

Implementación y manipulación de bases de datos relacionales

**Breve descripción:**

Una vez diseñadas, modeladas e implementadas las bases de datos, es preciso darles el uso adecuado según el lenguaje estándar y las herramientas necesarias en el proceso de desarrollo. Este componente centra sus esfuerzos en presentar las técnicas para la manipulación de una base de datos.

**Abril 2024**

Tabla de contenido

[Introducción 1](#_Toc165359934)

[1. Interface de línea de comandos de MySQL 2](#_Toc165359935)

[1.1. Aplicación cliente de MySQL 2](#_Toc165359936)

[1.2. Prueba de consultas básicas 3](#_Toc165359937)

[1.3. SQL – “Standard Query Language” 4](#_Toc165359938)

[2. SQL para la creación y eliminación de bases de datos – DDL 6](#_Toc165359939)

[2.1. SQL para la creación y eliminación de tablas 9](#_Toc165359940)

[2.2. Claves foráneas 9](#_Toc165359941)

[2.3. Mostrar y eliminar tablas y bases de datos 15](#_Toc165359942)

[3. SQL para la manipulación de datos – DML 18](#_Toc165359943)

[3.1. Inserción de registros 18](#_Toc165359944)

[3.2. Edición de registros 21](#_Toc165359945)

[3.3. Borrado de registros 24](#_Toc165359946)

[4. SQL para consulta de datos – DML 27](#_Toc165359947)

[4.1. Funciones en MySQL 27](#_Toc165359948)

[4.2. Operadores 33](#_Toc165359949)

[4.3. Ordenar resultados 37](#_Toc165359950)

[4.4. Listar y limitar resultados 38](#_Toc165359951)

[4.5. Agrupar filas 40](#_Toc165359952)

[5. Consulta multitablas – DML 42](#_Toc165359953)

[5.1. Combinación interna (INNER JOIN) 44](#_Toc165359954)

[5.2. Combinación externa 46](#_Toc165359955)

[Síntesis 49](#_Toc165359956)

[Material complementario 50](#_Toc165359957)

[Glosario 51](#_Toc165359958)

[Referencias bibliográficas 52](#_Toc165359959)

[Créditos 53](#_Toc165359960)

Introducción

Las siglas SQL, vienen del inglés “Structured Query Language”, es un lenguaje estándar que permite manipular los datos de una base de datos relacional. Casi todos los Sistemas de Gestión de Bases de Datos Relacionales (SGBD) implementan este lenguaje y mediante él se realiza todo tipo de actividad en la base de datos. En este componente formativo, se hace una presentación del lenguaje SQL, enfocándose en las sentencias de consulta de datos, empleando la interfaz de consola de línea de comandos, con el fin de fortalecer las técnicas de administración remota. Finalmente, se presentará la interfaz de administración gráfica para hacer las mismas operaciones.

# Interface de línea de comandos de MySQL

Al iniciar a trabajar con el SGBD, existen varias maneras de enlazar una comunicación con el servidor de MySQL. Generalmente en el “software”, se usará una API para hacer las consultas y otras operaciones con el servidor. Por ejemplo, en PHP, esta API está integrada con el lenguaje.

En este punto se usará MySQL de forma directa, por medio de un cliente ejecutándose desde una consola (una ventana de comandos en Windows). En otras secciones se explicarán las distintas API.

## Aplicación cliente de MySQL

Primero, se deben tener en cuenta las siguientes instrucciones, para conectarse a la base de datos a través del cliente de línea de comandos:

* **Paso 01**. Debe abrir inicio y buscar MySQL o teclear "mysql" y se hace clic en “MySQL Command Line Client”.
* **Paso 02**. Ingresa la contraseña de “root” usada en el proceso de instalación de MySQL.
* **Paso 03**. Una vez autenticado en la “interfaz” comando de línea, vemos este resultado.

En adelante, los resultados de “interfaz” de línea de comandos se presentarán como se presenta a continuación:

|  |
| --- |
| Bloque de código |
| Enter password: \*\*\*\*\*\*\*\*\*\*\*  Welcome to the MySQL monitor. Commands end with; or \g.  Your MySQL connection id is 90  Server version: 8.0.25 MySQL Community Server – GPL  Copyright (c) 2000, 2021, Oracle and/or its affiliates.  Oracle is a registered trademark of Oracle Corporation and/or its  affiliates. Other names may be trademarks of their respective  owners.  Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.  mysql> |

Para salir de la consola de comando, se usa el comando “quit” (en mayúscula o minúscula).

|  |
| --- |
| Bloque de código |
| mysql> QUIT  Bye  C:\mysql\bin> |

## Prueba de consultas básicas

Ya se conoce cómo entrar y salir del cliente MySQL, y se tiene la posibilidad de realizar consultas, ahora, lo más fácil es consultar varias variables del sistema o el valor de varias funcionalidades de MySQL. Para realizar esta clase de consultas se utiliza la sentencia “SQL SELECT”, por ejemplo:

|  |
| --- |
| Bloque de código |
| mysql> SELECT VERSION (), CURRENT\_DATE;  +-----------+--------------+  | VERSION () | CURRENT\_DATE |  +-----------+--------------+  | 8.0.25 | 2021-06-12 |  +-----------+--------------+  1 row in set (0.02 sec)  mysql> |

“SELECT” es la sentencia SQL para elegir datos de bases de datos, además se puede utilizar, como en esta situación, para consultar cambiantes del sistema o resultados de funcionalidades. En este caso, se ha consultado el resultado de la función “VERSION”, se sabe que es una función porque tiene paréntesis “()”, y de la variable “CURRENT\_DATE”.

Toda sentencia debe finalizar con punto y coma (;), luego debe dar enter.

## SQL – “Standard Query Language”

En el nivel teórico, existen dos lenguajes para el uso de bases de datos:

* **DDL (“Data Definition Language”).** Es el lenguaje que se emplea para establecer bases de datos y tablas, y para transformar sus estructuras. Este lenguaje trabaja sobre ciertas tablas especiales llamadas diccionario de datos.
* **DML (“Data Manipulation Language”)**. Es lo que se usa para cambiar y obtener datos desde las bases de datos.

SQL abarca ambos lenguajes: DDL + DML

En este punto se explicará el proceso para convertir del modelo lógico relacional, al modelo físico, empleando sentencias SQL. Ya no se empleará (por ahora), MySQL “Workbench”, y se observarán las particularidades específicas de MySQL.

# SQL para la creación y eliminación de bases de datos – DDL

Cada grupo de entidades que tienen un modelo completo, forma una base de datos. Desde la perspectiva de SQL, una base de datos es únicamente un conjunto de relaciones (o tablas) y para organizarlas o diferenciarlas, se accede a ellas por su nombre. A nivel de sistema operativo, cada base de datos se almacena en un directorio distinto por MySQL. Por este motivo, crear una base de datos es un trabajo muy sencillo, claro que, a la hora de crearla, la base de datos estará desocupada, es decir, no almacenará ninguna tabla.

Es fundamental que mientras se estudia, se ejecuten las sentencias propuestas aquí, de esa manera se consolidarán los conocimientos de cada tema tratado. Se debe tomar el tiempo necesario para escribir en la consola cada una de las sentencias, es un ejercicio fundamental para asimilar el conocimiento que se está adquiriendo.

De la misma forma, se deben revisar los resultados que le da la ejecución, en caso de algún error en la sintaxis, la respuesta dará una idea de qué puede estar fallando. Ahora, se va a crear y manejar una base de datos de práctica, a la vez que se familiarice con la manera de trabajar de MySQL.

Para iniciar, se creará una base de datos únicamente para practicar, y se llamará “prueba”:

Se usa la sentencia “CREATE DATABASE” para crear una base de datos

|  |
| --- |
| Bloque de código |
| mysql> CREATE DATABASE prueba;  Query OK, 1 row affected (0.03 sec)  mysql> |

Se puede saber cuántas bases de datos se encuentran en nuestro sistema usando la sentencia “SHOW DATABASES”:

|  |
| --- |
| Bloque de código |
| mysql> SHOW DATABASES;  +--------------------+  | Database |  +--------------------+  | farmacia |  | information\_schema |  | mysql |  | performance\_schema |  | prueba |  | sys |  +--------------------+  6 rows in set (0.00 sec)  mysql> |

Desde ahora, en los siguientes puntos, se trabajará con la base de datos creada, por lo cual, la elegirá como base de datos por defecto. Esto nos evitará poner el nombre de la base de datos en consultas. Para escoger una base de datos se emplea el comando USE, que no es puntualmente una sentencia SQL, sino una opción de MySQL:

|  |
| --- |
| Bloque de código |
| mysql> USE prueba;  Database changed  mysql> |

Para borrar una base de datos, se usa la sentencia “SQL DROP”. A continuación, se borrará la base de datos “prueba”, listaremos las existentes y la volveremos a crear:

|  |
| --- |
| Bloque de código |
| mysql> DROP DATABASE prueba;  Query OK, 0 rows affected (0.01 sec) |

Por último, Liste las bases de datos para comprobar que ya no existe la base de datos antes creada.

|  |
| --- |
| Bloque de código |
| mysql> SHOW DATABASES;  +--------------------+  | Database |  +--------------------+  | farmacia |  | information\_schema |  | mysql |  | performance\_schema |  | sys |  +--------------------+  5 rows in set (0.00 sec)  mysql> CREATE DATABASE prueba;  Query OK, 1 row affected (0.01 sec)  mysql> |

## SQL para la creación y eliminación de tablas

A continuación, la sentencia CREATE TABLE que se empleará para la creación de tablas. Esta sentencia tiene una sintaxis algo compleja, debido a que existen muchas opciones y se tienen varias posibilidades diferentes cuando se desea crear una tabla.

Para ampliar la información, lo invitamos a consultar el PDF “**Create Table**”, el cual se encuentra en la carpeta Anexos.

## Claves foráneas

En MySQL únicamente existe soporte para claves foráneas en tablas de tipo “InnoDB”.

“InnoDB” es el motor de almacenamiento por defecto de las últimas versiones de MySQL, no obstante, esto no imposibilita usarlas en otros tipos de tablas.

