MUSÉUM NATIONAL D'HISTOIRE NATURELLE

SEMIN-R

Utiliser SQL pour manipuler des données sous R

BAUDOIN, Raymond

Inventaire et suivi de la biodiversité - CBNBP
Département EGB
baudoin@mnhn.fr

Utiliser SQL pour manipuler des données sous 

➢ Pour un utilisateur de R

Quand on veut filtrer les données à traiter car :

- Elles sont trop nombreuses
- Qu'une partie seulement nous intéresse

Elles sont éclatées dans plusieurs entités (fichiers)

*C'est-à-dire chaque fois qu'un read.table() n'est pas totalement satisfaisant **

(*) permet cependant la restriction du nombre de lignes lues

➢ Pour un utilisateur SQL :

Quand on souhaite y appliquer des fonctions de R

Librairies utilisées :

- *pour la connexion aux bases de données : RODBC, SQLiteDF, RMySQL, RPostgreSQL, ROracle, DBI*
- *pour accéder aux données des :*
  - *fichiers .csv : sqldf*
  - *fichiers Excel : RODBC, xlsReadWrite, xlsx*
  - *bases de données : RODBC, SQLiteDF, RMySQL, RPostgreSQL, ROracle, DBI*
- *pour manipuler des data frames : sqldf*

Rappel des fonctions de lecture des données

Pour enregistrer un data.frame au format MS-Excel :
write.xls(datal, file="localisation.xls", rowNames = TRUE)

➢ Fichier texte

Choisir le fichier : fich <- file.choose()
Utiliser le presse papier (copier / coller) : fich <- "clipboard "

read.table (file = fich , sep="\t", dec=",", header=FALSE, skip=2, nrow=3)

Séparateur des valeurs
\t pour tabulation
\n pas de séparateur

Symbole décimal

Restriction sur les lignes lues

scan (file = fich , sep="\n", what = "character", skip=2, nlines=3)

➢ Fichier MS-Excel

- **Pour des fichiers *.xls : library (xlsReadWrite)**

```
read.xls( fich,
    colNames = TRUE,
    sheet = 1,
    colClasses = NA, # numeric, integer, logical, character, isodate, isotime, isodatetime…
    stringsAsFactors = TRUE… )
```

- **Pour des fichiers Excel97 : library (xlsx)**

```
read.xlsx(fich, sheetIndex, header=TRUE, colClasses=NA…)
```

Le langage SQL (Structured Query Language)

Langage unique pour contrôler, décrire, l'accès et interroger les bases de données relationnelles

Caractéristiques :

- Inventé en 1970
- Régi par une norme ANSI/ISO
  - Première normalisation ISO en 1987
  - SQL 2 correspond à la norme SQL/92
  - SQL 3 en 1999, actuellement SQL 2003
- Portable sur différentes plates-formes aussi bien matérielles que logicielles.
  Une commande SQL écrite dans un environnement Windows sous MS-ACCESS est utilisable directement dans un environnement ORACLE sous Unix.

Regroupe :

- un langage de contrôle des accès (DCL, Data Control Language)
- un langage de définition de données (DDL Data Definition Language)
  Pour créer, modifier ou supprimer des tables dans la base
- **un langage de manipulation de données** (DML, Data Manipulation Language)
  Pour **sélectionner**, insérer, modifier ou supprimer des données dans une table

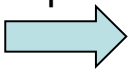
uniquement présenté ici

Elément de base : la table

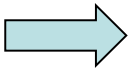
- **Les données sont réparties en tables** ou entités
- Une table est composée de **lignes**
- Une ligne est un ensemble fixe de **champs** (attributs)

Exemple :
la table Personnes

Chaque champ a :

un nom 

id	Nom	Prénom	numéro	id_localisation	Téléphone	Depuis
1	Lecesve	André	9	2	146542856	19/10/1920
2	Larrouy	Catherine	10	2	140920841	01/01/1999
3	Larrouy	Eric	10	2	140920841	01/01/1999
6	Meyer	Michel	15	3		05/06/1960
8	Auquier	Anne	21	3	157750048	05/09/2005
9	Auquier	Anne	21	3	636699001	05/09/2005
10	Auquier	Bernard	21	3	146428564	05/09/2005

- ***un type*** 

INTEGER CHARACTER INTEGER DECIMAL DATE

Propriétés d'une table :

- pas de nom de colonne dupliqué
- ordre des lignes sans signification
- ordre des colonnes (champs) sans signification
- chaque champ permet la sélection de lignes

