



Universidad del
Rosario

Educación
Continua

SQL Parte 2

2019

Sesión # 2

Lenguaje de manipulación de datos



Miguel Angel Orjuela Rocha

Ingeniero de Sistemas y Computación

Contenido

- Actualizar la base de datos con INSERT (Parte 1)
- Recuperar datos con SELECT
- Cómo construir sentencias SQL que:
 - use la cláusula WHERE para recuperar las filas que satisfacen varias condiciones;
 - ordene los resultados de la consulta usando ORDER BY;
 - utilizar las funciones agregadas de SQL;
 - agrupar datos utilizando GROUP BY;
 - usar subconsultas;
 - unir tablas
 - realizar operaciones de ajuste (UNION, INTERSECT, EXCEPT).
- Actualizar la base de datos con INSERT, UPDATE y DELETE (Parte 2).

Actualizar la base de datos con INSERT (Parte 1)

Actualizar la base de datos con INSERT

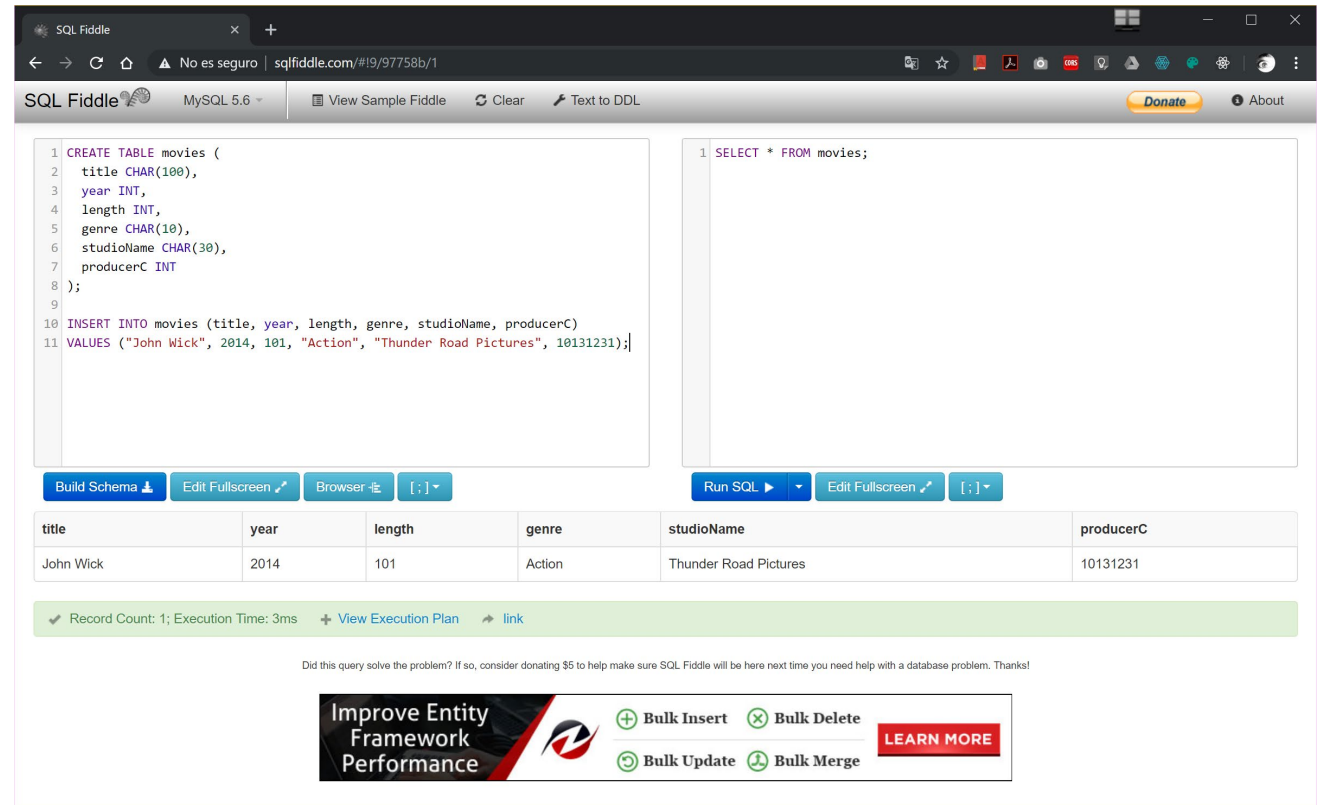
Para insertar una columna se usa el commando **INSERT INTO**, de acuerdo a la siguiente sintaxis:

```
INSERT INTO nombreTabla (col1, col2, ... )  
VALUES (valor1, valor2,...);
```

- **nombreTabla**: Tabla donde se agregará el registro
- **col1, col2, ...** : Nombre de las columnas de la table
- **valor1, valor2, ...** : Valores que irán en la nueva tupla

Actualizar la base de datos con INSERT

```
CREATE TABLE movies (  
  title CHAR(100),  
  year INT,  
  length INT,  
  genre CHAR(10),  
  studioName CHAR(30),  
  producerC INT  
);
```



