Package: GO.ddb (via r-universe)

May 14, 2026

Title Tidy Interface to GO Semantic SQL via DuckDB

Version 0.99.4

Description Provides a tidyverse-oriented user interface to Gene Ontology data via the Semantic SQL representation, accessed through DuckDB. Replaces the GO.db + AnnotationDbi::select nexus with lazy tibble-based operations for term lookup, ancestor/descendant traversal, and gene-GO annotation queries. The Semantic SQL resource is managed by the ontoProc2 package via BiocFileCache.

License Artistic-2.0

Encoding UTF-8

Depends R (>= 4.1.0)

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

biocViews Annotation, GO, Ontology

Imports ontoProc2, duckdb, DBI, dplyr, glue, BiocFileCache

Suggests testthat (>= 3.0.0), DT, BiocStyle, knitr, rmarkdown, withr, hedgehog, GO.db

VignetteBuilder knitr

Config/testthat/edition 3

Config/pak/sysreqs libicu-dev libssl-dev xz-utils zlib1g-dev

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

Date/Publication 2026-05-14 18:25:07 UTC

RemoteUrl https://github.com/BiocStaging/GO.ddb

RemoteRef HEAD

RemoteSha 2bab8f8a71eb02741709a92b819b72b987f90bd2

Contents

`build_parquet_cache` *Populate the local parquet cache from a semsql SQLite file*

Description

Converts the semsql SQLite tables to parquet format and writes them to the BiocFileCache-managed parquet directory. This is a one-time setup step - subsequent calls to `make_go_con(backend = "parquet")` or `make_go_con(backend = "auto")` will use the cached files without re-converting.

Usage

```
build_parquet_cache(
  sqlite_path = NULL,
  ontology = "go",
  out_dir = NULL,
  tables = c("statements", "entailed_edge", "term_association")
)
```

Arguments

`sqlite_path`	character scalar path to the semsql SQLite file. If `NULL` (default), retrieved via `ontoProc2::semsql_connect()`.
`ontology`	character scalar. Default `"go"`.
`out_dir`	path to desired folder for parquet cache management
`tables`	character vector of table names to convert. Default covers the two tables the package actively queries.

Value

the parquet cache directory path, invisibly.

Examples

```
if (!has_parquet_cache()) {
  build_parquet_cache()
}
```

`disconnect_go` *Close the cached GO semsql connection*

Description

Disconnects the DuckDB instance and clears the package cache. The next call to any query function will trigger reconnection automatically.

Usage

```
disconnect_go()
```

Value

`NULL` invisibly.

See Also

`make_go_con`, `go_connection_active`

Examples

```
GO.ddb::make_go_con()
GO.ddb::disconnect_go()
GO.ddb::go_connection_active()
```

get_go_con *Return the raw DuckDB connection from the package cache*

Description

Useful for diagnostics, custom queries, or passing to dbplyr directly. Returns `NULL` if no connection is active.

Usage

```
get_go_con()
```

Value

a `DBIConnection` or `NULL`.

Examples

```
make_go_con()
con <- get_go_con()
DBI::dbGetQuery(con, "SELECT database_name, schema_name, table_name
                      FROM duckdb_tables()")
disconnect_go()
```

go_ancestors *Retrieve ancestors of GO terms*

Description

Uses the precomputed transitive closure in the semsql `entailed_edge` table to find all ancestors of the supplied GO CURIEs under the specified relations. Returns a long-format lazy tibble suitable for direct use with `dplyr`, `ggraph`, or gene set enrichment tooling.

Usage

```
go_ancestors(
  ids,
  relations = unname(GO_RELATIONS[c("is_a", "part_of")]),
  include_self = FALSE
)
```

Arguments

ids	character vector of GO CURIEs.
relations	character vector of predicate CURIEs to traverse. Defaults to is_a and part_of from GO_RELATIONS. The full relation set is large (~230 predicates including Uberon and BSPO imports) — always specify an explicit subset.
include_self	logical. If FALSE (default), reflexive self-edges (subject == object) present in the transitive closure are excluded.

