Using Databases

2020-04-28

Chapter 1

Preface

Chapter 2

Introduction

With the advent of large and remote data sources comes a need properly understand the methods for interacting with such data. The Cal Poly statistics department, at the current time, lacks a formal remote data course. This leaves it up to the discresion of professor to include such material in thier courses, and possibly leaves students to learn these essential skills outside of the classroom.

The technical purpose of this book is to demostrate how to *connect to*, *upload data to*, and *read data from* the statistics department online dataset repository.

All the datasets are stored in a *relational database*, meaning that they are stored in a standardized, tabular format. The tables are related according to specific columns, or *keys*. Additionally, they are stored in a remote server that can be accesses from any computer with an internet connection and the proper usernames and passwords.

Chapter 3

Machine Setup

The manual takes uses the OCDB driver/ connection string method to connnect to the Stats repository.These drivers are proprietary to the SQL service you are using, so they need to be installed for each server type you wish to connect to.

Connection strings refer to the passwords and information needed to connect to the database, as well as the format that that information needs to be specified in. Connection strings are also specific to the database you are connecting to. For this example the driver and connection string can be found at the link below.

Microsoft SQL Server Drivers can be found at: https://docs.microsoft.com/en-us/sql/connect/odbc/microsoft-odbc-driver-for-sql-server?view=sql-server-ver15

After downloading, the drivers can be installed by running the following commands in a terminal.

Linux - Ubuntu

```
sudo su
curl https://packages.microsoft.com/keys/microsoft.asc | apt-key add -

#Download appropriate package for the OS version
#Choose only ONE of the following, corresponding to your OS version

#Ubuntu 16.04
curl https://packages.microsoft.com/config/ubuntu/16.04/prod.list > /etc/apt/sources.list.d/mssql

#Ubuntu 18.04
curl https://packages.microsoft.com/config/ubuntu/18.04/prod.list > /etc/apt/sources.list.d/mssql

#Ubuntu 19.10
```

```
curl https://packages.microsoft.com/config/ubuntu/19.10/prod.list > /etc/apt/sources.li

exit
sudo apt-get update
sudo ACCEPT_EULA=Y apt-get install msodbcsql17
# optional: for bcp and sqlcmd
sudo ACCEPT_EULA=Y apt-get install mssql-tools
echo 'export PATH="$PATH:/opt/mssql-tools/bin"' >> ~/.bash_profile
echo 'export PATH="$PATH:/opt/mssql-tools/bin"' >> ~/.bashrc
source ~/.bashrc
# optional: for unixODBC development headers
sudo apt-get install unixodbc-dev
```

Mac OS

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/maste
brew tap microsoft/mssql-release https://github.com/Microsoft/homebrew-mssql-release
brew update
HOMEBREW_NO_ENV_FILTERING=1 ACCEPT_EULA=Y brew install msodbcsql17 mssql-tools
```

Windows

download and open msodbcsql.msi file from: https://docs.microsoft.com/en-us/sql/connect/odbc/download-odbc-driver-for-sql-server?view=sql-server-ver15#download-for-windows

Other Connection Options

Once drivers are installed, you can set a dsn (data source name) for the connection to the database using the ODBC/DSN manager for your system.

This is OS-specific and while it will allow a single user to access the database more easily, it must be configured for every user who wishes to connect to the data source. Therefore, to more easily share code and to not have to mess with their behind-the-scenes settings, I use the connection string method from here further.

Chapter 4

Connecting to a Database

Julia

```
using ODBC

# driver/connection string found at
# https://docs.microsoft.com/en-us/sql/connect/odbc/microsoft-odbc-driver-for-sql-server?view=sql

#enter user information to be passed into the connection string
database= begin
    print("Enter Database Name: ")
    readline()
end

user= begin
    print("Enter Username: ")
    readline()
end

crypt=Base.getpass("Enter Password")
pass=read(crypt,String)

# setting up database
dsn = ODBC.DSN("Driver={ODBC Driver 17 for SQL Server};Address=24.205.251.117;Database=$database;
ODBC.disconnect()
```

SAS

```
/*from proc sql directly*/
```

```
/*this is pretty convoluted*/
proc sql;
connect to odbc as conn
required="Driver={ODBC Driver 17 for SQL Server};Address=24.205.251.117;Database=ntsb;

create table event as
select * from connection to conn
(select * from events) ;

disconnect from conn;
quit;