The screenshot shows the SQL Fiddle web application. The left pane contains the following SQL code:

```
1 CREATE TABLE movies (  
2   title CHAR(100),  
3   year INT,  
4   length INT,  
5   genre CHAR(10),  
6   studioName CHAR(30),  
7   producerC INT  
8 );  
9  
10 INSERT INTO movies (title, year, length, genre, studioName, producerC)  
11 VALUES ("John Wick", 2014, 101, "Action", "Thunder Road Pictures", 10131231);
```

The right pane contains the query:

```
1 SELECT * FROM movies;
```

Below the code panes, the "Run SQL" button has been clicked. A table displays the result of the query:

title	year	length	genre	studioName	producerC
John Wick	2014	101	Action	Thunder Road Pictures	10131231

A green status bar at the bottom indicates: "Record Count: 1; Execution Time: 3ms". Below this, a small text prompt asks if the query solved the problem, followed by a "Learn More" button for a service that improves entity framework performance.

```
INSERT INTO movies (title, year, length, genre, studioName, producerC)  
VALUES ("John Wick", 2014, 101, "Action", "Thunder Road Pictures", 10131231);
```

Recuperar datos con SELECT

Recuperar datos con SELECT

Tenemos una table llamada movies, recuperemos todos los campos de todas las películas

```
SELECT *  
FROM movies;
```


Conjunto de datos

1

Arquitectura del sistema

Conexión a una base de datos externa con cliente R



Configuración de RStudio como cliente

Pasos a seguir:

- Instalar y cargar paquete **RMySQL**
- Crear una conexión
- Seleccionar base de datos
- Hacer consultas y almacenar sus resultados

Información a consultar



Configuración de RStudio como cliente

- Instalar y cargar paquete RMySQL

```
# Librería para conexión con BD MySQL
# install.packages("RMySQL")
library(RMySQL)
```

- Crear una conexión

```
# Cadena de conexión
conexion <- dbConnect(MySQL(),
                      user="analytics-user",
                      password="analytics-user",
                      host="52.38.6.74",
                      port=3306,
                      dbname='')

```

- Seleccionar base de datos

```
# Función para seleccionar la base de datos
dbSendQuery(
  conexion,
  "USE world;"
)
```

- Hacer consultas y almacenar sus resultados

```
# Consulta de ejemplo: Retorna todos los campos de la tabla country
países <- dbGetQuery(conexion, "SELECT * FROM country")
```

Recuperar datos con SELECT

Recuperar datos con SELECT

Recuperar todas las filas y todas las columnas

```
SELECT  
  Code,  
  Name,  
  Continent,  
  Region,  
  SurfaceArea,  
  IndepYear,  
  Population,  
  LifeExpectancy,  
  GNP,  
  GNPOld,  
  LocalName,  
  GovernmentForm,  
  HeadOfState,  
  Capital,  
  Code2  
FROM  
  country;
```



```
SELECT *  
FROM country;
```

Recuperar datos con SELECT

Recuperar todas las filas y columnas específicas

```
SELECT  
  Code,  
  Name,  
  Continent,  
  Population,  
  LifeExpectancy  
FROM  
  country;
```



Recuperar datos con SELECT

Uso del **DISTINCT**

```
SELECT Continent  
FROM country;
```

Entrega todos los resultados, así tengan columnas repetidas

Elimina los repetidos

```
SELECT DISTINCT Continent  
FROM country;
```


Recuperar datos con SELECT

Columnas calculadas/derivadas

- Podemos realizar operaciones en una columna
- La columna toma el nombre de la operación (por defecto)

```
SELECT Name, SurfaceArea, Population, Population/SurfaceArea  
FROM country;
```

- Podemos poner un *alias* al nombre de la columna

```
SELECT Name, SurfaceArea, Population, Population/SurfaceArea AS HabsPorKm2  
FROM country;
```

La clausula
WHERE

La cláusula WHERE

```
SELECT Name, Population  
FROM city  
WHERE CountryCode = 'COL';
```



Condiciones/Predicados básicos:

- **Comparación:** Compara el valor de una expresión con el valor de otra expresión
- Prueba de **rango:** si el valor de una expresión cae dentro de un determinado rango de valores
- Establecer **pertenencia:** Probar si el valor de una expresión es igual a uno de un conjunto de valores
- Coincidencia de **patrón:** Comprueba si una cadena coincide con un patrón específico.
- **Null:** Comprueba si una columna tiene un valor nulo (desconocido).

La cláusula WHERE

Operadores de comparación:

COMPARISON OPERATORS	
SYMBOL	MEANING
=	Equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
<> or !=	Not equal to

```
SELECT
  Name,
  Continent,
  Population
FROM
  country
WHERE
  Population < 10000;
```

¿Qué países tienen más de 1.000.000.000 habitantes?

