Package 'supabaseR'

May 9, 2026

Title 'CRUD' Utils in 'R' for 'Supabase'

Version 1.0.0

Description An dual-paradigm interface to 'Supabase' (<https://supabase.com/>), an open-source backend-as-a-service platform. Provides comprehensive database operations including Create, Read, Update, Delete ('CRUD') functionality through both REST API endpoints and direct 'PostgreSQL' database connections. Simplifies authentication, data management, and schema operations for 'Supabase' projects.

License MIT + file LICENSE

URL https://deepanshkhurana.github.io/supabaseR/, https://github.com/DeepanshKhurana/supabaseR

BugReports https://github.com/DeepanshKhurana/supabaseR/issues

Encoding UTF-8

Language en-US

RoxygenNote 7.3.3

Imports checkmate, cli, DBI, dplyr, glue, httr2, RPostgres

Suggests jsonlite, mockery, testthat (>= 3.0.0), withr

Config/testthat/edition 3

NeedsCompilation no

Author Deepansh Khurana [aut, cre], Maciej Banas [aut]

Maintainer Deepansh Khurana <deepanshkhurana@outlook.com>

Depends R (>= 4.1.0)

Repository CRAN

Date/Publication 2026-04-21 18:10:09 UTC

Contents

`sb_api_connect` *Connect to Supabase via REST API*

Description

At least one key must be supplied alongside the URL. When both `key` and `secret_key` are available, the secret key is always preferred for requests because it has higher privileges (bypasses Row Level Security).

Usage

```
sb_api_connect(
  url = Sys.getenv("SUPABASE_URL"),
  key = Sys.getenv(if (nchar(Sys.getenv("SUPABASE_PUBLISHABLE_KEY")) > 0)
    "SUPABASE_PUBLISHABLE_KEY" else "SUPABASE_ANON_KEY"),
  secret_key = Sys.getenv(if (nchar(Sys.getenv("SUPABASE_SECRET_KEY")) > 0)
    "SUPABASE_SECRET_KEY" else "SUPABASE_ROLE_KEY")
)
```

Arguments

`url`	The Supabase URL (default: from SUPABASE_URL env var)
`key`	The Supabase API key. Accepts the legacy anon JWT key or the new publishable key (`sb_publishable_...`). Optional when `secret_key` is provided. (default: from SUPABASE_PUBLISHABLE_KEY or SUPABASE_ANON_KEY env var)
`secret_key`	The Supabase secret key. Accepts the legacy `service_role` JWT key or the new secret key (`sb_secret_...`). When present it is used for all requests, bypassing Row Level Security. (default: from SUPABASE_SECRET_KEY or SUPABASE_ROLE_KEY env var)

Value

Invisible list with API credentials

Examples

```
## Not run:
# Connect using environment variables (SUPABASE_URL + SUPABASE_PUBLISHABLE_KEY)
sb_api_connect()

# Connect with a secret key only (bypasses Row Level Security)
sb_api_connect(
  url = "https://xxx.supabase.co",
  secret_key = "sb_secret_..."
)

# Connect with both publishable and secret keys
```

```
sb_api_connect(
  url = "https://xxx.supabase.co",
  key = "sb_publishable_...",
  secret_key = "sb_secret_..."
)

# Disconnect when done
sb_api_disconnect()

## End(Not run)
```

`sb_api_delete` *Delete rows from a table via API*

Description

Delete rows from a table via API

Usage

```
sb_api_delete(table = NULL, where = NULL, schema = get_schema())
```

Arguments

`table`	The table name
`where`	A named list for filtering. Supports operators via nested lists.
`schema`	The schema name

Value

Number of rows deleted (invisibly)

Examples

```
## Not run:
sb_api_connect()

# Delete a row by id
sb_api_delete("users", where = list(id = 1))

# Delete with an operator
sb_api_delete("logs", where = list(created_at = list(lt = "2024-01-01")))

## End(Not run)
```

sb_api_disconnect *Disconnect from Supabase API*

Description

Clears the stored API credentials from the session.

