Package ‘MsBackendSql’

May 10, 2024

Title SQL-based Mass Spectrometry Data Backend

Version 1.4.0

Description SQL-based mass spectrometry (MS) data backend supporting also storage and handling of very large data sets. Objects from this package are supposed to be used with the Spectra Bioconductor package. Through the MsBackendSql with its minimal memory footprint, this package thus provides an alternative MS data representation for very large or remote MS data sets.

Depends R (>= 4.2.0), Spectra (>= 1.9.12)

Imports BiocParallel, S4Vectors, methods, ProtGenerics (>= 1.35.3), DBI, MsCoreUtils, IRanges, data.table, progress, BiocGenerics

Suggests testthat, knitr (>= 1.1.0), roxygen2, BiocStyle (>= 2.5.19), RSQLite, msdata, rmarkdown, microbenchmark, mzR

License Artistic-2.0

Encoding UTF-8

VignetteBuilder knitr

BugReports https://github.com/RforMassSpectrometry/MsBackendSql/issues

URL https://github.com/RforMassSpectrometry/MsBackendSql

biocViews Infrastructure, MassSpectrometry, Metabolomics, DataImport, Proteomics

Roxygen list(markdown=TRUE)

RoxygenNote 7.3.1

Collate 'MsBackendSql-functions.R' 'MsBackendSql.R' 'MsBackendOfflineSql.R'

git_url https://git.bioconductor.org/packages/MsBackendSql

git_branch RELEASE_3_19

git_last_commit 4b51add

git_last_commit_date 2024-04-30

Repository Bioconductor 3.19
MsBackendOfflineSql

SQL-based MS backend without active database connection

Description

The MsBackendOfflineSql backend extends the MsBackendSql() backend directly and inherits thus all of its functions as well as properties. The only difference between the two backend is that MsBackendSql keeps an active connection to the SQL database inside the object while the MsBackendOfflineSql backends reconnects to the SQL database for each query. While the performance of the latter is slightly lower (due to the need to connect/disconnect to the database for each function call) it can also be used in a parallel processing environment.

Usage

MsBackendOfflineSql()

## S4 method for signature 'MsBackendOfflineSql'
backendInitialize(
  object,
  drv = NULL,
  dbname = character(),
  user = character(),
  password = character(),
  host = character(),
  port = NA_integer_,
  data,
  ...
)

Arguments

object A MsBackendOfflineSql object.
Creation of backend objects

An empty instance of an `MsBackendOfflineSql` class can be created using the `MsBackendOfflineSql()` function. An existing `MsBackendSql` SQL database can be loaded with the `backendInitialize()` function. This function takes parameters `drv`, `dbname`, `user`, `password`, `host` and `port`, all parameters that are passed to the `dbConnect()` function to connect to the (existing) SQL database.

See `MsBackendSql()` for information on how to create a `MsBackend` SQL database.

Author(s)

Johannes Rainer

---

**Description**

The `MsBackendSql` is an implementation for the `MsBackend()` class for `Spectra()` objects which stores and retrieves MS data from a SQL database. New databases can be created from raw MS data files using `createMsBackendSqlDatabase()`.

**Usage**

```r
MsBackendSql()

createMsBackendSqlDatabase(
  dbcon,
  x = character(),
  backend = MsBackendMzR(),
)```
chunksize = 10L,
blob = TRUE,
partitionBy = c("none", "spectrum", "chunk"),
partitionNumber = 10L
)
## S4 method for signature 'MsBackendSql'
show(object)
## S4 method for signature 'MsBackendSql'
backendInitialize(object, dbcon, data, ...)
## S4 method for signature 'MsBackendSql'
dataStorage(object)
## S4 method for signature 'MsBackendSql'
x[i, j, ..., drop = FALSE]
## S4 method for signature 'MsBackendSql'
peaksData(object, columns = c("mz", "intensity"))
## S4 method for signature 'MsBackendSql'
peaksVariables(object)
## S4 replacement method for signature 'MsBackendSql'
intensity(object) <- value
## S4 replacement method for signature 'MsBackendSql'
mz(object) <- value
## S4 replacement method for signature 'MsBackendSql'
x$name <- value
## S4 method for signature 'MsBackendSql'
spectraData(object, columns = spectraVariables(object))
## S4 method for signature 'MsBackendSql'
reset(object)
## S4 method for signature 'MsBackendSql'
spectraNames(object)
## S4 replacement method for signature 'MsBackendSql'
spectraNames(object) <- value
## S4 method for signature 'MsBackendSql'
filterMsLevel(object, msLevel = uniqueMsLevels(object))
## MsBackendSql

### Methods

