下限値として読んでください。

8.1 A. 他フォーマットとの比較

8.1.1 A-1. 最新リリース（実測）

プロジェクト	最新版	リリース日	ソース
Apache Iceberg	1.11.0	**2026-05-19**	公式サイト「released on May 19, 2026」/ git tag の tagger date
Delta Lake	4.3.1	2026-07-08	GitHub Releases
Apache Hudi	1.2.0	2026-05-23	GitHub Releases
Apache Paimon	1.4.2	2026-06-23/24	git タグ / ASF dist
Apache XTable	**0.3.0-incubating**	**2025-06-04**	GitHub Releases
Apache Polaris	1.6.0	2026-07-09	GitHub Releases

補足:

- **Paimon は GitHub Releases を作成しない**運用（タグと ASF dist のみ）
- **Hudi は 1.2.0 の後に 0.x 系の保守リリースを継続**: 0.15.1（2026-05-21）、0.14.2（**2026-06-08**）。1.0.0 GA から 1 年半経った 2026 年に 0.14 系の保守が必要 = **旧バージョン残留ユーザーが相当数いる**ことの示唆
- **Iceberg のマイナー間隔は伸長傾向**: 1.8.0（2025-02-13）→ 1.9.0（+約 2.5 ヶ月）→ 1.10.0（+約 4.5 ヶ月）→ 1.11.0（+**約 8.3 ヶ月**）

8.1.2 A-2. 健全性シグナル（2026-07-17 実測）

> **※ 生のコミット数を横並びで比較してはいけません。**ボット比率が **0.4%〜85%** と 2 桁違います。本節はボット除外値を併記します。この注意を無視すると結論が逆転します（下記）。

	Iceberg	Delta Lake	Hudi	Paimon	Nessie	Polaris	XTable
コミット（過去 365 日、**生**）	1,696	1,203	1,141	**2,278**	1,652	2,100	42
うち**ボット**	309（18.2%）	0	4（0.4%）	13（0.6%）	**1,407 (85.2%)**	744（35.4%）	6
人間のコミット	**1,387**	1,203	1,137	**2,265**	**245**	1,356	36
ユニーク作者（過去 365 日）	**270**	140	78	166	**22**	106	17
全期間 contributor	407	391	382	363	79	156	47
Stars	9,054	8,911	6,188	3,340	---	2,014	1,194
Open issues（PR 含む）	851	1,560	**3,017**	677	---	---	162

ボットの内訳: Iceberg は全て `dependabot[bot]`、Nessie と Polaris は `renovate[bot]`（Nessie はさらにリリース自動化ボットを含む）。

読み取れること（事実として）：

- **Paimon の開発速度が最も速い。**人間コミットで **2,265 vs Iceberg 1,387 = 1.63 倍**。生の数字（1.34 倍）より**差は大きい**です。Iceberg のコミットの 18% が dependabot だからです。
- **Iceberg は直近 1 年のユニーク作者数で突出**（270 人、2 位 Paimon 166 人の 1.63 倍）。「特定ベンダー依存度が相対的に低い」ことを示す指標の一つです。> **ただしこの結論は指標定義に依存する脆いものです。全期間 contributor で測ると Iceberg 407 / Delta 391 / Hudi 382 / Paimon 363 とほぼ横並び（1.12 倍）で、「突出」とは言えません。**「直近 1 年の活動」を見るか「累積の関与者」を見るかで結論が変わります。所属組織の分布も**未確認**です（下記）。
- **Nessie の「1,652 コミット」は健全性の証拠になりません。85.2% が renovate ボットで、人間のコミットは 245 件。**しかも実質 1 人に集中しています（`snazy` が 212、残り全員で 33）。ユニーク作者は **22 人**（Iceberg 270 の 1/12）。リリース頻度が高いのは、依存更新を自動リリースしている構造の反映です。→ 04-catalog-implementations.md
- **Polaris の 2,100 も 35% がボット。**人間換算では **1,356 < Iceberg 1,387** で逆転します。
- **Hudi は直近 1 年のユニーク作者 78 人と最少**（XTable 除く）。Open issues 3,017 は突出。
- **主要プロジェクトはいずれも最終 push が測定日近辺 = どれも死んでいません。**

  方法論上の教訓:「コミット数が多い＝活発」は、**ボットを除外しない限り成立しません。**この報告書は初版でその罠を踏み、Nessie を過大評価していました。→ 99-methodology.md

8.1.2.1 なぜ「所属組織の分布」を出さないのか

「特定ベンダーへの依存度」を語るなら、本来は contributor の所属組織を見るべきです。実際に GitHub API から取得を試みましたが、**報告に耐える精度が出ないため、本報告書では出しません。**理由は 2 つの手法それぞれの限界にあります。

手法 1: コミットメールのドメイン集計（Iceberg・直近 365 日の実測）

ドメイン	件数	所属が分かるか
`users.noreply.github.com`	42	× 匿名化（GitHub の既定設定）
`gmail.com`	32	× フリーメール
`apache.org`	17	× **ASF のコミッタ用アドレス。雇用主を示さない**
`qq.com`	3	× フリーメール
`nvidia.com` / `amazon.com` 等	数件	○

約 9 割が判定不能でした。とくに `apache.org` は「ASF のコミッタである」ことしか示さず、所属企業とは無関係です。

手法 2: GitHub プロフィールの `company` フィールド

直近 365 日の上位コミッタ 10 名で試すと、**所属が読めたのは 4 名だけ**（残る 6 名は未設定）。しかもこのフィールドは**自己申告で、転職しても更新されない**ことが多く、表記も揺れます（`Databricks` と `Databricks,` `@databricks`、コミュニティ名の `@apache` など）。

結論: どちらの手法も、**所属を公開している一部の人だけが母集団になる**という偏りを避けられません。その偏った標本で「ベンダー中立性」を定量的に語ることは、B-12 に挙げた「実測値であっても指標の定義を確認しないと誤った結論を導く」の一例になります。また、匿名化されたメールから所属を推定する行為は、本人が公開していない属性の推測にあたります。**この指標は「調べていない」のではなく「この方法では信頼できる値が出せない」**という理解が正確です。

8.1.3 A-3. 設計思想の違い

4 つのフォーマットは「同じ問題を別の設計で解いている」ため、まず要点を横並びで示します。

	Iceberg	Paimon	Hudi	Delta Lake
中核データ構造	不変スナップショット + マニフェストツリー	bucket ごとに独立した **LSM-tree**	**timeline**（アクション履歴）+ インデックス	トランザクションログ
前提ワークロード	**バッチ**	**連続 upsert**（ストリーミング）	upsert + インデックス活用	Spark 中心（4.0 で他エンジンへ拡張）
エンジンとの関係	エンジン非依存を志向	**Flink と密結合**（Flink Table Store 由来）	複数エンジン対応	**Delta Kernel** で非 Spark エンジンも単一実装で読み書き
他フォーマットとの相互運用	---	Iceberg 互換メタデータを生成可（`metadata.iceberg.storage`）	---	**UniForm** で Iceberg メタデータを自動生成（GA）
注目すべき方向性	v4 でメタデータ構造を刷新中	Iceberg V3 の deletion vector が互換の基盤に	**AI/ML へ軸足**（VECTOR/BLOB 型、Lance 統合）	Delta 5.0 で Iceberg v4 のメタデータ構造採用を提案

以下、それぞれの詳細です。

Iceberg: 不変スナップショット + マニフェストツリー。OCC で、コミットはメタデータポインタのアトミックな CAS。**バッチ前提の設計**。

Paimon: パーティションを bucket に分割し、**各 bucket が独立した LSM-tree**（LevelDB/RocksDB と同系統）。Flink Table Store 由来のため設計前提が Flink と密結合。**連続 upsert に最適化**。Iceberg 互換メタデータの生成も可能（`metadata.iceberg.storage` 設定）で、**Iceberg V3 の deletion vector が互換の技術的基盤**になりました。

Hudi: timeline（アクション履歴）+ インデックス中心。1.2.0（2026-05-23）は table version 9 を 1.1.1 から維持（アップグレード不要）。**1.2.0 の方向性は AI/ML**: VECTOR・VARIANT・BLOB の 3 論理型、`read_blob()` / `hudi_vector_search()` の Spark SQL 関数、**Lance フォーマットのネイティブ統合**、Flink での Record-Level Index フルサポート。

> **Hudi は「テーブルフォーマット競争」から降りて AI/ML データレイクへ軸足を移している**ように読めます。ただしこれは筆者の**評価**であり、Hudi PMC の公式表明ではありません。

Delta Lake: Delta 4.0 で **Delta Kernel** を導入（Trino/Flink など非 Spark エンジンが単一のコア実装で読み書き）。UniForm は GA で、Delta テーブルから Iceberg メタデータを自動生成し、Iceberg クライアントがファイル書き換えなしに読めます。

> **UniForm の Hudi 対応は long-standing の "coming soon" のまま**で、2026 年 7 月時点の提供状況は未確認です。

8.1.4 A-4. 相互運用 — XTable は実測で停滞

ベンダー中立の相互運用を担う唯一のプロジェクトである XTable の実測値:

- 最新リリース **0.3.0-incubating = 2025-06-04（13 ヶ月以上リリースなし）**
- コミット: 直近 90 日 **10 件** / 365 日 **42 件** / 2024-07-17 以降の 2 年間で 148 件 → **明確な減速カーブ**
- Incubation 入りは **2024-02-11、2026 年 7 月時点でまだ incubating**（約 2 年 5 ヶ月）
- 直近コミットは Onehouse 関係者と dependabot が大きな比率

評価: 上記の実測値は、「XTable は停滞している」という判断を裏付けます。ただし**プロジェクトは放棄されていません**（最終 push 2026-07-14）。Incubator ステータスページに活動への懸念の明記は**確認できませんでした**。

対照的に、ベンダー主導の相互運用の方が実際に動いています:

- Databricks UniForm（GA）
- Microsoft OneLake のメタデータ仮想化（双方向）
- Fivetran: Parquet を 1 回書いて **Iceberg と Delta のメタデータを同時に維持**

> **これはロックインの観点で重要な指摘です。**相互運用が「ベンダーの善意」に依存しており、中立の仕組みは動いていません。

8.2 B.「Iceberg は事実上の標準になったのか」

「Iceberg が勝った」という言説は多いものの、根拠として挙がる事実と、それに反する事実の両方があります。まず優位を支持する事実、次に反証を並べ、最後に整理します。

8.2.1 B-1. Iceberg 優位を支持する事実

8.2.1.1 Databricks（Tabular 買収 2024 の「その後」）

- **2025-06-12**:「Announcing full Apache Iceberg support in Databricks」。UC の IRC API 経由で Managed Iceberg テーブルの読み書き。「Virtually any Iceberg client compatible with the REST spec, such as Apache Spark, Apache Flink, or Trino can read and write to Unity Catalog」
- **2026-05-28**: Managed Iceberg **GA**、Iceberg v3 **GA**、Foreign Iceberg **GA**、Iceberg クライアントへの External Sharing **GA**
- **最も重い事実**: 同ブログで Databricks は「**With Iceberg v4, we are rethinking the core metadata structure from the ground up**」と述べ、**Delta 5.0 が Iceberg v4 と同じ adaptive metadata tree 構造を採用することを提案**しています。

  つまり Delta の作者自身が、Delta の次期メジャーを Iceberg v4 のメタデータ設計に寄せる提案をしています。これは「競争」ではなく「統合」としてフレーミングされています。

