

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.

**Julio 2024**

Tabla de contenido

[Introducción 5](#_Toc174121293)

[1. Interfaz de línea de comandos de MySQL 7](#_Toc174121294)

[1.1. Aplicación cliente de MySQL 7](#_Toc174121295)

[1.2. Prueba de consultas básicas 9](#_Toc174121296)

[1.3. SQL – Standard Query Language 10](#_Toc174121297)

[2. SQL para la creación y eliminación de bases de datos – DDL 11](#_Toc174121298)

[Valores nulos 16](#_Toc174121299)

[Valores por defecto 17](#_Toc174121300)

[Claves primarias 18](#_Toc174121301)

[Columnas autoincrementadas 19](#_Toc174121302)

[Comentarios 20](#_Toc174121303)

[Descripción de la tabla 20](#_Toc174121304)

[Documentación 21](#_Toc174121305)

[Claves primarias compuestas 22](#_Toc174121306)

[Índices 24](#_Toc174121307)

[Claves únicas 25](#_Toc174121308)

[2.1. Claves foráneas 26](#_Toc174121309)

[2.2. Mostrar y eliminar tablas y bases de datos 34](#_Toc174121310)

[3. Gestión de datos en bases de datos 38](#_Toc174121311)

[3.1. Inserción de registros 38](#_Toc174121312)

[3.2. Edición de registros 42](#_Toc174121313)

[3.3. Borrado de registros 46](#_Toc174121314)

[Vaciar una tabla 49](#_Toc174121315)

[4. SQL para consulta de datos – DML 50](#_Toc174121316)

[4.1. Funciones en MySQL 50](#_Toc174121317)

[4.2. Operadores 56](#_Toc174121318)

[Operadores lógicos 56](#_Toc174121319)

[Operadores de igualdad 59](#_Toc174121320)

[Operadores de desigualdad 60](#_Toc174121321)

[4.3. Ordenar resultados 61](#_Toc174121322)

[4.4. Listar y limitar resultados. 62](#_Toc174121323)

[4.5. Agrupar filas 64](#_Toc174121324)

[5. Consulta multitablas – DML 67](#_Toc174121325)

[5.1. Combinación interna (INNER JOIN) 70](#_Toc174121326)

[5.2. Combinación externa 74](#_Toc174121327)

[LEFT JOIN 74](#_Toc174121328)

[RIGTH JOIN 76](#_Toc174121329)

[Síntesis 79](#_Toc174121330)

[Material complementario 80](#_Toc174121331)

[Glosario 81](#_Toc174121332)

[Referencias bibliográficas 83](#_Toc174121333)

[Créditos 85](#_Toc174121334)

Introducción

Las siglas SQL provienen del inglés Structured Query Language y representan un lenguaje estándar utilizado para manipular los datos almacenados en bases de datos relacionales. Este lenguaje es implementado por casi todos los Sistemas de Gestión de Bases de Datos Relacionales (SGBD), permitiendo realizar una amplia gama de operaciones dentro de la base de datos. En este componente, se presenta una introducción al lenguaje SQL, con un enfoque particular en las sentencias para la consulta de datos. Se utilizará la interfaz de consola de línea de comandos con el objetivo de fortalecer las técnicas de administración remota. Posteriormente, se introducirá la interfaz de administración gráfica, la cual permite realizar operaciones similares de una manera visualmente intuitiva.

Además de las operaciones básicas de consulta, SQL también facilita la modificación, inserción y eliminación de datos, lo cual es esencial para el mantenimiento y la actualización de las bases de datos.

Por ejemplo, las sentencias de actualización permiten modificar registros existentes para reflejar cambios en la información, mientras que las de inserción añaden nuevos datos a la base. Este lenguaje también soporta operaciones más avanzadas como transacciones y procedimientos almacenados, que ayudan a ejecutar secuencias de operaciones de forma segura y eficiente.

Por último, es importante destacar que SQL no solo se utiliza para la manipulación directa de datos. También desempeña un papel crucial en la optimización del rendimiento de las consultas a través de la indexación, la partición de tablas y otras técnicas de optimización de bases de datos. Estas características hacen de SQL una herramienta indispensable en el campo de la administración de bases de datos, ofreciendo soluciones eficaces para gestionar grandes volúmenes de datos de manera efectiva.

# Interfaz de línea de comandos de MySQL

Al comenzar a trabajar con un Sistema de Gestión de Bases de Datos (SGBD) como MySQL, existen diversas formas de establecer comunicación con el servidor. Habitualmente, en el software se utiliza una API para realizar consultas y otras operaciones con el servidor. Por ejemplo, en PHP, esta API está integrada directamente en el lenguaje.

En este apartado, se empleará MySQL directamente a través de un cliente que se ejecuta desde una consola (una ventana de comandos en Windows). En secciones posteriores, se explicarán las diferentes APIS disponibles.

## Aplicación cliente de MySQL

Dado que ya se conocen los métodos para ingresar y salir del cliente de MySQL y se dispone de la capacidad para efectuar consultas, resulta sencillo consultar múltiples variables del sistema o evaluar diversas funcionalidades de MySQL. Para llevar a cabo estas consultas, se emplea la sentencia SQL SELECT. Por ejemplo:

* **Paso 1**

Debe abrir inicio y buscar MySQL o teclear “mysql” y se hace Clic en MySQL Command Line Client.

* **Paso 2**

Ingresa la contraseña de root usada en el proceso de instalación de MySQL.

* **Paso 3**

Una vez autenticado en la interface comando de línea se aprecia este resultado.

* **Paso 4**

En adelante los resultados de la interfaz de línea de comandos se presentarán como sigue:

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 brmay be trademarks of

their respective owners.

