Package: motherduck (via r-universe)

October 31, 2026

Type Package

Title Utilities for Managing a 'Motherduck' Database

Version 0.2.4

Description Provides helper functions, metadata utilities, and workflows for administering and managing 'DuckDB' databases on the 'MotherDuck' cloud platform. Some features require a 'MotherDuck' account (<https://motherduck.com/>). Note: Windows is not supported; use 'WSL' (Windows Subsystem for Linux) instead (<https://motherduck.com/docs/integrations/language-apis-and-drivers/r/>).

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

LazyData true

Imports DBI, assertthat, cli, dbplyr, dplyr, duckdb (>= 1.3.0), glue, purrr, stringr, httr2, rlang, janitor, tibble

Suggests testthat (>= 3.0.0), quarto, contoso, openxlsx

Config/testthat/edition 3

Depends R (>= 4.1.0)

URL https://usrbinr.codeberg.page/motherduck/, https://codeberg.org/usrbinr/motherduck

BugReports https://codeberg.org/usrbinr/motherduck/issues

Config/Needs/website rmarkdown

Config/roxygen2/version 8.0.0

Config/pak/sysreqs libicu-dev libssl-dev

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

Date/Publication 2026-10-31 00:19:39 UTC

RemoteUrl https://github.com/usrbinr/motherduck

RemoteRef HEAD

RemoteSha e6c482172ac417a7847a931055ef163be80808cc

Contents

alter_table_schema . . . 3
cd . . . 4
config_csv . . . 5
config_db . . . 6
config_excel . . . 8
config_parquet . . . 9
configure_md_user_settings . . . 9
connect_to_motherduck . . . 11
copy_tables_to_new_location . . . 12
create_database . . . 13
create_if_not_exists_share . . . 14
create_md_access_token . . . 15
create_md_user . . . 17
create_or_replace_share . . . 18
create_r2_secret . . . 19
create_s3_secret . . . 20
create_schema . . . 22
create_table . . . 23
delete_and_create_schema . . . 24
delete_database . . . 25
delete_md_access_token . . . 25
delete_md_user . . . 26
delete_schema . . . 27
delete_table . . . 28
describe_share . . . 29
drop_share . . . 30
install_extensions . . . 31
launch_ui . . . 32
list_all_databases . . . 33
list_all_tables . . . 34
list_current_schemas . . . 34
list_current_tables . . . 35
list_extensions . . . 36
list_fns . . . 37
list_md_active_accounts . . . 38
list_md_user_instance . . . 39
list_md_user_tokens . . . 40
list_owned_shares . . . 41
list_setting . . . 42
list_shared_with_me_shares . . . 43
list_shares . . . 44
load_extensions . . . 45
pwd . . . 46
read_csv . . . 47
read_excel . . . 48
show_current_user . . . 49

show_information_schema 50
show_motherduck_token 51
summary 52
upload_database_to_md 53
validate_con 54
validate_database_exists 55
validate_database_schema_exists 55
validate_extension_install_status 56
validate_extension_load_status 57
validate_md_connection_status 59

Index **60**

`alter_table_schema` *Move Tables from One Schema to Another*

Description

Moves one or more tables from an existing schema to a new (target) schema using `ALTER TABLE ... SET SCHEMA`. If the target schema does not exist, it is created first.

Usage

```
alter_table_schema(.con, table_names, from_schema, new_schema, silent = FALSE)
```

Arguments

`.con`	A valid `DBI` connection (DuckDB / MotherDuck).
`table_names`	Character vector of table names to move.
`from_schema`	Source schema name (where the tables currently reside).
`new_schema`	Target schema name (where the tables will be moved).
`silent`	Logical; if `TRUE`, suppress CLI output. Default is `FALSE`.

Details

- Ensures `new_schema` exists (`CREATE SCHEMA IF NOT EXISTS`).
- For each table in `table_names`, runs: `ALTER TABLE from_schema.table SET SCHEMA new_schema`.
- Table and schema identifiers are safely quoted with `glue::glue_sql()`.

Value

Invisibly returns `NULL`. Side effects: creates `new_schema` if needed and moves tables.

See Also

Other db-manage: copy_tables_to_new_location(), create_database(), create_if_not_exists_share(), create_or_replace_share(), create_schema(), create_table(), delete_and_create_schema(), delete_database(), delete_schema(), delete_table(), describe_share(), drop_share(), list_owned_shares(), list_shared_with_me_shares(), upload_database_to_md()

---

cd *Change Active Database and Schema*

Description

Switches the active database and (optionally) schema for a valid DuckDB or MotherDuck connection. The function validates the target database and schema before executing the USE command and provides user feedback via CLI messages.

Usage

```
cd(.con, database_name, schema_name, silent = FALSE)
```

Arguments

.con	A valid DBI connection (DuckDB / MotherDuck).
database_name	A character string specifying the database to switch to. Must be one of the available databases returned by list_all_databases().
schema_name	(Optional) A character string specifying the schema to switch to within the given database. Must be one of the available schemas returned by list_current_schemas().
silent	to print message or not, default is FALSE

Details

The cd() function is analogous to a "change directory" command in a file system, but for database contexts. It updates the currently active database (and optionally schema) for the given connection. If the target database or schema does not exist, the function aborts with a descriptive CLI error message.

Value

Invisibly returns a message summarizing the new connection context. Side effects include printing CLI headers showing the current user and database context.

See Also

Other db-meta: launch_ui(), pwd(), summary()

Examples

```
## Not run:
# Connect to MotherDuck
con <- DBI::dbConnect(duckdb::duckdb(dbdir = tempfile()))

# List available databases
list_databases(con)

# Change to a specific database and schema
cd(con, database_name = "analytics_db", schema_name = "public")

# Disconnect
DBI::dbDisconnect(con)

## End(Not run)
```

`config_csv` *DuckDB CSV read configuration (config_csv)*

Description

A named character list of DuckDB CSV reading configuration settings used by the package. This includes options such as type detection, delimiter, quote handling, sample size, and other parser flags. These reflect the values returned by `config_csv`.

Usage

```
config_csv
```

Format

A named character list. Example names include:

all_varchar character; "true"/"false"

allow_quoted_nulls character; "true"/"false"

auto_detect character; "true"/"false"

auto_type_candidates character; types used for detection

buffer_size character; buffer size in bytes

columns character; column names/types struct or empty

delim character; delimiter string

skip character; number of lines to skip

Examples

```
## Not run:
config_csv$all_varchar

## End(Not run)
```

`config_db` *DuckDB runtime database configuration (config_db)*

Description

A named character list of DuckDB runtime configuration settings used by the package. This object includes allocator settings, thread counts, extension flags, storage and checkpoint options, security and secret settings, and other engine-level options. These settings reflect the values returned by `config_db` and are suitable as a template or reference for configuring DuckDB instances created via the package.

Usage

```
config_db
```

Format

A named character list. Example names and values include:

access_mode character; e.g. "automatic"

allocator_background_threads character; "true"/"false"

allocator_bulk_deallocation_flush_threshold character; e.g. "512MB"

allocator_flush_threshold character; e.g. "128MB"

allow_community_extensions character; "true"/"false"

allow_extensions_metadata_mismatch character; "true"/"false"

allow_persistent_secrets character; "true"/"false"

allow_unredacted_secrets character; "true"/"false"

allow_unsigned_extensions character; "true"/"false"

arrow_large_buffer_size character; "true"/"false"

arrow_lossless_conversion character; "true"/"false"

arrow_output_list_view character; "true"/"false"

autoinstall_extension_repository character; URL or empty string

autoinstall_known_extensions character; "true"/"false"

autoload_known_extensions character; "true"/"false"

ca_cert_file character; path to certificate file, or empty string

catalog_error_max_schemas character; numeric as string, e.g. "100"
checkpoint_threshold character; e.g. "16MB"
wal_autocheckpoint character; e.g. "16MB"
custom_extension_repository character; URL or empty string
custom_user_agent character; string or empty
default_block_size character; e.g. "262144"
default_collation character; collation name or empty
default_null_order character; e.g. "NULLS_LAST"
null_order character; e.g. "NULLS_LAST"
default_order character; e.g. "ASC"
default_secret_storage character; e.g. "local_file"
disabled_compression_methods character; list or empty
duckdb_api character; e.g. "cli"
enable_external_access character; "true"/"false"
enable_external_file_cache character; "true"/"false"
enable_fsst_vectors character; "true"/"false"
enable_http_metadata_cache character; "true"/"false"
enable_macro_dependencies character; "true"/"false"
enable_object_cache character; "true"/"false"
enable_server_cert_verification character; "true"/"false"
enable_view_dependencies character; "true"/"false"
extension_directory character; path or empty
external_threads character; numeric as string, e.g. "1"
force_download character; "true"/"false"
immediate_transaction_mode character; "true"/"false"
index_scan_max_count character; numeric as string, e.g. "2048"
index_scan_percentage character; numeric as string, e.g. "0.001"
lock_configuration character; "true"/"false"
max_vacuum_tasks character; numeric as string, e.g. "100"
old_implicit_casting character; "true"/"false"
parquet_metadata_cache character; "true"/"false"
preserve_insertion_order character; "true"/"false"
produce_arrow_string_view character; "true"/"false"
scheduler_process_partial character; "true"/"false"
secret_directory character; path for persistent secrets
storage_compatibility_version character; e.g. "v0.10.2"
temp_directory character; path or empty string
threads character; number of threads as string, e.g. "4"
worker_threads character; number of threads as string, e.g. "4"
username character; string or empty
user character; string or empty
zstd_min_string_length character; numeric as string, e.g. "4096"

Examples

```
# inspect the config
config_db
# access individual settings
config_db$threads
```

`config_excel` *DuckDB Excel read configuration (config_excel)*

Description

A named character list of DuckDB Excel reading configuration settings used by the package. Includes options such as reading binary as string, adding filename/row_number columns, Hive partitioning, and union by name. These reflect the values returned by `config_excel`.