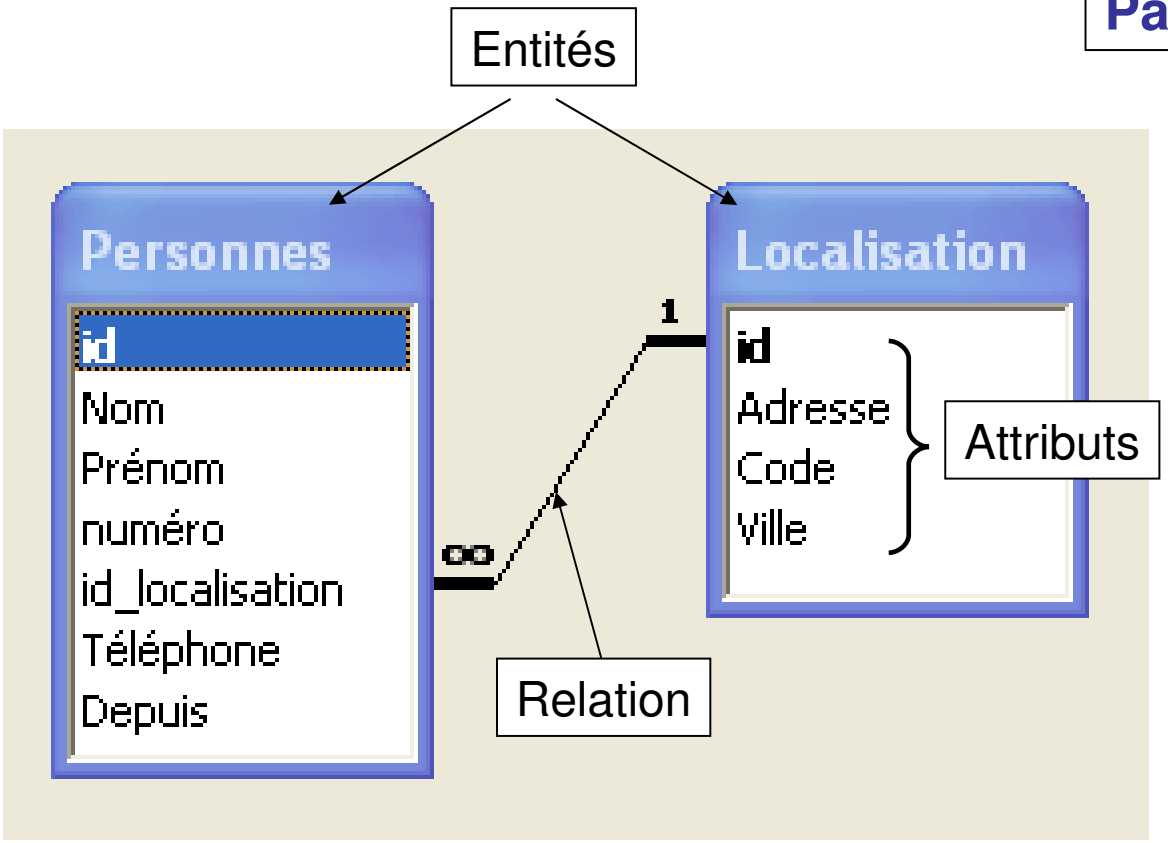
Avec ces propriétés, une table :

≈ un fichier

≈ une feuille xls

Le modèle relationnel

Objectif du modèle :
Pas d'information redondante



La **clef primaire** d'une table est un attribut ou un groupe d'attributs de cette table dont la valeur permet d'identifier de manière unique une ligne de la table.

Permet la création de liaisons fixes entre les tables en mettant en relation un ou plusieurs champs.

Cardinalité : dénombre le nombre d'éléments de l'entité départ en relation avec un élément de l'entité arrivée.

Relations possibles :
1/1, 1/n, n/1, n/m
Une relation n/m nécessite une table de jointure

SELECT
la commande d'interrogation des données

Syntaxe :

```
SELECT [ALL | DISTINCT] { * | col | expr [AS alias], ... }
FROM table [alias], ...
[ WHERE { conditions de recherche | sous conditions} ]
[ GROUP BY col, ...] [HAVING conditions de recherche ]
[ ORDER BY { col | num } {ASC | DESC}, ...] ;
```

Légende :
{ } : Une des valeurs séparées par '|' obligatoire.
[ ] : Valeur optionnelle.
... : Répétition.
__ : Valeur par défaut.

SELECT	Précise les colonnes qui vont apparaître dans la réponse
FROM	Précise la (ou les) table intervenant dans l'interrogation
WHERE	Précise les conditions à appliquer sur les lignes avec : - Des opérateurs de comparaison : =, >, <, >=, <=,<> - Des opérateurs logiques : AND, OR, NOT - Des prédicats : IN, LIKE, NULL, ALL, ANY...
GROUP BY	Précise la (ou les) colonne de regroupement
HAVING	Précise la (ou les) conditions associées à un regroupement
ORDER BY	Précise l'ordre dans lequel vont apparaître les lignes de la réponse : - ASC : ordre ascendant (par défaut) - DESC : ordre descendant

Exemple, sélection de toute la table Habitants : SELECT * FROM habitants;

Nécessite un outil logiciel de gestion cohérente des données : Système de Gestion de Base de Données

SGBD ou DBMS (Database management system)

Un ensemble de services pour :

- permettre le contrôle de l'accès aux données de façon simple
- autoriser un accès aux informations à de multiples utilisateurs
- manipuler les données présentes dans la base (insertion, suppression, modification)

Se décompose en trois sous-systèmes :

- un système de gestion de fichiers pour le stockage des informations sur le support physique
- un système pour gérer l'ordonnancement des informations
- une interface avec l'utilisateur

