

# Lenguaje SQL

## Índice

<b>Introducción</b>	<b>4</b>
<b>Lecturas y visualizaciones</b>	<b>4</b>
1. Historia de SQL	4
2. El estándar ANSI SQL	5
3. Funciones del SGBD	5
4. ¿Qué debe permitir un Lenguaje de Base de Datos?	6
5. Características del Lenguaje SQL	7
6. Buenas prácticas de SQL	7
7. Bases de datos relacionales	7
8. Sentencias del Lenguaje SQL	8
8.1 Sentencias DDL. Lenguaje de Definición de Datos.	9
8.2 Sentencias DML. Lenguaje de Manipulación de Datos.	9
8.3 Sentencias DQL. Lenguaje de Consulta de Datos.	9
8.4 Sentencias DCL. Lenguaje de Control de Datos.	9
8.5 Sentencias TCL. Lenguaje de Control Transaccional.	9
<b>Caso de estudio. ONG Médicos sin Fronteras</b>	<b>10</b>
<b>Parte 1.</b>	<b>11</b>
1. Instalar y conectarse al servidor del SGBD MySQL Server	11
Ejercicio 1.	11
2. Lenguaje de Definición de Datos	11
2.1 Crear o eliminar una base de datos	11
Ejercicio 2.	12
2.2 Creación de las tablas	13
Ejercicio 3.	15
2.3. Borrar una tabla	16
2.4. Modificar una tabla	16
Ejercicio 4.	17
Ejercicio 5.	18
3. Lenguaje de Manipulación de Datos	19

3.1 Insertar datos	19
3.2 Modificar datos	20
3.3 Borrar datos	22
<b>Parte 2.</b>	<b>24</b>
4. Lenguaje de Consulta de Datos	24
4.1 Estructura de una consulta	24
4.2 Consultas sencillas	25
4.3 Consultas con operadores de comparación	27
4.4 Operadores lógicos	29
4.4.1 Operador AND	30
4.4.2 Operador BETWEEN	31
4.4.3 Cláusula DISTINCT	34
4.4.4 Consultas con valores NULOS	35
4.4.5 Cláusula ORDER BY	37
4.4.6 Operadores IN y NOT IN	38
4.5 Operador LIKE	40
4.6 Operaciones válidas en el SELECT	43
4.7 Cláusula JOIN	45
4.7.1 Cláusula CROSS JOIN	45
4.7.2 Productos naturales	46
4.7.2.1 Cláusula NATURAL JOIN	47
4.7.2.2 Cláusula INNER JOIN	48
4.8 Operaciones de conjuntos	49
4.8.1 Unión	49
4.8.2 Intersección	50
<b>Bibliografía</b>	<b>51</b>



Este documento está disponible en acceso abierto bajo la licencia  
Atribución-NoComercial-CompartirIgual 2.5 Argentina (CC-BY-NC-SA 2.5 AR DEED)  
(<https://creativecommons.org/licenses/by-nc-sa/2.5/ar/>)

**Autora: Mariana Adó.**

## Introducción

El presente documento tiene por finalidad abordar conceptos vinculados con el **Lenguaje de Consulta Estructurado, SQL** (por sus siglas en inglés Structured Query Language), el cual provee distintas sentencias que nos permiten crear, borrar, modificar y manipular los objetos y los datos de una base de datos.

## Lecturas y visualizaciones

A continuación, se detalla una **guía de lectura y visualización de los recursos y videos** que complementan este documento, así como también **ejercicios prácticos**, que podrás realizar para ejercitar los conceptos abordados.

Se pone a disposición información en distintos formatos, con el fin de atender a las distintas formas de acercarse al conocimiento por parte del estudiantado. Te invito a que veas todo el material para un mejor aprendizaje, ya que todos los recursos se complementan entre sí.

### 1. Historia de SQL

El origen de SQL está ligado a las bases de datos relacionales. En 1970 E. F. Codd propone el modelo de datos relacional y asociado a éste, lenguajes de consulta procedural como el álgebra relacional y no procedural como el cálculo de tuplas o el cálculo de dominios. Estos lenguajes propuestos por Codd fueron la base para la creación de SQL (Structured Query Language, Lenguaje de Consultas Estructurado).

SQL termina siendo una de las principales razones del éxito de las bases de datos relacionales. Esto se debió a que la mayoría de los desarrolladores de los Sistemas Gestores de Bases de Datos (SGBD) optaron por SQL como el lenguaje de trabajo, generando de esta forma un estándar respetado en general, por la mayor parte de los productos de mercado. Así, una consulta SQL puede migrar de producto sin necesitar mayores cambios para ser ejecutada.


## 2. El estándar ANSI SQL

En 1986 SQL es estandarizado por el ANSI (American National Standard Institute, la organización de estándares de Estados Unidos), dando lugar a la primera versión estándar de este lenguaje, el “SQL-86” o “SQL1”. Al año siguiente este estándar fue adoptado por la ISO (International Standard Organization).

En 1992 se amplía la definición del estándar para cubrir mayores necesidades de los desarrolladores. Surge de esta forma el SQL92 o el SQL2.

El estándar tuvo varias modificaciones posteriores: en 1999 se creó el SQL2000, al que se le incorporaron expresiones regulares, consultas recursivas y características orientadas a objetos. En 2003, surge SQL3 que agrega características de XML y, posteriormente en 2006 ISO define ISO/IEC 9075-14:2006 con características que acercan a SQL al mundo W3C.

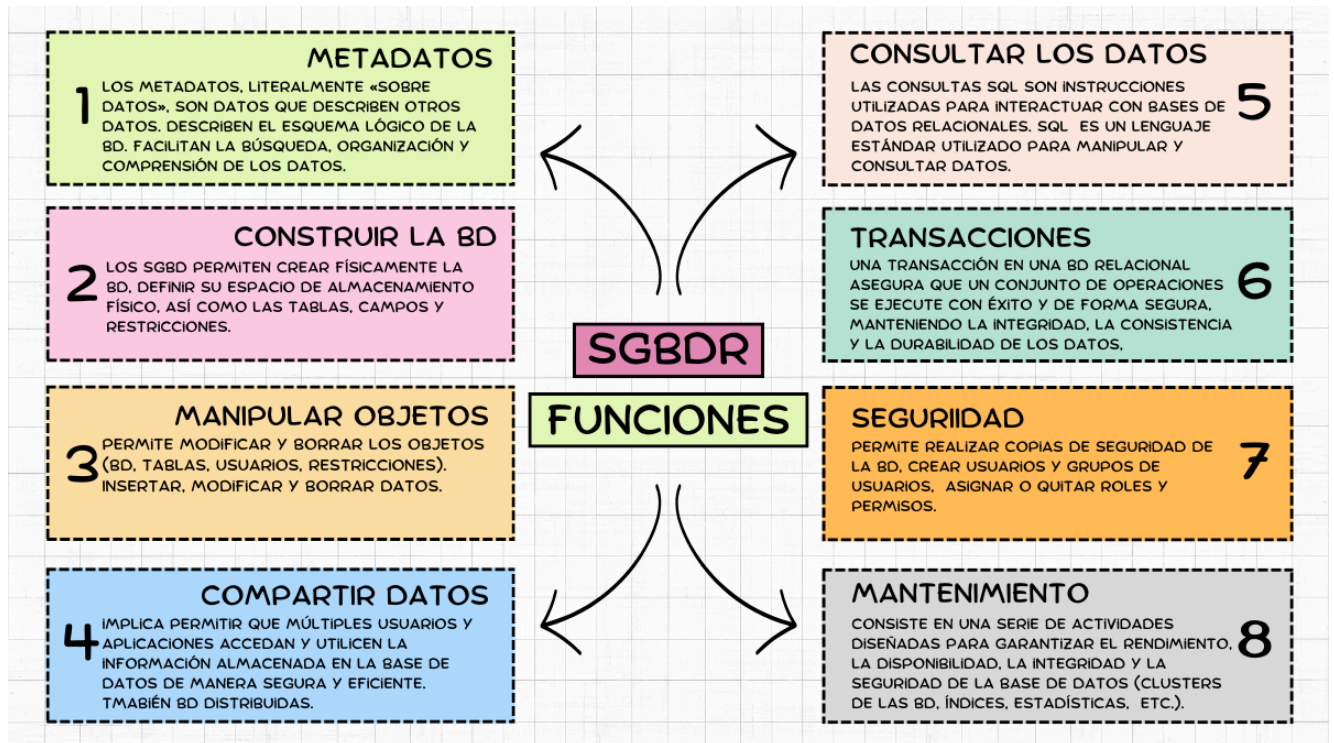
En este documento abordaremos el **estándar ANSI SQL92**, cuyas características son suficientes para un curso introductorio del lenguaje.

	<p><b>Visualizar la píldora educativa 1 titulada:</b>  <i>¿Qué es SQL? en 5 MINUTOS   STRUCTURED QUERY LANGUAGE.</i>          Autor: @Tecnobinaria - YouTube  <a href="#">Link al video</a></p>
---	---

## 3. Funciones del SGBD

Todo SGBD tiene distintas funciones que están relacionadas con la manipulación de los datos, la creación de los objetos, el mantenimiento de las bases de datos y de los datos, la seguridad, entre otras.

En el siguiente mapa conceptual, se puede observar algunas de las **funciones principales para los SGBD**.




#### 4. ¿Qué debe permitir un Lenguaje de Base de Datos?

- Crear estructuras de datos.
- Realizar tareas básicas de administración de datos (agregar, eliminar y modificar).
- Efectuar consultas complejas.
- Realizar las tareas básicas con mínimo esfuerzo y ser fácil de aprender.
- Que sea portable.

#### 5. Características del Lenguaje SQL

- Es un Lenguaje Estructurado de Consultas.
- Es un lenguaje declarativo, sólo se tiene que decir qué se quiere hacer. En cambio, en los lenguajes procedimentales hay que especificar cómo se tiene que hacer cualquier cosa sobre la base de datos.
- Hay **Independencia Física** de los Datos.

	<p><b>Visualizar la píldora educativa 2 titulada:</b>  <i>Administración de Bases de Datos - Introducción Conceptos Fundamentales (II) - Andrés Muñoz</i></p> <p><b>Tema: Independencia física y lógica de los datos</b>          Autor: UCAM Universidad Católica de Murcia @portalucam- YouTube  <a href="#">Link al video</a></p>
---	--

## 6. Buenas prácticas de SQL

- Las sentencias de SQL se pueden escribir tanto en mayúsculas como en minúsculas y lo mismo sucede con los nombres de las tablas y de las columnas, o cualquier objeto.
- Una buena práctica es usar **mayúsculas** para las **palabras clave del lenguaje** y **minúsculas** para los **nombres de los objetos**, con notación **snake\_case**.
- Las **sentencias** de SQL **terminan** siempre con el carácter **punto y coma (;)**.

## 7. Bases de datos relacionales

Las **bases de datos relacionales** se basan en la teoría del modelo relacional.

