

Package ‘icebergr’

October 12, 2026

Type Package

Title Read and Write 'Apache Iceberg' Tables

Version 0.2.0

Description A native client for 'Apache Iceberg', the open table format used by 'Snowflake', 'Databricks', 'BigQuery', 'AWS' and 'Dremio'. Rather than reaching 'Iceberg' through a query engine such as 'DuckDB', this package talks to it directly: it connects to REST and 'AWS Glue' catalogs, lists namespaces and tables, reads the schema and partition specification of a table, scans data with predicates and projections pushed down, travels back through snapshot history, and appends new data. 'Apache Arrow' is the interchange layer throughout, so scan results arrive in R without a serialisation round trip. Built on 'iceberg-rust', the Apache-governed 'Rust' implementation, via 'extendr'.
Supports table spec versions 1 and 2; see the 'README' for the full matrix of supported and unsupported features. This is a community package, not affiliated with or endorsed by The Apache Software Foundation; 'Apache', 'Apache Iceberg' and 'Iceberg' are trademarks of The Apache Software Foundation.

License GPL (>= 3)

URL <https://pursuitofdatascience.github.io/icebergr/>,
<https://github.com/PursuitOfDataScience/icebergr>

BugReports <https://github.com/PursuitOfDataScience/icebergr/issues>

Encoding UTF-8

Language en-GB

Depends R (>= 4.2)

Imports nanoarrow (>= 0.4.0), rlang (>= 1.1.0), tibble

Suggests bit64, dplyr, knitr, rmarkdown, testthat (>= 3.1.7), vctrs,
withr (>= 2.3.0)

Config/testthat/edition 3

Config/Needs/website rmarkdown

VignetteBuilder knitr

SystemRequirements Cargo (Rust's package manager), rustc >= 1.88, xz

NeedsCompilation yes

Config/roxygen2/version 8.1.0

Author Youzhi Yu [aut, cre],
The Apache Software Foundation [cph] (iceberg-rust, bundled under
Apache License 2.0)

Maintainer Youzhi Yu <yuyouzhi666@icloud.com>

Repository CRAN

Date/Publication 2026-10-12 07:20:02 UTC

Contents

icebergr_append	2
icebergr_catalog	4
icebergr_collect	6
icebergr_create_namespace	7
icebergr_create_table	8
icebergr_example_table	9
icebergr_list_namespaces	10
icebergr_list_tables	10
icebergr_partitions	11
icebergr_properties	12
icebergr_register_table	12
icebergr_reload	14
icebergr_scan	15
icebergr_scan_plan	17
icebergr_schema	18
icebergr_snapshots	19
icebergr_spec_support	20
icebergr_table	21
icebergr_table_exists	22
Index	23

icebergr_append	<i>Append rows to an Iceberg table</i>
-----------------	--

Description

Writes data as one or more new Parquet data files and commits a new snapshot. Nothing already in the table is rewritten or removed.

Usage

```
icebergr_append(
  tbl,
  data,
  compression = c("zstd", "snappy", "gzip", "lz4", "uncompressed"),
  properties = NULL
)
```

Arguments

<code>tbl</code>	An <code>icebergr_table</code> from <code>icebergr_table()</code> .
<code>data</code>	A data frame, or anything <code>nanoarrow::as_nanoarrow_array_stream()</code> accepts, such as an Arrow Table.
<code>compression</code>	Parquet compression: "zstd" (the default), "snappy", "gzip", "lz4" or "uncompressed".
<code>properties</code>	Optional named character vector recorded in the new snapshot's summary, for provenance. Do not put credentials here: snapshot summaries are stored in table metadata and are readable by anyone who can read the table. The keys Iceberg writes into a summary itself, such as "operation" and "added-records", are refused, since a value for one would corrupt the record of what the commit did.

Details

Columns are matched to the table by *name*, not position, so column order in data does not matter. Types are cast where they differ from the table's, and a column the table does not have is an error rather than being dropped silently.

Appending zero rows is a no-op: it warns, and returns the table unchanged rather than committing an empty snapshot that records that nothing happened.

The table must be unpartitioned. An append to a partitioned table would have to compute a partition value for every row, which this version does not do, so it is refused before any data is written rather than failing at the commit with files already left in the warehouse. Partitioned tables can still be read; see `icebergr_partitions()` and `icebergr_spec_support()`.

