Package 'nodbi'

March 6, 2024

Title 'NoSQL' Database Connector

Description Simplified document database access and manipulation, providing a common API across supported 'NoSQL' databases 'Elasticsearch', 'CouchDB', 'MongoDB' as well as 'SQLite/JSON1', 'PostgreSQL', and 'DuckDB'.

Version 0.10.4

License MIT + file LICENSE

LazyData true

URL https://docs.ropensci.org/nodbi/, https://github.com/ropensci/nodbi

BugReports https://github.com/ropensci/nodbi/issues

Depends R (>= 3.4.0)

Encoding UTF-8

Language en-US

Imports stringi, jsonlite, uuid, jqr, DBI, V8, R.utils

Suggests sofa (>= 0.3.0), elastic (>= 1.0.0), mongolite (>= 1.6), RSQLite (>= 2.3.5), duckdb (>= 0.6.0), RPostgres, testthat, withr, callr, webfakes (>= 1.2.0), knitr, rmarkdown, tibble

RoxygenNote 7.3.1

X-schema.org-applicationCategory Databases

X-schema.org-keywords database, MongoDB, Elasticsearch, CouchDB, SQLite, PostgreSQL, DuckDB, NoSQL, JSON, documents

X-schema.org-isPartOf https://ropensci.org

VignetteBuilder knitr

NeedsCompilation no

Author Ralf Herold [aut, cre] (<https://orcid.org/0000-0002-8148-6748>), Scott Chamberlain [aut] (<https://orcid.org/0000-0003-1444-9135>), Rich FitzJohn [aut], Jeroen Ooms [aut]

Maintainer Ralf Herold <ralf.herold@mailbox.org>

Repository CRAN

Date/Publication 2024-03-06 01:00:02 UTC

R topics documented:

`contacts` *Data set 'contacts'*

Description

Data set 'contacts'

Usage

```
contacts
```

Format

A JSON string with ragged, nested contact details

diamonds *Data set 'diamonds'*

Description

Data set 'diamonds'

Format

A data frame with 53940 rows and 10 variables:

- price price in US dollars (326-18,823 USD)
- carat weight of the diamond (0.2-5.01)
- cut quality of the cut (Fair, Good, Very Good, Premium, Ideal)
- color diamond colour, from J (worst) to D (best)
- clarity a measurement of how clear the diamond is (I1 (worst), SI1, SI2, VS1, VS2, VVS1, VVS2, IF (best))
- x length in mm (0-10.74)
- y width in mm (0-58.9)
- z depth in mm (0-31.8)
- depth total depth percentage = z / mean(x, y) = 2 * z / (x + y) (43-79)
- table width of top of diamond relative to widest point (43-95)

Source

from **ggplot2**

docdb_create *Create documents in a database*

Description

A message is emitted if the container `key` already exists.

Usage

```
docdb_create(src, key, value, ...)
```

Arguments

src	Source object, result of call to any of functions src_mongo(), src_sqlite(), src_elastic(), src_couchdb() src_duckdb() or src_postgres()
key	(character) The name of the container in the database backend (corresponds to collection for MongoDB, dbname for CouchDB, index for Elasticsearch, and to a table name for DuckDB, SQLite and PostgreSQL)
value	The data to be created in the database: a single data.frame, a JSON string, a list, or a file name or URL that points to NDJSON documents
...	Passed to functions sofa::db_bulk_create(), elastic::docs_bulk(), and mongolite::mongo()$insert()

Details

An error is raised for document(s) in value when their _id already exist(s) in the collection key; use docdb_update() to update such document(s).

Value

(integer) Number of successfully created documents

Identifiers

If value is a data.frame that has a column _id, or is a JSON string having a key _id at root level, or is a list having an item _id at its top level, this will be used as _id's and primary index in the database. If there are no such _id's in value, row names (if any exist) of value will be used as _id's, otherwise random _id's will be created (using uuid::UUIDgenerate() with use.time = TRUE for SQLite and PostgreSQL, or using DuckDB's built-in uuid()).

