R + SQLite

Pierre Tocquin

14/02/2019

```
knitr::opts_chunk$set(echo = TRUE)
# install.packages("RSQLite")
library(RSQLite)
library(DBI) # RSQLite est une sorte de `wrapper` SQLite de la librairie plus générique
             # `DBI` .Pensez à consulter aussi l'aide de ce package car certaines
             # fonctions ne sont pas re-décrites dans l'aide de RSQLite
```

Connection à la base de données

```
# travaillons sur une copie de notre base
file.copy(from = "foremgiga.sqlite", to = "test.sqlite", overwrite = T)
```

```
## [1] TRUE
```

```
system(command = "cp foremgiga.sqlite test.sqlite")

# créons la connexion vers le fichier SQLite de base de données
db.con <- dbConnect(drv = SQLite(), dbname="test.sqlite")
```

Requêtes

Le package `RSQLite` (qui étend le packages `DBI`) dispose d'une série de fonctions qui implémentent et étendent les commandes `sqlite3` telles que `.tables`.

```
tables <- dbListTables(conn = db.con)
fields <- lapply(tables, function(x) dbListFields(db.con, x))
names(fields) <- tables
fields$auteurs
```

```
## [1] "ID_aut" "nom"
```

L'interrogation de la base de données par des déclarations de type `select` est obtenue à l'aide de la commande `dbGetQuery`.

```
# statement <- paste("SELECT nom FROM", tables[1], "where ID_aut = 1") # paste() vs sprintf()
statement <- sprintf("SELECT nom FROM %s where ID_aut = 1", tables[1])
dbGetQuery(db.con, statement) # retourne une `data.frame`
```

```
##             nom
## 1 Alexandrov NN
```

Manipulation des données

Préliminaires

Par précaution, créons d'abord une copie de la table `auteurs` que nous souhaitons manipuler.

```r
dbCreateTable(db.con, "tmp_auteurs", c(nom="text", prenom="text"))
dbExecute(db.con, "INSERT INTO tmp_auteurs (nom) SELECT nom FROM auteurs;")
```

```
## [1] 71
```

```r
# Notez l'utilisation de `dbExecute()` plutôt que `dbGetQuery()` puisqu'on ne souhaite
# pas récupérer un résultat sous forme d'une data.frame.
# La valeur de retour de `dbExecute()` est le nombre de lignes affectées par la requête.
```

Affichons le résultat:

```r
head(dbGetQuery(db.con, "SELECT * FROM tmp_auteurs"))
```

```
##             nom prenom
## 1 Alexandrov NN   <NA>
## 2      Banks JA   <NA>
## 3    Barnaby NG   <NA>
## 4  Batchelor AK   <NA>
## 5   Beilinson V   <NA>
## 6  Bennetzen JL   <NA>
```

Vous noterez que la création de la colonne vide `prenom` conduit au remplissage de celle-ci par des valeurs nulles annotées `<NA>` dans R.

```r
head(dbGetQuery(db.con, "SELECT * FROM tmp_auteurs WHERE prenom is not null"))
```

```
## [1] nom    prenom
## <0 rows> (or 0-length row.names)
```

Le package `RSQLite` fournit des outils qui permettent de manipuler les tables sous forme de `data.frame`. Ainsi, la copie de la table `auteurs` vers la nouvelle table `tmp_auteurs` aurait pu être obtenue avec les commandes suivantes:

```r
auteurs <- dbReadTable(db.con, "auteurs")
auteurs$prenom <- NA
dbWriteTable(db.con, "tmp1_auteurs", auteurs, overwrite = TRUE)
head(dbGetQuery(db.con, "SELECT * FROM tmp1_auteurs"))
```

```
##   ID_aut           nom prenom
## 1      1 Alexandrov NN     NA
## 2      2      Banks JA     NA
## 3      3    Barnaby NG     NA
## 4      4  Batchelor AK     NA
## 5      5   Beilinson V     NA
## 6      6  Bennetzen JL     NA
```