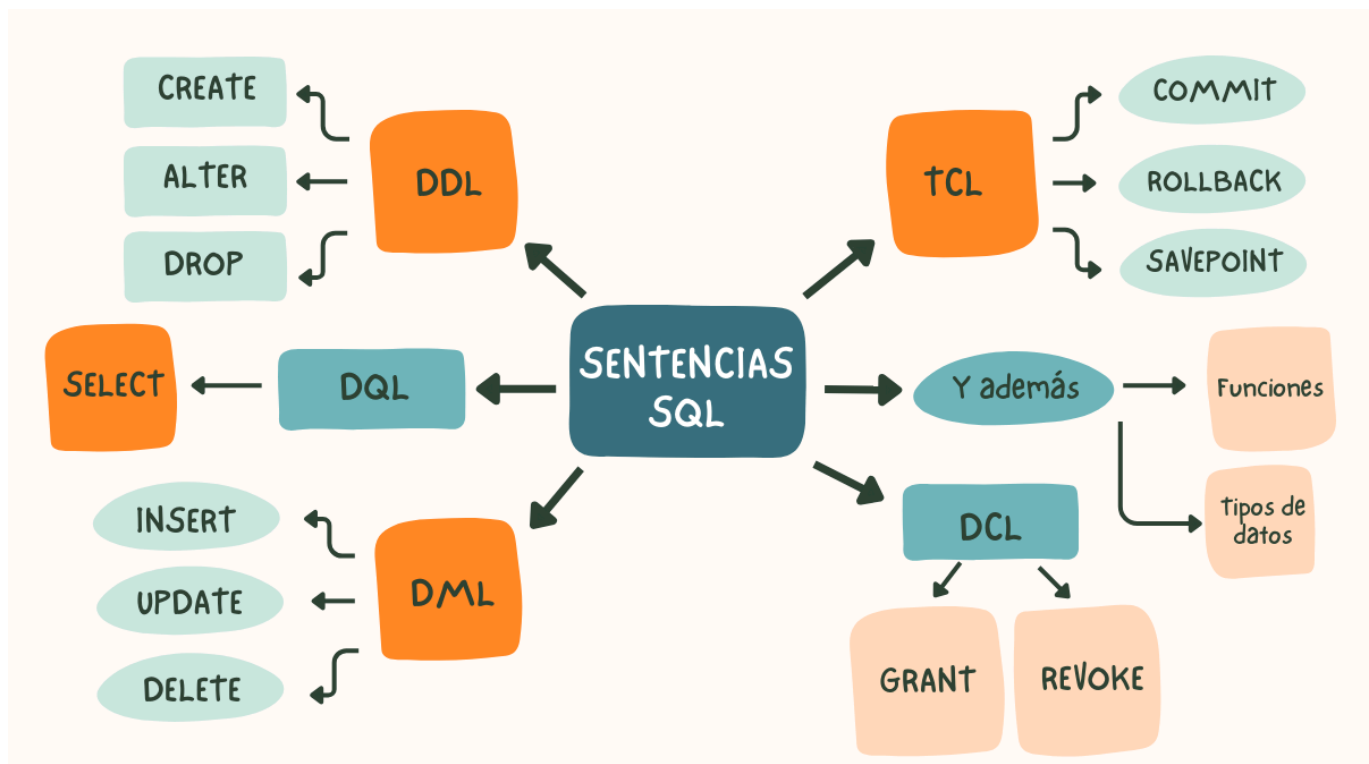
Recordemos brevemente sus características:

- Una base de datos relacional está formada por un conjunto de relaciones o tablas.
- A las relaciones, en SQL, se las denomina tablas.
- Cada tabla tiene una serie de columnas (son los atributos o campos).
- Cada columna tiene un nombre distinto y es de un tipo de datos (entero, real, carácter, fecha, etc.).
- En las tablas se insertan filas (son las tuplas), que después se pueden consultar, modificar o borrar.
- Se deben establecer las restricciones de PK, CK, FK y de dominios de datos.

## 8. Sentencias del Lenguaje SQL

SQL es un lenguaje de consultas de base de datos, que está compuesto por distintos grupos de sentencias, las que permiten llevar a cabo distintas tareas fundamentales en los SGBD, para cada base de datos en particular.

En el siguiente mapa conceptual se aprecian los distintos grupos de sentencias del SQL.



## 8.1 Sentencias DDL. Lenguaje de Definición de Datos.

- Son las sentencias que permiten crear tablas, alterar su definición y eliminarlas.
- En una base de datos relacional existen otros tipos de objetos además de las tablas, como las vistas, los índices, los disparadores, usuarios y roles, temas que no se abordarán en el presente documento.
- Las sentencias para crear, alterar y eliminar vistas, índices, disparadores y usuarios también pertenecen a este conjunto.
- Ejemplos: CREATE, ALTER, DROP, TRUNCATE.



## 8.2 Sentencias DML. Lenguaje de Manipulación de Datos.

- Son las sentencias que permiten insertar datos en las tablas, consultarlos, modificarlos y borrarlos.
- Ejemplos: INSERT, UPDATE, DELETE.

## 8.3 Sentencias DQL. Lenguaje de Consulta de Datos.


- La sentencia principal que permite consultar datos es SELECT.
- Esta sentencia se utiliza en conjunto con otra gran cantidad de sentencias del mismo tipo que permiten realizar consultas complejas sobre los datos de una base de datos.
- Estas sentencias son incluidas por algunos autores dentro del conjunto de sentencias DML SQL, aunque sólo permiten consultar datos y no manipularlos.
- Ejemplos: SELECT, FROM, WHERE, JOIN, UNION, INTERSECT, etc.

## 8.4 Sentencias DCL. Lenguaje de Control de Datos.

- Son las sentencias que utilizan los administradores de la base de datos para realizar sus tareas, como por ejemplo, crear usuarios y conceder o revocar los privilegios de éstos sobre los objetos de la base de datos.
- Ejemplos: GRANT, REVOKE.

## 8.5 Sentencias TCL. Lenguaje de Control Transaccional.

- Son comandos de SQL que permiten manejar transacciones en una base de datos relacional.
- Permiten controlar transacciones de múltiples usuarios sobre la base de datos de forma concurrente.
- Ejemplos: COMMIT, ROLLBACK.

	<p><b>Visualizar la píldora educativa 3 titulada:</b>  <i>02: Tipos de comandos SQL ( DDL,DML,DCL y TCL)   @Ingenioteka</i>          Autor: @Ingenioteka - YouTube  <a href="#">Link al video</a></p>
---	---


## Caso de estudio. ONG Médicos sin Fronteras

Partiendo del diseño del **esquema relacional** de la **base de datos** generado utilizando el IDE Gráfico MySQL WorkBench, se puede generar el esquema físico de forma automática por medio de la funcionalidad "Ingeniería hacia adelante", como se explicó en documentos anteriores.

En esta oportunidad, crearemos el **esquema físico** para la base de datos del **Caso de Estudio ONG Médicos sin Fronteras** por medio de **sentencias SQL DDL**. Luego, llevaremos a cabo el **poblado de la base de datos** por medio de **sentencias SQL DML**. Esto se corresponde con la **Parte 1** de este apartado.

Por último, realizaremos **consultas sencillas** utilizando las **sentencias SQL DQL**. Esto se corresponde con la **Parte 2** de este apartado.

Puedes **descargar** los archivos con los **ejercicios** de este documento para el **Caso de Estudio ONG Médicos sin Fronteras**, en el **link** que se detalla a continuación.

	<p><b>Carpeta con los ejercicios resueltos</b>  <i>Caso de Estudio Médicos sin Fronteras - Ejercicios SQL</i>          Autora: Mariana Adó  <a href="#">Link a la carpeta</a></p>
---	---


## Parte 1.

### 1. Instalar y conectarse al servidor del SGBD MySQL Server

En este documento, continuaremos trabajando con el IDE Gráfico MySQL WorkBench.

#### Ejercicio 1.

Visualizar la píldora educativa 4 con la instalación del SGBD MySQL Server como el IDE Gráfico MySQL WorkBench. Luego, instala en tu computadora el SGBD MySQL Server y el IDE Gráfico MySQL WorkBench.

	<p><b>Visualizar la píldora educativa 4 titulada:</b>  <i>Curso de MySQL: Instalación de MySQL</i>          Autor: Diego Moisset de Espanés @diegomoissetdeespanes - YouTube  <a href="#">Link al video</a></p>
---	---

## 2. Lenguaje de Definición de Datos

Para crear y administrar un esquema físico de datos, SQL presenta las sentencias de definición de datos (DDL SQL) y sus tres cláusulas básicas son: CREATE, DROP y ALTER. Las mismas se corresponden con crear, borrar o modificar el esquema físico existente y otros objetos de la base de datos, como restricciones, claves, índices, usuarios, permisos, etc.

### 2.1 Crear o eliminar una base de datos

Para **crear** una **base de datos** se utiliza la sentencia del SQL DDL:

**CREATE DATABASE nombre\_bd;**



Esta acción permite crear una base de datos con el nombre indicado en la sentencia, asignar el espacio en disco para el esquema físico de la misma, así como definir los metadatos en el catálogo (information\_schema) que el SGBD usará para manipular la base de datos recién creada. Estas acciones se llevan a cabo de manera implícita.

Para **eliminar** una **base de datos** utilizamos la sentencia del SQL DDL:

**DROP DATABASE nombre\_bd;**

La cual borra la base de datos. Esto incluye las tablas y los datos contenidos en ellas, como los metadatos y espacio asignado.

## Ejercicio 2.

	<p><b>Visualizar la píldora educativa 5 titulada:</b>  <i>Curso de MySQL: Creación de un base de datos (create database)</i>          Autor: Diego Moisset de Espanés @diegomoissetdeespanes - YouTube  <a href="#">Link al video</a></p>
	<p><b>Visualizar la píldora educativa 6 titulada:</b>  <i>03: CREAR base de datos con SQL. CREATE DATABASE</i>          Autor: @Ingenioteca - YouTube  <a href="#">Link al video</a></p>

Luego de visualizar las píldoras educativas 5 y 6, crear la base de datos llamada “ong” para el **Caso de Estudio ONG Médicos sin Fronteras** por medio de sentencias DDL SQL. Para ello, utilizaremos la sentencia **CREATE DATABASE ong;** como se visualiza en la Figura 1.

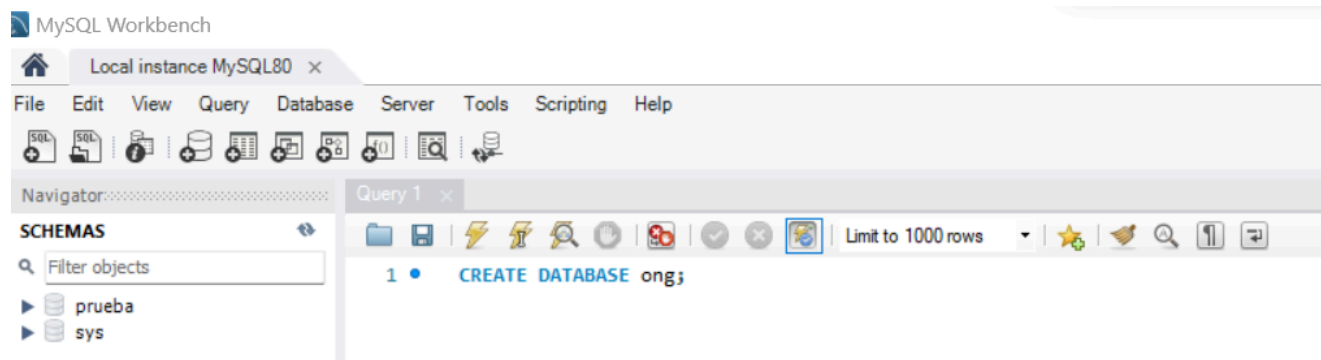


Figura 1. Creación de la base de datos “ong”.

## 2.2 Creación de las tablas

Una vez creada la base de datos, se deberán crear las tablas del esquema físico. Para generar una tabla en una base de datos, la sentencia SQL DDL es **CREATE TABLE**.

El siguiente ejemplo, en la Figura 2, presenta la creación de la tabla denominada “**países**” del **Caso de Estudio ONG Médicos sin Fronteras**.



Figura 2. Creación de la tabla “países” en la base de datos “ong”.

Para definir una tabla es necesario indicar cada una de las columnas que la componen, sus tipos de datos y todas las restricciones: PK, CK, FK y de dominio. Así como datos que permiten o no valores nulos.

Para el ejemplo dado, las columnas son:

- **idpais**: con dominio INTEGER sin signo, la columna no puede ser nula (NOT NULL) y es autoincremental. Esto es así, porque luego se definirá la restricción de clave primaria (PRIMARY KEY) subrogada (única, no nula, entera y autoincremental).
- **nombre**: con dominio VARCHAR (String) con una longitud máxima de 100 caracteres y no puede ser nula (NOT NULL), ya que es una CK, debe ser además, única (UNIQUE).
- **grado\_conflict**: con dominio INTEGER sin signo, no puede ser nula (NOT NULL) y tiene una restricción de dominio, por defecto su valor es cero (DEFAULT 0). Además, se define para esta columna una restricción de dominio a nivel de columna por medio de la sentencia CHECK para que solo tome valores enteros entre cero y diez. Esta restricción de dominio sólo tiene validez dentro del ámbito de la tabla.
- **continente**: con dominio VARCHAR (String) con una longitud máxima de 100 caracteres y no puede ser nula (NOT NULL).

La sintaxis utilizada, si bien es ANSI SQL, tiene similitud con la utilizada por el SGBD MySQL, por lo tanto, se puede copiar y ejecutar en MySQL WorkBench o descargar el

archivo con el SQL DDL ANSI para el **Caso de Estudio ONG Médicos sin Fronteras**, como se especificó anteriormente en este documento.

Los valores posibles para los dominios (tipos de datos) de las columnas están ligados directamente con los productos comerciales. Lo mismo ocurre con la definición de los atributos autoincrementales. No todos los SGBD implementan el estándar ANSI SQL, por ello, siempre hay que leer la documentación del producto con el cual se vaya a trabajar.

El siguiente ejemplo presenta la creación de otra tabla, donde se agrega la definición de una **clave foránea**. En la Figura 3, se muestra la creación de la tabla denominada **“lugares”** del **Caso de Estudio ONG Médicos sin Fronteras**.