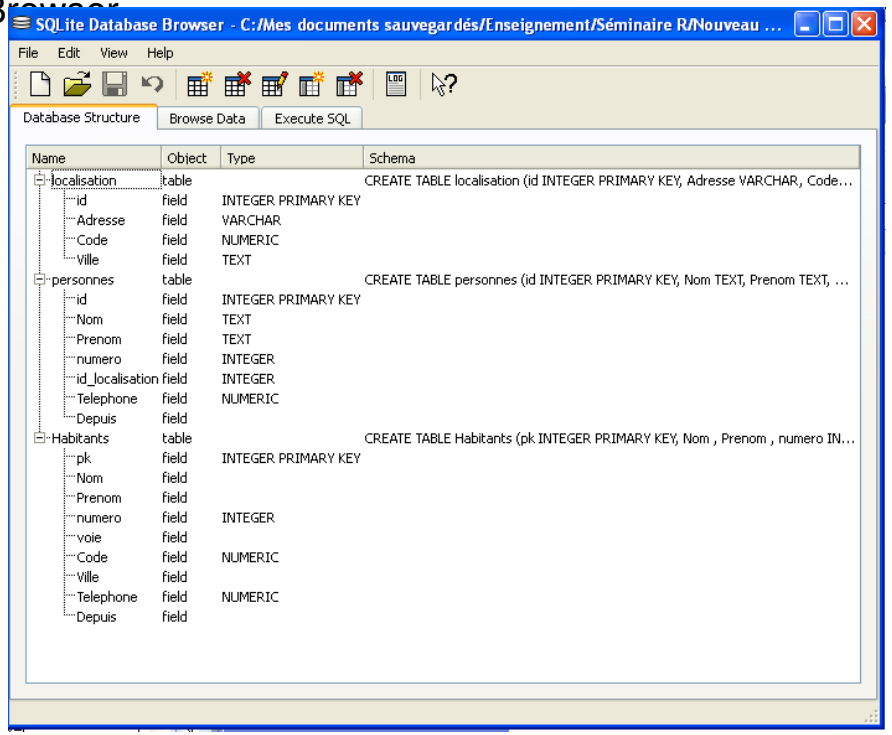
SQLite : SGBDR portable utilisable sous

- moteur de base de données relationnelles accessible par le langage SQL.
- implémente en grande partie le standard SQL-92
- la base de données (déclarations, tables, index et données) stockée dans un fichier
- utilisation sans restriction dans des projets aussi bien open source que propriétaires
- multi-plateforme

SQLite est le moteur de base de données le plus distribué au monde, grâce à son utilisation dans de nombreux logiciels grand public comme Firefox, Skype, Google Gears, dans certains produits d'Apple, d'Adobe et de McAfee et dans les librairies standards de nombreux langages comme PHP ou Python. De par son extrême légèreté (moins de 300 ko), elle est également très populaire sur les systèmes embarqués, notamment sur la plupart des smartphones modernes.

http://www.sqlite.org/

Une « interface » utilisateur : SQLite Database Browser



http://sqlitebrowser.sourceforge.net/

SQLite avec R : library (sqldf)

Permet :

- Connexion à une base de données **SQLite**
- L'utilisation des requêtes SQL sur des data frames
- La lecture de fichier texte filtrée par une requête

Utilise les librairies :

- DBI (data base interface)
- RSQLite : pilote (driver) R pour l'accès à une base SQLite

sqldf () :
connexion à une base de données **sqlite**

- Pour ouvrir une connexion à la base sqlite "nom_de_la_base" *

```
sqldf (dbname = "nom_de_la_base")
```

** exclusivement le nom de la base et non le chemin complet*

- Si dbname non précisé ou base non trouvée : ouverture d'une base virtuelle

```
( sqldf() ) # ouverture d'une connexion sur une base virtuelle
<SQLiteConnection:(1036,0)>
( sqldf() ) # fermeture de la connexion au 2ème appel
NULL
```

```
con <- sqldf ()
dbListTables(con)
```

Utilisation des commandes DBI :
récupération de l'identifiant de la connexion
liste les tables de la base "SQLite" connectée

(*)

Exemple d'un script de sélection de la base à connecter

```
setwd(dirname(dbase <- file.choose()))                          # change le répertoire
dbase = iconv(dbase,Encoding(dbase),"latin1")                   # conversion code caractère
dbase <- substring(dbase,max(unlist(gregexpr("\\",dbase,fixed=TRUE)))-1)
sqldf(dbname = dbase)                                           # connexion
```

SELECT SQL via sqldf

S'assurer qu'une connexion est bien ouverte

1 - Utilisation avec base sqlite

```
sqldf ("SELECT * FROM ma_table ;", method = "raw" )
```

Requête SQL

"raw" Données brutes
"auto" Types ajustés au "mieux"

2 - Utilisation avec un data.frame

Exemple : iris

```
sqldf ("SELECT * FROM iris ;", method = "raw", row.names = FALSE)
```

Requête SQL

"raw" Données brutes
"auto" Types ajustés

TRUE : ajoute une colonne row_names à la table si method= "raw" et SELECT * : ajoute une colonne row_names au data.frame

Attention – Dans les cas 2, 3 et 4

- si la connexion est sur une base nommée, il y aura copie de la (des) table(s) du FROM. Risque de conflit !
- si la connexion ne pointe aucune base nommée, c'est la base virtuelle qui est utilisée.