Usage

```
config_excel
```

Format

A named character list. Example names include:

binary_as_string character; "true"/"false"

encryption_config character; encryption struct or "-" if none

filename character; "true"/"false"

file_row_number character; "true"/"false"

hive_partitioning character; e.g., "(auto-detect)"

union_by_name character; "true"/"false"

Examples

```
## Not run:
config_excel$binary_as_string

## End(Not run)
```

config_parquet *DuckDB Parquet read configuration (config_parquet)*

Description

A named character list of DuckDB Parquet reading configuration settings used by the package. Includes options such as binary encoding, filename columns, row numbers, and union by name. Reflects `config_parquet`.

Usage

```
config_parquet
```

Format

A named character list. Example names include:

binary_as_string character; "true"/"false"

encryption_config character; encryption struct or "-" if none

filename character; "true"/"false"

file_row_number character; "true"/"false"

hive_partitioning character; e.g., "(auto-detect)"

union_by_name character; "true"/"false"

Examples

```
## Not run:
config_parquet$binary_as_string

## End(Not run)
```

configure_md_user_settings *Configure a MotherDuck user's settings*

Description

Updates a MotherDuck user's configuration settings, including token type, instance size, and flock size. This function uses the MotherDuck REST API to apply the changes for the specified user.

Usage

```
configure_md_user_settings(
  user_name,
  token_type = "read_write",
  instance_size = "pulse",
  flock_size = 0
)
```

Arguments

user_name	Character. The username of the MotherDuck user to configure.
token_type	Character. The type of access token for the user; must be "read_write" or "read_scaling" (default: "read_write").
instance_size	Character. The instance size for the user; must be one of "pulse", "standard", "jumbo", "mega", "giga" (default: "pulse").
flock_size	Numeric. The flock size for the user; must be a whole number between 0 and 60 (default: 0).

Details

This function validates each parameter before making a PUT request to the MotherDuck API. It ensures that:

- token_type is valid using validate_token_type().
- instance_size is valid using validate_instance_size().
- flock_size is a valid integer using validate_flock_size().

Uses the token from the MOTHERDUCK_TOKEN environment variable for authorization.

Value

A tibble containing the API response, including the updated settings for the user.

See Also

Other db-api: create_md_access_token(), create_md_user(), delete_md_access_token(), delete_md_user(), list_md_active_accounts(), list_md_user_instance(), list_md_user_tokens(), show_current_user()

Examples

```
## Not run:
configure_md_user_settings(
  user_name = "alice",
  token_type = "read_write",
  instance_size = "pulse",
  flock_size = 10
)
```

```
## End(Not run)
```

`connect_to_motherduck` *Create connection to motherduck*

Description

Establishes a connection to a MotherDuck account using DuckDB and the MotherDuck extension. The function handles token validation, database file creation, extension loading, and executes `PRAGMA MD_CONNECT` to authenticate the connection.

Usage

```
connect_to_motherduck(db_path = NULL, config)
```

Arguments

`db_path`	Character, optional. Path to a DuckDB database file or directory to use. If `NULL`, a temporary file is used.
`config`	List, optional. A list of DuckDB configuration options to be passed to `duckdb::duckdb()`.

Details

This function provides a convenient interface for connecting to MotherDuck. It:

- Uses the token from the `MOTHERDUCK_TOKEN` environment variable.
- Optionally specifies a persistent DuckDB database file or directory via `db_path`.
- Optionally provides custom DuckDB configuration options via `config`.
- Automatically loads the MotherDuck extension if not already loaded.

You must set the `MOTHERDUCK_TOKEN` environment variable in your `.Renviron` file before calling this function. Use `usethis::edit_r_environ()` to edit your `.Renviron` file.

If `db_path` is not supplied, a temporary DuckDB database file will be created in the session's temporary directory. Use `config` to pass any DuckDB-specific options (e.g., memory limits or extensions).

MotherDuck currently supports DuckDB versions 1.3.0 through 1.4.4. This function will warn if you're using an untested version.

Value

A `DBIConnection` object connected to your MotherDuck account.

Examples

```
## Not run:
# Connect using a token stored in your .Renviron
con <- connect_to_motherduck()

# Connect and specify a persistent database file
con <- connect_to_motherduck(db_path = "~/my_database.duckdb")

## End(Not run)
```

`copy_tables_to_new_location` *Copy Tables to a New Database/Schema*

Description

Copies one or more tables to a new location (database/schema) by creating new tables via `CREATE TABLE ... AS SELECT * FR`
Requires motherduck connection

Usage

```
copy_tables_to_new_location(
  .con,
  from_table_names,
  to_database_name,
  to_schema_name
)
```

Arguments

`.con`	A valid `DBI` connection (DuckDB / MotherDuck).
`from_table_names`	A tibble/data frame listing source tables, with columns `database_name`, `schema_name`, and `table_name`.
`to_database_name`	Target database name.
`to_schema_name`	Target schema name.

Details

- Input `from_table_names` must contain columns: `database_name`, `schema_name`, and `table_name`.
- For each source table, the function issues: `CREATE TABLE <to_db>.<to_schema>.<table> AS SELECT * FROM <src_c`
- On local DuckDB (non-MotherDuck), the target database name is ignored and defaults to the current database of the connection.

Value

Invisibly returns a character vector of fully-qualified destination table names that were created. Side effect: creates target DB/schema if needed and writes new tables.

See Also

Other db-manage: alter_table_schema(), create_database(), create_if_not_exists_share(), create_or_replace_share(), create_schema(), create_table(), delete_and_create_schema(), delete_database(), delete_schema(), delete_table(), describe_share(), drop_share(), list_owned_shares(), list_shared_with_me_shares(), upload_database_to_md()

---

create_database *Create (If Not Exists) and Switch to a Database*

Description

Ensures a database exists and sets it as the active database. If connected to MotherDuck, the function will run CREATE DATABASE IF NOT EXISTS followed by USE <database>. Prints CLI status information about the current user and database.

Usage

```
create_database(.con, database_name)
```

Arguments

.con	A valid DBI connection (DuckDB / MotherDuck).
database_name	Name of the database to create/ensure and switch to

Details

- Connection type is checked via validate_md_connection_status() (with return_type = "arg").
- On MotherDuck, executes:
  - CREATE DATABASE IF NOT EXISTS <database>
  - USE <database>
- Displays status and environment info with CLI messages.

Value

Invisibly returns NULL. Side effect: may create a database and switches to it; prints CLI status

See Also

Other db-manage: alter_table_schema(), copy_tables_to_new_location(), create_if_not_exists_share(), create_or_replace_share(), create_schema(), create_table(), delete_and_create_schema(), delete_database(), delete_schema(), delete_table(), describe_share(), drop_share(), list_owned_shares(), list_shared_with_me_shares(), upload_database_to_md()

Examples

```
## Not run:
con_md <- connect_to_motherduck()
create_database(con_md, "analytics")

## End(Not run)
```

`create_if_not_exists_share` *Create a MotherDuck database share if it does not exist*

Description

Creates a new share for a specified database in MotherDuck if it does not already exist. Allows you to configure access, visibility, and update settings for the share.

Usage

```
create_if_not_exists_share(
  .con,
  share_name,
  database_name,
  access = "PUBLIC",
  visibility = "LISTED",
  update = "AUTOMATIC"
)
```

Arguments

`.con`	A valid `DBI` connection (DuckDB / MotherDuck).
`share_name`	Character. The name of the new share to create.
`database_name`	Character. The name of the target database to share.
`access`	Character. Access level for the share; either `"RESTRICTED"` or `"PUBLIC"` (default: `"PUBLIC"`).
`visibility`	Character. Visibility of the share; either `"HIDDEN"` or `"LISTED"` (default: `"LISTED"`).
`update`	Character. Update policy for the share; either `"AUTOMATIC"` or `"MANUAL"` (default: `"AUTOMATIC"`).

Details

This function executes a `CREATE IF NOT EXISTS` SQL statement on the connected MotherDuck database to create a share for the specified database.

- `access` controls who can access the share.
- `visibility` controls whether the share is listed publicly or hidden.
- `update` controls whether changes to the source database are automatically reflected in the share. After creation, the current user is displayed for confirmation.

Value

A message confirming that the share has been created, if it did not already exist.

See Also

Other db-manage: alter_table_schema(), copy_tables_to_new_location(), create_database(), create_or_replace_share(), create_schema(), create_table(), delete_and_create_schema(), delete_database(), delete_schema(), delete_table(), describe_share(), drop_share(), list_owned_shares(), list_shared_with_me_shares(), upload_database_to_md()

Examples

```
## Not run:
con <- DBI::dbConnect(duckdb::duckdb(dbdir = tempfile()))

create_if_not_exists_share(
  .con = con,
  share_name = "analytics_share",
  database_name = "sales_db",
  access = "PUBLIC",
  visibility = "LISTED",
  update = "AUTOMATIC"
)

## End(Not run)
```

`create_md_access_token` *Create a MotherDuck access token*

Description

Creates a new access token for a specified MotherDuck user using the REST API. Tokens can be configured with a specific type, name, and expiration time.

Usage

```
create_md_access_token(
  user_name,
  token_type,
  token_name,
  token_expiration_number,
  token_expiration_unit
)
```

Arguments

user_name	A character string specifying the MotherDuck user name whose tokens should be listed.
token_type	Character. The type of token to create. Must be one of: "read_write" or "read_scaling".
token_name	Character. A descriptive name for the token.
token_expiration_number	Numeric. The duration of the token's validity, in the units specified by token_expiration_unit. Minimum value is 300 seconds.
token_expiration_unit	Character. The unit of time for the token expiration. One of "seconds", "minutes", "days", "weeks", "months", "years", or "never".

Details

This function calls the MotherDuck REST API endpoint https://api.motherduck.com/v1/users/{user_name}/tokens to create a new token for the specified user. The token's time-to-live (TTL) is calculated in seconds from token_expiration_number and token_expiration_unit.