8.2.1.2 Apache Polaris

- Snowflake と Dremio が共同作成し **2024 年 8 月 ASF へ寄贈**
- **TLP 卒業**（2026-02-19 公式アナウンス）。**卒業投票は +1 が 27 票、反対 0**
- PMC に Dremio, Snowflake, Google, Microsoft, Confluent, LanceDB のエンジニアが参加（**ASF 公式名簿での照合は未実施**）
- 健全: 1.6.0（2026-07-09）、過去 365 日で 2,100 コミット（うち 35% はボット、人間 1,356 → A-2）

8.2.1.3 各クラウド

- **AWS**: S3 Tables（Iceberg 専用のマネージドストレージ）。2026 年も拡張継続（GovCloud 2026-02-23、Taipei/New Zealand 2026-05-29）。**Iceberg V3 対応済み**
- **GCP**: **2026-04-20 に BigLake を「Lakehouse for Apache Iceberg」へ改称。製品名に Iceberg を冠しました。**Managed Iceberg tables **GA**、BigLake Metastore（IRC）**GA**
- **Azure/Microsoft**: OneLake は **Delta Lake がネイティブ既定フォーマット**。Iceberg は**メタデータ仮想化**で対応。**Iceberg V3 互換は「今後の重点」= 未完了**

8.2.1.4 Snowflake

Iceberg v3 GA（2026-05-07）、Snowflake ストレージ for Iceberg GA（2026-06-01）、**Databricks Unity Catalog への Iceberg 書き込み対応 GA on Azure（2026-04-06）**--- つまり**競合のカタログへ書きに行くことまでやっています。**

8.2.1.5 周辺エコシステム

Confluent Tableflow、Fivetran Managed Data Lake Service（**既定で Apache Polaris ベースの self-hosted Iceberg REST カタログを同梱**）。

8.2.2 B-2. 反証 — 「事実上の標準」と言い切れない事実

1. **Confluent Tableflow は Iceberg 専用ではありません。** Delta Lake + Databricks Unity Catalog 統合が **2025-10-29 に GA** し、**Iceberg と Delta を 1 トピックで同時有効化できます**。エコシステムは Iceberg に**収斂しておらず、両対応に張っています**。
2. **Fivetran も両対応。** Iceberg **または** Delta に変換し、Parquet を 1 回書いて**両方のメタデータを同時維持**する設計。「Iceberg を選んだ」のではなく「**選ばせない**」戦略です。
3. **Microsoft Fabric が標準で使うのは依然 Delta です。** Iceberg 対応は仮想化レイヤー経由で、V3 互換は未完了。Azure スタックでは Delta が本命の座を保っています。
4. **Databricks の Iceberg 対応には条件が付きます。** Onehouse の Kyle Weller 氏（VP of Product、**Iceberg 競合ベンダー = 利害関係者である点に留意**）が 2026-06-11 に指摘した内容のうち、Databricks 公式ドキュメントの引用に基づくものは重いです:
   - **Unity Catalog が必須** → Polaris や Nessie を代替カタログとして使えない
   - **非対称な相互運用**: IRC 接続には向きがあり、UC は**サーバとしては働くがクライアントとしては働きません**。外部エンジン（Spark/Flink/Trino）が UC に繋ぎに来ることはできますが、**Databricks 側から Polaris や Nessie のテーブルを IRC 経由で読みに行くことはできません**
   - Foreign Iceberg テーブルは **"read-only in Databricks and have limited platform support"**、**credential vending 非対応**
   - Managed Iceberg でも partition transforms / branching / tagging / streaming が欠落し、独自機能（Liquid Clustering 等）で代替
   - Databricks 自身の Lakeflow / Zerobus Ingest / AI Search は主に Delta 対応

   → **「Iceberg 対応」は「Iceberg 仕様のフルサポート」を意味しません。** ただし個別の制限事項の現時点での正確性は**未確認**（docs は頻繁に更新されるため）。
5. **Paimon が最速で開発されています**（人間コミット 2,265/年、Iceberg の 1.63 倍。生では 1.34 倍だが Iceberg は 18% が dependabot → A-2）。Iceberg に吸収される兆候はありません。
6. **Hudi は AI/ML 領域へ差別化**（VECTOR/BLOB 型、Lance 統合）。Iceberg にない機能軸です。
7. **新規参入があります**: **DuckLake v1.0 が 2026-04-13 にリリース**（production-ready 仕様 + DuckDB 拡張、後方互換保証）。**メタデータを全て SQL データベース（SQLite/PostgreSQL/DuckDB）に置く**という、Iceberg と根本的に異なる設計。v1.1 は 2026 年 9 月予定。**フォーマット戦争は終結していません。**
8. **Iceberg v4 は仕様として未採択です** → 02-table-spec.md。（ただし Java 実装は 1.10.0 以降 `format-version=4` を受理します。「未採択」=「作れない」ではありません。）Iceberg 自身が「バッチ前提のメタデータ設計」という積み残しを抱えたままです。

8.2.3 B-3. 結論

「デファクト標準になった」と単純に言うのは不正確です。 より事実に忠実な整理:

8.2.3.1 (a) カタログ・インターチェンジ層としては、IRC 仕様が事実上の共通語になりつつある

根拠がここに集中しています:

- Databricks が UC に IRC API を実装(2025-06)
- GCP の BigLake Metastore が IRC で GA
- Fivetran が Polaris を既定同梱
- Polaris が**反対 0 で TLP 卒業**し PMC が 6 社超に分散
- Snowflake が競合の UC へ Iceberg 書き込みを GA

「相互接続の合意点」としては Iceberg が勝っていると言えます。

8.2.3.2 (b) しかし「単一の実装フォーマット」としては勝っていない

Fabric のネイティブは Delta、Confluent と Fivetran は両対応、Paimon は Iceberg の 1.63 倍(人間コミット換算)の開発速度、DuckLake が 2026 年に v1.0 到達。

8.2.3.3 (c) 最も重要な事実は「統合」の方向

Databricks が Delta 5.0 に Iceberg v4 のメタデータ構造を採用提案(2026-05-28、公式ブログ)。これが実現すれば「Iceberg が勝った/Delta が負けた」ではなく**両者が同一メタデータを共有する**世界になります。ただし**提案段階であり、Delta 5.0 も Iceberg v4 もリリースされていません**。

8.3 C. Iceberg を使うべきでない場面

Iceberg が向かない場面もあります。以下は、設計上の性質から不向きと言えるケースと、その場合の代替です。

8.3.1 C-1. 低レイテンシ・高頻度コミットのストリーミング

なぜ不向きか(仕様レベル): Iceberg は OCC で、コミットごとに新しいメタデータツリーを書き、ポインタを atomic CAS で入れ替えます。競合すると**負けた書き手は新しい state に基づきメタデータツリーを書き直してリトライ**します。つまり**コミット頻度に対して書き込み増幅(write amplification)が効き、並行書き手が増えるほど衝突リトライが増えます**。

これは推測ではありません。 Snowflake の公式エンジニアリングブログ(2026-06-04、Iceberg Summit 2026 recap)自身がこう述べています:

> **Iceberg's metadata tree was built for batch workloads, and its write amplification creates commit latencies that streaming can't tolerate.**

そして v4 の adaptive metadata tree と single-file commit がこれを直接狙うと説明しています。**Iceberg の主要推進企業が制約を公式に認めている**、というのが最も強い根拠です。

代替: **Paimon**（bucket ごとの独立 LSM-tree で連続 upsert に最適化）。あるいはストリーミング層（Kafka/Flink state）で保持し、Iceberg へはマイクロバッチで落とす。

> **未確認**:「Paimon 1〜5 分 vs Iceberg 1 時間以上」といった具体的レイテンシ比較はベンダー/個人ブログ由来で、**一次情報での裏取りができていません**。自環境でのベンチマークを推奨します。

8.3.2 C-2. 頻繁な小規模更新 / 更新が激しい CDC

なぜ不向きか: CoW では小さな更新でもファイル全体を書き直します。MoR では delete file / DV が compaction 間に累積し、読み取り時のマージコストが増え続けます。結果として **compaction を運用し続けることが前提**になります。

さらに 02-table-spec.md の通り、**CDC 目的で v3 の row lineage を当てにしても、equality delete を使うエンジン（Flink upsert 等）では追跡されません。**

代替: Paimon（LSM の compaction がアーキテクチャに内包）、Hudi（MoR + Record-Level Index）。

8.3.3 C-3. 小規模データ

なぜ不向きか: Iceberg の利点（パーティション進化、スナップショット分離、大規模メタデータのプルーニング）は**大規模テーブル・複数エンジンでこそ効きます**。数 GB 規模では、カタログ運用・compaction・スナップショット期限切れ・orphan file の削除といった**保守ジョブの運用コストが利得を上回ります**（06-operations.md の作業量を見てください）。

代替: DuckDB + Parquet、**DuckLake**（メタデータを SQL DB に置くためカタログサーバーの運用が不要。v1.0 が 2026-04-13 に production-ready 宣言）、通常の RDBMS。

8.3.4 C-4. 単一エンジンしか使わない

なぜ不向きか: **Iceberg の存在理由はエンジン非依存の共有です。** Databricks だけ / Snowflake だけ/ BigQuery だけなら、そのプラットフォームのネイティブ形式の方が最適化（Liquid Clustering、Predictive Optimization 等）を享受できます。

実際 Databricks は Managed Iceberg でも **partition transforms / branching / tagging / streaming が使えず独自機能で代替**しているという指摘があり（Onehouse、2026-06-11、**要独立検証**）、**「Iceberg を選んだのに Iceberg の機能が使えない」**状態が起こりえます。

代替: そのプラットフォームのネイティブ形式。将来の可搬性が欲しいなら UniForm / OneLake メタデータ仮想化で「後から出す」方が合理的な場合があります。

8.3.5 C-5. Flink 中心のストリーミング特化

なぜ不向きか: Paimon は Flink Table Store 由来で**設計前提が Flink と密結合**です。Iceberg の Flink コネクタはチェックポイント間隔でコミットするため、**パーティション数 × チェックポイント頻度で小ファイルが機械的に量産されます**。

さらに 05-engine-support.md の通り、**Flink の Iceberg 書き込みは INSERT/UPSERT のみで MERGE/UPDATE/DELETE の SQL がありません。**

代替: Paimon。ただし Paimon は Iceberg 互換メタデータを生成できるので、**Paimon で書いて Iceberg として読ませる**構成が取れます。

8.3.6 C-6. Azure/Fabric 中心の環境

なぜ不向きか: OneLake のネイティブは Delta。Iceberg は仮想化経由で、**Iceberg V3 互換は Microsoft 自身が「今後の重点」と位置づけており未完了**です。V3 の DV / VARIANT を前提にした設計は現時点で成立しない可能性があります。

代替: Delta Lake（ネイティブ）+ 必要に応じてメタデータ仮想化で Iceberg 公開。

8.3.7 C-7. AI/ML の非構造データを同居させたい

なぜ不向きか: Iceberg 1.11.0 時点で VECTOR/BLOB 相当のネイティブ型やベクトル検索は**確認できませんでした**（1.11.0 の中身の完全な確認は未実施）。

代替: **Hudi 1.2.0**（VECTOR/VARIANT/BLOB 型、`hudi_vector_search()`、Lance フォーマット統合）。

8.3.8 C-8. PyIceberg だけで完結させたい

なぜ不向きか: 05-engine-support.md の通り、PyIceberg は **v3 テーブルを書けず、equality delete を読めず、MoR も使えず、compaction もありません。**「Python だけで Iceberg を運用する」は現時点で成立しません。