Primero se crea la tabla “persona”:

|  |
| --- |
| Bloque de código |
| mysql> CREATE TABLE personas (  -> INT AUTO\_INCREMENT PRIMARY KEY,  -> nombres VARCHAR(40)NOT NULL,  -> fecha DATE);  Query OK, 0 rows affected (0.13 sec)Query OK, 1 row affected (0.01 sec)  mysql> |

Si se quiere crear una base de datos que corresponda al modelo que se está presentando anteriormente, las sentencias para la tabla “telefonos” con la referencia a la tabla, sería:

|  |
| --- |
| Bloque de código |
| CREATE TABLE telefonos (  persona\_id INT NOT NULL REFERENCES personas (id) ON DELETE CASCADE ON UPDATE CASCADE,  numero VARCHAR(20) PRIMARY KEY,  tipo VARCHAR(10)  ); |

Se ha usado una definición a modo referencia para la columna “numero” de la tabla “telefonos”, señalando que es una clave foránea oportuna a la columna “id\_persona” de la tabla “persona” a través de la columna “persona\_id”. No obstante, aunque la sintaxis, se evidencia.

La expresión “DELETE CASCADE”, hace que, si se borra una fila de la tabla “persona” que tiene relacionados registros telefónicos a través de la columna “persona\_id” en la tabla “telefonos”, estos registros (localizados en la tabla “telefonos”) se borren si se borra el de la tabla “persona”; a este tipo de condiciones se le llama restricción de integridad referencial.

De igual forma “UPDATE CASCADE”, hace que, si el “id\_persona” en una fila de la tabla “persona” es actualizado a otro valor y existen referencias en la columna “persona\_id” de la tabla “telefonos” con el valor inicial, al cambiar el valor en el registro “id\_persona”, cambien también el valor en los registros en la columna “persona\_id” de la tabla “telefonos”. A este tipo de condiciones también se le llama restricción de integridad referencial.

Y ejecutándola en la consola, el resultado sería:

|  |
| --- |
| Bloque de código |
| mysql> CREATE TABLE personas (  -> id\_persona INT AUTO\_INCREMENT PRIMARY KEY,  -> nombres VARCHAR(40)NOT NULL,  -> fecha DATE);  Query OK, 0 rows affected (0.13 sec)  mysql> CREATE TABLE telefonos (  -> persona\_id INT NOT NULL REFERENCES personas (id) ON DELETE CASCADE ON UPDATE CASCADE,  -> numero VARCHAR(20) PRIMARY KEY,  -> tipo VARCHAR(10) );  Query OK, 0 rows affected (0.13 sec)  mysql> |

Miremos otra variante de la misma sintaxis definiendo explícitamente que se usará el motor “InnoDB” para que las reglas de restricción de integridad apliquen:

|  |
| --- |
| Bloque de código |
| CREATE TABLE personas2 (  id\_persona INT AUTO\_INCREMENT PRIMARY KEY,  nombres VARCHAR(45),  fecha DATE  )ENGINE=InnoDB; |

Y también para la tabla “telefonos2”:

|  |
| --- |
| Bloque de código |
| CREATE TABLE telefonos2 (  numero VARCHAR 20),  persona\_id INT NOT NULL,  KEY (numero),  FOREIGN KEY (persona\_id) REFERENCES personas2 (id\_persona)  ON DELETE CASCADE ON UPDATE CASCADE  ) ENGINE=InnoDB; |

Es necesario que la columna que posee una definición de clave foránea esté indexada KEY(numero). Sin embargo, esto no debe de generar preocupación, porque si no se hace de forma clara, MySQL lo hará de forma implícita.

Dicha forma precisa una clave foránea en la columna “persona\_id”, la cual hace referencia a la columna “id\_persona” de la tabla “personas2” (FOREIGN KEY (“persona\_id”) REFERENCES personas2). La definición contiene las tareas a efectuar cuando se excluya una fila en la tabla “personas2”.

#### ON DELETE <opcion>

define las acciones que deben de realizar en la tabla actual, cuando se elimine una fila en la tabla referenciada.

#### ON UPDATE <opcion>

es semejante, define las acciones que deben de realizar en la tabla actual, cuando se modifique o actualice la columna de una fila en la tabla referenciada.

Existen cinco opciones diferentes. A continuación, conozca cada una de ellas (MySQL 8.0 Reference Manual, 2021):

* **RESTRICT**. Esta opción imposibilita borrar o editar filas en la tabla referenciada si hay filas con el mismo valor de clave foránea.
* **CASCADE**. Borrar o modificar una clave en una fila en la tabla referenciada con un valor fijo de clave, conlleva borrar las filas con el mismo valor de clave foránea o cambiar los valores de esas claves foráneas.
* **SET NULL**. Borrar o editar una clave en una fila en la tabla referenciada con un valor fijo de clave, involucra determinar el valor NULL a las claves foráneas con el mismo valor.
* **NO ACTION**. Las claves foráneas no se alteran, ni se eliminan filas en la tabla que las contiene.
* **SET DEFAULT**. Borrar o alterar una clave en una fila en la tabla referenciada con un valor explícito, involucra asignar el valor por defecto a las claves foráneas con el mismo valor.

**Ejemplo**

|  |
| --- |
| Bloque de código |
| mysql> CREATE TABLE personas3 (  -> id\_persona INT AUTO\_INCREMENT PRIMARY KEY,  -> nombres VARCHAR(45),  -> fecha DATE )  -> ENGINE=InnoDB;  Query OK, 0 rows affected (0.16 sec)  mysql>  mysql> CREATE TABLE telefonos3 (  -> numero VARCHAR(20),  -> persona\_id INT NOT NULL,  -> KEY (numero),  -> FOREIGN KEY (persona\_id) REFERENCES personas3 (id\_persona)  -> ON DELETE RESTRICT ON UPDATE CASCADE  -> )ENGINE=InnoDB;  Query OK, 0 rows affected (0.19 sec)  mysql> |

Si se pretende borrar una fila de “personas3” con cierto valor de “id\_persona”, se provocará un error si existen filas en la tabla “telefonos3” en la columna “persona\_id” con el mismo valor. La fila de “personas3” no será eliminada, a menos que previamente se eliminen las filas que tienen el mismo valor de clave foránea en la tabla “telefonos3”, lo anterior debido a que se ha definido restricción DELETE RESTRICT.

Miremos otra variante de la misma sintaxis definiendo explícitamente que se usará el motor “InnoDB” para que las reglas de restricción de integridad apliquen.

Observe los datos de las tablas “personas3” y “telefonos3”:

|  |
| --- |
| Bloque de código – Datos de la tabla “personas3” |
| MariaDB [prueba]> SELECT \* FROM personas3;  +------------+-----------+------------+  | id\_persona | nombres | fecha |  +------------+-----------+------------+  | 1 | Fulanito | 2021-06-09 |  | 2 | Menganito | 2021-06-06 |  | 3 | Tulanito | 2021-06-23 |  | 4 | Miguelito | 2021-06-22 |  +------------+-----------+------------+  4 rows in set (0.000 sec)  MariaDB [prueba]> |

|  |
| --- |
| Bloque de código – Datos de la tabla relefono3 |
| MariaDB [prueba]> SELECT \* FROM telefonos3;  +------------+------------+  | numero | persona\_id |  +------------+------------+  | 2342479354 | 1 |  | 6443543534 | 1 |  | 578813343 | 3 |  | 2345224376 | 4 |  | 234235465 | 4 |  | 15698346 | 3 |  +------------+------------+  7 rows in set (0.000 sec)  MariaDB [prueba]> |

Si se intenta borrar la fila correspondiente a "Fulanito" se provocará un error, debido a que hay dos filas en “telefonos3” con el valor 1 en la columna “persona\_id”. Sí será posible borrar la fila correspondiente a "Menganito", debido a que no existe fila alguna en la tabla “telefonos3” con el valor 2 en la columna “persona\_id”. Si alteramos el valor de “id\_persona” en la fila conveniente a "Tulanito", por el valor 3, por ejemplo, se determinará el valor 3 a la columna “persona\_id” de las filas 3ª y 6ª de la tabla “telefonos3”.

## Mostrar y eliminar tablas y bases de datos

De vez en cuando es preciso eliminar una tabla, sea porque es más fácil crearla de nuevo que alterarla, o porque ya es prescindible.

Para borrar una tabla se usará la sentencia “DROP TABLE”.

|  |
| --- |
| Bloque de código |
| DROP TABLE [IF EXISTS] tbl\_name [, tbl\_name] ...  mysql> DROP TABLE telefonos2;  Query OK, 0 rows affected (0.75 sec)  mysql> |

Se pueden agregar las palabras “IF EXISTS” para impedir errores si la tabla a eliminar no existe.

|  |
| --- |
| Bloque de código |
| mysql> DROP TABLE telefonos5;  ERROR 1051 (42S02): Unknown table 'telefonos5'  mysql> DROP TABLE IF EXISTS telefonos5;  Query OK, 0 rows affected, 1 warning (0.00 sec)  mysql> |

De manera similar, se puede borrar bases de datos enteras, utilizando la sentencia “DROP DATABASE”. La sintaxis asimismo es muy sencilla, a continuación, se creará una base de datos, se creará una tabla y se borrará la base de datos.

|  |
| --- |
| Bloque de código |
| mysql> CREATE DATABASE borrame;  Query OK, 1 row affected (0.00 sec)  mysql> USE borrame;  Database changed  mysql> CREATE TABLE borrame (  -> id INT,  -> nombre VARCHAR(40)  -> );  Query OK, 0 rows affected (0.19 sec)  mysql> SHOW DATABASES;  +--------------------+  | Database |  +--------------------+  | borrame |  | mysql |  | prueba |  +--------------------+  4 rows in set (0.00 sec)  mysql> SHOW TABLES;  +-------------------+  | Tables\_in\_borrame |  +-------------------+  | borrame |  +-------------------+  1 row in set (0.00 sec)  mysql> DROP DATABASE IF EXISTS borrame;  Query OK, 1 row affected (0.13 sec)  mysql> DROP DATABASE IF EXISTS borrame;  Query OK, 0 rows affected, 1 warning (0.12 sec)  mysql> |

Lo invitamos a estudiar dos ejercicios que están en el documento anexo “Ejercicios Prácticos”. Ellos recogen la temática hasta ahora vista, con la diferencia que los ejercicios propuestos son más próximos a problemas de la vida real.