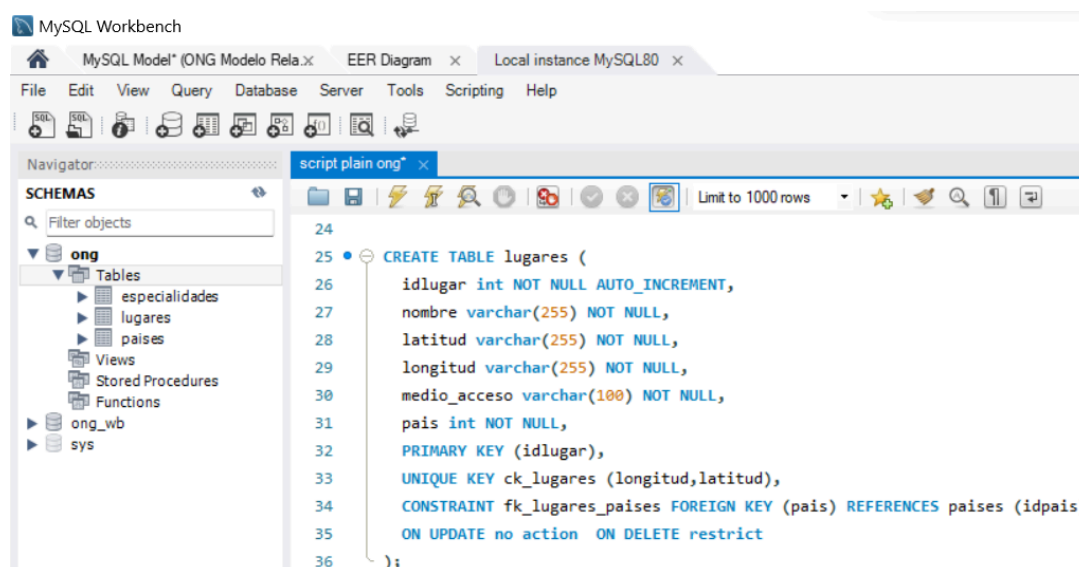


Figura 3. Creación de la tabla “lugares” en la base de datos “ong”.

En este caso, se definen todas las columnas que componen a la tabla, sus tipos de datos y todas las restricciones: PK, CK, FK y de dominio. Así como también, los datos que permiten o no valores nulos.

Luego, se define la restricción de clave foránea llamada **fk\_lugares\_paises** sobre la columna **“pais”** de la tabla **“lugares”**, que hace referencia a la tabla de donde se obtiene la PK, es decir, se referencia la PK de la tabla **“países”**. Se indica, además, el tipo de restricción de



integridad referencial definido: el borrado está restringido y por la modificación no se toma acción.

**CONSTRAINT fk\_lugares\_paises FOREIGN KEY (pais) REFERENCES paises (idpais)  
ON UPDATE no action ON DELETE restrict**

Ejercicio 3.

Creación de las tablas del **Caso de Estudio ONG Médicos sin Fronteras**.

Visualice la píldora educativa 7 y cree las tablas del esquema físico para el **Caso de Estudio ONG Médicos sin Fronteras**, teniendo en cuenta lo visto anteriormente en este apartado. Es decir, cree las tablas, las columnas que las componen, sus tipos de datos y todas las restricciones: PK, CK, FK y de dominio. Así como los datos que permiten o no valores nulos.

	<p><b>Visualizar la píldora educativa 7 titulada:</b>  <i>04: CREAR TABLA en SQL de forma sencilla. CREATE TABLE</i>          Autor: @Ingenioteca - YouTube  <a href="#">Link al video</a></p>
	<p><b>Archivo con sentencias de SQL DDL para crear el esquema físico de la BD “ong”</b>  <i>Caso de Estudio Médicos sin Fronteras - Ejercicios SQL</i>          Autora: Mariana Adó  <a href="#">Link al archivo</a></p>

### 2.3. Borrar una tabla

Para eliminar una tabla del esquema físico de la base de datos la sentencia SQL DDL es:  
**DROP TABLE nombre\_tabla;**

### 2.4. Modificar una tabla

Para modificar una tabla del esquema físico de la base de datos la sentencia SQL DDL es:

### ALTER TABLE nombre\_tabla ( ... ) ;

Esta sentencia debe indicar, además, que tipo de modificación se desea realizar sobre la tabla. Con lo cual, se pueden agregar, modificar o borrar columnas, índices o restricciones de integridad.

A modo de ejemplo, para el **Caso de Estudio ONG Médicos sin Fronteras**, la siguiente sentencia SQL DDL agrega la columna **cantidad\_habitantes** a la tabla **lugares** creada anteriormente, borra la columna **medio\_acceso** y modifica el tipo de dato de la columna **nombre** por un VARCHAR de longitud 100.

```
ALTER TABLE lugares (
ADD COLUMN cantidad_habitantes NUMERIC(10,2) NOT NULL,
DROP COLUMN medio_acceso,
ALTER COLUMN nombre VARCHAR(100) NOT NULL);
```

Ejercicio 4.

Se quiere ampliar el esquema físico de la base de datos del **Caso de Estudio ONG Médicos sin Fronteras**, para manejar la información de los socios, las cuotas que éstos pagan, los proveedores a los que la ONG compra insumos.

Para ello, se nos solicita crear las tablas socios, cuotas, proveedores, insumos y compras.

A continuación, en la Figura 4 se muestran las tablas y sus restricciones en el esquema relacional:

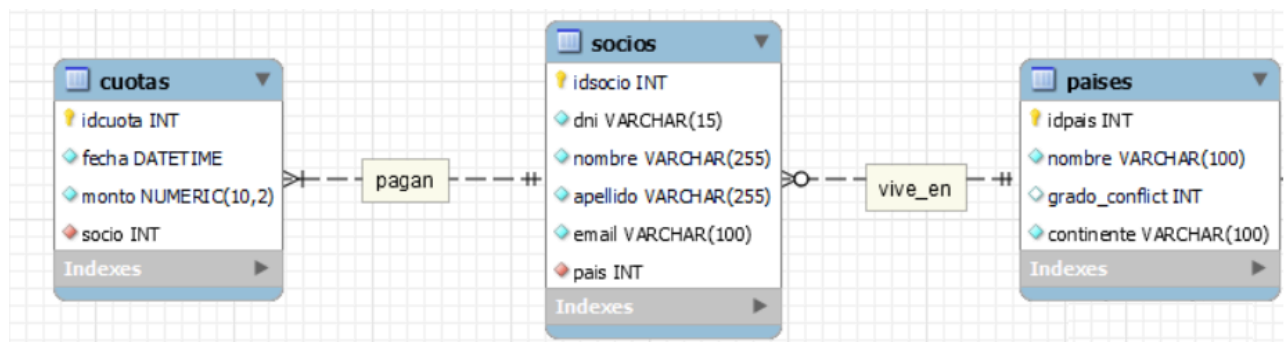





Figura 4. Esquema relacional de las tablas “cuotas”, “socios” y “países” en la base de datos “ong”.

Restricciones:

1. La columna monto de la tabla cuotas tiene por defecto el valor cero.
2. La tabla cuotas tiene una CK compuesta conformada por las columnas fecha y socio.
3. La tabla socios tiene una CK simple conformada por la columna email.

Se solicita escribir las sentencias en SQL DDL para ampliar el esquema físico.

	<p><b>Archivo con sentencias del Ejercicio 4</b>  <i>Caso de Estudio Médicos sin Fronteras - Ejercicios SQL</i>            Autora: Mariana Adó  <a href="#">Link al archivo</a></p>
--	---


Ejercicio 5.

- 1) Se necesita saber el origen de cada idioma. Para ello se nos solicita modificar la tabla idiomas, agregando la columna origen de tipo VARCHAR(15). Esta columna solo tomará valores del conjunto { 'GERMANICO', 'LATINO', 'AFROASIATICO', 'ESLAVO', 'INDOEUROPEO', 'AUSTROASIATICO', 'ASIATICO' }, agregando de esta forma, una restricción de dominio para esa columna.
- 2) Además, se deben agregar las siguientes restricciones de dominio que no se contemplaron en el Ejercicio 3, ellas son:
  - a) La columna continente de la tabla países, sólo tomará valores del conjunto: { 'AMERICA DEL NORTE', 'AMERICA CENTRAL', 'AMERICA DEL SUR', 'EUROPA', 'AFRICA', 'ASIA', 'OCEANIA', 'LA ANTARTIDA' }.
  - b) La columna medio\_acceso de la tabla lugares, sólo tomará valores del conjunto: { 'AUTO', '4x4', 'CAMION', 'AVION', 'HELICOPTERO', 'BARCO', 'TREN' }.

Las acciones llevadas a cabo en los ejercicios 4 y 5, además de modificar el esquema físico, también impactarán sobre los esquemas conceptual y lógico de la base de datos.

En este documento no nos ocuparemos de este tema. Se debe hacer ingeniería inversa en la herramienta MySQL WorkBench para modificar el esquema relacional de la base de datos y luego modificar el esquema lógico de alto nivel y el esquema conceptual.

Además, cabe aclarar que no creamos un dominio con la sentencia CREATE DOMAIN como propone el estándar ANSI SQL, ya que MySQL no lo implementa. Esto puede estudiarse en el documento sobre SQL en PostgreSQL.

	<p><b>Archivo con sentencias del Ejercicio 5</b>  <i>Caso de Estudio Médicos sin Fronteras - Ejercicios SQL</i>          Autor: Mariana Adó  <a href="#">Link al archivo</a></p>
---	--

### 3. Lenguaje de Manipulación de Datos

En esta sección del documento se presenta la sintaxis y semántica asociada de SQL2 (SQL92) sobre las sentencias del lenguaje de manipulación de datos (SQL DML) básicas para agregar, borrar o modificar información almacenada en una base de datos.

#### 3.1 Insertar datos

La sentencia para insertar datos, es decir, insertar filas en cada tabla en particular, es la siguiente:

**INSERT INTO nombre\_tabla [(campo1[, campo2[, ...]])]  
VALUES (valor1[, valor2[, ...]);**

La sentencia INSERT INTO inserta una sola fila cada vez que es ejecutada.

Veamos algunos ejemplos:

**Ejemplo 1:** se inserta una fila en la tabla países de la base de datos “ong” del **Caso de Estudio ONG Médicos sin Fronteras**.

```
SQL File 4* x
Limit to 1000 rows
1 -- Insertamos datos en la tabla paises
2 • INSERT INTO paises (nombre, grado_conflict, continente) VALUES ('ARGENTINA', 2, 'AMERICA DEL SUR');
```

Figura 5. Sentencia de INSERT para insertar una fila en la tabla “paises” de la base de datos “ong”.

Se puede observar que se indican tres de las cuatro columnas que componen a la tabla de países. La columna idpais está definida como una clave subrogada (entera, única, no nula y autoincremental), por este motivo, será el SGBD el encargado de asignarle el valor correspondiente. Si se intenta asignar un valor a una columna definida como autoincremental, la operación de inserción falla. No todos los SGBD cumplen con este último requisito, por tal razón, no hay que insertar ni modificar el valor de una clave subrogada.

**Ejemplo 2:** se inserta una fila en la tabla lugares de la base de datos “ong” del **Caso de Estudio ONG Médicos sin Fronteras**.

```
SQL File 4* x
Limit to 1000 rows
1 -- Insertamos datos en la tabla lugares
2 • INSERT INTO lugares (nombre, medio_acceso, latitud, longitud, pais)
3 VALUES ('Impenetrable, Chaco', 'AUTO', '-25.17176173156959', '-61.09796092747129', 1);
4
```

Figura 6. Sentencia de INSERT para insertar una fila en la tabla “lugares” de la base de datos “ong”.

Se debe tener cuidado, además, con la última columna especificada en la sentencia, ya que representa a una clave foránea. Al crear la tabla se definió una **restricción de integridad referencial**, por ende, se debe indicar un valor que sea válido. Esto significa que el número 1 correspondiente a el país debería tener su correlato en la tabla paises, es decir, debe haber un país que es la columna idpais tenga el valor 1, de lo contrario no se permite insertar la fila por falla en la integridad referencial.

A continuación, puedes descargar el archivo .sql con las sentencias de INSERT para poblar la base de datos “ong” del **Caso de Estudio ONG Médicos sin Fronteras**.