La cláusula WHERE

Se pueden generar predicados más complejos utilizando los operadores lógicos **AND**, **OR** y **NOT**, con paréntesis (si es necesario o se desea)

Las reglas para evaluar una expresión condicional son:

- Una expresión se evalúa de izquierda a derecha
- Las subexpresiones entre paréntesis se evalúan primero
- Los **NOT** se evalúan antes de los **AND** y los **OR**
- Los **AND** se evalúan antes que los **OR**

Siempre se recomienda el uso de paréntesis, para eliminar cualquier posible ambigüedad.

```
SELECT
    Name, Continent, Population
FROM
    country
WHERE
    Population < 10000 AND
    Continent <> 'Oceania' AND
    Continent <> 'Antarctica' ;
```

La cláusula WHERE

Rangos con **BETWEEN/NOT BETWEEN**

- Incluye los valores extremos

```
SELECT
    Name, Continent, SurfaceArea
FROM
    country
WHERE
    SurfaceArea NOT BETWEEN 15 AND 8000000;
```

La cláusula WHERE

Comparación de un valor en una lista de valores con **IN/NOT IN**

```
SELECT
    Name, Continent
FROM
    country
WHERE
    Continent IN ('Oceania', 'Antarctica');
```

La cláusula WHERE

SQL tiene dos símbolos especiales para buscar por coincidencia de patrón:

- **%** Representa una secuencia de cero o más caracteres (*comodin*)
- **_** Representa un solo character
- Se escapa con **#**

Que comiencen por Flor

```
SELECT Name
FROM city
WHERE CountryCode = 'COL' AND
      Name LIKE 'Flor%';
```

Que comiencen por Flor y tenga cinco caracteres después

```
SELECT Name
FROM city
WHERE CountryCode = 'COL' AND
      Name LIKE 'Flor_____';
```

Que terminen en ra

```
SELECT Name
FROM city
WHERE CountryCode = 'COL' AND
      Name LIKE '%ra';
```

Que tenga cinco caracteres después y termine en ra

```
SELECT Name
FROM city
WHERE CountryCode = 'COL' AND
      Name LIKE '_____ra';
```


La clausula **WHERE**

¿Cuántas ciudades en el mundo incian con la palabra **Santa**?

¿A qué países corresponden las ciudades?

¿Podríamos ver el resultado como un listado con nombres completos de países y sus respectivas ciudades ciudades?

La clausula WHERE

Un valor nulo (**nu1l**) es un valor desconocido, por lo que no se puede comparar con un string con = o <>

Para comparar con nulo se usa las palabras clave

IS NULL/IS NOT NULL

```
SELECT *  
FROM country  
WHERE LifeExpectancy IS NULL;
```

**Ordenando los
resultados con
ORDER BY**

Ordenando los resultados con ORDER BY

ORDER BY puede ordenar el resultado de la consulta en orden ascendente (**ASC**) o descendente (**DESC**)

```
SELECT *  
FROM country  
ORDER BY SurfaceArea DESC;
```

Se puede ordenar usando varias columnas

```
SELECT *  
FROM country  
ORDER BY Continent, Region, SurfaceArea DESC;
```

Agregación Resumen

Agregación/Resumen

Funciones de agregación/resumen. El estándar ISO define cinco funciones:

COUNT: devuelve el número de valores en una columna especificada

SUM: devuelve la suma de los valores en una columna específica

AVG: devuelve el promedio de los valores en una columna específica

MIN: devuelve el valor más pequeño en una columna específica

MAX: devuelve el mayor valor en una columna específica

Agregación/Resumen

COUNT: devuelve el número de valores en una columna especificada

SUM: devuelve la suma de los valores en una columna específica

AVG: devuelve el promedio de los valores en una columna específica

MIN: devuelve el valor más pequeño en una columna específica

MAX: devuelve el mayor valor en una columna específica

- Operan en una sola columna de la tabla
- Retornan un solo valor
- **COUNT**, **MIN**, **MAX** funciona con campos numéricos y no numéricos
- **MIN**, **MAX** solo funciona con campos numéricos
- Las funciones eliminan los nulos antes de operar (menos **COUNT (*)**)
- **COUNT (*)** es especial, cuenta todas las columnas de la tabla
- Se pueden eliminar duplicados antes de contar usando **DISTINCT** (no funciona para **MIN** ni **MAX**)
- Se pueden usar solo en la lista del **SELECT** o en la cláusula **HAVING**

Agregación/Resumen

¿Cuántas ciudades tiene Colombia?

```
SELECT COUNT(*)  
FROM city  
WHERE CountryCode = 'COL';
```

¿Cuántos países superan el promedio de Producto Interno Bruto?

¿Cuáles son?

```
SELECT AVG(GNP)  
FROM country;
```



```
SELECT COUNT(*)  
FROM country  
WHERE GNP > 122823.882427;
```


Recuperar datos con SELECT

¿Cuántos continentes hay?

```
SELECT COUNT(DISTINCT Continent)
FROM country;
```

Varias agregaciones sobre las ciudades de Colombia

```
SELECT
    COUNT(Name) AS NumCiudades,
    SUM(Population) AS PoblacionTotal
FROM city
WHERE CountryCode = 'COL';
```

```
SELECT
    MIN(Population) AS MinPoblacion,
    AVG(Population) AS PromPoblacion,
    MAX(Population) AS MaxPoblacion
FROM city WHERE CountryCode = 'COL';
```

Agrupaciones en resúmenes

Agrupaciones en resúmenes

Agrupaciones/Aggregaciones

Los resultados anteriores son como las filas de totales

Podemos hacer subtotales, agrupar (**GROUP BY**) por alguna de las columnas seleccionadas en el **SELECT**