Usage

```
sb_api_disconnect()
```

Value

Invisible NULL

Examples

```
## Not run:
sb_api_connect()

# Clear stored API credentials
sb_api_disconnect()

## End(Not run)
```

sb_api_insert *Insert rows into a table via API*

Description

Insert rows into a table via API

Usage

```
sb_api_insert(table = NULL, data = NULL, schema = get_schema())
```

Arguments

table	The table name
data	A data frame of rows to insert
schema	The schema name

Value

Number of rows inserted (invisibly)

Examples

```
## Not run:
sb_api_connect()

# Insert a single row
sb_api_insert("users", data.frame(name = "Alice", email = "alice@example.com"))

# Insert multiple rows
new_users <- data.frame(
  name  = c("Bob", "Carol"),
  email = c("bob@example.com", "carol@example.com")
)
sb_api_insert("users", new_users)

## End(Not run)
```

`sb_api_query` *Query a table via API*

Description

Query a table via API

Usage

```
sb_api_query(
  table = NULL,
  columns = "*",
  where = NULL,
  limit = 0,
  schema = get_schema()
)
```

Arguments

`table`	The table name
`columns`	Columns to select (default: all). Character vector or `"*"`.
`where`	A named list for filtering. Supports operators via nested lists: `list(id = 1)` for equality, `list(age = list(gt = 25))` for age > 25. Operators: eq, neq, gt, gte, lt, lte, like, ilike, in, is
`limit`	Maximum rows to return (0 for all)
`schema`	The schema name

Value

A `tibble`

Examples

```
## Not run:
sb_api_connect()

# Select specific columns
sb_api_query("users", columns = c("id", "name"))

# Filter with a simple equality condition
sb_api_query("users", where = list(status = "active"))

# Filter with operators
sb_api_query("orders", where = list(
  status = "pending",
  total = list(gte = 100)
))

# Available operators: eq, neq, gt, gte, lt, lte, like, ilike, in, is

# Limit results
sb_api_query("users", limit = 5)

## End(Not run)
```

---

sb_api_read *Read table data via API*

Description

Read table data via API

Usage

```
sb_api_read(table = NULL, limit = 0, schema = get_schema())
```

Arguments

table	The table name
limit	Maximum rows to return (0 for all)
schema	The schema name

Value

A tibble

Examples

```
## Not run:
sb_api_connect()

# Read all rows
sb_api_read("users")

# Read with a row limit
sb_api_read("users", limit = 10)

# Read from a non-default schema
sb_api_read("orders", schema = "billing")

## End(Not run)
```

`sb_api_schema` *Get table schema via API*

Description

Queries the PostgREST OpenAPI spec to retrieve column names and types for a table.

Usage

```
sb_api_schema(table = NULL, schema = get_schema())
```

Arguments

`table`	The table name
`schema`	The schema name

Value

A `tibble` with columns `column_name` and `data_type`

Examples

```
## Not run:
sb_api_connect()

# Get column names and types for a table
sb_api_schema("users")

# Get schema for a table in a non-default schema
sb_api_schema("orders", schema = "billing")

## End(Not run)
```

sb_api_status *Check API backend status*

Description

Check API backend status

Usage

```
sb_api_status()
```

Value

A list with API connection info

Examples

```
## Not run:
sb_api_connect()

sb_api_status()

## End(Not run)
```

sb_api_tables *List tables via API*

Description

Queries the PostgREST OpenAPI spec to list exposed tables.

Usage

```
sb_api_tables(schema = get_schema())
```

Arguments

schema	The schema name

Value

A character vector of table names

Examples

```
## Not run:
sb_api_connect()

# List all tables in the default schema
sb_api_tables()

# List tables in a specific schema
sb_api_tables(schema = "billing")

## End(Not run)
```

`sb_api_table_exists` *Check if a table exists via API*

Description

Check if a table exists via API

Usage

```
sb_api_table_exists(table = NULL, schema = get_schema())
```

Arguments

`table`	The table name
`schema`	The schema name

Value

`TRUE` if the table exists, `FALSE` otherwise

Examples

```
## Not run:
sb_api_connect()

sb_api_table_exists("users")

sb_api_table_exists("nonexistent_table")

## End(Not run)
```

sb_api_truncate *Truncate a table via API*

Description

Deletes all rows from the table using a DELETE request with no filter. Requires the secret key or RLS must be disabled for the table.