代替: PyIceberg は読み取りと軽量な書き込みに使い、メンテナンスと重い DML は Spark に任せる。

8.4 D. 実務上の含意

1. **フォーマット選択は以前ほど不可逆ではありません。** UniForm / OneLake 仮想化 / Paimon の Iceberg 互換 / Fivetran の二重メタデータにより、「主要ワークロードに最適な形式で書き、他形式は公開で賄う」が現実的です。ただし**この相互運用はベンダー実装が担っており、ベンダー中立の XTable は実測上停滞している**ことは、ロックインの観点でリスクとして認識すべきです。
2. **カタログ選択の方がフォーマット選択より重い可能性があります。** Databricks が UC 必須・外向き IRC 非対応であるなら、**フォーマットが Iceberg でもカタログでロックされます**。Polaris が反対 0 で TLP 卒業し PMC が分散していることは、中立カタログの選択肢として意味を持ちます → 04-catalog-implementations.md

3. **2026 年後半〜2027 年の最大の変数は Iceberg v4 と Delta 5.0 です。**Databricks の統合提案が実現するか、dev ML で分岐するかで絵が変わります。**現時点で v4 前提の設計判断をすべきではありません。**ただし理由は「作れないから」ではありません --- **Iceberg Java 1.10.0+ は `format-version=4` を受理してしまいます**（02）。**仕様が未採択のまま変わりうる**ことが理由です。作れてしまう分、むしろ注意が必要です。

8.5 未確認事項

- **Polaris の TLP 卒業日**: Incubator ステータスページは 2026-02-15、公式ブログ/プレスは 2026-02-19。理事会決議日と公表日の差と推測されるが**未確定**
- **Paimon / Iceberg の contributor 所属組織の分布**（ベンダー集中度）--- 未測定
- **Paimon vs Iceberg の具体的レイテンシ数値** --- 一次情報での裏取り不可
- **Onehouse が指摘する Databricks の個別制限事項の現時点での正確性** --- 独立検証未実施（同社は利害関係者）
- **UniForm の Hudi 対応**の 2026 年 7 月時点の提供状況
- **Iceberg 1.11.0 の機能詳細**（VECTOR 型等の有無を含む）--- リリースノート全文の確認は未実施
- **Delta 4.1.0 のリリース日**: GitHub タグは 2026-02-26 だが、一部二次情報は「2026 年 3 月」と記載。**GitHub 実測値を採用**

8.6 出典

- Apache Iceberg Releases / Reliability docs
- Databricks: Unity Catalog and the Next Era of Apache Iceberg (2026-05-28)
- Databricks: Announcing full Apache Iceberg support (2025-06-12)
- Databricks: UniForm GA / UniForm docs
- Apache Polaris: TLP 卒業 (2026-02-19) / Incubator status
- Snowflake: Iceberg Summit 2026 Recap --- V4 spec (2026-06-04)
- Apache XTable Incubator status / GitHub
- Apache Hudi 1.2 release notes
- Apache Paimon Iceberg compatibility
- AWS S3 Tables
- Google Cloud: Iceberg managed tables in BigQuery
- Microsoft: Use Iceberg tables with OneLake
- Confluent: Tableflow Delta Lake + Unity Catalog (2025-10-29)
- Fivetran Managed Data Lake Service
- DuckLake v1.0 (2026-04-13)
- Onehouse: Databricks Iceberg Support Has a Catch (2026-06-11) **※競合ベンダー発**

9 08. 選定ガイド（要求から技術選択へ）

調査基準日: 2026-07-17

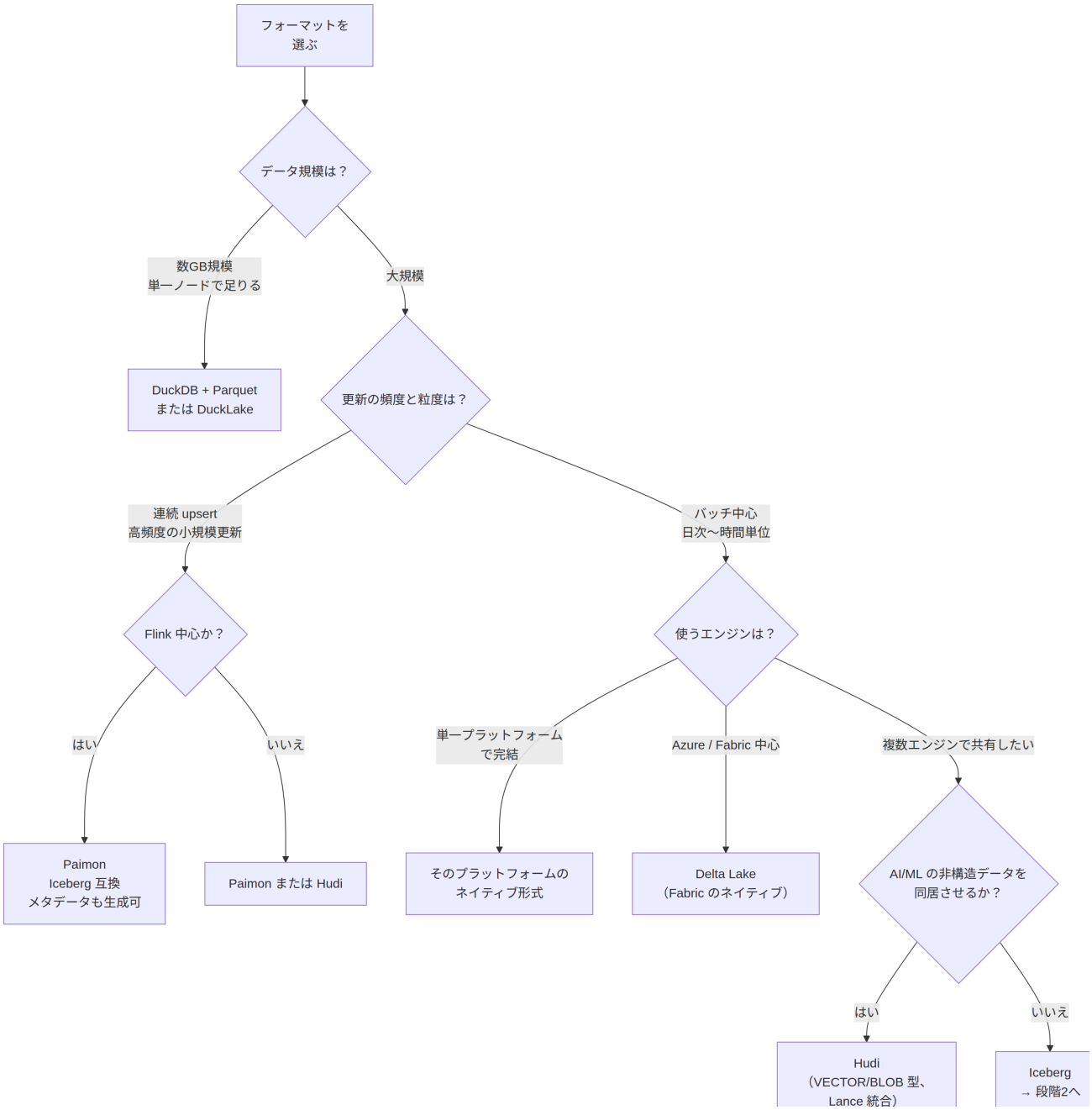
この章に新しい事実はありません。02〜07 で個別に述べた判断材料を、**要求から選択にたどり着く順序**に組み替えたものです。各分岐の根拠は元の章へリンクしています。

判断は 3 段階に分かれます。**上流の選択が下流を制約する**ので、この順に決めるのが手戻りが少なくなります。

1. **フォーマット** --- Iceberg を使うのか、他が適しているのか
2. **カタログ** --- Iceberg を選んだ場合、どのカタログか
3. **エンジン** --- やりたい操作がそのエンジンで本当にできるか

9.1 段階 1: フォーマットを選ぶ

Iceberg が常に最適とは限りません。設計上の性質から不向きなケースがあります（詳細は 07 C 章）。



各分岐の根拠:

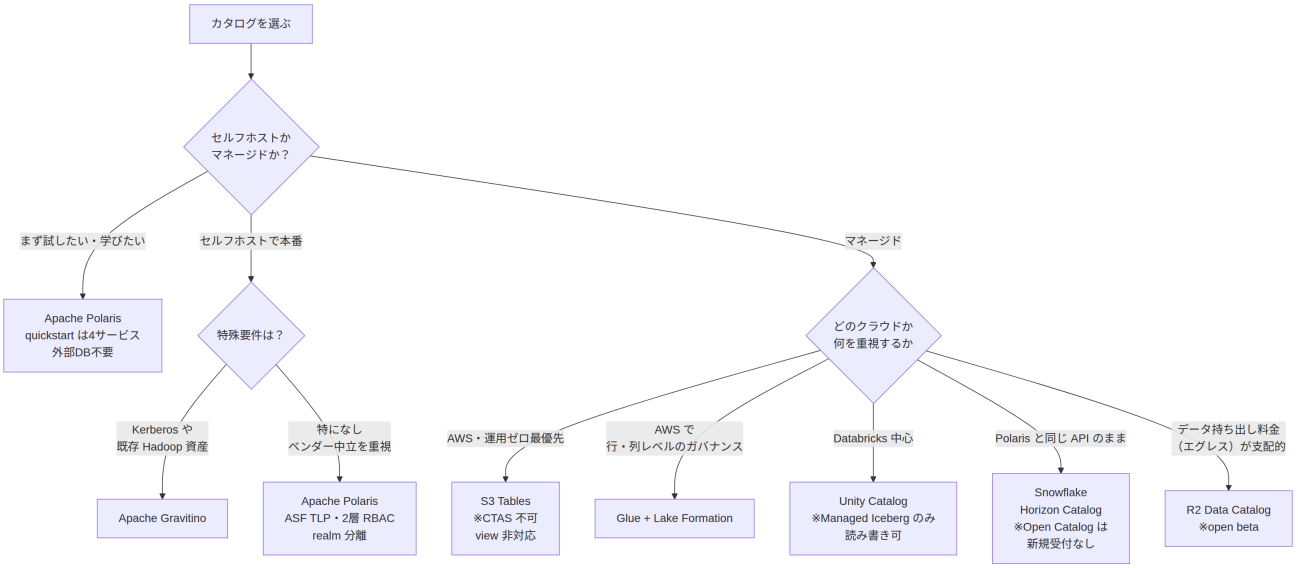
分岐	理由	詳細
小規模データ	カタログ運用・compaction・スナップショット期限切れの保守コストが利得を上回る	C-3
連続 upsert / 高頻度更新	Iceberg は OCC でコミットごとに書き込み増幅が効く。Paimon は bucket ごとの LSM-tree で compaction を内包	C-1, C-2

分岐	理由	詳細
Flink 中心	Iceberg の Flink 連携は INSERT/UPSERT のみで MERGE/UPDATE/DELETE の SQL が無い。Paimon は Flink Table Store 由来	C-5
単一プラットフォーム	Iceberg の存在理由はエンジン非依存の共有。1 つで完結するならネイティブ形式の最適化を享受できる	C-4
Azure / Fabric	OneLake のネイティブは Delta。Iceberg は仮想化経由で V3 互換は未完了	C-6
AI/ML の非構造データ	Iceberg 1.11.0 に VECTOR/BLOB 相当の型は確認できず(**機能詳細は未検証**)。Hudi 1.2.0 が対応	C-7