Uses the token from the MOTHERDUCK_TOKEN environment variable for authorization. The authenticated user must have administrative privileges.

Value

A tibble containing the API response, including the username and the token attributes.

See Also

Other db-api: configure_md_user_settings(), create_md_user(), delete_md_access_token(), delete_md_user(), list_md_active_accounts(), list_md_user_instance(), list_md_user_tokens(), show_current_user()

Examples

```
## Not run:
# Create a temporary read/write token for user "alejandro_hagan" valid for 1 hour
create_md_access_token(
  user_name = "alejandro_hagan",
  token_type = "read_write",
  token_name = "temp_token",
  token_expiration_number = 1,
  token_expiration_unit = "hours"
)

## End(Not run)
```

`create_md_user` *Create a new MotherDuck user*

Description

Sends a `POST` request to the MotherDuck REST API to create a new user within your organization. This operation requires administrative privileges.

Usage

```
create_md_user(user_name)
```

Arguments

`user_name`	A character string specifying the MotherDuck user name whose tokens should be listed.

Details

This function calls the MotherDuck Users API endpoint to create a new user under the authenticated account. The token from the `MOTHERDUCK_TOKEN` environment variable must belong to a user with permissions to manage organization-level accounts.

Value

A tibble summarizing the API response, typically containing the newly created username and associated metadata.

See Also

Other db-api: `configure_md_user_settings()`, `create_md_access_token()`, `delete_md_access_token()`, `delete_md_user()`, `list_md_active_accounts()`, `list_md_user_instance()`, `list_md_user_tokens()`, `show_current_user()`

Examples

```
## Not run:
# Create a new user in MotherDuck
create_md_user("new_user_name")

## End(Not run)
```

create_or_replace_share *Create or replace a MotherDuck database share*

Description

Creates a new share or replaces an existing share for a specified database in MotherDuck. This allows you to update the configuration of an existing share or create a new one if it does not exist.

Usage

```
create_or_replace_share(
  .con,
  share_name,
  database_name,
  access = "PUBLIC",
  visibility = "LISTED",
  update = "AUTOMATIC"
)
```

Arguments

.con	A valid DBI connection (DuckDB / MotherDuck).
share_name	Character. The name of the share to create or replace.
database_name	Character. The name of the database to be shared.
access	Character. Access level for the share; either "RESTRICTED" or "PUBLIC" (default: "PUBLIC").
visibility	Character. Visibility of the share; either "HIDDEN" or "LISTED" (default: "LISTED").
update	Character. Update policy for the share; either "AUTOMATIC" or "MANUAL" (default: "AUTOMATIC").

Details

This function executes a CREATE OR REPLACE SHARE SQL statement to create a new share or update an existing one.

- access controls who can access the share.
- visibility controls whether the share is listed publicly or hidden.
- update controls whether changes to the source database are automatically reflected in the share. The current user is displayed for confirmation before execution.

Value

A message indicating that the share has been created or replaced.

See Also

Other db-manage: alter_table_schema(), copy_tables_to_new_location(), create_database(), create_if_not_exists_share(), create_schema(), create_table(), delete_and_create_schema(), delete_database(), delete_schema(), delete_table(), describe_share(), drop_share(), list_owned_shares(), list_shared_with_me_shares(), upload_database_to_md()

Examples

```
## Not run:
con <- DBI::dbConnect(duckdb::duckdb(dbdir = tempfile()))
create_or_replace_share(
  .con = con,
  share_name = "analytics_share",
  database_name = "sales_db",
  access = "PUBLIC",
  visibility = "LISTED",
  update = "AUTOMATIC"
)

## End(Not run)
```

---

create_r2_secret *Register a Cloudflare R2 secret on a connection*

Description

Creates (or replaces) a DuckDB/MotherDuck secret of TYPE R2 so subsequent reads from r2:// paths authenticate against your Cloudflare R2 bucket. R2 is S3-compatible; this is a convenience wrapper over the generic create_s3_secret() that derives the endpoint from your account id.

Usage

```
create_r2_secret(
  .con,
  account_id,
  key_id,
  secret,
  name = "r2",
  persistent = FALSE
)
```

Arguments

.con	A valid DBI connection (DuckDB / MotherDuck).
account_id	Cloudflare R2 account id (the 32-character hex id).
key_id	R2 S3 access key id.

secret	R2 S3 secret access key.
name	Secret name. Defaults to "r2".
persistent	Logical; persist the secret to the local store. Defaults to FALSE (session-scoped).

Details

Requires the httpfs extension (load it first with load_extensions(.con, "httpfs")). The secret is session-scoped by default; pass persistent = TRUE to persist it to the local secret store. On MotherDuck the secret lives in the active session and is used by the cloud engine to fetch the objects.

Value

Invisibly, the secret name.

See Also

create_s3_secret(), load_extensions()

Other db-manage: alter_table_schema(), copy_tables_to_new_location(), create_database(), create_if_not_exists_share(), create_or_replace_share(), create_s3_secret(), create_schema(), create_table(), delete_and_create_schema(), delete_database(), delete_schema(), delete_table(), describe_share(), drop_share(), list_owned_shares(), list_shared_with_me_shares(), upload_database_to_md()

Examples

```
## Not run:
con <- connect_to_motherduck()
load_extensions(con, "httpfs")
create_r2_secret(
  con,
  account_id = Sys.getenv("R2_ACCOUNT_ID"),
  key_id     = Sys.getenv("AWS_ACCESS_KEY_ID"),
  secret     = Sys.getenv("AWS_SECRET_ACCESS_KEY")
)
DBI::dbGetQuery(con, "SELECT * FROM read_csv_auto('r2://my-bucket/data.csv')")

## End(Not run)
```

---

create_s3_secret *Register a generic S3 (S3-compatible) secret on a connection*

Description

Creates (or replaces) a DuckDB/MotherDuck secret of TYPE S3 for AWS S3 or any S3-compatible object store (Cloudflare R2, MinIO, Backblaze B2, etc.). For Cloudflare R2 prefer the create_r2_secret() shortcut.

Usage

```
create_s3_secret(
  .con,
  key_id,
  secret,
  endpoint = NULL,
  region = NULL,
  url_style = NULL,
  use_ssl = NULL,
  name = "s3",
  persistent = FALSE
)
```

Arguments

.con	A valid DBI connection (DuckDB / MotherDuck).
key_id	R2 S3 access key id.
secret	R2 S3 secret access key.
endpoint	Optional host of the S3-compatible endpoint, e.g. "<account>.r2.cloudflarestorage.com". Omit for AWS S3.
region	Optional region, e.g. "us-east-1". R2 ignores this.
url_style	Optional "path" or "vhost". S3-compatible stores usually need "path".
use_ssl	Optional logical; use HTTPS. Defaults to NULL (engine default).
name	Secret name. Defaults to "s3".
persistent	Logical; persist the secret to the local store. Defaults to FALSE (session-scoped).

Details

Requires the httpfs extension. Optional fields (endpoint, region, url_style, use_ssl) are only emitted when supplied, so the same helper covers AWS (region only) and self-hosted stores (custom endpoint + url_style = "path").

Value

Invisibly, the secret name.

See Also

create_r2_secret(), load_extensions()

Other db-manage: alter_table_schema(), copy_tables_to_new_location(), create_database(), create_if_not_exists_share(), create_or_replace_share(), create_r2_secret(), create_schema(), create_table(), delete_and_create_schema(), delete_database(), delete_schema(), delete_table(), describe_share(), drop_share(), list_owned_shares(), list_shared_with_me_shares(), upload_database_to_md()

Examples

```
## Not run:
con <- connect_to_motherduck()
load_extensions(con, "httpfs")
create_s3_secret(con, key_id = "AKIA...", secret = "...", region = "us-east-1")

## End(Not run)
```

`create_schema` *Create a Schema in a Database if It Does Not Exist*

Description

Ensures that a specified schema exists in the given database. If the connection is to a MotherDuck instance, the function switches to the specified database before creating the schema. It also prints helpful connection and environment information via CLI messages for transparency.

Usage

```
create_schema(.con, database_name, schema_name)
```

Arguments

`.con`	A valid `DBI` connection (DuckDB / MotherDuck).
`database_name`	Name of the database to create/use.
`schema_name`	Name of the schema to create if it does not exist.

Details

- Uses `DBI::dbExecute()` with `CREATE SCHEMA IF NOT EXISTS` to create the schema only when needed.
- If connected to MotherDuck (determined by `validate_md_connection_status()`), executes `USE <database>` before creating the schema.
- Displays connection/user/database information via internal CLI helpers.

Value

Invisibly returns `NULL`. Side effect: creates the schema if necessary and prints CLI messages.

See Also

Other db-manage: `alter_table_schema()`, `copy_tables_to_new_location()`, `create_database()`, `create_if_not_exists_share()`, `create_or_replace_share()`, `create_table()`, `delete_and_create_schema()`, `delete_database()`, `delete_schema()`, `delete_table()`, `describe_share()`, `drop_share()`, `list_owned_shares()`, `list_shared_with_me_shares()`, `upload_database_to_md()`

`create_table` *Create or Append a Table from a Tibble or DBI-Backed Table*

Description

A thin wrapper that routes to either `create_table_dbi()` (for dbplyr-backed lazy tables, class `"tbl_dbi"`) or `create_table_tbl()` (for in-memory tibbles / data frames), creating a physical table in the target database/schema. Supports **overwrite** and **append** write strategies and defers all heavy lifting to the specific implementation.

Usage

```
create_table(
  .data,
  .con,
  database_name,
  schema_name,
  table_name,
  write_type = "overwrite"
)
```

Arguments

`.data`	Tibble/data frame (in-memory) or a dbplyr/DBI-backed lazy table (class `"tbl_dbi"`).
`.con`	A valid DBI connection (DuckDB / MotherDuck).
`database_name`	Database name to create/use.
`schema_name`	Schema name to create/use.
`table_name`	Target table name to create or append to.
`write_type`	Write strategy: `"overwrite"` (drop/create) or `"append"` (insert rows). Defaults to `"overwrite"`.