Type ‘help;’ or ‘\h’ for help. Type ‘\c’ to clear the current input statement.

mysql>

* **Paso 5**

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

mysql> QUIT

Bye

C:\mysql\bin>

## Prueba de consultas básicas

Dado que ya se conocen los métodos para ingresar y salir del cliente de MySQL y se dispone de la capacidad para efectuar consultas, resulta sencillo consultar múltiples variables del sistema o evaluar diversas funcionalidades de MySQL. Para llevar a cabo estas consultas, se emplea 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 utilizada para seleccionar datos de bases de datos. Además, se puede emplear, como en esta situación, para consultar variables del sistema o resultados de funcionalidades. En este caso, se ha consultado el resultado de la función VERSION. Sabemos que es una función porque tiene paréntesis "()", y también se ha consultado la variable CURRENT\_DATE.

**¡Nota!**

Toda sentencia debe finalizar con un punto y coma (;), seguido de presionar Enter.

## SQL – Standard Query Language

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

* **Data Definition Language DDL**

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.

* **Data Manipulation Language DML**

Es lo que se usa para cambiar y obtener datos desde las bases de datos.

SQL abarca ambos lenguajes DDL + DML y serán estudiados en esta unidad, ya que ambos hacen parte del conjunto de sentencias de SQL.

En este punto se explicará el proceso para convertir del modelo lógico relacional, al modelo físico, empleando sentencias SQL.

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

Cada grupo de entidades que tiene un modelo completo forma una base de datos. Desde la perspectiva de SQL, una base de datos es simplemente un conjunto de relaciones (o tablas), y se organizan o diferencian accediendo a ellas por su nombre. A nivel del sistema operativo, MySQL almacena cada base de datos en un directorio distinto.

Por lo tanto, crear una base de datos es un proceso bastante sencillo. Sin embargo, al crearla, inicialmente estará vacía, es decir, no contendrá ninguna tabla.

* **¡Nota!**

Es esencial ejecutar las sentencias propuestas aquí mientras se estudia; de esta manera, se consolidarán los conocimientos adquiridos sobre cada tema tratado. Es necesario tomarse el tiempo para escribir en la consola cada una de las sentencias, ya que este ejercicio es fundamental para asimilar el conocimiento que se está adquiriendo.

De igual forma, se deben revisar los resultados obtenidos tras la ejecución; en caso de error en la sintaxis, la respuesta ofrecerá una indicación de lo que podría estar fallando.

Se procederá a crear y manejar una base de datos de práctica mientras se familiariza con el modo de operación de MySQL.

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

* **Paso 1**

Se usa la sentencia CREATE DATABASE, para crear una base de datos.

mysql> USE prueba;

Database changed

mysql>

* **Paso 2**

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

mysql> SHOW DATABASES;

+——————————+

| Database |

+——————————+

| arnecia |

| information\_schema |

| mysql |

| performance\_schema |

| prueba |

| sys |

+——————————+

6 rows in set (0.00 sec)

mysql>

* **Paso 3**

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

mysql> USE prueba;

Database changed

mysql>

* **Paso 4**

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:

mysql> DROP DATABASE prueba;

Query OK, 0 rows affected (0.01 sec)

* **Paso 5**

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

mysql> SHOW DATABASES;

+——————————+

| Database |

+——————————+

| arnecia |

| 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>

A continuación, se presenta la sentencia CREATE TABLE, que se utiliza para la creación de tablas. Esta sentencia posee una sintaxis algo compleja debido a la variedad de opciones disponibles para configurar una tabla. Se explorarán estas opciones gradualmente, y en breve se dominarán muchas de ellas.

En su forma más sencilla, la sentencia CREATE TABLE permite crear una tabla especificando todas las columnas requeridas. A continuación, se muestra un ejemplo de una tabla diseñada para almacenar el nombre de varias personas y sus respectivas fechas de nacimiento. Es necesario indicar el nombre de la tabla, así como los nombres y tipos de datos de cada columna:

Se creará una tabla denominada gente con dos columnas: nombre, que acepta cadenas de hasta 40 caracteres, y fecha, de tipo fecha.

mysql> USE prueba;

Database changed

mysql> CREATE TABLE gente (nombre VARCHAR(40), fecha DATE);

Query OK, 0 rows affected (0.53 sec)

mysql>

Se puede examinar cuántas tablas y qué nombres tienen en una base de datos, empleando la sentencia SHOW TABLES:

mysql> SHOW TABLES;

+----------------+

| Tables\_in\_prueba |

+----------------+

| gente |

+----------------+

1 row in set (0.01 sec)

mysql>

### Valores nulos

Al definir cada columna, se puede especificar si podrá o no contener valores nulos. Es importante recordar que las columnas que forman parte de una clave primaria no pueden almacenar valores nulos.

Cuando se define una columna como clave primaria, automáticamente se impide que contenga valores NULL (nulo). Sin embargo, no solo en este caso puede ser relevante restringir la asignación de valores nulos en una columna.

Por defecto, las columnas admiten valores nulos (NULL). Para que no se permitan, se utiliza la restricción NOT NULL. Por ejemplo:

mysql> CREATE TABLE ciudad (nombre VARCHAR(20)

NOT NULL, poblacion INT NULL);

Query OK, 0 rows affected (0.53 sec)

Listar las tablas existentes hasta ahora:

mysql> SHOW TABLES;

+----------------+

| Tables\_in\_prueba |

+----------------+

| gente |

+----------------+

1 row in set (0.01 sec)

mysql>

### Valores por defecto

En una columna, se puede especificar opcionalmente un valor por defecto. Este valor por defecto se asignará automáticamente a la columna cuando no se especifique un valor concreto al añadir un registro. Si una columna permite valores nulos y no se especifica un valor al insertar, el valor por defecto será NULL. Por ejemplo, en el caso mencionado anteriormente, el valor por defecto para la columna poblacion es NULL.