# SQL para la manipulación de datos – DML

Una base de datos sin datos no es muy útil, de manera que se verá cómo añadir, modificar o eliminar los datos que poseen las bases de datos.

A continuación, se creará una tabla con la que seguiremos ejemplificando la sintaxis del SQL:

|  |
| --- |
| Bloque de código |
| MariaDB [prueba]> CREATE TABLE gente (  -> nombre VARCHAR(50) KEY,  -> fecha DATE DEFAULT '2020-02-02',  -> edad INT DEFAULT 0);  Query OK, 0 rows affected (0.025 sec)  MariaDB [prueba]> |

## Inserción de registros

La forma más directa de implantar una fila nueva en una tabla es por medio de una sentencia “INSERT”.

De la manera más sencilla, en esta sentencia se debe indicar la tabla a la que se quiere agregar filas, y los valores de cada columna. Las columnas de tipo cadena o fechas tienen que estar entre comillas sencillas o dobles, esto no es obligatorio para las columnas numéricas, aunque también pueden estar entrecomilladas.

|  |
| --- |
| Bloque de código |
| MariaDB [prueba]> INSERT INTO gente (nombre, fecha, edad) VALUES ('Fulano','1974-04-12', 18);  Query OK, 1 row affected (0.05 sec)  MariaDB [prueba]> INSERT INTO gente (nombre, fecha, edad) VALUES ('Mengano','1978-06-15', 20);  Query OK, 1 row affected (0.04 sec)  MariaDB [prueba]> INSERT INTO gente (nombre, fecha, edad) VALUES  -> ('Tulano','2000-12-02', '22'),  -> ('Pegano','1993-02-10', '33');  Query OK, 2 rows affected (0.02 sec)  Records: 2 Duplicates: 0 Warnings: 0  MariaDB [prueba]> SELECT \* FROM gente;  +---------+------------+------+  | nombre | fecha | edad |  +---------+------------+------+  | Fulano | 1974-04-12 | 18 |  | Mengano | 1978-06-15 | 20 |  | Pegano | 1993-02-10 | 33 |  | Tulano | 2000-12-02 | 22 |  +---------+------------+------+  4 rows in set (0.08 sec)  MariaDB [prueba]> > |

Si no se requiere determinar un valor preciso para alguna columna, se puede determinar el valor por defecto indicado para esa columna cuando la tabla fue creada, usando la palabra “DEFAULT”:

|  |
| --- |
| Bloque de código |
| MariaDB [prueba]> INSERT INTO gente (nombre, fecha, edad) VALUES ('Perillo', DEFAULT, DEFAULT);  Query OK, 1 row affected (0.005 sec)  MariaDB [prueba]> |

Observe como “Perillo” tiene el valor por defecto definido en las columnas “fecha” y “edad”.

|  |
| --- |
| Bloque de código |
| MariaDB [prueba]> SELECT \* FROM gente;  +---------+------------+------+  | nombre | fecha | edad | |

El orden en la sentencia “INSERT” es muy importante, que conserve una correspondencia.

Finalmente, estamos listos para aprender cómo se actualizan los datos que se han registrado.

|  |
| --- |
| Bloque de código |
| INSERT INTO gente (nombre, fecha, edad) VALUES ('Fulano', '1974-04-12','18'); |

## Edición de registros

Tenemos la posibilidad de cambiar valores de las filas de una tabla utilizando la sentencia “UPDATE”. En su manera más fácil, los cambios se usan a cada una de las filas, y a las columnas que se les especifique (MySQL 8.0 Reference Manual, 2021).

|  |
| --- |
| Bloque de código |
| UPDATE [LOW\_PRIORITY] [IGNORE] tbl\_name  SET col\_name1=expr1 [, col\_name2=expr2 ...]  [WHERE where\_definition]  [ORDER BY ...]  [LIMIT row\_count] |

Estudiemos el siguiente ejemplo:

Que todas las personas de la tabla “gente” tengan la misma fecha utilizando esta sentencia “UPDATE”.

|  |
| --- |
| Bloque de código |
| MariaDB [prueba]> UPDATE gente SET fecha= '2012-12-12';  Query OK, 5 rows affected (0.005 sec)  Rows matched: 5 Changed: 5 Warnings: 0  MariaDB [prueba]> SELECT \* FROM gente;  +---------+------------+  | nombre | fecha |  +---------+------------+  | Fulano | 2012-12-12 |  | Mengano | 2012-12-12 |  | Pegano | 2012-12-12 |  | Perillo | 2012-12-12 |  | Tulano | 2012-12-12 | |

Al suponer que se necesita ponerle el incrementar la edad por 10 años a cada fila de la tabla “gente”.

|  |
| --- |
| Bloque de código |
| MariaDB [prueba]> UPDATE gente SET edad = edad +10;  Query OK, 5 rows affected (0.005 sec)  Rows matched: 5 Changed: 5 Warnings: 0  MariaDB [prueba]> SELECT \* FROM gente;  +---------+------------+------+  | nombre | fecha | edad |  +---------+------------+------+  | Fulano | 1974-04-12 | 28 |  | Mengano | 1978-06-15 | 30 |  | Pegano | 1993-02-10 | 43 |  | Perillo | 2020-02-02 | 10 |  | Tulano | 2000-12-02 | 32 |  +---------+------------+------+ |

Sin embargo, no se tienen que actualizar todas filas de la tabla, ya que se tiene la posibilidad de definir cuáles filas se desean actualizar. La primera es por medio de la cláusula “WHERE”. Utilizando esta cláusula se tiene la posibilidad de entablar una condición. Únicamente las filas que cumplan dicha condición van a ser actualizadas:

|  |
| --- |
| Bloque de código |
| MariaDB [prueba]> UPDATE gente SET edad = 55, fecha = '1970-12-31' WHERE nombre = 'Tulano';  Query OK, 1 row affected (0.005 sec)  Rows matched: 1 Changed: 1 Warnings: 0  MariaDB [prueba]> SELECT \* FROM gente;  +---------+------------+------+  | nombre | fecha | edad |  +---------+------------+------+  | Fulano | 1974-04-12 | 28 |  | Mengano | 1978-06-15 | 30 |  | Pegano | 1993-02-10 | 43 |  | Perillo | 2020-02-02 | 10 |  | Tulano | 1970-12-31 | 55 |  +---------+------------+------+  5 rows in set (0.000 sec)  MariaDB [prueba]> |

Otro ejemplo sería que se actualice la fecha al día 2012-12-12 para todos lo que sean mayores de 40 años de edad; esto se podría hacer así:

|  |
| --- |
| Bloque de código |
| MariaDB [prueba]> UPDATE gente SET fecha = '2012-12-12' WHERE edad > 30;  Query OK, 2 rows affected (0.005 sec)  Rows matched: 2 Changed: 2 Warnings: 0  MariaDB [prueba]> SELECT \* FROM gente;  +---------+------------+------+  | nombre | fecha | edad |  +---------+------------+------+  | Fulano | 1974-04-12 | 28 |  | Mengano | 1978-06-15 | 30 |  | Pegano | 2012-12-12 | 43 |  | Perillo | 2020-02-02 | 10 |  | Tulano | 2012-12-12 | 55 |  +---------+------------+------+  5 rows in set (0.000 sec)  MariaDB [prueba]> |

## Borrado de registros

Para borrar filas se utiliza la sentencia “DELETE”. La sintaxis es muy similar a la de “UPDATE”:

|  |
| --- |
| Bloque de código |
| DELETE [LOW\_PRIORITY] [QUICK] [IGNORE] FROM table\_name  [WHERE where\_definition]  [ORDER BY ...]  [LIMIT row\_count] |

Para borrar el registro que tiene por nombre Perillo se procede de la siguiente forma:

|  |
| --- |
| Bloque de código |
| MariaDB [prueba]> DELETE FROM gente WHERE nombre = 'Perillo';  Query OK, 1 row affected (0.005 sec)  MariaDB [prueba]> SELECT \* FROM gente;  +---------+------------+------+  | nombre | fecha | edad |  +---------+------------+------+  | Fulano | 1974-04-12 | 28 |  | Mengano | 1978-06-15 | 30 |  | Pegano | 2012-12-12 | 43 |  | Tulano | 2012-12-12 | 55 |  +---------+------------+------+  4 rows in set (0.000 sec)  MariaDB [prueba]> |

A continuación, se debe borrar aquello que tenga edad comprendida entre 30 (incluyendo 30) y 50 años (incluyendo 50), para eso se usa un operador llamado “AND”, se verán con más detalle estos operadores.

|  |
| --- |
| Bloque de código |
| MariaDB [prueba]> DELETE FROM gente WHERE edad >= 30 AND edad <= 50;  Query OK, 2 rows affected (0.005 sec)  MariaDB [prueba]> SELECT \* FROM gente;  +--------+------------+------+  | nombre | fecha | edad |  +--------+------------+------+  | Fulano | 1974-04-12 | 28 |  | Tulano | 2012-12-12 | 55 |  +--------+------------+------+  2 rows in set (0.000 sec)  MariaDB [prueba]> |

### Vaciar una tabla

Cuando se deseen borrar todas las filas de una tabla, se puede utilizar una sentencia “DELETE” sin condiciones, como se vio en el punto anterior. Pero existe una sentencia alternativa, “TRUNCATE”, que hace la misma tarea de una forma mucho más veloz. La diferencia es que “DELETE” hace un borrado secuencial de la tabla, fila por fila y “TRUNCATE” elimina la tabla y la vuelve a crear desocupada, lo que es mucho más eficaz.

|  |
| --- |
| Bloque de código |
| MariaDB [prueba]> TRUNCATE gente;  Query OK, 0 rows affected (0.037 sec)  MariaDB [prueba]> SELECT \* FROM gente;  Empty set (0.001 sec)  MariaDB [prueba]> |

# SQL para consulta de datos – DML

Para encontrar la información dentro de una base de datos, es preciso escudriñar dentro de la estructura de los datos su estructura y su naturaleza, por ejemplo, el tipo de datos fecha tiene una estructura YYYY-MM-DD, y si necesito listar los registros de un año determinado, se necesita una función que refiera a ese dato particular dentro de la columna; por este motivo se necesita revisar algunas funciones antes de empezar a trabajar consultas de datos.