Tienen que tener valor único por grupo
Para que funcionen, en el **SELECT** solo pueden haber nombres de columnas, agregaciones, o constantes

Columnas de agrupación

Toda columna del **SELECT** debe estar acá, a menos que sea agregación

```
SELECT
  District,
  MIN(Population) AS MinPoblacion,
  AVG(Population) AS PromPoblacion,
  MAX(Population) AS MaxPoblacion
FROM city
WHERE CountryCode = 'COL'
GROUP BY District
ORDER BY District;
```

Agrupaciones en resúmenes: CONSULTAS ANIDADAS

El estándar permite que en el **SELECT** hayan consultas anidadas

VS.

Agrupaciones

```
SELECT
  DISTINCT city_o.District District,
  { ( SELECT MIN(Population) FROM city city_i
    WHERE city_i.CountryCode = 'COL' AND
      city_i.District = city_o.District) AS MinPoblacion,
  { ( SELECT AVG(Population) FROM city city_i
    WHERE city_i.CountryCode = 'COL' AND
      city_i.District = city_o.District) AS PromPoblacion,
  { ( SELECT MAX(Population)
    FROM city city_i
    WHERE city_i.CountryCode = 'COL' AND
      city_i.District = city_o.District) AS MaxPoblacion
FROM city city_o
WHERE city_o.CountryCode = 'COL'
GROUP BY city_o.District
ORDER BY city_o.District ASC;
```

```
SELECT
  District,
  AVG(Population) AS PromPoblacion
FROM city
WHERE CountryCode = 'COL'
GROUP BY District
ORDER BY District;
```

Agrupaciones en resúmenes

```
# Consulta agregada en R
ciudades_col <- dbGetQuery(conexion, "SELECT * FROM city WHERE CountryCode = 'COL'");

library(dplyr)
ciudades_col %>%
  group_by(District) %>%
  summarise(
    MinPoblacion = min(Population),
    PromPoblacion = mean(Population),
    MaxPoblacion = max(Population)) %>%
  View()
```

```
SELECT
  DISTINCT city_o.District District,
  ( SELECT MIN(Population) FROM city city_i
    WHERE city_i.CountryCode = 'COL' AND
          city_i.District = city_o.District) AS MinPoblacion,
  ( SELECT AVG(Population) FROM city city_i
    WHERE city_i.CountryCode = 'COL' AND
          city_i.District = city_o.District) AS PromPoblacion,
  ( SELECT MAX(Population)
    FROM city city_i
    WHERE city_i.CountryCode = 'COL' AND
          city_i.District = city_o.District) AS MaxPoblacion
FROM city city_o
WHERE city_o.CountryCode = 'COL'
GROUP BY city_o.District
ORDER BY city_o.District ASC;
```

```
SELECT
  District,
  MIN(Population) AS MinPoblacion,
  AVG(Population) AS PromPoblacion,
  MAX(Population) AS MaxPoblacion
FROM city
WHERE CountryCode = 'COL'
GROUP BY District
ORDER BY District;
```