Details

- If `.data` is a dbplyr lazy table (class `"tbl_dbi"`), the call is delegated to `create_table_dbi()`.
- If `.data` is an in-memory tibble/data frame (class including `"data.frame"`), the call is delegated to `create_table_tbl()`.
- Any other input classes trigger an error.

Value

Invisibly returns `NULL`. Side effect: writes a table to the database by delegating to the appropriate helper.

See Also

Other db-manage: alter_table_schema(), copy_tables_to_new_location(), create_database(), create_if_not_exists_share(), create_or_replace_share(), create_schema(), delete_and_create_schema(), delete_database(), delete_schema(), delete_table(), describe_share(), drop_share(), list_owned_shares(), list_shared_with_me_shares(), upload_database_to_md()

`delete_and_create_schema` *Drop and Recreate a Schema in a MotherDuck / DuckDB Database*

Description

Drops an existing schema (if it exists) in the specified database and then creates a new empty schema. If the connection is to a MotherDuck instance, the function switches to the given database first, then drops and recreates the schema. Displays helpful CLI output about the current connection, user, and database.

Usage

```
delete_and_create_schema(.con, database_name, schema_name)
```

Arguments

`.con`	A valid `DBI` connection (DuckDB / MotherDuck).
`database_name`	The name of the database where the schema should be dropped and recreated.
`schema_name`	The name of the schema to drop and recreate.

Details

- Executes `DROP SCHEMA IF EXISTS ... CASCADE` to remove an existing schema and all contained objects.
- Executes `CREATE SCHEMA IF NOT EXISTS` to recreate it.
- If connected to MotherDuck (detected by `validate_md_connection_status()`), performs a `USE <database>` first.
- Prints a summary of the current connection and schema creation status using internal CLI helpers.

Value

Invisibly returns `NULL`. Side effect: drops and recreates the schema and prints CLI status messages.

See Also

Other db-manage: alter_table_schema(), copy_tables_to_new_location(), create_database(), create_if_not_exists_share(), create_or_replace_share(), create_schema(), create_table(), delete_database(), delete_schema(), delete_table(), describe_share(), drop_share(), list_owned_shares(), list_shared_with_me_shares(), upload_database_to_md()

delete_database *Drop a Database*

Description

Drops a database from the current DuckDB or MotherDuck connection if it exists. Prints a CLI status report after performing the operation.

Usage

```
delete_database(.con, database_name)
```

Arguments

.con	A valid DBI connection (DuckDB / MotherDuck).
database_name	Name of the database to drop.

Details

- Executes DROP DATABASE IF EXISTS <database_name> to remove the database.
- Intended for DuckDB or MotherDuck connections.
- Prints user, database and action details using CLI helper functions.

Value

Invisibly returns NULL. Side effect: drops the database and prints CLI status messages.

See Also

Other db-manage: alter_table_schema(), copy_tables_to_new_location(), create_database(), create_if_not_exists_share(), create_or_replace_share(), create_schema(), create_table(), delete_and_create_schema(), delete_schema(), delete_table(), describe_share(), drop_share(), list_owned_shares(), list_shared_with_me_shares(), upload_database_to_md()

delete_md_access_token *Delete a MotherDuck user's access token*

Description

Deletes a specific access token for a given MotherDuck user using the REST API. This operation requires administrative privileges.

Usage

```
delete_md_access_token(user_name, token_name)
```

Arguments

user_name	A character string specifying the MotherDuck user name whose tokens should be listed.
token_name	Character. The name of the access token to delete.

Details

This function calls the MotherDuck REST API endpoint `https://api.motherduck.com/v1/users/{user_name}/tokens/` using a `DELETE` request to remove the specified token.

Uses the token from the `MOTHERDUCK_TOKEN` environment variable for authorization. The authenticated user must have sufficient permissions to perform token management.

Value

A tibble summarizing the API response, typically including the username and deletion status of the token.

See Also

Other db-api: `configure_md_user_settings()`, `create_md_access_token()`, `create_md_user()`, `delete_md_user()`, `list_md_active_accounts()`, `list_md_user_instance()`, `list_md_user_tokens()`, `show_current_user()`

Examples

```
## Not run:
# Delete a token named "temp_token" for user "alejandro_hagan"
delete_md_access_token(
  user_name = "alejandro_hagan",
  token_name = "temp_token"
)

## End(Not run)
```

---

delete_md_user *Delete a MotherDuck user*

Description

Sends a `DELETE` request to the MotherDuck REST API to permanently remove a user from your organization. This operation requires administrative privileges.

Usage

```
delete_md_user(user_name)
```

Arguments

user_name	A character string specifying the MotherDuck user name whose tokens should be listed.

Details

This function calls the MotherDuck Users API endpoint to delete the specified user. The authenticated user (via the MOTHERDUCK_TOKEN environment variable) must have sufficient permissions to perform user management actions.

Value

A tibble summarizing the API response, including the username and deletion status.

See Also

Other db-api: configure_md_user_settings(), create_md_access_token(), create_md_user(), delete_md_access_token(), list_md_active_accounts(), list_md_user_instance(), list_md_user_tokens(), show_current_user()

Examples

```
## Not run:
# Delete a user named "bob_smith"
delete_md_user("bob_smith")

## End(Not run)
```

---

delete_schema *Drop a Schema from a Database*

Description

Drops a schema from a specified database. Optionally cascades the deletion to all objects within the schema. Prints helpful CLI information about the current connection and action.

Usage

```
delete_schema(.con, database_name, schema_name, cascade = FALSE)
```

Arguments

.con	A valid DBI connection (DuckDB / MotherDuck).
database_name	Name of the database containing the schema.
schema_name	Name of the schema to drop.
cascade	Logical; if TRUE (default), use CASCADE to drop all dependent objects in the schema. If FALSE, drop only if empty.

Details

- Runs DROP SCHEMA IF EXISTS <db>.<schema> with optional CASCADE.
- Intended for DuckDB or MotherDuck connections.
- Uses CLI helpers to show current connection and report the deletion.

Value

Invisibly returns NULL. Side effect: drops the schema (and contained objects if cascade = TRUE) and prints CLI status.

See Also

Other db-manage: alter_table_schema(), copy_tables_to_new_location(), create_database(), create_if_not_exists_share(), create_or_replace_share(), create_schema(), create_table(), delete_and_create_schema(), delete_database(), delete_table(), describe_share(), drop_share(), list_owned_shares(), list_shared_with_me_shares(), upload_database_to_md()

---

delete_table *Drop a Table*

Description

Drops a table from the specified database and schema if it exists. Uses DROP TABLE IF EXISTS for safety and prints a CLI status report.

Usage

```
delete_table(.con, database_name, schema_name, table_name)
```

Arguments

.con	A valid DBI connection (DuckDB / MotherDuck).
database_name	Name of the database containing the table.
schema_name	Name of the schema containing the table.
table_name	Name of the table to drop.

Value

Invisibly returns NULL. Side effect: drops the table and prints CLI status messages.

See Also

Other db-manage: alter_table_schema(), copy_tables_to_new_location(), create_database(), create_if_not_exists_share(), create_or_replace_share(), create_schema(), create_table(), delete_and_create_schema(), delete_database(), delete_schema(), describe_share(), drop_share(), list_owned_shares(), list_shared_with_me_shares(), upload_database_to_md()

describe_share *Describe a MotherDuck share*

Description

Retrieves detailed metadata about a specific share in MotherDuck, including the objects it contains, their types, and privileges granted.

Usage

```
describe_share(.con, share_name)
```

Arguments

.con	A valid DBI connection (DuckDB / MotherDuck).
share_name	Character. The name of the shared path to describe.

Details

This function executes the md_describe_database_share system function to obtain comprehensive information about the specified share. The result is returned as a tibble for easy inspection and manipulation in R.

Value

A tibble containing metadata about the share, including object names, types, and privileges associated with the share.

See Also

Other db-manage: alter_table_schema(), copy_tables_to_new_location(), create_database(), create_if_not_exists_share(), create_or_replace_share(), create_schema(), create_table(), delete_and_create_schema(), delete_database(), delete_schema(), delete_table(), drop_share(), list_owned_shares(), list_shared_with_me_shares(), upload_database_to_md()

Examples

```
## Not run:
con <- DBI::dbConnect(duckdb::duckdb(dbdir = tempfile()))
share_info <- describe_share(con, "analytics.sales_share")
print(share_info)

## End(Not run)
```

drop_share *Drop a MotherDuck share*

Description

Drops (deletes) a specified share from your MotherDuck account. If the share does not exist, a warning is displayed. This function safely validates the connection and share name before executing the operation.

Usage

```
drop_share(.con, share_name)
```

Arguments

.con	A valid DBI connection (DuckDB / MotherDuck).
share_name	Character. The name of the share to be dropped.

Details

The function first validates that the connection is active. It then checks whether the specified share exists in the account. If it does, the share is dropped using a DROP SHARE IF EXISTS SQL command. If the share does not exist, a warning is shown. After the operation, the current user is displayed.

Value

Invisibly returns NULL. Side effect: the specified share is removed if it exists.

See Also

Other db-manage: alter_table_schema(), copy_tables_to_new_location(), create_database(), create_if_not_exists_share(), create_or_replace_share(), create_schema(), create_table(), delete_and_create_schema(), delete_database(), delete_schema(), delete_table(), describe_share(), list_owned_shares(), list_shared_with_me_shares(), upload_database_to_md()

Examples

```
## Not run:
drop_share(con_md, "test_share")

## End(Not run)
```

`install_extensions` *Install DuckDB/MotherDuck Extensions*

Description

Installs valid DuckDB or MotherDuck extensions for the current connection.

