**FORMATO PARA EL DESARROLLO DE COMPONENTE FORMATIVO**

|  |  |
| --- | --- |
| PROGRAMA DE FORMACIÓN | Aseguramiento de la calidad de *software* |

|  |  |  |  |
| --- | --- | --- | --- |
| COMPETENCIA |  | RESULTADOS DE APRENDIZAJE |  |

|  |  |
| --- | --- |
| NÚMERO DEL COMPONENTE FORMATIVO | 9 |
| NOMBRE DEL COMPONENTE FORMATIVO | 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. |
| PALABRAS CLAVE | Bases de datos, consultas, descripción de datos, manipulación de datos, modelo físico de datos. |

|  |  |
| --- | --- |
| ÁREA OCUPACIONAL | N/A |
| IDIOMA | Español |

1. **Tabla de contenidos**

**Introducción**

1. **Interface de línea de comandos de MySQL**
   1. Aplicación cliente de MySQL
   2. Prueba de consultas básicas
   3. SQL – *Standard Query Languaje*
2. **SQL para la creación y eliminación de bases de datos - DDL**
   1. SQL para la creación y eliminación de tablas
   2. Claves foráneas
   3. Mostrar y eliminar tablas y bases de datos
3. **SQL para la manipulación de datos - DML**
   1. Inserción de registros
   2. Edición de registros
   3. Borrado de registros
4. **SQL para consulta de datos - DML**
   1. Funciones en MySQL
   2. Operadores
   3. Ordenar resultados
   4. Listar y limitar resultados
   5. Agrupar filas
5. **Consulta multitablas - DML**
   1. Combinación interna (INNER JOIN)
   2. Combinación externa
6. **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.

1. **DESARROLLO DE CONTENIDOS**
2. **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 un 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 APIs.

* 1. **Aplicación cliente de MySQL**

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

|  |
| --- |
| CF9\_1\_1\_Conectarse BBDD |

En adelante, los resultados de interface de línea de comandos se presentarán como se presenta a continuación:

**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>**

Y, para salir de la consola de comando, se usa el comando *quit* (en mayúscula o minúscula).

**mysql> QUIT**

**Bye**

**C:\mysql\bin>**

* 1. **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 consulta,s se utiliza la sentencia SQL SELECT, por ejemplo:

**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.**

* 1. **SQL – *Standard Query Language***

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

|  |
| --- |
| CF9\_1\_3\_Lenguajes |

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 Workbeach, y se observarán las particularidades específicas de MySQL.

1. **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.

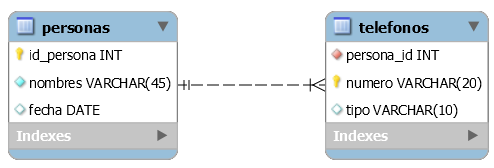
Para iniciar, se creará una base de datos únicamente para practicar, y se llamará "**prueba**":

|  |
| --- |
| CF9\_2\_Prueba |

* 1. **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 ver el PDF “**CREATE TABLE”.**

* 1. **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:

**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)**

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

**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 cambar 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:

**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:

**CREATE TABLE personas2 (**

**id\_persona INT AUTO\_INCREMENT PRIMARY KEY,**

**nombres VARCHAR(45),**

**fecha DATE**

**)ENGINE=InnoDB;**

Y también para la tabla *telefonos2*:

**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**.

|  |
| --- |
| CF9\_2\_2\_Tareas |

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

|  |
| --- |
| CF9\_2\_2\_Opciones |

**Ejemplo**:

**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 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 **teléfonos3**, lo anterior debió que se ha definido restricción **DELETE RESTRICT.**

Si se modifica el valor de la columna **'id\_persona'** en la tabla **'personas3'**, se modificarán los valores de la columna **'persona\_id'** para conservar la analogía.

Observe los datos de las tablas **personas3** y **telefonos3**:

|  |  |
| --- | --- |
| Datos de la tabla *personas3* | Datos de la tabla *telefonos3* |
| 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]> | 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 **'teléfonos3'** 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'**:

* 1. **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**.

**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.

**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.

**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>**

Nota. MySQL 8.0 Reference Manual (2021).

**Ejercicios prácticos**

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.

1. **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:

**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]>**

* 1. **Inserción de registros**

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

|  |
| --- |
| CF9\_3\_1\_Insercion |

* 1. **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).

**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:

|  |
| --- |
| CF9\_3\_2\_Edicion\_de\_registros |

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:

**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]>**

* 1. **Borrado de registros**

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

**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:

**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.

**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]>**

Nota. Tomado de MySQL 8.0 Reference Manual (2021).