Si se desea que el valor por defecto sea 50,000 para la columna poblacion, la tabla se creará de la siguiente manera:

CREATE TABLE ciudad2 (nombre VARCHAR(20) NOT NULL,

poblacion INT NULL DEFAULT 5000);

A continuación, se debe redactar la sentencia, pero esta vez se hará por partes:

1. Primero se escribirá: CREATE TABLE ciudad2 (nombre VARCHAR(20) NOT NULL,
2. Se presiona la tecla ENTER.
3. Luego se escribe el resto de la sentencia poblacion INT NULL DEFAULT 50000)

Recuerde que MySQL interpreta una sentencia solo cuando encuentra el carácter punto y coma (;). El resultado será:

mysql> CREATE TABLE ciudad2 (nombre VARCHAR(20) NOT NULL,

poblacion INT NULL DEFAULT 50000);

Query OK, 0 rows affected (0.29 sec)

### Claves primarias

Es posible definir una clave primaria para una columna utilizando la palabra clave [PRIMARY KEY]. Cada tabla puede tener solo una clave primaria, y la columna designada como tal no puede contener valores NULL, como se mencionó anteriormente. Si esto no se especifica explícitamente, MySQL lo gestionará automáticamente.

Por ejemplo, si se desea crear un índice para la columna nombre en la tabla ciudad3, la tabla se crearía de la siguiente manera:

mysql> CREATE TABLE ciudad3 (nombre VARCHAR(20) NOT NULL PRIMARY KEY,

poblacion INT NULL DEFAULT 50000);

Query OK, 0 rows affected (0. sec)

* **¡Nota!**

En la terminología SQL estándar, el término KEY se utiliza generalmente en la definición de índices y PRIMARY KEY para claves primarias específicas.

El uso de NOT NULL PRIMARY KEY equivale a PRIMARY KEY en cuanto a la restricción de valores nulos y la definición de la clave primaria. Es importante destacar que PRIMARY KEY implica automáticamente NOT NULL. Por lo tanto, PRIMARY KEY, NOT NULL o simplemente KEY pueden parecer equivalentes en términos generales, pero en la práctica, KEY se utiliza más comúnmente para definir índices que no necesariamente son claves primarias.

* **¡Nota!**

Existen sintaxis más específicas para crear claves primarias que son generalmente preferibles por ser más potentes. Por ejemplo, la sintaxis general permite crear claves primarias compuestas por varias columnas, algo que no es posible con el alias simplificado que se ha explicado anteriormente.

### Columnas autoincrementadas

En MySQL, es posible crear una columna autoincrementada; sin embargo, esta debe ser de tipo entero y actuar como clave primaria. Cuando se inserta una fila y se omite el valor de la columna autoincrementada, o se intenta insertar un valor nulo en ella, MySQL calculará automáticamente su valor. Esto se hace tomando el valor más alto actual en esa columna y sumándole uno, generando así una secuencia. Este mecanismo garantiza un valor único y secuencial para cada fila de la tabla.

Estas columnas son comúnmente utilizadas como claves primarias secuenciales. SQL limita las columnas autoincrementadas a números enteros, lo que significa que no se puede aplicar autoincremento a columnas de otro tipo. Por ello, una columna que sea entera y autoincrementada resulta ideal para ser utilizada como clave primaria artificial.

mysql> CREATE TABLE ciudad4 (clave INT AUTO\_INCREMENT PRIMARY KEY,

nombre VARCHAR(20) NOT NULL,

poblacion INT NULL DEFAULT 50000);

Query OK, 0 rows affected (0.21 sec)

### Comentarios

Adicionalmente, al momento de crear la tabla, se puede agregar un comentario a cada columna. Este comentario sirve como información adicional sobre características específicas de la columna y se incluye en el apartado de documentación de la base de datos. Esto facilita la comprensión y el mantenimiento del esquema de la base de datos por otros desarrolladores o por uno mismo en el futuro.

mysql> CREATE TABLE ciudad5

(clave INT AUTO\_INCREMENT PRIMARY KEY COMMENT 'Esta es Clave principal',

nombre VARCHAR(50) NOT NULL,

poblacion INT NULL DEFAULT 50000);

Query OK, 0 rows affected (0.08 sec)

### Descripción de la tabla

SQL tiene una sentencia que permite revisar la descripción de una tabla; esta sentencia es DESC <nombre\_tabla>.

mysql> DESC ciudad5;

+-----------+----------+------+-----+----------+----------------+

| Field | Type | Null | Key | Default | Extra |

+-----------+----------+------+-----+---------+----------------+

| clave | int | NO | PRI | NULL | auto\_increment |

| nombre | char(50) | NO | | NULL | |

| poblacion | int | YES | | 5000 | |

+-----------+----------+------+-----+---------+----------------+

3 rows in set (0.01 sec)

### Documentación

SQL ofrece diversas opciones al momento de definir columnas. Además de especificar el tipo y el nombre, es posible determinar valores por defecto, permitir o prohibir valores tipo NULL, crear claves primarias, indexar, entre otras funciones. La sintaxis para definir las columnas es:

nombre\_columna tipo\_dato [NOT NULL | NULL]

[DEFAULT valor\_por\_defecto] [AUTO\_INCREMENT]

[PRIMARY KEY] [COMMENT ‘string’] [definición\_referencial]

Esta documentación se puede leer de la siguiente forma:

* nombre\_columna: es el nombre que se le dará a la columna a definir.
* tipo\_dato: es el tipo de datos de la columna (INT, FLOAT, VARCHAR).
* [NOT NULL | NULL]: todo lo que se encuentre encerrado entre corchetes ([ ]) significa que es opcional; puede incluirse o no. Dentro de las opciones, lo que está separado por el carácter "pipe" (|) significa que debe escogerse una de las opciones separadas por el "pipe".
* DATE NOT NULL: si se quiere definir un campo que se llame fecha de tipo DATE y que no sea nullable.