A table registered with `icebergr_register_table()` must also have been registered from a metadata file named the way Iceberg names them, `<version>-<uuid>.metadata.json`, inside the table's own metadata directory, because a memory or glue catalog writes the next one beside it and derives its name from that one. Every engine writes files like that; a renamed, moved or hand-made one reads fine and is refused here, again before anything is written. A REST catalog's server chooses the next location itself, so its tables are not held to this.

This is an append. Row-level deletes, overwrites and MERGE are not supported; see `icebergr_spec_support()`.

Value

An updated `icebergr_table` handle that sees the new snapshot. The handle passed in is unchanged, so reassign it: `tbl <- icebergr_append(tbl, x)`.

Examples

```
warehouse <- tempfile("warehouse")
dir.create(warehouse)
catalog <- icebergr_catalog("memory", warehouse = warehouse)
icebergr_create_namespace(catalog, "db")

events <- data.frame(id = 1:3, amount = c(1.5, 2.5, 3.5))
tbl <- icebergr_create_table(catalog, "db.events", events)
tbl <- icebergr_append(tbl, events)
icebergr_collect(tbl)
```

icebergr_catalog	<i>Connect to an Iceberg catalog</i>
------------------	--------------------------------------

Description

Builds a catalog handle. For type = "rest" and type = "glue" this does **not** contact the server: iceberg-rust opens the connection lazily, on the first operation that needs it. So a mistyped uri, an unreachable host or a missing credential all return a handle here and fail later, at [icebergr_list_namespaces\(\)](#) or [icebergr_table\(\)](#), which can look like a fault in those functions rather than in the connection.

Usage

```
icebergr_catalog(
  type = c("rest", "memory", "glue"),
  uri = NULL,
  warehouse = NULL,
  ...,
  storage = c("auto", "local", "s3"),
  name = "icebergr"
)
```

Arguments

type	Catalog type. "rest" for an Iceberg REST catalog, "memory" for an in-process catalog over a local warehouse directory, "glue" for AWS Glue. There is deliberately no "hadoop" option: iceberg-rust does not implement a Hadoop or filesystem catalog. Use type = "memory" with warehouse for a table on local disk.
uri	Catalog URI. Required for type = "rest", ignored otherwise.
warehouse	Warehouse location. A directory for type = "memory"; for REST catalogs, the warehouse name or location the server expects.
...	Further catalog properties, passed through to iceberg-rust as name-value pairs. Use this for non-secret configuration such as "s3.path-style-access" or "prefix". TRUE and FALSE are sent as "true" and "false", and a number is written out in full rather than in scientific notation. A property iceberg-rust does not know is ignored without a word, so check the spelling against its documentation.

storage	Storage backend. "auto" infers it from warehouse, "local" forces the local filesystem, "s3" forces object storage. S3 requires the package to have been compiled with the s3 Cargo feature.
name	A label for the connection, used in error messages.

Details

To find out straight away, ask the catalog something: `icebergr_list_namespaces(catalog)` is the cheapest round trip.

`type = "memory"` is the exception, and is checked here: its warehouse has to be an existing directory, because there is no server to ask later.

Value

An `icebergr_catalog` object.

Credentials

Credentials are read from environment variables, never from arguments:

`ICEBERGR_REST_TOKEN` Bearer token for a REST catalog.

`ICEBERGR_REST_CREDENTIAL` OAuth2 client credential.

`ICEBERGR_REST_OAUTH2_SERVER_URI` OAuth2 token endpoint.

`ICEBERGR_REST_SCOPE` OAuth2 scope.

`ICEBERGR_S3_ACCESS_KEY_ID`, `ICEBERGR_S3_SECRET_ACCESS_KEY`, `ICEBERGR_S3_SESSION_TOKEN`
Object storage credentials.

The standard `AWS_*` variables are a fallback for the `ICEBERGR_S3_*` ones, but only for a connection that addresses object storage: `storage = "s3"`, `type = "glue"`, or an `s3://` warehouse. They are *not* forwarded to a catalog that has no object storage in sight, because a third-party REST catalog controls each table's location and may answer with its own `s3.endpoint`, at which point ambient keys would sign requests to a host it chose. Set `storage = "s3"` if a REST catalog identified by name needs them.

A credential is never sent over an unencrypted connection: an `http://` uri or OAuth2 endpoint is an error whenever any credential property is populated. An Authorization, Proxy-Authorization, Cookie or X-API-Key header passed as a header . property counts as one. A loopback address is exempt, since developing against a local catalog is ordinary, and `ICEBERGR_ALLOW_INSECURE_CREDENTIALS=true` overrides the check.