後戻りは以前より容易です。UniForm（Delta → Iceberg）、Paimon の Iceberg 互換メタデータ、OneLake のメタデータ仮想化により、「主要ワークロードに最適な形式で書き、他形式は公開で賄う」が現実的になっています。ただしこの相互運用は**ベンダー実装が担っており、ベンダー中立の XTable は停滞している**点は認識しておくべきです(A-4)。

9.2 段階 2: カタログを選ぶ

Iceberg を選んだら、次はカタログです。**フォーマットより重い選択になりうる**点に注意してください。Databricks UC のように「そのカタログを使うこと」が条件になる実装があり、フォーマットが Iceberg でもカタログでロックインされます(07 D)。



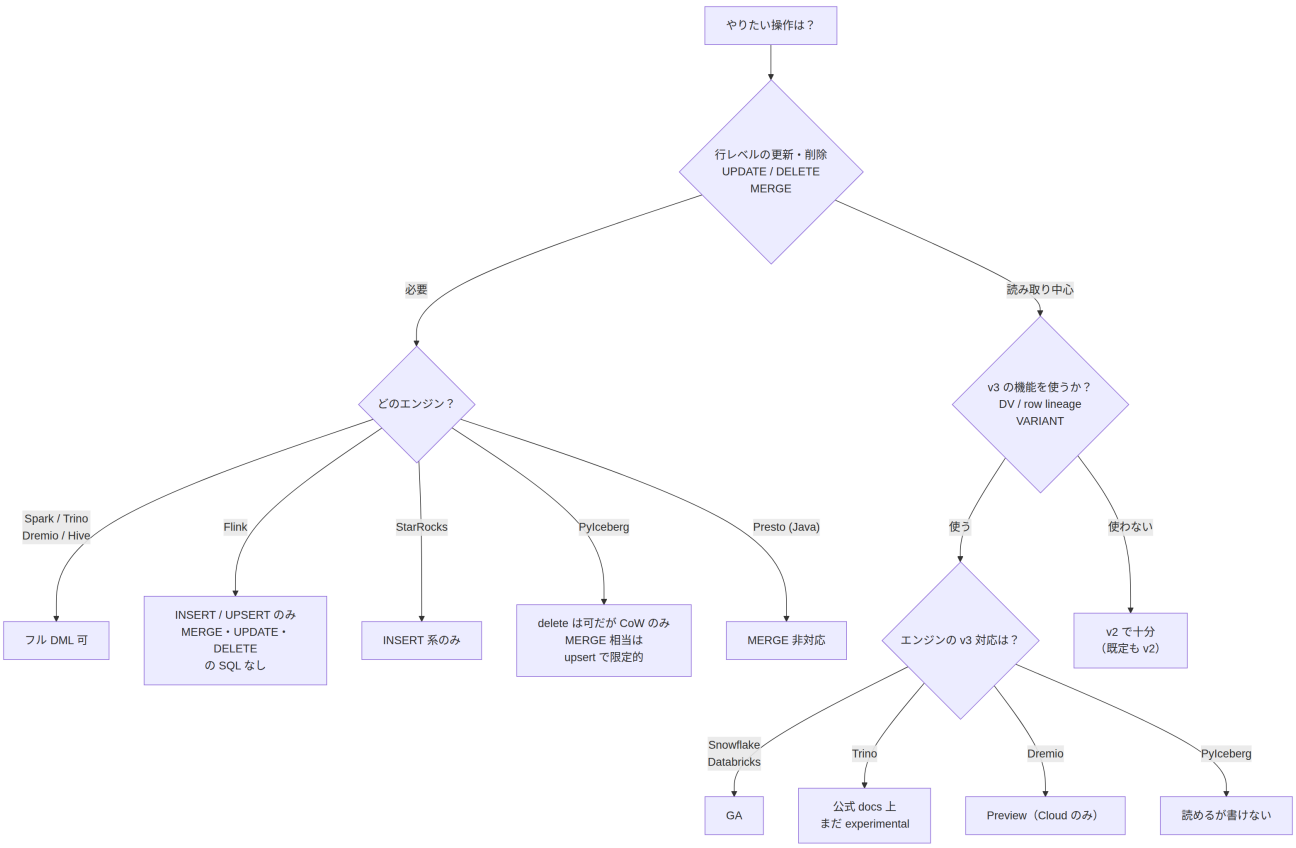
選定時に効く制約(詳細は 04):

選択肢	見落としやすい制約
S3 Tables	**CTAS 不可**(stage-create なし)、**view 非対応**。さらに**タグやブランチを 1 つ作るだけで自動スナップショット管理がテーブル全体で停止**(fail-closed)し、失敗は課金増加としてしか現れない→ 06 C-1
Databricks UC	Managed Iceberg のみ読み書き可。**Foreign Iceberg は読み取り専用で credential vending 非対応**。また **IRC 接続は一方通行**で、外部エンジンが UC に繋ぎに来ることはできても、**Databricks が Polaris など他のカタログへ繋ぎに行くことはできません**(既存カタログがある場合、UC に寄せる必要が生じます)
Snowflake Open Catalog	**新規顧客はサインアップできない**(Horizon へ誘導)。既存顧客は継続利用可
Lakekeeper	技術的な出来は良いが **pre-1.0・実質コア 2 名・公表採用事例ゼロ**。リスクを取れる場合の選択肢
Hadoop catalog	**オブジェクトストレージで安全でない**。仕様も deprecated としており v4 で削除予定。推奨代替は **JDBC catalog** → 04

> **用語**: **エグレス費用**(egress cost)は、クラウドから**データを外部へ持ち出すときにかかる通信料金**です。ストレージに置くことや同一クラウド内で読むことは安価でも、インターネットや他社クラウドへ出すと GB 単位で課金されます。複数クラウドにまたがって同じテーブルを共有する構成では、これが総コストを左右することがあります。

9.3 段階 3: エンジンで「本当にできるか」を確認する

最後に、**やりたい操作がそのエンジンで実際にできるか**を確認します。ここを飛ばすと、カタログまで決めた後で「この DML が書けない」と気づくことになります。



確認すべき代表的な落とし穴（詳細は 05）：

前提しがちなこと	実際
「v3 にすれば deletion vector が使われる」	**既定は copy-on-write のまま。**`write.delete.mode` を明示しない限り DV は使われない → 02
「PyIceberg で一通りできる」	**v3 を書けない / equality delete を読めない / MoR は CoW に落ちる / compaction が無い** → 05
「Trino は v3 対応済み」	PR はマージ済みだが **docs は experimental** とし、row-level updates/deletes/OPTIMIZE を非対応と明記
「Flink で CDC を組んで row lineage で追跡する」	**equality delete を使うエンジンでは row lineage は追跡されない** → 02
「マネージドなら運用は不要」	compaction と GC は肩代わりされるが、**パーティション設計・sort order・`write.metadata.delete-after-commit.enabled` の初期設定は自分の責任** → 06

9.4 使い方の注意

このチャートは**典型的なケースを最短で絞り込む**ためのもので、すべての要求を尽くしてはいません。とくに以下は分岐に含めていません。

- **既存資産との整合**（すでに動いている基盤、社内の運用スキル）
- **コンプライアンス要件**（データ所在地、監査要件）
- **コスト構造**（エグレス、コンピュート課金の形）

いずれも実際の選定では支配的な要因になりえます。チャートで絞り込んだ候補を、これらの観点で再評価してください。

また、**ベンダーの対応状況は数ヶ月で変わります**。GA / Preview の別やバージョン番号を判断に使う場合は、各章の出典から一次情報を再確認してください。

A 90. 用語集

調査基準日: 2026-07-17

本報告書で使う用語の一覧です。各項目に**出典**を付けています。

出典の種別:

- **仕様** --- Apache Iceberg Table Spec（生ソース: `format/spec.md`）
- **IRC 仕様** --- REST Catalog OpenAPI
- **公式ドキュメント** --- `iceberg.apache.org` 配下の各ドキュメント（生ソースは `apache/iceberg` の `docs/docs/`）
- **本報告書の整理** --- 仕様に定義された語ではなく、本報告書が説明のために用いている表現。**引用の際は注意してください**

A.1 メタデータ構造

A.1.1 snapshot（スナップショット）

ある時点のテーブルの状態。そのスナップショットに属する全データファイルの集合を表します。仕様は「The data in a snapshot is the **union of all files in its manifests**」と定義しています。

出典: 仕様 / → 01

A.1.2 metadata file（`*.metadata.json`）

テーブルの状態を記録した JSON ファイル。スキーマ、パーティション仕様、sort order、スナップショット一覧、テーブルプロパティを含みます。**カタログが保持するのは、この「現在の metadata ファイルへのポインタ」1 つだけ**です。

出典: 仕様 / → 01

A.1.3 manifest list（`snap-*.avro`）

1 スナップショットにつき 1 つ作られる Avro ファイル。そのスナップショットに属する manifest の一覧と、各 manifest のパーティション要約統計（`field_summary`）を持ちます。

出典: 仕様 / → 01

A.1.4 manifest（*-m0.avro）

データファイル（または削除ファイル）の一覧を記録した Avro ファイル。各ファイルのパーティション値と列統計を含みます。

仕様上の重要な性質が 2 つあります。**マニフェストはパーティションに紐づきません**（「Manifests can track data files with **any subset of a table and are not associated with partitions**」）。また**データ用と削除用は同一 manifest に混在できません**（削除ファイルはジョブプランニング時に先に読む必要があるため）。

出典: 仕様 / → 01

A.1.5 Puffin

統計とインデックスを格納する blob コンテナ形式。現在 2 つの blob 型が定義されています（`apache-datasketches-theta-v1`、`deletion-vector-v1`）。

出典: Puffin Spec / → 01

A.1.6 sequence number（シーケンス番号）

コミット成功ごとに単調増加する番号。削除ファイルをどのデータファイルに適用するかの判定に使われます。v2 で導入されました。

`sequence_number`（データシーケンス番号 = 内容の相対的な古さ）と `file_sequence_number`（追加時のスナップショット番号）は別物で、仕様は「**The file sequence number can't be used for pruning delete files**」と明記しています。

出典: 仕様 / → 01

A.2 テーブル仕様とバージョン

A.2.1 format version（フォーマットバージョン）

テーブルのメタデータがどの仕様に従って書かれているかを示すバージョン。**既定は 2**（1.4.0 以降）。前方互換性が壊れる変更が入るときにだけ上がります。

出典: 仕様 / 公式ドキュメント（`configuration.md`） / → 02

A.2.2 schema evolution（スキーマ進化）

列の追加・削除・改名・型変更を、既存データファイルを書き換えずに行うこと。**列は名前ではなく field ID で解決される**ため、改名しても既存ファイルはそのまま読めます。

出典: 仕様 / → 02

A.2.3 field ID

各列に割り当てられる不変の識別子。仕様は「Columns in Iceberg data files are **selected by field id**」「Renaming an existing field must change the name, **but not the field ID**」と定めています。

出典: 仕様 / → 02

A.2.4 partition spec（パーティション仕様）

どの列をどの transform（`identity` / `bucket[N]` / `truncate[W]` / `year` / `month` / `day` / `hour` / `void`）でパーティションするかの定義。

出典: 仕様 / → 02

A.2.5 partition evolution（パーティション進化）

パーティションの切り方を後から変更すること。**既存データは書き換えられず**、新旧の spec が同一テーブル内に共存します。読み取り時、各 manifest は「それを書いた時点の spec」で解釈されます。

