# Part 1 - Rsqldf Setup

### 1. Load Required Packages

install.packages("sqldf")   
library("sqldf")   
library(readxl) # only if reading excel xlsx file.

### 2. Importing Data to Environment

orders <- read\_excel("/Users/shayne\_lin/Desktop/work experiennce/UCSB/PSTAT10/LunchCost/orders.xlsx")

### 3. Examples from sqldf package

name\_counts\_emponly <- sqldf("SELECT firstname, COUNT(firstname) as occurances   
 FROM employees   
 WHERE firstname != 'rudi'   
 GROUP BY firstname")

# PART 2 - SQL Full Tutorial

## SECTION 1: Selecting Columns

### 4. Selcting Multiple Columns

To select multiple columns from a table, simply separate the column names with \*\*commas\*\*

### 5. Select All Columns From a Table

Typing out every column name would be a pain, so there's a handy shortcut: "SELECT \*":

SELECT \*   
 FROM people;

### 6. Return a Certain Number of Results

you can use the LIMIT keyword to limit the number of rows returned:

SELECT \*  
 FROM people  
 LIMIT 10;

**Instructions:**

Get the title, release year and country for every film.

SELECT title,release\_year,country  
 FROM films;

Get all columns from the films table.

SELECT \*  
 FROM films;

### 7. SELECT DISTINCT

If you want to select all the unique values from a column, you can use the **DISTINCT** keyword. This might be useful if, for example, you’re interested in knowing which languages are represented in the films table:

SELECT DISTINCT language  
 FROM films;

**Instructions:**

Get all the unique countries represented in the films table.

SELECT DISTINCT country  
 FROM films;

### 8. LEARNING TO COUNT

What if you want to count the number of employees in your employees table? The **COUNT( )** function lets you do this by returning the number of rows in one or more columns.

For example, this code gives the number of rows in the people table:

SELECT COUNT(\*)  
 FROM people;

### 9. Practice with COUNT

if you want to count the number of non-missing values in a particular column, you can call **COUNT( )** on just that column. Count returns with non-missing values;

For example, to count the number of birth dates present in the people table:

SELECT COUNT(birthdate)  
 FROM people;

It’s also common to combine **COUNT( )** with **DISTINCT** to count the number of distinct values in a column.

SELECT COUNT(DISTINCT birthdate)  
 FROM peoplesql

**Instructions:**  
Count the number of *rows* in the people table.

SELECT COUNT(\*)  
 FROM people;

Count the number of (non-missing) birth dates in the people table.

SELECT COUNT(birthdate)  
 FROM people;

Count the number of unique birth dates in the people table.

SELECT COUNT(DISTINCT birthdate)  
 FROM people;

## SECTION 2: Filtering Rows

### 10. WHERE AND

You can build up your WHERE queries by combining multiple conditions with the AND keyword:

SELECT title  
 FROM films  
 WHERE release\_year > 1994  
 AND release\_year < 2000;

OR, you can have different limitations for different fields. And You can add as many AND conditions as you need!

SELECT \*  
 FROM films  
 WHERE release\_year > 2000  
 AND release\_year < 2010  
 AND language = 'Spanish'

### 11. WHERE and OR

What if you want to select rows based on multiple conditions where some but not all of the conditions need to be met? For this, SQL has the OR operator.

For example, the following returns all films released in either 1994 or 2000:

SELECT title  
 FROM films  
 WHERE release\_year = 1994  
 OR release\_year = 2000;

Note that you need to specify the column for every OR condition, so the following is invalid:

SELECT title  
 FROM films  
 WHERE release\_year = 1994 OR 2000;

When combining AND and OR, be sure to enclose the individual clauses in parentheses, like so:

SELECT title, release\_year  
 FROM films  
 WHERE (release\_year >= 1990 AND release\_year < 2000)  
 AND (language = 'French' OR language = 'Spanish')  
 AND gross > 2000000;

Otherwise, due to SQL’s precedence rules, you may not get the results you’re expecting!

### 12. WHERE IN

WHERE is very useful for filtering results. However, if you want to filter based on many conditions, WHERE can get unwieldy. For example:

SELECT name  
 FROM kids  
 WHERE age = 2  
 OR age = 4  
 OR age = 6  
 OR age = 8  
 OR age = 10;

So, the *IN* operator will allow you to specify multiple values in a WHERE clause, making it easier and quicker to specify multiple OR conditions!

SELECT name  
 FROM kids  
 WHERE age IN (2, 4, 6, 8, 10);

### 13. BETWEEN

As you’ve learned, you can use the following query to get titles of all films released in and between 1994 and 2000:

SELECT title  
 FROM films  
 WHERE release\_year >= 1994  
 AND release\_year <= 2000;

Checking for ranges like this is very common, so in SQL the BETWEEN keyword provides a useful shorthand for filtering values within a specified range. This query is equivalent to the one above:

SELECT title  
 FROM films  
 WHERE release\_year  
 BETWEEN 1994 AND 2000;

It’s important to remember that BETWEEN is inclusive, meaning the beginning and end values are included in the results!

### 14. BETWEEN & AND & OR

Similar to the WHERE clause, the BETWEEN clause can be used with multiple AND and OR operators, so you can build up your queries and make them even more powerful! Things to note: BETWEEN AND is an inclusive function. Here’s an example:

SELECT title, release\_year  
 FROM films  
 WHERE release\_year BETWEEN 1990 AND 2000  
 AND budget > 100000000  
 AND (language = 'Spanish' OR language= 'French');

### 15. Introduction to NULL and IS NULL

In SQL, NULL represents a missing or unknown value. You can check for NULL values using the expression IS NULL. For example, to count the number of missing birth dates in the people table:

FROM people  
 WHERE birthdate IS NULL;

As you can see, IS NULL is useful when combined with WHERE to figure out what data you’re missing.