Catalog properties are never printed, logged or included in error messages, and `user:password@` in a uri is redacted when a catalog is printed. A credential property passed through ... anyway is accepted but warned about, since a script is the one place it should not be.

Examples

```
# A local warehouse needs no catalog server and no credentials.
warehouse <- tempfile("warehouse")
dir.create(warehouse)
catalog <- icebergr_catalog("memory", warehouse = warehouse)
```

```

catalog

## Not run:
# A REST catalog. The token comes from the environment, not from here.
Sys.setenv(ICEBERGR_REST_TOKEN = "...")
catalog <- icebergr_catalog("rest", uri = "https://catalog.example.com")

## End(Not run)

```

icebergr_collect	<i>Materialise a scan or a table</i>
------------------	--------------------------------------

Description

Materialise a scan or a table

Usage

```

icebergr_collect(x, ...)

## S3 method for class 'icebergr_scan'
icebergr_collect(x, ...)

## S3 method for class 'icebergr_table'
icebergr_collect(x, ...)

## S3 method for class 'icebergr_scan'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'icebergr_table'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

```

Arguments

x	An icebergr_scan from icebergr_scan() , or an icebergr_table (equivalent to scanning all of it).
...	Unused, for S3 consistency.
row.names	Unused, for consistency with base::as.data.frame() .
optional	Unused, for consistency with base::as.data.frame() .

Details

Data crosses from Rust into R over the Arrow C stream interface, so batches are handed over by pointer rather than serialised.

If the dplyr package is installed, `dplyr::collect()` also works on these objects.

Value

A tibble.

Examples

```
tbl <- icebergr_example_table(rows = 10)

# A scan, materialised
icebergr_collect(icebergr_scan(tbl, filter = id > 1000, select = c("id", "amount")))

# A whole table, materialised
icebergr_collect(tbl)
```

`icebergr_create_namespace`*Create a namespace*

Description

Create a namespace

Usage

```
icebergr_create_namespace(catalog, namespace)
```

Arguments

catalog	An icebergr_catalog from icebergr_catalog() .
namespace	The namespace to create. Accepts "db" or c("a", "b").

Value

catalog, invisibly.

Examples

```
warehouse <- tempfile("warehouse")
dir.create(warehouse)
catalog <- icebergr_catalog("memory", warehouse = warehouse)
icebergr_create_namespace(catalog, "db")
icebergr_list_namespaces(catalog)
```

icebergr_create_table *Create an Iceberg table*

Description

The table's schema is taken from data, so an existing data frame is enough to define one. Iceberg field ids are assigned automatically, since a data frame has no concept of them.

Usage

```
icebergr_create_table(catalog, table, data, location = NULL)
```

Arguments

catalog	An icebergr_catalog from icebergr_catalog() .
table	A table identifier, "namespace.table". The namespace must already exist; see icebergr_create_namespace() .
data	A data frame whose columns define the schema. No rows are written; only the column names and types are used. An Arrow schema is also accepted.
location	Where to store the table. NULL lets the catalog decide, which is almost always what you want. A local path is expanded and made absolute first, since it is recorded in the table's metadata as written.

Details

The table is created unpartitioned. Partitioned table creation, like partition evolution, is out of scope for this version; see [icebergr_spec_support\(\)](#).

Value

An icebergr_table handle for the new, empty table.

Examples

```
warehouse <- tempfile("warehouse")
dir.create(warehouse)
catalog <- icebergr_catalog("memory", warehouse = warehouse)
icebergr_create_namespace(catalog, "db")

tbl <- icebergr_create_table(
  catalog, "db.events",
  data.frame(id = integer(), amount = double(), label = character())
)
icebergr_schema(tbl)
```

icebergr_example_table

A small Iceberg table for offline examples and tests

Description

Builds a real Iceberg table in a local warehouse directory: two appends, so there is snapshot history to travel through and more than one data file for a filter to prune. Everything is local; no catalog server, network access or credentials are involved.

Usage

```
icebergr_example_table(warehouse = tempfile("icebergr-warehouse"), rows = 500L)
```

Arguments

warehouse	Directory to build the warehouse in. The default is a fresh temporary directory, created if needed.
rows	Rows per append. Two appends are made, so the table has twice this many rows.