出典: 仕様 / → 02

A.2.6 hidden partitioning

クエリにパーティション列を書かなくてよい仕組み。エンジンが論理列への述語（`ts > X`）を inclusive projection でパーティション述語（`ts_day >= day(X)`）に自動変換します。

> **注**:「hidden partitioning」は公式ドキュメントでも使われる呼称ですが、**仕様本文が定義する用語ではありません**。仕様が定めているのは inclusive projection の規則です。

出典: 公式ドキュメント / 仕様（inclusive projection） / → 01

A.2.7 inclusive projection

スキャン述語をパーティション述語に変換する規則。**ある行が元の述語を満たすなら、その行のパーティションは必ず変換後の述語を満たす**という保証を持ちます。対偶により、変換後の述語を満たさないパーティションは丸ごとスキップできます。

意図的に緩く作られており、**false positive（該当行が無いパーティションを読む）は許容、false negative（該当行があるパーティションを飛ばす）は不可**という非対称な保証です。

出典: 仕様 / → 01

A.2.8 strict projection

inclusive projection の対。**そのファイルの全行が確実に述語を満たすときだけ**マッチする、厳しい側に倒した変換。residual predicate（読んだファイル内で行ごとに再評価すべき条件）の計算に使われます。

出典: 仕様 / → 01

A.3 更新と削除

A.3.1 copy-on-write（CoW）

変更が及ぶデータファイルを丸ごと書き直す方式。書き込みは重いが、読み取り側は特別な処理が不要です。**`write.{delete,update,merge}.mode` の既定値**。

出典: 公式ドキュメント（`configuration.md`） / → 02

A.3.2 merge-on-read（MoR）

元のデータファイルは変更せず、「この行は削除された」という情報を別ファイルに追記する方式。書き込みは軽いが、読み取り時にマージのコストがかかります。

出典: 公式ドキュメント（`configuration.md`） / → 02

A.3.3 position delete file

「どのデータファイルの何行目を削除したか」を記録するファイル（v2 で導入）。**v3 テーブルへの新規追加は禁止**されました（既存のものは有効なまま）。

出典: 仕様 / → 02

A.3.4 equality delete file

「この列がこの値の行を削除する」という条件を記録するファイル（v2 で導入）。読み取り時に全データファイルと照合が必要になりえます。

注意すべき制約が 2 つあります。 unpartitioned spec で書かれた equality delete は**グローバル削除**として全データファイルに適用されます。また **v3 の row lineage は equality delete では追跡されません**。

出典: 仕様 / → 02

A.3.5 deletion vector（DV）

v3 で導入された削除の記録形式。単一データファイル内の削除位置を Roaring bitmap で表し、Puffin ファイルに格納します。

スナップショット内で 1 データファイルにつき DV は最大 1 つという不変条件が仕様で保証されており、これが v2 の position delete（際限なく積み上がりうる）との決定的な違いです。

出典: 仕様 / Puffin Spec / → 02

A.3.6 row lineage

v3 で導入された行の来歴追跡。`_row_id` と `_last_updated_sequence_number` の 2 フィールドを持ちます。**v3 では必須で、オプトインではありません。**

出典: 仕様 / → 02

A.4 トランザクションと並行制御

A.4.1 楽観的並行制御（OCC: Optimistic Concurrency Control）

ロックを取らず、「他の書き手が先に変更していないこと」をコミット時に検証する方式。競合したら、負けた側が最新の状態を土台に載せ直してリトライします。

出典: 仕様（「Writers create table metadata files **optimistically**」）/ → 01

A.4.2 atomic swap / check-and-put

コミットの実体。新しい metadata ファイルを書いたうえで、カタログに対し「現在が期待どおりなら、これに差し替えて」と要求する操作です。

> **注**: 仕様は commit の atomic 操作そのものを標準化していません（「The atomic operation used to commit metadata depends on how tables are tracked and **is not standardized by this spec**」）。**カタログ実装に依存します。**

出典: 仕様 / → 01

A.4.3 snapshot-log

「いつ、どのスナップショットが current になったか」を時刻つきで並べた時系列のログ。**時刻指定のタイムトラベルはこれを使わないと正しく答えられません**（親子系譜とは別物で、ロールバック時に両者は食い違います）。

出典: 仕様 / → 01

A.4.4 branch / tag

スナップショットを指す名前つきの参照(snapshot reference)。**tag** は動かない名札、**branch** は本流と別に書き込める枝です。`main` ブランチは常に存在し、期限切れで消えることはありません。

出典: 仕様 / → 01

A.4.5 WAP（Write-Audit-Publish）

監査用ブランチに書き込み → データ品質を検証 → 問題なければ `main` に取り込んで公開、という手順。バッチごとに「ステージング → 検証 →本番反映」のゲートを設ける運用です。

出典: 公式ドキュメント（`branching.md`） / → 01

A.5 REST Catalog

A.5.1 IRC（Iceberg REST Catalog）

カタログを HTTP/JSON の単一の OpenAPI 契約に統一する仕様。0.14.0 で導入されました。

仕様に意味のあるバージョン番号はありません（`info.version` は `0.0.1` のまま）。機能の有無は Iceberg のリリース番号と、実行時の `GET /v1/config` の `endpoints` フィールドで判断します。

出典: IRC 仕様 / 公式ドキュメント（`terms.md`） / → 03

A.5.2 prefix

IRC のパスに含まれるパスパラメータ。仕様上の定義は「An optional prefix in the path」のみです。

> **注**: マルチテナントの実現手段として使われますが、**仕様は prefix の意味論を定義しておらず、サーバ実装に委ねています**。「prefix はマルチテナント用」と断定する公式記述は確認できていません。

出典: IRC 仕様 / → 03

A.5.3 warehouse

カタログ内の格納先を指す識別子。`GET /v1/config?warehouse=` で指定します。

> **注**: 値の意味は実装依存です。Apache Polaris では**カタログ名**であり、S3 パスではありません。

出典: IRC 仕様 / 本報告書の実測 / → 03

A.5.4 credential vending

カタログが**テーブル単位・prefix 単位・短命**のストレージ認証情報をクライアントへ払い出す仕組み。エンジンに長命かつ広範な認証情報を配らずに済みます。

複数の認証情報が返る場合、クライアントは**最長 prefix のもの**を選びます（longest-prefix-wins）。

出典: IRC 仕様 / → 03

A.5.5 remote signing

クライアントがストレージへのリクエストの署名をカタログに依頼する方式。**クライアントは認証情報をそもそも保持しません**。

出典: IRC 仕様 / → 03

A.5.6 server-side scan planning

「どのファイルを読むか」の計画をカタログサーバ側で行う仕組み。クライアントが manifest を自分で読む必要がなくなります。仕様は 1.7.0、クライアント実装は 1.11.0 で入りました。

出典: IRC 仕様 / 公式リリースノート / → 03

A.6 運用

A.6.1 compaction

小さなデータファイルを結合して適切なサイズにまとめる操作（`rewrite_data_files`）。戦略は bin-pack（既定） / sort / z-order。

公式は compaction を "Optional" に分類しています。また **`expire_snapshots` を回さない限り容量は解放されません**（古いファイルがスナップショットから参照され続けるため）。

出典: 公式ドキュメント（`maintenance.md`, `spark-procedures.md`） / → 06

A.6.2 expire_snapshots

古いスナップショットを期限切れにし、参照されなくなったファイルを削除する操作。**公式分類は "Recommended"**（compaction より優先度が高い）。

branch / tag から参照されているスナップショットは削除されません。タグを 1 つ付けるだけで、そのスナップショットと依存ファイルが無期限に残ります。

出典: 公式ドキュメント（`maintenance.md`, `spark-procedures.md`） / → 06

A.6.3 orphan file

どのメタデータからも参照されていないファイル。ジョブ失敗などで発生します（データを先に書き、メタデータを後でコミットする順序のため構造的に避けられません）。

`remove_orphan_files` で削除できますが、**本報告書で最も危険と位置づけている操作**です。進行中の書き込みを誤って削除するとテーブルが壊れます。

出典: 公式ドキュメント（`maintenance.md`, `spark-procedures.md`） / → 06

A.6.4 fail-closed

失敗時に処理を止める設計。本報告書では Amazon S3 Tables の自動スナップショット管理について使っています。**タグやブランチを 1 つ作るだけで、テーブル全体の自動メンテナンスが停止します**（しかも失敗は課金増加としてしか現れません）。

出典: AWS S3 Tables ドキュメント / → 06

A.7 エコシステム

A.7.1 UniForm

Delta Lake の機能。Delta テーブルから Iceberg のメタデータを自動生成し、Iceberg クライアントがファイルを書き換えずに読めるようにします。

出典: Databricks ドキュメント / → 07

A.7.2 LSM-tree（Log-Structured Merge-tree）

書き込みを追記で受けて後からマージする木構造（LevelDB / RocksDB と同系統）。Apache Paimon が bucket ごとに採用しており、連続 upsert に向く理由です。

> **注**: Iceberg の用語ではありません。Paimon の設計を説明するために用いています。

出典: 本報告書の整理 / Paimon ドキュメント / → 07

A.7.3 エグレス費用（egress cost）

クラウドから**データを外部へ持ち出すとき**にかかる通信料金。同一クラウド内の読み取りは安価でも、他社クラウドやインターネットへ出すと GB 単位で課金されます。複数クラウドで同じテーブルを共有する構成では総コストを左右します。

出典: 本報告書の整理 / → 08

A.7.4 TLP（Top-Level Project）

Apache Software Foundation の正式プロジェクト。Incubator を卒業すると TLP になります。Apache Polaris は 2026 年 2 月に卒業しました（**公式ソース 3 つで日付が食い違っている**点は 04 を参照）。

出典: ASF / → 04

A.7.5 GA / Preview

製品の提供段階を示す**ベンダー用語**です。

> **重要**: **Apache Iceberg 公式には「v3 が GA」という宣言は存在しません。**公式ステータスは仕様冒頭の「complete and adopted by the community」までです。GA / Preview はベンダー側（Snowflake / Databricks / Dremio など）の製品提供状況を指します。両者を混同すると「v3 は GA なのに Trino では experimental」といった混乱が生じます。

出典: 仕様 / 各ベンダーのドキュメント / → 02, 99

A.8 本報告書が判定に使う語

以下は本報告書がエンジン対応状況を記述するために定めた区分です。**仕様や公式の用語ではありません。**

区分	意味
【確認】	公式ソースで対応と確認できたもの
【明示的非対応】	**公式ソースが非対応と明記している**もの
【未確認】	**ソースが見つからなかった**もの。「非対応」ではない

「マージ済み ≠ リリース済み ≠サポート対象」も同様に、本報告書が繰り返し用いる区別です。PR がマージされていても、リリースに含まれるとは限らず、公式ドキュメントがサポートを認めているとも限りません。

出典: 本報告書の整理 / → 05, 99

B 99. 調査方法論と未確認事項

この報告書を更新する人、および Iceberg 周辺で同種の調査を行う人は、まずここを読むことを勧めます。

この調査は人間と AI の協業で行いました。誤りを減らすため、同じ問いを複数回・独立に調べ、結果が食い違った点は改めて一次情報に当たって決着させています。その過程で**再現性のある罠**がいくつも見つかりました。以下はその記録です。

B.1 A. 証跡の優先順位

公式ドキュメント
  > リポジトリの実ファイル / ソースコード
    > 公式ブログ・プレスリリース
      > 技術ブログ

上位と下位が矛盾したら上位を採用し、矛盾の存在自体を記録します。

B.1.1 なぜこの順序なのか — 実例

Dremio の v3 対応で、この順序が決定的でした。

ソース	主張
プレスリリース（2026-04-06）	「ships the latest V3 spec **at GA**」「full read and write support」
docs	見出しに **“Iceberg v3 Preview”**