Examples

```
## Not run:
src <- src_sqlite()
docdb_create(src,
  key = "diamonds_small",
  value = as.data.frame(diamonds[1:3000L, ])
)
head(docdb_get(src, "diamonds_small"))
docdb_create(src, key = "contacts", value = contacts)
docdb_get(src, "contacts")[["friends"]]

## End(Not run)
```

docdb_delete *Delete documents or container*

Description

Delete documents or container

Usage

```
docdb_delete(src, key, ...)
```

Arguments

src	Source object, result of call to any of functions src_mongo(), src_sqlite(), src_elastic(), src_couchdb() src_duckdb() or src_postgres()
key	(character) The name of the container in the database backend (corresponds to collection for MongoDB, dbname for CouchDB, index for Elasticsearch, and to a table name for DuckDB, SQLite and PostgreSQL)
...	Optionally, specify query parameter with a JSON string as per docdb_query() to identify documents to be deleted. If not specified (default), deletes the container key.

Value

(logical) Success of operation. Typically TRUE if document(s) or collection existed, and FALSE if document(s) did not exist, or collection did not exist, or delete was not successful.

Examples

```
## Not run:
src <- src_sqlite()
docdb_create(src, "iris", iris)
docdb_delete(src, "iris", query = '{"Species": {"$regex": "a$"}}')
docdb_delete(src, "iris")

## End(Not run)
```

docdb_exists *Check if container exists in database*

Description

Check if container exists in database

Usage

```
docdb_exists(src, key, ...)
```

Arguments

src	Source object, result of call to any of functions src_mongo(), src_sqlite(), src_elastic(), src_couchdb() src_duckdb() or src_postgres()
key	(character) The name of the container in the database backend (corresponds to collection for MongoDB, dbname for CouchDB, index for Elasticsearch, and to a table name for DuckDB, SQLite and PostgreSQL)
...	Passed to functions DBI::dbListTables(), elastic::index_exists(), and sofa::db_info()

Value

(logical) TRUE or FALSE to indicate existence of container key in database. Note this does not indicate if the container holds any documents.

Examples

```
## Not run:
src <- src_sqlite()
docdb_exists(src, "nonexistingcontainer")
docdb_create(src, "mtcars", mtcars)
docdb_exists(src, "mtcars")

## End(Not run)
```

---

docdb_get *Get all documents from container in database*

Description

Get all documents from container in database

Usage

```
docdb_get(src, key, limit = NULL, ...)
```

Arguments

src	Source object, result of call to any of functions src_mongo(), src_sqlite(), src_elastic(), src_couchdb() src_duckdb() or src_postgres()
key	(character) The name of the container in the database backend (corresponds to collection for MongoDB, dbname for CouchDB, index for Elasticsearch, and to a table name for DuckDB, SQLite and PostgreSQL)
limit	(integer) Maximum number of documents to be returned. If NULL or not set (default), 10,000 for Elasticsearch and all documents for MongoDB, SQLite, CouchDB, PostgreSQL, and DuckDB.

... Passed on to functions:

- MongoDB: find() in mongolite::mongo()
- SQLite: ignored
- Elasticsearch: elastic::Search()
- CouchDB: sofa::db_alldocs()
- PostgreSQL: ignored
- DuckDB: ignored

Value

A data frame, one document per row

Examples

```
## Not run:
src <- src_sqlite()
docdb_create(src, "mtcars", mtcars)
docdb_get(src, "mtcars", limit = 10L)

## End(Not run)
```

---

docdb_list *List containers in database*

Description

List containers in database

Usage

```
docdb_list(src, ...)
```

Arguments

src	Source object, result of call to any of functions src_mongo(), src_sqlite(), src_elastic(), src_couchdb() src_duckdb() or src_postgres()
...	Passed to function DBI::dbListTables()

Value

(vector) Names of containers that can be used as parameter key with other functions such as docdb_create().

Examples

```
## Not run:
src <- src_sqlite()
docdb_create(src, "iris", iris)
docdb_list(src)

## End(Not run)
```

---

docdb_query *Get documents or parts with filtering query*

Description

Complements the databases' native query and filtering functions by using jqr. If `query = "{}"` and neither `fields` nor `listfields` is specified, runs `docdb_get()`.