Details

This is generated on demand rather than shipped as a committed table because Iceberg records absolute paths in its metadata and manifests: a table built on one machine does not resolve on another.

Value

An `icebergr_table` handle for `db.events`, with columns `id`, `event`, `amount`, `day` (a Date) and `recorded_at` (a POSIXct).

Examples

```
tbl <- icebergr_example_table(rows = 50)
tbl

icebergr_collect(icebergr_scan(tbl, filter = id > 1000, select = c("id", "amount")))

# Two snapshots, so the earlier state is still readable.
icebergr_snapshots(tbl)[, c("snapshot_id", "operation", "added_records")]
```

`icebergr_list_namespaces`*List namespaces in a catalog*

Description

List namespaces in a catalog

Usage

```
icebergr_list_namespaces(catalog, parent = NULL)
```

Arguments

<code>catalog</code>	An <code>icebergr_catalog</code> from <code>icebergr_catalog()</code> .
<code>parent</code>	Optional parent namespace, to list only its children. Accepts "a.b" or <code>c("a", "b")</code> .

Value

A character vector of namespaces, dot-separated when nested.

Examples

```
warehouse <- tempfile("warehouse")
dir.create(warehouse)
catalog <- icebergr_catalog("memory", warehouse = warehouse)
icebergr_list_namespaces(catalog)
```

`icebergr_list_tables` *List tables in a namespace*

Description

List tables in a namespace

Usage

```
icebergr_list_tables(catalog, namespace)
```

Arguments

<code>catalog</code>	An <code>icebergr_catalog</code> from <code>icebergr_catalog()</code> .
<code>namespace</code>	The namespace to list. Accepts "db" or <code>c("a", "b")</code> .

Value

A character vector of table names, without the namespace prefix.

Examples

```
warehouse <- tempfile("warehouse")
dir.create(warehouse)
catalog <- icebergr_catalog("memory", warehouse = warehouse)
# A namespace has to exist before it can hold tables.
icebergr_create_namespace(catalog, "db")
icebergr_list_tables(catalog, "db")
```

icebergr_partitions	<i>The partition specification of an Iceberg table</i>
---------------------	--

Description

The partition specification of an Iceberg table

Usage

```
icebergr_partitions(tbl)
```

Arguments

tbl An icebergr_table from [icebergr_table\(\)](#).

Details

Only the table's *default* (current) partition spec is reported. Reading historical specs would be part of partition evolution, which is out of scope for this version.

Value

A tibble with one row per partition field: spec_id, field_id, name, transform, source_id and source_name. An unpartitioned table returns zero rows.

Examples

```
# The example table is unpartitioned, so this has zero rows.
tbl <- icebergr_example_table(rows = 10)
icebergr_partitions(tbl)
```

icebergr_properties	<i>The properties of an Iceberg table</i>
---------------------	---

Description

Table properties are the free-form key-value settings Iceberg stores in table metadata (write defaults, compaction targets, engine-specific hints), as whichever engine created or last configured the table left them.

Usage

```
icebergr_properties(tbl)
```

Arguments

`tbl` An `icebergr_table` from `icebergr_table()`.

Details

These are read-only here. Setting them is an `update_properties` transaction, which is out of scope for this version; see `icebergr_spec_support()`.

Not to be confused with the `properties` argument of `icebergr_append()`, which records provenance in a single snapshot's summary rather than on the table.

Value

A tibble of name and value, ordered by name. A table with no properties returns zero rows.

Examples

```
tbl <- icebergr_example_table(rows = 10)
icebergr_properties(tbl)
```

icebergr_register_table	<i>Register an existing table with a catalog</i>
-------------------------	--

Description

Points a catalog at a table that already exists on disk, by giving it the table's metadata file. This is how a warehouse directory becomes visible to an in-process memory catalog, which keeps no persistent registry of its own.

Usage

```
icebergr_register_table(catalog, table, metadata_location, confine = TRUE)
```

Arguments

catalog	An icebergr_catalog from <code>icebergr_catalog()</code> .
table	A table identifier, "namespace.table", to register it under.
metadata_location	Path to the table's metadata.json, or its location in object storage, such as "s3://bucket/db/events/metadata/...". A local path has to exist; a remote one is left to the catalog to find.
confine	Whether to require metadata_location to sit inside the catalog's own warehouse. TRUE (the default) refuses anything outside it; FALSE allows any path. Ignored when the catalog has no local warehouse directory to confine against, such as a REST catalog identified by name or one whose warehouse is in object storage.