マーケティング文書は “GA” と書きたがります。 docs を採用して **Preview** と判定しました。AI の調査結果の一つはプレスリリースだけを根拠に GA と報告しており、これが本調査で見つかった**唯一の実質的な誤り**でした。

B.2 B. この調査で踏んだ罠

B.2.1 B-1. リリースノートは遡及更新されない

Snowflake の外部エンジンからの v3 書き込みで、複数の調査結果の間で主張が食い違いました。

- 2026-05-07 の GA リリースノート:「Writes from external engines to Snowflake-managed Iceberg v3 tables through Horizon Catalog **aren't supported yet**」
- 現行 docs:「read and write to Snowflake-managed Iceberg **v2 and v3** tables」

矛盾ではありませんでした。2026-05-26 に外部エンジン write が GA 化して制約が解消され、**リリースノートは当時の記述のまま残っていた**だけです。

> **教訓: リリースノートを「現在の状態」の根拠に使わないこと。**仕様上、後から更新されません。その時点の記述として読み、**現況は必ず現行 docs で確認する**こと。

B.2.2 B-2.「マージ済み ≠ リリース済み ≠サポート対象」

この罠は **Trino・Presto・Spark で繰り返し**現れました。

事例	実態
Trino の v3	v3 の PR は**すべてマージ済み**（#27786〜#27893）。しかし **docs は今も「Support for format version 3 is experimental」**とし、row-level updates/deletes/OPTIMIZE を非対応と宣言。**default values も#27837 がマージされているのに docs は非対応と書く**
Spark の geometry/geography	PR #17073 は 2026-07-08 マージ済み。しかし **milestone は 1.12.0 = 未リリース**。今日使えない
Presto の v3	DV + UPDATE/MERGE（#28058/#27943）はトランクにマージ済みだが、**どのリリースにも入っていない**

> **教訓: マージ済み PR のリストを「サポート済み」と読まないこと。**リリースタグとマイルストーンを確認し、最終的には docs を authoritative とすること。

B.2.3 B-3. リリースノートの要約が誤読を誘う

Spark の default values が典型例です。

1.11.0 のリリースノートに「Add schema conversion support for default values (#14407)」とあります。しかし:

- 実タイトルは **"Spark 4.0: Add schema conversion support for default values"**
- 変更は `TypeToSparkType.java` のみ = **スキーマ変換だけ**
- 同じ 1.11.0 タグのコードに `throw new UnsupportedOperationException("Cannot add column %s since setting default values in Spark is currently unsupported")` が**実在**

教訓: リリースノートの要約タイトルではなく、PR の実タイトルと変更ファイルを見ること。

B.2.4 B-4. ドキュメントサイトがバージョンに追従していない

PyIceberg の py.iceberg.apache.org には**バージョンセレクタ/バナーがなく**、どのバージョンに対応するか明示されません(= main 追従の可能性)。

そこで本調査は**リリースタグ pyiceberg-0.11.1 のソースを直接参照して照合**しました。その結果、docs だけでは分からない精度で確定できました:

- table.maintenance の実在(table/__init__.py L1129、table/maintenance.py)
- **v3 は「型定義は存在するが model_dump_json が NotImplementedError を投げる=書き込み不可」**

教訓: docs が信用できないときは、リリースタグのソースを直接読むこと。

B.2.5 B-5. 公式ソース同士が矛盾する

Flink の VARIANT:

ソース	主張
flink.md 型表(1.11.0 タグ・main 双方)	variant | | Not supported
1.11.0 リリースノート	「Support Variant to Flink 2.1 (#15265)」
1.11.0 タグの実コード	flink/v2.1/.../TypeToFlinkType.java に variant 参照 3 件を実測。テストも存在

実装は入っており、**型対応表が未更新**と判断しました。**ただし公式ソースが「Not supported」と書いている事実は残る**ため、両方を記載しています。

B.2.6 B-6. iceberg.apache.org は JS レンダリングで本文が取れない

これは**技術的な障害**ですが、対処法が確立しました。

取れないページ	代わりに取るもの
iceberg.apache.org/spec/	https://raw.githubusercontent.com/apache/iceberg/main/form (2137 行)
iceberg.apache.org/rest-catalog-spec/ (Swagger UI)	https://raw.githubusercontent.com/apache/iceberg/main/open api/rest-catalog-open-api.yaml (5822 行)
iceberg.apache.org/docs/latest/*	https://raw.githubusercontent.com/apache/iceberg/main/docs

Databricks の docs も同様です。curl で HTTP 200・約 50KB が返りますが、JS レンダリングのシェルのみで本文は含まれません。**「curl して grep したら Preview が出ない= GA」という推論は成立しません。**

教訓: 生ソース(raw.githubusercontent.com)を第一の取得先にすること。

B.2.7 B-7.「GA」はベンダー用語であって Apache の宣言ではない

Apache Iceberg 公式には「v3 が GA」という宣言が存在しません。公式ステータスは仕様冒頭の「complete and adopted by the community」までです。

「GA」は Snowflake / Databricks / Dremio といった**ベンダー側の製品提供状況**を指します。両者を混同すると、「v3 は GA なのに Trino では experimental」といった混乱が生じます。

B.2.8 B-8.「未確認」と「非対応」は別物

例	判定
Trino の v3 row-level updates	**【明示的非対応】** --- docs が「are not supported」と**明言している**
Flink の row lineage 生成	**【未確認】** --- **ソースが見つからなかった**だけ
Redshift の Iceberg 書き込み	**【未確認】** --- AWS の docs に**肯定も否定も見つからない**

「見つからなかった」を「非対応」と書くのは捏造です。本報告書では 3 値で厳密に区別しています。

B.2.9 B-9. 上流の前提が古いことがある

調査依頼の前提自体が一次情報で否定された例:

依頼中の前提	実際
`s3.path-style-access` を設定する	**PyIceberg にこのキーは存在しない**。実在するのは `s3.force-virtual-addressing`。しかも `s3.endpoint` を設定すれば自動で path-style になるので**何も設定しなくてよい**
MinIO を使う	**`minio/minio` は 2026-04-25 にリポジトリごとアーカイブ済み**
Polaris は incubating	**2026-02 に TLP 卒業済み**
Polaris の `getting-started/eclipselink/`	**撤去済み(404)**。`jdbc` に置換

> **教訓: 依頼者の前提も検証対象に含めること。**

B.2.10 B-10. 二次情報が「事実」として流通している

流通している主張	実際
「Nessie は Polaris に統合されて廃止」	**2024 年の Dremio CMO 発言に基づく報道**。2026 年 7 月時点で Nessie は **archived ではなくリリースも継続中**（0.108.2、2026-07-17）。ただし「活発」も過大評価で、**人間のコミットは年間 245 件・実質 1 人**（B-12）。正確には**「retire はされていないが、実質は少数保守モード」**
「`iceberg-rest-fixture` は公式が本番使用を禁止している」	**README にそのような明示的免責文は存在しない（未確認）**。よく引用される警告は RCK 側のもの。ただし命名・ソース配置（`testFixtures`）・既定値（インメモリ SQLite）が**すべてテスト用途を示す**のは事実
「Polaris が MinIO をやめたのは AGPL 化のため」	**Polaris の一次情報（PR #3482 / 公式 docs）にライセンスを理由とする記述はない**。公式が一貫して挙げる理由は「**maintenance mode / セキュリティ修正が来ない**」。しかも **MinIO のライセンスは AGPLv3 のままで変わっていない**
「Hive は死んだ」	**Attic に入っていない**（実際に attic.apache.org を取得して確認）。4.2.0（2025-11-23）リリース済み、4.3.0 の議論で Iceberg v3 が最優先スコープ

> **教訓: 「みんなが言っていること」ほど一次情報に当たること。**

B.2.11 B-11. 実際に動かすと、読解だけでは分からない問題が出る

検証用のラボ環境（Apache Polaris + PyIceberg）を構築・実行した際に、ドキュメントの読解では気づけなかった問題が出ました。

症状: PyIceberg でコミットすると

```
pyiceberg.exceptions.CommitStateUnknownException:
  StsException: (Service: Sts, Status Code: 400, Request ID: null)
```

原因: Polaris のカタログ storage config に `roleArn` を設定していたため、Polaris が credential vending のために **STS AssumeRole** を呼びに行った。RustFS も MinIO も **STS に対応していない**。

なぜ読んでも分からなかったか: Polaris の公式ガイドは `roleArn` を**書いていない**だけで、「S3 互換ストレージでは書いてはいけない」とは**どこにも書いていません**。「AWS の例に `roleArn` があるから付けておこう」という自然な発想が罠になります。**公式サンプルの「無いもの」に意味があるケース**は、差分を取らないと気づけません。

副次的な発見: このエラーメッセージは誤解を招きます。`CommitStateUnknownException` は仕様上「コミットが成功したか失敗したか分からない」という**最も危険な状態**を意味しますが（03-rest-catalog.md 参照）、実態は単なる設定ミスでした。仕様上の意味と実装のエラー分類が一致しないことがあります。

> **教訓: 動かせるものは動かすこと。**「公式サンプルをコピーして少し足す」は、その「少し」が事故になります。足したものは疑ってください。

あわせて、報告書の主張 11 件を実測で確認しました（既定 v2、CoW 既定、V3 書き込み不可、MoR フォールバック、`s3.path-style-access` 不在、field ID 不変、credential vending の prefix スコープ、409 の再現、`prefix` が overrides で配られること、など）。読解に基づく主張と実測で確認した主張は、報告書の中で区別しています。

B.2.12 B-12. 実測値は、指標の定義を確認しないと誤った結論を導く

初版はこの問題を含んでおり、公開前のファクトチェックで判明しました。実測した数値であっても、それだけでは正しい結論を導けません。

初版は「Nessie は直近 52 週 **1,630 コミット**で活発に開発継続中」と書いていました。数字自体は正確でした。**内訳を見なかったのが誤りです。**

プロジェクト	総コミット（365 日）	ボット	**人間**
Iceberg	1,696	309（18.2%、全て dependabot）	1,387
Paimon	2,278	13（0.6%）	**2,265**
Hudi	1,141	4（0.4%）	1,137
Delta Lake	1,203	**0**	1,203
Nessie	1,652	**1,407（85.2%、renovate）**	**245**
Polaris	2,100	744（35.4%、renovate）	1,356

ボット比率が 0.4% 〜 85% と 2 桁違うため、生のコミット数を横並びにした表は指標として成立していませんでした。具体的に何が壊れたか:

1. **Nessie の「活発」は幻。**人間のコミットは 245 件で、しかも実質 1 人（`snazy` が 212）。「1,630 コミット」で Hudi（1,141）や Delta（1,203）より上位に見せていたのは、**renovate の依存更新を数えていただけ**です。
2. **Polaris と Iceberg の順位が逆転。**生では Polaris 2,100 > Iceberg 1,696 ですが、人間では 1,356 < 1,387 です。
3. **Paimon の優位を過小評価していた。**生では Iceberg の 1.34 倍ですが、人間では **1.63 倍**。Iceberg の 18% が dependabot だからです。

さらに指標定義の混在もありました。「Iceberg 267 / Paimon 164」は**直近 365 日のユニーク作者**、「Polaris 156 / Nessie 79」は**全期間 contributor** で、同じ表に並べていました。しかも「Iceberg は contributor 数で突出」という結論は**直近 365 日ベースでのみ成立**し、全期間で測ると Iceberg 407 / Delta 391 / Hudi 382 / Paimon 363 で**ほぼ横並び**です。