Sometimes, you’ll want to filter out missing values so you only get results which are not NULL. To do this, you can use the IS NOT NULL operator. For example, this query gives the names of all people whose birth dates are not missing in the people table.

SELECT name  
 FROM people  
 WHERE birthdate IS NOT NULL;

### 16. LIKE and NOT LIKE

As you’ve seen, the WHERE clause can be used to filter text data. However, so far you’ve only been able to filter by specifying the exact text you’re interested in. In the real world, often you’ll want to search for a pattern rather than a specific text string.

In SQL, the LIKE operator can be used in a WHERE clause to search for a pattern in a column. To accomplish this, you use something called a wildcard as a placeholder for some other values. There are two wildcards you can use with LIKE:

The % wildcard will match zero, one, or many characters in text. For example, the following query matches companies like ‘Data’, ‘DataC’ ‘DataCamp’, ‘DataMind’, and so on:

SELECT name  
 FROM companies  
 WHERE name LIKE 'Data%';

The \_ wildcard will match a single character. For example, the following query matches companies like ‘DataCamp’, ‘DataComp’, and so on:

SELECT name  
 FROM companies  
 WHERE name LIKE 'DataC\_mp';

You can also use the **NOT LIKE** operator to find records that don’t match the pattern you specify.

SELECT name  
 FROM people  
 WHERE name NOT LIKE 'A%';

## SECTION 3: Aggregate functions

Often, you will want to perform some calculation on the data in a database. SQL provides a few functions, called aggregate functions, to help you out with this.

For example,

SELECT AVG(budget)  
 FROM films;

gives you the average value from the budget column of the films table. Similarly, the MAX() function returns the highest budget:

SELECT MAX(budget)  
 FROM films;

The SUM() function returns the result of adding up the numeric values in a column:

SELECT SUM(budget)  
 FROM films;

You can probably guess what the MIN() function does!

However, when you want to find AVG of a calculation, it is not possible. You can find the average of an object first, then do the calculation:

Example: Get the average duration in hours for all films, aliased as avg\_duration\_hours.

SELECT AVG(duration/60.0) AS avg\_duration\_hours  
 FROM films;

This is wrong. Here’s the correct version:

SELECT AVG(duration)/60.0 AS avg\_duration\_hours  
 FROM films;

### 17. Combining aggregate functions with WHERE

Aggregate functions can be combined with the WHERE clause to gain further insights from your data.

For example, to get the total budget of movies made in the year 2010 or later:

SELECT SUM(budget)  
 FROM films  
 WHERE release\_year >= 2010;

### 18. A NOTE on ARITHMATIC

In addition to using aggregate functions, you can perform basic arithmetic with symbols like +, -, \*, and /.

So, for example, this gives a result of 12:

SELECT (4 \* 3);

However, the following gives a result of 1:

SELECT (4 / 3);

What’s going on here?

SQL assumes that if you divide an integer by an integer, you want to get an integer back. So be careful when dividing!

If you want more precision when dividing, you can add decimal places to your numbers. For example,

SELECT (4.0 / 3.0) AS result;

gives you the result you would expect: 1.333.

### 19. It’s AS simple AS aliasing

You may have noticed in the first exercise of this chapter that the column name of your result was just the name of the function you used. For example,

SELECT MAX(budget)  
 FROM films;

gives you a result with one column, named max. But what if you use two functions like this?

SELECT MAX(budget), MAX(duration)  
 FROM films;

Well, then you’d have two columns named max, which isn’t very useful!

To avoid situations like this, SQL allows you to do something called aliasing. Aliasing simply means you assign a temporary name to something. To alias, you use the AS keyword, which you’ve already seen earlier in this course.

For example, in the above example we could use aliases to make the result clearer:

SELECT MAX(budget) AS max\_budget,  
 MAX(duration) AS max\_duration  
 FROM films;

With calculations:

SELECT title, (gross-budget) AS net\_profit  
 FROM films;

### 20. MORE ALIASING

Practice:  
Get the percentage of people who are no longer alive. Alias the result as percentage\_dead. Remember to use 100.0 and not 100!

SELECT COUNT(deathdate) \* 100.0 / COUNT(\*) AS percentage\_dead  
 FROM people;

Because count only returns variables that are not null. If you count something, you should not also specify that the thing is not null, which will cause mistakes.

Get the number of years between the newest film and oldest film. Alias the result as difference.

SELECT MAX(release\_year)-MIN(release\_year) AS difference  
 FROM films;

Get the number of decades the films table covers. Alias the result as number\_of\_decades. The top half of your fraction should be enclosed in parentheses.

SELECT (MAX(release\_year) - MIN(release\_year))/10 AS number\_of\_decades  
 FROM films;

## SECTION 4: Sorting and Grouping

### 21. ORDER BY

In SQL, the ORDER BY keyword is used to sort results in ascending or descending order according to the values of one or more columns.

By default ORDER BY will sort in ascending order. If you want to sort the results in descending order, you can use the DESC keyword. For example,

SELECT title  
 FROM films  
 ORDER BY release\_year DESC;

### 22. Sorting Single Columns Practice

Get the title of films released in 2000 or 2012, in the order they were released.