Value

An icebergr_table handle.

Registering a metadata file you did not write

A metadata file names its table's location, its manifest list and every data file, all as absolute paths, and registering it makes this package read them. Those paths are not constrained by where the metadata file itself sits, so a file from a shared drive or an issue attachment can point anywhere on disk. With the s3 feature compiled in it can also point at an s3:// or https:// location, which turns opening a nominally offline memory-catalog table into an outbound request to a host of its author's choosing.

confine = TRUE is the guard: the metadata file has to be inside the catalog's warehouse, which is the directory you nominated. It does not vet the paths *within* the file, so treat confine = FALSE as equivalent to running the file's author's code against your filesystem.

Examples

```
# Build a table, then re-attach it from a second catalog, as you would in a
# new session: a memory catalog keeps no registry between sessions.
warehouse <- tempfile("warehouse")
dir.create(warehouse)
catalog <- icebergr_catalog("memory", warehouse = warehouse)
icebergr_create_namespace(catalog, "db")
tbl <- icebergr_create_table(catalog, "db.events", data.frame(id = 1:3))
tbl <- icebergr_append(tbl, data.frame(id = 1:3))

# Iceberg writes one metadata file per commit; the newest is the current
# state of the table.
files <- list.files(warehouse,
  pattern = "metadata\\.json$", recursive = TRUE,
  full.names = TRUE
)
newest <- files[order(file.mtime(files))][length(files)]

reopened <- icebergr_catalog("memory", warehouse = warehouse)
icebergr_create_namespace(reopened, "db")
```

```
again <- icebergr_register_table(reopened, "db.events", newest)
icebergr_collect(again)
```

icebergr_reload	<i>Re-read a table's metadata from its catalog</i>
-----------------	--

Description

A table handle is a snapshot of the metadata as it was when the handle was opened, which is what makes a read consistent. That also means a handle never sees a commit made after it: `icebergr_append()` hands back an updated handle for your own writes, but a commit from another session, process or engine is invisible until the metadata is read again. This is how to do that without going back to the catalog by name.

Usage

```
icebergr_reload(tbl)
```

Arguments

`tbl` An `icebergr_table` from `icebergr_table()`.

Value

A new `icebergr_table` handle seeing the table's current state. The handle passed in is unchanged, so reassign it: `tbl <- icebergr_reload(tbl)`.

Examples

```
warehouse <- tempfile("warehouse")
dir.create(warehouse)
catalog <- icebergr_catalog("memory", warehouse = warehouse)
icebergr_create_namespace(catalog, "db")
events <- data.frame(id = 1:3L)
tbl <- icebergr_create_table(catalog, "db.events", events)

# A second handle on the same table, as another session would hold.
stale <- icebergr_table(catalog, "db.events")
tbl <- icebergr_append(tbl, events)

# The second handle still sees the table as it was when it was opened.
nrow(icebergr_collect(stale))
nrow(icebergr_collect(icebergr_reload(stale)))
```

icebergr_scan	<i>Scan an Iceberg table</i>
---------------	------------------------------

Description

Describes a read without performing it. Pass the result to `icebergr_collect()` or `as.data.frame()` to materialise it.

Usage

```
icebergr_scan(
  tbl,
  filter = NULL,
  select = NULL,
  limit = NULL,
  snapshot_id = NULL,
  as_of = NULL,
  batch_size = NULL,
  case_sensitive = TRUE
)
```

Arguments

<code>tbl</code>	An <code>icebergr_table</code> from <code>icebergr_table()</code> .
<code>filter</code>	An unquoted R expression, pushed down to Iceberg. See <i>Pushdown</i> below for what can be expressed.
<code>select</code>	Character vector of columns to read. NULL reads all of them.
<code>limit</code>	Maximum number of rows to return, or NULL for no limit. See <i>Pushdown</i> for an important caveat.
<code>snapshot_id</code>	Read this snapshot instead of the current one. A character id from <code>icebergr_snapshots()</code> .
<code>as_of</code>	Read the table as it was at this time, a POSIXct or Date. Resolved against the table's snapshot log to the snapshot that was current at that moment, so a snapshot a rollback abandoned, or one that only ever existed on another branch, is not selected even though it carries a matching timestamp. Cannot be combined with <code>snapshot_id</code> .
<code>batch_size</code>	Rows per Arrow batch, at least 1, or NULL for the default. Affects memory use, not results.
<code>case_sensitive</code>	Whether column names in <code>filter</code> and <code>select</code> are matched case-sensitively. When FALSE, each name is resolved to the table's own spelling before the scan is planned, so <code>select = "ID"</code> reads the column the table calls <code>id</code> . An exact match always wins. Iceberg column names are case-sensitive, so a table may hold both <code>id</code> and <code>ID</code> ; asking for <code>ID</code> reads <code>ID</code> , not whichever the two happen to be ordered. A name that matches no column exactly and more than one case-insensitively is ambiguous, and is an error rather than a silent choice between them.