```r
## S4 method for signature 'MsBackendSql'
filterRt(object, rt = numeric(), msLevel. = integer())

## S4 method for signature 'MsBackendSql'
filterDataOrigin(object, dataOrigin = character())

## S4 method for signature 'MsBackendSql'
filterPrecursorMzRange(object, mz = numeric())

## S4 method for signature 'MsBackendSql'
filterPrecursorMzValues(object, mz = numeric(), ppm = 20, tolerance = 0)

## S4 method for signature 'MsBackendSql'
uniqueMsLevels(object, ...)

## S4 method for signature 'MsBackendSql'
backendMerge(object, ...)

## S4 method for signature 'MsBackendSql'
precScanNum(object)

## S4 method for signature 'MsBackendSql'
centroided(object)

## S4 method for signature 'MsBackendSql'
smoothed(object)

## S4 method for signature 'MsBackendSql'
tic(object, initial = TRUE)

## S4 method for signature 'MsBackendSql'
supportsSetBackend(object, ...)

## S4 method for signature 'MsBackendSql'
backendBpparam(object, BPPARAM = bpparam())
```

### Arguments

- **dbcon**: Connection to a database.
- **x**: For `createMsBackendSqlDatabase()`: character with the names of the raw data files from which the data should be imported. For other methods an `MsqlBackend` instance.
- **backend**: For `createMsBackendSqlDatabase()`: MS backend that can be used to import MS data from the raw files specified with parameter `x`.
- **chunksize**: For `createMsBackendSqlDatabase()`: integer(1) defining the number of
input that should be processed per iteration. With chunksize = 1 each file
specified with x will be imported and its data inserted to the database. With
chunksize = 5 data from 5 files will be imported (in parallel) and inserted to
the database. Thus, higher values might result in faster database creation, but
require also more memory.

**blob**
For createMsBackendSqlDatabase(): logical(1) whether individual m/z
and intensity values should be stored separately (blob = FALSE) or if the m/z
and intensity values for each spectrum should be stored as a single BLOB SQL
data type (blob = TRUE, the default).

**partitionBy**
For createMsBackendSqlDatabase(): character(1) defining if and how the
peak data table should be partitioned. "none" (default): no partitioning, "spectrum":
peaks are assigned to the partition based on the spectrum ID (number), i.e. spec-
tra are evenly (consecutively) assigned across partitions. For partitionNumber
= 3, the first spectrum is assigned to the first partition, the second to the sec-
ond, the third to the third and the fourth spectrum again to the first partition.
"chunk": spectra processed as part of the same chunk are placed into the same
partition. All spectra from the next processed chunk are assigned to the next
partition. Note that this is only available for MySQL/MariaDB databases, i.e.,
if con is a MariaDBConnection. See details for more information.

**partitionNumber**
For createMsBackendSqlDatabase(): integer(1) defining the number of
partitions the database table will be partitioned into (only supported for MySQL/MariaDB
databases).

**object**
A MsBackendSql instance.

**data**
For backendInitialize(): optional DataFrame with the full spectra data that
should be inserted into a (new) MsBackendSql database. If provided, it is as-
sumed that dbcon is a (writeable) connection to an empty database into which
data should be inserted. data could be the output of spectraData from another
backend.

... For :: ignored. For backendInitialize, if parameter data is used: additional
parameters to be passed to the function creating the database such as blob.

**i**
For :: integer or logical to subset the object.

**j**
For :: ignored.

**drop**
For :: logical(1), ignored.

**columns**
For spectraData(): character() optionally defining a subset of spectra vari-
ables that should be returned. Defaults to columns = spectraVariables(object)
​hence all variables are returned. For peaksData accessor: optional character
with requested columns in the individual matrix of the returned list. Defaults
to columns = c("mz", "intensity") but all columns listed by peaksVariables
would be supported.

**value**
For all setter methods: replacement value.

**name**
For <- : character(1) with the name of the spectra variable to replace.