## Funciones en MySQL

Considerando que MySQL es rico en lo que respecta a operadores y en lo que se refiere a funciones es también muy amplio, MySQL dispone de multitud de funciones. Solo se explicarán algunas funciones necesarias para entender cómo funcionan en general.

A continuación, se listarán las que se usarán para el ejemplo que sigue:

1. Algunas funciones de MySQL

| Función | Descripción |
| --- | --- |
| *LENGTH* | Devuelve la longitud de una cadena (en *bytes*). |
| *SUM* | Calcula la suma de un conjunto de valores. |
| *MAX* | Devuelve el valor máximo en un conjunto de valores. |
| *MIN* | Devuelve el valor mínimo en un conjunto de valores. |
| *CEIL* | Redondea el valor entero más pequeño que es mayor o igual que un número. |
| *COUNT* | Devuelve el número de registros devueltos por una consulta de selección. |
| *AVG* | Devuelve el valor medio de una expresión. |
| *YEAR* | Devuelve la parte del año para una fecha determinada. |
| *MONTH* | Devuelve la parte del mes para una fecha determinada. |
| *DAY* | Devuelve el día del mes para una fecha determinada. |
| *DATE* | Extrae la parte de la fecha de una expresión de fecha y hora. |
| *TIME* | Extrae la parte de tiempo de una hora / fecha y hora determinadas. |
| *SEC\_TO\_TIME* | Devuelve un valor de tiempo basado en los segundos especificados. |

Para este ejemplo se usará una base de datos de llamadas telefónicas, esta base de datos tiene la estructura que se verá a continuación y con ella se aprenderá a hacer consultas.

### Ejemplo llamadas telefónicas

Descargue el archivo para crear la base de datos “cdr” (recuerde que debe descomprimirlo) y de él extraiga el archivo cdr.sql. Este archivo crea una base de datos llamada “cdr” y dentro una tabla con el mismo nombre.

Importe el archivo con la sentencia source <ruta del archivo sql>.

|  |
| --- |
| Bloque de código |
| MariaDB [prueba]> source D:/Peter/Downloads/cdr.sql; |

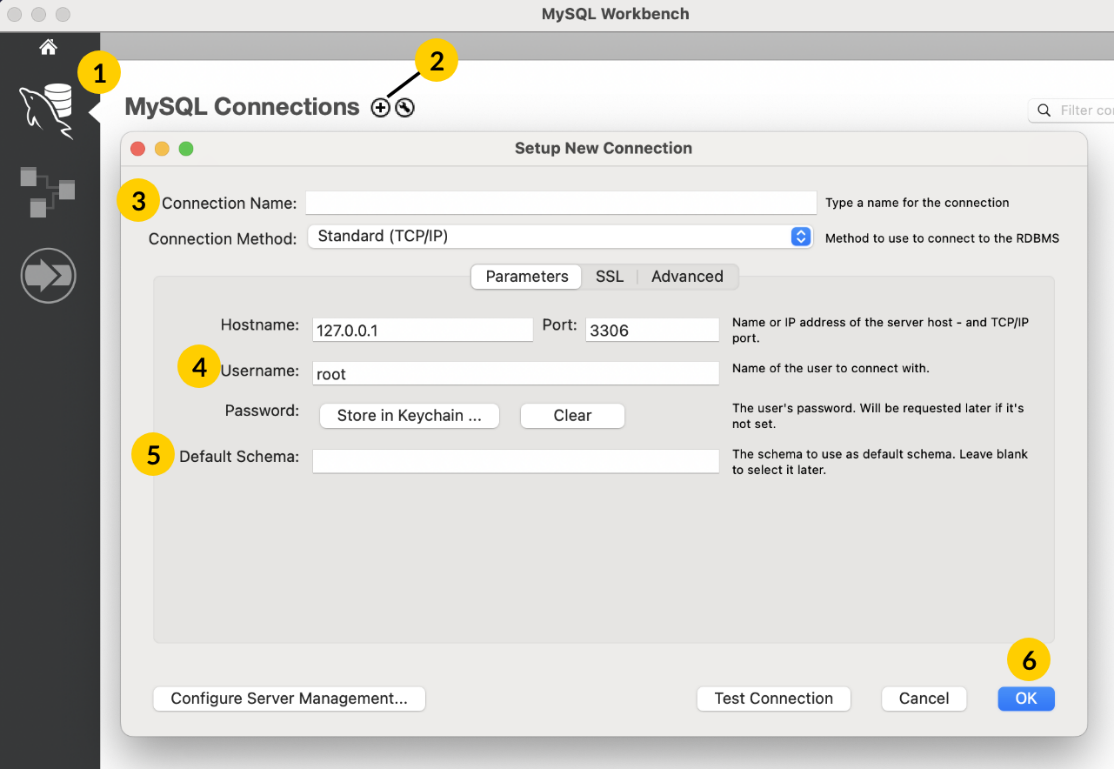
Conozca los datos de prueba con los que se trabajará.

|  |  |
| --- | --- |
| Diccionario de datos | |
| Tabla | CDR (Call Record Data) |
| Fecha | DD/MM/YYYY |
| Propósito | Guarda el registro de llamadas entrantes y salientes de un sistema de telefonía PBX. |

|  |  |  |  |
| --- | --- | --- | --- |
| Campo | Tipo | Longitud | Descripción |
| Registro | *DATETIME* | 0 | Fecha y hora de registro de la llamada. |
| Origen | *VARCHAR* | 80 | Número de teléfono o extensión del llamante. |
| Destino | *VARCHAR* | 80 | Número de teléfono o extensión del llamado o destino. |
| Estado | *VARCHAR* | 30 | *ANSWERED, BUSY, FAILED, NO ANSWER*, que representa lo que sucede finalmente con la llamada. |
| Duración | INT | 10 | Lo que dura la llamada incluyendo el tiempo de timbrado. |
| Facturar | INT | 10 | Lo que dura la llamada desde el momento en que es contestada la llamada. |

A partir de este punto usaremos MySQL Workbench para visualizar mejor los resultados.

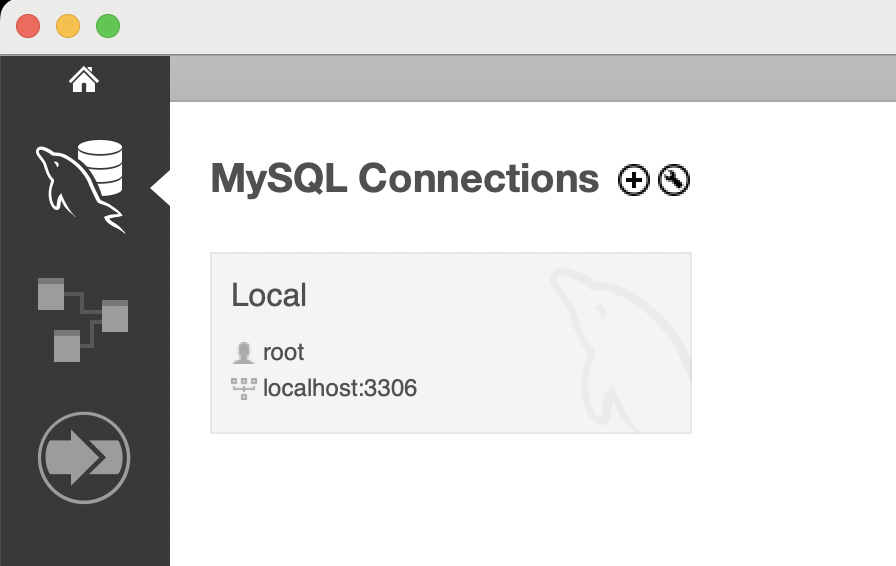
1. Interfaz de MySQL Workbench



1. Haga clic en la pestaña de uso de bases de datos.
2. Cree una nueva conexión.
3. Póngale el nombre que prefiera a esa conexión.
4. Ponga el usuario root.
5. Digite el nombre de la base de datos que usará: cdr.
6. Haga clic en ok.

Ahora debe conectar dando doble clic.

1. Interfaz de MySQL Connections



Ya se dispone de bases de datos y se sabe cómo agregar y cambiar datos. A continuación se aprenderá a extraer datos de una base de datos. Para ello se volverá a utilizar la sentencia “SELECT”.

La sintaxis de “SELECT” es complicada, Sin embargo, en este punto no se explicará todas sus opciones. Una manera más general reside en la siguiente sintaxis:

|  |
| --- |
| Bloque de código |
| SELECT [ALL | DISTINCT | DISTINCTROW]  expresion\_select,...  FROM referencias\_de\_tablas  WHERE condiciones  [GROUP BY {nombre\_col | expresion | posicion}  [ASC | DESC], ... [WITH ROLLUP]]  [HAVING condiciones]  [ORDER BY {nombre\_col | expresion | posicion}  [ASC | DESC] ,...]  [LIMIT {[desplazamiento,] contador | contador OFFSET desplazamiento}] |

Para listar todos los elementos de la tabla se debe usar la sentencia:

|  |
| --- |
| Bloque de código |
| SELECT \* FROM cdr |

Escriba la sentencia anterior. Al decir \* estamos diciendo que seleccione todas las columnas de la tabla “cdr”.