Value

An `icebergr_scan` object.

Pushdown

`filter` and `select` are pushed down into scan planning, which is the whole performance argument for Iceberg over reading raw Parquet: manifests carry per-file statistics, so entire files and row groups are skipped before any bytes are read. Inspect the effect with `icebergr_scan_plan()`.

`limit` is **not** pushed down: `iceberg-rust` has no row limit in its scan API, so the same files are planned and rows are counted as batches arrive. It bounds how much is decoded and converted, not how much is planned.

Filters may use `==`, `!=`, `<`, `<=`, `>`, `>=`, `&`, `|`, `!`, `%in%`, `is.na()`, `is.nan()` and `startsWith()`. A bare name is read as a column when the table has a column of that name, and otherwise evaluated in the calling environment, so `filter = year == target` works with a local target. That holds wherever the name appears: an Iceberg predicate compares a column with a value, so a filter naming a column on both sides, such as `a > b`, is refused rather than reading `b` from the calling environment. Anything more elaborate should be applied in R after collecting.

A filter keeps the rows R's own evaluation of it would keep, NA and NaN included, although Iceberg's rules for them differ: no comparison matches a NaN, `is.na()` is TRUE for one, `x == 0` matches `-0` as well as `0`, and `!(x %in% c(1, 2))` keeps the rows where `x` is NA, since `%in%` is never NA in R. The exception is an ordering comparison between strings, which follows Iceberg's byte order; that agrees with R only in the C locale.

`startsWith()` is pushed down only against a string column, since Iceberg defines a prefix comparison for no other type.

A filter on a decimal column is pushed down, but with `iceberg-rust`'s row-level selection turned off for that scan: in 0.10 that stage drops every row of an ordering comparison against a decimal, so `price > 2.25` returned nothing at all. File and row-group pruning still apply, so such a scan is a little less selective and still correct.

Column names and time travel

Iceberg records a schema per snapshot, so `filter` and `select` are resolved against the schema of the snapshot actually being read: the one named by `snapshot_id` or `as_of`, and otherwise the current snapshot. A column another engine has since renamed or dropped is therefore still nameable as of a snapshot that had it, and one added afterwards is refused for a snapshot that did not. `icebergr_schema()` takes the same `snapshot_id` and reports what those columns are.

The current snapshot's schema is usually the table's current schema, but not straight after another engine changes the schema without writing anything, such as an `ALTER TABLE ... ADD COLUMN`. Until the next commit, a read still has the columns the current snapshot was written with, under their old names, while `icebergr_schema()` already reports the new ones.

Examples

```
tbl <- icebergr_example_table(rows = 10)

# Projection and predicate pushdown
scan <- icebergr_scan(tbl, filter = id > 1000 & amount > 900, select = c("id", "amount"))
```

```

scan
icebergr_collect(scan)

# A local variable is usable in a filter: a bare name is read as a column
# only when the table has one of that name.
cutoff <- 1005
icebergr_collect(icebergr_scan(tbl, filter = id > cutoff, select = "id"))

# Time travel, to the state before the second append
history <- icebergr_snapshots(tbl)
nrow(icebergr_collect(icebergr_scan(tbl, snapshot_id = history$snapshot_id[[1]])))
nrow(icebergr_collect(icebergr_scan(tbl, as_of = history$timestamp[[1]])))

```

icebergr_scan_plan	<i>Inspect the file plan for a scan</i>
--------------------	---

Description

Reports which data files a scan would read, without reading them. This is how pushdown is verified rather than assumed: a filtered scan should plan fewer files, and fewer records, than an unfiltered one.

Usage

```
icebergr_scan_plan(scan)
```

Arguments

scan An icebergr_scan from `icebergr_scan()`.