**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.

**MariaDB [prueba]> TRUNCATE gente;**

**Query OK, 0 rows affected (0.037 sec)**

**MariaDB [prueba]> SELECT \* FROM gente;**

**Empty set (0.001 sec)**

**MariaDB [prueba]>**

1. **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.

* 1. **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:

**Tabla 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**

|  |
| --- |
| CF9\_4\_1\_Ejemplo |

* 1. **Operadores**

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

1. **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**.

Se ejecuta la siguiente sentencia en la consulta para que mirar la naturaleza de los datos.

**MariaDB [prueba]> SELECT TRUE, FALSE, NULL;**

**+------+-------+------+**

**| TRUE | FALSE | NULL |**

**+------+-------+------+**

**| 1 | 0 | NULL |**

**+------+-------+------+**

**1 row in set (0.000 sec)**

**MariaDB [prueba]>**

Conozcamos algunos operadores:

|  |
| --- |
| CF9\_4\_2\_Operadores\_logicos |

1. **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:

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

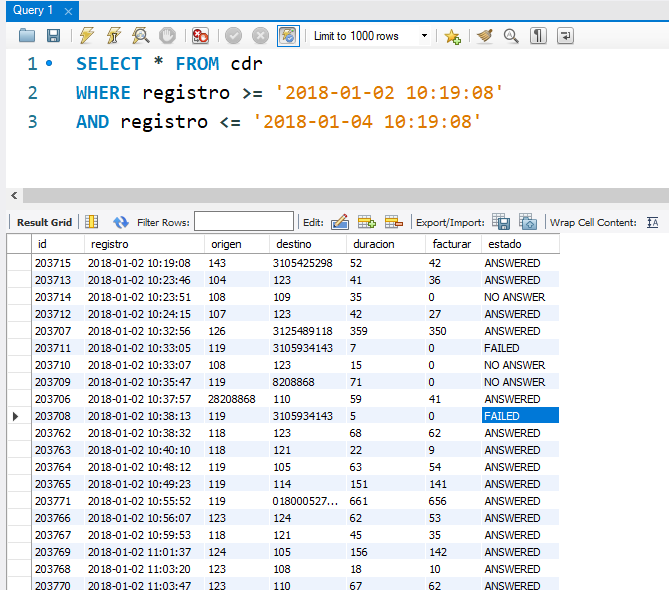
1. **Operadores de desigualdad**

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

**SELECT \* FROM cdr**

**WHERE registro >= '2018-01-02 10:19:08'**

**AND registro <= '2018-01-04 10:19:08'**



* 1. **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.

|  |
| --- |
| CF9\_4\_3\_Ordenar |

* 1. **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**

|  |
| --- |
| CF9\_4\_4\_Listar |

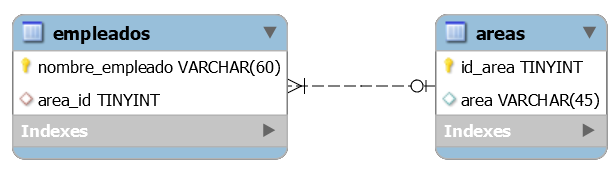
* 1. **Agrupar filas**

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**

|  |
| --- |
| CF9\_4\_5\_Agrupar |

1. **Consulta multitablas - DML**

Para explicar este tipo de ejercicios, crearemos una base de datos de empleados llamada ehr, y las tablas empleados y áreas, 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:

**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:

[**INSERT**](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/5.5/en/insert.html)**INTO areas (id\_area, area) [VALUES](http://localhost/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/5.5/en/miscellaneous-functions.html" \l "function_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:

|  |  |
| --- | --- |
| Empleados | Áreas |
| 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]> | 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]> |

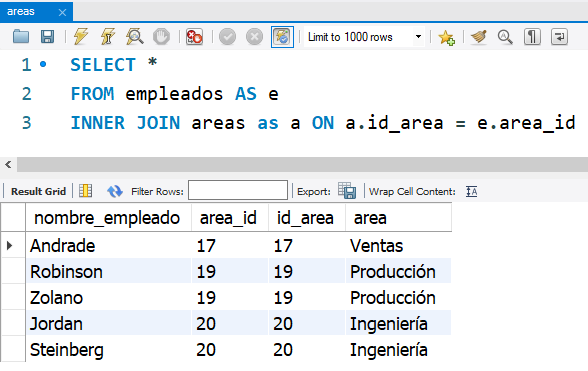
* 1. **Combinación interna (INNER JOIN)**

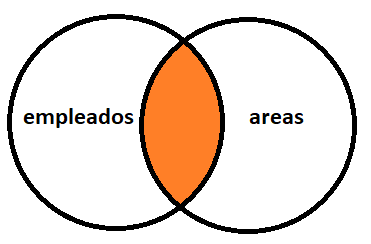
Con esta operación, cada fila en la tabla **empleados** es combinada (o unida) con los correspondientes de la tabla **áreas** que satisface la condición que se especifiquen en el predicado del **JOIN**. Cualquier fila de la tabla **empleados** o de la tabla **áreas** que no tenga uno correspondiente en la otra tabla (**áreas**) 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.