1. MySQL Workbench ejemplo de listado de elementos

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

También se pueden seleccionar solo algunas, para esto, se deben separar por una coma, así:

|  |
| --- |
| Bloque de código |
| SELECT origen, destino, duracion FROM cdr |

Cabe mencionar que es posible asignar un alias a cualquiera de las expresiones “SELECT” y se puede realizar empleando la palabra AS, aunque esta palabra es opcional:

|  |
| --- |
| Bloque de código |
| SELECT origen AS num\_orig, destino, duracion AS tiempo\_llamada FROM cdr |

Suponga que se necesita saber cuánto cobrar por las llamadas salientes realizadas a celular o larga distancia nacional (que el destino tenga 10 dígitos), sabiendo que el valor del minuto es $85 (ochenta y cinco pesos):

Lo primero que se necesita tener en cuenta, es que hay que filtrar las llamadas que tienen un destino de 10 dígitos. Esto se logra con la función “LENGTH” y el número de segundos que tiene la llamada desde que es atendida, se guarda en la columna facturar. Pero para saber cuántos minutos duró la llamada, se debe dividir el número de segundos por 60 y eso nos da el número de minutos. Ahora, el número de minutos no es entero y tiene datos decimales, se debe redondear hacia arriba con la función CEIL y este valor ya redondeado, se puede multiplicar por 85 que es el valor del minuto.

|  |
| --- |
| Bloque de código |
| SELECT \* ,  facturar/60 AS minutos,  CEIL(facturar/60) AS minutos\_redondeado,  CEIL(facturar/60) \* 85 AS valor  FROM cdr WHERE LENGTH(destino) = 10 |

## Operadores

SQL tiene muchos operadores diferentes según el tipo de columna. Esos operadores se emplean para construir expresiones que se usan en las consultas.

### Operadores lógicos

Los operadores lógicos se emplean para crear expresiones lógicas complejas. Permite el uso de álgebra booleana y ayuda a crear condiciones o filtros de información mucho más precisos.

Recuerde que en el álgebra booleana o álgebra de bool, existen dos valores posibles para ser operados y son los resultados: verdadero y falso. MySQL dispone de dos constantes para esos valores: “TRUE” y “FALSE”, respectivamente.

En SQL se añade un tercer valor. Esto es para hacer posible trabajar con valores “NULL”. El valor verdadero se implementa como 1 o “TRUE”, el falso como 0 o “FALSE” y el desconocido como “NULL”.

|  |
| --- |
| Bloque de código |
| MariaDB [prueba]> SELECT TRUE, FALSE, NULL;  +------+-------+------+  | TRUE | FALSE | NULL |  +------+-------+------+  | 1 | 0 | NULL |  +------+-------+------+  1 row in set (0.000 sec)  MariaDB [prueba]> |

Conozcamos algunos operadores:

#### Operador AND

En SQL se puede usar el operador “AND”, que se trata de un operador binario, es decir, requiere de dos operandos. El resultado de la operación es verdadero, solo si ambos operandos son verdaderos, falso si cualquier operando es falso o nulo si alguno de los dos es nulo.

Esto se representa mediante la siguiente tabla de verdad:

|  |  |  |
| --- | --- | --- |
| A | B | A *AND* B |
| FALSO | FALSO | FALSO |
| FALSO | VERDADERO | FALSO |
| VERDADERO | FALSO | FALSO |
| VERDADERO | VERDADERO | VERDADERO |
| FALSO | NULL | FALSO |
| NULL | FALSO | FALSO |
| VERDADERO | NULL | NULL |
| NULL | VERDADERO | NULL |

|  |
| --- |
| Bloque de código |
| MariaDB [prueba]> SELECT 1 AND 0, 1 AND NULL, 0 AND NULL, 1 AND 0 AND 1;  +---------+------------+------------+---------------+  | 1 AND 0 | 1 AND NULL | 0 AND NULL | 1 AND 0 AND 1 |  +---------+------------+------------+---------------+  | 0 | NULL | 0 | 0 |  +---------+------------+------------+---------------+  1 row in set (0.000 sec) |

#### Operador OR

El operador OR también es binario. Si ambos operandos siendo diferentes de “NULL” y el resultado es verdadero si alguno de ellos es verdadero y falso si los dos son falsos. Si uno de los operandos es “NULL” el resultado es verdadero si el otro es verdadero y “NULL” si el otro es falso.

La tabla de verdad es:

|  |  |  |
| --- | --- | --- |
| A | B | A OR B |
| FALSO | FALSO | FALSO |
| FALSO | VERDADERO | VERDADERO |
| VERDADERO | FALSO | VERDADERO |
| VERDADERO | VERDADERO | VERDADERO |
| FALSO | NULL | NULL |
| NULL | FALSO | NULL |
| VERDADERO | NULL | VERDADERO |
| NULL | VERDADERO | VERDADERO |

|  |
| --- |
| Bloque de código |
| MariaDB [prueba]> SELECT 1 OR 0, 1 OR NULL, 0 OR NULL, 1 OR 0 OR 1;  +--------+-----------+-----------+-------------+  | 1 OR 0 | 1 OR NULL | 0 OR NULL | 1 OR 0 OR 1 |  +--------+-----------+-----------+-------------+  | 1 | 1 | NULL | 1 |  +--------+-----------+-----------+-------------+  1 row in set (0.000 sec)  MariaDB [prueba]> |

### Operadores de igualdad

El operador igual (=) compara dos expresiones y da como resultado “TRUE” si son iguales o “FALSE” si son diferentes. Ya lo hemos usado en ejemplos anteriormente:

|  |
| --- |
| Bloque de código |
| SELECT \* FROM cdr WHERE registro = '2018-01-02 10:19:08' |

### Operadores de desigualdad

|  |  |
| --- | --- |
| Operador | Descripción |
| <= | Menor o igual |
| < | Menor |
| > | Mayor |
| >= | Mayor o igual |

|  |
| --- |
| Bloque de código |
| SELECT \* FROM cdr  WHERE registro >= '2018-01-02 10:19:08'  AND registro <= '2018-01-04 10:19:08' |

## Ordenar resultados

También, se puede agregar una cláusula de orden ORDER BY para lograr resultados ordenados por la columna que se quiera y se puede hacer ascendente o descendente con la cláusula “ASC” y “DESC” respectivamente.

Por ejemplo, se desean listas las llamadas en orden de duración descendente (de mayor a menor):

|  |
| --- |
| Bloque de código |
| SELECT \* FROM cdr ORDER BY duracion DESC |

Y también de manera ascendente por la fecha:

|  |
| --- |
| Bloque de código |
| SELECT \* FROM cdr ORDER BY registro ASC |

Igualmente, se puede ordenar por el resultado de operación, aunque el rendimiento de la consulta disminuye. Ejemplo, para llamadas de larga distancia o celular a 85 pesos el minuto, que se ordene por la llamada que más costo tenga a la que menos costo tenga.

|  |
| --- |
| Bloque de código |
| SELECT \*, CEIL(facturar/60) \* 85 AS valor FROM cdr  WHERE LENGTH(destino) = 10  ORDER BY CEIL(facturar/60) \* 85 DESC |

Se puede agrupar por varios criterios a la vez, por ejemplo si se ordenan los resultados por la duración y si hay repetidos, que el siguiente criterio sea la fecha de registro.

|  |
| --- |
| Bloque de código |
| SELECT \* FROM cdr  ORDER BY facturar DESC , registro ASC |

## Listar y limitar resultados

La principal forma de limitar resultados es filtrando condiciones en los datos con la cláusula “WHERE”, sin embargo cuando a pesar de eso los datos son muchos, es necesario paginarlo o presentarlo por partes, para ello se miraría la cláusula “LIMIT”.

#### Ejemplo

Se desea filtrar las llamadas larga distancia o celular de todo el año 2016 que cuestan más de $1.000, y presentarlas ordenadas de mayor a menor por el precio.

|  |
| --- |
| Bloque de código |
| SELECT \*, CEIL(facturar/60) \* 85 AS valor  FROM cdr  WHERE YEAR(registro) = 2016  AND LENGTH(destino) = 10  AND CEIL(facturar/60) \* 85 > 200  ORDER BY valor DESC |

Los datos que la consulta arroja seguramente son muchos. Para saber cuántos resultados cumplen con las condiciones de la cláusula WHERE, es preciso contarlos con la función “COUNT”, pero la consulta no debe usar la cláusula ORDER BY porque la función “COUNT” solamente arroja una fila y con una única columna, por lo tanto, el resultado no se puede ordenar.

|  |
| --- |
| Bloque de código |
| SELECT COUNT(\*)  FROM cdr  WHERE YEAR(registro) = 2016  AND LENGTH(destino) = 10  AND CEIL(facturar/60) \* 85 > 200 |

Como se puede ver, son muchos registros, pero si el interés es mostrar los primeros 10, utilice la cláusula “LIMIT”.

|  |
| --- |
| Bloque de código |
| SELECT \*, CEIL(facturar/60) \* 85 AS valor  FROM cdr  WHERE YEAR(registro) = 2016  AND LENGTH(destino) = 10  AND CEIL(facturar/60) \* 85 > 200  ORDER BY valor DESC  LIMIT 10 |

Para lograr hacer esto, la cláusula “LIMIT” permite dos parámetros. Cuando se utilizan los dos, el primero muestra el número de la primera fila a traer, y el segundo el número de filas a traer. Se puede por ejemplo, rescatar las filas de dos en dos:

|  |
| --- |
| Bloque de código |
| SELECT \*, CEIL(facturar/60) \* 85 AS valor  FROM cdr  WHERE YEAR(registro) = 2016  AND LENGTH(destino) = 10  AND CEIL(facturar/60) \* 85 > 200  ORDER BY valor DESC  LIMIT 10, 10 |

Y con la siguiente consulta se traen los próximos 10.

|  |
| --- |
| Bloque de código |
| SELECT \*, CEIL(facturar/60) \* 85 AS valor  FROM cdr  WHERE YEAR(registro) = 2016  AND LENGTH(destino) = 10  AND CEIL(facturar/60) \* 85 > 200  ORDER BY valor DESC  LIMIT 20, 10 |

## Agrupar filas

Se pueden agrupar filas en la salida con la sentencia “SELECT”, según los valores de una columna, usando la cláusula GROUP BY. Y se usa con funciones de agrupación como lo son, “AVG”, “SUM”, “MAX”, “COUNT”, entre otros.