Usage

```
install_extensions(.con, extension_names)
```

Arguments

`.con`	A valid `DBI` connection (DuckDB / MotherDuck).
`extension_names`	A character vector of DuckDB/MotherDuck extension names to install.

Details

The `install_extensions()` function validates the provided DuckDB/MotherDuck connection, then checks which of the requested extensions are valid. Valid extensions that are not already installed are installed using the `INSTALL` SQL command. Invalid extensions are reported to the user via CLI messages. This function provides a summary report describing which extensions were successfully installed and which were invalid.

Unlike `load_extensions()`, this function focuses purely on installation and does not automatically load extensions after installing.

Value

Invisibly returns `NULL`. A detailed CLI report of installation success/failure is printed.

See Also

Other db-con: `load_extensions()`, `show_motherduck_token()`, `validate_extension_install_status()`, `validate_extension_load_status()`

Examples

```
## Not run:
con <- DBI::dbConnect(duckdb::duckdb(dbdir = tempfile()))

# Install the 'motherduck' extension
install_extensions(con, "motherduck")

# Install multiple extensions
install_extensions(con, c("fts", "httpfs"))

DBI::dbDisconnect(con)
```

```
## End(Not run)
```

---

launch_ui *Launch the DuckDB UI in your browser*

Description

The `launch_ui()` function installs and launches the DuckDB UI extension for an active DuckDB database connection. This allows users to interact with the database via a web-based graphical interface.

The function will check that the connection is valid before proceeding.

Usage

```
launch_ui(.con)
```

Arguments

`.con`	A valid `DBI` connection (DuckDB / MotherDuck).

Details

The function performs the following steps:

- Checks that the provided DuckDB connection is valid. If the connection is invalid, it aborts with a descriptive error message.
- Installs the `ui` extension into the connected DuckDB instance.
- Calls the `start_ui()` procedure to launch the DuckDB UI in your browser.

This provides a convenient way to explore and manage DuckDB databases interactively without needing to leave the R environment.

Value

The function is called for its side effects and does not return a value. It launches the DuckDB UI and opens it in your default web browser.

See Also

Other db-meta: `cd()`, `pwd()`, `summary()`

Examples

```
## Not run:
# Connect to DuckDB
con_db <- DBI::dbConnect(duckdb::duckdb(dbdir = tempfile()))

# Launch the DuckDB UI
launch_ui(con_db)

# Clean up
DBI::dbDisconnect(con_db, shutdown = TRUE)

## End(Not run)
```

---

list_all_databases *List Databases Visible to the Connection*

Description

Returns a lazy tibble of distinct database (catalog) names visible through the current connection, using `information_schema.tables`.

Usage

```
list_all_databases(.con)
```

Arguments

`.con`	A valid `DBI` connection (DuckDB / MotherDuck).

Details

The result is a `dbplyr` lazy table (`tbl_dbi`). Use `dplyr::collect()` to bring results into R as a local tibble.

Value

A `dbplyr` lazy tibble with one column: `table_catalog`.

See Also

Other db-list: `list_all_tables()`, `list_current_schemas()`, `list_current_tables()`, `list_extensions()`, `list_fns()`, `list_setting()`, `list_shares()`

list_all_tables *List All Tables Visible to the Connection*

Description

Returns a lazy tibble of all tables visible to the current connection by querying `information_schema.tables` (across all catalogs/databases and schemas).

Usage

```
list_all_tables(.con)
```

Arguments

`.con`	A valid `DBI` connection (DuckDB / MotherDuck).

Details

The result is a `dbplyr` lazy table (`tbl_dbi`). Use `collect()` to bring results into R as a local tibble.

Value

A `dbplyr` lazy tibble with columns:

- `table_catalog` — database/catalog name
- `table_schema` — schema name
- `table_name` — table name

See Also

Other db-list: `list_all_databases()`, `list_current_schemas()`, `list_current_tables()`, `list_extensions()`, `list_fns()`, `list_setting()`, `list_shares()`

list_current_schemas *List Schemas in the Current Database*

Description

Returns a lazy tibble of all schemas in the **current database** of the connection. Queries `information_schema.schemata` and filters to the current database (`catalog_name = current_database()`).

Usage

```
list_current_schemas(.con)
```

Arguments

.con	A valid DBI connection (DuckDB / MotherDuck).

Details

- This function assumes the connection is valid (checked with validate_con()).
- Returns a dbplyr lazy table; use collect() to bring the result into R.

Value

A dbplyr lazy tibble with columns:

- catalog_name — the current database name.
- schema_name — each schema within that database.

See Also

Other db-list: list_all_databases(), list_all_tables(), list_current_tables(), list_extensions(), list_fns(), list_setting(), list_shares()

---

list_current_tables *List Tables in the Current Database and Schema*

Description

Returns a lazy tibble of all tables that exist in the **current database** and **current schema** of the active connection. Queries the standard information_schema.tables view and filters to current_database() and current_schema().

Usage

```
list_current_tables(.con)
```

Arguments

.con	A valid DBI connection (DuckDB / MotherDuck).

Details

- This function validates that the connection is valid with validate_con().
- Result is a dbplyr lazy table (tbl_dbi); call collect() to bring it into R.

Value

A dbplyr lazy tibble with columns:

- table_catalog — the current database.
- table_schema — the current schema.
- table_name — each table name.

See Also

Other db-list: `list_all_databases()`, `list_all_tables()`, `list_current_schemas()`, `list_extensions()`, `list_fns()`, `list_setting()`, `list_shares()`

`list_extensions` *List MotherDuck/DuckDB Extensions*

Description

Retrieves all available DuckDB or MotherDuck extensions along with their descriptions, installation and load status.

Usage

```
list_extensions(.con)
```

Arguments

`.con`	A valid `DBI` connection (DuckDB / MotherDuck).

Details

The `list_extensions()` function queries the database for all extensions that are available in the current DuckDB or MotherDuck connection. The returned tibble includes information such as:

- `extension_name`: Name of the extension.
- `description`: Short description of the extension.
- `installed`: Logical indicating if the extension is installed.
- `loaded`: Logical indicating if the extension is currently loaded.

This is useful for determining which extensions can be installed or loaded using `install_extensions()` or `load_extensions()`.

Value

A tibble with one row per extension and columns describing its metadata and current status.

See Also

Other db-list: `list_all_databases()`, `list_all_tables()`, `list_current_schemas()`, `list_current_tables()`, `list_fns()`, `list_setting()`, `list_shares()`

Examples

```
## Not run:
con <- DBI::dbConnect(duckdb::duckdb(dbdir = tempfile()))

# List all available extensions
list_extensions(con)

DBI::dbDisconnect(con)

## End(Not run)
```

`list_fns` *List Database Functions (DuckDB/MotherDuck)*

Description

Returns a lazy table listing available SQL functions from the current DuckDB/MotherDuck connection using `duckdb_functions()`.

Usage

```
list_fns(.con)
```

Arguments

`.con`	A valid `DBI` connection (DuckDB / MotherDuck).

Details

This wrapper validates the connection and then queries `duckdb_functions()` to enumerate function metadata. The result is a `dbplyr` lazy tibble (`tbl_dbi`); call `collect()` to materialize it in R.

Value

A `dbplyr` lazy tibble (`tbl_dbi`) with function metadata (e.g., `function_name`, `schema`, `is_aggregate`, `is_alias`, etc.).

See Also

Other db-list: `list_all_databases()`, `list_all_tables()`, `list_current_schemas()`, `list_current_tables()`, `list_extensions()`, `list_setting()`, `list_shares()`

`list_md_active_accounts` *List active MotherDuck accounts*

Description

Retrieves a list of active MotherDuck accounts available to the authenticated user, returning the results as a tidy tibble.

Usage

```
list_md_active_accounts()
```

Details

This function queries the MotherDuck REST API endpoint (`https://api.motherduck.com/v1/active_accounts`) using the token from the `MOTHERDUCK_TOKEN` environment variable.

Value

A tibble with two columns:

- `account_settings`: configuration keys for the active accounts.
- `account_values`: corresponding configuration values.

See Also

Other db-api: `configure_md_user_settings()`, `create_md_access_token()`, `create_md_user()`, `delete_md_access_token()`, `delete_md_user()`, `list_md_user_instance()`, `list_md_user_tokens()`, `show_current_user()`

Examples

```
## Not run:
# Retrieve active accounts for the authenticated user
accounts_tbl <- list_md_active_accounts()
print(accounts_tbl)

## End(Not run)
```

`list_md_user_instance` *List a MotherDuck user's instance settings*

Description

Retrieves configuration and instance-level settings for a specified MotherDuck user, returning the results as a tidy tibble.

Usage

```
list_md_user_instance(user_name)
```

Arguments

`user_name`	A character string specifying the MotherDuck user name whose tokens should be listed.

Details

This function calls the MotherDuck REST API endpoint `https://api.motherduck.com/v1/users/{user_name}/instanc` to fetch information about the user's active DuckDB instances and their configuration parameters.

Uses the token from the `MOTHERDUCK_TOKEN` environment variable for authorization.

Value

A tibble with two columns:

- `instance_desc`: names or descriptions of instance configuration settings.
- `instance_values`: corresponding values for each configuration field.

See Also

Other db-api: `configure_md_user_settings()`, `create_md_access_token()`, `create_md_user()`, `delete_md_access_token()`, `delete_md_user()`, `list_md_active_accounts()`, `list_md_user_tokens()`, `show_current_user()`

Examples

```
## Not run:
# List instance settings for a specific user
instance_tbl <- list_md_user_instance(user_name = "bob_smith")

## End(Not run)
```

`list_md_user_tokens` *List a MotherDuck user's tokens*

Description

Retrieves all active authentication tokens associated with a specific MotherDuck user account, returning them as a tidy tibble.

Usage

```
list_md_user_tokens(user_name)
```

Arguments

`user_name`	A character string specifying the MotherDuck user name whose tokens should be listed.