Usage

```
docdb_query(src, key, query, ...)
```

Arguments

`src`	Source object, result of call to any of functions `src_mongo()`, `src_sqlite()`, `src_elastic()`, `src_couchdb()` `src_duckdb()` or `src_postgres()`
`key`	(character) The name of the container in the database backend (corresponds to `collection` for MongoDB, `dbname` for CouchDB, `index` for Elasticsearch, and to a table name for DuckDB, SQLite and PostgreSQL)
`query`	(character) A JSON query string, see examples. Can use comparisons / tests (`$lt`, `$lte`, `$gt`, `$gte`, `$ne`, `$in`, `$regex`), with logic operators (`$and`, `$or`, `(`, `)`), including nested queries, see examples.
`...`	Optional parameters: • Specify `fields` as a JSON string of fields to be returned from anywhere in the tree, or to be excluded from being returned, e.g. `fields = '{"nameOfMy.SubFieldToInclude:" 1, "_id": 0}'` and see examples. If `fields` is not specified, the complete JSON document is returned. For `src_postgres()`, only fewer than 50 fields can be requested to be returned by the function. • Specify `limit` (integer) for the maximum number of documents to be returned. If `NULL` or not set (default), 10,000 for Elasticsearch and all documents for MongoDB, SQLite, CouchDB, PostgreSQL, and DuckDB. • Specify `listfields = TRUE` to return just the names of all fields, from all documents or from the maximum number of documents as specified in `limit`.

Value

Data frame with requested documents, may have nested lists in columns; `NULL` if no documents could be found. If `listfields` is specified: a vector of all field names in dot path notation.

Note

A dot in `query` or `fields` is interpreted as a dot path, pointing to a field nested within another, e.g. `friends.id` in the example.

Examples

```
## Not run:
src <- src_sqlite()

docdb_create(src, "myKey", mtcars)
docdb_create(src, "myKey", contacts)
docdb_create(src, "myKey", mapdata)

docdb_query(src, "myKey", query = '{"mpg":21}')
docdb_query(src, "myKey", query = '{"mpg":21, "gear": {"$lte": 4}}')
docdb_query(src, "myKey", query = '{"mpg":21}', fields = '{"_id":0, "mpg":1, "cyl":1}')
docdb_query(src, "myKey", query = '{"_id": {"$regex": "^.+0.*$"}}', fields = '{"gear": 1}')

# complex query, not supported for src_elastic and src_couchdb backends at this time:
docdb_query(src, "myKey", query = '{"$and": [{"mpg": {"$lte": 18}}, {"gear": {"$gt": 3}}]}')
docdb_query(src, "myKey", query = '{}', fields = '{"_id":0, "mpg":1, "cyl":1}')

docdb_query(src, "myKey", query = '{"$and": [{"age": {"$gt": 21}},
 {"friends.name": {"$regex": "^B[a-z]{3,9}.*"}}]}')
docdb_query(src, "myKey", query = '{"$or": [{"rows.elements.status": "OK"}, {"$and": [
 {"_id": "5cd6785325ce3a94dfc54096"}, {"friends.name": {"$regex": "^B[a-z]{3,90}.*"}}]}]}')
docdb_query(src, "myKey", query = '{"$and": [{"_id": "5cd6785325ce3a94dfc54096"},
 {"friends.name": {"$regex": "^B[a-z]{3,90}.*"}}]}')
docdb_query(src, "myKey", query = '{"origin_addresses": {"$in": ["Santa Barbara, CA, USA",
 "New York, NY, USA"]}}', fields = '{"age": 1, "friends.id": 1, "_id": 0,
 "rows.elements.status": 1}')

docdb_query(src, "myKey", query = '{"rows.elements.status": "OK"}', listfields = TRUE)


## End(Not run)
```

---

docdb_update *Update documents*

Description

Documents are updated by patching their JSON with `value`. Documents are identified by a `query` or by `_id`'s in `value`, where the latter takes precedence. `value` can have multiple documents (with `_id`'s), which then are iteratively updated.

Usage

```
docdb_update(src, key, value, query, ...)
```

Arguments

src	Source object, result of call to any of functions src_mongo(), src_sqlite(), src_elastic(), src_couchdb() src_duckdb() or src_postgres()
key	(character) The name of the container in the database backend (corresponds to collection for MongoDB, dbname for CouchDB, index for Elasticsearch, and to a table name for DuckDB, SQLite and PostgreSQL)
value	The data to be created in the database: a single data.frame, a JSON string, a list, or a file name or URL that points to NDJSON documents
query	(character) A JSON query string, see examples. Can use comparisons / tests ($lt, $lte, $gt, $gte, $ne, $in, $regex), with logic operators ($and, $or, (,)), including nested queries, see examples.
...	Passed on to functions elastic::docs_bulk_update(), and mongolite::mongo()$update().

Details

Uses native functions in MongoDB (mongolite::mongo()$update()), SQLite (jsonb_update()), DuckDB (jsonb_merge_patch()), Elasticsearch (elastic::docs_bulk_update()); a plpgsql function added when calling src_postgres(), and a jqr programme for CouchDB.