#### Ejemplo

Se desea saber cuántas llamadas entrantes se realizaron durante cada uno de los meses del año 2017. Las llamadas entrantes son aquellas que no provienen de una extensión local, es decir, el origen tiene más de tres dígitos.

|  |
| --- |
| Bloque de código |
| SELECT MONTH(registro) AS mes, COUNT(\*) AS cantidad  FROM cdr  WHERE LENGTH(origen) > 3 AND YEAR(registro) = 2017  GROUP BY mes |

Pero si se desea mostrar también la agrupación de los años 2017, 2018 y 2019 y los meses de cada uno de estos años, sería:

|  |
| --- |
| Bloque de código |
| SELECT YEAR(registro) AS año, MONTH(registro) AS mes, COUNT(\*) AS cantidad  FROM cdr  WHERE LENGTH(origen) > 3 AND YEAR(registro) IN (2017,2018,2019)  GROUP BY año, mes  ORDER BY año, mes |

Es de resaltar que para agregar más de una opción en el año, se usó la cláusula IN y se ordena por año y como segundo criterio por mes.

# Consulta multitablas – DML

Para explicar este tipo de ejercicios, crearemos una base de datos de empleados llamada “ehr”, y las tablas “empleados” y “areas”, se relacionan así: un empleado pertenece a un área y un área puede tener varios empleados.

Note que “area\_id” en la tabla “empleados” puede ser nuleable:

|  |
| --- |
| Bloque de código |
| MariaDB [ehr]> CREATE TABLE IF NOT EXISTS areas (  -> id\_area TINYINT NOT NULL,  -> area VARCHAR(45) NULL,  -> PRIMARY KEY (id\_area))  -> ENGINE = InnoDB;  Query OK, 0 rows affected (0.019 sec)  MariaDB [ehr]> CREATE TABLE IF NOT EXISTS empleados (  -> nombre\_empleado VARCHAR(60) NOT NULL,  -> area\_id TINYINT NULL,  -> PRIMARY KEY (nombre\_empleado),  -> INDEX fk\_empleados\_areas\_idx (area\_id ASC),  -> CONSTRAINT fk\_empleados\_areas  -> FOREIGN KEY (area\_id)  -> REFERENCES areas (id\_area)  -> ON DELETE NO ACTION  -> ON UPDATE NO ACTION)  -> ENGINE = InnoDB;  Query OK, 0 rows affected (0.023 sec) |

Se insertarán los siguientes datos:

|  |
| --- |
| Bloque de código |
| INSERT INTO areas (id\_area, area) VALUES ('17', 'Ventas'), ('19', 'Producción'), ('20', 'Ingeniería') ('21', 'Mercadeo');  MariaDB [ehr]> INSERT INTO `areas` (`id\_area`, `area`) VALUES  -> ('17', 'Ventas'),  -> ('19', 'Producción'),  -> (20, 'Ingeniería'),  -> (21, 'Mercadeo');  Query OK, 4 rows affected, 2 warnings (0.015 sec)  Records: 4 Duplicates: 0 Warnings: 2  MariaDB [ehr]>  MariaDB [ehr]> INSERT INTO empleados (nombre\_empleado, area\_id) VALUES  -> ('Andrade', '17'),  -> ('Jordan', '20'),  -> ('Steinberg', '20'),  -> ('Robinson', '19'),  -> ('Zolano', '19'),  -> ('Gaspar', NULL);  Query OK, 6 rows affected, 2 warnings (0.006 sec)  Records: 6 Duplicates: 0 Warnings: 2 |

Y los datos quedan así:

|  |
| --- |
| Bloque de código - Empleados |
| MariaDB [ehr]> SELECT \* FROM empleados;  +-----------------+---------+  | nombre\_empleado | area\_id |  +-----------------+---------+  | Gaspar | NULL |  | Andrade | 17 |  | Robinson | 19 |  | Zolano | 19 |  | Jordan | 20 |  | Steinberg | 20 |  +-----------------+---------+  6 rows in set (0.000 sec)  MariaDB [ehr]> |

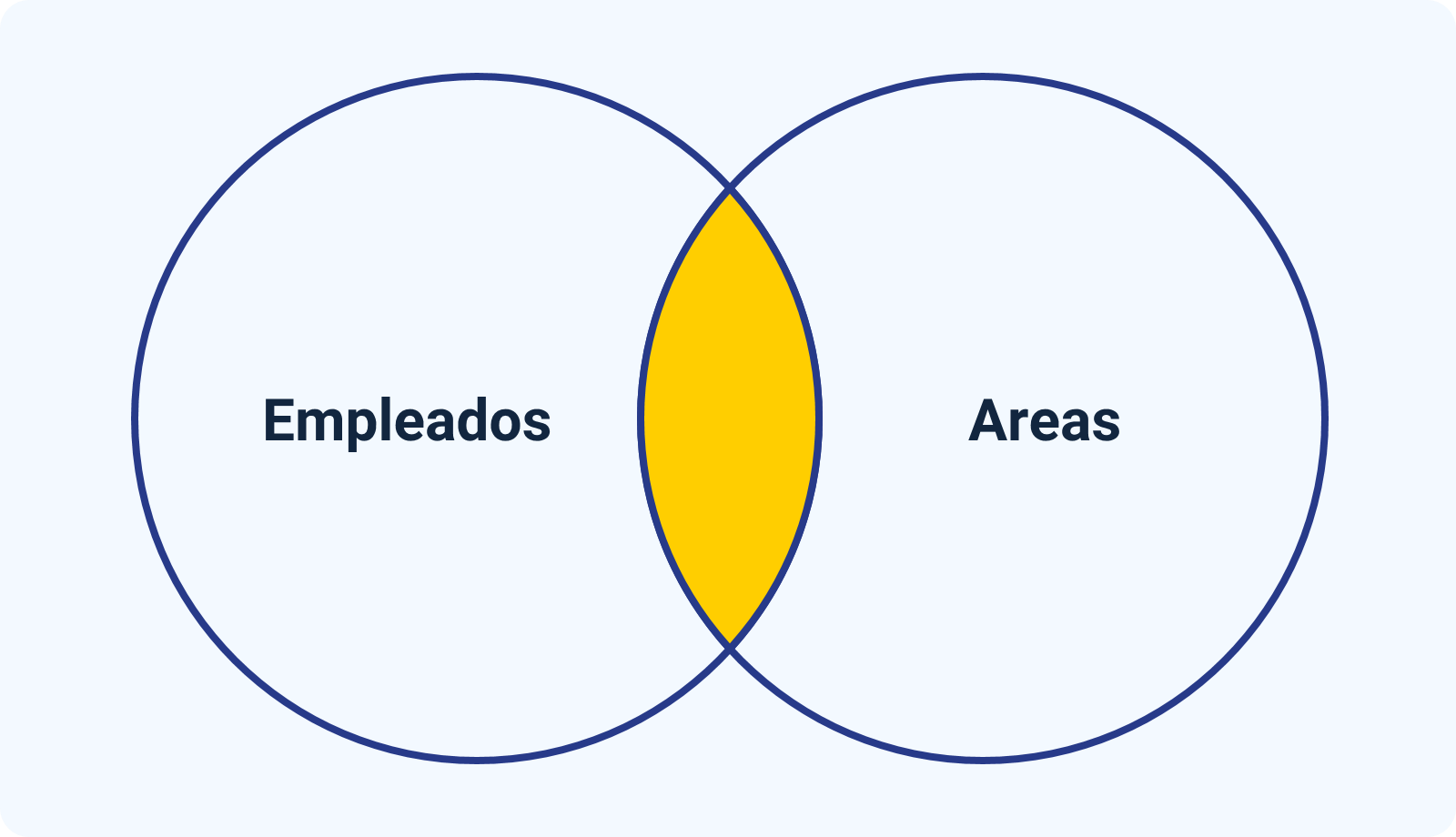
|  |
| --- |
| Bloque de código - Áreas |
| MariaDB [ehr]> SELECT \* FROM areas;  +---------+------------+  | id\_area | area |  +---------+------------+  | 17 | Ventas |  | 19 | Producción |  | 20 | Ingeniería |  | 21 | Mercadeo |  +---------+------------+  4 rows in set (0.000 sec)  MariaDB [ehr]> |

## Combinación interna (INNER JOIN)

Con esta operación, cada fila en la tabla “empleados” es combinada (o unida) con los correspondientes de la tabla “areas” que satisface la condición que se especifiquen en el predicado del “JOIN”. Cualquier fila de la tabla “empleados” o de la tabla “areas” que no tenga uno correspondiente en la otra tabla (“areas”) es excluido y solo aparecerán los que tengan correspondencia en la otra tabla. Este es el tipo de JOIN más utilizado, por lo que es considerado el tipo de combinación predeterminado.

|  |
| --- |
| Bloque de código - Áreas |
| SELECT \*  FROM empleados AS e  INNER JOIN areas as a ON a.id\_area = e.area\_id |

