**Lecture Notes 5**
**R for Data Science: Ch 13**

**Relational Data & SQL**

## 1 Edgar F Codd

Mathematician and former IBM fellow known as the "Father of Database Management Systems". Codd formed the concepts for organizing and accessing data that are embodied in the relational database.

Codd worked in IBM's San Jose Research Lab in California in 1970, when he proposed that data be organized according to principles based on identified relations between various kinds of data. One of Codd's key ideas was the process of organizing data into the appropriate number of tables – a process known as normalization. He envisioned an easy to use query language based on a foundation of relational set theory. Additionally, he believed a DBMS should provide a standard access approach so that an application program did not have to be aware of how the data was organized. As a result, IBM in 1982 came out with the first version of what later became the Structured Query Language (SQL).

## 2 Relational Data

Data analysis typically requires you to combine multiple tables of data to answers the questions you are interested in. Relational data refers to these multiple tables of data where it is the relations and not the individual datasets that are most important. These relations will always be defined for a **pair** of tables and all other relationships can be built from that. To connect each pair of tables, you will need to use variables called keys. There are 3 types of keys:

1) **Primary key:** uniquely identifies an observation in its own table.
2) **Foreign key:** uniquely identifies an observation in another table.
3) **Surrogate key:** variable that can be created when a primary key doesn't exist. It makes it easier to match observations in the original data. This type of key is frequently used in R.

Notice that these classifications are not mutually exclusive. A variable can be both a primary key and a foreign key.

## 3 SQL

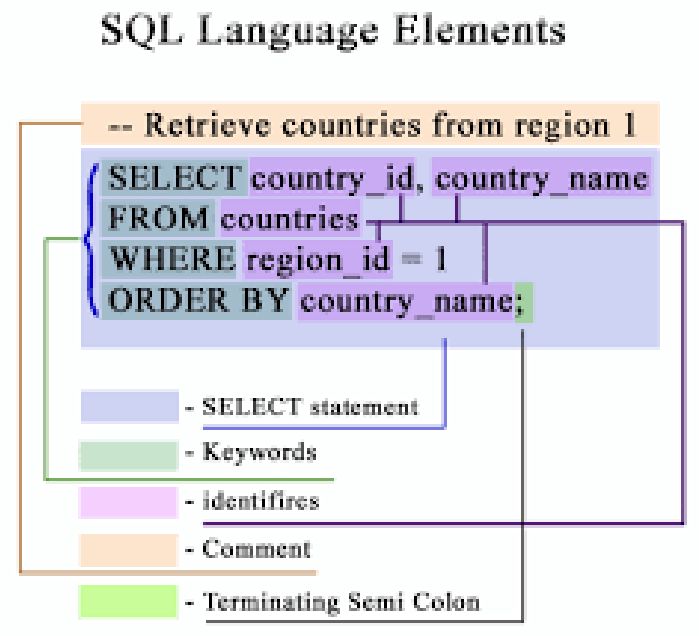
Relational database management systems (RDBMS) are the most common place to find relational data. Any relational database is a collection of data items organized as a set of formally-described tables from which data can be accessed or reassembled in many ways without having to reorganize the database tables.

SQL stands for Structured Query Language and is the standard language used to communicate with a database. Some common RDBMS that use SQL are:

- Oracle
- Sybase
- Microsoft SQL Server
- Access
- Ingres

**3.1 Syntax**

The SELECT, FROM, WHERE, and ORDER BY conditions are clauses in the SQL statement. Some are mandatory like SELECT, FROM while other are optional such as the WHERE clause.




**Note: SQL is not case sensitive. This means that the RDBMS treats SELECT and select the same. We will use uppercase letters for SQL keywords and lowercase letters for identifiers.**

The SELECT statement is used to query the database and retrieve selected data that match the criteria that you specify. To query data in SQL, you use the SELECT statement. This clause contains the syntax for selecting columns, selecting rows, grouping data, joining tables, and performing simple calculations. The SELECT statement is the most complex command in SQL. The following illustrates the basic syntax of the SELECT statement that retrieves data from a single table.

You specify a list of comma-separated columns that you want to see the data in the SELECT clause and the table in which you query the data in the FROM clause. When evaluating the SELECT statement, the RDBMS evaluates the FROM clause first before the SELECT clause. The semicolon (;) is not the part of a query. Typically, the RDBMS uses it to separate two SQL queries.

Besides the SELECT and FROM clauses, the SELECT statement has many other clauses. For example:

- **WHERE Clause**: filter data based on specified conditions using the WHERE clause.
- **AND operator**: combine multiple Boolean expressions using the AND clause to create a flexible condition.
- **OR operator**: show you how to use another logical operator OR to combine multiple Boolean expression
- **BETWEEN Operator**: guide you to use the BETWEEN operator to select data within a range of values.
- **IN Operator**: show you how to use the IN operator to check whether a value equals a list of values.
- **LIKE Operator**: do you want to query data based on a specific pattern? use the LIKE operator.
- **IS NULL Operator**: introduce the NULL concepts and show you how to check whether an expression is NULL or not
- **NOT operator**: show you how to negate a Boolean expression using the NOT operator.