Si quiere que el valor por defecto de esta columna sea la fecha actual del sistema puedo usar la sintaxis DEFAULT CURRENT\_DATE así:

fecha DATE NOT NULL DEFAULT CURRENT\_DATE

### Claves primarias compuestas

La sintaxis para definir claves primarias es:

definición\_columnas

PRIMARY KEY (index\_nombre\_col, …)

Usando esta sintaxis, el ejemplo anterior para crear claves primarias quedaría de la siguiente forma:

mysql> CREATE TABLE ciudad6 (nombre CHAR(20) NOT NULL,

poblacion INT NULL DEFAULT 5000,

PRIMARY KEY (nombre));

Query OK, 0 rows affected (0.17 sec)

Sin embargo, esta metodología ofrece una mayor flexibilidad; por ejemplo, entre los paréntesis se pueden incluir varios nombres de columnas para formar claves primarias compuestas por múltiples columnas:

mysql> CREATE TABLE persona (

tipo\_documento CHAR(2) NOT NULL,

documento\_identidad CHAR(10) NOT NULL,

nombre CHAR(30),

PRIMARY KEY (tipo\_documento, documento\_identidad));

Query OK, 0 rows affected (0.09 sec)

mysql>

Una vez definida la tabla, es importante prestar especial atención a la columna de salida llamada 'Key':

mysql> DESC persona;

+--------------------+----------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+--------------------+----------+------+-----+---------+-------+

| tipo\_documento | char(2) | NO | PRI | NULL | |

| documento\_identidad| char(10) | NO | PRI | NULL | |

| nombre | char(30) | YES | | NULL | |

+--------------------+----------+------+-----+---------+-------+

3 rows in set (0.00 sec)

mysql>

### Índices

Existen tres tipos de índices en bases de datos. El primero se relaciona con las claves primarias, que, como se ha visto, también se pueden crear en la sección de definición de columnas.

#### ¿Cuándo es oportuno crear índices?

* **Cuando se define una llave foránea:**

Esto ayuda a mejorar el rendimiento de las operaciones de consulta y actualización que involucran la llave foránea.

* **Cuando se anticipa que se realizarán consultas directas:**

Crear índices en columnas frecuentemente consultadas puede significativamente acelerar el tiempo de respuesta en las búsquedas.

El segundo tipo de índice permite definir índices sobre una columna específica. Para establecer estos índices, se pueden utilizar indistintamente las opciones KEY o INDEX:

mysql> CREATE TABLE indices1 (

-> id INT,

-> nombre VARCHAR(20),

-> INDEX (nombre));

Query OK, 0 rows affected (0.14 sec)

O su semejante:

mysql> CREATE TABLE indices2 (

|-> id INT,

|-> nombre VARCHAR(20),

|-> KEY (nombre));

Query OK, 0 rows affected (0.16 sec)

### Claves únicas

El tercer tipo de índices se utiliza para crear claves únicas, ya sea sobre una columna o varias. Para definir índices con claves únicas, se debe usar la opción UNIQUE.

mysql> CREATE TABLE indices3 (

-> id INT,

-> nombre CHAR(19),

-> UNIQUE (nombre));

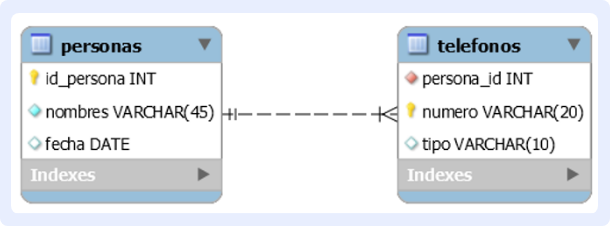
Query OK, 0 rows affected (0.09 sec)

La diferencia entre un índice único y un índice normal radica en que los índices únicos no permiten la inclusión de filas con datos duplicados en la columna o columnas de la tabla. Sin embargo, es importante tener en cuenta una excepción: el valor NULL, que sí se puede repetir varias veces en un índice único.

## Claves foráneas

En MySQL, el soporte para claves foráneas está limitado a las tablas del tipo InnoDB. Aunque InnoDB es el motor de almacenamiento por defecto en las últimas versiones de MySQL, esto no impide su uso en otros tipos de tablas.

1. Claves foráneas



La figura describe una relación entre dos tablas en una base de datos utilizando claves foráneas. La tabla "personas" contiene los campos id\_persona (INT), nombres (VARCHAR(45)) y fecha (DATE). La tabla "telefonos" incluye los campos persona\_id (INT), numero (VARCHAR(20)) y tipo (VARCHAR(10)). La relación entre ambas tablas se establece a través de la clave foránea persona\_id en la tabla "telefonos", que hace referencia al campo id\_persona en la tabla "personas".

