Appendix B

Connecting SQL to Python or R

In this Appendix we will briefly look at how to connect SQL to Python or R. Unfortunately connecting both programming languages can be difficult. There are a number of different ways to make the connection.

1 Connecting to any database: ODBC and JDBC

ODBC is an API for connecting to database systems. It was originally developed in the early 1990's, though it is still in use today. RODBC and PyODBC are the primary ways of connecting R and Python to ODBC interfaces.

JDBC is a JAVA extension of ODBC. It is the primary way that most SQL clients connect to databases. In order to use JDBC, a JDBC driver for a database must be provided. In the case of PopSQL, the driver is built-in, though most other SQL clients require the drivr to be downloaded.

JayDeBeApi and RJDBC are the two common tools for connecting Python and R to databases via JDBC drivers.

In both cases (ODBC and JDBC), the driver provides a standard interface between the client application and the database. These interfaces are specific to the database – you can think of them as printer drivers for your database. The PostgreSQL JDBC driver will not allow you to connect to a MS-SQL database.

2 Connecting only to PostgreSQL

If you only wish to connect to a single database variant, then you can use packages and programs that are built around that server, rather than more generalized packages.

Generally speaking variant specific tools tend to be more robust and easier to use. The downside is that information about one cannot necessarily be used for other variants.

For Python, the most common tool for connecting to PostgreSQL is the package `psycopg2` while R has a package `RPostgreSQL`. Using either tool requires setting up a connect and then sending queries through that connection.

As an example, the following Python code attempts to create a table. If there is an error, the connection is reset. Note the use of both the "commit" command and the "rollback" command. These are necessary because `psycopg2` does not automatically commit its transaction. Keep in mind that the code below will not run without providing host, db name, user and password information.

Installing `psycopg2` can be difficult. On Macs I install it using brew, though it can be installed via other library management tools.

```
conn_string = "host='%s' dbname='%s' user='%s' password='%s'" % (ahost, adbname, aUser, apass)
Sconn = psycopg2.connect(conn_string)
Scur = Sconn.cursor()

cmds = [ """create table cls.cars (
    year int
    , countyname varchar(20)
    , motorvehicle varchar(3)
    , vehiclecat varchar(15)
    , vehicletype varchar(55)
    , tonnage  varchar(30)
    , registrations int
    , annualfee float
    , completecategory varchar(90)
);"""]

for x in cmds:
        try:
                Scur.execute(x)
                Sconn.commit()
        except psycopg2.ProgrammingError:
                print( """CAUTION FAILED: '%s' """ % x)
                Sconn.rollback()
```

Using RPostgreSQL, the code below will create an object with the result of the query:

```
require("RPostgreSQL")
drv <- dbDriver("PostgreSQL")
con <- dbConnect(drv, dbname = "XXX", host = "localhost"
                 , port = 5432, user = "XXX", password = "XXX")
df_postgres <- dbGetQuery(con, "SELECT * from cls.traffic;")
```