SELECT SQL sur un fichier texte

3 – Avec la syntaxe sqldf

```
fich <- file(file.choose())                    # ouvre une connexion sur un fichier texte au choix
attr(fich , "file.format") <- list(sep = "\t" , header = TRUE)      # affectation des attributs
```

Séparateur des champs

Si nom des colonnes en 1ère ligne

```
sqldf ("SELECT * FROM fich;", method = "auto")
```

Requête SQL

- Si pas de nom de colonne : nomme des colonnes de V1 à Vncol
- Si header = TRUE :
  - si length(colnames) = ncol : nomme les colonnes avec les noms
  - si length(colnames) = ncol -1 ajoute une colonne row_names à la table
    - si method= "raw" et SELECT * ajoute une colonne row_names au data.frame

4 – Avec un read.csv filtré par SELECT SQL

```
read.csv2.sql (file.choose(), sql = "select * from file" )
```

«file» mot clé obligatoire

Remarque :
dbRemoveTable(con, "ma_table") # supprime la table "ma_table" de la base connectée

SELECT
la commande d'interrogation des données

Syntaxe :

```
SELECT [ALL | DISTINCT] { * | col | expr [AS alias], ... }
FROM table [alias], ...
[ WHERE { conditions de recherche | sous conditions} ]
[ GROUP BY col, ...] [HAVING conditions de recherche ]
[ ORDER BY { col | num } {ASC | DESC}, ...] ;
```

Légende :
{ } : Une des valeurs séparées par '|' obligatoire.
[ ] : Valeur optionnelle.
... : Répétition.
__ : Valeur par défaut.

SELECT	Précise les colonnes qui vont apparaître dans la réponse
FROM	Précise la (ou les) table intervenant dans l'interrogation
WHERE	Précise les conditions à appliquer sur les lignes avec :
	- Des opérateurs de comparaison : =, >, <, >=, <=,<>
	- Des opérateurs logiques : AND, OR, NOT
	- Des prédicats : IN, LIKE, NULL, ALL, ANY...
GROUP BY	Précise la (ou les) colonne de regroupement
HAVING	Précise la (ou les) conditions associées à un regroupement
ORDER BY	Précise l'ordre dans lequel vont apparaître les lignes de la réponse :
	- ASC : ordre ascendant (par défaut)
	- DESC : ordre descendant

SELECT SQL 1/5

Alias

renommer une colonne ou une table *colnames()*

```
sql <- "SELECT Adresse AS Voie, Code AS [Code postal], Ville AS Commune
        FROM Localisation ;"
```

```
            Voie Code postal   Commune
1  avenue Verdun       92170    VANVES
2 rue Gay Lussac       92320 CHATILLON
3    rue Roissis       92140   CLAMART
```

DISTINCT

Supprimer les lignes identiques *unique()*

```
sql <- "SELECT DISTINCT Nom FROM Personnes;"
```

COUNT ()

Compter des enregistrements

```
sql <- "SELECT count(*) FROM Personnes;"   # nombre d'enregistements de la table
```

ncol()

```
  count(*)
3    13
```

```
sql <- "SELECT count(Distinct Nom) FROM Personnes;"
```

```
  count(Distinct Nom)
1                   9
```

Exécution de la requête SQL : sqldf (sql)

SELECT SQL 2/5

Introduire des conditions de recherche

clause WHERE

sql <- "SELECT * FROM Localisation **WHERE** ville = 'VANVES' ;"

```
  id       Adresse  Code  Ville
1  1 avenue Verdun 92170 VANVES
```

sql <- "SELECT * FROM Localisation **WHERE** Code < 92200 ;"

```
  id       Adresse  Code   Ville
1  1 avenue Verdun 92170  VANVES
2  3   rue Roissis 92140 CLAMART
```

sql <- "SELECT * FROM Localisation **WHERE** Ville **LIKE '%N%'** ;"

```
  id         Adresse  Code     Ville
1  1   avenue Verdun 92170    VANVES
2  2 rue Gay Lussac  92320 CHATILLON
```

à l'intérieur d'une chaîne
% symbole de troncature
_ remplace 1 caractère

sql <- "SELECT * FROM Localisation **WHERE** Ville **IN** ('CHATILLON', 'CLAMART') ;"

```
  id        Adresse  Code     Ville
1  2 rue Gay Lussac 92320 CHATILLON
2  3    rue Roissis 92140   CLAMART
```

sql <- "SELECT * FROM Localisation **WHERE** Code **BETWEEN** 92140 **AND** 92200 ;"

```
  id       Adresse  Code   Ville
1  1 avenue Verdun 92170  VANVES
2  3   rue Roissis 92140 CLAMART
```

sql <- "SELECT * FROM Localisation **WHERE** Ville LIKE 'c%' **AND** Code > 92200 ;"

```
  id         Adresse  Code     Ville
1  2 rue Gay Lussac  92320 CHATILLON
```

opérateurs
logiques OR, AND,
NOT

Exécution de la requête SQL : sqldf (sql)

SELECT SQL 3/5

GROUP BY

Agrégation des lignes

Permet faire un dénombrement (count) ou le calcul d'une fonction d'agrégation pour les champs des lignes regroupées.