> **教訓 1: 「コミット数が多い＝活発」は、ボットを除外しない限り成立しません。教訓 2: 複数プロジェクトを比較する表では、全セルが同じ指標定義であることを確認してください。教訓 3: 実測は「測った」で終わりではありません。何を測ったかを疑ってください。**数字はそれ自体では嘘をつきませんが、**内訳を見ないと嘘の結論を支えます**。

B.2.13 B-13. 「未採択」と「実装が受け付けない」は別物

初版は「**2026 年 7 月時点で `format-version=4` は設定できません**」と書いていました。**誤りです。**

- **仕様**:「Version 4 is under active development and **has not been formally adopted**」← これは正しい

- **Iceberg Java 1.10.0+（2025-09-11〜）**: `SUPPORTED_TABLE_FORMAT_VERSION = 4`。バリデーションは `formatVersion <= SUPPORTED_TABLE_FORMAT_VERSION` なので **v4 は通過する**
- **PyIceberg 0.11.1**: `TableVersion = Literal[1, 2, 3]` で**受理しない**

リリースタグ実測: 1.8.0 → 3、1.9.0 → 3、**1.10.0 → 4**、1.11.0 → 4。

「仕様が未採択」から「実装が拒否する」を推論したのが誤りでした。実際には Java は既に v4 テーブルを作れます。しかも `MIN_FORMAT_VERSION_PARQUET_MANIFESTS = 4` / `MIN_FORMAT_VERSION_OPTIONAL_LOCATION = 4` と、v4 機能のゲートがコードに入っています。

> **教訓: 仕様と実装は別のものです。**「仕様がこう書いてあるから実装もこうだろう」は推論であって事実ではありません。**実装の主張をするなら実装を読んでください。**この誤りは、仕様書だけを読んで実装を確認しなかったことに起因します。

実務上の含意も逆になります。「作れないから安全」ではなく、**「作れてしまうので、仕様が変わるリスクを自分で避ける必要がある」**のです。

B.2.14 B-14. 実測できるものは実測する

プロジェクトの健全性は、印象ではなく GitHub API で測れます。

XTable: 最新リリース 2025-06-04（13 ヶ月前）、直近 90 日 10 コミット、2 年 5 ヶ月 incubating のまま
Iceberg: 直近 365 日 1,696 コミット（うち人間 1,387）、ユニーク作者 270 名
Paimon: 直近 365 日 2,278 コミット（うち人間 2,265、人間コミットで Iceberg の 1.63 倍）

「XTable は停滞している」「Paimon は Iceberg に吸収されていない」といった判断は、**この実測値が支持します**。

B.3 C. 調査結果の食い違いをどう扱ったか

本調査では、独立に行った複数の調査の結果が 4 つの論点で食い違いました。それぞれ改めて一次情報に当たって決着させた結果:

論点	結果
Databricks の v3 は GA か Preview か	**どちらも正しかった**（2026-04-09 Preview → 2026-05-28 GA。見ていた時点が違った）
Snowflake の外部エンジン v3 書き込み	**どちらも正しかった**（5/7 時点の制約が 5/26 に解消。リリースノートが古い）
Dremio の v3 は GA か Preview か	**一方が誤り**（プレスリリースを採り、docs と突き合わせていなかった）
Snowflake Open Catalog は新規顧客にクローズか	**主張は公式 docs で裏取りできた**（ただし課金開始時期のみ決着つかず）

食い違い 4 件中 3 件が「時系列の異なる時点を切り取った」ことによるものでした。

教訓: 食い違いを見たら、まず「両方が別の時点で正しい可能性」を疑うこと。日付を押さえれば大半は解決します。

B.3.1 調査プロセス上の事故についての注記

AI が生成した調査結果の一つに、**事実を捏造した箇所**がありました(「Redshift の書き込みが 2026-04-23 に GA」「Athena は v2 のみ」の 2 件)。この捏造はその調査の途中で検出・撤回され、最終結果には持ち込まれていませんでした。

その後の一次情報による検証でも、**この 2 件が本報告書に混入している形跡は見つかりませんでした**。4 論点の範囲でも、明確な捏造は検出されていません。

この事故を記録に残すのは、それがまさにこの調査が防ごうとしていた失敗モードだからです。LLM ベースの調査では、以下が構造的に必要だと考えます:

1. **一次情報の URL を必ず添えさせる**(検証可能にする)
2. **同じ問いを複数回・独立に調べてクロスチェックし、食い違いを検出する**
3. **食い違った点は改めて一次情報に当たって潰す**
4. **「未確認」を書くことを明示的に許可・推奨する**(無理に埋めさせない)

「見つからなかった」ことも記録すべき結果です。空欄を埋めようとすると捏造が生じます。

B.4 D. 未確認事項の一覧

「未確認」は「非対応」ではありません。以下は前者、つまり「調べたが確証が得られなかった」項目です。

B.4.1 仕様(02)

- v4 の Adaptive Metadata Trees、column families、single-file commits --- **ブログ・二次情報のみ。公式マイルストーンには載っていない**(載っているのは相対パスと列統計改善の 2 提案のみ)
- v4 のタイムライン(「2027 年前後」等) --- 公式には裏取り不可

B.4.2 REST Catalog(03)

- Hive Metastore の具体的批判(Thrift/JVM 必須、ロッキング問題) --- Apache の一次情報には見つからず。ベンダーブログ由来
- Hadoop catalog の限界を REST catalog の動機として語る公式記述 --- 未発見
- `prefix` = マルチテナント用という明示 --- 構造から導かれる強い推測に留まる
- `prefix` に `/` を含められるか --- 仕様が明示せず
- `Idempotency-Key` が `CommitStateUnknownException` 対策として導入されたという因果 --- 明示的な記述は未発見
- credential vending の「カタログ ACL バイパス防止」動機 --- 公式がこの言葉で説明する箇所は未発見
- function 系エンドポイントと SQL UDF spec(#14117)の関係
- iceberg-rust の scan planning 対応状況

B.4.3 カタログ実装（04）

- **Lakekeeper の REST 仕様準拠バージョンとエンドポイント単位の適合表** --- 公式 docs にもリポジトリにも記載なし
- **Lakekeeper の本番採用事例** --- 公式情報源にゼロ
- Polaris 1.5 の Generic Table vending 拡張 --- Snowflake ブログと公式 docs が食い違う
- **S3 Tables ネイティブエンドポイントの vending 有無** --- AWS 公式に記述なし、コミュニティでは失敗報告あり
- Glue REST の view / RenameTable 対応、vending ヘッダ機構
- R2 Data Catalog の view 対応と GA 時期
- Gravitino の remote signing 対応と詳細 RBAC モデル
- **Hive Metastore の非推奨化** --- **Iceberg 公式に非推奨宣言は存在しない**
- Nessie の長期的な将来 --- 2024 年の Dremio 発言と 2026 年の実態が矛盾
- Polaris の TLP 卒業日（Incubator 2026-02-15 vs 公式ブログ 2026-02-19）
- `apache/polaris:latest` の実体バージョン、`iceberg-rest-fixture` の `latest` と `1.10.1` の異同
- Nessie の公式 docker compose 例

B.4.4 エンジン（05）

- **Flink の row lineage 生成（通常書き込みパス）/ DV 読み取り / multi-arg transforms / default values 書き込み** --- 一次情報を探した限り**本当に見つからなかった**
- Spark 3.5 での row lineage compaction 保存（PR タイトルが “Spark 4.0” 限定）
- Trino / Presto の CoW vs MoR 設定
- **Redshift の Iceberg 書き込み可否** --- **本調査で最大の真の未知**
- Databricks の新規 managed テーブルの既定 format version
- Snowflake の externally-managed テーブルにおける MERGE の制限
- Dremio の並行ライタ / 外部エンジンとの競合セマンティクス
- Databricks の foreign Iceberg credential vending（**ブログは GA、REST API docs は非対応。未解決**）

B.4.5 運用（06）

- **v3 の DV に対する専用 compaction procedure** --- `spark-procedures.md` に見つからず
- **公式なパーティションあたり推奨データ量 / パーティション数上限/ manifest エントリ数 / スナップショット保持推奨値** --- **いずれも見つからなかった**（一般に流布する目安に公式の裏付けなし）
- 各エンジンがどのバージョンで remote scan planning を実際に使うか
- S3 Tables / Databricks の `rewrite_manifests` 相当の自動操作--- どちらにも見つからず
- Databricks Predictive Optimization の VACUUM が Iceberg テーブルでどのプロパティを尊重するか

B.4.6 エコシステム（07）

- Paimon / Iceberg の contributor 所属組織の分布（ベンダー集中度）
- Paimon vs Iceberg の具体的レイテンシ数値 --- 一次情報での裏取り不可
- Onehouse が指摘する Databricks の個別制限事項の現時点での正確性（同社は利害関係者）
- UniForm の Hudi 対応の現況

- Iceberg 1.11.0 の機能詳細（VECTOR 型等の有無）

B.4.7 PyIceberg / ラボ環境

- RustFS の非対応 S3 API 一覧、公式の production readiness 宣言
- SeaweedFS の Console UI 有無・単一コンテナ起動可否
- MinIO object-browser の Console 機能削除の具体的コミット（`github.com/minio/object-browser` が 404）
- `polaris.bootstrap.credentials` が公式 Configuration Reference の表に未掲載（README/compose では確認済み）

B.5 E. この報告書を更新するときのチェックリスト

- □ **生ソース（raw.githubusercontent.com）から取得したか？** 公式サイトは JS レンダリングで本文が取れない
- □ **リリースノートを「現況」の根拠に使っていないか？** 遡及更新されない
- □ **PR の実タイトルと変更ファイルを見たか？** リリースノートの要約は誤読を誘う
- □ **マイルストーンを確認したか？** マージ済み ≠ リリース済み
- □ **docs とブログ/プレスリリースが食い違っていないか？** docs が authoritative
- □ **「未確認」を「非対応」と書いていないか？**
- □ **数値目安に出典があるか？** なければ「公式な推奨値は見つからなかった」と書く
- □ **「GA」が Apache の宣言かベンダーの宣言か区別したか？**
- □ **健全性の主張を GitHub API で実測したか？**
- □ **実測値の内訳を見たか？** コミット数ならボット比率、比較表なら全セルが同じ指標定義か
- □ **仕様の記述から実装の挙動を推論していないか？** 実装の主張には実装の証拠を
- □ **動かせる主張は動かして確かめたか？** 読んだだけの主張と区別しているか
- □ **公式サンプルに「足したもの」を疑ったか？** 公式が書いていないことに意味がある場合がある
- □ **利害関係者（競合ベンダー）の主張であることを明示したか？**

C ファクトチェック報告書

対象: 本リポジトリの調査報告書（README.md + docs/*.md、3,357 行、106 URL） **実施日**: 2026-07-17 **手法**: 全 URL の一括取得（Puppeteer） + 独立した 3 系統の主張検証（一次情報ベース） + GitHub API / PyPI API による実測のやり直し **方針**: **報告書の主張を追認せず、反証を探す立場**で検証

C.1 サマリ

判定	件数
検証済み（一次情報と一致）	大多数（デフォルト値 11 件、v3 リリース経緯、ベンダー GA/Preview 判定 4 件など）
不正確 --- 修正済み	**2 件（うち 1 件は結論に影響）**
誤解を招く --- 修正済み	**4 件**
リンク切れ --- 修正済み	2 件
陳腐化 --- 更新済み	3 件