Agrupaciones en resúmenes

Restringir los grupos con **HAVING**

Filtra por resultados de las agregaciones

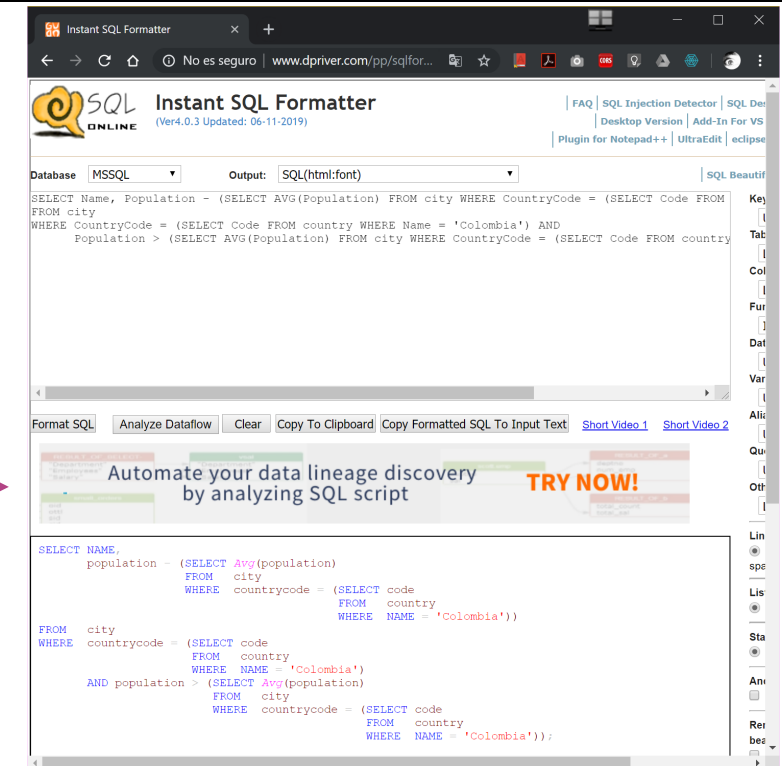
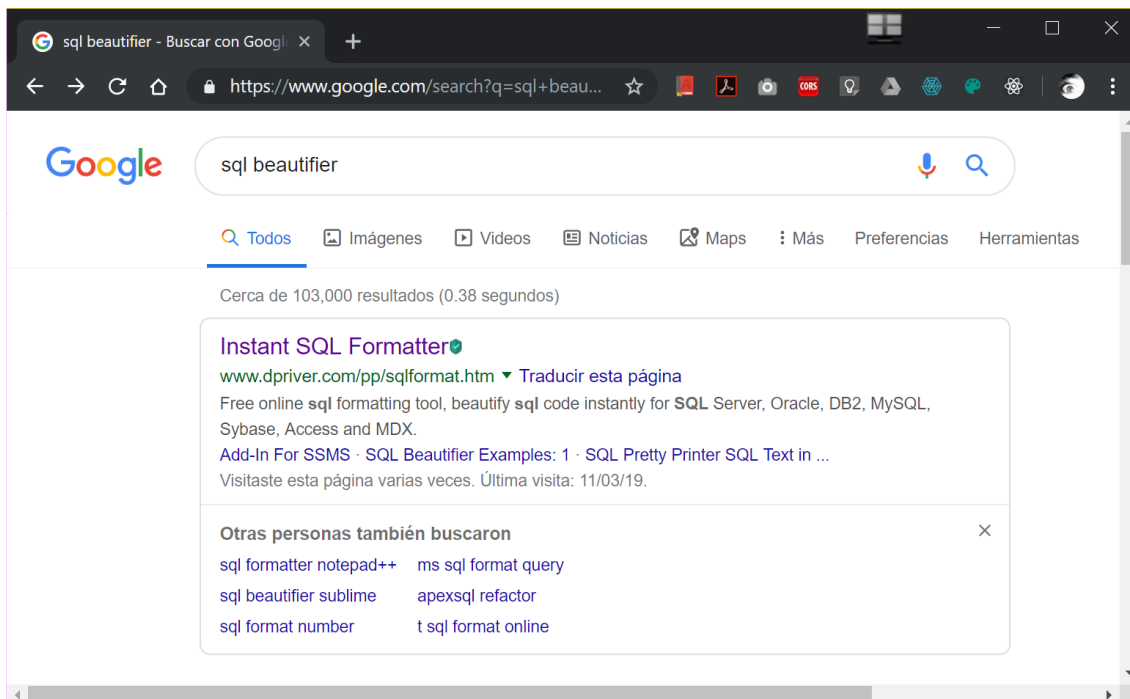
```
SELECT
    District,
    COUNT(Population) AS CntPoblacion,
    MIN(Population) AS MinPoblacion,
    AVG(Population) AS PromPoblacion,
    MAX(Population) AS MaxPoblacion
FROM city
WHERE CountryCode = 'COL'
GROUP BY District
HAVING COUNT(Population) > 1
ORDER BY COUNT(Population) DESC;
```

Subconsultas

Subconsultas

Subconsultas: Es una sentencia **SELECT** embebida dentro de otra sentencia **SELECT**. El resultado de la consulta **interna** determina el contenido del resultado final de la consulta

```
SELECT Name, Population - (SELECT AVG(Population) FROM city WHERE CountryCode = (SELECT Code FROM country WHERE Name = 'Colombia'))
FROM city
WHERE CountryCode = (SELECT Code FROM country WHERE Name = 'Colombia') AND
      Population > (SELECT AVG(Population) FROM city WHERE CountryCode = (SELECT Code FROM country WHERE Name = 'Colombia'));
```



Subconsultas

Subconsultas: Es una sentencia **SELECT** embebida dentro de otra sentencia **SELECT**. El resultado de la consulta **interna** determina el contenido del resultado final de la consulta

[illegible]

Subconsultas

Hay tres tipos de subconsultas:

- **Escalares:** Devuelven una columna con un solo valor
- De **fila:** Devuelve muchas columnas pero una sola fila
- De **tabla:** Múltiples filas y columnas en la salida

Subconsultas

Uso de **ANY/SOME**

Por lo menos uno que cumpla la condición en el grupo

```
SELECT District, Name, Population
FROM city
WHERE CountryCode = 'COL' AND District = 'Antioquia';
```

¿Qué resultado esperaríamos si se cambia el signo > por <?

```
SELECT District, Name, Population
FROM city
WHERE CountryCode = 'COL' AND
      Population > SOME ( SELECT Population
                          FROM city
                          WHERE CountryCode = 'COL' AND District = 'Antioquia')
ORDER BY Population DESC;
```

Subconsultas

Uso de **ALL**

Todos deben cumplir la condición en el grupo

```
SELECT District, Name, Population
FROM city
WHERE CountryCode = 'COL' AND District = 'Antioquia';
```

¿Qué resultado esperaríamos si se cambia el signo > por <?