**SELECT \***

**FROM empleados AS e**

**INNER JOIN areas as a ON a.id\_area = e.area\_id**





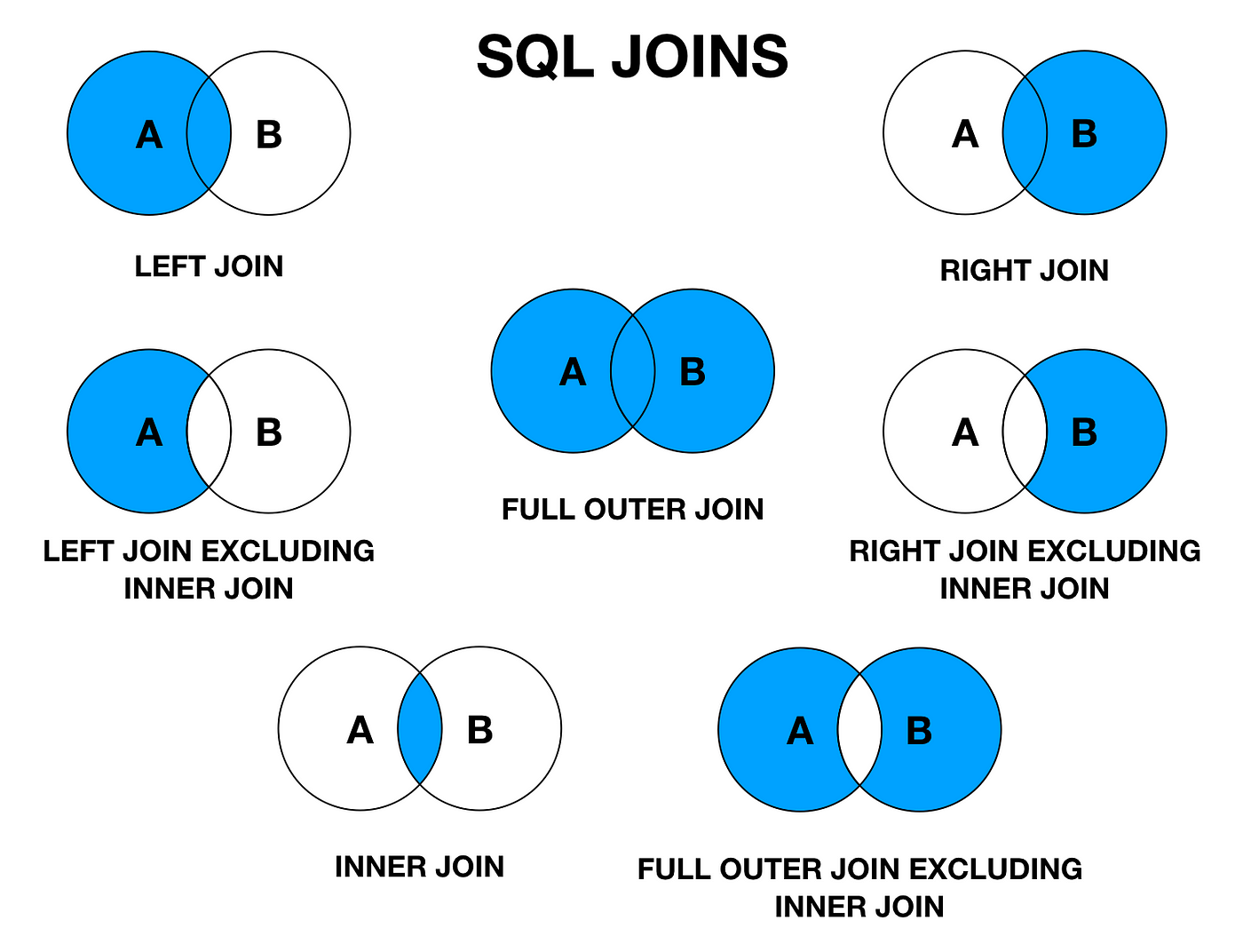
La condición de la relación (**a.id\_area= e.area\_id**) consiste en que debe estar presente en ambas tablas el valor. Cuando el empleado tiene **NULL** en **area\_id** no aparece, debido a que no puede cumplirla. Esto se representa con un Diagrama de Venn en la conjunción, porque el valor debe estar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

