Package: xml2relational (via r-universe)

August 20, 2024

Type Package

Title Converting XML Documents into Relational Data Models

Description Import an XML document with nested object structures and convert it into a relational data model. The result is a set of R dataframes with foreign key relationships. The data model and the data can be exported as SQL code of different SQL flavors.

Version 0.1.0

Maintainer Joachim Zuckarelli <joachim@zuckarelli.de>

Depends R (>= 3.5.0)

License GPL-3

Imports xml2, stringr, tidyr, fs, stats, utils, lubridate, rlang

BugReports https://github.com/jsugarelli/xml2relational/issues

URL https://github.com/jsugarelli/xml2relational/

Encoding UTF-8

ByteCompile true

RoxygenNote 7.1.0

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

RemoteUrl https://github.com/jsugarelli/xml2relational

RemoteRef HEAD

RemoteSha f8254675ff427566461c6638b55058edc3592a7b

Contents

getCreateSQL *Exporting the relational data model and data to a database*

Description

Produces ready-to-run SQL INSERT statements to import the data transformed with toRelational() into a SQL database.

Usage

```
getCreateSQL(
  ldf,
  sql.style = "MySQL",
  tables = NULL,
  prefix.primary = "ID_",
  prefix.foreign = "FKID_",
  line.break = "\n",
  datatype.func = NULL,
  one.statement = FALSE
)
```

Arguments

ldf	A **l**ist of **d**ata**f**rames created by toRelational() (the data tables transformed from XML to a relational schema).
sql.style	The SQL flavor that the produced CREATE statements will follow. The supported SQL styles are "MySQL", "TransactSQL" and "Oracle". You can add your own SQL flavor by providing a dataframe with the required information instead of the name of one of the predefined SQL flavors as value for sql.style. See the Details section for more information on working with different SQL flavors.
tables	A character vector with the names of the tables for whichs SQL CREATE statements will be produced. If null (default) CREATE statements will be produced for all tables in in the relational data model of ldf.
prefix.primary	The prefix that is used in the relational data model of ldf to identify primary keys. "ID_" by default.
prefix.foreign	The prefix that is used in the relational data model of ldf to identify foreign keys. "FKID_" by default.
line.break	Line break character that is added to the end of each CREATE statement (apart from the semicolon that is added automatically). Default is "\n".
datatype.func	A function that is used to determine the data type of the table fields. The function must take the field/column from the data table (basically the result of SELCT field FROM table) as its sole argument and return a character vector providing the data type. If null (default), the built-in mechanism will be used to determine the data type.

one.statement Determines whether all CREATE statements will be returned as one piece of SQL code (one.statement = TRUE) or if each CREATE statement will be stored in a separate element of the return vector.

Details

If you want to produce SQL CREATE statements that follow a different SQL dialect than one of the built-in SQL flavors (i.e. MySQL, TransactSQL and Oracle) you can provide the necessary information to getCreateSQL() via the sql.style argument. In this case the sql.style argument needs to be a dataframe with the folling fields:

Column	Type	Description
Style	character	Name of the SQL flavor.
NormalField	character	Template string for a normal, nullable field.
NormalFieldNotNull	character	Template string for non-nullable field.
PrimaryKey	character	Template string for the definition of a primary key.
ForeignKey	character	Template string for the definition of a foreign key. "FOREIGN KEY (%FIELDNAME%)
PrimaryKeyDefSeparate	logical	Indicates if primary key needs additional definition like a any other field.
ForeignKeyDefSeparate	logical	Indicates if foreign key needs additional definition like a any other field.
Int		character
Int.MaxSize	numeric	Size limit of integer data type.
BigInt	character	Name of data type for integers larger than the size limit of the normal integer data t
Decimal	character	Name of data type for floating point numbers.
VarChar	character	Name of data type for variable-size character fields.
VarChar.MaxSize	numeric	Size limit of variable-size character data type.
Text	character	Name of data type for string data larger than the size limit of the variable-size char
Date	character	Name of data type date data.
Time	character	Name of data type time data
Date	character	Name of data type for combined date and time data.

In the template strings you can use the following placeholders, as you also see from the MySQL example in the table:

1. %FIELDNAME%: Name of the field to be defined.
2. %DATATYPE%: Datatype of the field to be defined.
3. %REFTABLE%: Table referenced by a foreign key.
4. %REFPRIMARYKEY%: Name of the primary key field of the table referenced by a foreign key.