/*by using a libname*/
libname conn2 odbc
required ="Driver={ODBC Driver 17 for SQL Server};Address=24.205.251.117;Database=ntsb
proc sql;
create table event as
select * from conn2.events ;
quit;
```

R

```r
library(odbc)
library(DBI)

# note: if files are private, can replace with raw strings of database, uid, and pwd
database = rstudioapi::showPrompt("Database name","Database name")
uid = rstudioapi::showPrompt("Database username", "Database username")
pwd = rstudioapi::askForPassword("Database password")

# open connection
conn <- dbConnect(
  odbc(),
  Driver = "ODBC Driver 17 for SQL Server",
  Server = "24.205.251.117",
  Database = database,
  UID = uid,
  PWD = pwd
)

# close connection when done
dbDisconnect(conn)
```

Python

The driver used for Python may depend on your setup.

```
import pyodbc

driver = "{ODBC Driver 17 for SQL Server}" # PC Users
# driver = "/usr/local/lib/libmsodbcsql.17.dylib" # Mac Users

# note: if files are private, can replace with strings
database = input("Database name: ")
username = input("Username: ")
password = getpass.getpass(prompt = "Password: ")

# open connection
conn = pyodbc.connect(
    ";".join([
        "Driver="+driver,
        "Address=24.205.251.117",
        "Database="+database,
        "UID="+username,
        "PWD="+password
    ])
)

# get cursor
cursor = conn.cursor()

# close connection when done
conn.close()
```

JSL

```
opendatabaseconnection(
        "Driver={ODBC Driver 17 for SQL Server};Address=24.205.251.117;Database=ntsb;UID=ntsb;PWD
)
```

Chapter 5

Writing to a Database

Julia

```
using ODBC, CSV, JuliaDB, Query

dsn = ODBC.DSN("Driver={ODBC Driver 17 for SQL Server};Address=24.205.251.117;Database=$database;
#=
depending on your ODBC driver, you could do something as simple as

ODBC.load(database, "name of sql table", julia_table)

some ODBC drivers/database systems don't support this however, including MS SQL Server
so we must use an alternative method
=#

## create the tables
ODBC.execute!(dsn,"""
create table projects
(Class VARCHAR(10), id INT, StartDate INT, EndDate INT)
""")

ODBC.execute!(dsn,
"create table users
(Class VARCHAR(10), Fname VARCHAR(16), Lname VARCHAR(16), Email VARCHAR(36), Phone VARCHAR(9), De

## make a sql statement with "blank" values that we'll put in actual values into
insertstmt=ODBC.prepare(dsn,"insert into projects values(?,?,?,?)")

## now for each row in the rows of gridprojects, run the previous insert statemet, with the value
## inserted into the previous "blank rows" ie those question marks

for row in rows(gridprojects)
    ODBC.execute!(insertstmt,row)
end

## make a sql statement with "blank" values that we'll put in actual values into
insertstmt=ODBC.prepare(dsn,"insert into users values(?,?,?,?,?,?,?)")

## now for each row in the rows of gridusers, run the previous insert statemet, with t
## inserted into the previous "blank rows" ie those question marks
for row in rows(gridusers)
    ODBC.execute!(insertstmt,row)
end

# select top 50 rows from sample table, select as a julia DB indexed table.
results=ODBC.query(dsn,"select TOP 50 * from projects") |>
        table



#==============================  Loading Data Bigger than Memory =====================
# ok so this isn't actually larger than memory, but it should work for any delimited t

ODBC.execute!(dsn,""""
create table iris
(Sepal_length Float, Sepal_width Float, Petal_length FLOAT, Petal_width FLOAT, Species
""")

insertstmt=ODBC.prepare(dsn,"insert into iris values(?,?,?,?,?)")