**Figura 1**

*SQL JOINS*



* 1. **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**

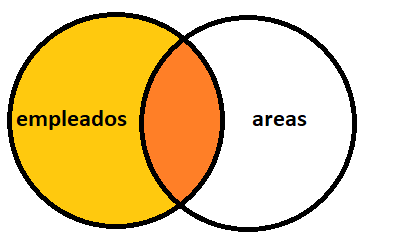
La sintaxis es muy similar a la anterior.

**SELECT \***

**FROM empleados AS e**

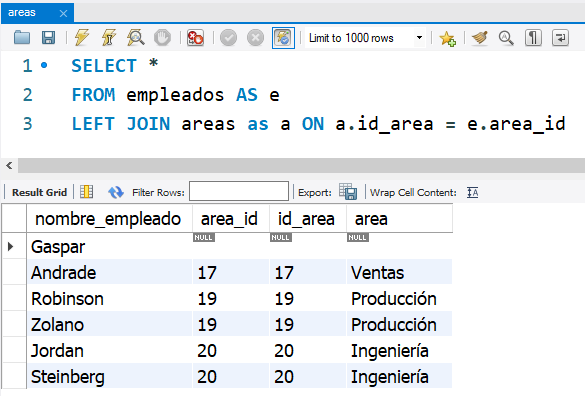
**LEFT JOIN areas as a ON a.id\_area = e.area\_id**

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**).

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 (áreas) 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 áreas.



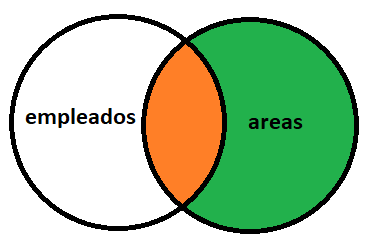
**RIGTH JOIN**

La sentencia **RIGHT JOIN** retorna todos los valores de la tabla derecha (áreas) 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.