Details

This function queries the MotherDuck REST API endpoint `https://api.motherduck.com/v1/users/{user_name}/toke` to list the tokens available for the specified user.

Uses the token from the `MOTHERDUCK_TOKEN` environment variable for authorization.

Value

A tibble with two columns:

- `token_settings`: metadata fields associated with each token.
- `token_values`: corresponding values for those fields.

See Also

Other db-api: `configure_md_user_settings()`, `create_md_access_token()`, `create_md_user()`, `delete_md_access_token()`, `delete_md_user()`, `list_md_active_accounts()`, `list_md_user_instance()`, `show_current_user()`

Examples

```
## Not run:
# List tokens for a specific user
tokens_tbl <- list_md_user_tokens(user_name = "alejandro_hagan")
print(tokens_tbl)

## End(Not run)
```

`list_owned_shares` *List all shares owned by the current user*

Description

Retrieves all database objects that are owned by the current authenticated user in MotherDuck.

Usage

```
list_owned_shares(.con)
```

Arguments

`.con`	A valid `DBI` connection (DuckDB / MotherDuck).

Details

This function executes the `LIST SHARES;` command to return metadata about all shares created by the current user. The returned tibble includes details such as the share name, type of object shared, and privileges granted.

Value

A tibble with one row per share owned by the current user, including columns for share name, object type, and granted privileges.

See Also

Other db-manage: `alter_table_schema()`, `copy_tables_to_new_location()`, `create_database()`, `create_if_not_exists_share()`, `create_or_replace_share()`, `create_schema()`, `create_table()`, `delete_and_create_schema()`, `delete_database()`, `delete_schema()`, `delete_table()`, `describe_share()`, `drop_share()`, `list_shared_with_me_shares()`, `upload_database_to_md()`

Examples

```
## Not run:
con <- DBI::dbConnect(duckdb::duckdb(dbdir = tempfile()))
owned_tbl <- list_owned_shares(con)
print(owned_tbl)

## End(Not run)
```

list_setting *List Database Settings*

Description

The `list_setting()` function provides a convenient way to inspect the active configuration of a DuckDB or MotherDuck connection. It executes the internal DuckDB function `duckdb_settings()` and returns the results as a tidy tibble for easy viewing or filtering.

Usage

```
list_setting(.con)
```

Arguments

`.con`	A valid `DBI` connection (DuckDB / MotherDuck).

Details

This function is particularly useful for debugging or auditing runtime environments. All settings are returned as character columns, including their names, current values, and default values.

Value

A tibble::tibble containing one row per setting with columns describing the setting name, current value, description, and default value.

See Also

Other db-list: `list_all_databases()`, `list_all_tables()`, `list_current_schemas()`, `list_current_tables()`, `list_extensions()`, `list_fns()`, `list_shares()`

Examples

```
## Not run:
# Connect to DuckDB
con <- DBI::dbConnect(duckdb::duckdb(dbdir = tempfile()))

# List all database settings
list_setting(con)

# Disconnect
DBI::dbDisconnect(con)

## End(Not run)
```

`list_shared_with_me_shares` *List all shares shared with the current user*

Description

Retrieves all database objects that have been shared with the current authenticated user in MotherDuck.

Usage

```
list_shared_with_me_shares(.con)
```

Arguments

`.con`	A valid `DBI` connection (DuckDB / MotherDuck).

Details

This function queries the `MD_INFORMATION_SCHEMA.SHARED_WITH_ME` view to return metadata about all shares granted to the current user, including the owner, object name, type, and privileges. The result is returned as a tidy tibble for easy manipulation in R.

Value

A tibble containing one row per shared object, with columns describing the owner, object type, object name, and granted privileges.

See Also

Other db-manage: `alter_table_schema()`, `copy_tables_to_new_location()`, `create_database()`, `create_if_not_exists_share()`, `create_or_replace_share()`, `create_schema()`, `create_table()`, `delete_and_create_schema()`, `delete_database()`, `delete_schema()`, `delete_table()`, `describe_share()`, `drop_share()`, `list_owned_shares()`, `upload_database_to_md()`

Examples

```
## Not run:
con <- DBI::dbConnect(duckdb::duckdb(dbdir = tempfile()))
shared_tbl <- list_shared_with_me_shares(con)
print(shared_tbl)

## End(Not run)
```

`list_shares` *List MotherDuck Shares*

Description

The `list_shares()` function provides a convenient wrapper around the MotherDuck SQL command `LIST SHARES;`. It validates that the supplied connection is an active MotherDuck connection before executing the query. If the connection is not valid, the function returns `0` instead of a table.

Usage

```
list_shares(.con)
```

Arguments

`.con`	A valid `DBI` connection (DuckDB / MotherDuck).

Details

MotherDuck supports object sharing, which allows users to list and access data shared between accounts. This function helps programmatically inspect available shares within an authenticated MotherDuck session.

Value

A tibble::tibble containing details of available shares if the connection is an MD connection or an empty tibble if not

See Also

Other db-list: `list_all_databases()`, `list_all_tables()`, `list_current_schemas()`, `list_current_tables()`, `list_extensions()`, `list_fns()`, `list_setting()`

Examples

```
## Not run:
# Connect to MotherDuck
con <- DBI::dbConnect(duckdb::duckdb(dbdir = tempfile()))

# List shares
list_shares(con)

# Disconnect
DBI::dbDisconnect(con)

## End(Not run)
```

load_extensions *Load and Install DuckDB/MotherDuck Extensions*

Description

Installs (if necessary) and loads valid DuckDB or MotherDuck extensions for the active connection.

Usage

```
load_extensions(.con, extension_names)
```

Arguments

.con	A valid DBI connection (DuckDB / MotherDuck).
extension_names	A character vector of DuckDB/MotherDuck extension names to load/install.

Details

The load_extensions() function first validates the provided DuckDB/MotherDuck connection, then checks which of the requested extensions are valid and not already installed. Valid extensions are installed and loaded into the current session. Invalid extensions are reported to the user. The function provides a detailed CLI report summarizing which extensions were successfully installed and loaded, and which were invalid.

It is especially useful for ensuring that required extensions, such as motherduck, are available in your database session. The CLI messages also provide guidance on listing all available extensions and installing additional DuckDB extensions.

Value

Invisibly returns NULL. The function prints a CLI report of the extension status.

See Also

Other db-con: install_extensions(), show_motherduck_token(), validate_extension_install_status(), validate_extension_load_status()

Examples

```
## Not run:
con <- DBI::dbConnect(duckdb::duckdb(dbdir = tempfile()))

# Install and load the 'motherduck' extension
load_extensions(con, "motherduck")

# Load multiple extensions
load_extensions(con, c("motherduck", "httpfs"))
```

```
DBI::dbDisconnect(con)

## End(Not run)
```

---

pwd *Print Current MotherDuck Database Context*

Description

Displays the current database, schema, and role for the active DuckDB/MotherDuck connection. This mirrors the behavior of pwd in Linux by showing your current "working database."

Usage

```
pwd(.con, silent = FALSE)
```

Arguments

.con	A valid DBI connection (DuckDB / MotherDuck).
silent	to print message or not, default is FALSE

Details

The pwd() function is a helper for inspecting the current database context of a DuckDB or MotherDuck connection. It queries the database for the current database, schema, and role. The database and schema are returned as a tibble for easy programmatic access, while the role is displayed using a CLI alert. This is especially useful in multi-database environments or when working with different user roles, providing a quick way to verify where SQL queries will be executed.

Value

A tibble with columns:

current_database The active database name.

current_schema The active schema name.

The current role is printed to the console via cli.

See Also

Other db-meta: cd(), launch_ui(), summary()

Examples

```
## Not run:
con <- DBI::dbConnect(duckdb::duckdb(dbdir = tempfile()))
pwd(con)

## End(Not run)
```

read_csv *Read a CSV file into a DuckDB/MotherDuck table*

Description

Loads the DuckDB **excel** extension and creates a table from a CSV file using the `read_csv_auto()` table function. The destination is fully qualified as `<database>.<schema>.<table>`. Only the options you supply are forwarded to `read_csv_auto()` (e.g., `header`, `all_varchar`, `sample_size`, `names`, `types`, `skip`, `union_by_name`, `normalize_names`, `allow_quoted_nulls`, `ignore_errors`). If `names` or `types` are not supplied, they are ignored. See the DuckDB read_csv_auto() documentation for more information.

Usage

```
read_csv(
  .con,
  to_database_name,
  to_schema_name,
  to_table_name,
  file_path,
  header,
  all_varchar,
  sample_size,
  names,
  types,
  skip,
  union_by_name,
  normalize_names,
  allow_quoted_nulls,
  ignore_errors,
  write_type,
  ...
)
```

Arguments

`.con`	A valid `DBI` connection (DuckDB / MotherDuck).
`to_database_name`	Target database name (new or existing)
`to_schema_name`	Target schema name (new or existing)
`to_table_name`	Target table name to create (new or existing)
`file_path`	Path to the Excel file (`.xlsx`)
`header`	Logical; if `TRUE`, first row is header
`all_varchar`	Logical; coerce all columns to `VARCHAR`
`sample_size`	Numeric; number of rows used for type inference

names	Character vector; optional column names to assign instead of reading from the file
types	Named or unnamed character vector; column types (named preferred, unnamed paired to names)
skip	Integer; number of rows to skip at the beginning of the file
union_by_name	Logical; union multiple CSVs by column name
normalize_names	Logical; normalize column names (lowercase, replace spaces)
allow_quoted_nulls	Logical; treat "NULL" in quotes as NULL
ignore_errors	Logical; continue on row parse errors
write_type	Character; either "overwrite" or "append", controls table creation behavior
...	Additional arguments passed to read_csv_auto() in format listed in duckdb documentation (optional)

Value

Invisibly returns NULL. Side effect: creates <database>.<schema>.<table> with the CSV data