### Archivo con sentencias de INSERT para poblar la BD “ong”

*Caso de Estudio Médicos sin Fronteras - Ejercicios SQL*

Autor: Mariana Adó

[Link al archivo](#)

## 3.2 Modificar datos

La sentencia para modificar datos, es decir, modificar una columna o varias columnas de una tabla en particular, es la siguiente:

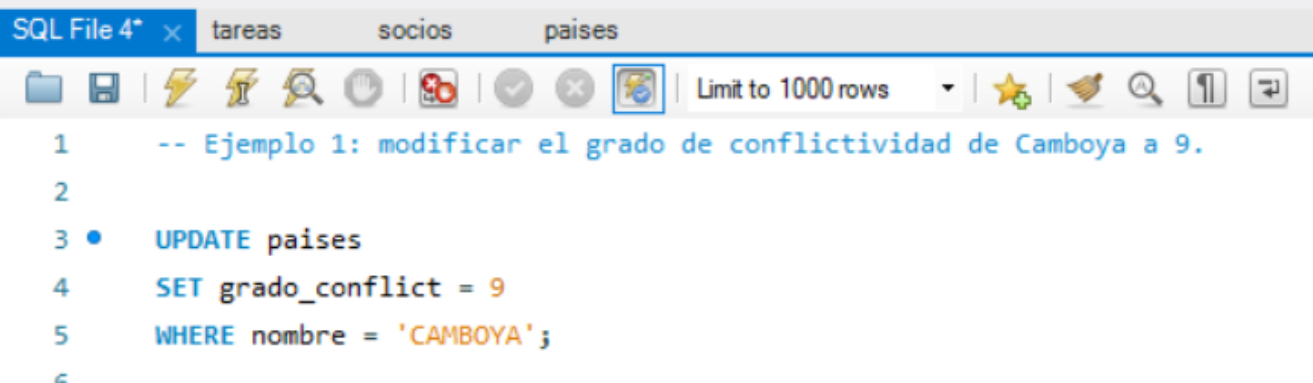
```
UPDATE nombre_tabla
SET columna = {expresion|DEFAULT|NULL}
[, columna = {expresion|DEFAULT|NULL}...]
[WHERE condiciones];
```

La cláusula WHERE no es obligatoria, se pueden modificar los datos de una tabla sin que éstos deban cumplir una condición.

**IMPORTANTE:** aunque el SGBD permita modificar una el valor de una columna con una restricción de PK, nunca es conveniente modificarla.

Veamos algunos ejemplos:

**Ejemplo 1:** modificar el grado de conflictividad de Camboya a 9.

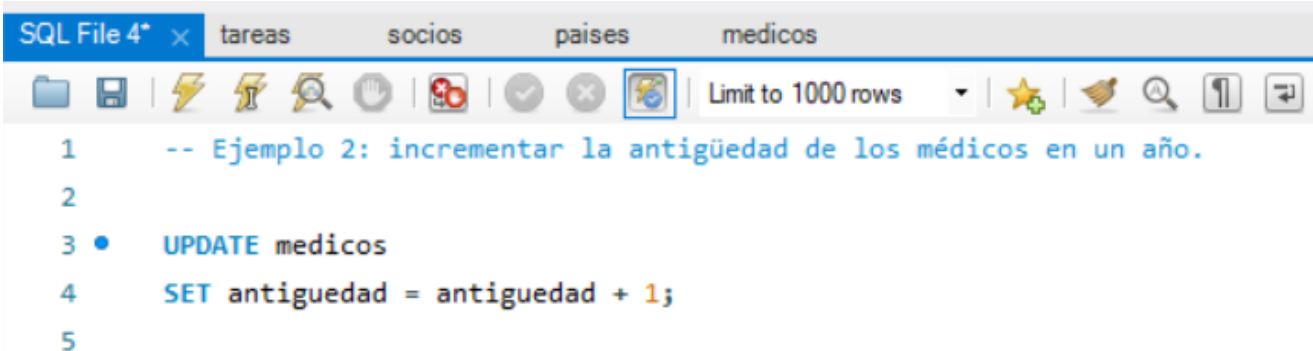


```
SQL File 4* x  tareas  socios  países
Limit to 1000 rows
1  -- Ejemplo 1: modificar el grado de conflictividad de Camboya a 9.
2
3  •  UPDATE países
4     SET grado_conflict = 9
5     WHERE nombre = 'CAMBOYA';
6
```

Figura 7. Sentencia de UPDATE para modificar el grado de conflictividad de Camboya a 9.

Se debe tener en cuenta que si varias filas cumplen la condición especificada en el WHERE, todas ellas serán modificadas. Para el ejemplo, solo hay una fila que cumpla la condición debido a que el nombre es una clave candidata en la tabla países.

**Ejemplo 2:** incrementar la antigüedad de los médicos en un año.




```

SQL File 4* x  tareas  socios  países  medicos
Limit to 1000 rows
1  -- Ejemplo 2: incrementar la antigüedad de los médicos en un año.
2
3  •  UPDATE medicos
4    SET antigüedad = antigüedad + 1;
5

```

Figura 8. Sentencia de UPDATE para incrementar la antigüedad de los médicos en un año.

	<p><b>Archivo con sentencias de UPDATE - Ejemplos 1 y 2</b>  Caso de Estudio Médicos sin Fronteras - Ejercicios SQL  Autora: Mariana Adó  <a href="#">Link al archivo</a></p>
---	---

### 3.3 Borrar datos

La sentencia para borrar una fila o un conjunto de filas de una tabla en particular, es la siguiente:

**DELETE FROM nombre\_tabla**  
**[ WHERE condiciones];**

La cláusula WHERE no es obligatoria, se pueden borrar los datos de una tabla sin que éstos deban cumplir una condición. Si no se escribe la condición del WHERE se borrarán todas las filas de la tabla.

El SGBD borrará la fila o filas siempre que la restricción de integridad referencial lo permita.


**Ejemplo 1:** borrar el idioma con código ID000006.



```

1      -- Nos conectamos a la BD ong antes de usarla
2
3  •   USE ong;
4
5      -- Ejemplo 1: borrar el idioma con código ID000006
6
7  •   DELETE FROM idiomas
8      WHERE codigo = 'ID000006';
  
```

Figura 9. Sentencia de DELETE para borrar el idioma con código ID000006.

	<p><b>Archivo con sentencias de DELETE - Ejemplos 1</b>  Caso de Estudio Médicos sin Fronteras - Ejercicios SQL  Autora: Mariana Adó  <a href="#">Link al archivo</a></p>
---	---

## Parte 2.

### 4. Lenguaje de Consulta de Datos

En esta sección del documento se presenta la sintaxis y semántica asociada de SQL2 (SQL92) para recuperar información de una base de datos, es decir, sobre las sentencias del lenguaje de consulta de datos (SQL DQL).

Se sabe que aproximadamente el 80% de las operaciones que se realizan en una base de datos son de consulta, por lo tanto, se hace de suma importancia estudiar y conocer en detalle las sentencias para llevarlas a cabo.

En este documento solo se abordarán consultas sencillas, dejando para posteriores documentos el conjunto de sentencias que permiten realizar consultas complejas sobre la base de datos, el procesamiento y la optimización de las mismas.

#### 4.1 Estructura de una consulta

La estructura básica de una consulta SQL tiene el siguiente formato:

```
SELECT lista de columnas  
FROM lista de tablas  
[WHERE predicado];
```

Componentes de la consulta:

- La **lista de columnas** indica los nombres de las columnas que serán presentadas en el resultado. Estas columnas deben formar parte de las tablas especificadas en el FROM de la consulta. Los nombres de las columnas deben ir separadas por comas.
- La **lista de tablas** indica las tablas de la base de datos necesarias para resolver la consulta; sobre las tablas indicadas en la lista se realiza un producto cartesiano. Los nombres de las tablas deben ir separadas por comas.
- El **predicado** indica qué condición lógica deben cumplir las filas de las tablas para ser mostradas en el resultado final de la consulta. La cláusula WHERE no es obligatoria,

se pueden consultar los datos de una tabla o tablas sin que éstos deban cumplir una condición.


- Como toda sentencia SQL debe terminar con punto y coma (;).


Una consulta SQL siempre retorna como resultado una tabla temporal en memoria.

Si en vez de escribir la lista de columnas escribimos el comodín asterisco (\*), la consulta nos devolverá todas las columnas de las tablas involucradas en la cláusula FROM.

**SELECT \***  
**FROM lista de tablas**  
**[WHERE predicado];**

## 4.2 Consultas sencillas

	<p><b>Visualizar la píldora educativa 8 titulada:</b>  <i>18: Cómo MOSTRAR DATOS con el comando SELECT</i>          Autor: @Ingenioteca - YouTube  <a href="#">Link al video</a></p>
---	--

	<p><b>Archivo con sentencias de SELECT - Ejemplos</b>  <i>Caso de Estudio Médicos sin Fronteras - Ejercicios SQL</i>          Autora: Mariana Adó  <a href="#">Link al archivo</a></p>
---	--



**Ejemplo 1:** listar los nombres y el email de los voluntarios de la ONG Médicos sin Fronteras.

```

Ejemplos SELECT x
Limit to 1000 rows
3 • USE ong;
4
5 -- Ejemplo 1: listar los nombres y el email de los voluntarios de la ONG Médicos sin Fronteras
6
7 • SELECT nombre, email
8 FROM voluntarios;
9

```

Figura 10. Consulta del nombre y el email de los voluntarios de la ONG Médicos sin Fronteras.

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
nombre	email			
GIMENA	gimena_fern@hotmail.com			
LUCAS	lucas_caponi@gmail.com			
MARIA INES	mariaineslorenzetti@hotmail.com			
GERONIMO	geronimososa@gmail.com			
GRACIELA	gracielamancuso@hotmail.com			
LUCAS	lucaspereyra@hotmail.com			
LUCRECIA	lucreciamartinez@hotmail.com			
PAMELA	pamelagaray@hotmail.com			
SERGIO	sergiomanrique@hotmail.com			
AGUSTINA	agustina_masias@gmail.com			
MARCOS	marcos_andrade@gmail.com			

voluntarios 1 x

Figura 11. Resultado de la consulta del nombre y el email de los voluntarios de la ONG Médicos sin Fronteras.

**Ejemplo 2:** listar todos los datos de los socios de la ONG Médicos sin Fronteras.

```

Ejemplos SELECT x
Limit to 1000 rows
10 -- Ejemplo 2: listar todos los datos de los socios de la ONG Médicos sin Fronteras.
11
12 • SELECT *
13 FROM socios;
14

```

Figura 12. Consulta de todos los datos de los socios de la ONG Médicos sin Fronteras.

Result Grid						
		Filter Rows:			Edit:	Export/Import:
	idsocio	dni	nombre	apellido	email	pais
▶	1	24567890	LUCIA	MANELLI	lu_manelli@gmail.com	1
	2	26987652	JUAN PABLO	GIMENEZ	jpgim@gmail.com	1
	3	25678545	DIEGO	LUCELLI	diego_luc@yahoo.com.ar	1
	4	14786900	MARISA	DUARTE	marid67@hotmail.com	1
	5	13567890	LUIS ALBERTO	ROSSI	luisarossi@gmail.com	1
✱	NULL	NULL	NULL	NULL	NULL	NULL

Figura 13. Resultado de la consulta de todos los datos de los socios de la ONG Médicos sin Fronteras.

### 4.3 Consultas con operadores de comparación

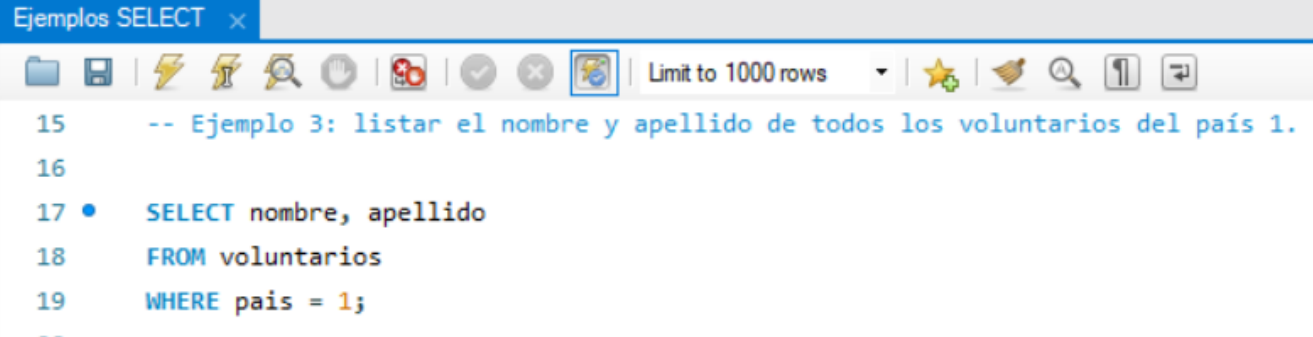
Los **operadores de comparación** comprueban dos expresiones. Luego de evaluar la expresión de comparación, retorna verdadero (TRUE) o falso (FALSE) dependiendo del resultado de la evaluación. Se deben usar en expresiones que se escriban en el predicado de la cláusula del WHERE.

La Tabla 1 muestra los operadores de comparación de SQL2 (SQL92).

Operador	Significado
=	Igual a
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
<>	No es igual a

Tabla 1. Operadores de comparación de SQL2 (SQL92).