1. INNER JOIN

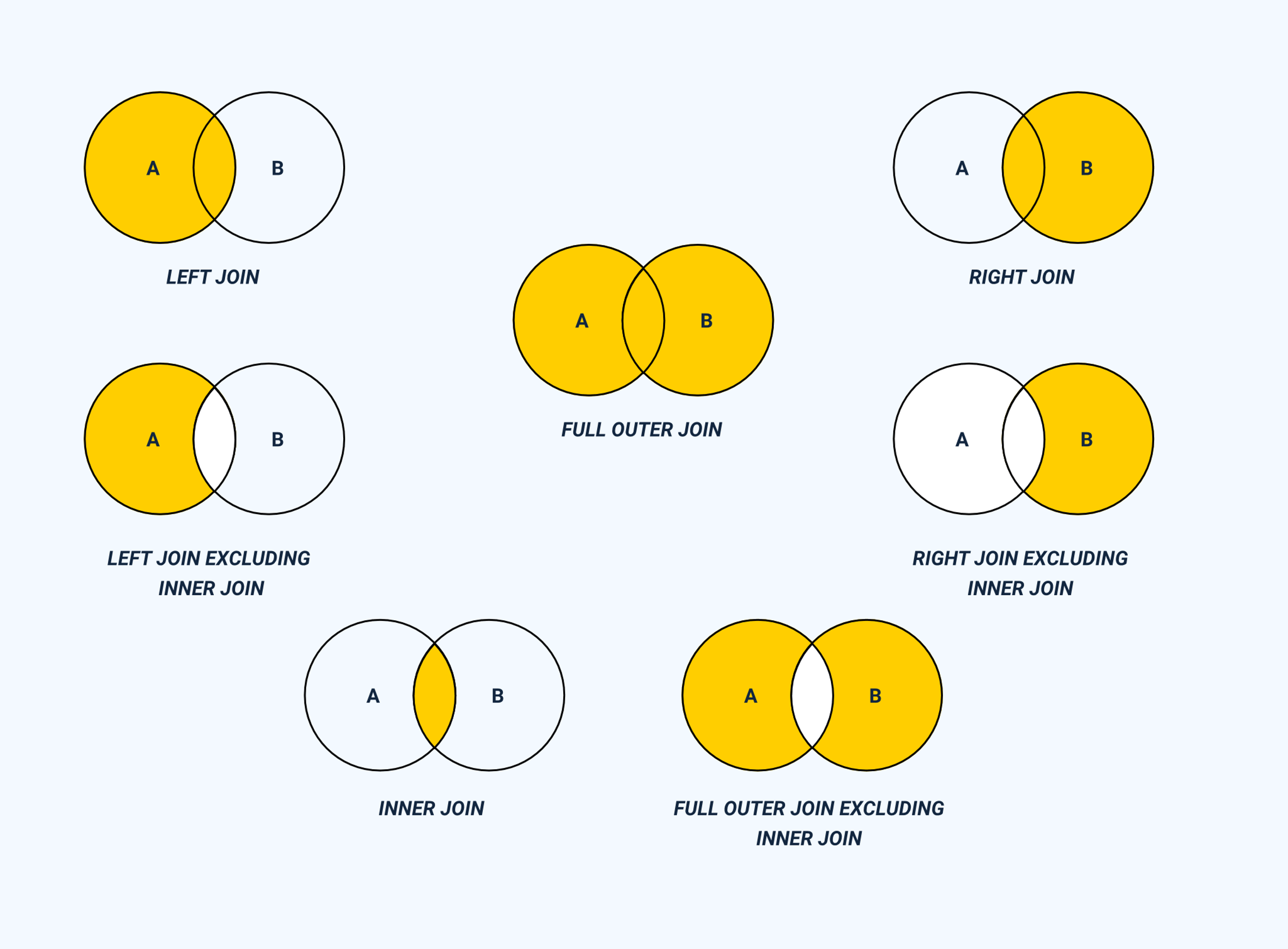


La condición de la relación (“a.id\_area”= “e.area\_id”) consiste en que debe estar presente en ambas tablas el valor. Cuando el empleado tiene NULL en “area\_id” no aparece, debido a que no puede cumplirla. Esto se representa con un diagrama de Venn en la conjunción, porque el valor debe estar en ambas tablas.

Existe otros tipos de consultas, las “FULL JOIN”, “FULL OUTER JOIN”, “CROSS JOIN”, que no se verán porque son muy poco empleadas, sin embargo, con el concepto principal se puede acceder a la documentación en la Web para comprobar su funcionamiento.

A través de la siguiente imagen se representan también otros tipos de relaciones, diagramas de Venn que permiten visualizar resultados retornados. Se puede usar para ayudar a comprender los resultados que se deben esperar con cada variante de las sentencias JOIN.

1. SQL JOINS



## Combinación externa

Es una variedad de composición de tablas que permite seleccionar algunas filas de una tabla aunque estas no tengan correspondencia con las filas de la otra tabla con la que se combina.

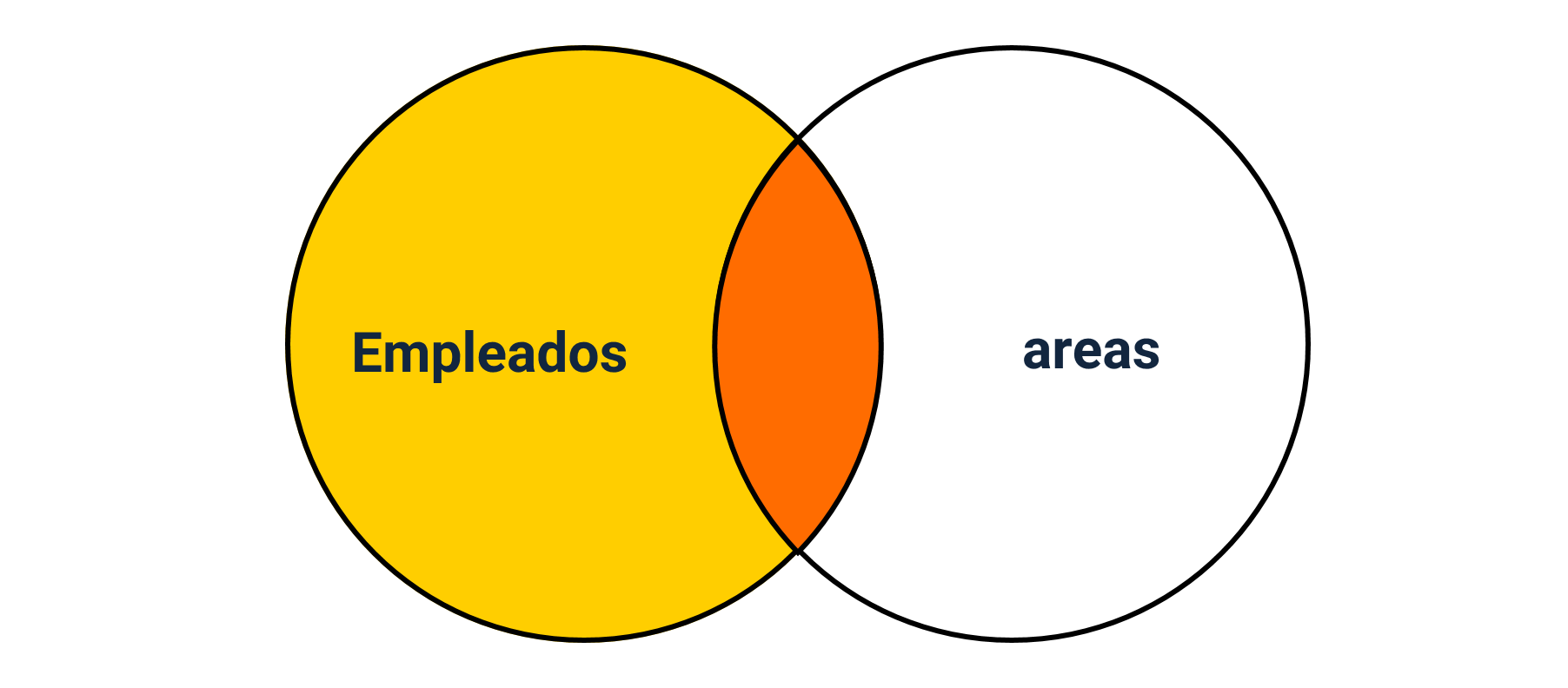
#### LEFT JOIN

La sintaxis es muy similar a la anterior.

|  |
| --- |
| Bloque de código - Áreas |
| SELECT \*  FROM empleados AS e  LEFT JOIN areas as a ON a.id\_area = e.area\_id |

El resultado de este tipo de consulta contiene todos los registros de la tabla de la izquierda (la primera tabla que se define en el “FROM” en la consulta), independiente de si existe un registro correspondiente en la tabla de la derecha (la que se define en el JOIN).

1. JOIN



La sentencia “LEFT JOIN” devuelve la pareja de todos los valores de la tabla izquierda (“empleados”), con los valores de la tabla de la derecha (“areas”) correspondientes, si los hay, o retorna un valor nulo NULL en los campos de la tabla derecha cuando no haya correspondencia.

A diferencia del resultado presentado en el ejemplo INNER JOIN donde no se mostraba el empleado cuya área no existía, en el ejemplo siguiente se presentarán los empleados con su respectiva área, y adicionalmente se presentará un empleado cuya área no existe.

El empleado que no tiene área se encuentra en la región amarilla del diagrama de la derecha, mientras que los empleados con área están en la franja naranja, en la intersección de empleados y “areas”.