Value

a lazy tbl_duckdb with columns:

id the query term CURIE

ancestor_id ancestor term CURIE

relation predicate CURIE

Call dplyr::collect() to materialize.

See Also

go_descendants, GO_RELATIONS

Examples

```
make_go_con()

go_ancestors("GO:0006954") |> dplyr::collect()

# is_a only
go_ancestors("GO:0006954",
  relations = GO_RELATIONS["is_a"]) |> dplyr::collect()

# multiple query terms
go_ancestors(c("GO:0006954", "GO:0008150"),
  relations = unname(GO_RELATIONS[c("is_a", "part_of")])) |>
  dplyr::count(id)

disconnect_go()
```

---

go_connection_active *Report whether a live GO connection is cached*

Description

Lightweight predicate used in examples and tests to guard against running query code when no connection has been established and no automatic reconnection is desired.

Usage

```
go_connection_active()
```

Value

logical scalar.

`go_descendants` *Retrieve descendants of GO terms*

Description

Uses the precomputed transitive closure in the semsql `entailed_edge` table to find all descendants of the supplied GO CURIEs under the specified relations. Returns a long-format lazy tibble.

Usage

```
go_descendants(
  ids,
  relations = unname(GO_RELATIONS[c("is_a", "part_of")]),
  include_self = FALSE
)
```

Arguments

`ids`	character vector of GO CURIEs.
`relations`	character vector of predicate CURIEs to traverse. Defaults to `is_a` and `part_of` from `GO_RELATIONS`.
`include_self`	logical. If `FALSE` (default), reflexive self-edges are excluded.

Value

a lazy `tbl_duckdb` with columns:

id the query term CURIE

descendant_id descendant term CURIE

relation predicate CURIE

Call `dplyr::collect()` to materialize.

See Also

`go_ancestors`, `GO_RELATIONS`

Examples

```
make_go_con()

go_descendants("GO:0006950") |> dplyr::collect()

# count descendants per ontology namespace
go_descendants("GO:0008150") |>
  dplyr::left_join(
    go_terms() |> dplyr::select(descendant_id = id, ontology),
    by = "descendant_id"
  ) |>
  dplyr::count(ontology) |>
  dplyr::collect()

disconnect_go()
```

`go_entailed_edges` *Lazy tbl for the semsql entailed_edge table*

Description

The `entailed_edge` table contains the precomputed transitive closure of all object property relations in GO, including `rdfs:subClassOf` (is_a) and `BFO:0000050` (part_of). It is the basis for `go_ancestors` and `go_descendants`.

Usage

```
go_entailed_edges(con = NULL, schema = NULL)
```

Arguments

`con`	optional `DBIConnection`. If `NULL` (default), the package cache is used via `.get_con()`.
`schema`	optional character schema name. If `NULL` (default), the cached schema is used.

Details

Self-edges (`subject == object`) encode reflexivity under the transitive closure and are excluded by default in `go_ancestors` and `go_descendants`.

Value

a lazy `tbl_duckdb` with columns `subject`, `predicate`, `object`.

Examples

```
make_go_con()
go_entailed_edges()
disconnect_go()
```

GO_RELATIONS *Curated GO relation CURIEs for ontology traversal*

Description

A named character vector mapping human-readable relation names to their CURIE representations as they appear in the semsql `entailed_edge` table. Counts are from a representative 2024 GO build:

Usage

```
GO_RELATIONS
```

Format

An object of class `character` of length 5.

Details

is_a `rdfs:subClassOf` — 1,360,314 entailed edges

part_of `BFO:0000050` — 353,639 edges

has_part `BFO:0000051` — 1,038,218 edges (inverse of part_of)

occurs_in `BFO:0000066` — 26,234 edges (biological process)

located_in `RO:0001025` — 1,459 edges (cellular component)

Pass one or more values from this vector as the `relations` argument to `go_ancestors` and `go_descendants`.