**Ejemplo 3:** listar el nombre y apellido de todos los voluntarios del país 1.



Ejemplos SELECT x

```

15  -- Ejemplo 3: listar el nombre y apellido de todos los voluntarios del país 1.
16
17  •  SELECT nombre, apellido
18      FROM voluntarios
19      WHERE país = 1;
20

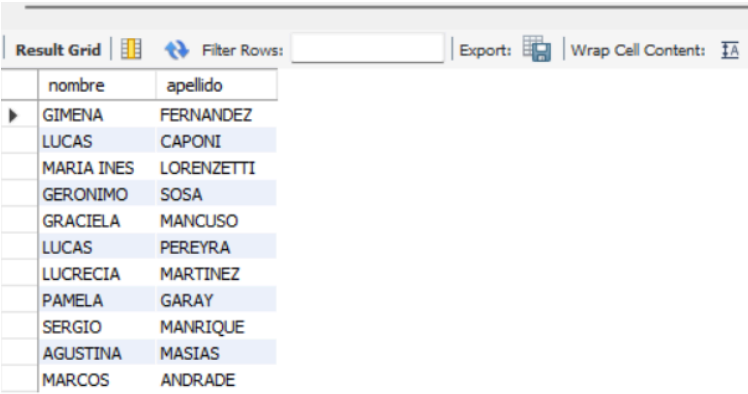
```

Figura 14. Consulta del nombre y apellido de todos los voluntarios del país 1.

En este caso es necesario utilizar la cláusula WHERE para filtrar en el resultado las filas deseadas.

Se puede notar que en el predicado se utiliza el operador de comparación “Igual a” para comparar que el país de los voluntarios sea efectivamente igual a 1.

Se debe comparar por valores que se correspondan, es decir, que tengan el mismo tipo de dato. En este caso, la columna país es de tipo entera porque tiene una restricción de FK.



nombre	apellido
GIMENA	FERNANDEZ
LUCAS	CAPONI
MARIA INES	LORENZETTI
GERONIMO	SOSA
GRACIELA	MANCUSO
LUCAS	PEREYRA
LUCRECIA	MARTINEZ
PAMELA	GARAY
SERGIO	MANRIQUE
AGUSTINA	MASIAS
MARCOS	ANDRADE

voluntarios 3 x

Figura 15. Resultado de consulta del nombre y apellido de todos los voluntarios del país 1.

**Ejemplo 4:** listar todos los datos de las tareas que tienen menos de 40 horas semanales como máximo.

```

Ejemplos SELECT* x
Limit to 1000 rows
21  -- Ejemplo 4: listar todos los datos de las tareas que tienen menos de 40 horas semanales como máximo.
22
23  • SELECT *
24  FROM tareas
25  WHERE horasmax < 40;
26

```

Figura 16. Consulta sobre las tareas que tienen menos de 40 horas semanales como máximo.

	id tarea	codigo	nombre	fecha_inicio	fecha_fin	horasmin	horasmax	presupuesto	coordinador	lugar
▶	7	TAR0007	COVID - 19 TAR0007	2020-02-15	NULL	8.00	35.00	1000000.00	NULL	6
	8	TAR0008	COVID - 19 TAR0008	2020-02-08	NULL	10.00	30.00	1500000.00	4	7
	10	TAR0010	COVID - 19 TAR0010	2020-02-23	NULL	8.00	35.00	1500000.00	NULL	9
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figura 17. Resultado de la consulta sobre las tareas que tienen menos de 40 horas semanales como máximo.

## 4.4 Operadores lógicos

Los **operadores lógicos** comprueban el valor de verdad de alguna condición o predicado. Al igual que los operadores de comparación, retornan valores del tipo de dato BOOLEAN con el valor TRUE, FALSE o UNKNOWN. Se deben usar en expresiones que se escriban en el predicado de la cláusula del WHERE.

La Tabla 2 muestra los operadores lógicos de SQL2 (SQL92).

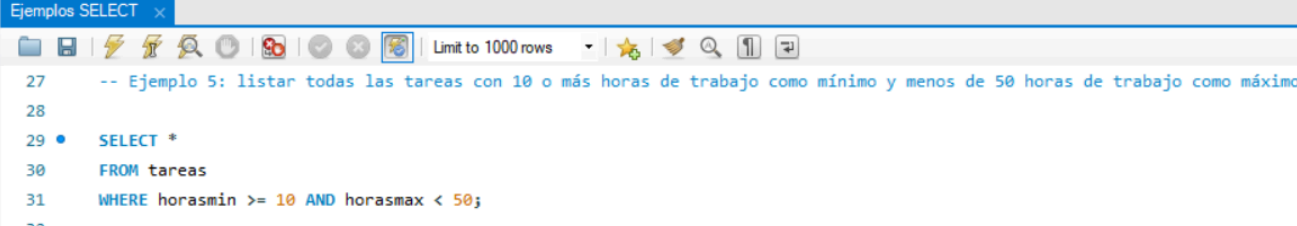
Operador	Significado
<b>ALL</b>	Retorna TRUE si el conjunto completo de comparaciones es TRUE.
<b>AND</b>	Retorna TRUE si ambas expresiones booleanas son TRUE.

<b>ANY</b>	Retorna TRUE si cualquier miembro del conjunto de comparaciones es TRUE.
<b>BETWEEN</b>	Retorna TRUE si el operando está dentro del intervalo especificado.
<b>EXISTS</b>	Retorna TRUE si una subconsulta contiene al menos una fila.
<b>IN</b>	Retorna TRUE si el operando es igual a uno de la lista de expresiones. Se puede comparar con la operación de pertenencia de conjuntos.
<b>LIKE</b>	Retorna TRUE si el operando coincide con el patrón especificado.
<b>NOT</b>	Invierte el valor de cualquier otro operador booleano.
<b>OR</b>	Retorna TRUE si cualquiera de las dos expresiones booleanas es TRUE.
<b>SOME</b>	Retorna TRUE si alguna de las comparaciones de un conjunto es TRUE.

Tabla 2. Operadores de lógicos de SQL2 (SQL92).

#### 4.4.1 Operador AND

**Ejemplo 5:** listar todas las tareas con 10 o más horas de trabajo como mínimo y menos de 50 horas de trabajo como máximo.



```

27 -- Ejemplo 5: listar todas las tareas con 10 o más horas de trabajo como mínimo y menos de 50 horas de trabajo como máximo.
28
29 • SELECT *
30 FROM tareas
31 WHERE horasmin >= 10 AND horasmax < 50;
32

```

Figura 18. Consulta sobre las tareas con 10 o más horas de trabajo como mínimo y menos de 50 horas de trabajo como máximo.

Result Grid

Filter Rows:

Edit:

Export/Import:

Wrap Cell Content:

	id tarea	codigo	nombre	fecha_inicio	fecha_fin	horasmin	horasmax	presupuesto	coordinador	lugar
▶	1	TAR0001	COVID - 19 TAR0001	2020-03-16	NULL	10.00	40.00	1000000.00	4	1
	6	TAR0006	COVID - 19 TAR0006	2020-02-03	NULL	10.00	40.00	2000000.00	5	12
	8	TAR0008	COVID - 19 TAR0008	2020-02-08	NULL	10.00	30.00	1500000.00	4	7
✱	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figura 19. Resultado de la consulta sobre las tareas con 10 o más horas de trabajo como mínimo y menos de 50 horas de trabajo como máximo.

En el Ejemplo 5 es necesario usar el operador lógico **AND**, ya que ambas expresiones del predicado de la cláusula WHERE deben ser verdaderas para que el resultado de la consulta muestre a la fila que las cumpla.

#### 4.4.2 Operador BETWEEN

El operador **BETWEEN** permite probar fácilmente si una expresión está dentro de un rango de valores. Los valores pueden ser texto, fechas o números. Se puede usar en una sentencia SELECT, INSERT, UPDATE o DELETE y debe utilizarse en la cláusula WHERE. La cláusula BETWEEN devolverá las filas para las cuales la expresión está dentro del rango de valores especificados.

**Ejemplo 6:** listar las tareas cuya fecha de inicio corresponde con el mes de marzo de 2020.

```

Ejemplos SELECT x
Limit to 1000 rows
33 -- Ejemplo 6: listar las tareas cuya fecha de inicio se corresponda con el mes de marzo de 2020.
34
35 • SELECT *
36 FROM tareas
37 WHERE fecha_inicio BETWEEN '2020-03-01' AND '2020-03-31';
38

```

Figura 20. Consulta sobre las tareas cuya fecha de inicio corresponde con el mes de marzo de 2020.

Result Grid

Filter Rows:

Edit:

Export/Import:

Wrap Cell Content:

	id tarea	codigo	nombre	fecha_inicio	fecha_fin	horasmin	horasmax	presupuesto	coordinador	lugar
▶	1	TAR0001	COVID - 19 TAR0001	2020-03-16	NULL	10.00	40.00	1000000.00	4	1
	2	TAR0002	COVID - 19 TAR0002	2020-03-16	NULL	15.00	50.00	1000000.00	5	2
✱	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figura 21. Resultado de la consulta sobre las tareas cuya fecha de inicio corresponde con el mes de marzo de 2020.

**Ejemplo 7:** listar las tareas cuya fecha de inicio corresponde con el mes de marzo de 2020.

En este ejemplo se resuelve la misma consulta del ejemplo 6 pero con expresiones lógicas y el operador lógico AND.

Ejemplos SELECT
<pre> 39  -- Ejemplo 7: listar las tareas cuya fecha de inicio corresponde con el mes de marzo de 2020. 40 41  •  SELECT * 42     FROM tareas 43     WHERE fecha_inicio &gt;= '2020-03-01' AND fecha_inicio &lt;='2020-03-31'; 44 </pre>

Figura 22. Consulta sobre las tareas cuya fecha de inicio corresponde con el mes de marzo de 2020.

Result Grid

Filter Rows:

Edit:

Export/Import:

Wrap Cell Content:

	id tarea	codigo	nombre	fecha_inicio	fecha_fin	horasmin	horasmax	presupuesto	coordinador	lugar
▶	1	TAR0001	COVID - 19 TAR0001	2020-03-16	NULL	10.00	40.00	1000000.00	4	1
	2	TAR0002	COVID - 19 TAR0002	2020-03-16	NULL	15.00	50.00	1000000.00	5	2
•	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figura 23. Resultado de la consulta sobre las tareas cuya fecha de inicio corresponde con el mes de marzo de 2020.

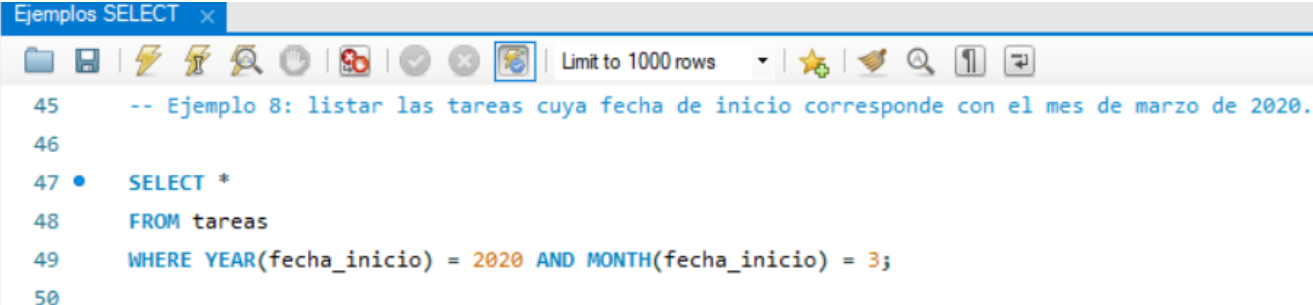
Se puede observar que la consulta devuelve como resultado el mismo conjunto de filas que la consulta del Ejemplo 6.

**Ejemplo 8:** listar las tareas cuya fecha de inicio corresponde con el mes de marzo de 2020.

En este ejemplo se resuelve la misma consulta del ejemplo 6 pero con funciones de fecha y el operador lógico AND.

Se debe tener en cuenta que las funciones para fechas son implementadas en cada SGBD de forma particular, por lo cual, no forman parte del estándar SQL2 (SQL92). Siempre se recomienda leer la documentación del mismo antes de trabajar con datos que incluyan fechas.

En la solución del ejemplo se utilizaron las funciones de fecha implementadas en el SGBD MySQL.