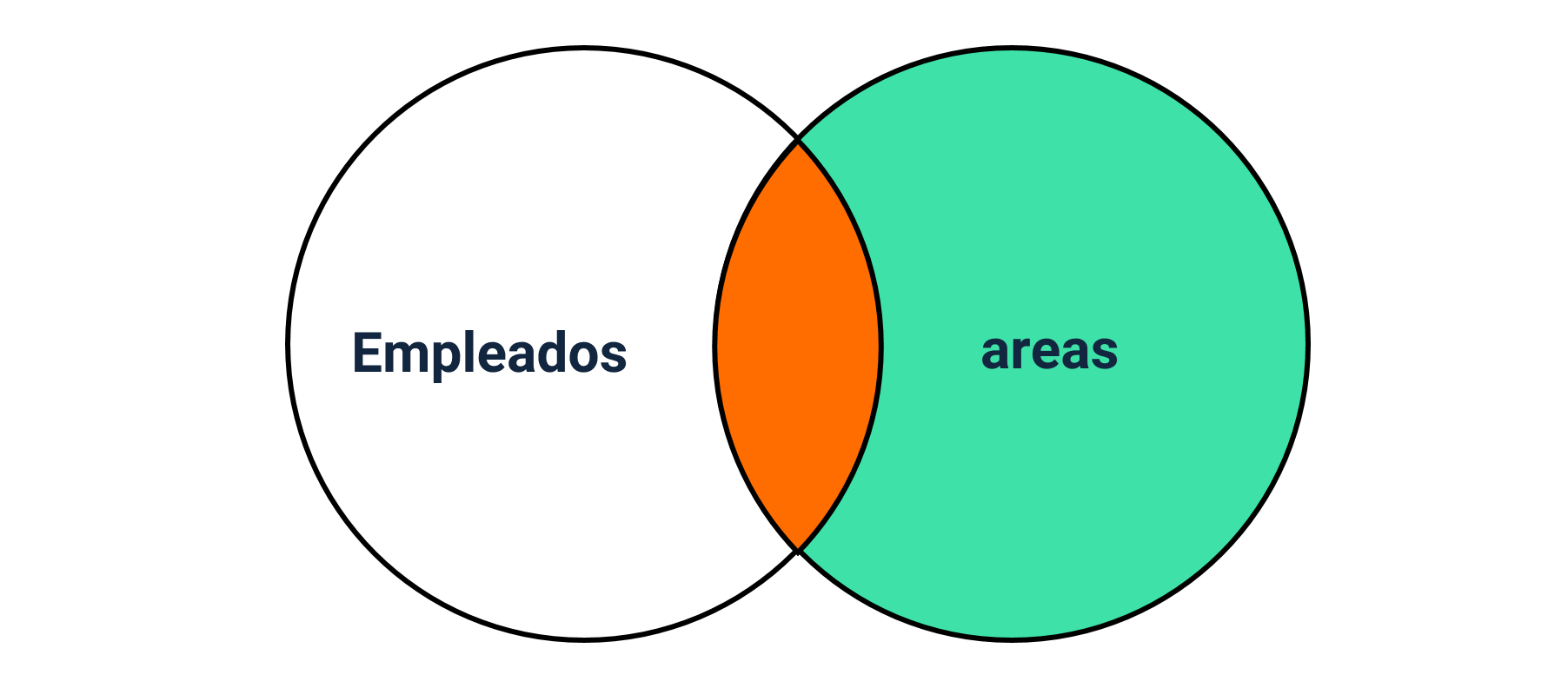
#### RIGTH JOIN

La sentencia “RIGHT JOIN” retorna todos los valores de la tabla derecha (“areas”) con los valores de la tabla de la izquierda (“empleados”) correspondientes, si existen, o retorna un valor nulo “NULL” para los campos de la tabla izquierda cuando no haya correspondencia.

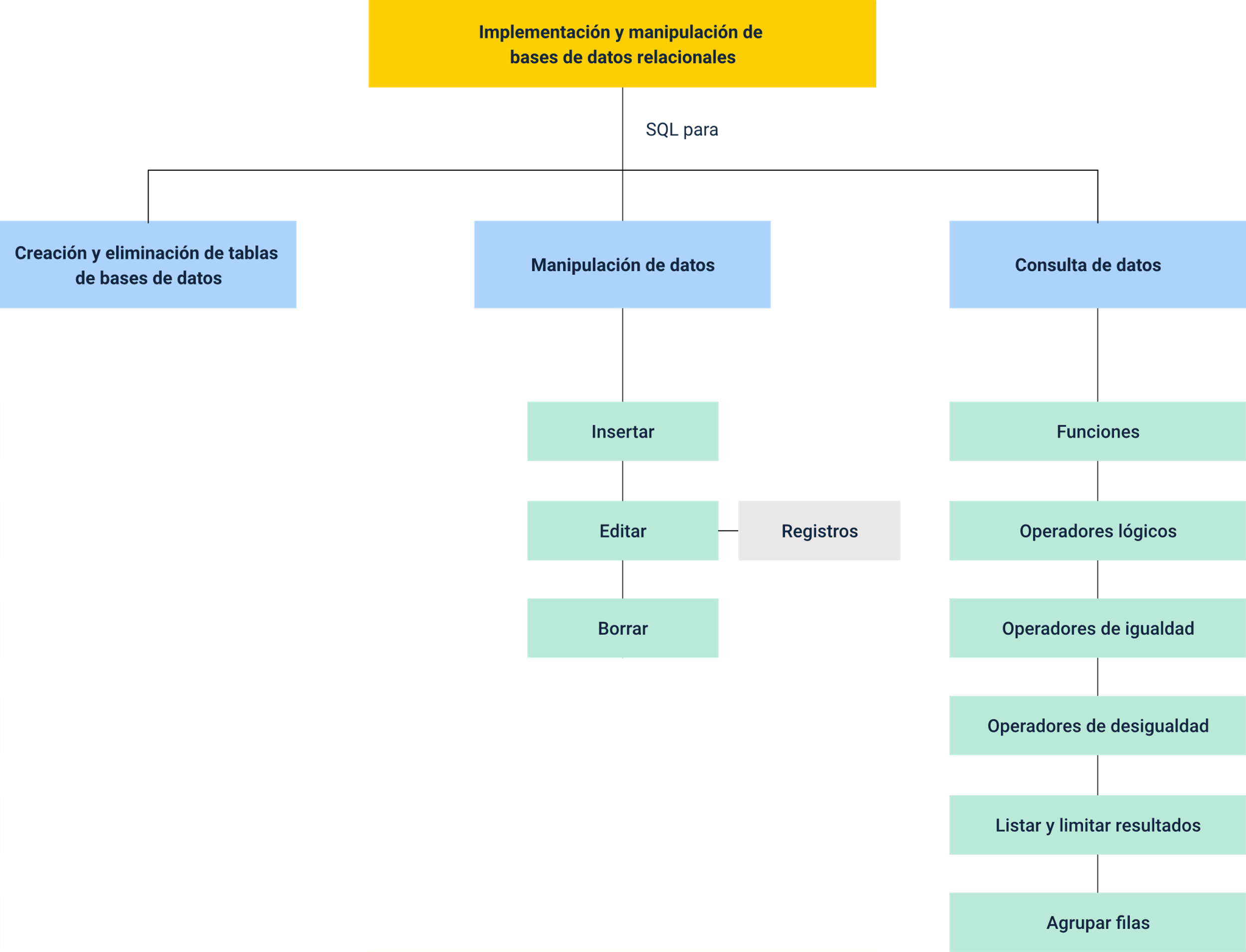
|  |
| --- |
| Bloque de código - Áreas |
| SELECT \*  FROM empleados AS e  RIGHT JOIN areas as a ON a.id\_area = e.area\_id |

En el diagrama las áreas que no tienen empleados están en el área verde mientras que las áreas con empleados están en el área anaranjada, en la intersección de empleados y áreas.

1. RIGHT JOIN



Síntesis



Material complementario

| Tema | Referencia | Tipo de material | Enlace del recurso |
| --- | --- | --- | --- |
| Bases de datos | Microsoft. (2021). Fundamentos de la normalización de bases de datos. | Página web | <https://learn.microsoft.com/es-es/office/troubleshoot/access/database-normalization-description> |

Glosario

**Base de datos**: es un sistema organizado de almacenamiento de datos que permite el acceso, la gestión y la actualización de grandes cantidades de información de manera eficiente.

**Diccionario de datos**: es una colección estructurada de metadatos que contiene definiciones y representaciones de elementos de datos dentro de un sistema o aplicación.

**Metadatos**: son datos que proporcionan información sobre otros datos. El propósito de los metadatos es describir características relevantes de los datos a los que hacen referencia, facilitando su gestión, uso y localización eficiente.

Referencias bibliográficas

Henry.Sudarshan S., S. A. (2002). Fundamentos de bases de datos (5.a ed.). McGraw-Hill Interamericana de España S.L.

Kyocera. (2021). Conceptos sobre base de datos orientada a objetos. KYOCERA Document Solutions España S.A. <https://www.kyoceradocumentsolutions.es/es/smarter-workspaces/business-challenges/paperless/conceptos-sobre-base-de-datos-orientada-a-objetos.html>

López, J. (2009). Algoritmos y programación (guía para docentes). <http://www.eduteka.org/GuiaAlgoritmos.php>

MySQL 8.0 Reference Manual. (2021).15.1.20.5 FOREIGN KEY Constraints. MySQL. <https://dev.mysql.com/doc/refman/8.0/en/create-table-foreign-keys.html>

MySQL 8.0 Reference Manual. (2021). 5.3.2 Creating a Table. MySQL. <https://dev.mysql.com/doc/refman/8.0/en/creating-tables.html>

MySQLTutorial. (2020). MySQL AND Operator. MySQL. <https://www.mysqltutorial.org/mysql-and>

MySQLTutorial. (2020). MySQL OR Operator. MySQL. <https://www.mysqltutorial.org/mysql-or>

Créditos

| Nombre | Cargo | Regional y Centro de Formación |
| --- | --- | --- |
| Milady Tatiana Villamil Castellanos | Responsable del Ecosistema | Dirección General |
| Olga Constanza Bermúdez Jaimes | Responsable de Línea de Producción | Centro de Servicios de Salud - Regional Antioquia |
| Peter Emerson Pinchao Solis | Experto Temático | Centro de Teleinformática y Producción Industrial - Regional Cauca |
| David Eduardo Lozada Cerón | Experto Temático | Centro de Teleinformática y Producción Industrial - Regional Cauca |
| Paola Alexandra Moya | Evaluadora Instruccional | Centro de Servicios de Salud - Regional Antioquia |
| Andrés Felipe Herrera Roldán | Diseñador de Contenidos Digitales | Centro de Servicios de Salud - Regional Antioquia |
| Edwin Sneider Velandia Suárez | Desarrollador Fullstack | Centro de Servicios de Salud - Regional Antioquia |
| Edgar Mauricio Cortés García | Actividad Didáctica | Centro de Servicios de Salud - Regional Antioquia |
| Daniela Muñoz Bedoya | Animador y Productor Multimedia | Centro de Servicios de Salud - Regional Antioquia |
| Laura Gisselle Murcia Pardo | Animador y Productor Multimedia | Centro de Servicios de Salud - Regional Antioquia |
| Andrés Felipe Guevara Ariza | Locución | Centro de Servicios de Salud - Regional Antioquia |
| Luis Gabriel Urueta Álvarez | Validador de Recursos Educativos Digitales | Centro de Servicios de Salud - Regional Antioquia |
| Jaime Hernán Tejada Llano | Validador de Recursos Educativos Digitales | Centro de Servicios de Salud - Regional Antioquia |
| Margarita Marcela Medrano Gómez | Evaluador para Contenidos Inclusivos y Accesibles | Centro de Servicios de Salud - Regional Antioquia |
| Daniel Ricardo Mutis Gómez | Evaluador para Contenidos Inclusivos y Accesibles | Centro de Servicios de Salud - Regional Antioquia |