**msLevel**
For filterMsLevel(): integer specifying the MS levels to filter the data.

**rt**
For filterRt(): numeric(2) with the lower and upper retention time. Spectra
with a retention time >= rt[1] and <= rt[2] are returned.
MsBackendSql

msLevel: For filterRt(): integer with the MS level(s) on which the retention time filter should be applied. For filterDataOrigin(): character with data origin values to which the data should be subsetted.

dataOrigin

mz: For filterPrecursorMzRange(): numeric(2) with the desired lower and upper limit of the precursor m/z range. For filterPrecursorMzValues(): numeric with the m/z value(s) to filter the object.

ppm: For filterPrecursorMzValues(): numeric with the m/z-relative maximal acceptable difference for a m/z value to be considered matching. Can be of length 1 or equal to length(mz).

tolerance: For filterPrecursorMzValues(): numeric with the absolute difference for m/z values to be considered matching. Can be of length 1 or equal to length(mz).

initial: For tic(): logical(1) whether the original total ion count should be returned (initial = TRUE, the default) or whether it should be calculated on the spectra’s intensities (initial = FALSE).

BPPARAM: for backendBpparam(): BiocParallel parallel processing setup. See bpparam() for more information.

Details

The MsBackendSql class is principally a read-only backend but by extending the MsBackendCached() backend from the Spectra package it allows changing and adding (temporarily) spectra variables without changing the original data in the SQL database.

Value

See documentation of respective function.

Creation of backend objects

New backend objects can be created with the MsBackendSql() function. SQL databases can be created and filled with MS data from raw data files using the createMsBackendSqlDatabase() function or using backendInitialize() and providing all data with parameter data. In addition it is possible to create a database from a Spectra object changing its backend to a MsBackendSql or MsBackendOfflineSql using the setBackend() function. Existing SQL databases (created previously with createMsBackendSqlDatabase() or backendInitialize() with the data parameter) can be loaded using the conventional way to create/initialize MsBackend classes, i.e. using backendInitialize().

- createMsBackendSqlDatabase(): create a database and fill it with MS data. Parameter dbcon is expected to be a database connection, parameter x a character vector with the file names from which to import the data. Parameter backend is used for the actual data import and defaults to backend = MsBackendMzR() hence allowing to import data from mzML, mzXML or netCDF files. Parameter chunksize allows to define the number of files (x) from which the data should be imported in one iteration. With the default chunksize = 10L data is imported from 10 files in x at the same time (if backend supports it even in parallel) and this data is then inserted into the database. Larger chunk sizes will require more memory and also larger disk space (as data import is performed through temporary files) but might eventually be faster. Parameter blob allows to define whether m/z and intensity values from a spectrum should be
stored as a **BLOB** SQL data type in the database (blob = TRUE, the default) or if individual m/z and intensity values for each peak should be stored separately (blob = FALSE). The latter case results in a much larger database and slower performance of the peaksData function, but would allow to define custom (manual) SQL queries on individual peak values. While data can be stored in any SQL database, at present it is suggested to use MySQL/MariaDB databases. For dbcon being a connection to a MySQL/MariaDB database, the tables will use the **ARIA** engine providing faster data access and will use table partitioning: tables are split into multiple partitions which can improve data insertion and index generation. Partitioning can be defined with the parameters partitionBy and partitionNumber. By default partitionBy = "none" no partitioning is performed. For blob = TRUE partitioning is usually not required. Only for blob = FALSE and very large datasets it is suggested to enable table partitioning by selecting either partitionBy = "spectrum" or partitionBy = "chunk". The first option assigns consecutive spectra to different partitions while the latter puts spectra from files part of the same chunk into the same partition. Both options have about the same performance but partitionBy = "spectrum" requires less disk space. Note that, while inserting the data takes a considerable amount of time, also the subsequent creation of database indices can take very long (even longer than data insertion for blob = FALSE).