#================================== Utilizing the CSV package (more optimized)========

for row in CSV.Rows("iris.csv",skipto=2 )
    ODBC.execute!(insertstmt,row)
end

#================================== In base Julia (less optimized)===================

ODBC.execute!(dsn,"""
create table iris2
(Sepal_length Float, Sepal_width Float, Petal_length FLOAT, Petal_width FLOAT, Species
""")

insertstmt=ODBC.prepare(dsn,"insert into iris values(?,?,?,?,?)")
```

```
## open the file
open("iris.csv") do f
  #put this readline in if you'd like to skip the first row
    readline(f)
    #for the remaining lines, read in as a single string, break apart at the commas, then insert
   for lines in readlines(f)
    rawline=map(String,split(lines, ","))
    ODBC.execute!(insertstmt,rawline)
   end
end



#========================================Bulk Insert ==========================================
#=
If you have bulk insert permissions you can do something like below to write data without reading
Note, this is not tested, since I do not have bulk load permissions, so use at own risk
more info can be found at

https://docs.microsoft.com/en-us/sql/relational-databases/import-export/import-bulk-data-by-using
=#

#=
ODBC.execute!(dsn,"""
BULK INSERT iris
FROM 'iris.csv'
""")

ODBC.execute!(dsn,
"""
INSERT INTO iris (Sepal_length, Sepal_width, Petal_length, Petal_width, Species)
SELECT *
FROM OPENROWSET (
    BULK 'iris.csv') AS b ;
""")
=#
ODBC.disconnect!(dsn)
```

R

Simple Table: Iris

```
# drop table from database, if it exists
dbExecute(conn, 'drop table if exists iris;')
```

```r
# create new table from iris dataset
dbWriteTable(conn, name = "iris",  value = iris)

# make sure table exists in database
dbListTables(conn, table_name = "iris")

# list fields the new table
dbListFields(conn, name = "iris")
```

Related Tables Connected by Keys: Bakery Files

```r
# load dataframes
customers <- read.csv('BAKERY/customers.csv', stringsAsFactors = FALSE)
goods <- read.csv('BAKERY/goods.csv', stringsAsFactors = FALSE)
items <- read.csv('BAKERY/items.csv', stringsAsFactors = FALSE)
receipts <- read.csv('BAKERY/receipts.csv', stringsAsFactors = FALSE)

# clean Item column
items$Item <- trimws(items$Item)

# can't drop a table if it is referenced by another table
dbExecute(conn, 'drop table if exists items;')
dbExecute(conn, 'drop table if exists goods;')
dbExecute(conn, 'drop table if exists receipts;')
dbExecute(conn, 'drop table if exists customers;')

## --- customers ---
# create new table from customers dataset
dbWriteTable(conn, name = "customers",  value = customers)

# the column we want as a primary key must not be null, then we can add the constraint
dbExecute(conn, 'alter table customers alter column Id int not null;')
dbExecute(conn, 'alter table customers add primary key (Id);')

# check key usage in this table
dbGetQuery(conn, "select * from information_schema.key_column_usage where TABLE_NAME =

## --- goods ---

# create new table from goods dataset
dbWriteTable(conn, name = "goods",  value = goods)

# the column we want as a primary key must not be null, then we can add the constraint
dbExecute(conn, 'alter table goods alter column Id varchar(20) not null;')
dbExecute(conn, 'alter table goods add primary key (Id);')

# check key usage in this table
dbGetQuery(conn, "select * from information_schema.key_column_usage where TABLE_NAME = 'goods';")

## --- receipts ---

# create new table from receipts dataset
dbWriteTable(conn, name = "receipts",  value = receipts)

# the column we want as a primary key must not be null, then we can add the constraint
dbExecute(conn, 'alter table receipts alter column ReceiptNumber int not null;')
dbExecute(conn, 'alter table receipts add primary key (ReceiptNumber);')

dbExecute(conn, 'alter table receipts alter column CustomerID int not null;')
dbExecute(conn, 'alter table receipts add foreign key (CustomerID) references customers (Id);')

# check key usage in this table
dbGetQuery(conn, "select * from information_schema.key_column_usage where TABLE_NAME = 'receipts'

## --- items ---

# create new table from items dataset
dbWriteTable(conn, name = "items",  value = items)

# the column we want as a primary key must not be null, then we can add the constraint
dbExecute(conn, 'alter table items alter column Receipt int not null;')
dbExecute(conn, 'alter table items alter column Ordinal int not null;')
dbExecute(conn, 'alter table items add primary key (Receipt, Ordinal);')

dbExecute(conn, 'alter table items add foreign key (Receipt) references receipts (ReceiptNumber);

dbExecute(conn, 'alter table items alter column Item varchar(20) not null;')
dbExecute(conn, 'alter table items add foreign key (Item) references goods (Id);')