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 desea crear una base de datos que refleje el modelo presentado en el diagrama anterior, las sentencias para la tabla teléfonos con la referencia a la tabla persona serían las siguientes:

CREATE TABLE telefonos (

pokemon\_id INT NOT NULL REFERENCES pokemons (id) ON DELETE CASCADE ON UPDATE CASCADE,

numero VARCHAR(20) PRIMARY KEY,

tipo VARCHAR(10)

);

Se ha utilizado una definición como referencia para la columna numero de la tabla telefonos, indicando que es una clave foránea asociada a la columna id\_persona de la tabla persona mediante la columna persona\_id. Aunque la sintaxis es correcta, es importante destacar las funcionalidades de las expresiones DELETE CASCADE y UPDATE CASCADE en términos de la integridad referencial.

De manera similar, la expresión UPDATE CASCADE asegura que, si el id\_persona en una fila de la tabla persona se actualiza a otro valor y existen registros en la tabla telefonos con el valor original en la columna persona\_id, estos registros en telefonos actualizarán su valor en la columna persona\_id para reflejar el cambio. Este procedimiento también es una forma de restricción de integridad referencial.

— Creación de la tabla ‘personas’

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)

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)

Revisemos 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 tiene una definición de clave foránea esté indexada mediante KEY(numero). Sin embargo, esto no debe generar preocupación, ya que si no se especifica explícitamente, MySQL realizará la indexación de manera implícita.

Esta configuración requiere una clave foránea en la columna persona\_id, que hace referencia a la columna id\_persona de la tabla personas2 (FOREIGN KEY (persona\_id) REFERENCES personas2(id\_persona)). La definición incluye las acciones a realizar cuando se elimina una fila en la tabla personas2.

* **ON DELETE <option>**

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

* **ON UPDATE <option>**

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.

Hay cinco opciones distintas. A continuación, se explica lo que hace cada una de ellas:

RESTRICT

Impide borrar o editar filas en la tabla referenciada si existen filas con el mismo valor de clave foránea.

CASCADE

Borrar o modificar una clave en una fila en la tabla referenciada conlleva eliminar las filas con el mismo valor de clave foránea o cambiar los valores de esas claves foráneas.

SET NULL

Al borrar o editar una clave en una fila en la tabla referenciada, se establece el valor NULL en 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 conlleva asignar el valor por defecto a las claves foráneas con el mismo valor.

**Ejemplo:**

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)

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)

Si se intenta eliminar una fila de la tabla personas3 con un cierto valor en id\_persona, se producirá un error si existen filas en la tabla telefonos3 en la columna persona\_id con el mismo valor. La fila en personas3 no será eliminada a menos que se eliminen previamente las filas correspondientes en la tabla telefonos3 que comparten el mismo valor de clave foránea, debido a que se ha definido una restricción de tipo DELETE RESTRICT.

Si se modifica el valor de la columna id\_persona en la tabla personas3, también se deberán modificar los valores de la columna persona\_id en la tabla telefonos3 para mantener la integridad referencial.

Revisar los datos de las tablas personas3 y telefonos3.

+------------+-----------+------------+

| id\_persona | nombre | fecha |

+------------+-----------+------------+

| 1 | Fulanito | 2021-06-09 |

| 2 | Menganito | 2021-06-06 |

| 3 | Tutanito | 2021-06-23 |

| 4 | Migueíto | 2021-06-22 |

+------------+-----------+------------+

4 rows in set (0.000 sec)

MariaDB [prueba]>

Datos de la tabla personas3.

+--------+-----------+

| numero | persona\_id|

+--------+-----------+

| 234279534 | 1 |

| 654345534 | 1 |

| 578819343 | 3 |

| 234294276 | 4 |

| 234293465 | 4 |

| 156096465 | 3 |

+--------+-----------+

7 rows in set (0.000 sec)

MariaDB [prueba]>

Datos de la tabla telefonos3.

* Si se intenta eliminar la fila correspondiente a "Fulanito", se producirá un error debido a que existen dos filas en la tabla telefonos3 con el valor 1 en la columna persona\_id.
* Será posible eliminar la fila correspondiente a "Menganito", ya que no hay ninguna fila en la tabla telefonos3 con el valor 2 en la columna persona\_id.
* Si cambiamos el valor de id\_persona en la fila correspondiente a "Tulanito" por el valor 3, por ejemplo, este nuevo valor, 3, se asignará a la columna persona\_id en las filas tercera y sexta de la tabla telefonos3.

## Mostrar y eliminar tablas y bases de datos

De vez en cuando es preciso eliminar una tabla, ya 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 puede 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, 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>

#### Ejercicios prácticos

Se propone revisar y estudiar dos ejercicios de repaso incluidos en el documento adjunto “**Ejercicios Prácticos**”. Estos ejercicios, completamente desarrollados y un poco más extensos, abarcan la temática vista hasta ahora, con la diferencia de que los problemas propuestos son más cercanos a situaciones de la vida real.

Lo invitamos a consultar el archivo “**Ejercicios prácticos**”, el cual se encuentra en la carpeta Anexos.

# Gestión de datos en bases de datos

Una base de datos sin datos no es muy útil. Por lo tanto, vamos a revisar cómo añadir, modificar o eliminar los datos contenidos en las bases de datos.

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]>

## Inserción de registros

La forma más directa de añadir una fila nueva a una tabla es mediante una sentencia INSERT. En la versión más sencilla de esta sentencia, se debe indicar la tabla a la que se quieren agregar filas y los valores para cada columna. Las columnas de tipo cadena o fecha deben estar entre comillas sencillas o dobles, aunque esto no es necesario para las columnas numéricas, las cuales también pueden estar entrecomilladas.

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 (0.00 sec)

MariaDB [prueba]>

Si no es necesario determinar un valor preciso para alguna columna, se puede asignar el valor por defecto especificado para esa columna al momento de crear la tabla, utilizando la palabra DEFAULT.