- **backendInitialize()**: get access and initialize a MsBackendSql object. Parameter object is supposed to be a MsBackendSql instance, created e.g. with MsBackendSql(). Parameter dbcon is expected to be a connection to an existing MsBackendSql SQL database (created e.g. with createMsBackendSqlDatabase()). backendInitialize() can alternatively also be used to create a new MsBackendSql database using the optional data parameter. In this case, dbcon is expected to be a writeable connection to an empty database and data a DataFrame with the full spectra data to be inserted into this database. The format of data should match the format of the DataFrame returned by the spectraData() function and requires columns "mz" and "intensity" with the m/z and intensity values of each spectrum. The backendInitialize() call will then create all necessary tables in the database, will fill these tables with the provided data and will return an MsBackendSql for this database. Thus, the MsBackendSql supports the setBackend method from Spectra to change from (any) backend to a MsBackendSql. Note however that chunk-wise (or parallel) processing needs to be disabled in this case by passing eventually f = factor() to the setBackend() call.

- **supportsSetBackend()**: whether MsBackendSql supports the setBackend() method to change the MsBackend of a Spectra object to a MsBackendSql. Returns TRUE, thus, changing the backend to a MsBackendSql is supported if a writeable database connection is provided in addition with parameter dbcon (i.e. setBackend(sps, MsBackendSql(), dbcon = con) with con being a connection to an empty database) or would store the full spectra data from the Spectra object sps into the specified database and would return a Spectra object that uses a MsBackendSql).

- **backendBpparam()**: whether a MsBackendSql supports parallel processing. Takes a MsBackendSql and a parallel processing setup (see bpparam() for details) as input and always returns a SerialParam() since MsBackendSql does not support parallel processing.

- **dbconn()**: returns the connection to the database.

**Subsetting, merging and filtering data**

MsBackendSql objects can be subsetted using the [ function. Internally, this will simply subset the integer vector of the primary keys and eventually cached data. The original data in the database is not affected by any subsetting operation. Any subsetting operation can be undone by resetting
the object with the `reset()` function. Subsetting in arbitrary order as well as index replication is supported.

Multiple `MsBackendSql` objects can also be merged (combined) with the `backendMerge()` function. Note that this requires that all `MsBackendSql` objects are connected to the same database. This function is thus mostly used for combining `MsBackendSql` objects that were previously splitted using e.g. `split()`.

In addition, `MsBackendSql` supports all other filtering methods available through `MsBackendCached()`. Implementation of filter functions optimized for `MsBackendSql` objects are:

- **filterDataOrigin()**: filters the object retaining spectra with `dataOrigin` spectra variable values matching the provided ones with parameter `dataOrigin`. The function returns the results in the order of the values provided with parameter `dataOrigin`.
- **filterMsLevel()**: filters the object based on the MS levels specified with parameter `msLevel`. The function does the filtering using SQL queries. If "msLevel" is a local variable stored within the object (and hence in memory) the default implementation in `MsBackendCached` is used instead.