**Note: When you are working with strings in SQL, you will need to use single quotes.**

**3.2 Distinct**

The DISTINCT operator is used to remove duplicates from a result set. You will need to insert it in the SELECT clause as indicated below:

```
SELECT DISTINCT  column1, column 2

FROM tables;
```

If you use one column after the DISTINCT operator, the RDBMS uses that column to evaluate duplicate. In case you use two or more columns, the RDBMS uses the *combination of*

*those columns* to evaluate duplicate.

### 3.3 Literals, Keywords and Identifiers

**Literals**: explicit values which are also known as constants. SQL provides three kinds of literals:

1) **String:** consists of one or more alphanumeric characters surrounded by single quotes, for example: 'John', '1990-01-01', '50'
   **NOTE: Remember to check that your characters are in single quotes**
2) **Numeric:** a sequence of digits proceeded by an optional sign (+/-). Ex: 7, or 3.14159.
3) **Binary:** represented by numeric values 0 and 1. In addition, the reserved keywords FALSE and TRUE can be used as aliases for 0 and 1 respectively.

**Keywords:** SQL has many keywords that have special meanings such as SELECT, INSERT, UPDATE, DELETE, DROP, etc. These keywords are the reserved words, therefore, you cannot use them as the name of tables, columns, indexes, views, stored procedures, triggers, etc.

**Identifiers:** refer to specific objects in the database such as tables, columns, indexes, etc. SQL is case insensitive with respect to keywords and identifiers. The following statements are equivalent:

```
Select * From employees;

SELECT * FROM EMPLOYEES;

select * from employees;

SELECT * FROM employees;
```

### 3.4 Sorting, Conditions, Aggregates, Grouping

**Sorting:** the ORDER BY clause is used to sort the date in ascending or descending order. You can use a single column or a list of columns using the ORDER BY clause. The basis syntax of the clause is as follows:

```
SELECT column
  FROM table
ORDER BY column
```

**Conditions:** Conditions can take several forms. The WHERE clause filters for rows that meet a certain criteria. Syntax is as follows:

```
SELECT column
  FROM table
WHERE condition
```

The HAVING clause introduces a condition on aggregations. This clause filters records that work on summarized GROUP BY results. Only the groups that meet the HAVING criteria will be returned. HAVING requires that a GROUP BY clause is present. Syntax is as follows:

```
SELECT column
  FROM table
WHERE condition
GROUP BY column
HAVING condition
```

Notice that WHERE and HAVING can be in the same query. However, it is not required to combine these two.

**Grouping:** The GROUP BY clause groups records into summary row. Clause returns one record for each group and typically also involves the use of an aggregate. Syntax below:

```
SELECT column
```

```
FROM table

GROUP BY column
```

**Aggregates:** aggregate functions are used to compute against a returned column of numeric data. They are used to summarize the results of a column of selected data. The table below summarizes the different aggregate functions and what they do.

| | |
|---|---|
| **MIN** | find the minimum value in a set |
| **MAX** | find the maximum value in a set |
| **SUM** | return the sum of a set |
| **AVG** | calculate the average value of a set |
| **COUNT** | Returns the total number of values in a set |
| **COUNT()** | return the number of rows in a set |

Clauses are used in combination with SELECT. For example:

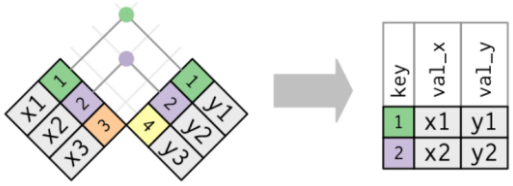
```
SELECT AVG column

FROM table

WHERE condition
```

**Note:** If you want to learn more SQL clauses look at the website below.
**http://www.dofactory.com/sql/tutorial**

## 5 Relational Data in R
## 5.1 R Joins

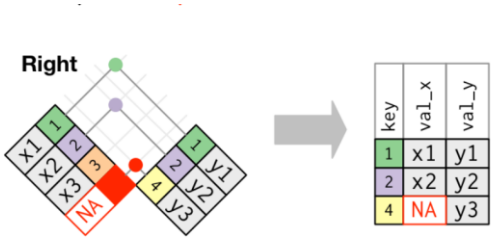
Joins in R are not as flexible as SQL joins, but are still an essential operation in the data analysis process. The diagrams below explain the joining functions and what they do. For each diagram, the colored column represents the "key" variable while the grey column represents the "value" column.

- **Inner join:** matches pairs of observations whenever their keys are equal




- **Outer_join:** keeps observations that appear in at least one of the tables. There are three types of outer joins:
    - **Left_join:** keeps all observations in x.


    - **Right_join:** keeps all observations in y.



- **Full_join** keeps all observations in x and y. Used to join multiple tables by including rows from both tables where or not the rows has a matching row from another table.