When you use your own defintion of an SQL flavor, then sql.style must be a one-row dataframe providing the fields described in the table above.

You can use the datatype.func argument to provide your own function to determine how the data type of a field is derived from the values in that field. In this case, the values of the columns Int, Int.MaxSize, VarChar, VarChar.MaxSize, Decimal and Text in the sql.style dataframe are ignored. They are used by the built-in mechanism to determine data types. Providing your own function allows you to determine data types in a more differentiated way, if you like. The function that is provided needs to take a vectors of values as its argument and needs to provide the SQL data type of these values as a one-element character vector.

Value

A character vector with exactly one element (if argument `one.statement = TRUE`) or with one element per `CREATE` statement.

See Also

Other xml2relational: `getInsertSQL()`, `savetofiles()`, `toRelational()`

Examples

```
# Find path to custmers.xml example file in package directory
path <- system.file("", "customers.xml", package = "xml2relational")
db <- toRelational(path)

sql.code <- getCreateSQL(db, "TransactSQL", "address")
```

`getInsertSQL` *Exporting the relational data model and data to a database*

Description

Produces ready-to-run SQL `INSERT` statements to import the data transformed with `toRelational()` into a SQL database.

Usage

```
getInsertSQL(
  ldf,
  table.name,
  line.break = "\n",
  one.statement = FALSE,
  tz = "UTC"
)
```

Arguments

`ldf`	A **l**ist of **d**ata**f**rames created by `toRelational()` (the data tables transformed from XML to a relational schema).
`table.name`	Name of the table from the data table list `ldf` for which `INSERT` statements are to be created.
`line.break`	Line break character that is added to the end of each `INSERT` statement (apart from the semicolon that is added automatically). Default is `"\n"`.

one.statement	Determines whether all INSERT statements will be returned as one piece of SQL code (one.statement = TRUE) or if each INSERT statement will be stored in a separate element of the return vector. In the former case the return vector will have just one element, in the latter case as many elements as there are data records to insert. Default is FALSE (return vector has one element per INSERT statement.
tz	The code of the timezone used for exporting timestamp data. Default it "UTC" (Coordinated Universal Time).

Value

A character vector with exactly one element (if argument one.statement = TRUE) or with one element per INSERT statement.

See Also

Other xml2relational: getCreateSQL(), savetofiles(), toRelational()

Examples

```
# Find path to custmers.xml example file in package directory
path <- system.file("", "customers.xml", package = "xml2relational")
db <- toRelational(path)

sql.code <- getInsertSQL(db, "address")
```

---

savetofiles *Saving the relational data*

Description

Saves a list of dataframes created from an XML source with toRelational() to CSV files, one file per dataframe (i.e. table in the relational data model). File names are identical to the dataframe/table names.

Usage

```
savetofiles(ldf, dir, sep = ",", dec = ".")
```

Arguments

ldf	A **l**ist of **d**ata**f**rames created by toRelational() (the data tables transformed from XML to a relational schema).#' @param dir Directory where the files will be stored. Default is the current working directory.
dir	The directory to save the CSV files in. Per default the working directory.
sep	Character symbol to separate fields in the CSV fil, comma by default.
dec	Decimal separator used for numeric fields in the CSV file, point by default.

Value

No return vaue.

See Also

Other xml2relational: `getCreateSQL()`, `getInsertSQL()`, `toRelational()`

Examples

```
# Find path to custmers.xml example file in package directory
path <- system.file("", "customers.xml", package = "xml2relational")
db <- toRelational(path)

savetofiles(db, dir = tempdir())
```

`toRelational` *Converting an XML document into a relational data model*

Description

Imports an XML document and converts it into a set of dataframes each of which represents one table in the data model.