SELECT title  
 FROM films  
 WHERE release\_year in (2000, 2012)  
 ORDER BY release\_year;

### 23. Sorting Multiple Columns Practice

ORDER BY can also be used to sort on multiple columns. It will sort by the first column specified, then sort by the next, then the next, and so on. For example,

SELECT birthdate, name  
 FROM people  
 ORDER BY birthdate, name;

sorts on birth dates first (oldest to newest) and then sorts on the names in alphabetical order. The order of columns is important!Remember, to specify multiple columns you separate the column names with a comma.

### 24. GROUP BY

Now you know how to sort results! Often you’ll need to aggregate results. For example, you might want to count the number of male and female employees in your company. Here, what you want is to group all the males together and count them, and group all the females together and count them. In SQL, GROUP BY allows you to group a result by one or more columns, like so:

SELECT sex, count(\*)  
 FROM employees  
 GROUP BY sex;

This might give, for example:

| sex | count |
| --- | --- |
| male | 15 |
| female | 19 |

Commonly, GROUP BY is used with aggregate functions like COUNT() or MAX(). Note that GROUP BY always goes after the FROM clause!

### 25. GROUP BY practice

As you’ve just seen, combining aggregate functions with GROUP BY can yield some powerful results!

A word of warning: SQL will return an error if you try to SELECT a field that is not in your GROUP BY clause without using it to calculate some kind of value about the entire group.

Note that you can combine GROUP BY with ORDER BY to group your results, calculate something about them, and then order your results. For example,

SELECT sex, count(\*)  
 FROM employees  
 GROUP BY sex  
 ORDER BY count DESC;

might return something like:

| sex | count |
| --- | --- |
| female | 19 |
| male | 15 |

because there are more females at our company than males. Note also that ORDER BY always goes after GROUP BY.

**EXERCISES:**  
1.Get the release year and count of films released in each year.

SELECT release\_year,COUNT(id)  
 FROM films  
 GROUP BY release\_year;

2.Get the release year and average duration of all films, grouped by release year.

SELECT release\_year,AVG(duration)  
 FROM films  
 GROUP BY release\_year;

3.Get the IMDB score and count of film reviews grouped by IMDB score in the reviews table.

SELECT imdb\_score,COUNT(id)  
 FROM reviews  
 GROUP BY imdb\_score;

### 26. GROUP BY practice (2)

Now practice your new skills by combining GROUP BY and ORDER BY with some more aggregate functions!

Make sure to always put the ORDER BY clause at the end of your query. You can’t sort values that you haven’t calculated yet!

Get the release year, country, and highest budget spent making a film for each year, for each country. Sort your results by release year and country.

SELECT release\_year,country,MAX(budget)  
 FROM films  
 GROUP BY release\_year,country  
 ORDER BY release\_year,country;

### 27. WHERE and AGGREGATE functions:AVG,MAX,MIN,SUM,GROUP BY cannot be together

In SQL, aggregate functions can’t be used in WHERE clauses. For example, the following query is *invalid*:

SELECT release\_year  
 FROM films  
 GROUP BY release\_year  
 WHERE COUNT(title) > 10;

This means that if you want to filter based on the result of an aggregate function, you need another way! That’s where the HAVING clause comes in. For example,

SELECT release\_year  
 FROM films  
 GROUP BY release\_year  
 HAVING COUNT(title) > 10;

shows only those years in which more than 10 films were released.

In how many different years were more than 200 movies released?

SELECT release\_year  
 FROM films  
 GROUP BY release\_year  
 HAVING COUNT(title) > 200;

### 28. All together

Time to practice using ORDER BY, GROUP BY and HAVING together.

Now you’re going to write a query that returns the average budget and average gross earnings for films in each year after 1990, if the average budget is greater than $60 million.

SELECT release\_year, AVG(budget) AS avg\_budget, AVG(gross) AS avg\_gross  
 FROM films  
 GROUP BY release\_year  
 HAVING AVG(budget)> 60000000  
 AND release\_year > 1990  
 ORDER BY avg\_gross DESC;

### 29. All together(2)

If you only want to return a certain number of results, you can use the LIMIT keyword to *limit* the number of rows returned

Get the country, average budget, and average gross take of countries that have made more than 10 films. Order the result by country name, and limit the number of results displayed to 5. You should alias the averages as avg\_budget and avg\_gross respectively.

SELECT country,AVG(budget) AS avg\_budget,AVG(gross) AS avg\_gross  
 FROM films  
 GROUP BY country  
 HAVING COUNT(title) > 10  
 ORDER BY country  
 LIMIT 5;

### 30. A taste of things to more-join example

SELECT title, imdb\_score  
 FROM films  
 JOIN reviews  
 ON films.id = reviews.film\_id  
 WHERE title = 'To Kill a Mockingbird';

# PART 3 - JOINING DATA IN SQL

## SECTION 1: Introduction to joins

### 31. Introduction to INNER JOIN

Basic frame of inner join:

SELECT \*  
 FROM left\_table  
 INNER JOIN right\_table  
 ON left\_table.id = right\_table.id;

example:

SELECT p1.country,p1.continent,prime\_minister,president  
 FROM prime\_misters AS p1  
 INNER JOIN presidents AS p2  
 ON p1.country = p2.country;

Now, use inner join to combine *cities* table with *countries* table, alias and joinning on cities.country\_code and countries.code.

SELECT \*  
 FROM cities AS city  
 INNER JOIN countries AS country  
 ON cities.country\_code = countries.code;

### 32. INNER JOIN (2)

Making alias with inner join’s table and inner join’s table name;

SELECT c1.name AS city, c2.name AS country  
 FROM cities AS c1  
 INNER JOIN countries AS c2  
 ON c1.country\_code = c2.code;

Notice that to select a field in your query that appears in multiple tables, you’ll need to identify which table/table alias you’re referring to by using a . in your SELECT statement.