Vous noterez néanmoins que cette opération n'est pas conseillée car elle 'écrase' la définition du schéma de votre table. Vous pouvez le vérifier à l'aide de la commande `.schema tmp1_auteurs` de `sqlite3` ou en interrogeant une table créée automatiquement par SQLite, `sqlite_master`, pour stocker le schéma de votre base de données.

```r
dbGetQuery(db.con, "SELECT * FROM sqlite_master WHERE tbl_name = 'tmp1_auteurs'")
```

```
##    type         name     tbl_name rootpage
## 1 table tmp1_auteurs tmp1_auteurs      106
##                                                                                  sql
## 1 CREATE TABLE `tmp1_auteurs` (\n  `ID_aut` INTEGER,\n  `nom` TEXT,\n  `prenom` INTEGER\n)
```

INSERT, UPDATE et DELETE

La réalisation des opérations de type `INSERT`, `UPDATE` ou `DELETE` implique l'utilisation de la fonction `dbExecute`. Voici un exemple avec la déclaration `INSERT`:

```
dbExecute(db.con, "INSERT INTO tmp_auteurs (nom, prenom) values ('Dupont', 'T.')")
```

```
## [1] 1
```

```
dbGetQuery(db.con, "SELECT * from tmp_auteurs WHERE nom like 'Dup%'")
```

```
##      nom prenom
## 1 Dupont     T.
```

`DELETE` et `UPDATE` s'utilisent tout aussi simplement:

```
dbExecute(db.con, "UPDATE tmp_auteurs SET nom = 'Dupond D.' where nom = 'Dupont D.'")
```

```
## [1] 0
```

```
dbExecute(db.con, "DELETE FROM tmp_auteurs where nom = 'Dupont T.'")
```

```
## [1] 0
```

```
dbGetQuery(db.con, "SELECT * from tmp_auteurs WHERE nom like 'Dup%'")
```

```
##      nom prenom
## 1 Dupont     T.
```

A vous de jouer...

1. **Utilisez les possibilités que vous offre R pour modifier les données de la table tmp_auteurs. Déplacer les initiales des prénoms des auteurs dans la colonne `prenom`. Le schéma de cette table ne doit pas être modifié après cette opération.**

Une fois cette opération effectuée, vous pouvez remplacer la table `auteurs` par cette nouvelles table `tmp_auteurs`. Pour ce faire, il suffit de de renommer la table `auteurs` en `auteurs_old` par exemple grâce à la déclaration SQLite suivante:

```
dbExecute(db.con, "ALTER TABLE auteurs RENAME TO auteurs_old")
```

```
## [1] 0
```

Puis faire de même avec la table `tmp_auteurs`:

```
dbExecute(db.con, "ALTER TABLE tmp_auteurs RENAME TO auteurs")
```

```
## [1] 0
```

Lorsque vous avez vérifié que tout est en ordre, vous pouvez - si vous le désirez - supprimer l'ancienne table `auteurs`:

```
dbExecute(db.con, "DROP TABLE auteurs_old")
```

```
## [1] 0
```

2. **Réalisez un script qui vous permet de remplacer toutes les chaines de caractères 'NULL' des tables de la base de données par des valeurs NULL**. Commencez par évaluer quels sont les tables et champs concernés. Dans le cas où c'est la clé primaire (ou un champs ne pouvant être nul) qui contient la chaine 'NULL', il faudra supprimer l'enregistrement plutôt que de le mettre à jour !

3. **Téléchargez le fichier ftp://ftp.ncbi.nlm.nih.gov/pub/taxonomy/taxdump.tar.gz qui contient l'ensemble des données taxonomiques du NCBI. Le fichier tabulaire `names.dmp` est celui qui nous intéresse. Il est formé de 4 colonnes: la première contient les numéros taxonomiques, la seconde le nom officiel de l'espèce et la dernière le type d'information. Commencez par selectionner les informations de type 'scientific name' puis mettez à jour le champs 'nom' de notre table organismes en utilisant ces informations, seulement si le nom du fichier 'names.dmp' est différent de celui de notre table. Combien d'enregistrements sont-ils modifiés ? Compléter ensuite votre base de données en recherchant et ajoutant les noms des taxons Parents qui ne seraient pas encodés dans la table.** Pour ce dernier exercice, je vous encourage à découvrir, installer et utiliser le package `data.table` qui facilite la lecture et le filtrage des tables *lourdes*.