Working with relational databases (continued)

The SELECT statement

- There is a programming language called SQL (Structured Query Language) to manipulate relational DB. We will learn how to use SQL to read data from relational DB.
- Remember that a database is a data file managed by a database system. And to access the data we need to connect and interact with the database system.

The SELECT statement

- As we have seen there are many different database systems. And how we connect to the database depends on the database system, and on the computing environment.
- However, once connection is established, we use the same SQL language for all relational databases.
- In this class, we will connect to SQLite and MySQL databases from within R, using the packages `RSQLite` and `RMYSQL`.

The SELECT statement

- The SELECT command builds a table from the tables in database. Here is a very basic syntax:

  `SELECT column(s) FROM Table(s) WHERE row_conditions;`

- This statement will select the specified column(s) from the specified table(s) and will return all rows matching the `row_conditions`.
- SQL is not case sensitive, but we will use capital letters for the key words of the language such as `SELECT`, `FROM`, `WHERE`.
- Note however that the naming of tables and columns is case sensitive.

The SELECT statement

- The column(s) may be
  - a comma-separated list of column names, e.g. Name, Addr
  - the character * to indicate all columns
  - aggregate function such as MIN(Balance), AVG(Balance), SUM(Balance), COUNT(Balance), etc.
- The Table(s) is the name of a single table or a comma separated list of tables.
- The statement ends with a semi-colon.
- The result is a new table.

WHERE clause

- The WHERE clause allows you to identify a subset of the rows. As in

```
SELECT column(s) FROM Table(s)
                    [WHERE row_conditions];
```

- The conditions that appears in the WHERE clause will be a logical statement of some sort.
- The square brackets `[ ]` are not to be written. It is used here only to indicate that the WHERE clause if optional.

WHERE clause

- `X = 3` will select all the rows where `X` is 3.
- `X > 3 AND Y < Z` will select all the rows where `X` is greater than 3 and the value in `Y` is less than `Z`.
- `Letter IN ('A' ,'B', 'X', 'Y', 'Z')` will select all the rows where the column `Letter` has one of the five values provided.
- In the WHERE clause, use quotations for columns that takes character vectors. For instance, write `X = 3` if X is numeric, and `X = '3'` if X is character vectors.

Query a table with SELECT

- The file "Test.db" is a small sqlite database available in ctools. It contains the CO2 emissions (in metric ton) per $\$1,000$ of GDP for the year 2001-2006 for a number of country.
- The database contains a single table "emissions" with the variables:

```
"country", "val_2001","val_2002","val_2003",
        "val_2004","val_2005","val_2006".
```

- SQlite documentation is freely available online at `http://www.sqlite.org/`.

Connecting to SQLite databases in R

There is a standard approach to connecting to a DB in R. First

- Load the necessary package

```
#install.packages("RSQLite",dependencies=T)
##if necessary
library("RSQLite")
```

- Load the driver and connect to the database

```
drv=dbDriver("SQLite")
conn=dbConnect(drv,"Test.db")
```

- Work with the database... When you're done, close the connection.

```
dbDisconnect(conn);
dbUnloadDriver(drv);
```

Connecting to SQLite databases in R

- Currently we are working with a database that is physically in our working directory.
- But if the database is located on a remote computer, and/or requires login information, we can provide this information to dbConnect via the arguments user, password, host, port.
- We will see an example later.

Connecting to SQLite databases in R

- Once you open the connection, all supported DBs (SQLite, MySQL, PostgreSQL, ROracle) are handled in R exactly the same way.
- For instance, for MySQL databases, we basically make the following changes

```
library("RMySQL")
drv=dbDriver("MySQL")
```

- For oracle databases, use library Roracle (change "MySQL" to "Oracle" in the two lines above).

Connecting to SQLite databases in R

- For PostgreSQL

```
library("RPostgreSQL")
drv=dbDriver("PostgreSQL")
```

- A notable exception is Microsoft DB systems (MS Access and MS SQL Server). For those, use package RODBC, which has a slightly different interface.