Usage

```
toRelational(
  file,
  prefix.primary = "ID_",
  prefix.foreign = "FKID_",
  keys.unique = TRUE,
  keys.dim = 6
)
```

Arguments

`file`	The XML document to be processed.
`prefix.primary`	A prefix for the tables' primary keys (unique numeric identifier for a data record/row in the table) . Default is `"ID_"`. The primary key field name will consist of the prefix and the table name.
`prefix.foreign`	A prefix for the tables' foreign keys (). Default is `"FKID_"`. The rest of the foreign key field name will consist of the prefix and the table name.
`keys.unique`	Defines if the primary keys must be unique across all tables of the data model or only within the table of which it is the primary key. Default is `TRUE` (unique across all tables).

keys.dim — Size of the 'key space' reserved for primary keys. Argument is a power of ten. Default is 6 which means the namespace for primary keys extends from 1 to 1 million.

Details

toRelational() converts the hierarchical XML structure into a flat tabular structure with one dataframe for each table in the data model. toRelational() determines automatically which XML elements need to be stored in a separate table. The relationship between the nested objects in the XML data is recreated in the dataframes with combinations of foreign and primary keys. The foreign keys refer to the primary keys that toRelational() creates automatically when adding XML elements to a table.

Column	Type	Description
Style	character	Name of the SQL flavor.
NormalField	character	Template string for a normal, nullable field.
NormalFieldNotNull	character	Template string for non-nullable field.
PrimaryKey	character	Template string for the definition of a primary key.
ForeignKey	character	Template string for the definition of a foreign key.
PrimaryKeyDefSeparate	logical	Indicates if primary key needs additional definition like a any other field.
ForeignKeyDefSeparate	logical	Indicates if foreign key needs additional definition like a any other field.
Int		character
Int.MaxSize	numeric	Size limit of integer data type.
BigInt	character	Name of data type for integers larger than the size limit of the normal integer data t
Decimal	character	Name of data type for floating point numbers.
VarChar	character	Name of data type for variable-size character fields.
VarChar.MaxSize	numeric	Size limit of variable-size character data type.
Text	character	Name of data type for string data larger than the size limit of the variable-size char
Date	character	Name of data type date data.
Time	character	Name of data type time data
Date	character	Name of data type for combined date and time data.

In the template strings you can use the following placeholders, as you also see from the MySQL example in the table:

1. %FIELDNAME%: Name of the field to be defined.
2. %DATATYPE%: Datatype of the field to be defined.
3. %REFTABLE%: Table referenced by a foreign key.
4. %REFPRIMARYKEY%: Name of the primary key field of the table referenced by a foreign key.

When you use your own defintion of an SQL flavor, then sql.style must be a one-row dataframe providing the fields described in the table above.

You can use the datatype.func argument to provide your own function to determine how the data type of a field is derived from the values in that field. In this case, the values of the columns Int, Int.MaxSize, VarChar, VarChar.MaxSize, Decimal and Text in the sql.style dataframe are ignored. They are used by the built-in mechanism to determine data types. Providing your own function allows you to determine data types in a more differentiated way, if you like. The function

that is provided needs to take a vectors of values as its argument and needs to provide the SQL data type of these values as a one-element character vector.

Value

A list of standard R dataframes, one for each table of the data model. The tables are named for the elements in the XML document.

See Also

Other xml2relational: `getCreateSQL()`, `getInsertSQL()`, `savetofiles()`

Examples

```
# Find path to custmers.xml example file in package directory
path <- system.file("", "customers.xml", package = "xml2relational")
db <- toRelational(path)
```

`xml2relational` *Package 'xml2relational'*

Description

Transforming a hierarchical XML document into a relational data model.

What is `xml2relational`

The `xml2relational` package is designed to 'flatten' XML documents with nested objects into relational dataframes. `xml2relational` takes an XML file as input and converts it into a set of dataframes (tables). The tables are linked among each other with foreign keys and can be exported as CSV or ready-to-use SQL code (`CREATE TABLE` for the data model, `INSERT INTO` for the data).

How to use `xml2relational`

- First, use `toRelational()` to read in an XML file and to convert into a relational data model.
- This will give you a list of dataframes, one for each table in the relational data model. Tables are linked by foreign keys. You can specify the naming convention for the tables' primary and foreign keys as arguments in `toRelational()`.
- You can now export the data structures of the tables (or a selection of tables) using `getCreateSQL()`. It support multiple SQL dialects, and you also provide syntax and data type information for additional SQL dialects.
- You can also export the data as SQL `INSERT` statements with the `getInsertSQL()`. If you only want to export the data as CSV use `savetofiles()` to save the dataframes produced by `toRelational()` as comma-separated files.

Index