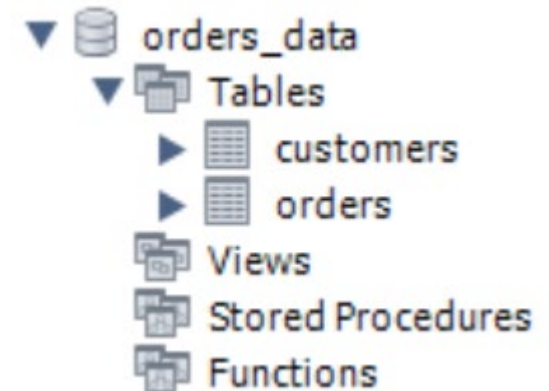
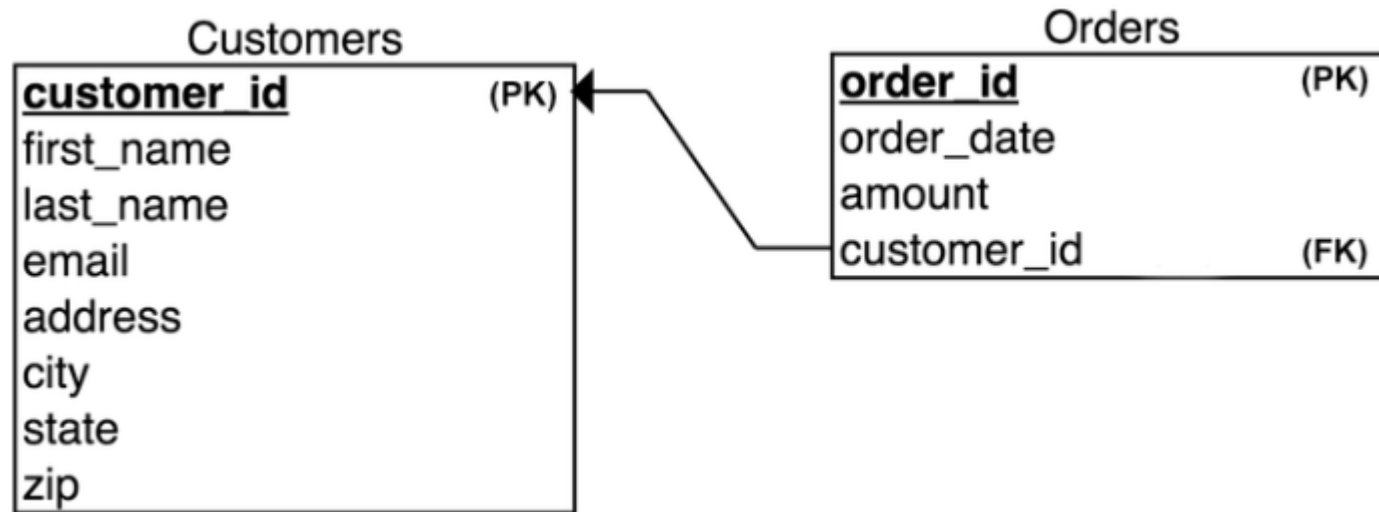
```
SELECT District, Name, Population
FROM city
WHERE CountryCode = 'COL' AND
      Population > ALL ( SELECT Population
                        FROM city
                        WHERE CountryCode = 'COL' AND District = 'Antioquia')
ORDER BY Population DESC;
```

Consultas en varias tablas

Consultas en varias tablas

Consultas multi-tabla

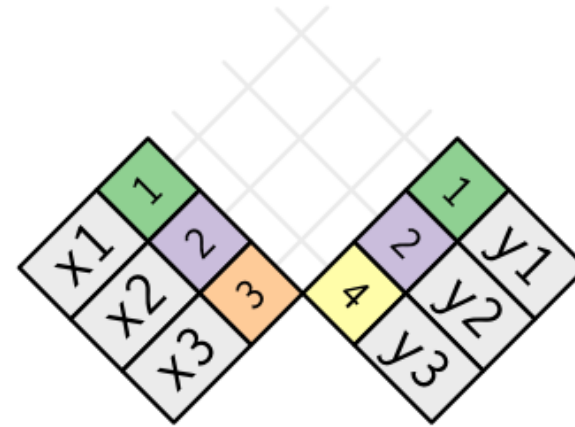
JOIN: Instrucción que combina datos de dos conjuntos de datos en una sola tabla



Consultas en varias tablas

Producto cartesiano: Combina todas las filas de una tabla con las filas de otra tabla (arma todos los posibles pares)

x		y	
1	x1	1	y1
2	x2	2	y2
3	x3	4	y3



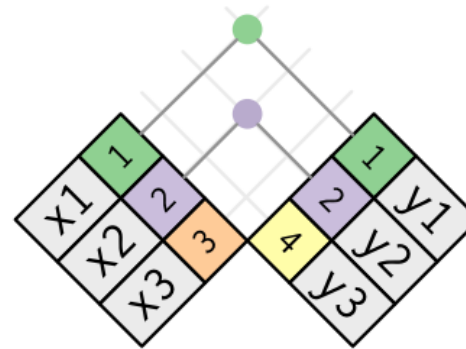
```
SELECT x.*, y.*  
FROM tabla1 x, tabla2 y;
```

¿Cuántos resultados tendrá el producto cartesiano de **customers** y **orders**?