Critères d'agrégation : min, max, avg, sum ou une expression

```
sql <- "SELECT Ville, count(*) AS Nb_personnes FROM Habitants  GROUP BY Ville ;"
      Ville Nb_personnes
1 CHATILLON            5
2   CLAMART            5
3    VANVES            3
```

HAVING

Introduire une condition dans un GROUP BY

Exemple : Qui a plus d'un téléphone dans les villes ayant un CP > 92150 ?

```
sql <- "SELECT Nom, Prenom, Count(Telephone) AS NbTéléphone
  FROM Habitants WHERE Code > 92150
  GROUP BY Nom, Prenom
  HAVING Count (Telephone) >1 ;"

     Nom     Prenom NbTéléphone
1 Berrue Christiane           2
```

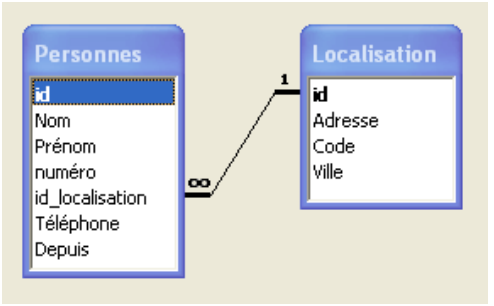
Exécution de la requête SQL : sqldf (sql)

SELECT SQL 4/5

Jointure entre tables

Jointure interne

1.1 - Equi-jointure : relier avec une relation d'égalité des tables par un champ commun



- Dans la clause WHERE

```
sql <- "SELECT Nom, Prenom, Ville FROM Localisation, Personnes
      WHERE Localisation.id = id_localisation AND Ville='VANVES';"
```

- Dans le FROM avec INNER JOIN ... ON

```
sql <- "SELECT Nom, Prenom, Ville FROM Localisation INNER JOIN Personnes
      ON Localisation.id = id_localisation
      WHERE Ville='VANVES';"
```

```
      Nom     Prenom  Ville
1 Mahier     Ludovic VANVES
2 Berrue Christiane VANVES
3 Berrue Christiane VANVES
```

1.2 - Auto-jointure : jointure d'une table avec elle-même avec l'utilisation d'un alias

Exemple : Qui partage le même téléphone ?

```
sql <- "SELECT a.Nom, a.Prenom, a.Telephone FROM Habitants a, Habitants b
      WHERE a.Telephone = b.Telephone AND a.pk <> b.pk;"
```

```
      Nom    Prenom Telephone
1 Larrouy Catherine 140920841
2 Larrouy      Eric 140920841
```

Exécution de la requête SQL : sqldf (sql)

SELECT imbriqués

Permet d'utiliser dans la clause WHERE d'un SELECT le résultat d'un autre SELECT

En fonction de la condition de la clause WHERE du SELECT principal, on utilisera pour la liaison entre le champ du SELECT principal et les valeurs renvoyées par la sous requête :

- un des opérateurs de comparaison : =, >, <, >=, <=,<>
- un des prédicats : IN, ANY, ALL ou EXISTS

```
sql <- "SELECT DISTINCT Nom, Prenom, numero, Ville FROM Localisation, Personnes
      WHERE Localisation.id = id_localisation
      AND Ville != 'CHATILLON'
      AND numero > (SELECT max(numero) FROM Habitants WHERE Ville = 'CHATILLON' );"
```

jointure

condition : non à Chatillon

Sous requête

```
  Nom  Prenom numero   Ville
1 Auquier    Anne     21 CLAMART
2 Auquier Bernard     21 CLAMART
3 Foucher Georges     17 CLAMART
```

➢ Dans quelle(s) rue(s) des habitants n'ont pas de téléphone ?

```
sql <- "SELECT Adresse, Ville FROM Localisation
      WHERE id IN (SELECT id_localisation FROM Personnes WHERE Telephone ='NA' );"
```

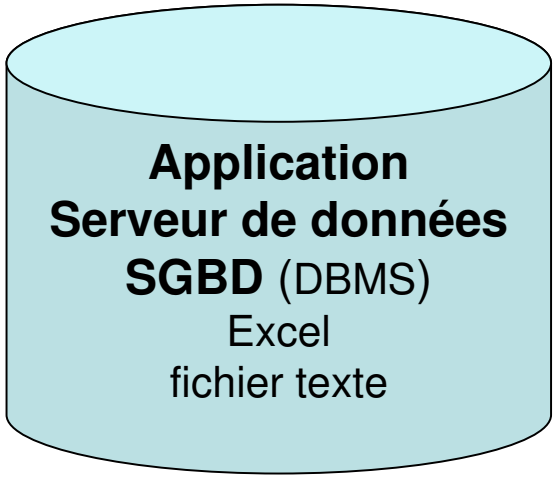
```
      Adresse   Ville
1 rue Roissis CLAMART
```

Exécution de la requête SQL : sqldf (sql)

Accéder à l'application gérant les données

DBI
database interface

ODBC
Open DataBase Connectivity



Interface spécialisée :
l'application à un pilote ODBC

Dans ce cas la connexion se fait directement sur le pilote ODBC de l'application

L'interface doit :