Connecting to SQLite databases in R

- Once connected to a database, use the function dbListTables to list the tables, use dbListFields to list the variables in a table.

```
dbListTables(conn)
[1] "emissions"
```

- List all the fields (columns) of table 'emissions'.

```
dbListFields(conn,'emissions')
[1] "country"  "val_2001" "val_2002"
[4] "val_2003" "val_2004" "val_2005"
[7] "val_2006"
```

Query a table with SELECT

- Use dbGetQuery to send in a SQL query. The returned object of the function dbGetQuery is a dataframe that we can readily use in our analysis.
- Suppose we want to retrieve the entire data:

```
sqltext="SELECT * FROM emissions;";
dt1=dbGetQuery(conn,sqltext)
```

  The $\star$ is a wildcard that indicates that we want all the variables in the table.

- What if we want specific variables like country and `val_2001`?

```
sqltext="SELECT country, val_2001 FROM emissions;";
dt2=dbGetQuery(conn,sqltext);
```

Query a table with SELECT

- If we don't like the number format, we can modify it.

```
sqltext="SELECT country, round(val_2001,3) AS v_2001,
     round(val_2006,3) AS v_2006 FROM emissions;"
     dt3=dbGetQuery(conn,sqltext);
```

- Here `'round()'` is a function applied to the selected columns. the clause `AS` changes the the name of the column to the specified name.
- Note that we typically don't want to round numbers if the goal is to perform a numerical data analysis later on.

Query a table with SELECT

- We may want to select specific countries.

```
sqltext="SELECT country, round(val_2001,3) AS v_2001,
     round(val_2006,3) AS v_2006 FROM emissions WHERE
     country IN (’\"United States\"’, ’\"China\"’,
         ’\"Brazil\"’, ’\"Egypt\"’);"
     dt3=dbGetQuery(conn,sqltext);
```

- The IN clause allows to search for specific list of values.
- Note that in the database the country names are written within quotes. So normally we need to write something like `"select... IN (’"USA"’,...);"`.
- To do that within confusing R we need the escape character `\`.

Query a table with SELECT

- SQL also implements some regular expressions with the clause LIKE. For instance suppose we want all the countries in the databases with names starting with U.

```
sqltext="SELECT country, round(val_2001,3), AS v_2001,
      round(val_2006,3) AS v_2006 FROM emissions WHERE
      country LIKE '\"U%';"
      dt3=dbGetQuery(conn,sqltext);
```

- Here % is a special character meaning any sequence of characters. Another special character is _ which match any single character.
- Full-fledged regular expression available with `RLIKE`. More of that later.

Query a table with SELECT

Excercise:

1. Select all the countries where the 2006 emission is lower than the 2001 emission.
2. Select all countries with name starting with letter U or B that have 2006 emission lower than 2001.

Connecting to SQLite databases in R

Another Example: The baseball database.

- We have access to a Baseball database that contains 21 tables. These provide information about major league baseball teams and players from 1884 through 2003.
- The database comes from http://baseball1.info and was created and licensed by Sean Lahman. We are entitled to use it for educational purposes.
- The variables in the different tables are described in Baseball Archive documentation http://seanlahman.com/files/database/readme2014.txt.

Connecting to SQLite databases in R

Let us connect to that database.

```
install.packages("RSQLite",dependencies=T)
library("RSQLite")
drv=dbDriver("SQLite")
conn=dbConnect(drv,"baseball.db")
dbListTables(conn)
dbListFields(conn,"Master")
dbListFields(conn,"Salaries")
```

Baseball Example Queries

Suppose we want to know the names and detail of all players in the history of professional baseball that have graduated from Michigan.

```
dt1=dbGetQuery(conn,"SELECT * from Master
    where college='Michigan';")
```

Baseball Example Queries

What if we want players from both Michigan and Michigan State?

```
dt2=dbGetQuery(conn,"SELECT * from Master where
        college in ('Michigan','Michigan State');")
```

or

```
dt2=dbGetQuery(conn,"SELECT * from Master where
        college = 'Michigan'
          or college = 'Michigan State';")
```

Baseball Example Queries

How about player that went to either Michigan or Michigan state and born after 1979?

```
dt3=dbGetQuery(conn,"SELECT * from Master where
    college in ('Michigan','Michigan State')
        AND birthYear>1979")
```

How about finding all the players not born in Michigan but went to college in UM or MSU?