Value

(integer) Number of successfully updated documents

Examples

```
## Not run:
src <- src_sqlite()
docdb_create(src, "mtcars", mtcars)
docdb_update(src, "mtcars", value = mtcars[3, 4:5], query = '{"gear": 3}')
docdb_update(src, "mtcars", value = '{"carb":999}', query = '{"gear": 5}')
docdb_update(src, "mtcars", value = '{"_id":"Fiat 128", "carb":888}', query = '{}')
docdb_get(src, "mtcars")

## End(Not run)
```

---

mapdata *Data set 'mapdata'*

Description

Data set 'mapdata'

Usage

```
mapdata
```

Format

A JSON string with ragged, nested travel details

`src` *Setup database connections*

Description

There is a `src_*()` function to setup a connection to each of the database backends. The backends may have specific parameters in the respective function `src_*()`, but all other `nodbi` functions are independent of the backend (e.g., see `docdb_query()`).

Details

- MongoDB - `src_mongo()`
- SQLite - `src_sqlite()`
- Elasticsearch - `src_elastic()`
- CouchDB - `src_couchdb()`
- PostgreSQL - `src_postgres()`
- DuckDB - `src_duckdb()`

Documentation details for each database:

- MongoDB - https://docs.mongodb.com/
- SQLite/JSON1 - https://www.sqlite.org/json1.html
- Elasticsearch - https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html
- CouchDB - http://docs.couchdb.org/
- PostgreSQL - https://www.postgresql.org/docs/current/functions-json.html
- DuckDB - https://duckdb.org/docs/extensions/json

Documentation of R packages used by `nodbi` for the databases:

- mongolite - https://CRAN.R-project.org/package=mongolite
- RSQLite - https://CRAN.R-project.org/package=RSQLite
- elastic - https://CRAN.R-project.org/package=elastic
- sofa - https://CRAN.R-project.org/package=sofa
- RPostgres - https://CRAN.R-project.org/package=RPostgres
- duckdb - https://CRAN.R-project.org/package=duckdb

`src_couchdb` *Setup a CouchDB database connection*

Description

Setup a CouchDB database connection

Usage

```
src_couchdb(
  host = "127.0.0.1",
  port = 5984,
  path = NULL,
  transport = "http",
  user = NULL,
  pwd = NULL,
  headers = NULL
)
```

Arguments

`host`	(character) host value, default: 127.0.0.1
`port`	(integer/numeric) Port. Remember that if you don't want a port set, set this parameter to NULL. Default: 5984
`path`	(character) context path that is appended to the end of the url, e.g., bar in http://foo.com/bar. Default: NULL, ignored
`transport`	(character) http or https. Default: http
`user`	(character) Username, if any
`pwd`	(character) Password, if any
`headers`	(list) list of named headers

Details

Uses **sofa** as backend. **nodbi** creates or uses a CouchDB database with JSON documents. If documents do not have root-level `_id`'s, UUID's are created as `_id`'s. Function `docdb_update()` uses `jqr::jqr()` to implement patching JSON. For a benchmark, see https://github.com/ropensci/nodbi#benchmark.

Value

A `nodbi` source object

Examples

```
## Not run:
con <- src_couchdb()
print(con)

## End(Not run)
```

`src_duckdb` *Setup a DuckDB database connection*

Description

Setup a DuckDB database connection

Usage

```
src_duckdb(drv = duckdb::duckdb(), dbdir = attr(drv, "dbdir"), ...)
```

Arguments

`drv`	Object returned by `duckdb()`
`dbdir`	Location for database files. Should be a path to an existing directory in the file system. With the default, all data is kept in RAM
`...`	Additional named parameters passed on to `DBI::dbConnect()`

Details

Uses `duckdb::duckdb()` as backend. **nodbi** creates or uses a DuckDB table, with columns `_id` and `json` created and used by package `nodbi`, applying SQL functions as per https://duckdb.org/docs/extensions/json to the `json` column. Each row in the table represents a `JSON` document. Any root-level `_id` is extracted from the document(s) and used for column `_id`, otherwise a UUID is created as `_id`. The table is indexed on `_id`. For a benchmark, see https://github.com/ropensci/nodbi#benchmark.

