

Package ‘mtgjsonsdk’

May 9, 2026

Title 'DuckDB'-Backed Query Client for 'MTGJSON' Card Data

Version 0.1.0

Description Auto-downloads Parquet data from the 'MTGJSON' CDN and exposes the full Magic: The Gathering dataset through R6-based query interfaces backed by 'DuckDB'.

URL <https://mtgjson.com>

BugReports <https://github.com/mtgjson/mtgjson-sdk-r/issues>

Depends R (>= 4.1.0)

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

Imports R6, DBI, duckdb, httr2, jsonlite

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

NeedsCompilation no

Author Zachary Halpern [aut, cre],
Robert Pratt [aut]

Maintainer Zachary Halpern <zach@mtgjson.com>

Repository CRAN

Date/Publication 2026-03-16 16:20:02 UTC

Contents

MtgjsonSDK	2
SqlBuilder	4
Index	8

MtgjsonSDK	<i>MTGJSON SDK - DuckDB-backed query client for MTGJSON card data.</i>
------------	--

Description

MTGJSON SDK - DuckDB-backed query client for MTGJSON card data.

MTGJSON SDK - DuckDB-backed query client for MTGJSON card data.

Details

Auto-downloads Parquet data from the MTGJSON CDN and provides R6-based query interfaces for the full MTG dataset.

Active bindings

cards CardQuery interface.
sets SetQuery interface.
prices PriceQuery interface.
decks DeckQuery interface.
sealed SealedQuery interface.
skus SkuQuery interface.
identifiers IdentifierQuery interface.
legalities LegalityQuery interface.
tokens TokenQuery interface.
enums EnumQuery interface.
booster BoosterSimulator interface.
meta MTGJSON build metadata.
views Currently registered DuckDB views/tables.

Methods

Public methods:

- `MtgjsonSDK$new()`
- `MtgjsonSDK$sql()`
- `MtgjsonSDK$refresh()`
- `MtgjsonSDK$export_db()`
- `MtgjsonSDK$close()`
- `MtgjsonSDK$clone()`

Method `new()`: Initialize the SDK.

Usage:

```
MtgjsonSDK$new(  
  cache_dir = NULL,  
  offline = FALSE,  
  timeout = 120,  
  on_progress = NULL  
)
```

Arguments:

`cache_dir` Directory for cached data files. NULL for platform default.

`offline` If TRUE, never download from CDN.

`timeout` HTTP request timeout in seconds.

`on_progress` Optional callback function(filename, downloaded, total).

Method `sql()`: Execute raw SQL against the DuckDB database.

Usage:

```
MtgjsonSDK$sql(query, params = NULL)
```

Arguments:

`query` SQL query string.

`params` Optional list of positional parameters.

Returns: A data.frame.

Method `refresh()`: Check for new MTGJSON data and reset if stale.

Usage:

```
MtgjsonSDK$refresh()
```

Returns: TRUE if data was stale and reset, FALSE otherwise.

Method `export_db()`: Export all loaded data to a persistent DuckDB file.

Usage:

```
MtgjsonSDK$export_db(path)
```

Arguments:

`path` Output path for the .duckdb file.

Returns: The output path (invisibly).

Method `close()`: Close the DuckDB connection and free resources.

Usage:

```
MtgjsonSDK$close()
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
MtgjsonSDK$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

SqlBuilder

SQL Builder for parameterized DuckDB queries.

Description

SQL Builder for parameterized DuckDB queries.

SQL Builder for parameterized DuckDB queries.

Details

All user-supplied values go through DuckDB parameter binding (\$1, \$2, ...), never through string interpolation. Methods return `invisible(self)` for chaining.

Methods

Public methods:

- `SqlBuilder$new()`
- `SqlBuilder$select()`
- `SqlBuilder$distinct()`
- `SqlBuilder$join()`
- `SqlBuilder$where()`
- `SqlBuilder$where_like()`
- `SqlBuilder$where_in()`
- `SqlBuilder$where_eq()`
- `SqlBuilder$where_gte()`
- `SqlBuilder$where_lte()`
- `SqlBuilder$where_regex()`
- `SqlBuilder$where_fuzzy()`
- `SqlBuilder$where_or()`
- `SqlBuilder$group_by()`
- `SqlBuilder$having()`
- `SqlBuilder$order_by()`
- `SqlBuilder$limit()`
- `SqlBuilder$offset()`
- `SqlBuilder$build()`
- `SqlBuilder$clone()`

Method `new()`: Create a builder targeting the given table or view.

Usage:

```
SqlBuilder$new(base_table)
```

Arguments:

`base_table` The DuckDB table/view name to query from.

Method `select()`: Set the columns to select (replaces default *).

Usage:

```
SqlBuilder$select(...)
```

Arguments:

... Column names or expressions.

Method `distinct()`: Add DISTINCT to the SELECT clause.

Usage:

```
SqlBuilder$distinct()
```

Method `join()`: Add a JOIN clause.

Usage:

```
SqlBuilder$join(clause)
```

Arguments:

clause Full JOIN clause string.

Method `where()`: Add a WHERE condition with \$N placeholders.

Usage:

```
SqlBuilder$where(condition, ...)
```

Arguments:

condition SQL condition with \$1, \$2, ... placeholders.

... Values bound to the placeholders.

Method `where_like()`: Add a case-insensitive LIKE condition.

Usage:

```
SqlBuilder$where_like(column, value)
```

Arguments:

column Column name.

value LIKE pattern.

Method `where_in()`: Add an IN condition with parameterized values.

Usage:

```
SqlBuilder$where_in(column, values)
```

Arguments:

column Column name.

values List of values for the IN clause.

Method `where_eq()`: Add an equality condition.

Usage:

```
SqlBuilder$where_eq(column, value)
```

Arguments:

column Column name.

value Value to match.

Method `where_gte()`: Add a \geq condition.

Usage:

```
SqlBuilder$where_gte(column, value)
```

Arguments:

column Column name.

value Minimum value (inclusive).

Method `where_lte()`: Add a \leq condition.

Usage:

```
SqlBuilder$where_lte(column, value)
```

Arguments:

column Column name.

value Maximum value (inclusive).

Method `where_regex()`: Add a regex match condition.

Usage:

```
SqlBuilder$where_regex(column, pattern)
```

Arguments:

column Column name.

pattern Regular expression pattern.

Method `where_fuzzy()`: Add a fuzzy string match condition (Jaro-Winkler similarity).

Usage:

```
SqlBuilder$where_fuzzy(column, value, threshold = 0.8)
```

Arguments:

column Column name.

value Target string.

threshold Minimum similarity score (0-1, default 0.8).

Method `where_or()`: Add OR-combined conditions.

Usage:

```
SqlBuilder$where_or(...)
```

Arguments:

... Lists of `c(condition_sql, param_value)`.

Method `group_by()`: Add GROUP BY columns.

Usage:

```
SqlBuilder$group_by(...)
```

Arguments:

... Column names or expressions.

Method `having()`: Add a HAVING condition.

Usage:

```
SqlBuilder$having(condition, ...)
```

Arguments:

`condition` SQL condition with \$N placeholders.
... Values bound to the placeholders.

Method `order_by()`: Add ORDER BY clauses.

Usage:

```
SqlBuilder$order_by(...)
```

Arguments:

... Order clauses (e.g. "name ASC").

Method `limit()`: Set the maximum number of rows to return.

Usage:

```
SqlBuilder$limit(n)
```

Arguments:

`n` Non-negative integer limit.

Method `offset()`: Set the number of rows to skip.

Usage:

```
SqlBuilder$offset(n)
```

Arguments:

`n` Non-negative integer offset.

Method `build()`: Build the final SQL string and parameter list.

Usage:

```
SqlBuilder$build()
```

Returns: A list with `sql` (character) and `params` (list).

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
SqlBuilder$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Index

MtgjsonSDK, 2

SqlBuilder, 4