Consultas en varias tablas

JOIN simple: Combina todas las filas de una tabla con las filas de otra table, y deja los resultados donde las llaves de ambas tablas corresponden

x		y	
1	x1	1	y1
2	x2	2	y2
3	x3	4	y3



key	val_x	val_y
1	x1	y1
2	x2	y2

```
SELECT x.*, y.*  
FROM tabla1 x, tabla2 y  
WHERE x.id = y.id;
```

=

```
SELECT x.*, y.*  
FROM tabla1 x JOIN tabla2 y ON x.id = y.id;
```

```
SELECT x.*, y.*  
FROM tabla1 x JOIN tabla2 y USING id = id;
```

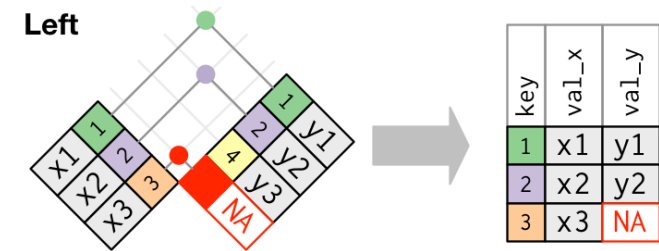
¿Cuántos resultados tendrá la union simple de customers y orders?

Consultas en varias tablas

OUTER JOIN : Es como un join simple, pero retiene las filas que no satisfacen la condición. Tiene tres tipos

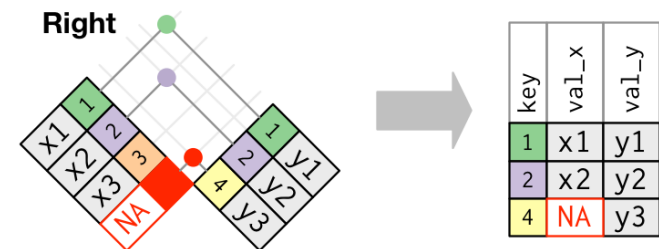
▪ LEFT

```
SELECT x.*, y.*  
FROM tabla1 x LEFT OUTER JOIN  
      tabla2 y ON x.id = y.id;
```



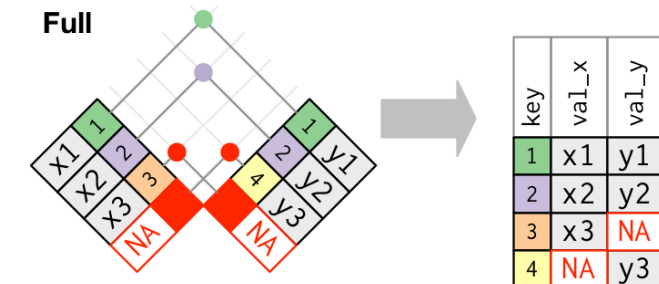
▪ RIGHT

```
SELECT x.*, y.*  
FROM tabla1 x RIGHT OUTER JOIN  
      tabla2 y ON x.id = y.id;
```



▪ FULL

```
SELECT x.*, y.*  
FROM tabla1 x FULL OUTER JOIN  
      tabla2 y ON x.id = y.id;
```

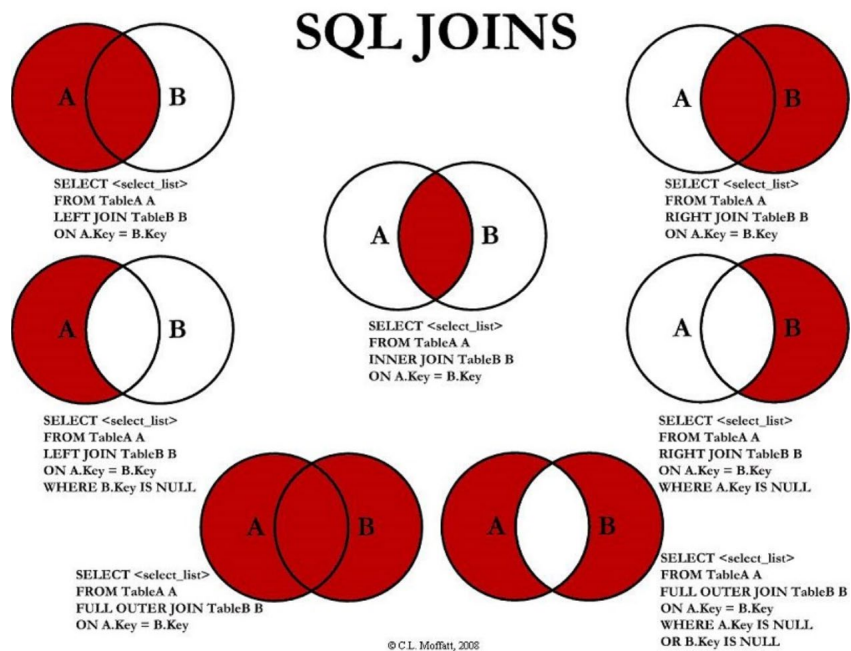


Consultas en varias tablas

Genere la union de las tablas de `customers` y `orders` usando **LEFT**, **RIGHT** y **FULL OUTER JOIN**, ¿cuántos registros tiene cada tabla?