MariaDB [prueba]> INSERT INTO gente (nombre, fecha, edad) VALUES (‘Perillo’, DEFAULT, DEFAULT);

Query OK, 1 row affected (0.005 sec)

MariaDB [prueba]>

Revise como Perillo tiene el valor por defecto definido en la columna fecha y edad.

MariaDB [prueba]> SELECT \* FROM gente;

+---------+------------+------+

| nombre | fecha | edad |

+---------+------------+------+

| Fulano | 1974-04-12 | 18 |

| Mengano | 1978-06-15 | 20 |

| Pegano | 1993-02-10 | 33 |

| Perillo | 2020-02-02 | 0 |

| Tulano | 2000-12-02 | 22 |

+---------+------------+------+

5 rows in set (0.000 sec)

MariaDB [prueba]>

El orden en la sentencia INSERT es muy importante, conserve una correspondencia como se presenta en la siguiente imagen.

1. Correspondencia en la sentencia INSERT



Esta imagen presenta un ejemplo de una consulta SQL para insertar datos en una tabla llamada "gente". La consulta agrega un registro con los valores Fulano para el campo nombre, 1974-04-12 para el campo fecha, y 18 para el campo edad.

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

## 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.

UPDATE [LOW\_PRIORITY] [IGNORE] tbl\_name

SET col\_name1=expr1 [, col\_name2=expr2 …]

[WHERE where\_definition]

[em [ORDER BY …]

[em [LIMIT row\_count]

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

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 |

+————+——————+

5 rows in set (0.000 sec)

MariaDB [prueba]>

Supongamos que es necesario incrementar la edad en 10 años para cada fila de la tabla ‘gente’:

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 |

+---------+------------+------+

5 rows in set (0.000 sec)

MariaDB [prueba]>

Sin embargo, no es necesario actualizar todas las filas de la tabla, ya que se puede definir cuáles filas se desean actualizar. Esto se logra mediante la cláusula WHERE, que permite establecer una condición. Solo las filas que cumplan con dicha condición serán actualizadas.

Esta versión clarifica el uso de la cláusula WHERE y mejora la estructura de las oraciones para una lectura más fluida.

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 los que sean mayores de 40 años, 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]>

## 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 |

| Tulano | 2012-12-12 | 55 |

| Pegano | 2012-12-12 | 43 |

+---------+------------+------+

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.

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 desea borrar todas las filas de una tabla, es posible utilizar la sentencia DELETE sin especificar condiciones, como se mencionó anteriormente. Sin embargo, existe una alternativa más rápida: la sentencia TRUNCATE. A diferencia del DELETE, que elimina las filas de manera secuencial, TRUNCATE remueve todos los datos borrando y recreando la tabla vacía, lo cual resulta mucho más eficiente.

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 información dentro de una base de datos, es crucial entender la estructura y naturaleza de los datos. Por ejemplo, los datos de tipo fecha se estructuran en el formato YYYY-MM-DD. Si se necesita listar registros de un año específico, es necesario emplear una función que pueda referenciar ese dato particular dentro de la columna. Por esta razón, es importante revisar algunas funciones específicas antes de comenzar a trabajar con consultas de datos.

## Funciones en MySQL

Considerando que MySQL es versátil en lo que respecta a operadores y también muy extenso en cuanto a funciones, dispone de una gran variedad de ellas. Sin embargo, solo se explicarán algunas funciones esenciales para entender cómo funcionan en general. Algunas funciones son:

1. Funciones en MySQL

| Función | Definición |
| --- | --- |
| 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, utilizaremos una base de datos de llamadas telefónicas. La estructura de esta base de datos se presentará a continuación, y con ella aprenderemos a realizar consultas.

#### Ejemplo llamadas telefónicas:

1. 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 llamadas cdr y dentro una tabla con el mismo nombre.
2. Importe el archivo con la sentencia source

<ruta del archivo sql>

MariaDB [prueba]> source D:/Peter/Downloads/cdr.sql;

#### BASE DE DATOS

Para crear la base de datos cdr., Lo invitamos a consultar el archivo “**Base de datos**”, el cual se encuentra en la carpeta Anexos.

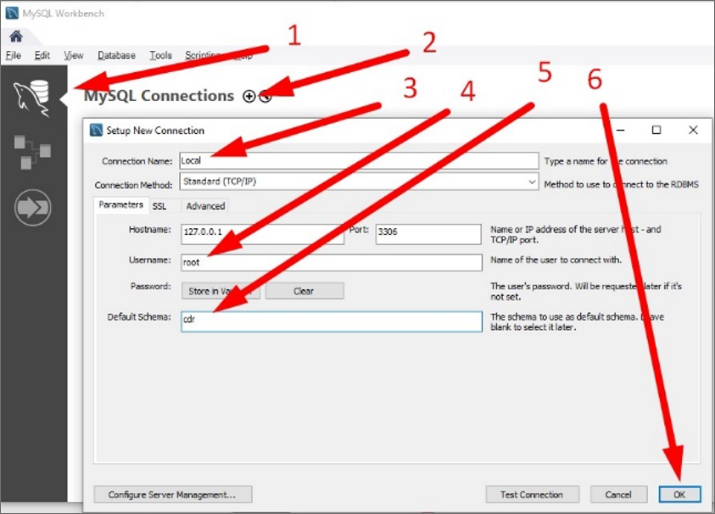
Conozca los datos de prueba con los que se trabajará.