Value

A tibble with one row per planned file task: data_file_path, record_count, file_size_in_bytes, start and length.

record_count is NA for a task covering part of a file, since a partial read has no meaningful record count from the manifest.

Examples

```

tbl <- icebergr_example_table(rows = 10)

# The example table has two data files, one per append.
all_files <- icebergr_scan_plan(icebergr_scan(tbl))
hot_files <- icebergr_scan_plan(icebergr_scan(tbl, filter = id > 1000))

nrow(hot_files) < nrow(all_files)
sum(hot_files$record_count) < sum(all_files$record_count)

```

icebergr_schema	<i>The schema of an Iceberg table</i>
-----------------	---------------------------------------

Description

The schema of an Iceberg table

Usage

```
icebergr_schema(tbl, snapshot_id = NULL)
```

Arguments

tbl	An icebergr_table from icebergr_table() .
snapshot_id	Report the schema as it was at this snapshot rather than the current one. A character id from icebergr_snapshots() .

Details

Iceberg records a schema per snapshot, so a table whose columns were changed by another engine has more than one. `snapshot_id` is how the earlier one is read, and it is also what [icebergr_scan\(\)](#) resolves filter and select against when it is given a `snapshot_id` or an `as_of`: a column that has since been renamed or dropped is still nameable as of the snapshot that had it.

Value

A tibble with one row per top-level field: `field_id`, `name`, `type` (the Iceberg type), `required` and `doc`.

Examples

```
tbl <- icebergr_example_table(rows = 10)
icebergr_schema(tbl)

# The schema as of the first snapshot.
history <- icebergr_snapshots(tbl)
icebergr_schema(tbl, snapshot_id = history$snapshot_id[[1]])
```

icebergr_snapshots	<i>Snapshot history of an Iceberg table</i>
--------------------	---

Description

Snapshot history of an Iceberg table

Usage

```
icebergr_snapshots(tbl)
```

Arguments

tbl An icebergr_table from `icebergr_table()`.

Value

A tibble of snapshots, oldest first, with columns:

snapshot_id Character. See the note on ids below.

parent_snapshot_id Character, NA for the first snapshot.

sequence_number Numeric.

timestamp POSIXct in UTC, when the snapshot was committed.

operation "append", "overwrite", "replace" or "delete".

schema_id Integer, the schema in force for that snapshot.

added_records, total_records Numeric, from the snapshot summary, NA when the writer did not record them.

summary The full snapshot summary as a JSON string.

manifest_list Path to the snapshot's manifest list.

Snapshot ids are character

Iceberg assigns snapshot ids as random 64-bit integers, and R's numeric type holds only 53 bits of integer precision. A large id passed through a double would come back subtly altered and then silently select the wrong snapshot, so ids are character throughout, and `icebergr_scan()` accepts them as such.

Every snapshot, not only the current line

This is the table's snapshot *list*: every snapshot the metadata still carries, ordered by commit time. That is not always the same as the states the table passed through. A rollback leaves the snapshot it abandoned in the list, and a snapshot committed to another branch appears here too, in both cases with a timestamp at which it was never the table's current state. Reading with `icebergr_scan(as_of = )` follows Iceberg's snapshot log instead, so it is not misled by either; any id listed here can still be read directly with `icebergr_scan(snapshot_id = )`.

Examples

```
# Two appends, so there is history to travel through.
tbl <- icebergr_example_table(rows = 10)

history <- icebergr_snapshots(tbl)
history[, c("snapshot_id", "operation", "added_records", "total_records")]

# Read the table as it was at its first snapshot.
icebergr_collect(icebergr_scan(tbl, snapshot_id = history$snapshot_id[[1]]))
```

icebergr_spec_support *What this build of icebergr supports*

Description

Reports the supported Iceberg spec versions and a feature-by-feature matrix, resolved against the optional Cargo features this particular binary was compiled with. Checking here is more reliable than inferring from the documentation, because optional features change what is available.

Usage

```
icebergr_spec_support()
```

Value

A list with class `icebergr_spec_support`:

`iceberg_rust_version` The pinned iceberg-rust version.

`arrow_version` The version of the Rust arrow crate the interchange layer was built against.

`spec_versions` Iceberg table spec versions that can be read and written.

`catalogs` Catalog types available in this build.

`cargo_features` Optional Cargo features compiled in.

`features` A tibble of feature, supported and reason. `supported` is TRUE, FALSE, or NA for something this build supports only in part, with `reason` saying which part. A feature that depends on an optional Cargo feature is resolved against this build, so it is TRUE or FALSE here and never NA.

