Package: catalog (via r-universe)

October 10, 2024

Type Package

Title Access the 'Spark Catalog' API via 'sparklyr'

Version 0.1.1

Maintainer Nathan Eastwood <nathan.eastwood@icloud.com>

Description Gain access to the 'Spark Catalog' API making use of the
'sparklyr' API. 'Catalog'
<https://spark.apache.org/docs/2.4.3/api/java/org/apache/spark/sql/catalog/Catalog.html>
is the interface for managing a metastore (aka metadata
catalog) of relational entities (e.g. database(s), tables,
functions, table columns and temporary views).

URL https://nathaneastwood.github.io/catalog/,
https://github.com/nathaneastwood/catalog

BugReports https://github.com/nathaneastwood/catalog/issues

Depends R (>= 3.5)

Imports dplyr, sparklyr

Suggests DBI, knitr, rmarkdown, roxygen2, tinytest

License GPL (>= 2)

Encoding UTF-8

RoxygenNote 7.2.1

Roxygen list(markdown = TRUE)

Repository https://nathaneastwood.r-universe.dev

RemoteUrl https://github.com/nathaneastwood/catalog

RemoteRef HEAD

RemoteSha 3421fdfcfef6a74c21b648734babfb0c16c1d455

Contents

`cache_table` *Cache And Uncache Tables*

Description

Spark SQL can cache tables using an in-memory columnar format by calling `cache_table()`. Spark SQL will scan only required columns and will automatically tune compression to minimize memory usage and GC pressure. You can call `uncache_table()` to remove the table from memory. Similarly you can call `clear_cache()` to remove all cached tables from the in-memory cache. Finally, use `is_cached()` to test whether or not a table is cached.

Usage

```
cache_table(sc, table)

clear_cache(sc)

is_cached(sc, table)

uncache_table(sc, table)
```

Arguments

`sc`	A `spark_connection`.
`table`	`character(1)`. The name of the table.

Value

- `cache_table()`: If successful, `TRUE`, otherwise `FALSE`.
- `clear_cache()`: `NULL`, invisibly.
- `is_cached()`: A `logical(1)` vector indicating `TRUE` if the table is cached and `FALSE` otherwise.
- `uncache_table()`: `NULL`, invisibly.

See Also

`create_table()`, `get_table()`, `list_tables()`, `refresh_table()`, `table_exists()`, `uncache_table()`

Examples

```
## Not run:
sc <- sparklyr::spark_connect(master = "local")
mtcars_spark <- sparklyr::copy_to(dest = sc, df = mtcars)

# By default the table is not cached
is_cached(sc = sc, table = "mtcars")

# We can manually cache the table
cache_table(sc = sc, table = "mtcars")
# And now the table is cached
is_cached(sc = sc, table = "mtcars")

# We can uncache the table
uncache_table(sc = sc, table = "mtcars")
is_cached(sc = sc, table = "mtcars")

## End(Not run)
```

`create_table` *Create A Table*

Description

Creates a table, in the hive warehouse, from the given path and returns the corresponding `DataFrame`. The table will contain the contents of the file that is in the `path` parameter.

Usage

```
create_table(sc, table, path, source, ...)
```

Arguments

sc	A spark_connection.
table	character(1). The name of the table to create.
path	character(1). The path to use to create the table.
source	character(1). The data source to use to create the table such as "parquet", "csv", etc.
...	Additional options to be passed to the createTable method.

Details

The default data source type is parquet. This can be changed using source or setting the configuration option spark.sql.sources.default when creating the spark session using or after you have created the session using

```
config <- sparklyr::spark_config()
config[["spark.sql.sources.default"]] <- "csv"
```

Value

A tbl_spark.

See Also

cache_table(), get_table(), list_tables(), refresh_table(), table_exists(), uncache_table()

---

current_database *Get The Current Database*

Description

Returns the current database in this session. By default your session will be connected to the "default" database (named "default") and to change database you can use set_current_database().

Usage

```
current_database(sc)
```

Arguments

sc	A spark_connection.

Value

character(1), the current database name.

See Also

set_current_database(), database_exists(), list_databases()

Examples

```
## Not run:
sc <- sparklyr::spark_connect(master = "local")
current_database(sc = sc)

## End(Not run)
```

---

database_exists *Check If A Database Exists*

Description

Check if the database with the specified name exists. This will check the list of hive databases in the current session to see if the database exists.

Usage

```
database_exists(sc, name)
```

Arguments

sc	A spark_connection.
name	character(1). The name of the database to set the current database to.

Value

A logical(1) vector indicating TRUE if the database exists and FALSE otherwise.