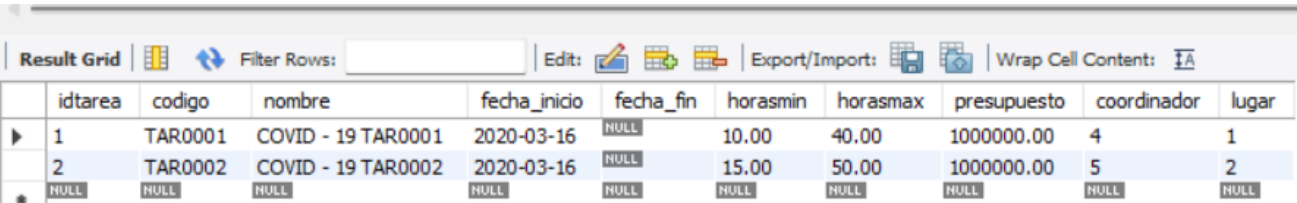


```

45  -- Ejemplo 8: listar las tareas cuya fecha de inicio corresponde con el mes de marzo de 2020.
46
47  •  SELECT *
48     FROM tareas
49     WHERE YEAR(fecha_inicio) = 2020 AND MONTH(fecha_inicio) = 3;
50

```

Figura 24. Consulta sobre las tareas cuya fecha de inicio corresponde con el mes de marzo de 2020.



	id tarea	codigo	nombre	fecha_inicio	fecha_fin	horasmin	horasmax	presupuesto	coordinador	lugar
▶	1	TAR0001	COVID - 19 TAR0001	2020-03-16	NULL	10.00	40.00	1000000.00	4	1
	2	TAR0002	COVID - 19 TAR0002	2020-03-16	NULL	15.00	50.00	1000000.00	5	2
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figura 25. Resultado de la consulta sobre las tareas cuya fecha de inicio corresponde con el mes de marzo de 2020.

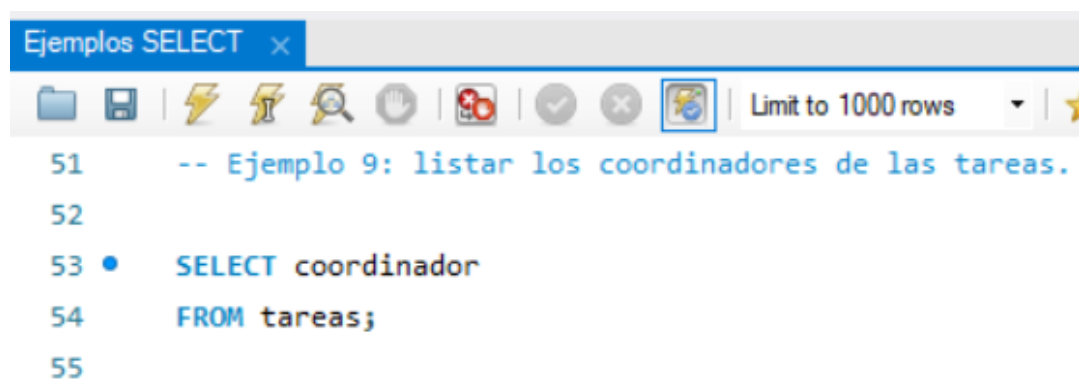
Se puede observar que la consulta devuelve como resultado el mismo conjunto de filas que las consultas del Ejemplo 6 y el Ejemplo 7, pero no está resuelta con sentencias del estándar SQL2 (SQL 92).



#### 4.4.3 Cláusula DISTINCT

La cláusula **DISTINCT** permite eliminar los valores duplicados en el resultado de una consulta. Al utilizar DISTINCT, se garantiza que cada fila en el resultado de la consulta sea única en cuanto a los campos especificados en la cláusula SELECT.

**Ejemplo 9:** listar los coordinadores de las tareas.



```

51      -- Ejemplo 9: listar los coordinadores de las tareas.
52
53 •    SELECT coordinador
54      FROM tareas;
55
  
```

Figura 26. Consulta sobre los coordinadores de las tareas.



coordinador
NULL
NULL
NULL
NULL
4
4
4
5
5
5

Figura 27. Resultado de la consulta sobre los coordinadores de las tareas.

En el ejemplo vemos que en el resultado de la ejecución de la consulta los números 4 y 5, como el valor NULL (por aquellas tareas que no tienen coordinador) aparecen más de una vez. Esto se debe a que tanto el coordinador 4 y 5 coordinan más de una tarea y más de una tarea no tiene coordinador.

En determinadas circunstancias puede no ser necesario que el mismo resultado aparezca varias veces. Para eliminar las filas repetidas se debe utilizar la cláusula DISTINCT.

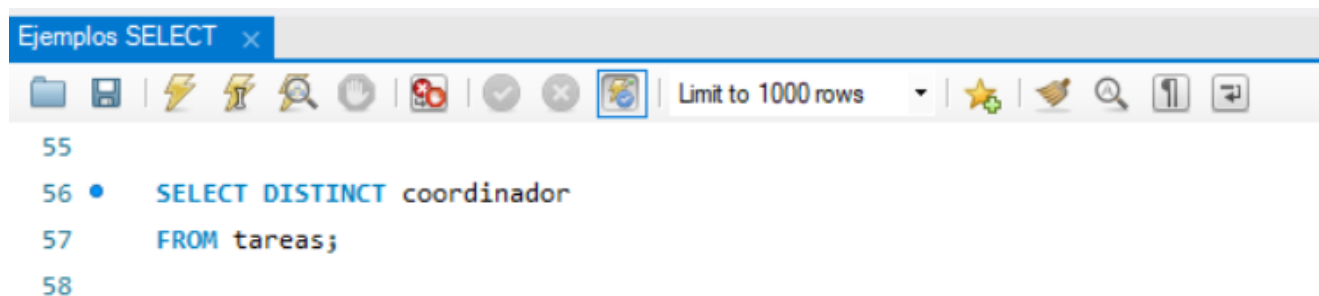


Figura 28. Consulta sobre los coordinadores de las tareas con la cláusula DISTINCT.



coordinador
NULL
4
5

Figura 29. Resultado de la consulta sobre los coordinadores de las tareas con la cláusula DISTINCT.

#### 4.4.4 Consultas con valores NULOS

Un dato que tiene **valor nulo (NULL)** significa que el dato no existe o se desconoce su existencia. Como hemos visto en documentos anteriores, hay que evitar los valores nulos en la base de datos tanto como sea posible para evitar inconsistencias.

En SQL tenemos dos cláusulas para trabajar con columnas que tengan valores nulos, ellas son las siguientes:

- **IS NULL:** esta cláusula realiza una comparación booleana. Retorna verdadero (TRUE) si el valor proporcionado es nulo (NULL) y falso (FALSE) si el valor proporcionado no es nulo.
- **IS NOT NULL:** esta cláusula realiza una comparación booleana. Retorna verdadero (TRUE) si el valor proporcionado no es nulo y falso (FALSE) si el valor proporcionado es nulo (NULL).

Se pueden usar en una sentencia SELECT, INSERT, UPDATE o DELETE y debe utilizarse en la cláusula WHERE.

**Ejemplo 10:** listar todos los datos para aquellas tareas que no tengan coordinador.

```

Ejemplos SELECT x paisas lugares
Limit to 1000 rows
58
59 -- Ejemplo 10: listar todos los datos para aquellas tareas que no tengan coordinador.
60
61 • SELECT *
62 FROM tareas
63 WHERE coordinador IS NULL;

```

Figura 30. Consulta sobre las tareas para aquellas tareas que no tengan coordinador.

	id tarea	codigo	nombre	fecha_inicio	fecha_fin	horasmin	horasmax	presupuesto	coordinador	lugar
▶	3	TAR0003	COVID - 19 TAR0003	2020-04-01	NULL	15.00	50.00	2000000.00	NULL	3
	5	TAR0005	COVID - 19 TAR0005	2020-02-10	NULL	15.00	60.00	1000000.00	NULL	11
	7	TAR0007	COVID - 19 TAR0007	2020-02-15	NULL	8.00	35.00	1000000.00	NULL	6
	10	TAR0010	COVID - 19 TAR0010	2020-02-23	NULL	8.00	35.00	1500000.00	NULL	9

Figura 31. Resultado de la consulta sobre las tareas para aquellas tareas que no tengan coordinador.

**Ejemplo 11:** listar el nombre y el coordinador para aquellas tareas que sí tengan coordinador.

```

Ejemplos SELECT x tareas
Limit to 1000 rows
65 -- Ejemplo 11: listar el nombre y el coordinador para aquellas tareas que sí tengan coordinador.
66
67 • SELECT nombre, coordinador
68 FROM tareas
69 WHERE coordinador IS NOT NULL;

```

Figura 32. Consulta sobre el nombre y el coordinador para aquellas tareas que sí tengan coordinador.

Result Grid			Filter Rows:
	nombre	coordinador	
▶	COVID - 19 TAR0001	4	
	COVID - 19 TAR0004	4	
	COVID - 19 TAR0008	4	
	COVID - 19 TAR0002	5	
	COVID - 19 TAR0006	5	
	COVID - 19 TAR0009	5	

Figura 33. Resultado de la consulta sobre el nombre y el coordinador para aquellas tareas que sí tengan coordinador.

#### 4.4.5 Cláusula ORDER BY

Según la teoría del modelo relacional, las filas de las tablas no están ordenadas. En el ejemplo 11, en la Figura 33, se puede observar que las filas del resultado no cumplen ningún criterio orden.

Es decir, en la tabla temporal devuelta primero aparece la tarea con nombre 'COVID - 19 TAR0008' y luego las tareas con nombre 'COVID - 19 TAR0002' y 'COVID - 19 TAR0006', si bien aparecen primero todas las tareas del coordinador 4 y luego las del coordinador 5, pero ningún criterio de orden fue especificado.

En algunas situaciones es necesario presentar las filas que retorna la consulta ordenadas por algún criterio. La cláusula **ORDER BY** permite ordenar las tuplas resultantes por la columna indicada.

**Ejemplo 12:** listar el nombre y el coordinador para aquellas tareas que sí tengan coordinador, ordenadas por el nombre de la tarea.

```

Ejemplos SELECT x tareas
71 -- Ejemplo 12: listar el nombre y el coordinador para aquellas tareas que sí tengan coordinador, ordenadas por el nombre de la tarea.
72
73 • SELECT nombre, coordinador
74 FROM tareas
75 WHERE coordinador IS NOT NULL
76 ORDER BY nombre;
77

```

Figura 34. Consulta sobre el nombre y el coordinador para aquellas tareas que sí tengan coordinador, ordenadas por el nombre de la tarea.

Result Grid			Filter Rows:
	nombre	coordinador	
▶	COVID - 19 TAR0001	4	
	COVID - 19 TAR0002	5	
	COVID - 19 TAR0004	4	
	COVID - 19 TAR0006	5	
	COVID - 19 TAR0008	4	
	COVID - 19 TAR0009	5	

*Figura 35. Resultado de la consulta sobre el nombre y el coordinador para aquellas tareas que sí tengan coordinador, ordenadas por el nombre de la tarea.*

En la Figura 35, se puede observar que ahora las filas del resultado de la consulta aparecen ordenadas según el criterio especificado en la cláusula ORDER BY, es decir, están ordenadas por la columna nombre.

Por defecto, las filas de una consulta aparecen ordenadas en orden ascendente (ORDER BY ASC), por lo que la palabra clave ASC puede obviarse. En cambio, si queremos que el orden sea descendente, debemos especificar la palabra clave DESC luego de especificar el nombre de la columna o columnas que formarán el criterio de ordenamiento de las filas de la tabla temporal.

#### 4.4.6 Operadores IN y NOT IN

El operador **IN** retorna TRUE si el operando es igual a uno de la lista de valores. La lista de valores puede ser simplemente especificada o ser proporcionada por la declaración de una cláusula SELECT separada (esto se llama una subconsulta).

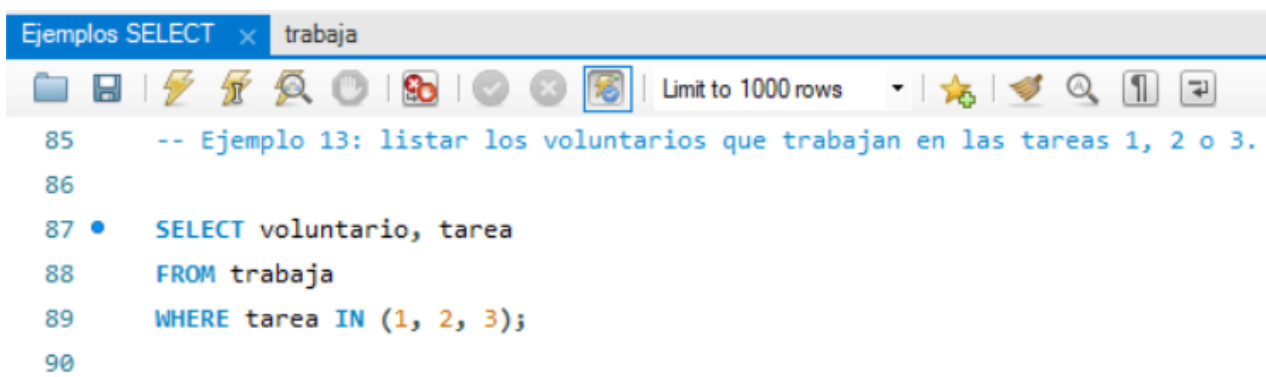
Se puede definir como la operación de **pertenencia de conjuntos**.