- **filterPrecursorMzRange()**: filters the data keeping only spectra with a precursor m/z within the m/z value range provided with parameter `mz` (i.e. all spectra with a precursor m/z \(\geq m[z][1L]\) and \(\leq m[z][2L]\)).
- **filterPrecursorMzValues()**: filters the data keeping only spectra with precursor m/z values matching the values provided with parameter `mz`. Parameters `ppm` and `tolerance` allow to specify acceptable differences between compared values. Lengths of `ppm` and `tolerance` can be either 1 or equal to `length(mz)` to use different values for ppm and tolerance for each provided m/z value.
- **filterRt()**: filters the object keeping only spectra with retention times within the specified retention time range (parameter `rt`). Optional parameter `msLevel` allows to restrict the retention time filter only on the provided MS level(s) returning all spectra from other MS levels.

Accessing and modifying data

The functions listed here are specifically implemented for `MsBackendSql`. In addition, `MsBackendSql` inherits and supports all data accessor, filtering functions and data manipulation functions from `MsBackendCached()

- `$`, `$<-`: access or set (add) spectra variables in object. Spectra variables added or modified using the `$<-` are cached locally within the object (data in the database is never changed). To restore an object (i.e. drop all cached values) the `reset` function can be used.
- `dataStorage()`: returns a character vector same length as there are spectra in object with the name of the database containing the data.
- `intensity<-`: not supported.
- `mz<-`: not supported.
- `peaksData()`: returns a list with the spectras’ peak data. The length of the list is equal to the number of spectra in object. Each element of the list is a matrix with columns according to parameter `columns`. For an empty spectrum, a matrix with 0 rows is returned. Use `peaksVariables(object)` to list supported values for parameter `columns`.
- `peaksVariables()`: returns a character with the available peak variables, i.e. columns that could be queried with `peaksData()`.
- `reset()`: restores an `MsBackendSql` by re-initializing it with the data from the database. Any subsetting or cached spectra variables will be lost.
• `spectraData()`: gets general spectrum metadata. `spectraData()` returns a `DataFrame` with the same number of rows as there are spectra in object. Parameter `columns` allows to select specific spectra variables.

• `spectraNames()`, `spectraNames<-`: returns a character of length equal to the number of spectra in object with the primary keys of the spectra from the database (converted to character). Replacing spectra names with `spectraNames<-` is not supported.

• `uniqueMsLevels()`: returns the unique MS levels of all spectra in object.

• `tic()`: returns the originally reported total ion count (for `initial = TRUE`) or calculates the total ion count from the intensities of each spectrum (for `initial = FALSE`).

**Implementation notes**

Internally, the `MsBackendSql` class contains only the primary keys for all spectra stored in the SQL database. Keeping only these integer in memory guarantees a minimal memory footprint of the object. Still, depending on the number of spectra in the database, this integer vector might become very large. Any data access will involve SQL calls to retrieve the data from the database. By extending the `MsBackendCached()` object from the `Spectra` package, the `MsBackendSql` supports to (temporarily, i.e. for the duration of the R session) add or modify spectra variables. These are however stored in a `data.frame` within the object thus increasing the memory demand of the object.

**Note**

The `MsBackendSql` backend keeps an (open) connection to the SQL database with the data and hence does not support saving/loading of a backend to disk (e.g. using `save` or `saveRDS`). Also, for the same reason, the `MsBackendSql` does not support parallel processing. The `backendBpparam()` method for `MsBackendSql` will thus always return a `SerialParam()` object.

The `MsBackendOfflineSql()` could be used as an alternative as it supports saving/loading the data to/from disk and supports also parallel processing.

**Author(s)**

Johannes Rainer

**Examples**