Exercise: get data from both the countries and economies tables to examine the inflation rate for both 2010 and 2015.

SELECT c.code AS country\_code,c.name,e.year,e.inflation\_rate  
 FROM countries AS c  
 INNER JOIN economies AS e  
 ON c.code = e.code  
 WHERE e.year IN (2010,2015);

### 33. INNER JOIN- combining more than two tables at once

The ability to combine multiple joins in a single query is a powerful feature of SQL, e.g:

SELECT \*  
 FROM left\_table  
 INNER JOIN right\_table  
 ON left\_table.id = right\_table.id  
 INNER JOIN another\_table  
 ON left\_table.id = another\_table.id;

As you can see here it becomes tedious to continually write long table names in joins. This is when it becomes useful to alias each table using the first letter of its name (e.g. countries AS c)!

Now, for each country, you want to get the country name, its region, the fertility rate, and the unemployment rate for both 2010 and 2015.

SELECT c.code,c.name,c.region,e.year,p.fertility\_rate,e.unemployment\_rate  
 FROM countries AS c  
 INNER JOIN economies AS e  
 ON c.code = e.code  
 INNER JOIN populations AS p  
 ON c.code = p.country\_code  
 WHERE e.year in (2010,2015);

The above is wrong because of the last statement in inner join. When selecting fields that are in multiple tables, remember to making the connection on multiples.

explaination:The trouble with doing your last join on c.code = e.code and not also including year is that e.g. the 2010 value for fertility\_rate is also paired with the 2015 value for unemployment\_rate. Fix your previous query: in your last ON clause, use AND to add an additional joining condition. In addition to joining on code in c and e, also join on year in e and p.

-- Select fields  
 SELECT c.code, name, region, e.year, f rtility\_rate, unemployment\_rate  
 -- From countries (alias as c)  
 FROM countries AS c  
 -- Join to populations (as p)  
 INNER JOIN populations AS p  
 -- Match on country code  
 ON c.code = p.country\_code  
 -- Join to economies (as e)  
 INNER JOIN economies AS e  
 -- Match on country code and year  
 ON c.code = e.code AND e.year = p.year;

### 34. INNER JOIN with USING

Inner join with using  
When joining tables with a common field name, e.g.

SELECT \*  
 FROM countries  
 INNER JOIN economies  
 ON countries.code = economies.code

You can use USING as a shortcut:

SELECT \*  
 FROM countries  
 INNER JOIN economies  
 USING(code)

### 35. SELF JOIN

Self-joins are used to compare values in a field to other values of the same field from within the same table.

<Example:finishing> off the self-join on prime\_misnisters

SELECT p1.country AS country1,p2.country AS country2,p1.continent  
 FROM prime\_ministers AS p1  
 INNER JOIN prime\_minister AS p2  
 ON p1.continent = p2.continent AND p1.country = p2.country  
 LIMIT 13;

**Instructions**  
**1/3**

* Join populations with itself ON country\_code.
* Select the country\_code from p1 and the size field from both p1 and p2. SQL won’t allow same-named fields, so alias p1.size as size2010 and p2.size as size2015.

-- Select fields with aliases  
 SELECT p1.country\_code,  
 p1.size AS size2010,  
 p2.size AS size2015  
 -- From populations (alias as p1)  
 FROM populations AS p1  
 -- Join to itself (alias as p2)  
 INNER JOIN populations AS p2  
 -- Match on country code  
 ON p1.country\_code = p2.country\_code;

**Instructions**  
**2/3**

* Notice from the result that for each *country\_code* you have four entries laying out all combinations of 2010 and 2015.
* Extend the *ON* in your query to include only those records where the *p1.year* (2010) matches with *p2.year - 5* (2015 - 5 = 2010). This will omit the three entries per *country\_code* that you aren’t interested in.
* Even though you did not select it from the SELECT, as it is matching two completely same tables, you need to consider replication of data as you apply.

-- Select fields with aliases  
 SELECT p1.country\_code,  
 p1.size AS size2010,  
 p2.size AS size2015  
 -- From populations (alias as p1)  
 FROM populations as p1  
 -- Join to itself (alias as p2)  
 INNER JOIN populations as p2  
 -- Match on country code  
 ON p1.country\_code = p2.country\_code  
 -- and year (with calculation)  
 AND p1.year = p2.year-5;

**Instructions**  
**3/3**

* As you just saw, you can also use SQL to calculate values like *p2.year - 5* for you. With two fields like *size2010* and *size2015*, you may want to determine the percentage increase from one field to the next:
* With two numeric fields and , the percentage growth from to can be calculated as (B-A)/A\*100.0
* Add a new field to *SELECT*, aliased as *growth\_perc*, that calculates the percentage population growth from 2010 to 2015 for each country, using *p2.size* and *p1.size*

-- Select fields with aliases  
 SELECT p1.country\_code,  
 p1.size AS size2010,   
 p2.size AS size2015,  
 -- Calculate growth\_perc  
 ((p2.size - p1.size)/p1.size\* 100.0) AS growth\_perc  
 -- From populations (alias as p1)  
 FROM populations AS p1  
 -- Join to itself (alias as p2)  
 INNER JOIN populations AS p2  
 -- Match on country code  
 ON p1.country\_code = p2.country\_code  
 -- and year (with calculation)  
 AND p1.year = p2.year - 5;

### 36. Case when and then

Often it’s useful to look at a numerical field not as raw data, but instead as being in different categories or groups.