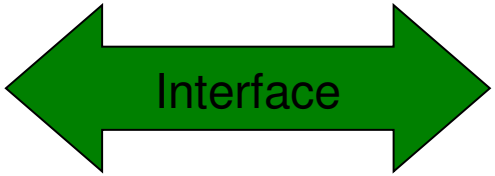
rôle des objets de

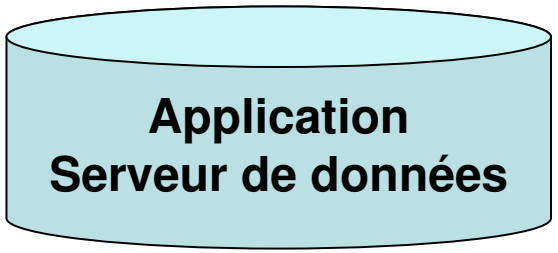
Identifier l'application : DBI Driver

Etablir la connexion : DBI Connexion

Renvoyer les données : DBI Résultats

Des packages R spécifiques* R ≥ 2.9





RMySQL	Package source: RMySQL_0.7-4.tar.gz MacOS X binary: RMySQL_0.7-4.tgz Windows binary: RMySQL_0.7-4.zip	MySQL
RPostgreSQL	Package source: RPostgreSQL_0.1-6.tar.gz MacOS X binary: Non disponible Windows binary: RPostgreSQL_0.1-6.zip	PostgreSQL
ROracle	Package source: ROracle_0.5-9.tar.gz MacOS X binary: Non disponible Windows binary: Non disponible	Oracle
RSQLite	Package source: RSQLite_0.8-2.tar.gz MacOS X binary: RSQLite_0.8-2.tgz Windows binary: RSQLite_0.8-2.zip	SQLite
RODBC	Package source: RODBC_1.3-1.tar.gz MacOS X binary: RODBC_1.3-1.tgz Windows binary: RODBC_1.3-1.zip	Bases de données ayant un pilote ODBC : Access, Excel, SQLite...

* *Contient à la fois le driver et l'interface (DBI)*

La démarche

Connexion **via DBI** | **via ODBC**

exemple : SGBD sqlite

1. **charger la librairie**
   - library (sqlite) | library (RODBC)
3. **charger le pilote**
   - drv <- dbDriver ("SQLite")
5. **ouvrir une connexion sur les données**
   - con <- dbConnect (drv, dbname=*...) | channel <- odbcDriverConnect()
   - * nom de la base uniquement | ou channel <- odbcConnectExcel(…)
   - channel <- odbcConnectAccess(…)

- **voir la structure des données** (non obligatoire)
  - dbListTables (con) | sqlTables (channel,…)
  - dbListFields (con, "sqtable") | sqlColumns (channel, sqtable,…)
  - *dbGetRowCount (res); dbColumnInfo (res)*

13. **charger les données avec tous les champs**
    - dbReadTable (con, sqtable) | sqlFetch (channel, sqtable,…)

- **charger les données avec un SELECT** (sql)
  - res <- dbSendQuery (con, sql) | sqlQuery (channel, sql,…)
  - fetch (res, n = -1)
  - dbGetQuery (channel, sql,…)

- **fermer la connexion** : OBLIGATOIRE pour libérer l'accès
  - dbDisconnect (con) | close(channel) ou odbcClose(channel)
  - odbcCloseAll()

- **libérer le pilote** : dbUnloadDriver (drv)

Connexion à une base de données via ODBC

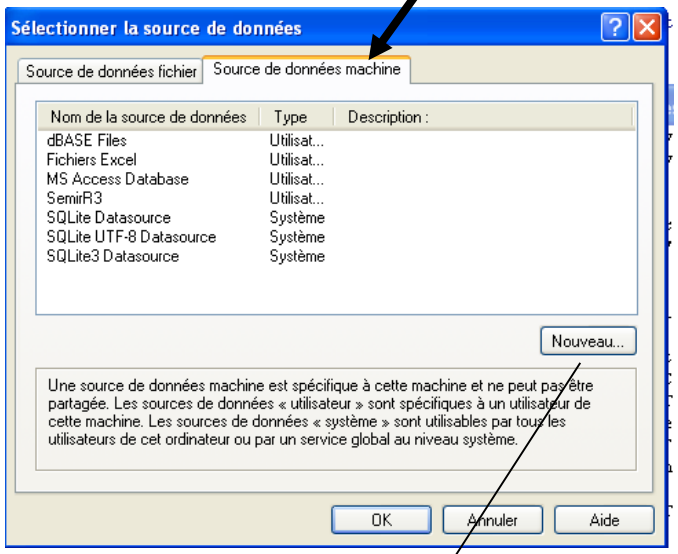
channel <- odbcDriverConnect ()