Se puede usar en una sentencia SELECT, INSERT, UPDATE o DELETE y debe utilizarse en la cláusula WHERE.

El operador **NOT IN** invierte el valor del operador IN.

En este documento solo se abordarán ejemplos sencillos para los operadores IN o NOT IN, no se usarán subconsultas.

**Ejemplo 13:** listar los voluntarios que trabajan en las tareas 1, 2 o 3.

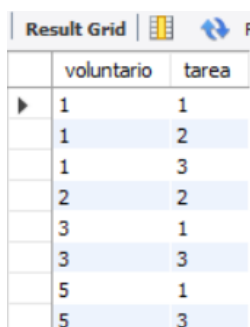


```

85  -- Ejemplo 13: listar los voluntarios que trabajan en las tareas 1, 2 o 3.
86
87  •  SELECT voluntario, tarea
88      FROM trabaja
89      WHERE tarea IN (1, 2, 3);
90

```

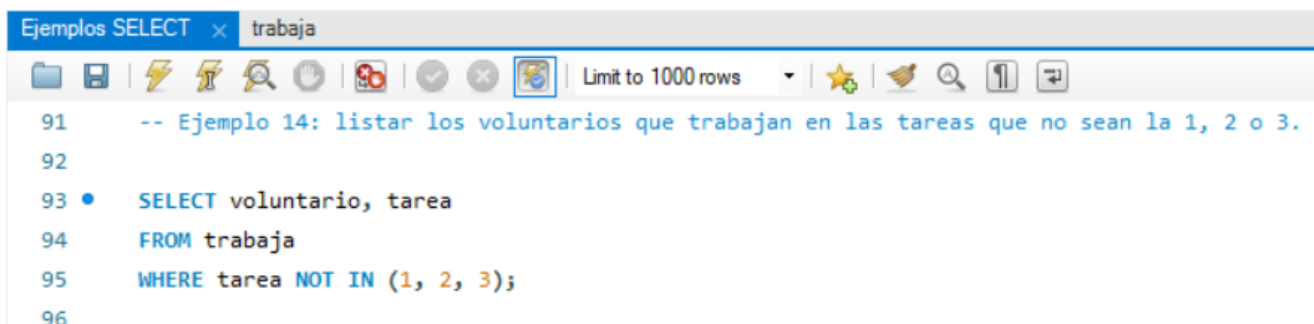
Figura 36. Consulta sobre los voluntarios que trabajan en las tareas 1, 2 o 3.



	voluntario	tarea
▶	1	1
	1	2
	1	3
	2	2
	3	1
	3	3
	5	1
	5	3

Figura 37. Resultado de la consulta sobre los voluntarios que trabajan en las tareas 1, 2 o 3.

**Ejemplo 14:** listar los voluntarios que trabajan en las tareas que no sean la 1, 2 o 3.



```

91  -- Ejemplo 14: listar los voluntarios que trabajan en las tareas que no sean la 1, 2 o 3.
92
93  •  SELECT voluntario, tarea
94      FROM trabaja
95      WHERE tarea NOT IN (1, 2, 3);
96

```

Figura 38. Consulta sobre los voluntarios que trabajan en las tareas que no sean la 1, 2 o 3.

	voluntario	tarea
4	5	
5	4	
6	7	
7	6	
8	9	
9	8	
10	8	
11	10	

Figura 39. Resultado de la consulta sobre los voluntarios que trabajan en las tareas que no sean la 1, 2 o 3.

## 4.5 Operador LIKE

Cuando una cadena de caracteres se compara por igualdad (=) el resultado será verdadero (TRUE) si ambas cadenas son iguales, falso (FALSE) en caso contrario.

Cuando no se desea comparar por igualdad, si no, por otras características como la letra de comienzo o final de una cadena, o una subcadena contenida en una cadena de caracteres, parafraseando a Bertone y Thomas (2011), una de las características más interesantes y que dotan a SQL de gran potencia en la generación de consultas que relacionan cadenas de caracteres, resulta del uso del operador de comparación **LIKE**.

**Ejemplo 15:** listar a todos los voluntarios cuyo nombre comience con la letra G.

```

Ejemplos SELECT x voluntarios
-- Ejemplo 15: listar a todos los voluntarios cuyo nombre comience con la letra G.
SELECT nombre, apellido
FROM voluntarios
WHERE nombre LIKE 'G%';

```

Figura 40. Consulta sobre los voluntarios cuyo nombre comience con la letra G.

	nombre	apellido
▶	GIMENA	FERNANDEZ
	GERONIMO	SOSA
	GRACIELA	MANCUSO

Figura 41. Resultado de la consulta sobre los voluntarios cuyo nombre comience con la letra G.

El operador LIKE se combina con el carácter reservado '%'. Se compara la columna nombre de la tabla voluntarios contra el string que empieza con G y continúa con cualquier substring (ese es el comportamiento del carácter %, el cual considera válida a partir de la posición donde aparece, cualquier cadena de caracteres, inclusive la cadena vacía).

Entonces, cualquier fila cuya columna nombre comience con G será presentada en el resultado final.

Existe además otro carácter reservado, '\_', el guión bajo sustituye sólo el carácter del lugar donde aparece.

**Ejemplo 16:** listar a todos los voluntarios cuyo apellido contenga la subcadena 'AN'.

```

Ejemplos SELECT x voluntarios
Limit to 1000 rows
Find
102
103 -- Ejemplo 16: listar a todos los voluntarios cuyo apellido contenga la subcadena 'AN'.
104
105 • SELECT nombre, apellido
106 FROM voluntarios
107 WHERE apellido LIKE '%AN%';

```

Figura 42. Consulta sobre los voluntarios cuyo apellido contenga la subcadena 'AN'.



	nombre	apellido
▶	GIMENA	FERNANDEZ
	GRACIELA	MANCUSO
	SERGIO	MANRIQUE
	MARCOS	ANDRADE

Figura 43. Resultado de la consulta sobre los voluntarios cuyo apellido contenga la subcadena 'AN'.

El carácter reservado '%' indica que la cadena puede comenzar y terminar con cualquier string, y además en algún lugar de la misma debe aparecer el substring 'AN'.

**Ejemplo 17:** listar el nombre de los voluntarios que tengan DNI que comience con 23 millones.

```

Ejemplos SELECT x voluntarios
-- Ejemplo 17: listar el nombre de los voluntarios que tengan DNI que comience con 23 millones.
SELECT dni, nombre, apellido
FROM voluntarios
WHERE dni LIKE '23%';

```

Figura 45. Consulta sobre los voluntarios que tengan DNI que comience con 23 millones.

	dni	nombre	apellido
▶	23567890	GIMENA	FERNANDEZ
	23456754	LUCRECIA	MARTINEZ

Figura 46. Resultado de la consulta sobre los voluntarios que tengan DNI que comience con 23 millones.

**Ejemplo 18:** listar el nombre de los voluntarios para los cuales la tercera letra de su nombre sea una M.

```

Ejemplos SELECT x voluntarios
-- Ejemplo 18: listar el nombre de los voluntarios para los cuales la tercera letra de su nombre sea una M.
SELECT nombre, apellido
FROM voluntarios
WHERE nombre LIKE '__M%';

```

Figura 47. Consulta sobre los voluntarios para los cuales la tercera letra de su nombre sea una M.

Result Grid		Filter Row
	nombre	apellido
▶	GIMENA	FERNANDEZ
	PAMELA	GARAY

Figura 48. Resultado de la consulta sobre los voluntarios para los cuales la tercera letra de su nombre es M.

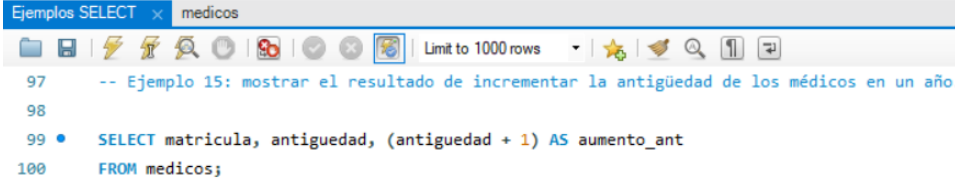
En la consulta del ejemplo 18 se observa que la cadena comienza con dos guiones bajos que representan a las dos primeras letras del nombre, la tercera letra es una M y luego la cadena puede continuar con cualquier substring, ya que se completa con el carácter especial %.

Se debe tener en cuenta que las funciones para cadenas son implementadas en cada SGBD de forma particular, por lo cual, no forman parte del estándar SQL2 (SQL92). Siempre se recomienda leer la documentación del mismo antes de trabajar con datos que incluyan fechas.

## 4.6 Operaciones válidas en el SELECT

Los columnas proyectadas en el SELECT de una consulta SQL pueden tener asociadas operaciones válidas para sus dominios de datos.

**Ejemplo 19:** mostrar el resultado de incrementar la antigüedad de los médicos en un año.



```

97  -- Ejemplo 15: mostrar el resultado de incrementar la antigüedad de los médicos en un año.
98
99  • SELECT matricula, antigüedad, (antigüedad + 1) AS aumento_ant
100 FROM medicos;
101

```

Figura 49. Consulta sobre el resultado de incrementar la antigüedad de los médicos en un año.



	matricula	antigüedad	aumento_ant
▶	52869	17.00	18.00
	68589	12.00	13.00
	42125	22.00	23.00
	67529	14.00	15.00
	55897	15.00	16.00
	71256	11.00	12.00

Figura 50. Resultado de la consulta sobre el resultado de incrementar la antigüedad de los médicos en un año.

Esta consulta retorna a la matrícula de cada médico, el valor actual de su antigüedad y el valor correspondiente de incrementar la antigüedad en un año, pero no modifica el valor de la columna como sucedió en el ejemplo 2 del apartado 3.2 de este documento, cuando sí se modifica la antigüedad usando la cláusula UPDATE.

En las Figura 40 y 41, vemos que el nombre de la columna que tiene el cálculo se renombra como “aumento\_ant” por medio de la cláusula **AS**, que permite renombrar o dar un **alias** tanto a columnas como a tablas, que solo tiene validez en la tabla temporal del resultado de la consulta.

Sobre las columnas definidas en un SELECT se pueden realizar cualquiera de las operaciones básicas definidas para su dominio. Cada SGBD en particular define su mapa de operaciones, por lo que siempre se recomienda leer su documentación.

## 4.7 Cláusula JOIN

Hasta el momento, se presentaron consultas que solamente involucran a una sola tabla del esquema físico de la base de datos. En esta sección se verán algunas de las cláusulas existentes en el lenguaje SQL para hacer consultas sobre dos o más tablas de forma simultánea.

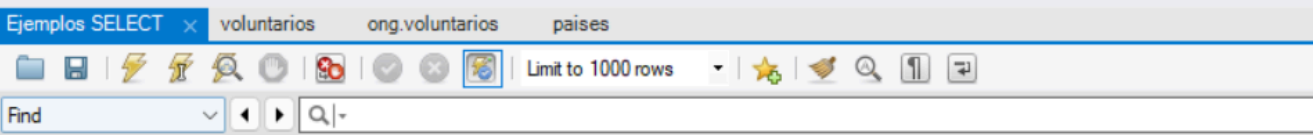
Para ello se usa la cláusula **JOIN**, que permite realizar distintas reuniones o tipos de productos sobre dos o más tablas.

### 4.7.1 Cláusula CROSS JOIN

El **producto cartesiano** en SQL es equivalente al producto cartesiano entre conjuntos. Se aplica a dos tablas de la base de datos, vinculando a cada fila de una tabla con cada una de las filas de la otra tabla.

Para realizar un producto cartesiano basta con poner en la cláusula FROM dos o más tablas separadas por comas. El SQL2 también permite usar la cláusula **CROSS JOIN**, pero no todos los SGBD implementan estas palabras claves, dejando como única opción el uso de las comas entre las tablas para llevarlo a cabo.

**Ejemplo 20:** listar el dni, nombre y apellido de los voluntarios y el nombre del país de origen.



```

127  -- Ejemplo 20: listar el dni, nombre y apellido de los voluntarios y el nombre del país de origen.
128
129  •  SELECT dni, v.nombre AS nombre_voluntario, apellido, p.nombre AS nombre_pais
130     FROM voluntarios v, países p
131     WHERE v.pais = p.idpais;
  
```

Figura 51. Consulta sobre el dni, nombre y apellido de los voluntarios y el nombre del país de origen.