You can use CASE with WHEN, THEN, ELSE, and END to define a new grouping field.

**Instructions:**  
Using the *countries* table, create a new field *AS geosize\_group* that groups the countries into three groups:

* If *surface\_area* is greater than 2 million, *geosize\_group* is *‘large’*.
* If *surface\_area* is greater than 350 thousand but not larger than 2 million, *geosize\_group* is *‘medium’*.
* Otherwise, *geosize\_group* is *‘small’*.

SELECT name, continent, code, surface\_area,  
 -- First case  
 CASE WHEN surface\_area> 2000000 THEN 'large'  
 -- Second case  
 WHEN surface\_area > 350000 THEN 'medium'  
 -- Else clause + end  
 ELSE 'small' END  
 -- Alias name  
 AS geosize\_group  
 -- From table  
 FROM countries;

### 37. INTO to save edited table as a new table

The table you created with the added *geosize\_group* field has been loaded for you here with the name *countries\_plus*. Observe the use of (and the placement of) the *INTO* command to create this *countries\_plus* table:

SELECT name, continent, code, surface\_area,  
 CASE WHEN surface\_area > 2000000  
 THEN 'large'  
 WHEN surface\_area > 350000  
 THEN 'medium'  
 ELSE 'small' END  
 AS geosize\_group  
 INTO countries\_plus  
 FROM countries;

You will now explore the relationship between the size of a country in terms of surface area and in terms of population using grouping fields created with *CASE*.

By the end of this exercise, you’ll be writing two queries back-to-back in a single script.

**Instructions**  
**1/3**  
Using the *populations* table focused only for the *year* 2015, create a new field aliased as *popsize\_group* to organize population *size* into

* *‘large’* (> 50 million),
* *‘medium’* (> 1 million), and
* *‘small’* groups.  
  Select only the country code, population size, and this new *popsize\_group* as fields.

SELECT country\_code,size,  
 CASE WHEN size > 50000000  
 THEN 'large'  
 WHEN size > 1000000  
 THEN 'medium'  
 ELSE 'small'   
 END AS popsize\_group  
 FROM populations  
 WHERE year = 2015;

**Instructions**  
**2/3**  
\* Use *INTO* to save the result of the previous query as *pop\_plus*. You can see an example of this in the *countries\_plus* code in the assignment text. Make sure to include a *;* at the end of your *WHERE* clause!

* Then, include another query below your first query to display all the records in *pop\_plus* using SELECT \* FROM pop\_plus; so that you generate results and this will display *pop\_plus* in the query result.

SELECT country\_code,size,  
 CASE WHEN size > 50000000  
 THEN 'large'  
 WHEN size > 1000000  
 THEN 'medium'  
 ELSE 'small'   
 END AS popsize\_group  
 INTO pop\_plus  
 FROM populations  
 WHERE year = 2015;  
   
 SELECT \*  
 FROM pop\_plus

* Keep the first query intact that creates *pop\_plus* using *INTO*.
* Write a query to join *countries\_plus AS c* on the left with *pop\_plus AS p* on the right matching on the country code fields.
* Sort the data based on *geosize\_group*, in ascending order so that large appears on top.
* Select the name, continent, geosize\_group, and *popsize\_group* fields.

SELECT country\_code, size,  
 CASE WHEN size > 50000000  
 THEN 'large'  
 WHEN size > 1000000  
 THEN 'medium'  
 ELSE 'small' END  
 AS popsize\_group  
 INTO pop\_plus   
 FROM populations  
 WHERE year = 2015;  
   
 -- Select fields  
 SELECT name,continent,geosize\_group,popsize\_group  
 -- From countries\_plus (alias as c)  
 FROM countries\_plus AS c  
 -- Join to pop\_plus (alias as p)  
 INNER JOIN pop\_plus AS p  
 -- Match on country code  
 ON p.country\_code=c.code  
 -- Order the table   
 ORDER BY geosize\_group;

## SECTION 2: Outer joins and cross joins

Outer joins is reaching out to another table while keeping all of the records of the original table. While inner joins keep only the records IN both tables. Outer joins include (1) Left joins (2) Right joins (3) Full joins.

The syntax of a LEFT JOIN:

SELECT p1.country, prime\_minister, president  
FROM prime\_ministers AS p1  
LEFT JOIN presidents AS p2  
ON p1.country = p2.country;

### 38. Left Join

Combine two datasets together with inner joins and left joins, genenrally, left joins should end up with more variables at the end and inner joins end up with less variable at the end.

-- Select fields  
 SELECT region, AVG(gdp\_percapita) AS avg\_gdp  
 -- From countries (alias as c)  
 FROM countries AS c  
 -- Left join with economies (alias as e)  
 LEFT JOIN economies as e  
 -- Match on code fields  
 ON c.code = e.code  
 -- Focus on 2010  
 WHERE year = 2010  
 -- Group by region  
 GROUP BY c.region  
 -- Order by descending avg\_gdp  
 ORDER BY avg\_gdp DESC;

\*\*Left Joins are functionally the same as right joins using reverse, even in multiple tables’ combination case.

-- convert this code to use RIGHT JOINs instead of LEFT JOINs  
 /\*  
 SELECT cities.name AS city, urbanarea\_pop, countries.name AS country,  
 indep\_year, languages.name AS language, percent  
 FROM cities  
 LEFT JOIN countries  
 ON cities.country\_code = countries.code  
 LEFT JOIN languages  
 ON countries.code = languages.code  
 ORDER BY city, language;  
 \*/  
   
 SELECT cities.name AS city, urbanarea\_pop, countries.name AS country,  
 indep\_year, languages.name AS language, percent  
 FROM languages  
 RIGHT JOIN countries  
 ON countries.code = languages.code  
 RIGHT JOIN cities  
 ON cities.country\_code = countries.code  
 ORDER BY city, language;

### 39. Full joins

Full joins is a union of the combination of the sets, including both table\_1 and table\_2 attributes at the same time.