1. Diccionario de datos

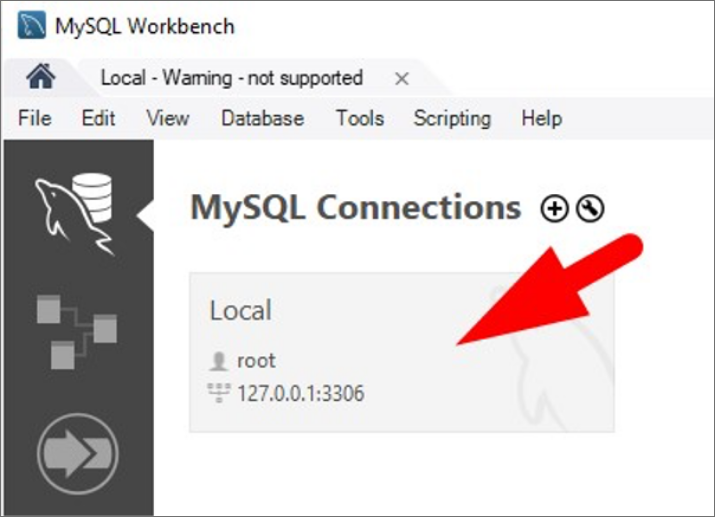
| Función | Tipo | Longitud | Descripción |
| --- | --- | --- | --- |
| Registro | DATETIME | 0 | Fecha y hora de registro de la llamada. |
| Origen | VARCHAR | 80 | Número de teléfono o extensión del llamante. |
| Destino | VARCHAR | 80 | Número de teléfono o extensión del llamado o destino. |
| Estado | 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. |
| Facturas | 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. MySQL WorkBench



1. Dé clic en la pestaña de uso de bases de datos.
2. Cree una nueva conexión.
3. Póngale un nombre que usted prefiera a esa conexión.
4. Ponga el usuario root.
5. Digite el nombre de la base de datos que usará: cdr.
6. Dele clic en ok.
7. Representación del uso de MySQL WorkBeach



1. Ahora debe conectar dando doble clic.

Ya contamos con bases de datos y sabemos cómo agregar y modificar datos. A continuación, aprenderemos a extraer datos de una base de datos utilizando nuevamente la sentencia SELECT.

La sintaxis de SELECT puede ser complicada; sin embargo, en este punto no se explicarán todas sus opciones. Una forma más general de presentar su sintaxis es la siguiente:

* **Listar todos los elementos**

Para listar todos los elementos de la tabla cdr, se utiliza el comodín \*, que indica que se deben seleccionar todas las columnas.

SELECT \* FROM cdr

* **Selección de columnas específicas**

Para seleccionar solo algunas columnas específicas de la tabla cdr, se enumeran las columnas deseadas, separadas por comas.

SELECT origen, destino, duracion FROM cdr

* **Uso de alias en columnas**

Asignar un alias a las columnas en las consultas SQL usando la palabra clave AS, que es opcional. Aquí, a la columna origen se le asigna el alias num\_orig y a duracion el alias tiempo\_llamada.

SELECT origen AS num\_orig, destino, duracion AS tiempo\_llamada FROM cdr

* **Cálculo de cobro por llamadas**

Para calcular el cobro de llamadas salientes a números de celular o a larga distancia nacional con 10 dígitos en el destino, primero se filtran estas llamadas. Se usa la función LENGTH para identificar las que tienen 10 dígitos. Luego, se calcula el número de minutos dividiendo los segundos de la duración por 60 y redondeando hacia arriba con la función CEIL. El valor final se obtiene multiplicando el número de minutos redondeado por el costo por minuto, que es $85.

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.

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]>

#### Operador AND

SQL 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, sólo si ambos operandos son verdaderos y, es falso si cualquier operando es falso o nulo, o si alguno de los dos es nulo. Esto se representa mediante la siguiente tabla de verdad:

1. Tabla de verdad operador AND

| 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 |

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

SQL este operador también tiene dos formas equivalentes OR. El operador OR también es binario. Si ambos operandos siendo diferentes de NULL, el resultado es verdadero si alguno de ellos es verdadero y, es falso si los dos son falsos. Si uno de los operandos es NULL el resultado será verdadero si el otro operando es verdadero, y será NULL si el otro operando es falso.

La tabla de verdad es:

1. Tabla de verdad operador OR

| 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 |

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)

### 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’

### Operadores de desigualdad

En SQL, los operadores de desigualdad son utilizados para filtrar resultados en una consulta basada en comparaciones numéricas entre columnas o valores fijos.

1. Operadores de desigualdad

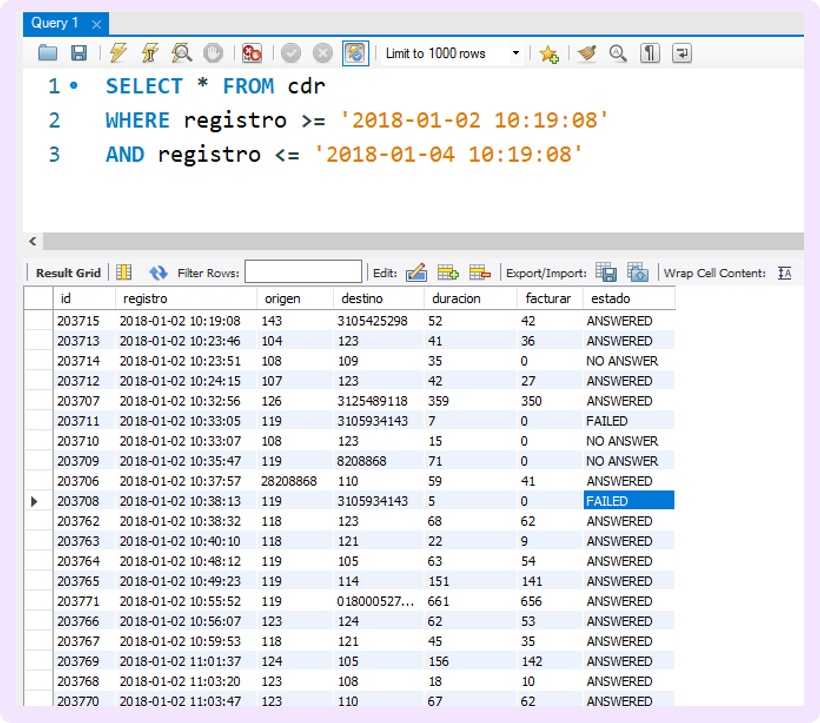
| Operador | Descripción |
| --- | --- |
| <= | Menor o igual |
| < | Menor |
| > | Mayor |
| >= | Mayor o igual |