Usage

```
sb_api_truncate(table = NULL, schema = get_schema())
```

Arguments

table	The table name
schema	The schema name

Value

Invisible NULL

Examples

```
## Not run:
sb_api_connect()

# Remove all rows from a table
# Requires a secret key or Row Level Security to be disabled
sb_api_truncate("logs")

## End(Not run)
```

sb_api_update *Update rows in a table via API*

Description

Update rows in a table via API

Usage

```
sb_api_update(table = NULL, data = NULL, where = NULL, schema = get_schema())
```

Arguments

table	The table name
data	A named list of column = value pairs to set
where	A named list for filtering. Supports operators via nested lists.
schema	The schema name

Value

Number of rows updated (invisibly)

Examples

```
## Not run:
sb_api_connect()

# Update a row by id
sb_api_update(
  "users",
  data  = list(email = "newemail@example.com"),
  where = list(id = 1)
)

# Update with an operator
sb_api_update(
  "products",
  data  = list(in_stock = FALSE),
  where = list(quantity = list(lte = 0))
)

## End(Not run)
```

---

sb_api_upsert *Upsert rows into a table via API*

Description

Insert rows, or update on conflict with specified columns.

Usage

```
sb_api_upsert(
  table = NULL,
  data = NULL,
  conflict_columns = NULL,
  schema = get_schema()
)
```

Arguments

table	The table name
data	A data frame of rows to upsert
conflict_columns	Column(s) to check for conflicts (e.g., primary key)
schema	The schema name

Value

Number of rows affected (invisibly)

Examples

```
## Not run:
sb_api_connect()

# Upsert a single row (insert or update on conflict)
sb_api_upsert(
  "users",
  data.frame(id = 1, name = "Alice Updated"),
  conflict_columns = "id"
)

# Upsert multiple rows with a composite key
sb_api_upsert(
  "orders",
  data.frame(
    user_id  = c(1, 2),
    order_id = c(10, 11),
    status   = c("shipped", "pending")
  ),
  conflict_columns = c("user_id", "order_id")
)

## End(Not run)
```

---

sb_connect *Connect to Supabase*

Description

Unified connection that auto-detects or uses specified backend.

Usage

```
sb_connect(
  backend = c("auto", "db", "api"),
  schema = Sys.getenv("SUPABASE_SCHEMA", "public")
)
```

Arguments

backend	Backend to use: "auto", "db", or "api"
schema	The schema name (db backend only)

Value

Invisible connection info

Examples

```
## Not run:
# Auto-detect backend based on available env vars
sb_connect()

# Explicitly use DBI backend
sb_connect(backend = "db")

# Explicitly use API backend
sb_connect(backend = "api")

# Disconnect when done
sb_disconnect()

## End(Not run)
```

---

sb_db_connect *Connect to Supabase*

Description

Uses DBI/PostgreSQL to connect directly to your Supabase database. Get connection parameters from the Connect tab in your Supabase Dashboard (Project Settings > Database > Connection Strings). All three modes work: Direct Connection, Transaction Pooler, and Session Pooler. If Direct connection doesn't work, try Transaction or Session Pooler. Remember to use the correct host, user, and password from your chosen connection mode.

Usage

```
sb_db_connect(schema = Sys.getenv("SUPABASE_SCHEMA", "public"))
```

Arguments

schema	The schema name

Value

Invisible connection object

Examples

```
## Not run:
# Connect using environment variables
sb_db_connect()

# Connect with a specific schema
sb_db_connect(schema = "public")

# Disconnect when done
sb_db_disconnect()

## End(Not run)
```

`sb_db_creds` *Read Supabase credentials from environment variables*

Description

Connection parameters should be obtained from the Connect tab in your Supabase Dashboard (Project Settings > Database > Connection Strings). All three connection modes work (Direct, Transaction Pooler, Session Pooler), but if Direct connection fails, try Transaction or Session Pooler modes. Be sure to use the correct host, port, user, and password for your chosen mode. Port defaults to 6543 if not specified.

Usage

```
sb_db_creds()
```

Value

A list of Supabase credentials

`sb_db_delete` *Delete rows from a table*

Description

Delete rows from a table

Usage

```
sb_db_delete(table = NULL, where = NULL, schema = get_schema())
```

Arguments

table	The table name
where	A named list for WHERE clause. Supports operators via nested lists.
schema	The schema name

Value

Number of rows deleted (invisibly)

Examples

```
## Not run:
# Delete by id
sb_db_delete("users", where = list(id = 1))

# Delete with operator
sb_db_delete("sessions", where = list(expires_at = list(lt = Sys.time())))

## End(Not run)
```