MORE SQL: GROUP BY and HAVING

- Grouping allows you to group rows with a similar value into a single row and operate on them together.
- In addition, the HAVING clause can accompany the GROUP BY clause.
- HAVING restricts the groups that are to be selected. It works similarly to the WHERE clause.
- Note that they are not interchangeable: the WHERE clause modifies the FROM, and the HAVING modifies the GROUP BY.

Examples

The baseball example: Let us see whether the distribution of home runs has increased over the years.

```
sqltext = "SELECT yearID,
    SUM(HR) as total_HR FROM Teams GROUP BY yearID;"
dt4=dbGetQuery(conn,sqltext)
plot(dt4[,1],dt4[,2],xlab='Years',
    ylab='Number of HRs',type='b',col='blue')
```

Examples

But we need to adjust for the number of games played by year.

```
sqltext = "SELECT yearID, sum(HR) as total_HR, 0.5*sum(G)
     as total_G from Teams GROUP BY yearID";
     dt5=dbGetQuery(conn, sqltext)
plot(dt5[,1],dt5[,2]/dt5[,3],xlab='Years',
    ylab='Number of HRs',type='b',col='blue')
```

Examples

Another example of grouping, What college produced the highest number of major league baseball players born after 1950? Show only colleges with at least 10 players.

```
dt7=dbGetQuery(conn, "SELECT college,
    count(playerID) as total_players from Master WHERE
        birthYear> 1950 GROUP BY college
        HAVING total_players > 10")
```

- Here, note how the `WHERE` limits the total players set, and the `HAVING` limit the groups.
- We can also order the result using the clause `ORDER BY`.

Joining Multiple Tables

Suppose that we have two tables:

CNo	Name	Address
1	Smith, F	101 Elm
2	Brown, D	17 Spruce

OrderNo	CNo	OrderDate	DelivAddr
201	1	"1-12-1999"	Address
301	2	"08-27-2001"	Address
302	2	"02-21-2004"	Address

Joining Multiple Tables

Then

```
SELECT * FROM Customers INNER JOIN Orders;
```

will given the following result.

Customer.CNo	Name	Address	OrderNo	Clients.CNo	OrderDate	DelivAddr
1	Smith, F	101 Elm	201	1	"1-12-1999"	Address
1	Smith, F	101 Elm	301	2	"08-27-2001"	Address
1	Smith, F	101 Elm	302	2	"02-21-2004"	Address
2	Brown, D	17 Spruce	201	1	"1-12-1999"	Address
2	Brown, D	17 Spruce	301	2	"08-27-2001"	Address
2	Brown, D	17 Spruce	302	2	"02-21-2004"	Address

Joining Multiple Tables

- Of course most of these combined tuples (records) are not interesting. The "ON" clause of the INNER JOIN asks to keep only the combined tuples that satisfy the ON condition.
- For the example above:

```
SELECT * FROM Customers INNER JOIN Orders
            ON Customers.CNo=Orders.CNo;
```

will given the following result.

Customer.CNo	Name	Address	OrderNo	Clients.CNo	OrderDate	DelivAddr
1	Smith, F	101 Elm	201	1	"1-12-1999"	Address
2	Brown, D	17 Spruce	301	2	"08-27-2001"	Address
2	Brown, D	17 Spruce	302	2	"02-21-2004"	Address

Joining Multiple Tables

- The condition attached to the ON clause could be any valid condition, not necessarily $V1 = V2$. It could be $V1 \leq V2$, or $V1 = V2$ AND $V3 = V4$ etc...
- But the INNER JOIN only return rows which hold data in the attributes (variables) involved in the ON clause and satisfy the ON condition.