Examples

```
support <- icebergr_spec_support()
support

# Check a capability before relying on it.
features <- support$features
features[features$feature == "MERGE / upsert", ]
```

icebergr_table	<i>Open an Iceberg table</i>
----------------	------------------------------

Description

Open an Iceberg table

Usage

```
icebergr_table(catalog, table)
```

Arguments

catalog	An icebergr_catalog from icebergr_catalog() .
table	A table identifier, "namespace.table". Nested namespaces are written "a.b.table".

Details

The handle is a snapshot of the table's metadata at the moment it was opened. Appending with [icebergr_append\(\)](#) returns an updated handle rather than mutating this one, so a handle always reads a consistent view.

Value

An icebergr_table handle.

Examples

```
# A local warehouse, so this runs offline.
warehouse <- tempfile("warehouse")
dir.create(warehouse)
catalog <- icebergr_catalog("memory", warehouse = warehouse)
icebergr_create_namespace(catalog, "db")
icebergr_create_table(catalog, "db.events", data.frame(id = integer()))

tbl <- icebergr_table(catalog, "db.events")
icebergr_schema(tbl)

## Not run:
# The same call against a REST catalog, which needs a server.
catalog <- icebergr_catalog("rest", uri = "https://catalog.example.com")
tbl <- icebergr_table(catalog, "db.events")

## End(Not run)
```

icebergr_table_exists *Whether a table exists in a catalog*

Description

Asks the catalog directly, so an absent table is an answer rather than an error to be caught.

Usage

```
icebergr_table_exists(catalog, table)
```

Arguments

catalog	An icebergr_catalog from <code>icebergr_catalog()</code> .
table	A table identifier, "namespace.table".

Details

A namespace that does not exist gives FALSE rather than an error, since it cannot hold the table either way. Any other failure, such as an unreachable catalog or a rejected credential, is still an error, because reporting one of those as "no such table" would be a confident wrong answer.

Value

TRUE or FALSE.

Examples

```
warehouse <- tempfile("warehouse")
dir.create(warehouse)
catalog <- icebergr_catalog("memory", warehouse = warehouse)
icebergr_create_namespace(catalog, "db")

icebergr_table_exists(catalog, "db.events")
icebergr_create_table(catalog, "db.events", data.frame(id = integer()))
icebergr_table_exists(catalog, "db.events")
```

Index

`as.data.frame()`, [15](#)
`as.data.frame.icebergr_scan`
 (`icebergr_collect`), [6](#)
`as.data.frame.icebergr_table`
 (`icebergr_collect`), [6](#)

`base::as.data.frame()`, [6](#)

`icebergr_append`, [2](#)
`icebergr_append()`, [12](#), [14](#), [21](#)
`icebergr_catalog`, [4](#)
`icebergr_catalog()`, [7](#), [8](#), [10](#), [13](#), [21](#), [22](#)
`icebergr_collect`, [6](#)
`icebergr_collect()`, [15](#)
`icebergr_create_namespace`, [7](#)
`icebergr_create_namespace()`, [8](#)
`icebergr_create_table`, [8](#)
`icebergr_example_table`, [9](#)
`icebergr_list_namespaces`, [10](#)
`icebergr_list_namespaces()`, [4](#)
`icebergr_list_tables`, [10](#)
`icebergr_partitions`, [11](#)
`icebergr_partitions()`, [3](#)
`icebergr_properties`, [12](#)
`icebergr_register_table`, [12](#)
`icebergr_register_table()`, [3](#)
`icebergr_reload`, [14](#)
`icebergr_scan`, [15](#)
`icebergr_scan()`, [6](#), [17–19](#)
`icebergr_scan_plan`, [17](#)
`icebergr_scan_plan()`, [16](#)
`icebergr_schema`, [18](#)
`icebergr_schema()`, [16](#)
`icebergr_snapshots`, [19](#)
`icebergr_snapshots()`, [15](#), [18](#)
`icebergr_spec_support`, [20](#)
`icebergr_spec_support()`, [3](#), [8](#), [12](#)
`icebergr_table`, [21](#)
`icebergr_table()`, [3](#), [4](#), [11](#), [12](#), [14](#), [15](#), [18](#), [19](#)
`icebergr_table_exists`, [22](#)

`nanoarrow::as_nanoarrow_array_stream()`,
 [3](#)