See Also

current_database(), set_current_database(), list_databases()

Examples

```
## Not run:
sc <- sparklyr::spark_connect(master = "local")
database_exists(sc = sc, name = "default")
database_exists(sc = sc, name = "fake_database")

## End(Not run)
```

drop_global_temp_view *Temporary View*

Description

- `drop_global_temp_view()`: Drops the global temporary view with the given view name in the catalog.
- `drop_temp_view()`: Drops the local temporary view with the given view name in the catalog. Local temporary view is session-scoped. Its lifetime is the lifetime of the session that created it, i.e. it will be automatically dropped when the session terminates. It's not tied to any databases.

Usage

```
drop_global_temp_view(sc, view)

drop_temp_view(sc, view)
```

Arguments

`sc`	A `spark_connection`.
`view`	`character(1)`. The name of the temporary view to be dropped.

Value

A `logical(1)` vector indicating whether the temporary view was dropped (`TRUE`) or not (`FALSE`).

See Also

`list_tables()`

Examples

```
## Not run:
sc <- sparklyr::spark_connect(master = "local")
mtcars_spark <- sparklyr::copy_to(dest = sc, df = mtcars)

# We can check which temporary tables are in scope
list_tables(sc = sc)

# And then drop those we wish to drop
drop_temp_view(sc = sc, view = "mtcars")

## End(Not run)
```

`function_exists` *Check If A Function Exists*

Description

Check if the function with the specified name exists in the specified database.

Usage

```
function_exists(sc, fn, database = NULL)
```

Arguments

`sc`	A `spark_connection`.
`fn`	`character(1)`. The name of the function.
`database`	`character(1)`. The name of the database for which the functions should be listed (default: `NULL`).

Details

`function_exists()` includes in-built functions such as `abs`. To see if a built-in function exists you must use the unqualified name. If you create a function you can use the qualified name. If you want to check if a built-in function exists specify the `database` as `NULL`.

Value

A `logical(1)` vector indicating `TRUE` if the function exists within the specified database and `FALSE` otherwise.

Examples

```
## Not run:
sc <- sparklyr::spark_connect(master = "local")
function_exists(sc = sc, fn = "abs")

## End(Not run)
```

get_function *Get A Function*

Description

Get the function with the specified name.

Usage

```
get_function(sc, fn, database = NULL)
```

Arguments

sc	A spark_connection.
fn	character(1). The name of the function.
database	character(1). The name of the database for which the functions should be listed (default: NULL).

Details

If you are trying to get an in-built function then use the unqualified name and pass NULL as the database name.

Value

A spark_jobj which includes the class name, database, description, whether it is temporary and the name of the function.

See Also

function_exists(), list_functions()

Examples

```
## Not run:
sc <- sparklyr::spark_connect(master = "local")
get_function(sc = sc, fn = "Not")

## End(Not run)
```

get_table *Get A Table*

Description

Get the table or view with the specified name in the specified database. You can use this to find the table's description, database, type and whether it is a temporary table or not.

Usage

```
get_table(sc, table, database = NULL)
```

Arguments

sc	A spark_connection.
table	character(1). The name of the table.
database	character(1). The name of the database for which the functions should be listed (default: NULL).

Value

An object of class spark_jobj and shell_jobj.

See Also

cache_table(), create_table(), list_tables(), refresh_table(), table_exists(), uncache_table()

list_columns *List Columns*

Description

Returns a list of columns for the given table/view in the specified database. The result includes the name, description, dataType, whether it is nullable or if it is partitioned and if it is broken in buckets.

Usage

```
list_columns(sc, table, database = NULL)
```

Arguments

sc	A spark_connection.
table	character(1). The name of the table.
database	character(1). The name of the database for which the functions should be listed (default: NULL).

Value

A `tibble` with 6 columns:

- `name` - The name of the column.
- `description` - Description of the column.
- `dataType` - The column data type.
- `nullable` - Whether the column is nullable or not.
- `isPartition` - Whether the column is partitioned or not.
- `isBucket` - Whether the column is broken in buckets.

Examples

```
## Not run:
sc <- sparklyr::spark_connect(master = "local")
mtcars_spark <- sparklyr::copy_to(dest = sc, df = mtcars)
list_columns(sc = sc, table = "mtcars")

## End(Not run)
```

`list_databases` *List Databases*

Description

Returns a list of databases available across all sessions. The result contains the name, description and locationUri of each database.

Usage

```
list_databases(sc)
```

Arguments

`sc`	A `spark_connection`.

Value

A `tibble` containing 3 columns:

- `name` - The name of the database.
- `description` - Description of the database.
- `locationUri` - Path (in the form of a uri) to data files.