Value

A `nodbi` source object

Examples

```
## Not run:
con <- src_duckdb()
print(con)

## End(Not run)
```

`src_elastic` *Setup an Elasticsearch database connection*

Description

Setup an Elasticsearch database connection

Usage

```
src_elastic(
  host = "127.0.0.1",
  port = 9200,
  path = NULL,
  transport_schema = "http",
  user = NULL,
  pwd = NULL,
  force = FALSE,
  ...
)
```

Arguments

`host`	(character) the base url, defaults to localhost (http://127.0.0.1)
`port`	(character) port to connect to, defaults to 9200 (optional)
`path`	(character) context path that is appended to the end of the url. Default: `NULL`, ignored
`transport_schema`	(character) http or https. Default: http
`user`	(character) User name, if required for the connection. You can specify, but ignored for now.
`pwd`	(character) Password, if required for the connection. You can specify, but ignored for now.
`force`	(logical) Force re-load of connection details
`...`	Further args passed on to `elastic::connect()`

Details

Uses **elastic** as backend. **nodbi** creates or uses an Elasticsearch index, in which `nodbi` creates JSON documents. Any root-level `_id` is extracted from the document(s) and used as document ID `_id`, otherwise a UUID is created as document ID `_id`. Only lowercase is accepted for container names (in parameter `key`). Opensearch can equally be used. For a benchmark, see https://github.com/ropensci/nodbi#benchmark

Value

A `nodbi` source object

Examples

```
## Not run:
con <- src_elastic()
print(con)

## End(Not run)
```

`src_mongo` *Setup a MongoDB database connection*

Description

Setup a MongoDB database connection

Usage

```
src_mongo(collection = "test", db = "test", url = "mongodb://localhost", ...)
```

Arguments

`collection`	(character) Name of collection
`db`	(character) Name of database
`url`	(character) Address of the MongoDB server in Mongo connection string URI format, see to `mongolite::mongo()`
`...`	Additional named parameters passed on to `mongolite::mongo()`

Details

Uses **monoglite** as backend. **nodbi** creates or uses a MongoDB collection, in which `nodbi` creates JSON documents. If documents do not have root-level `_id`'s, UUID's are created as `_id`'s. MongoDB but none of the other databases require to specify the container already in the `src_mongo()` function. For a benchmark, see `https://github.com/ropensci/nodbi#benchmark`

Value

A `nodbi` source object

Examples

```
## Not run:
con <- src_mongo()
print(con)

## End(Not run)
```

src_postgres *Setup a PostgreSQL database connection*

Description

Setup a PostgreSQL database connection

Usage

```
src_postgres(dbname = "test", host = "localhost", port = 5432L, ...)
```

Arguments

dbname	(character) name of database, has to exist to open a connection
host	(character) host of the database, see RPostgres::Postgres()
port	(integer) port of the database, see RPostgres::Postgres()
...	additional named parameters passed on to RPostgres::Postgres()

Details

Uses **RPostgres** as backend. **nodbi** creates or uses a PostgreSQL table, with columns `_id` and `json` created and used by package `nodbi`, applying SQL functions as per https://www.postgresql.org/docs/current/functions-json.html to the `json` column. Each row in the table represents a `JSON` document. Any root-level `_id` is extracted from the document(s) and used for column `_id`, otherwise a UUID is created as `_id`. The table is indexed on `_id`. A custom `plpgsql` function jsonb_merge_patch() is used for `docdb_update()`. The order of variables in data frames returned by `docdb_get()` and `docdb_query()` can differ from their order the input to `docdb_create()`. For a benchmark, see https://github.com/ropensci/nodbi#benchmark

Value

A `nodbi` source object

Examples

```
## Not run:
con <- src_postgres()
print(con)

## End(Not run)
```

src_sqlite *Setup a RSQLite database connection*

Description

Setup a RSQLite database connection

Usage

```
src_sqlite(dbname = ":memory:", ...)
```

Arguments

dbname	(character) name of database file, defaults to ":memory:" for an in-memory database, see RSQLite::SQLite()
...	additional named parameters passed on to RSQLite::SQLite()

Details

Uses **RSQLite** as backend. **nodbi** creates or uses an SQLite table, with columns `_id` and `json` created and used by package `nodbi`, applying SQL functions as per https://www.sqlite.org/json1.html to the `json` column. Each row in the table represents a `JSON` document. Any root-level `_id` is extracted from the document(s) and used for column `_id`, otherwise a UUID is created as `_id`. The table is indexed on `_id`. For a benchmark, see https://github.com/ropensci/nodbi#benchmark

Value

A `nodbi` source object

Examples

```
## Not run:
con <- src_sqlite()
print(con)

## End(Not run)
```

Index