---

sb_db_disconnect *Disconnect from Supabase*

Description

Disconnect from Supabase

Usage

```
sb_db_disconnect()
```

Value

Invisible NULL

sb_db_insert *Insert rows into a table*

Description

Insert rows into a table

Usage

```
sb_db_insert(table = NULL, data = NULL, schema = get_schema())
```

Arguments

table	The table name
data	A data frame of rows to insert
schema	The schema name

Value

Number of rows inserted (invisibly)

Examples

```
## Not run:
# Insert a single row
sb_db_insert("users", data.frame(name = "Alice", email = "alice@example.com"))

# Insert multiple rows
new_users <- data.frame(
  name = c("Bob", "Carol"),
  email = c("bob@example.com", "carol@example.com")
)
sb_db_insert("users", new_users)

## End(Not run)
```

sb_db_query *Query a table*

Description

Query a table

Usage

```
sb_db_query(
  table = NULL,
  columns = "*",
  where = NULL,
  limit = 0,
  sql = NULL,
  schema = get_schema()
)
```

Arguments

table	The table name (ignored if sql is provided)
columns	Columns to select (default: all)
where	A named list for WHERE clause. Supports operators via nested lists: list(id = 1) for equality, list(age = list(gt = 25)) for age > 25. Operators: eq, neq, gt, gte, lt, lte, like, ilike, in, is
limit	Maximum rows to return
sql	Raw SQL query (DBI backend only)
schema	The schema name

Value

A data frame with query results

Examples

```
## Not run:
# Select specific columns
sb_db_query("users", columns = c("id", "name"))

# Filter with where clause
sb_db_query("users", where = list(status = "active"))

# Filter with operators
sb_db_query("orders", where = list(
  status = "pending",
  total = list(gte = 100)
))

# Available operators: eq, neq, gt, gte, lt, lte, like, ilike, in, is

# Raw SQL query
sb_db_query(sql = "SELECT COUNT(*) FROM users WHERE status = 'active'")

## End(Not run)
```

sb_db_read *Read table data*

Description

Read table data

Usage

```
sb_db_read(table = NULL, limit = 0, schema = get_schema())
```

Arguments

table	The table name
limit	Maximum rows to return (0 for all)
schema	The schema name

Value

A data frame with table data

Examples

```
## Not run:
# Read all rows from a table
users <- sb_db_read("users")

# Read with a row limit
recent <- sb_db_read("orders", limit = 100)

# Read from a specific schema
products <- sb_db_read("products", schema = "inventory")

## End(Not run)
```

sb_db_schema *Get table schema*

Description

Get table schema

Usage

```
sb_db_schema(table = NULL, schema = get_schema())
```

Arguments

table	The table name
schema	The schema name

Value

A data frame with column_name and data_type

Examples

```
## Not run:
# Get column info for a table
sb_db_schema("users")

# Returns: column_name, data_type, is_nullable

## End(Not run)
```

---

sb_db_status *Check DBI backend status*

Description

Check DBI backend status

Usage

```
sb_db_status()
```

Value

A list with DBI connection info

---

sb_db_tables *List tables in schema*

Description

List tables in schema

Usage

```
sb_db_tables(schema = get_schema())
```

Arguments

schema	The schema name

Value

A character vector of table names

Examples

```
## Not run:
# List all tables in current schema
sb_db_tables()

# List tables in a specific schema
sb_db_tables(schema = "auth")

## End(Not run)
```

---

sb_db_table_exists *Check if table exists*

Description

Check if table exists

Usage

```
sb_db_table_exists(table = NULL, schema = get_schema())
```

Arguments

table	The table name
schema	The schema name

Value

TRUE if table exists, FALSE otherwise

---

sb_db_truncate *Truncate a table*

Description

Truncate a table

Usage

```
sb_db_truncate(table = NULL, schema = get_schema())
```

Arguments

table	The table name
schema	The schema name

Value

Invisible NULL

Examples

```
## Not run:
# Remove all rows from a table (use with caution!)
sb_db_truncate("temp_data")

## End(Not run)
```

---

sb_db_update *Update rows in a table*

Description

Update rows in a table

Usage

```
sb_db_update(table = NULL, data = NULL, where = NULL, schema = get_schema())
```

Arguments

table	The table name
data	A named list of column = value pairs to set
where	A named list for WHERE clause. Supports operators via nested lists.
schema	The schema name

Value

Number of rows affected (invisibly)

Examples

```
## Not run:
# Update a row by id
sb_db_update(
  "users",
  data = list(email = "newemail@example.com"),
  where = list(id = 1)
)
```

```
# Update with operator
sb_db_update(
  "products",
  data = list(in_stock = FALSE),
  where = list(quantity = list(lte = 0))
)

## End(Not run)
```

---

sb_db_upsert *Upsert rows into a table*

Description

Insert rows, or update if conflict on specified columns.