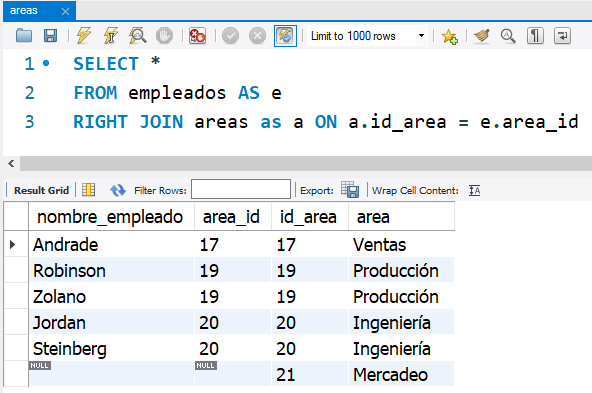
**SELECT \***

**FROM empleados AS e**

**RIGHT JOIN areas as a ON a.id\_area = e.area\_id**

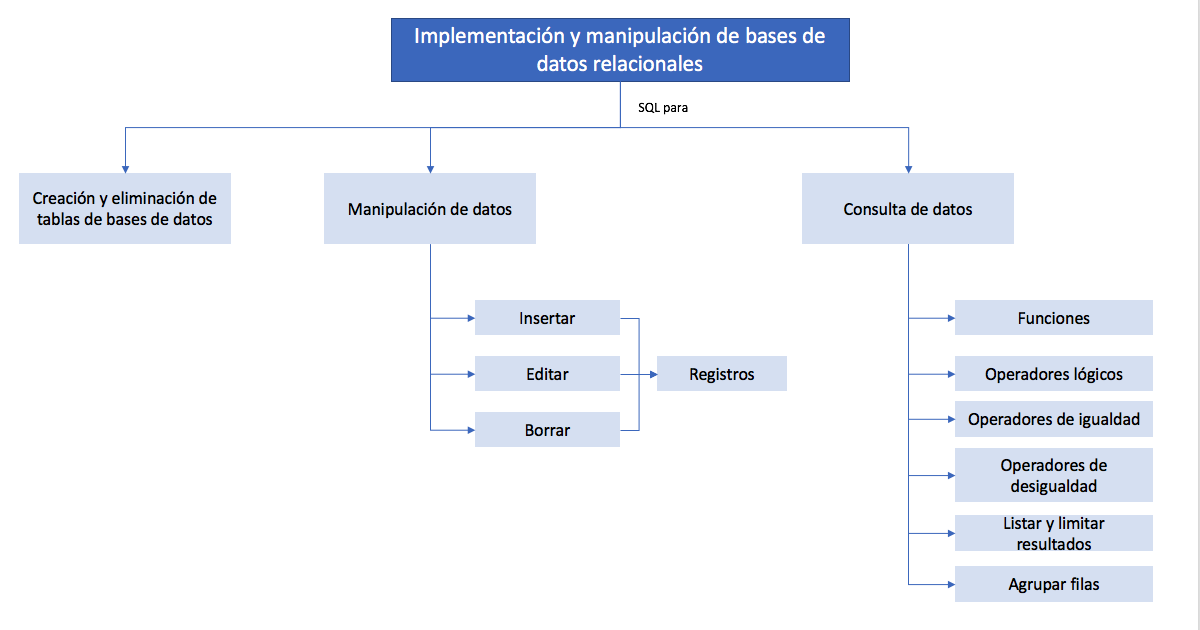
****

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. **SÍNTESIS**

A continuación, se presenta una síntesis de la temática estudiada en el componente formativo.



1. **Actividades didácticas (opcionales si son sugeridas)**

|  |  |
| --- | --- |
| DESCRIPCIÓN DE ACTIVIDAD DIDÁCTICA | |
| Nombre de la Actividad | Funciones de MySQL |
| Objetivo de la actividad | Identificar algunas de las principales funciones de MySQL. |
| Tipo de actividad sugerida | Relación de términos. |
| Archivo de la actividad  (Anexo donde se describe la actividad propuesta) | CF9\_Actividad\_didactica.docx |

1. **Material complementario**

|  |  |  |  |
| --- | --- | --- | --- |
| Tema | Referencia APA del Material | Tipo de material  (Video, capítulo de libro, artículo, otro) | Enlace del Recurso o  Archivo del documento o material |
| Bases de datos | Microsoft. (2021). *Fundamentos de la* *normalización de la base de datos.* | Página web | <https://learn.microsoft.com/es-es/office/troubleshoot/access/database-normalization-description> |

1. **Glosario**

|  |  |
| --- | --- |
| TÉRMINO | SIGNIFICADO |
| Base datos: | conjunto ordenado y finito de operaciones que permite hallar la solución de un problema. |
| Diccionario de datos: | conjunto de programas, instrucciones y reglas informáticas para ejecutar ciertas tareas en una computadora. |
| Metadatos: | conjunto unitario de instrucciones que permite a una computadora realizar funciones diversas, como el tratamiento de textos, el diseño de gráficos, la resolución de problemas matemáticos, el manejo de bancos de datos. |

1. **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).13.1.20.5 FOREIGN KEY Constraints. MySQL. <https://dev.mysql.com/doc/refman/8.0/en/create-table-foreign-keys.html>

MySQL 8.0 Reference Manual. (2021). 3.3.2 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>

1. **Control del documento**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Nombre | Cargo | Dependencia  *(Para el SENA indicar Regional y Centro de Formación)* | Fecha |
| Autor (es) | Henry Eduardo Bastidas Paruma | Instructor | Regional Cauca, Centro de teleinformática y producción industrial | Marzo 25 de 2021 |
| Peter Emerson Pinchao Solis | Instructor | Regional Cauca, Centro de teleinformática y producción industrial | Marzo 25 de 2021 |
| Luz Aída Quintero Velásquez | Diseñadora y evaluadora Instruccional | Centro de Gestión Industrial – Distrito Capital | Junio 2021 |
| Alix Cecilia Chinchilla Rueda | Evaluadora Instruccional | Centro de Gestión Industrial – Distrito Capital | Junio 2021 |
| Jhon Jairo Rodríguez Pérez | Diseñador y evaluador instruccional | Centro para la Industria de la Comunicación Gráfica. Regional Distrito Capital | Junio de 2021 |

1. **Control de cambios**

**(Diligenciar únicamente si realiza ajustes a la Unidad Temática)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Nombre | Cargo | Dependencia | Fecha | Razón del Cambio |
| Autor (es) | Ana Catalina Córdoba Sus | Evaluadora instruccional | Regional Antioquia - Centro de Servicios de Salud | Febrero 2024 | Actualización |
|  | Olga Constanza Bermúdez Jaimes | Responsable Línea de Producción Antioquia | Regional Antioquia - Centro de Servicios de Salud | Febrero 2024 | Actualización |