結論: 報告書の大枠の論旨は維持されましたが、2 つの実質的な誤りが見つかり修正しました。いずれも一次情報には当たっていたものの、記述の解釈が不十分だったことによるものです。

C.2 重大な誤り（修正済み）

C.2.1 1.「format-version=4 は設定できない」— 誤り

初版の記述:

> 2026 年 7 月時点で `format-version=4` は設定できません。

実際（`TableMetadata.java` をリリースタグごとに実測）:

タグ	`SUPPORTED_TABLE_FORMAT_VERSION`
apache-iceberg-1.8.0	3
apache-iceberg-1.9.0	3
apache-iceberg-1.10.0（2025-09-11）	**4**

タグ	SUPPORTED_TABLE_FORMAT_VERSION
apache-iceberg-1.11.0	4

バリデーションは formatVersion <= SUPPORTED_TABLE_FORMAT_VERSION なので、**Iceberg Java 1.10.0 以降 format-version=4 は通過します**。さらに main には v4 機能のゲートが存在します:

```
static final int MIN_FORMAT_VERSION_PARQUET_MANIFESTS = 4;
static final int MIN_FORMAT_VERSION_OPTIONAL_LOCATION = 4;
```

一方 **PyIceberg 0.11.1 は `TableVersion = Literal[1, 2, 3]` で受理しません**。

誤りの構造: 仕様の「未採択」という記述から、実装も同様に拒否するという**推論**が入っていた。仕様と実装は別のものであり、実装の挙動を述べるには実装側の証拠が必要になる。

実務上の含意も逆転します:「作れないから安全」ではなく、**「作れてしまうので、仕様変更のリスクを自分で避ける必要がある」**。

修正: 02-table-spec.md を全面改稿。README・07 の波及箇所も修正。方法論に B-13 として記録。

C.2.2 2. 健全性シグナルのボット混入 — 結論に影響

初版の記述:

> Nessie は直近 52 週 1,630 コミット、contributor 79 名で**活発に開発継続中**

実際（GitHub API で内訳を実測）:

プロジェクト	総コミット(365 日)	ボット	**人間**
Iceberg	1,696	309（18.2%、dependabot）	1,387
Paimon	2,278	13（0.6%）	**2,265**
Hudi	1,141	4（0.4%）	1,137
Delta Lake	1,203	**0**	1,203
Nessie	1,652	**1,407（85.2%、renovate）**	**245**
Polaris	2,100	744（35.4%、renovate）	1,356

ボット比率が 0.4%〜85% と 2 桁違うため、生コミット数の横並び比較が成立していませんでした。

壊れていた結論:

1. **Nessie の「活発」は幻**。人間コミット 245 件、しかも実質 1 人（snazy が 212、残り全員で 33）。ユニーク作者 22 人（Iceberg 270 の 1/12）
2. **Polaris と Iceberg の順位が逆転**。生 2,100 > 1,696 だが、人間 1,356 < 1,387
3. **Paimon の優位を過小評価**。生 1.34 倍 → 人間 **1.63 倍**

加えて指標定義の混在:「Iceberg 267 / Paimon 164」は直近 365 日のユニーク作者、「Polaris 156 / Nessie 79」は全期間 contributor で、同じ表に並べていた。しかも「Iceberg は contributor 数で突出」は**直近 365 日でのみ成立**し、全期間では Iceberg 407 / Delta 391 / Hudi 382 / Paimon 363 と**ほぼ横並び**。

修正: 07-ecosystem.md の表にボット除外値を併記。04 の Nessie 評価を「活発」→「retire はされていないが実質は少数保守モード」に修正。方法論に B-12 として記録。

C.3 誤解を招く記述（修正済み）

#	初版	実際	対応
3	Polaris 卒業日の食い違いは「2026-02-15 vs 02-19 の 2 つ」。incubator ページは「projected 扱いで stale」	**公式ソース 3 つが 3 つとも違う**（ブログ 02-19 / ステータスページ 02-15 / プロジェクト一覧 **02-18**）。後者 2 つは同一ドメイン内の自己矛盾。**「projected」の語はページ全文に 0 件**で、確定形で書かれている	3 つ巴として記載、日付を断定しない方針に。「projected」の記述を削除
4	Iceberg 1.11.0 = **2026-05-20**	**2026-05-19**（公式サイト「released on May 19, 2026」、git tag の tagger date）。05-20 は GitHub 上でリリースノートを公開した操作時刻に過ぎない	2026-05-19 に統一
5	v4 節に相対パスと `content_stats` が書かれている	v4 概要節と Appendix E が挙げるのは**相対パスのみ**。`content_stats` は別の「Content Stats」節とマニフェストスキーマに記述	記載場所を訂正
6	Open Catalog の「Billing will begin in the first half of 2026」を引用	記述は実在するが、**現在 7 月で H1 は経過済み。docs が更新されていない**（同ページは今も “free to use” とも述べる）	「決着つかず」として注記済み（初版から記載あり）

C.4 リンク切れ（修正済み）

106 URL を一括取得した結果:

ステータス	件数
200	95
304（キャッシュ）	1
403（ボット遮断）	2
404	**7**
タイムアウト	1

404 のうち 5 件は本報告書の URL 抽出時のバッククォート混入（Markdown のコードスパン内 URL）で、実際のリンクは生きていました（再テストで全て 200）。

本物の問題は 2 件:

URL	問題	修正
docs.aws.amazon.com/.../API_s3tables_PutTableMaintenanceConfiguration.html	**404**（API 更新によりページ名変更）	修正済み。**正しくは API_s3Buckets_...** GetTableMaintenanceJobStatus のリンクも追加
paimon.apache.org/docs/master/iceberg/overview/	**404**（docs/master/ は廃止）	/docs/1.4/ なら 200 バージョン固定 URL に変更

その他:

- prestodb.io（403）：適切な UA なら 200。ボット遮断であり生きている
- bigdatawire.com（403 + リダイレクト）：**ドメインが hpcwire.com/bigdatawire/ に移転**。移転先に更新済み（ニュースサイトのため 403 は継続）
- oreilly conferences PDF: Puppeteer ではタイムアウトしたが curl では 200（低速なだけ）

C.5 陳腐化（更新済み）

項目	初版	実測（2026-07-17）
Nessie 最新版	0.108.1（2026-06-24）	**0.108.2（2026-07-17、検証当日にリリース）**
RustFS 最新版	1.0.0-beta.10-preview.4（2026-07-16）	**1.0.0-beta.10（2026-07-17）**。preview.5 と beta.10 が後続。**数時間おきにリリースされており、特定版の記載は公開時点で必ず古くなる**
Hudi「最新リリース」	1.2.0（2026-05-23）	正しいが、**その後 0.14.2（2026-06-08）が公開**。published_at 順では 0.14.2 が最新（旧系列の保守リリース）

C.6 検証済み（一次情報と一致）

以下は独立した検証で**一次情報と逐語レベルで一致**しました。

C.6.1 仕様

- 「Version 4 is under active development and has not been formally adopted.」が `format/spec.md` に現存
- 「Versions 1, 2 and 3 … complete and adopted by the community」も同様
- **V4 マイルストーン #58 は Open 2 / Closed 0、期日なし**（#13153 Column Stats Improvements、#13141 Relative Path Support） --- GitHub API で確認、件数の変化なし
- 「File System Tables は v4 で削除予定」

C.6.2 デフォルト値（`configuration.md` の 11 項目すべて）

`format-version`=2 / `write.{delete,update,merge}.mode`=copy-on-write / `write.target-file-size-bytes`=536870912 / `write.delete.target-file-size-bytes`=67108864 / `write.parquet.compression-codec`=zstd / `write.metadata.metrics.default`=truncate(16) / `max-inferred-column-defaults`=100 / `delete-after-commit.enabled`=false / `previous-versions-max`=100 / `history.expire.max-snapshot-age-ms`=432000000 / `commit.retry.num-retries`=4

C.6.3 v3 のリリース経緯

1.8.0 / 1.9.0 / 1.10.0 / 1.11.0 の各項目が `releases.md` の記述と一致

C.6.4 「Apache は v3 を GA と宣言していない」

リポジトリを clone し `format/`・`site/docs/`（全ブログ記事含む）・`docs/docs/` を全文検索。**“generally available” / “general availability” / “GA” のヒット 0 件**。公式表現は一貫して “complete and adopted by the community”。

> **補足を追加すべき点**: ASF はそもそも “GA” をリリース区分に用いない慣行があるため、「GA 宣言がない = v3 が未成熟」と読ませる論法には注意が必要です。

C.6.5 ベンダーの GA/Preview 判定（4 件すべて）

- **Databricks v3 = GA**: 2026-04-09 Preview → **2026-05-28 GA**。GA ブログに「**Iceberg v3 (GA)**」の逐語あり。docs に Preview バナーなし。非対応リスト（defaults / unknown 型 / ns timestamp / multi-arg transforms）も逐語一致
- **Snowflake v3 = GA + 外部エンジン write 対応**: **2026-05-26 のリリースノートが実在**し「read and write to Snowflake-managed Iceberg **v2 and v3** tables」と明記。5/7 の制約記述は当時正しく、5/26 に解消
- **Dremio v3 = Preview**: docs に **“Iceberg v3 Preview”** の見出しが現存。「v2 remains the default format version in Dremio」も逐語一致。Software 側 docs に v3 の記述なし
- **Open Catalog は新規顧客にクローズ**: 逐語 4 点すべて存在

C.6.6 プロジェクトの状態

- XTable はまだ incubating（2024-02-11 incubation 入り、卒業記載なし）
- Nessie は `archived: false`
- **Hive は Attic に入っていない**（`attic.apache.org/projects.html` 全文検索でヒットは Beehive / HiveMind のみ）。Hive 4.2.0 = 2025-11-23 も公式表記と一致
- **minio/minio は 2026-04-25 アーカイブ**（GraphQL `archivedAt` で確認）、**ライセンスは AGPLv3 のまま**> **注**: REST API の `archived_at` は null を返すため、**GraphQL でなければアーカイブ日時を取得できません。**初版が示した検証手順は不完全でした
- PyIceberg 0.11.1 = 2026-03-03（PyPI `upload_time_iso_8601`）
- iceberg-python#1551 / iceberg#14638 / prestodb/presto#27198 すべて Open

C.6.7 健全性の数値（ボット問題を除く）

実測とのずれは全て +1〜+20 程度で、測定日差の増分として整合。桁も順位も動かず。

C.7 方法論への反映

この検証で見つかった 2 つの失敗パターンを、99-methodology.md に追加しました。

- **B-12: 実測値も、指標の定義を疑わないと嘘になる** ---「測った」で終わらせず、内訳を見る
- **B-13:「未採択」と「実装が受け付けない」は別物** --- 仕様から実装の挙動を推論しない

チェックリストにも追加:

- □ 実測値の内訳を見たか？（ボット比率、指標定義の統一）
- □ 仕様の記述から実装の挙動を推論していないか？

C.8 この検証の限界

- **エンジン対応状況（Spark / Flink / Trino / Presto）の個別主張は再検証していません。**初版の調査時に一次情報で確認済みですが、今回のファクトチェックの対象外です
- **Onehouse（競合ベンダー）が指摘する Databricks の制限事項**は独立検証していません
- **URL の死活は確認しましたが、106 件すべての内容一致（D-CONTENT）は検証していません。**高優先の主張に絞っています
- 検証は 2026-07-17 時点のスナップショットです