SELECT countries.name, code, languages.name AS language  
 -- From languages  
 FROM languages   
 -- Join to countries  
 FULL JOIN countries  
 -- Match on code  
 USING (code)  
 -- Where countries.name starts with V or is null  
 WHERE countries.name LIKE 'v%' OR countries.name IS NULL  
 -- Order by ascending countries.name  
 ORDER BY countries.name

### 40. CROSSING the rubicon-CROSS JOINS

CROSS JOINs create all possibe combinations of two tables. It helps generate all possible values of combinations of the table attributes. For example, table 1 has 3 attributes and table 2 has 3 attributes, together the make the cross joins of 9 attributes, assuming none of the values are the same.

syntax:

SELECT   
FROM   
CROSS JOIN

### 41. A table of two cities

**Instructions**  
**1/2**

* Create a *CROSS JOIN* with *cities AS c* on the left and *languages AS l* on the right.
* Make use of *LIKE* and *Hyder%* to choose Hyderabad in both countries.
* Select only the city name *AS city* and language name *AS language*

SELECT c.name AS city,l.name AS language  
FROM cities AS c  
CROSS JOIN languages AS l  
WHERE c.name LIKE 'Hyder%';

### 42. Outer challenge

In terms of life expectancy for 2010, determine the names of the lowest five countries and their regions.

**Instructions**  
\* Select country name AS country, region, and life expectancy AS life\_exp. \* Make sure to use LEFT JOIN, WHERE, ORDER BY, and LIMIT.

SELECT c.name AS country,c.region,p.life\_expectancy AS life\_exp  
FROM countries AS c  
LEFT JOIN populations AS p  
ON c.code = p.country\_code   
WHERE p.year = 2010  
ORDER BY life\_exp  
LIMIT 5;

## SECTION 3: Set theory clauses

### 43. Introduction to Set Theory Venn Theories

**Union**  
Includes only one overlap，which means the table will only include both of the attributes from the tables, but have everything appers only once.  
**Union All**  
Includes two overlaps, which means the table will also reveal the duplicates in the combination  
**Basic Syntax Example for UNION**

SELECT prime\_minister AS leader, country  
FROM prime\_ministers  
UNION  
SELECT monarch,country  
FROM monarchs  
ORDER BY country;

**Note**  
When you use UNION for the tables, make sure that the combined fields should have same type of data. For example, that a character type cannot be combined with a numeric type.

**Basic Syntax Example for UNION ALL**

SELECT prime\_minister AS leader, country  
FROM prime\_ministers  
UNION ALL  
SELECT monarch, country  
FROM monarchs  
ORDER BY country  
LIMIT 10;

**Intersect**  
Only includes the overlap once and only the overlap, only include the intersection of two tables.  
**Except**  
Only includes one side of the table and exclude the overlap of the two tables.

### 44. Union

**Instructions**  
\* Combine the two new tables into one table containing all of the fields in economies2010.  
\* Sort this resulting single table by country code and then by year, both in ascending order.

SELECT \*  
FROM economies2010  
UNION  
SELECT \*  
FROM economies2015  
ORDER BY code,year;

### 45. Union to Count Attributes Occurrences of a field for Multiple Tables(2)

UNION can also be used to determine all occurrences of a field across multiple tables. Try out this exercise with no starter code.

**Instructions:**

* Determine all (non-duplicated) country codes in either the cities or the currencies table. The result should be a table with only one field called country\_code.
* Sort by country\_code in alphabetical order.

SELECT c1.code  
FROM currencies AS c1  
UNION  
SELECT c2.country\_code  
FROM cities AS c2  
ORDER BY c2.country\_code

### 46. Union all

* As you saw, duplicates were removed from the previous two exercises by using UNION.
* To include duplicates, you can use UNION ALL.

**Instructions**

* Determine all combinations (include duplicates) of country code and year that exist in either the `economies or the populations tables. Order by code then year.
* The result of the query should only have two columns/fields. Think about how many records this query should result in.
* You’ll use code very similar to this in your next exercise after the video. Make note of this code after completing it.

-- Select fields  
 SELECT code,year  
 -- From economies  
 FROM economies  
 -- Set theory clause  
 UNION ALL  
 -- Select fields  
 SELECT country\_code, year  
 -- From populations  
 FROM populations  
 -- Order by code, year  
 ORDER BY code, year;

### 47. Introduction to Intersect

**Syntax**

SELECT id  
FROM left\_one  
INTERSECT  
SELECT id  
FROM right\_one;

**Distinction from joins and union all**

* INTERSECT looks for records in common, not individual key fields like what a join does to match.
* INTERSECT will only return records that both tables have in common.

**Instruction for Exercise**

Finding same information from both of the tables, across the table.  
Question: As you think about major world cities and their corresponding country, you may ask which countries also have a city with the same name as their country name?