# check key usage in this table
dbGetQuery(conn, "select * from information_schema.key_column_usage where TABLE_NAME = 'items';")
```

Strategy for Large Tables

```
# drop table from database, if it exists
dbExecute(conn, 'drop table if exists iris;')

dbExecute(conn, paste(
  "create table iris(",
    "SepalLength decimal(5,2),",
    "SepalWidth decimal(5,2),",
    "PetalLength decimal(5,2),",
    "PetalWidth decimal(5,2),",
    "Species varchar(50)",
  ");",
  sep = ' '
))

f <- file("iris.csv", open = "r")
first = TRUE
while (length(oneLine <- readLines(f, n = 1)) > 0) {
  if (first) {
    first = FALSE
    next
  }
  myLine <- unlist((strsplit(oneLine, ",")))
  insert_statement <- paste(
    "insert into iris(",
      "SepalLength,",
      "SepalWidth,",
      "PetalLength,",
      "PetalWidth,",
      "Species",
    ") values (",
      as.numeric(myLine[1]), ",",
      as.numeric(myLine[2]), ",",
      as.numeric(myLine[3]), ",",
      as.numeric(myLine[4]),",'",
      str_replace_all(myLine[5], '\"',''), "');"
  )
  dbExecute(conn, insert_statement)
}
close(f)
```

Python

Simple Table: Iris

```
# load in data
with open('iris.csv') as file:
    iris_lines = file.readlines()[1:]
```

```
# drop table if it already exists
cursor.execute("drop table if exists iris;")

# create new table
cursor.execute(" ".join([
    "create table iris(",
        "SepalLength decimal(5,2),",
        "SepalWidth decimal(5,2),",
        "PetalLength decimal(5,2),",
        "PetalWidth decimal(5,2),",
        "Species varchar(50)"
    ");"
]))

# single input: write one item to table
line = iris_lines[0].split(',')
cursor.execute(
    " ".join([
        "insert into iris(",
            "SepalLength,",
            "SepalWidth,",
            "PetalLength,",
            "PetalWidth,",
            "Species"
        ")"
        "values(?, ?, ?, ?, ?)"]),
    line[0],
    line[1],
    line[2],
    line[3],
    line[4].strip()
)

# --OR--

# bulk input: write many items to table
cursor.executemany(
    " ".join([
            "insert into iris(",
                "SepalLength,",
                "SepalWidth,",
                "PetalLength,",
                "PetalWidth,",
                "Species"
            ")"
            "values(?, ?, ?, ?, ?)"]),
    [line.split(',') for line in iris_lines[1:]]
)
```

Related Tables Connected by Keys: Bakery Files

```
# drop tables if they already exists
# can't drop a table if it is referenced by another table
cursor.execute("drop table if exists items;")
cursor.execute("drop table if exists goods;") # referenced by items
cursor.execute("drop table if exists receipts;") # referenced by items
cursor.execute("drop table if exists customers;") # referenced by receipts

# create new tables
cursor.execute(" ".join([
    "create table customers(",
        "Id int,",
        "LastName varchar(50),",
        "FirstName varchar(50),",
        "constraint customers_pk primary key (Id)"
    ");"
]))

cursor.execute(" ".join([
    "create table receipts(",
        "ReceiptNumber int,",
        "Date date,",
        "CustomerId int,",
        "constraint receipts_pk primary key (ReceiptNumber),",
        "constraint receipts_customers_fk foreign key (CustomerId) references customers
    ");"
]))

cursor.execute(" ".join([
    "create table goods(",
        "Id varchar(50),",
        "Flavor varchar(50),",
        "Food varchar(50),",
        "Price decimal(5,2),",
        "constraint goods_pk primary key (Id)"
    ");"
]))

cursor.execute(" ".join([
    "create table items(",
        "Receipt int,",
        "Ordinal int,",
        "Item varchar(50),",
        "constraint items_pk primary key (Receipt, Ordinal),",
        "constraint items_goods_fk foreign key (Item) references goods (Id),",
        "constraint items_receipts_fk foreign key (Receipt) references receipts (ReceiptNumber)"
    ");"
]))

