R & SQL

Datastorm - B. Thieurmel

Si nécessaire, installer préalablement les packages suivants :

```
install.packages(c("DBI", "RMySQL", "RPostgres", "RSQLite",
                   "config", "glue", "nycflights13", "dplyr", "dbplyr"))
```

Premiers pas avec l'interface DBI

1. Ouvrir une connexion avec `dbConnect`

```
require(DBI)
require(RPostgres) # require(RMySQL)

# ouverture d'une connexion
con <- dbConnect(RPostgres::Postgres(), # RMySQL::MySQL()
                 host = "*****",
                 port = 123567,
                 dbname = "******",
                 user = "******",
                 password = "****")

con
```

2. Lister les tables présentes dans la base de données (`dbListTables`)
3. Lister les champs disponibles dans la table **preveol_ref** (`dbListFields`)
4. Récupérer l'ensemble de la table **preveol_ref** avec `dbReadTable`
5. Récupérer l'ensemble de la table **preveol_ref** avec une requête SQL (`SELECT * FROM preveol_ref`) et `dbGetQuery`
6. Combien de lignes de prédictions pour chaque identifiant dans la table **preveol_point** ? (`SELECT COUNT(*)  FROM preveol_point GROUP BY "ID"`)

Sécuriser les informations de connexion

Nous allons utiliser le package **config** pour sécuriser nos informations de connexion. Suivez la vignette du package pour vous connecter à la base de données sans que les paramètres de connexion apparaissent en clair dans votre code **R**

https://cran.r-project.org/web/packages/config/vignettes/introduction.html

1. Créer le fichier *config.yml*, avec *à minima* l'environnement `default`
2. Ajouter les champs des informations de connexion
3. Vérifier la bonne lecture de ce fichier (`conf <- config::get()`)
4. Connecter vous à la base en utilisant la variable `conf` ci-dessus

Attention à l'indentation du fichier YAML (tabulation), qui doit également contenir une ligne vide à la fin sous peine de warning. . . .

Aller plus loin : définir plusieurs environnements et faire quelques tests

Requêtes paramétrables

En amont, nous allons initier localement une base de données `SQLite` :

```
require(DBI)
require(RSQLite)
library(nycflights13)

head(nycflights13::flights)
head(nycflights13::airports)
head(nycflights13::airlines)

# initialisation d'une base sqlite sur disque
con <- DBI::dbConnect(RSQLite::SQLite(),
                      dbname = "tp_r_sql_flights.sqlite")

if(dbExistsTable(con, "flights")) dbRemoveTable(con, "flights")
if(dbExistsTable(con, "airports")) dbRemoveTable(con, "airports")
if(dbExistsTable(con, "airlines")) dbRemoveTable(con, "airlines")

# ecriture de l'historique des vols
copy_to(con,
        nycflights13::flights,
        "flights",
        temporary = FALSE,
        indexes = list(
          c("year", "month", "day"),
          "carrier",
          "tailnum",
          "dest"
        )
)
# ecriture des référentiel
copy_to(con,
        nycflights13::airports,
        "airports",
        temporary = FALSE,
        indexes = list("faa")
)

copy_to(con,
        nycflights13::airlines,
        "airlines",
        temporary = FALSE,
        indexes = list("carrier")
)

dbListTables(con)

# if(dbExistsTable(con, "flights")) dbRemoveTable(con, "flights")
# if(dbExistsTable(con, "airports")) dbRemoveTable(con, "airports")
# if(dbExistsTable(con, "airlines")) dbRemoveTable(con, "airline")
```

1. Que fait le code suivant ?

```
ex_bind <-  dbSendQuery(con, "SELECT * FROM flights WHERE year = ? AND month = ?")
data_01_2013 <- dbFetch(dbBind(ex_bind, list(2013, 1)))
data_04_2013 <- dbFetch(dbBind(ex_bind, list(2013, 4)))
```

2. Utiliser le package **glue** et la fonction **glue_sql()** pour faire la même requête

N.b : Syntaxe `{nom_var}` en remplacement des `?`

3. Modifier votre requête **glue** pour pouvoir sélectionner plusieurs mois

N.b : côté **SQ**L, opérateur `IN` avec des parenthèse, côté R syntaxe `{nom_var*}` en remplacement des `?`

Utilisation de dbplyr

Rappels dplyr :

- `select()` : sélectionner des colonnes d'un tableau de données
- `filter()`: filtrer les lignes à conserver
- `arrange()` : permet d'ordonner les données
- `mutate()` : créer une ou plusieurs nouvelles colonnes
- `group_by()` : définir les sous-population
- `summarize()` : calculer des indicateurs sur nos sous-populations et récupérer directement le tableau agrégé
- Utilisation de la fonction `n()` pour compter le nombre de lignes

Rappels dbplyr :

1. Ouverture de la connexion à la base de données avec `con <- dbConnect(...)`
2. Branchement à une table de la base de données avec `tbl(con, "nom_table")`
3. Utilisation *classique* de la syntaxe **dplyr**.
   - **dplyr** va se charger de créer la requête **SQL** correspondante
   - Il est possible de récupérer cette requête avec `show_query()`
   - la requête ne sera exécutée que quand le code **R** suivant aura effectivement besoin du résultat
4. Récupération du résultat dans **R** avec `collect()`

Documentation

https://dbplyr.tidyverse.org/

https://rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf

1. Se brancher à la table **flights** (`tbl`)
2. Sélectionner les lignes qui ont pour `origin` l'aéroport **JFK**
3. Sélectionner les lignes qui ont pour `origin` l'aéroport **JFK** et comme date de vol le mois (`month`) de juillet.
4. Compléter la requête précédente pour ordonner le résultat par `day` décroissant, et enlever les colonnes `origin`, `year` et `month` du résultat
5. Calculer le nombre de vols qui démarrent de **JFK** par mois
6. Calculer le nombre de vols qui démarrent de **JFK** par mois et par jour
7. Calculer le nombre moyen de vol par jour (sur tous les mois)

Aide

- le résultat de la question **6** nous renvoie un object **dplyr** après un `collect()`
- Nous pouvons donc enchaîner avec le `%>%` sur d'autres calculs après le `collect()`

- Si un `group_by` a été utilisé en amont, **dplyr** conserve cette propriété. Nous pouvons faire un `ungroup()` pour l'enlever
- La fonction `summarise` peut être utilisée sur l'ensemble des données (c-à-d sans sous-population...)

8. Calculer la moyenne des retards au départ (`dep_delay`) et à l'arrivée (`arr_delay`) par compagnie (`carrier`).
9. Enrichir la requête précédente (après le `summarise`) en créant une nouvelle colonne `mean_delay`, moyenne des deux colonnes calculées en amont. (`mutate`) et trier le résultat . Pire compagnie ?
10. Regarder la requête **SQL** générée dans la question *9* (`show_query`)
11. Calculer la vitesse moyenne et le nombre de vol par itinéraire

Aide

- itinéraire = concaténation de `origin` et `dest`
- `speed` égale à `distance` /(`air_time`/60)

12. Calculer le nombre de vols au départ de chaque aéroport (`origin`) et rajouter les informations du référentiel des aéroports au résultat

Aide

- Créer un nouveau `tbl` pour la table `airports`
- Identifier la clé commune
- Utiliser un `left_join()` une fois le nombre de vols calculé. Clé(s) avec des noms différents ? :

```
data_x %>% left_join(data_y, by = c("cle_x" = "cle_y"))
```

13. Rajouter le nom de la compagnie au résultat de la question `9`

PENSER MAINTENANT A BIEN FERMER LA/LES CONNEXION(S) OUVERTE(s)!!!