Usage

```
sb_db_upsert(
  table = NULL,
  data = NULL,
  conflict_columns = NULL,
  schema = get_schema()
)
```

Arguments

table	The table name
data	A data frame of rows to upsert
conflict_columns	Column(s) to check for conflicts (e.g., primary key)
schema	The schema name

Value

Number of rows affected (invisibly)

Examples

```
## Not run:
# Upsert: insert or update on conflict
sb_db_upsert(
  "users",
  data = data.frame(id = 1, name = "Alice", email = "alice@new.com"),
  conflict_columns = "id"
)
```

```
# Upsert multiple rows with composite key
sb_db_upsert(
  "order_items",
  data = data.frame(
    order_id = c(1, 1),
    product_id = c(10, 20),
    quantity = c(5, 3)
  ),
  conflict_columns = c("order_id", "product_id")
)

## End(Not run)
```

---

sb_delete *Delete rows from a table*

Description

Delete rows from a table

Usage

```
sb_delete(table = NULL, where = NULL, schema = get_schema())
```

Arguments

table	The table name
where	A named list for WHERE clause
schema	The schema name

Value

Number of rows deleted (invisibly)

---

sb_disconnect *Disconnect from Supabase*

Description

Disconnect from Supabase

Usage

```
sb_disconnect()
```

Value

Invisible NULL

sb_insert *Insert rows into a table*

Description

Insert rows into a table

Usage

```
sb_insert(table = NULL, data = NULL, schema = get_schema())
```

Arguments

table	The table name
data	A data frame of rows to insert
schema	The schema name

Value

Number of rows inserted (invisibly)

sb_query *Query a table*

Description

Query a table

Usage

```
sb_query(
  table = NULL,
  columns = "*",
  where = NULL,
  limit = 0,
  sql = NULL,
  schema = get_schema()
)
```

Arguments

table	The table name
columns	Columns to select
where	A named list for WHERE clause
limit	Maximum rows to return
sql	Raw SQL query (db backend only)
schema	The schema name

Value

A data frame with query results

`sb_read` *Read table data*

Description

Read table data

Usage

```
sb_read(table = NULL, limit = 0, schema = get_schema())
```

Arguments

`table`	The table name
`limit`	Maximum rows to return (0 for all)
`schema`	The schema name

Value

A data frame with table data

`sb_schema` *Get table schema*

Description

Get table schema

Usage

```
sb_schema(table = NULL, schema = get_schema())
```

Arguments

`table`	The table name
`schema`	The schema name

Value

A data frame with column info

sb_status *Get connection status*

Description

Returns availability and connection state for both backends, the active backend (if `sb_connect()` has been called), and the current schema.

Usage

```
sb_status()
```

Value

A list with `dbi`, `api`, `backend`, and `schema` entries

sb_tables *List tables*

Description

List tables

Usage

```
sb_tables(schema = get_schema())
```

Arguments

`schema`	The schema name

Value

A character vector of table names

sb_table_exists *Check if table exists*

Description

Check if table exists

Usage

```
sb_table_exists(table = NULL, schema = get_schema())
```

Arguments

table	The table name
schema	The schema name

Value

TRUE if table exists

sb_truncate *Truncate a table*

Description

Truncate a table

Usage

```
sb_truncate(table = NULL, schema = get_schema())
```

Arguments

table	The table name
schema	The schema name

Value

Invisible NULL

sb_update *Update rows in a table*

Description

Update rows in a table

Usage

```
sb_update(table = NULL, data = NULL, where = NULL, schema = get_schema())
```

Arguments

table	The table name
data	A named list of column = value pairs to set
where	A named list for WHERE clause
schema	The schema name

Value

Number of rows affected (invisibly)

sb_upsert *Upsert rows into a table*

Description

Upsert rows into a table

Usage

```
sb_upsert(
  table = NULL,
  data = NULL,
  conflict_columns = NULL,
  schema = get_schema()
)
```

Arguments

table	The table name
data	A data frame of rows to upsert
conflict_columns	Column(s) to check for conflicts
schema	The schema name

Value

Number of rows affected (invisibly)

Index