Consultas en varias tablas

MySQL no tiene **FULL OUTER JOIN**, hay que simularlo



```
SELECT *  
FROM customers cus LEFT OUTER JOIN  
orders ord ON cus.customer_id = ord.customer_id  
  
UNION  
SELECT *  
FROM customers cus RIGHT OUTER JOIN  
orders ord ON cus.customer_id = ord.customer_id;
```

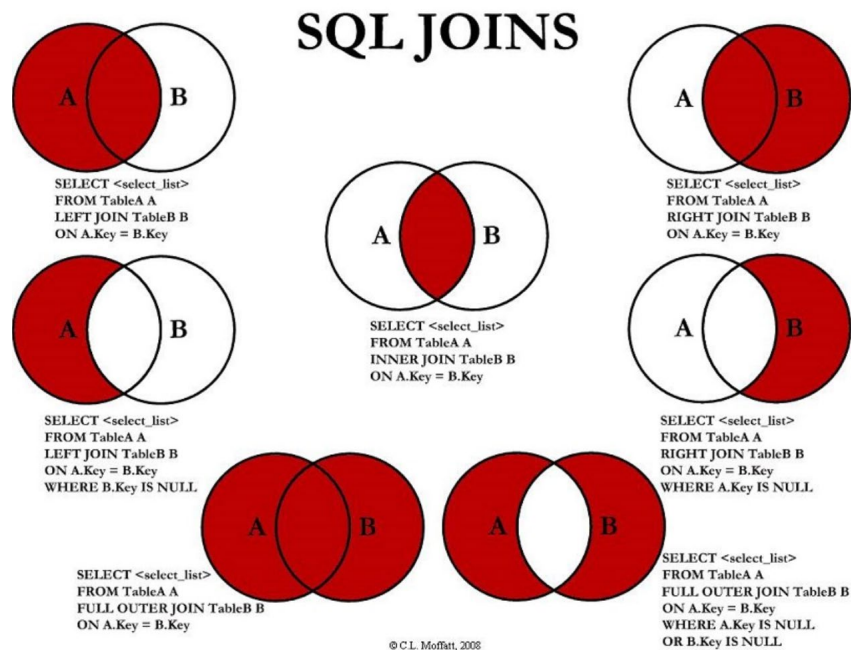
```
SELECT *  
FROM customers cus LEFT OUTER JOIN  
orders ord ON cus.customer_id = ord.customer_id  
  
UNION ALL  
SELECT *  
FROM customers cus RIGHT OUTER JOIN  
orders ord ON cus.customer_id = ord.customer_id;
```

¿Hay diferencia entre **UNION** y **UNION ALL**?

Consultas en varias tablas

Combinando tablas de resultados con **UNION**, **INTERSECT** y **EXCEPT**

Las tablas a unir deben tener el mismo número de columnas



```
SELECT *  
FROM customers cus LEFT OUTER JOIN  
orders ord ON cus.customer_id = ord.customer_id  
  
UNION  
SELECT *  
FROM customers cus RIGHT OUTER JOIN  
orders ord ON cus.customer_id = ord.customer_id;
```

```
SELECT *  
FROM customers cus LEFT OUTER JOIN  
orders ord ON cus.customer_id = ord.customer_id  
  
UNION ALL  
SELECT *  
FROM customers cus RIGHT OUTER JOIN  
orders ord ON cus.customer_id = ord.customer_id;
```

Agregar **ALL** conserva las filas repetidas

Tablas múltiples y subconsultas

Tablas múltiples y subconsultas

Las palabras **EXISTS** y **NOT EXISTS** están diseñadas solo para subconsultas

Retornan **TRUE** si la subconsulta tiene resultados o **FALSE** si es vacía

```
SELECT *  
FROM customers cus  
WHERE EXISTS (    SELECT *  
                  FROM orders ord  
                  WHERE ord.customer_id = cus.customer_id);
```

Actualizar la base de datos con **INSERT** (Parte 2)

Actualizar la base de datos con INSERT

INSERT sencillo

```
INSERT INTO orders
  (order_id, order_date, amount, customer_id)
VALUES (34, '03/14/1760', 45.6, 1);
```

INSERT con tabla como parámetro

```
INSERT INTO orders
  (order_id,
   order_date,
   amount,
   customer_id)
VALUES
  (SELECT * FROM orders WHERE order_id = 1);
```


La sentencia

UPDATE

La sentencia UPDATE

Actualizar todas las filas

```
UPDATE orders  
SET amount = amount * 1.10;
```

Actualizar todas las filas que correspondan con la clausula **WHERE**

```
UPDATE orders  
SET amount = amount * 1.10  
WHERE customer_id = 1;
```

Actualizar columnas múltiples

```
UPDATE orders  
SET amount = amount * 1.10, order_date = REPLACE(order_date, '/', '-')  
WHERE customer_id = 1;
```

**La sentencia
DELETE**

La sentencia UPDATE

Borra todas las filas

```
DELETE FROM orders;
```

Borra todas las filas que correspondan con la clausula **WHERE**

```
DELETE FROM orders  
WHERE customer_id = 1;
```