Result Grid				
Filter Rows:				
Export:				
	dni	nombre_voluntario	apellido	nombre_pais
▶	23567890	GIMENA	FERNANDEZ	ARGENTINA
	32567897	LUCAS	CAPONI	ARGENTINA
	38789123	MARIA INES	LORENZETTI	ARGENTINA
	21897678	GERONIMO	SOSA	ARGENTINA
	18654345	GRACIELA	MANCUSO	ARGENTINA
	26897564	LUCAS	PEREYRA	ARGENTINA
	23456754	LUCRECIA	MARTINEZ	ARGENTINA
	21675321	PAMELA	GARAY	ARGENTINA
	32809343	SERGIO	MANRIQUE	ARGENTINA
	33454872	AGUSTINA	MASIAS	ARGENTINA
	24678906	MARCOS	ANDRADE	ARGENTINA

Figura 52. Resultado de la consulta sobre el dni, nombre y apellido de los voluntarios y el nombre del país de origen.

Para resolver esta consulta, se debe notar que la información requerida se encuentra definida en dos tablas. Entonces, es necesario agrupar la información de las dos tablas para poder satisfacer el pedido.

Para agrupar las tuplas de ambas tablas, se utiliza un producto cartesiano entre las tablas voluntarios y países.

La cláusula WHERE es necesaria para filtrar aquellas filas que tienen sentido, es decir, aquellas combinaciones en donde el valor de la FK de la columna país de la tabla de voluntarios coincida con el valor de la PK de la columna idpais de la tabla países. Asimismo, en la consulta se puede observar que fue necesario definir un alias para las columnas nombre de ambas tablas, ya que se encontraban repetidas en ambas tablas. Además, se definió un alias para cada tabla, en este caso, se puede obviar la cláusula AS colocando el mismo a continuación del nombre de cada tabla.

#### 4.7.2 Productos naturales

El producto cartesiano relaciona cada fila de la primera tabla con cada una de las filas de la segunda tabla, y luego evalúa el predicado de la cláusula WHERE. Esto hace que el tamaño de las tablas temporales puede ser muy grande si las tablas tienen muchos registros, con lo cual es el más ineficiente de los productos.

La definición del **producto natural** surge a partir de la necesidad de evitar esta situación. El producto natural directamente reúne las filas de la primera tabla que se relacionan con la segunda tabla, descartando las filas no relacionadas. Esto conlleva una gran ventaja en

cuanto a performance, dado que no se generan filas que luego se descartan evaluando un predicado, sino que sólo se generan las filas necesarias para devolver el resultado de la consulta.

Las sentencias disponibles al efecto son: INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN.

A continuación, veremos algunos de los productos definidos en el lenguaje SQL. En documentos posteriores se abordará la totalidad de tipos de JOIN posibles en SQL2 (SQL92).

#### 4.7.2.1 Cláusula NATURAL JOIN

La cláusula **NATURAL JOIN** se usa cuando las columnas por las cuales se referencian las tablas tienen el mismo nombre. Hay que tener especial cuidado cuando se usa por si las tablas comparten más de una columna con el mismo nombre.

No todos los SGBD lo implementan.

La sentencia sería de la forma siguiente:

```
SELECT lista de columnas  
FROM tabla1 NATURAL JOIN tabla2  
[WHERE predicado];
```

En el predicado no deben especificarse los campos por los cuales se reúnen ambas tablas, ya que esta cláusula lo hace de forma automática. Solo deben especificarse otros predicados necesarios para filtrar las filas del resultado.

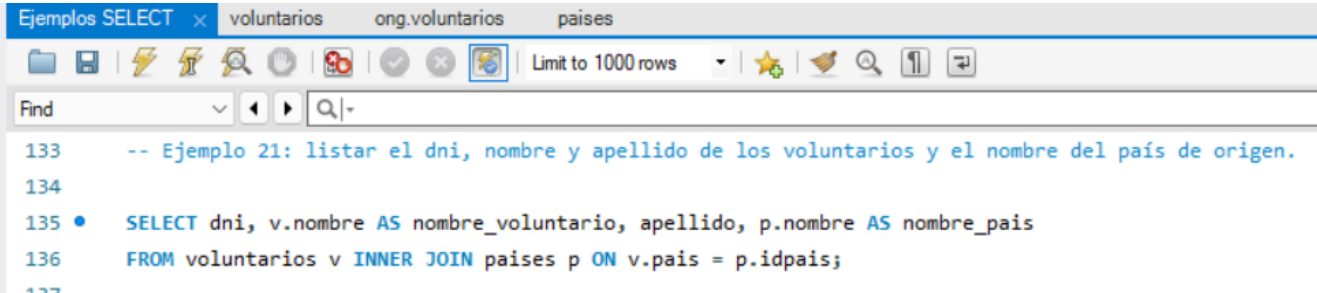
En el ejemplo del **Caso de Estudio ONG Médicos sin Fronteras**, las FK no tienen el mismo nombre que las PK a las cuales referencian, por lo cual, no abordaremos un ejemplo de este producto.

La forma utilizada para nombrar PK y FK utilizada para el caso es una buena práctica, ya que, fuerza a usar otros tipos de reuniones (JOIN) entre tablas, para evitar los inconvenientes que se plantearon al inicio de esta sección.

#### 4.7.2.2 Cláusula INNER JOIN

La cláusula **INNER JOIN** es el producto natural con condición, es decir, reúne directamente las filas de la primera tabla que se relacionan con la segunda tabla, descartando las filas no relacionadas, pero debe indicarse la condición de reunión entre las tablas por medio de la cláusula ON.

**Ejemplo 21:** listar el dni, nombre y apellido de los voluntarios y el nombre del país de origen.



```

Ejemplos SELECT x voluntarios ong.voluntarios paises
Limit to 1000 rows
Find
133 -- Ejemplo 21: listar el dni, nombre y apellido de los voluntarios y el nombre del país de origen.
134
135 • SELECT dni, v.nombre AS nombre_voluntario, apellido, p.nombre AS nombre_pais
136 FROM voluntarios v INNER JOIN paises p ON v.pais = p.idpais;
137

```

Figura 53. Consulta sobre el dni, nombre y apellido de los voluntarios y el nombre del país de origen.

Result Grid				
Filter Rows:				
Export:				
	dni	nombre_voluntario	apellido	nombre_pais
▶	23567890	GIMENA	FERNANDEZ	ARGENTINA
	32567897	LUCAS	CAPONI	ARGENTINA
	38789123	MARIA INES	LORENZETTI	ARGENTINA
	21897678	GERONIMO	SOSA	ARGENTINA
	18654345	GRACIELA	MANCUSO	ARGENTINA
	26897564	LUCAS	PEREYRA	ARGENTINA
	23456754	LUCRECIA	MARTINEZ	ARGENTINA
	21675321	PAMELA	GARAY	ARGENTINA
	32809343	SERGIO	MANRIQUE	ARGENTINA
	33454872	AGUSTINA	MASIAS	ARGENTINA
	24678906	MARCOS	ANDRADE	ARGENTINA

Figura 54. Resultado de la consulta sobre el dni, nombre y apellido de los voluntarios y el nombre del país de origen.

## 4.8 Operaciones de conjuntos

En este documento sólo veremos las dos operaciones más básicas sobre conjuntos: la unión y la intersección.

En documentos posteriores se abordará la totalidad de operaciones de conjuntos posibles en SQL2 (SQL92).

### 4.8.1 Unión

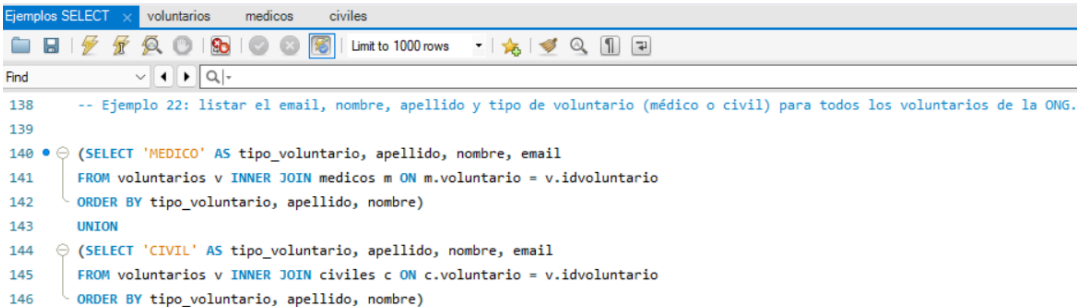
La cláusula **UNION** es equivalente a la unión matemática de conjuntos.

Se aplica a dos tablas o, generando una nueva tabla cuyo contenido es el contenido de cada una de las filas de las tablas originales involucradas.

Debe tenerse en cuenta que los dos conjuntos a unir deben ser compatibles. Esto es, unir dos conjuntos que tengan la misma cantidad de columnas, y además la *i*-ésima columna de la primera tabla y la *i*-ésima columna de la segunda tabla deben tener el mismo dominio de datos (*i*: 1..*n*). Caso contrario, el resultado obtenido no representa ninguna información útil.

En SQL92 la UNION se resuelve de manera tal que las filas duplicadas en ambas subconsultas solo quedan una vez en el resultado final. Para obtener las tuplas duplicadas se puede utilizar la cláusula **UNION ALL**.

**Ejemplo 22:** listar el email, nombre, apellido y tipo de voluntario (médico o civil) para todos los voluntarios de la ONG.



```

138  -- Ejemplo 22: listar el email, nombre, apellido y tipo de voluntario (médico o civil) para todos los voluntarios de la ONG.
139
140  (SELECT 'MEDICO' AS tipo_voluntario, apellido, nombre, email
141   FROM voluntarios v INNER JOIN medicos m ON m.voluntario = v.idvoluntario
142   ORDER BY tipo_voluntario, apellido, nombre)
143  UNION
144  (SELECT 'CIVIL' AS tipo_voluntario, apellido, nombre, email
145   FROM voluntarios v INNER JOIN civiles c ON c.voluntario = v.idvoluntario
146   ORDER BY tipo_voluntario, apellido, nombre)

```

Figura 55. Consulta sobre el email, nombre, apellido y tipo de voluntario (médico o civil) para todos los voluntarios de la ONG.



Result Grid   Filter Rows:   Export:   Wrap Cell Content:				
	tipo_voluntario	apellido	nombre	email
▶	MEDICO	FERNANDEZ	GIMENA	gimena_fern@hotmail.com
	MEDICO	LORENZETTI	MARIA INES	mariaineslorenzetti@hotmail.com
	MEDICO	CAPONI	LUCAS	lucas_caponi@gmail.com
	MEDICO	SOSA	GERONIMO	geronimososa@gmail.com
	MEDICO	MANCUSO	GRACIELA	gracielamancuso@hotmail.com
	MEDICO	PEREYRA	LUCAS	lucaspereyra@hotmail.com
	CIVIL	MARTINEZ	LUCRECIA	lucreciamartinez@hotmail.com
	CIVIL	GARAY	PAMELA	pamelagaray@hotmail.com
	CIVIL	MANRIQUE	SERGIO	sergiomanrique@hotmail.com
	CIVIL	MASIAS	AGUSTINA	agustina_masias@gmail.com
	CIVIL	ANDRADE	MARCOS	marcos_andrade@gmail.com

Figura 56. Resultado de la consulta sobre el email, nombre, apellido y tipo de voluntario (médico o civil) para todos los voluntarios de la ONG.

#### 4.8.2 Intersección

La cláusula **INTERSECT** es equivalente a la intersección matemática de conjuntos.

Se aplica a dos tablas o, generando una nueva tabla cuyo contenido es el contenido de cada una de las filas de las tablas originales involucradas.

Debe tenerse en cuenta que los dos conjuntos a intersecar deben ser compatibles. Es decir, deben ser dos conjuntos que tengan la misma cantidad de columnas, y además la *i*-ésima columna de la primera tabla y la *i*-ésima columna de la segunda tabla deben tener el mismo dominio de datos (*i*: 1..*n*). Caso contrario, el resultado obtenido no representa ninguna información útil.

No todos los SGBD implementan la cláusula **INTERSECT**, con lo que puede simularse con la cláusula **INNER JOIN**. Por esta razón, no veremos ningún ejemplo de intersección en MySQL Server.



## Bibliografía

Bertone, R., & Thomas, P. (2011). *Introducción a las bases de datos: fundamentos y diseño*. Pearson Educación.

Elmasri, R., & Navathe, S. B. (2007). *Fundamentos de sistemas de bases de datos, 5a ed.* Pearson Educación.

*MySQL 5.0 Reference Manual*. (n.d.). MySQL Community Downloads. Recuperado el 30 de septiembre de 2023, desde

<https://downloads.mysql.com/docs/refman-5.0-es.pdf>

Silberschatz, A., Korth, H. F., & Sudarshan, S. (2002). *Fundamentos de bases de datos* (L. Grau Fernández, Ed.). McGraw Hill.