# load in data
with open('BAKERY/customers.csv') as file:
    header = True
    # iterating through lines in this way doesn't read everything to memory at once
    # useful strategy for large files!
    for line in file:
        line = line.split(', ')
        if header:
            header = False
        else:
            cursor.execute(
                " ".join([
                    "insert into customers(",
                        "Id,",
                        "FirstName,",
                        "LastName",
                    ")"
                    "values(?, ?, ?)"]),
                line[0],
                line[1],
                line[2].strip()
            )

# load in data
with open('BAKERY/receipts.csv') as file:
    header = True
    for line in file:
        line = line.split(', ')
        if header:
            header = False
        else:
            # YYYYMMDD from DD-Mon-YYYY
            date = line[1].strip().replace("'",'')
            year = date.split('-')[2]
            day = date.split('-')[0]
            if len(day) == 1:
                day = '0'+day
            month = str(month_abbr_dict[date.split('-')[1]])
            date2 = year+month+day
            cursor.execute(
                " ".join([
                    "insert into receipts(",
                        "ReceiptNumber,",
                        "Date,",
                        "CustomerId",
                    ")"
                    "values(?, ?, ?)"]),
                line[0],
                date2,
                line[2].strip()
            )

# load in data
with open('BAKERY/goods.csv') as file:
    header = True
    # iterating through lines in this way doesn't read everything to memory at once
    # useful strategy for large files!
    for line in file:
        line = line.strip().split(',')
        if header:
            header = False
        else:
            cursor.execute(
                " ".join([
                    "insert into goods(",
                        "Id,",
                        "Flavor,",
                        "Food,",
                        "Price"
                    ")"
                    "values(?, ?, ?, ?)"]),
                line[0],
                line[1],
                line[2],
                float(line[3])
            )

# load in data
with open('BAKERY/items.csv') as file:
    header = True
    # iterating through lines in this way doesn't read everything to memory at once
    # useful strategy for large files!
    for line in file:
        line = line.split(', ')
        if header:
            header = False
        else:
            cursor.execute(
                " ".join([
                    "insert into items(",
                        "Receipt,",
                        "Ordinal,",
                        "Item",
                    ")"
                    "values(?, ?, ?)"]),
                int(line[0]),
                int(line[1]),
                line[2].strip()
            )
```

SAS

```
/*Window Prompt Courtesy of SAS Documentation */
/** This code is for the SAS windowing environment only. **/

/** %WINDOW defines the prompt **/
%window info
  #5 @5 'Please enter userid:'
  #5 @26 id 8 attr=underline
  #7 @5 'Please enter password:'
  #7 @28 pass 8 attr=underline display=no;

/** %DISPLAY invokes the prompt **/
%display info;


/*by using a libname, much more straightforward*/
libname conn2 odbc
required ="Driver={ODBC Driver 17 for SQL Server};Address=24.205.251.117;Database=NickDb;UID=&id;

/*from proc sql directly*/
/*this is pretty convoluted*/
proc sql;
connect to odbc as conn
required="Driver={ODBC Driver 17 for SQL Server};Address=24.205.251.117;Database=NickDb;UID=&id;P
```

```
create table event as
select * from connection to conn
(select * from events) ;

disconnect from conn;
quit;

/*writing data from work directory to database*/
filename download url "https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/da

data iris;
infile download delimiter = "," firstobs=2;
input var1-var4 species $;
run;


/*using the libname option again, turn on the bulkload for improved performance when lo
/*since sas doesn't load everything into memory we can directly put sas datasets into t
libname conn2 odbc
required ="Driver={ODBC Driver 17 for SQL Server};Address=24.205.251.117;Database=NickE
bulkload = YES;

proc sql;
create table conn3.iris as
(select * from work.iris);
quit;
```

Chapter 6

Querying a Database

Julia

R

```
# get full dataframe of table
data <- dbReadTable(conn, name = "iris")
head(data)

# get query results: returns dataframe
species <- dbGetQuery(conn, 'select distinct Species from iris')
species
```

Python

Standard Python

```
# execute a select statement
cursor.execute("select * from iris")
iris_out = cursor.fetchall()

# output is a list of pyodbc.Row objects
print(type(iris_out[1]))

# can access by column names
print(iris_out[1].SepalLength)

# confirm it's the same as the input
```

```
print(iris_lines[1].split(',')[0])
```

Pandas

```
import pandas as pd

# returns a Pandas DataFrame
iris_df_out = pd.read_sql("select * from iris", conn)
print(type(iris_df_out))
```

SAS