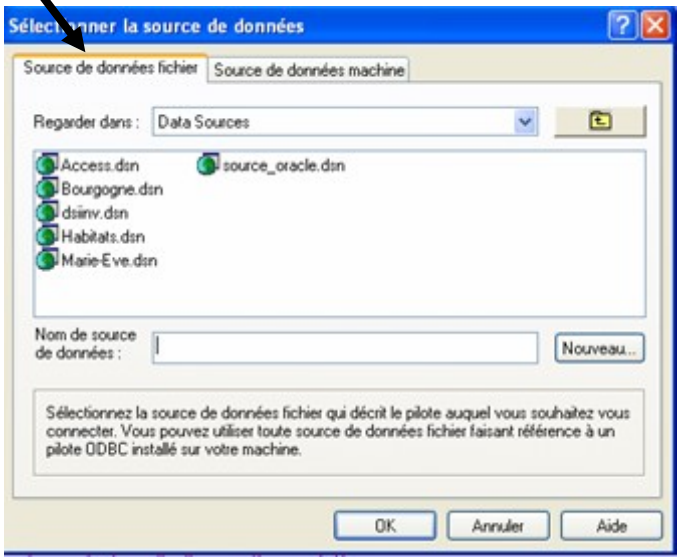
Configurer un lien ODBC sous Windows
- Démarrer
- Panneau de configuration
- Outils d'administration
- Sources de données (ODBC)

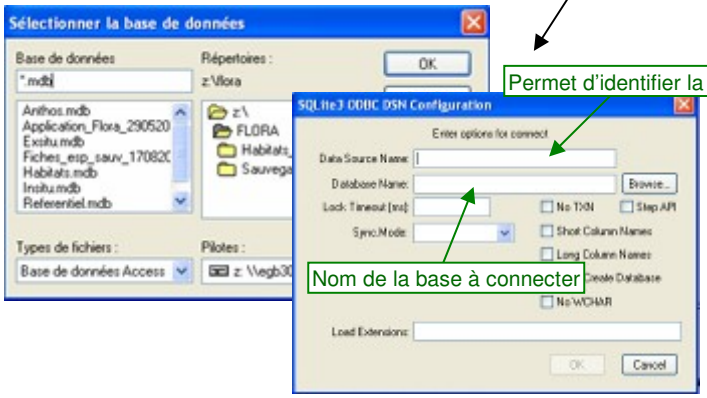
La base est dans un répertoire accessible

La base est sur un serveur distant

L'accès à la base est décrit dans le dsn associé dans C:\Program Files\Fichiers communs\ODBC\Data Sources

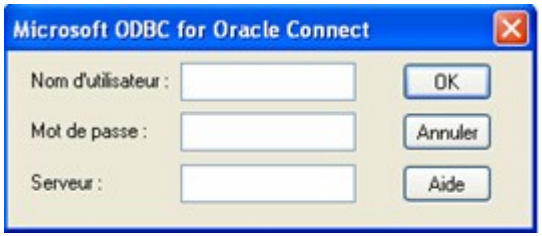






Permet d'identifier la source

Nom de la base à connecter



Contrôle de l'accès si activé

Exemples de connexion

à une base sqlite : "SeminR.db"

Via DBI

```
library (RSQLite)
# Se mettre dans le répertoire de la base
setwd(choose.dir())
db <- "SeminR.db"
drv <- dbDriver("SQLite")
con <- dbConnect(drv, dbname=db)

drv
<SQLiteDriver:(2192)>
con
<SQLiteConnection:(2192,0)>

dbListConnections(drv )
[[1]]
<SQLiteConnection:(2192,0)>
dbGetInfo(con)
$dbname
[1] "SeminR.db"
$serverVersion
[1] "3.6.21"
$rsId
integer(0)
$loadableExtensions
[1] "off"
$flags
[1] 6
$vfs
[1] ""
```

Via ODBC

```
library (RODBC)

c_sqlite <- odbcDriverConnect()

c_sqlite
RODBC Connection 10
Details:
  case=nochange
  DSN=SemirR3
  Database=C:\Mes documents
    sauvegardés\Enseignement\Séminaire
    R\Nouveau 2010\SeminR.db
  StepAPI=0
  SyncPragma=NORMAL
  NoTXN=0
  Timeout=
  ShortNames=0
  LongNames=1
  NoCreat=0
  NoWCHAR=0
  JournalMode=
  LoadExt=
```

à une base Access

```
library (RODBC)
(conn <- file.choose())
[1] "C:\\...\\Séminaire R\\RODBC.mdb"

channel <- odbcConnectAccess (conn)

channel
RODBC Connection 11
Details:
  case=nochange
  DBQ=C:\Mes documents
     sauvegardés\Enseignement\Séminair
     e R\RODBC.mdb
  Driver={Microsoft Access Driver (*.mdb)}
  DriverId=25
  FIL=MS Access
  MaxBufferSize=2048
  PageTimeout=5
  UID=admin
```

Accès aux structures

Tables

DBI

dbListTables(con)

```
[1] "Habitants"    "localisation" "personnes"
```

sqlTables (c_sqlite)

```
 TABLE_CAT TABLE_SCHEM   TABLE_NAME TABLE_TYPE REMARKS
1      <NA>        <NA>    Habitants      TABLE    <NA>
2      <NA>        <NA> localisation      TABLE    <NA>
3      <NA>        <NA>    personnes      TABLE    <NA>
```