-- Select fields  
 SELECT countries.name  
 -- From countries  
 FROM countries  
 -- Set theory clause  
 INTERSECT  
 -- Select fields  
 SELECT cities.name  
 -- From cities  
 FROM cities;

### 48. Introduction to Except

Remember, You saw earlier that there are some monarchs that also act as the prime minister for their country. One way to determine those monarchs in the monarchs table that do not also hold the title of prime minister is to use the EXCEPT clause. This SQL query selects the monarch field from monarchs and then looks for common entries with the prime\_ministers field, while also keeping track of the country for each leader. You can see in the resulting query that only the two European monarchs are not also prime ministers in the leaders database.  
Syntax:

SELECT monarch, country  
FROM monarchs  
EXCEPT   
SELECT prime\_minister,country  
FROM prime\_ministers;

Exercise: Get the names of cities in cities which are not noted as capital cities in countries as a single field result.

Note that there are some countries in the world that are not included in the countries table, which will result in some cities not being labeled as capital cities when in fact they are.

Instructions 100 XP Order the resulting field in ascending order. Can you spot the city/cities that are actually capital cities which this query misses?

SELECT countries.capital  
FROM countries  
EXCEPT  
SELECT cities.name  
FROM cities  
ORDER BY capital;

### 49. Introduction to Semi-joins and Anti-joins

* **Semi Join:** A semi-join matches records by key field in the right table with those in the left. It then picks out only the rows in the left table that match that condition.
* **Anti-join:** The anti-join picks out those columns in the left table that do not match the condition on the right table.
* **Difference:** In comparison to joins we learnt before, joins we learnt were to combine the tables. But now we are using other tables to give condition for our current left table.  
  **Syntax**

Finish the semi-join (an intro to subqueries)

SELECT president, country, continent  
FROM presidents  
WHERE country IN  
 (SELECT name  
 FROM states  
 WHERE indep\_year < 1800);

HERE’s the Anti-join

SELECT president,country,continent  
FROM presidents  
WHERE continent LIKE '%America'  
 AND country NOT IN  
 (SELECT name  
 FROM states  
 WHERE indep\_year < 1000);

### 50. Semi-join Exercises-retrives languages spoken in the Middle East

**Instructions** **1/3**  
You are now going to use the concept of a semi-join to identify languages spoken in the Middle East. Begin by selecting all country codes in the Middle East as a single field result using SELECT, FROM, and WHERE.

SELECT code  
FROM countries  
WHERE region = 'Middle East';

**Instructions**  
**2/3**  
\* Below the commented code, select only unique languages by name appearing in the languages table. \* Order the resulting single field table by name in ascending order.

SELECT DISTINCT name  
FROM languages  
ORDER BY name;

**Instructions**  
**3/3**  
\* Combine the previous two queries into one query by adding a WHERE IN statement to the SELECT DISTINCT query to determine the unique languages spoken in the Middle East.  
\* Order the result by name in ascending order.

SELECT DISTINCT name  
 FROM languages  
WHERE code in  
 (SELECT code  
 FROM countries  
 WHERE region = 'Middle East')  
ORDER BY NAME;

### 51. Relating semi-jin to a tweaked inner join.

**Using subqueries in semi-joins:**

SELECT DISTINCT name  
 FROM languages  
 WHERE code IN  
 (SELECT code  
 FROM countries  
 WHERE region = 'Middle East')  
 ORDER BY name;

**Using inner join to solve the same problem**

SELECT DISTINCT languages.name AS language  
FROM languages  
INNER JOIN countries  
ON languages.code = countries.code  
WHERE region = 'Middle East'  
ORDER BY language;

### 52. Diagnosing problems using anti-join

* Another powerful join in SQL is the anti-join. It is particularly useful in identifying which records are causing an incorrect number of records to appear in join queries.
* You will also see another example of a subquery here, as you saw in the first exercise on semi-joins. Your goal is to identify the currencies used in Oceanian countries!

**Instructions**  
**1/3**  
Begin by determining the number of countries in countries that are listed in Oceania using SELECT, FROM, and WHERE.

SELECT COUNT(code)  
FROM countries  
WHERE continent = 'Oceania';

**Instructions**  
**2/3**

* Complete an inner join with countries AS c1 on the left and currencies AS c2 on the right to get the different currencies used in the countries of Oceania.
* Match ON the code field in the two tables.
* Include the country code, country name, and basic\_unit AS currency.

Observe the query result and make note of how many different countries are listed here.

SELECT c1.code,c1.name,c2.basic\_unit AS currency  
FROM countries AS c1  
INNER JOIN currencies AS c2  
USING(code)  
WHERE continent = 'Oceania';

**Instructions**  
**3/3**  
Note that not all countries in Oceania were listed in the resulting inner join with currencies. Use an anti-join to determine which countries were not included!

* Use NOT IN and (SELECT code FROM currencies) as a subquery to get the country code and country name for the Oceanian countries that are not included in the currencies table.

SELECT code,name  
FROM countries   
WHERE continent = 'Oceania'  
 AND code NOT IN  
 (SELECT code   
 FROM currencies);

### 53. Set theory challenge

Your task here will be to incorporate two of UNION/UNION ALL/INTERSECT/EXCEPT to solve a challenge involving three tables. In addition, you will use a subquery as you have in the last two exercises!

**Instructions**  
\* Identify the country codes that are included in either economies or currencies but not in populations.  
\* Use that result to determine the names of cities in the countries that match the specification in the previous instruction.

SELECT c1.name  
FROM countries AS c1  
 WHERE code IN  
 (SELECT e.code  
 FROM economies AS e  
 AND  
 SELECT c2.code  
 FROM currencies AS c2  
 NOT  
 SELECT p.country\_code  
 FROM populations)

The above is wrong. To select, use union and except in the subquery.