Joining Multiple Tables

Example: find the winning percentage of baseball teams and their salary mass in 2000. Table "Teams" looks like this:

yearID	Year
lgID	League
teamID	Team
Rank	Position in final standings
G	Games played
GHome	Games played at home
W	Wins
L	Losses
LgWin	League Champion(Y or N)
WSWin	World Series Winner (Y or N)
...	
...	

Joining Multiple Tables

Table Salaries looks like this:

`yearID`	`Year`
`teamID`	`Team`
`lgID`	`League`
`playerID`	`Player ID code`
`salary`	`Player's salary during that year`

Joining Multiple Tables

```
dt6=dbGetQuery(conn2, "SELECT name, W, G, SUM(salary)
as salaries from Teams INNER JOIN Salaries ON
Teams.teamID=Salaries.teamID
     AND Teams.yearID = Salaries.YearID WHERE
yearID=2000 GROUP BY Teams.teamID")
```

Joining Multiple Tables

Example: For all the teams, find their annual salary and winning records after 1980. First find the salaries for each team and each year after 1980.

```
sqltext="SELECT Salaries.teamID AS team,
Salaries.yearID AS year, sum(salary) AS salaries
FROM Salaries WHERE Salaries.yearID>=1980
GROUP BY Salaries.teamID, Salaries.yearID"
dt6=dbGetQuery(conn,sqltext)
plot(dt6[,3]/10000000)
```

Joining Multiple Tables

Then join with table Teams.

```
sqltext="SELECT name, W,G, Salaries.teamID AS team,
 Salaries.yearID AS year, sum(salary) AS salaries
  FROM Teams inner join Salaries
  ON Teams.teamID=Salaries.teamID
  AND Teams.yearID=Salaries.yearID
  WHERE Teams.yearID>=1980
  GROUP BY Teams.teamID, Teams.yearID"
dt6=dbGetQuery(conn,sqltext)
```

Examples

Another example: The ENRON database.

- This is a MySQL database. More professional system with more security features.
- The University has agreed to install the ENRON database on one of its server for us. But the connection requires a usename and a password.
- The database has 4 tables ("employeelist", "message", "recipientinfo", "referenceinfo").
- As a way to familiarize with the database and apply the "INNER JOIN" structure, let us find the distribution of sent emails in the company.

Examples

```
install.packages("RMySQL",dependencies=T)
library('RMySQL')
drv=dbDriver('MySQL')
conn=dbConnect(drv,host='webapps-db-dev.web.itd.umich.edu',
dbname='statapp',user='statapp_ro',password='srdb406')
```

Use dbListTables and dbListFields to explore the database.

```
dbListTables(conn)
dbListFields(conn, 'employeelist')
```

Examples

Find the number of sent email by employee.

```
SQLtext="select firstname, lastname, Email_id,
  count(message_id) as mess_sent
FROM employeelist inner join message
ON employeelist.Email_id=message.sender
GROUP BY Email_id order by mess_sent desc"
dt1=dbGetQuery(conn, SQLtext)
```

Examples

Find the total number of messages sent in the company each month over time.

```
SQLtext="SELECT extract(YEAR_MONTH from date)
    as newdate, count(mid) as total
FROM message GROUP BY newdate"
dt2=dbGetQuery(conn, SQLtext)
```

Conclusion

In conclusion:

- We have seen the general structure of relational database and studied in detail the SELECT command of the SQL language.
- We have worked with the packages **RSQLite** and **RMYSQL** and seen how to use these packages to connect to database from within R. Some other useful database packages in R include **RORACLE, RODBC**.

Conclusion

- There are many good SQL resource online including wikipedia which has a number of well written article on various aspects of SQL.
- See also the links

  `http://www.w3schools.com/sql/default.asp`

  `https://www.stat.berkeley.edu/~spector/sql.pdf`