See Also

Other db-read: read_excel()

---

read_excel *Read an Excel file into a DuckDB/MotherDuck table*

Description

Loads the DuckDB **excel** extension and creates a table from an Excel file using the read_xlsx() table function. The destination is fully qualified as <database>.<schema>.<table>. Only the options you supply are forwarded to read_xlsx() (e.g., sheet, header, all_varchar, ignore_errors, range, stop_at_empty, empty_as_varchar). See 'duckdb extension read_excel for more information

Usage

```
read_excel(
  .con,
  to_database_name,
  to_schema_name,
  to_table_name,
  file_path,
  header,
  sheet,
  all_varchar,
```

```
  ignore_errors,
  range,
  stop_at_empty,
  empty_as_varchar,
  write_type
)
```

Arguments

.con	A valid DBI connection (DuckDB / MotherDuck).
to_database_name	Target database name (new or existing)
to_schema_name	Target schema name (new or existing)
to_table_name	Target table name to create (new or existing)
file_path	Path to the Excel file (.xlsx)
header	Logical; if TRUE, first row is header
sheet	Character; sheet name to read (defaults to first sheet)
all_varchar	Logical; coerce all columns to VARCHAR
ignore_errors	Logical; continue on cell/row errors
range	Character; Excel range like "A1" or "A1:C100"
stop_at_empty	Logical; stop at first completely empty row
empty_as_varchar	Logical; treat empty columns as VARCHAR
write_type	Logical, will drop previous table and replace with new table

Value

Invisibly returns NULL. Side effect: creates <database>.<schema>.<table> with the Excel data.

See Also

Other db-read: read_csv()

---

show_current_user *Show current database user*

Description

Return or print the current database user for a MotherDuck / DuckDB connection.

Usage

```
show_current_user(.con, return = "msg")
```

Arguments

.con	A valid DBI connection (DuckDB / MotherDuck).
return	Character scalar, one of "msg" or "arg". Default: "msg".

Details

This helper queries the active DB connection for the current user (via SELECT current_user).

The function supports two output modes:

- "msg" — prints a small informative message and returns the result invisibly (useful for interactive use),
- "arg" — returns a tibble containing the current_user column.

Value

a tibble

See Also

Other db-api: configure_md_user_settings(), create_md_access_token(), create_md_user(), delete_md_access_token(), delete_md_user(), list_md_active_accounts(), list_md_user_instance(), list_md_user_tokens()

Examples

```
## Not run:
# Using an existing connection
con <- connect_to_motherduck()
show_current_user(.con = con, return = "msg")

## End(Not run)
```

---

show_information_schema

Show Information Schema Schemas

Description

List all schemas available in the connected database.

This function queries the information_schema.schemata view to return metadata about schemas that exist in the database associated with the provided DBI connection. It is intended for inspection and debugging purposes and does not modify database state.

Usage

```
show_information_schema(.con)
```

Arguments

.con	A valid DBI connection (DuckDB / MotherDuck).

Details

The structure and contents of the returned tibble depend on the database backend. Databases may expose additional or fewer columns in information_schema.schemata.

Value

A tibble with one row per schema. Columns typically include catalog name, schema name, schema owner, and other database-specific metadata as defined by the SQL information schema.

Examples

```
## Not run:
library(DBI)
con <- dbConnect(duckdb::duckdb())

show_information_schema(con)

dbDisconnect(con, shutdown = TRUE)

## End(Not run)
```

---

show_motherduck_token *Show Your MotherDuck Token*

Description

Displays the active MotherDuck authentication token associated with the current connection. Useful for debugging or verifying that your session is authenticated correctly.

Usage

```
show_motherduck_token(.con)
```

Arguments

.con	A valid DBI connection (DuckDB / MotherDuck).

Details

The show_motherduck_token() function executes the internal MotherDuck pragma print_md_token and returns the token information. This function should only be used in secure environments, as it exposes your authentication token in plain text. It requires a valid MotherDuck connection established with DBI::dbConnect().

Value

A tibble containing the current MotherDuck token.

See Also

Other db-con: `install_extensions()`, `load_extensions()`, `validate_extension_install_status()`, `validate_extension_load_status()`

Examples

```
## Not run:
con <- DBI::dbConnect(duckdb::duckdb(dbdir = tempfile()))
show_motherduck_token(con)
DBI::dbDisconnect(con)

## End(Not run)
```

`summary` *Summarize a Lazy DBI Table*

Description

The `summary.tbl_lazy()` method provides a database-aware summary interface for lazy tables created via **dbplyr**. Instead of collecting data into R, it constructs a SQL `SUMMARIZE` query and executes it remotely, returning another lazy table reference.

Usage

```
## S3 method for class 'tbl_lazy'
summary(object, ...)
```

Arguments

`object`	A `tbl_lazy` object representing a remote database table or query.
`...`	Additional arguments (currently unused). Present for S3 method compatibility.

Details

This method does **not** pull data into memory. Instead, it creates a new lazy query object representing the database-side summary. To retrieve the summarized data, use `collect()` on the returned object.

Value

A `tbl_lazy` object containing the summarized results, still backed by the remote database connection.

See Also

Other db-meta: `cd()`, `launch_ui()`, `pwd()`

Examples

```
## Not run:
library(DBI)
library(duckdb)
library(dplyr)

con <- dbConnect(duckdb::duckdb(dbdir = tempfile()))
dbWriteTable(con, "mtcars", mtcars)

tbl_obj <- tbl(con, "mtcars")

# Returns a lazy summary table
summary(tbl_obj)

dbDisconnect(con)

## End(Not run)
```

`upload_database_to_md` *Upload a Local Database to MotherDuck*

Description

Creates a new database on MotherDuck (if it does not exist) and copies all objects from an existing local database into it using the `COPY FROM DATABASE` command.

Usage

```
upload_database_to_md(.con, from_db_name, to_db_name)
```

Arguments

`.con`	A valid `DBI` connection (DuckDB / MotherDuck).
`from_db_name`	The local database name to copy from.
`to_db_name`	The target MotherDuck database to create/overwrite.

Details

- Runs `CREATE DATABASE <to_db_name>` if the target database does not exist.
- Then runs `COPY FROM DATABASE <from_db_name> TO <to_db_name>` to copy all objects (tables, views, etc.) from the local database.
- Prints a CLI status report (connection, user, current DB) after completion.

Value

Invisibly returns NULL. Side effect: creates the target database and copies all objects; prints a CLI action report.

See Also

Other db-manage: alter_table_schema(), copy_tables_to_new_location(), create_database(), create_if_not_exists_share(), create_or_replace_share(), create_schema(), create_table(), delete_and_create_schema(), delete_database(), delete_schema(), delete_table(), describe_share(), drop_share(), list_owned_shares(), list_shared_with_me_shares()

Examples

```
## Not run:
con_db <- DBI::dbConnect(duckdb::duckdb(dbdir = tempfile()))
create_table(.con=con_db,.data=mtcars,database_name="memory",schema_name="main",table_name="mtcars")
con_md <- connect_to_motherduck()

upload_database_to_md(con_md, from_db_name = "memory", to_db_name = "analytics")

## End(Not run)
```

---

validate_con *Validate connection is DuckDB*

Description

Validates that your connection object is a DuckDB connection

Usage

```
validate_con(.con)
```

Arguments

.con	A valid DBI connection (DuckDB / MotherDuck).

Value

logical value or error message

Examples

```
## Not run:
con <- DBI::dbConnect(duckdb::duckdb())
validate_duckdb_con(con)

## End(Not run)
```

validate_database_exists *Check if a Database Exists*

Description

Checks whether a database with the specified name exists on the current connection.

Usage

```
validate_database_exists(.con, database_name)
```

Arguments

.con	A valid DBI connection (DuckDB / MotherDuck).
database_name	A string specifying the database name to check.

Details

This function queries the available databases for the connection using motherduck::list_all_databases() and performs an exact match check. Returns TRUE if the database exists, FALSE otherwise.

Value

Logical TRUE if the database exists, FALSE otherwise.

Examples

```
## Not run:
con <- DBI::dbConnect(duckdb::duckdb())
validate_database_exists(con, "test_db")

## End(Not run)
```

validate_database_schema_exists *Check if a Schema Exists in a Database*

Description

Checks whether a schema with the specified name exists within a given database.

Usage

```
validate_database_schema_exists(.con, database_name, target_schema)
```

Arguments

.con	A valid DBI connection (DuckDB / MotherDuck).
database_name	The database name to check.
target_schema	The schema name to check within the database.

Details

This function queries the information schema to check if the specified schema exists within the given database.

Value

Logical TRUE if the schema exists, FALSE otherwise.

Examples

```
## Not run:
con <- DBI::dbConnect(duckdb::duckdb())
validate_database_schema_exists(con, "memory", "main")

## End(Not run)
```

---

validate_extension_install_status

Validate Installed MotherDuck/DuckDB Extensions

Description

Checks whether specified DuckDB or MotherDuck extensions are installed and provides a detailed status report.

Usage

```
validate_extension_install_status(.con, extension_names, return_type = "msg")
```

Arguments

.con	A valid DBI connection (DuckDB / MotherDuck).
extension_names	A character vector of extensions to validate.
return_type	One of "msg", "ext", or "arg". Determines the type of return value.

- "msg" prints CLI messages.
- "ext" returns a list of installed and failed extensions.
- "arg" returns TRUE if all requested extensions are installed, FALSE otherwise.

Details

The `validate_extension_install_status()` function validates the current connection and checks which of the requested extensions are installed. It produces a detailed CLI report showing which extensions are installed, not installed, or missing.

The function can return different outputs based on the `return_type` argument:

- `"msg"`: prints a CLI report with extension statuses.
- `"ext"`: returns a list containing `success_ext` (installed) and `fail_ext` (not installed).
- `"arg"`: returns a logical value indicating whether all requested extensions are installed.