Examples

```
GO_RELATIONS
GO_RELATIONS["is_a"]
unname(GO_RELATIONS[c("is_a", "part_of")])
```

go_statements *Lazy tbl for the semsql statements table*

Description

The `statements` table contains all RDF triples from the GO OWL source, including term labels (`rdfs:label`), definitions (`IAO:0000115`), namespaces (`oio:hasOBONamespace`), and deprecation flags (`owl:deprecated`).

Usage

```
go_statements(con = NULL, schema = NULL)
```

Arguments

con	optional DBIConnection. If NULL (default), the package cache is used via .get_con().
schema	optional character schema name. If NULL (default), the cached schema is used.

Value

a lazy tbl_duckdb with columns stanza, subject, predicate, object, value, datatype, language.

Examples

```
make_go_con()
go_statements()
disconnect_go()
```

---

go_synonyms *Lazy tbl of GO term synonyms*

Description

Retrieves all synonym types for GO terms from the semsql statements table. Four synonym scopes are recognised by the OBO format and all are present in the GO semsql build: hasExactSynonym, hasRelatedSynonym, hasNarrowSynonym, and hasBroadSynonym.

Usage

```
go_synonyms(ids = NULL, types = c("exact", "related", "narrow", "broad"))
```

Arguments

ids	optional character vector of GO CURIEs to restrict results. If NULL (default), synonyms for all non-deprecated GO terms are returned.
types	character vector of synonym scopes to include. Default includes all four. Elements must be one or more of "exact", "related", "narrow", "broad".

Value

a lazy tbl_duckdb with columns:

id GO CURIE

synonym synonym string

scope one of "exact", "related", "narrow", "broad"

Call dplyr::collect() to materialize.

See Also

go_terms, select_go

Examples

```
GO.ddb::make_go_con()

# All synonyms for a term
GO.ddb::go_synonyms("GO:0006954") |> dplyr::collect()

# Exact synonyms only across all terms
GO.ddb::go_synonyms(types = "exact") |>
  dplyr::collect()

# Synonyms for multiple terms
GO.ddb::go_synonyms(c("GO:0006954", "GO:0008150")) |>
  dplyr::collect()

GO.ddb::disconnect_go()
```

---

go_terms *Lazy tbl of GO term metadata*

Description

Reconstructs term labels, definitions, ontology namespace (BP/MF/CC), and deprecation status from the semsql `statements` table. Filters to GO-prefixed identifiers, excluding imported terms from Uberon, CHEBI, RO, and other ontologies present in the GO OWL source.

Usage

```
go_terms(include_deprecated = FALSE, con = NULL, schema = NULL)
```

Arguments

`include_deprecated`	logical. If `FALSE` (default), obsolete GO terms (`owl:deprecated = "true"`) are excluded. Set `TRUE` to include them; the `deprecated` column is always present in the output regardless.
`con`	optional `DBIConnection` for testing.
`schema`	optional character schema name for testing.

Details

Uses `%like%` rather than `startsWith()` for the GO prefix filter — `startsWith()` is not translated by dbplyr to DuckDB's `starts_with()` function and will cause a catalog error.

Value

a lazy tbl_duckdb with columns:

id GO CURIE, e.g. "GO:0006954"

label human-readable term name

definition IAO:0000115 term definition

ontology namespace string: "biological_process", "molecular_function", or "cellular_component"

deprecated logical

See Also

lookup_curie, go_ancestors

Examples

```
make_go_con()

# All non-deprecated terms
go_terms()

# Biological process terms only
go_terms() |>
  dplyr::filter(ontology == "biological_process") |>
  dplyr::collect()

# Include deprecated terms
go_terms(include_deprecated = TRUE) |>
  dplyr::filter(deprecated) |>
  dplyr::select(id, label) |>
  dplyr::collect()

disconnect_go()
```

---

has_parquet_cache *Test whether a local parquet cache exists for an ontology Checks that the parquet cache directory exists and contains at least the two required files:* statements.parquet *and* entailed_edge.parquet.

Description

Test whether a local parquet cache exists for an ontology Checks that the parquet cache directory exists and contains at least the two required files: statements.parquet and entailed_edge.parquet.