ODBC

sqlTables (channel)

```
                    TABLE_CAT TABLE_SCHEM        TABLE_NAME   TABLE_TYPE REMARKS
1 C:\\...\\Séminaire R\\RODBC        <NA> MSysAccessObjects SYSTEM TABLE     <NA>
2 C:\\...\\Séminaire R\\RODBC        <NA>     MSysAccessXML SYSTEM TABLE     <NA>
3 C:\\...\\Séminaire R\\RODBC        <NA>          MSysACEs SYSTEM TABLE     <NA>
4 C:\\...\\Séminaire R\\RODBC        <NA>       MSysObjects SYSTEM TABLE     <NA>
5 C:\\...\\Séminaire R\\RODBC        <NA>       MSysQueries SYSTEM TABLE     <NA>
6 C:\\...\\Séminaire R\\RODBC        <NA> MSysRelationships SYSTEM TABLE     <NA>
7 C:\\...\\Séminaire R\\RODBC        <NA>         Habitants         TABLE     <NA>
8 C:\\...\\Séminaire R\\RODBC        <NA>      Localisation         TABLE     <NA>
9 C:\\...\\Séminaire R\\RODBC        <NA>         Personnes         TABLE     <NA>
```

Champs

sqtable = "localisation"

DBI

dbExistsTable (con, sqtable)

```
[1] TRUE
```

dbListFields(con, sqtable)

```
[1] "id"      "Adresse" "Code"    "Ville"
```

ODBC

sqlColumns (c_sqlite, sqtable)[,3:6]

```
  TABLE_NAME COLUMN_NAME DATA_TYPE TYPE_NAME
1 localisation          id         4   INTEGER
2 localisation     Adresse        -9   VARCHAR
3 localisation        Code         8   NUMERIC
4 localisation       Ville       -10      TEXT
```

sqlColumns (channel, sqtable)[,3:6]

```
  TABLE_NAME COLUMN_NAME DATA_TYPE TYPE_NAME
1 localisation          id         4   COUNTER
2 localisation     Adresse        12   VARCHAR
3 localisation        Code         4   INTEGER
4 localisation       Ville        12   VARCHAR
```

Accès aux données

➢ **Pas de sélection de colonnes**

DBI

(dDBI <- dbReadTable(con, sqtable))

```
 id        Adresse  Code     Ville
1  1  avenue Verdun 92170    VANVES
2  2 rue Gay Lussac 92320 CHATILLON
3  3    rue Roissis 92140   CLAMART
```

str(dDBI)

```
'data.frame':   3 obs. of  4 variables:
 $ id     : int  1 2 3
 $ Adresse: chr  "avenue Verdun" "rue Gay Lussac" "rue Roissis"
 $ Code   : int  92170 92320 92140
 $ Ville  : chr  "VANVES" "CHATILLON" "CLAMART"
```

ODBC

(dODBC <- sqlFetch (c_sqlite, sqtable))

```
 localisation.id localisation.Adresse localisation.Code localisation.Ville
1               1        avenue Verdun             92170            VANVES
2               2       rue Gay Lussac             92320         CHATILLON
3               3          rue Roissis             92140           CLAMART
```

str(dODBC)

```
'data.frame':   3 obs. of  4 variables:
 $ localisation.id     : int  1 2 3
 $ localisation.Adresse: Factor w/ 3 levels "avenue Verdun",..: 1 2 3
 $ localisation.Code   : num  92170 92320 92140
 $ localisation.Ville  : Factor w/ 3 levels "CHATILLON","CLAMART",..: 3 1 2
```

Pour avoir des caractères, mettre le paramètre stringsAsFactors = FALSE

Voir le pdf : «Utiliser R pour travailler avec une base de données» pour l'utilisation de sqlFetch() avec des données MS-Excel et un accès ligne à ligne (sqlFetchMore()).

Accès aux données

➢ Sélection des colonnes par un SELECT SQL

sql <- "SELECT * FROM localisation ;"

DBI

dbGetQuery(con, sql)

Première utilisation

```
  id        Adresse  Code     Ville
1  1  avenue Verdun 92170    VANVES
2  2 rue Gay Lussac 92320 CHATILLON
3  3    rue Roissis 92140   CLAMART
```

res <- dbSendQuery(con, sql) # Prépare la requête

fetch (res, n = 2) # Affiche n lignes (curseur), si n = -1 → jusqu'à la fin

Deuxième utilisation

```
  id        Adresse  Code     Ville
1  1  avenue Verdun 92170    VANVES
2  2 rue Gay Lussac 92320 CHATILLON
```

dbGetRowCount(res) # nb lignes lues

```
[1] 2
```

dbHasCompleted(res) # test la fin de l'affichage

```
[1] FALSE
```

fetch(res, n = -1)

```
  id     Adresse  Code   Ville
3  3 rue Roissis 92140 CLAMART
```

dbHasCompleted(res)

```
[1] TRUE
```

Autres fonctions :

dbClearResult (res) # libère le lien à la table. A faire pour un nouveau dbSendQuery() si dbHasCompleted(res)→FALSE

dbColumnInfo (res) # ≈ à sqlColumns()

dbGetStatement (res) # affiche le SELECT

ODBC

sqlQuery(c_sqlite, sql)

```
  localisation.id localisation.Adresse localisation.Code localisation.Ville
1               1        avenue Verdun            92170            VANVES
2               2       rue Gay Lussac            92320         CHATILLON
3               3          rue Roissis            92140           CLAMART
```