See Also

`current_database()`, `database_exists()`, `set_current_database()`

Examples

```
## Not run:
sc <- sparklyr::spark_connect(master = "local")
list_databases(sc = sc)

## End(Not run)
```

`list_functions` *List Functions*

Description

Returns a list of functions registered in the specified database. This includes all temporary functions. The result contains the class name, database, description, whether it is temporary and the name of each function.

Usage

```
list_functions(sc, database = NULL)
```

Arguments

`sc`	A `spark_connection`.
`database`	`character(1)`. The name of the database for which the functions should be listed (default: `NULL`).

Value

A `tibble` containing 5 columns:

- `name` - Name of the function.
- `database` - Name of the database the function belongs to.
- `description` - Description of the function.
- `className` - The fully qualified class name of the function.
- `isTemporary` - Whether the function is temporary or not.

See Also

`function_exists()`, `get_function()`

Examples

```
## Not run:
sc <- sparklyr::spark_connect(master = "local")
list_functions(sc = sc)
list_functions(sc = sc, database = "default")

## End(Not run)
```

`list_tables` *List Tables In A Spark Connection*

Description

Returns a list of tables/views in the current database. The result includes the name, database, description, table type and whether the table is temporary or not.

Usage

```
list_tables(sc, database = NULL)
```

Arguments

`sc`	A `spark_connection`.
`database`	`character(1)`. The name of the database for which the functions should be listed (default: `NULL`).

Value

A `tibble` containing 5 columns:

- `name` - The name of the table.
- `database` - Name of the database the table belongs to.
- `description` - Description of the table.
- `tableType` - The type of table (e.g. view/table)
- `isTemporary` - Whether the table is temporary or not.

See Also

`cache_table()`, `create_table()`, `get_table()`, `refresh_table()`, `table_exists()`, `uncache_table()`

Examples

```
## Not run:
sc <- sparklyr::spark_connect(master = "local")
mtcars_spakr <- sparklyr::copy_to(dest = sc, df = mtcars)
list_tables(sc = sc)

## End(Not run)
```

`refresh` *Refreshing Data*

Description

- `recover_partitions()`: Recovers all the partitions in the directory of a table and update the catalog. This only works for partitioned tables and not un-partitioned tables or views.
- `refresh_by_path()`: Invalidates and refreshes all the cached data (and the associated metadata) for any Dataset that contains the given data source path. Path matching is by prefix, i.e. "/" would invalidate everything that is cached.
- `refresh_table()`: Invalidates and refreshes all the cached data and metadata of the given table. For performance reasons, Spark SQL or the external data source library it uses might cache certain metadata about a table, such as the location of blocks. When those change outside of Spark SQL, users should call this function to invalidate the cache. If this table is cached as an `InMemoryRelation`, drop the original cached version and make the new version cached lazily.

Usage

```
recover_partitions(sc, table)

refresh_by_path(sc, path)

refresh_table(sc, table)
```

Arguments

`sc`	A `spark_connection`.
`table`	`character(1)`. The name of the table.
`path`	`character(1)`. The path to refresh.

Value

`NULL`, invisibly. These functions are mostly called for their side effects.

See Also

`cache_table()`, `create_table()`, `get_table()`, `list_tables()`, `table_exists()`, `uncache_table()`

set_current_database *Set The Current Database*

Description

Sets the current default database in this session.

Usage

```
set_current_database(sc, name)
```

Arguments

sc	A spark_connection.
name	character(1). The name of the database to set the current database to.

Value

If successful, TRUE, otherwise errors.

See Also

current_database(), database_exists(), list_databases()

Examples

```
## Not run:
sc <- sparklyr::spark_connect(master = "local")
set_current_database(sc = sc, name = "new_db")

## End(Not run)
```

table_exists *Check If A Table Exists*

Description

Check if the table or view with the specified name exists in the specified database. This can either be a temporary view or a table/view.

Usage

```
table_exists(sc, table, database = NULL)
```

Arguments

sc	A spark_connection.
table	character(1). The name of the table.
database	character(1). The name of the database for which the functions should be listed (default: NULL).

Details

If database is NULL, table_exists refers to a table in the current database (see current_database()).

Value

A logical(1) vector indicating TRUE if the table exists within the specified database and FALSE otherwise.

See Also

cache_table(), create_table(), get_table(), list_tables(), refresh_table(), uncache_table()

Examples

```
## Not run:
sc <- sparklyr::spark_connect(master = "local")
mtcars_spark <- sparklyr::copy_to(dest = sc, df = mtcars)
table_exists(sc = sc, table = "mtcars")

## End(Not run)
```

Index