-- Select the city name  
SELECT name  
 -- Alias the table where city name resides  
 FROM cities AS c1  
 -- Choose only records matching the result of multiple set theory clauses  
 WHERE country\_code IN  
(  
 -- Select appropriate field from economies AS e  
 SELECT e.code  
 FROM economies AS e  
 -- Get all additional (unique) values of the field from currencies AS c2   
 UNION  
 SELECT c2.code  
 FROM currencies AS c2  
 -- Exclude those appearing in populations AS p   
 EXCEPT  
 SELECT p.country\_code  
 FROM populations AS p  
);

## SECTION 4: Subqueries

### 54. Subqueries inside WHERE and SELECT clauses

**Asian countries below average fert\_rate**

SELECT name, fert\_rate  
FROM states  
WHERE continent = 'Asia'  
 AND fert\_rate <   
 (SELECT AVG(fert\_rate)  
 FROM states);

**Subqueries inside SELECT clausess - setup**

SELECT DISTINCT continent,  
 (SELECT COUNT(\*)  
 FROM states  
 WHERE prime\_ministers.continent = states.continent) AS countries  
FROM prime\_ministers;

### 55. Subquery inside where

You’ll now try to figure out which countries had high average life expectancies (at the country level) in 2015.

**Instructions** **1/2**  
Begin by calculating the average life expectancy across all countries for 2015.

SELECT AVG(life\_expectancy)  
FROM populations  
WHERE year = 2015;

**Instructions** **2/2**  
Recall that you can use SQL to do calculations for you. Suppose we wanted only records that were above 1.15 \* 100 in terms of life expectancy for 2015:

SELECT \*  
 FROM populations  
WHERE life\_expectancy > 1.15 \* 100  
 AND year = 2015;

Select all fields from populations with records corresponding to larger than 1.15 times the average you calculated in the first task for 2015. In other words, change the 100 in the example above with a subquery.

-- Select fields  
SELECT \*  
 -- From populations  
FROM populations  
-- Where life\_expectancy is greater than  
WHERE Life\_expectancy >1.15 \*   
(SELECT AVG(life\_expectancy)  
FROM populations  
WHERE year = 2015)   
AND year = 2015;

### 56. Subquery inside where (2)

Use your knowledge of subqueries in WHERE to get the urban area population for only capital cities.

**Instructions**

* Make use of the capital field in the countries table in your subquery.
* Select the city name, country code, and urban area population fields.

SELECT cities.name,countries.code,urbanarea\_pop  
FROM cities  
WHERE cities.name IN (  
SELECT countries.capital  
FROM countries  
)  
ORDER BY urbanarea\_pop DESC;

### 57. Subquery inside select

This exercise is using either join or subquery to write queries.

You have seen previously how to use GROUP BY with aggregate functions and an inner join to get summarized information from multiple tables.

The code given in the first query selects the top nine countries in terms of number of cities appearing in the cities` table. Recall that this corresponds to the most populous cities in the world. Your task will be to convert the second query to get the same result as the provided code.

**Initial code:**

SELECT countries.name AS country, COUNT(\*) AS cities\_num  
 FROM cities  
 INNER JOIN countries  
 ON countries.code = cities.country\_code  
GROUP BY country  
ORDER BY cities\_num DESC, country  
LIMIT 9;

**Transformation**

SELECT countries.name AS country, (  
SELECT COUNT(cities.name)  
FROM cities  
WHERE countries.code = cities.country\_code) AS cities\_num  
FROM countries  
ORDER BY cities\_num DESC, country  
LIMIT 9;

### 58. Subquery inside FROM clause

**Example**

SELECT DISTINCT monarchs.continent,subquery.max\_perc  
FROM monarchs,  
 (SELECT continent,MAX(women\_parli\_perc) AS max\_perc  
 FROM states  
 GROUP BY continent) AS subquery  
WHERE monarchs.continent = subquery.continent  
ORDER BY continent;

### 59. Subquery inside from

The last type of subquery you will work with is one inside of FROM.

You will use this to determine the number of languages spoken for each country, identified by the country’s local name! (Note this may be different than the name field and is stored in the local\_name field.)

**Instructions**  
**1/2**

* Begin by determining for each country code how many languages are listed in the languages table using SELECT, FROM, and GROUP BY.
* Alias the aggregated field as lang\_num.

My version here, which is wrong because I forgot to include code to match, where condition…

SELECT code,name, subquery.num AS lang\_num  
FROM languages,   
 (SELECT COUNT(name) AS num  
 FROM languages  
 GROUP BY code) AS subquery;

Correct version:

-- Select fields  
SELECT countries.local\_name,subquery.lang\_num  
 -- From countries  
 FROM countries,  
 -- Subquery (alias as subquery)  
 (SELECT code, COUNT(\*) AS lang\_num  
 FROM languages  
 GROUP BY code) AS subquery  
 -- Where codes match  
 WHERE countries.code = subquery.code  
-- Order by descending number of languages  
 ORDER BY lang\_num DESC;

### 60. Advanced Subquery

You can also nest multiple subqueries to answer even more specific questions.

name | continent

————- | —————-  
 | North America  
 | Africa  
 | Oceania  
 | Europe

### Summary: Differences Between Union and Joins

**Join**

* Join combines data from many tables based on a matched condition between them.
* It combines data into new columns.
* Number of columns selected from each table may not be the same.
* Datatypes of corresponding columns selected from each table can be different.
* It may not return distinct columns.

**Union**

* SQL combines the result-set of two or more SELECT statements.
* It combines data into new rows.
* Number of columns selected from each table should be same.
* Datatypes of corresponding columns selected from each table should be same.
* It returns distinct rows.