```r
# Create a new MsBackendSql database

# Define a file from which to import the data
data_file <- system.file("microtofq", "MM8.mzML", package = "msdata")

# Create a database/connection to a database
library(RSQLite)
db_file <- tempfile()
dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQLite(), db_file)

dbc <- dbConnect(SQ...
## Initialize a MsBackendSql

dbc <- dbConnect(SQLite(), db_file)
be <- backendInitialize(MsBackendSql(), dbc)

be

## Original data source
head(be$dataOrigin)

## Data storage
head(dataStorage(be))

## Access all spectra data
spd <- spectraData(be)
spd

## Available variables
spectraVariables(be)

## Access mz values
mz(be)

## Subset the object to spectra in arbitrary order
be_sub <- be[c(5, 1, 1, 2, 4, 100)]
be_sub

## The internal spectrum IDs (primary keys from the database)
be_sub$spectrum_id_

## Add additional spectra variables
be_sub$new_variable <- "B"

## This variable is *cached* locally within the object (not inserted into
## the database)
be_sub$new_variable
Index

[,MsBackendSql-method (MsBackendSql), 3
$<-,MsBackendSql-method (MsBackendSql), 3
backendBpparam,MsBackendSql-method (MsBackendSql), 3
backendInitialize,MsBackendOfflineSql-method (MsBackendSql), 2
backendInitialize,MsBackendSql-method (MsBackendSql), 3
backendMerge,MsBackendOfflineSql-method (MsBackendSql), 3
backendMerge,MsBackendSql-method (MsBackendSql), 3
bpparam(), 7, 8
centroided,MsBackendSql-method (MsBackendSql), 3
createMsBackendSqlDatabase (MsBackendSql), 3
dataStorage,MsBackendOfflineSql-method (MsBackendSql), 3
dataStorage,MsBackendSql-method (MsBackendSql), 3
dbconn,MsBackendOfflineSql-method (MsBackendSql), 3
dbconn,MsBackendSql-method (MsBackendSql), 3
dbConnect(), 3
filterDataOrigin,MsBackendOfflineSql-method (MsBackendSql), 3
filterDataOrigin,MsBackendSql-method (MsBackendSql), 3
filterMsLevel,MsBackendOfflineSql-method (MsBackendSql), 3
filterMsLevel,MsBackendSql-method (MsBackendSql), 3
filterPrecursorMzRange,MsBackendSql-method (MsBackendSql), 3
filterPrecursorMzRange,MsBackendOfflineSql-method (MsBackendSql), 3
filterPrecursorMzValues,MsBackendOfflineSql-method (MsBackendSql), 3
filterPrecursorMzValues,MsBackendSql-method (MsBackendSql), 3
filterRt,MsBackendOfflineSql-method (MsBackendSql), 3
filterRt,MsBackendSql-method (MsBackendSql), 3
intensity<-,MsBackendSql-method (MsBackendSql), 3
MsBackend(), 3
MsBackendCached(), 7, 9, 10
MsBackendOfflineSql, 2
MsBackendOfflineSql(), 10
MsBackendOfflineSql-class (MsBackendOfflineSql), 2
MsBackendSql, 3
MsBackendSql(), 2, 3
MsBackendSql-class (MsBackendSql), 3
mz<-,MsBackendSql-method (MsBackendSql), 3
peaksData,MsBackendOfflineSql-method (MsBackendSql), 3
peaksData,MsBackendSql-method (MsBackendSql), 3
peaksVariables,MsBackendOfflineSql-method (MsBackendSql), 3
peaksVariables,MsBackendSql-method (MsBackendSql), 3
precScanNum,MsBackendSql-method (MsBackendSql), 3
reset,MsBackendOfflineSql-method (MsBackendSql), 3
reset,MsBackendSql-method (MsBackendSql), 3
filterPrecursorMzRange,MsBackendOfflineSql-method (MsBackendSql), 3
filterPrecursorMzRange,MsBackendSql-method (MsBackendSql), 3

12
SerialParam(), 8, 10
setBackend(), 7
show, MsBackendOfflineSql-method
  (MsBackendSql), 3
show, MsBackendSql-method
  (MsBackendSql), 3
smoothed, MsBackendSql-method
  (MsBackendSql), 3
Spectra(), 3
spectraData, MsBackendOfflineSql-method
  (MsBackendSql), 3
spectraData, MsBackendSql-method
  (MsBackendSql), 3
spectraNames, MsBackendSql-method
  (MsBackendSql), 3
spectraNames<-, MsBackendSql-method
  (MsBackendSql), 3
supportsSetBackend, MsBackendOfflineSql-method
  (MsBackendSql), 3
supportsSetBackend, MsBackendSql-method
  (MsBackendSql), 3
tic, MsBackendOfflineSql-method
  (MsBackendSql), 3
tic, MsBackendSql-method (MsBackendSql), 3
uniqueMsLevels, MsBackendOfflineSql-method
  (MsBackendSql), 3
uniqueMsLevels, MsBackendSql-method
  (MsBackendSql), 3