Value

Depending on `return_type`:

- `"msg"`: prints CLI messages (invisible `NULL`).
- `"ext"`: list with `success_ext` and `fail_ext`.
- `"arg"`: logical indicating if all requested extensions are installed.

See Also

Other db-con: `install_extensions()`, `load_extensions()`, `show_motherduck_token()`, `validate_extension_load_s`

Examples

```
## Not run:
con <- DBI::dbConnect(duckdb::duckdb(dbdir = tempfile()))

# Print CLI report
validate_extension_install_status(con, extension_names = c("arrow", "excel"), return_type = "msg")

# Return a list of installed and failed extensions
validate_extension_install_status(con, extension_names = c("arrow", "excel"), return_type = "ext")

# Return logical indicating if all requested extensions are installed
validate_extension_install_status(con, extension_names = c("arrow", "excel"), return_type = "arg")

## End(Not run)
```

`validate_extension_load_status`

Validate Loaded MotherDuck/DuckDB Extensions

Description

Checks whether specified DuckDB or MotherDuck extensions are loaded in the current session and provides a detailed status report.

Usage

```
validate_extension_load_status(.con, extension_names, return_type = "msg")
```

Arguments

.con	A valid DBI connection (DuckDB / MotherDuck).
extension_names	A character vector of extensions to validate.
return_type	One of "msg", "ext", or "arg". Determines the type of return value: • "msg" prints CLI messages. • "ext" returns a list with success_ext, fail_ext, and missing_ext. • "arg" returns TRUE if all requested extensions are loaded, FALSE otherwise.

Details

The validate_extension_load_status() function validates the current connection, then checks which of the requested extensions are loaded. It produces a detailed CLI report showing which extensions are loaded, failed to load, or missing.

Depending on the return_type argument, the function can either print messages, return a list of extension statuses, or return a logical value indicating whether all requested extensions are successfully loaded.

Value

Depending on return_type:

- "msg": prints CLI messages (invisible NULL).
- "ext": list with success_ext, fail_ext, and missing_ext.
- "arg": logical indicating if all requested extensions are loaded.

See Also

Other db-con: install_extensions(), load_extensions(), show_motherduck_token(), validate_extension_instal

Examples

```
## Not run:
con <- DBI::dbConnect(duckdb::duckdb(dbdir = tempfile()))

# Print CLI report
validate_extension_load_status(con, extension_names = c("excel", "arrow"), return_type = "msg")

# Return a list of loaded, failed, and missing extensions
validate_extension_load_status(con, extension_names = c("excel", "arrow"), return_type = "ext")

# Return logical indicating if all requested extensions are loaded
validate_extension_load_status(con, extension_names = c("excel", "arrow"), return_type = "arg")
```

```
## End(Not run)
```

---

validate_md_connection_status
Validate Mother Duck Connection Status

Description

Validates if you are successfully connected to motherduck database and will return either a logical value or print a message

Usage

```
validate_md_connection_status(.con, return_type = "msg")
```

Arguments

.con	A valid DBI connection (DuckDB / MotherDuck).
return_type	return message or logical value of connection status

Value

logical value or warning message

See Also

connect_to_motherduck()

Examples

```
## Not run:
con <- DBI::dbConnect(duckdb::duckdb())
validate_md_connection_status(con)

## End(Not run)
```

Index

∗ **datasets**
config_csv, 5
config_db, 6
config_excel, 8
config_parquet, 9
∗ **db-api**
configure_md_user_settings, 9
create_md_access_token, 15
create_md_user, 17
delete_md_access_token, 25
delete_md_user, 26
list_md_active_accounts, 38
list_md_user_instance, 39
list_md_user_tokens, 40
show_current_user, 49
∗ **db-con**
install_extensions, 31
load_extensions, 45
show_motherduck_token, 51
validate_extension_install_status, 56
validate_extension_load_status, 57
∗ **db-list**
list_all_databases, 33
list_all_tables, 34
list_current_schemas, 34
list_current_tables, 35
list_extensions, 36
list_fns, 37
list_setting, 42
list_shares, 44
∗ **db-manage**
alter_table_schema, 3
copy_tables_to_new_location, 12
create_database, 13
create_if_not_exists_share, 14
create_or_replace_share, 18
create_r2_secret, 19
create_s3_secret, 20
create_schema, 22
create_table, 23
delete_and_create_schema, 24
delete_database, 25
delete_schema, 27
delete_table, 28
describe_share, 29
drop_share, 30
list_owned_shares, 41
list_shared_with_me_shares, 43
upload_database_to_md, 53
∗ **db-meta**
cd, 4
launch_ui, 32
pwd, 46
summary, 52
∗ **db-read**
read_csv, 47
read_excel, 48

alter_table_schema, 3, *13*, *15*, *19*, *22*, *24*, *25*, *28–30*, *41*, *43*, *54*
alter_table_schema(), *20, 21*

cd, 4, *32*, *46*, *53*
config_csv, 5
config_db, 6
config_excel, 8
config_parquet, 9
configure_md_user_settings, 9, *16, 17*, *26*, *27*, *38–40*, *50*
connect_to_motherduck, 11
connect_to_motherduck(), *59*
copy_tables_to_new_location, *4*, 12, *13*, *15*, *19*, *22*, *24*, *25*, *28–30*, *41*, *43*, *54*
copy_tables_to_new_location(), *20, 21*
create_database, *4*, 13, *13*, *15*, *19*, *22*, *24*, *25*, *28–30*, *41*, *43*, *54*
create_database(), *20, 21*

create_if_not_exists_share, *4*, *13*, 14, *19*, *22*, *24, 25*, *28–30*, *41*, *43*, *54*
create_if_not_exists_share(), *20, 21*
create_md_access_token, *10*, 15, *17*, *26, 27*, *38–40*, *50*
create_md_user, *10*, *16*, 17, *26, 27*, *38–40*, *50*
create_or_replace_share, *4*, *13*, *15*, 18, *22*, *24, 25*, *28–30*, *41*, *43*, *54*
create_or_replace_share(), *20, 21*
create_r2_secret, 19
create_r2_secret(), *20, 21*
create_s3_secret, 20
create_s3_secret(), *19, 20*
create_schema, *4*, *13*, *15*, *19*, 22, *24, 25*, *28–30*, *41*, *43*, *54*
create_schema(), *20, 21*
create_table, *4*, *13*, *15*, *19*, *22*, 23, *24, 25*, *28–30*, *41*, *43*, *54*
create_table(), *20, 21*
create_table_dbi(), *23*
create_table_tbl(), *23*

delete_and_create_schema, *4*, *13*, *15*, *19*, *22*, 24, *24, 25*, *28–30*, *41*, *43*, *54*
delete_and_create_schema(), *20, 21*
delete_database, *4*, *13*, *15*, *19*, *22*, *24*, 25, *28–30*, *41*, *43*, *54*
delete_database(), *20, 21*
delete_md_access_token, *10*, *16, 17*, 25, *27*, *38–40*, *50*
delete_md_user, *10*, *16, 17*, 26, *26*, *38–40*, *50*
delete_schema, *4*, *13*, *15*, *19*, *22*, *24, 25*, 27, *28–30*, *41*, *43*, *54*
delete_schema(), *20, 21*
delete_table, *4*, *13*, *15*, *19*, *22*, *24, 25*, 28, *28–30*, *41*, *43*, *54*
delete_table(), *20, 21*
describe_share, *4*, *13*, *15*, *19*, *22*, *24, 25*, *28*, 29, *30*, *41*, *43*, *54*
describe_share(), *20, 21*
drop_share, *4*, *13*, *15*, *19*, *22*, *24, 25*, *28, 29*, 30, *41*, *43*, *54*
drop_share(), *20, 21*

install_extensions, 31, *45*, *52*, *57, 58*

launch_ui, *4*, 32, *46*, *53*
list_all_databases, 33, *34–37*, *42*, *44*
list_all_databases(), *4*
list_all_tables, *33*, 34, *35–37*, *42*, *44*
list_current_schemas, *33*, 34, *34*, *36, 37*, *42*, *44*
list_current_schemas(), *4*
list_current_tables, *33, 34*, 35, *35–37*, *42*, *44*
list_extensions, *33–35*, 36, *36, 37*, *42*, *44*
list_fns, *33–36*, 37, *42*, *44*
list_md_active_accounts, *10*, *16, 17*, *26, 27*, 38, *39, 40*, *50*
list_md_user_instance, *10*, *16, 17*, *26, 27*, *38*, 39, *40*, *50*
list_md_user_tokens, *10*, *16, 17*, *26, 27*, *38, 39*, 40, *50*
list_owned_shares, *4*, *13*, *15*, *19*, *22*, *24, 25*, *28–30*, 41, *43*, *54*
list_owned_shares(), *20, 21*
list_setting, *33–37*, 42, *44*
list_shared_with_me_shares, *4*, *13*, *15*, *19*, *22*, *24, 25*, *28–30*, *41*, 43, *54*
list_shared_with_me_shares(), *20, 21*
list_shares, *33–37*, *42*, 44
load_extensions, *31*, 45, *52*, *57, 58*
load_extensions(), *20, 21*

pwd, *4*, *32*, 46, *53*

read_csv, 47, *49*
read_excel, 48, *48*

show_current_user, *10*, *16, 17*, *26, 27*, *38–40*, 49
show_information_schema, 50
show_motherduck_token, *31*, *45*, 51, *57, 58*
summary, *4*, *32*, *46*, 52

tbl_lazy, *52*
tibble::tibble, *42*, *44*

upload_database_to_md, *4*, *13*, *15*, *19*, *22*, *24, 25*, *28–30*, *41*, *43*, 53
upload_database_to_md(), *20, 21*

validate_con, 54
validate_database_exists, 55
validate_database_schema_exists, 55
validate_extension_install_status, *31*, *45*, *52*, 56, *58*
validate_extension_load_status, *31*, *45*, *52*, 57, *57*
validate_md_connection_status, 59