Usage

```
has_parquet_cache(ontology = "go")
```

Arguments

ontology	character scalar. Default "go".

Value

logical scalar.

Examples

```
has_parquet_cache()
```

---

lookup_curie *Look up GO term metadata by CURIE*

Description

Maps one or more GO CURIEs to their term label, definition, or ontology namespace. Returns a lazy tibble — call dplyr::collect() to materialize results.

Usage

```
lookup_curie(curies, mapto = c("term", "definition", "ontology", "all"))
```

Arguments

curies	character vector of GO CURIEs in the form "GO:nnnnnnn", e.g. c("GO:0006954", "GO:0008150").
mapto	character scalar, one of: "term" term label (human-readable name) "definition" IAO:0000115 full definition "ontology" namespace: "biological_process", "molecular_function", or "cellular_component" "all" all three columns Partial matching is supported via match.arg().

Value

a lazy tbl_duckdb with column id and the requested field(s). Call dplyr::collect() to materialize.

See Also

go_terms, GO_RELATIONS

Examples

```
make_go_con()

lookup_curie(c("GO:0006954", "GO:0008150"), mapto = "term") |>
  dplyr::collect()

lookup_curie("GO:0006954", mapto = "all") |>
  dplyr::collect()

# partial matching works
lookup_curie("GO:0006954", mapto = "def") |>
  dplyr::collect()

disconnect_go()
```

`make_go_con` *Establish a connection to the GO semsql database*

Description

Retrieves GO data either from a local parquet cache or from the semsql SQLite file managed by `ontoProc2::semsql_connect()`, then loads it into an in-process DuckDB instance.

Usage

```
make_go_con(ontology = "go", backend = c("auto", "parquet", "sqlite"))
```

Arguments

`ontology`	character scalar. Default `"go"`.
`backend`	one of:
	`"auto"` use parquet if `has_parquet_cache` returns `TRUE`, otherwise SQLite
	`"parquet"` require parquet cache — error if absent
	`"sqlite"` use SQLite via DuckDB scanner and materialize hot tables into native DuckDB storage

Value

`NULL` invisibly.

See Also

`build_parquet_cache`, `has_parquet_cache`, `disconnect_go`

Examples

```
# auto selects parquet if available, SQLite otherwise
make_go_con()
go_connection_active()
disconnect_go()

# force parquet (must have run build_parquet_cache() first)
if (has_parquet_cache()) {
  make_go_con(backend = "parquet")
  disconnect_go()
}
```

`select_go` *Emulate AnnotationDbi::select for GO.db users*

Description

Provides a familiar interface for users migrating from `AnnotationDbi::select(GO.db, ...)`. Unlike the lazy tibbles returned by `go_terms` and `lookup_curie`, this function returns an eager `data.frame` to match the `AnnotationDbi` contract.

Usage

```
select_go(
  keys,
  columns = c("TERM", "DEFINITION", "ONTOLOGY"),
  keytype = "GOID"
)
```

Arguments

`keys`	character vector of GO CURIEs (e.g. `"GO:0006954"`).
`columns`	character vector of columns to return. Valid values: `"GOID"`, `"TERM"`, `"DEFINITION"`, `"ONTOLOGY"`, `"SYNONYM"`.
`keytype`	character scalar. Only `"GOID"` is currently supported, matching the GO.db constraint.

Value

a `data.frame` with column `GOID` and the requested additional columns, in the same format as `AnnotationDbi::select(GO.db, ...)`.

See Also

`lookup_curie`, `go_terms`

Examples

```
GO.ddb::make_go_con()

# Direct replacement for AnnotationDbi::select(GO.db, ...)
GO.ddb::select_go(
  keys    = c("GO:0006954", "GO:0008150"),
  columns = c("TERM", "ONTOLOGY")
)

# With synonyms
GO.ddb::select_go(
  keys    = "GO:0006954",
  columns = c("TERM", "DEFINITION", "ONTOLOGY", "SYNONYM")
)

GO.ddb::disconnect_go()
```

Index