SELECT \* FROM cdr

WHERE registro >= ‘2018-01-02 10:19:08’

AND registro <= ‘2018-01-04 10:19:08

1. Operadores de desigualdad



## Ordenar resultados

Se puede agregar una cláusula ORDER BY para ordenar los resultados por la columna deseada en orden ascendente (ASC) o descendente (DESC).

* **Orden ascendente por fecha**

Ejemplo para ordenar de manera ascendente por la fecha: SELECT \* FROM cdr ORDER BY registro ASC.

* **Ordenar por resultado de operación**

Ejemplo para ordenar llamadas de larga distancia o celular a 85 pesos el minuto, por el costo mayor al menor: SELECT \*, CEIL(facturar/60) \* 85 AS valor FROM cdr WHERE LENGTH(destino) = 10 ORDER BY CEIL(facturar/60) \* 85 DESC.

* **Agrupar por múltiples criterios**

Se puede agrupar por varios criterios a la vez, por ejemplo, ordenar los resultados por la duración y si hay repetidos, que el siguiente criterio sea la fecha de registro: SELECT \* FROM cdr ORDER BY facturar DESC, registro ASC.

## Listar y limitar resultados.

La principal forma de limitar resultados en una consulta es mediante el uso de la cláusula WHERE para filtrar condiciones en los datos. Sin embargo, cuando aun así los resultados son demasiados, es necesario paginarlos o presentarlos por partes. Para esto, se utiliza la cláusula LIMIT.

* **Filtrar y ordenar llamadas**

Filtra las llamadas de larga distancia o celular del año 2016 que cuesten más de $1.000 y preséntalas ordenadas de mayor a menor por el precio:

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 DES

* **Contar resultados**

Para contar los resultados que cumplen con ciertas condiciones sin ordenar:

SELECT COUNT(\*)

FROM cdr

WHERE YEAR(registro) = 2016

AND LENGTH(destino) = 10

AND CEIL(facturar/60) \* 85 > 200

* **Limitar resultados**

Para mostrar solo un número limitado de resultados, utiliza LIMIT. Por ejemplo, para ver los primeros 10:

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

* **Paginación de resultados**

Para rescatar filas de manera paginada, por ejemplo, las siguientes 10 después de las primeras 10.

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

## Agrupar filas

Es posible agrupar filas en el resultado de una consulta SELECT basándose en los valores de una columna específica, utilizando la cláusula GROUP BY. Esta cláusula se emplea junto con funciones de agregación como AVG, SUM, MAX, COUNT, entre otras.

#### Ejemplo:

Supongamos que deseamos saber cuántas llamadas entrantes se realizaron durante cada uno de los meses del año 2017. Se consideran llamadas entrantes aquellas que no provienen de una extensión local, identificadas porque el origen tiene más de tres dígitos.

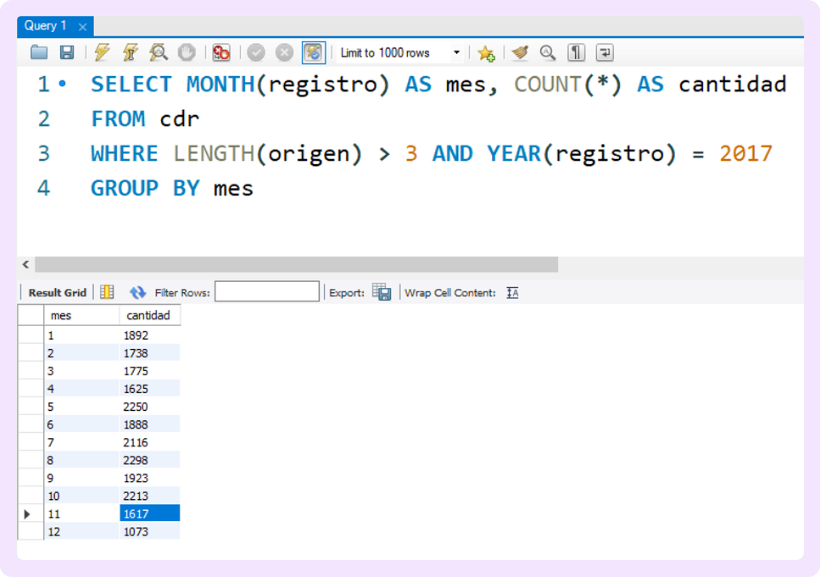
SELECT MONTH(registro) AS mes, COUNT(\*) AS cantidad

FROM cdr

WHERE LENGTH(origen) > 3 AND YEAR(registro) = 2017

GROUP BY mes

1. Ejemplo llamadas entrantes 2017



Pero si se desea mostrar también la agrupación por años y meses para los años 2017, 2018 y 2019, la consulta sería:

SELECT YEAR(registro) AS año, MONTH(registro) AS mes, COUNT(\*) AS cantidad

FROM cdr

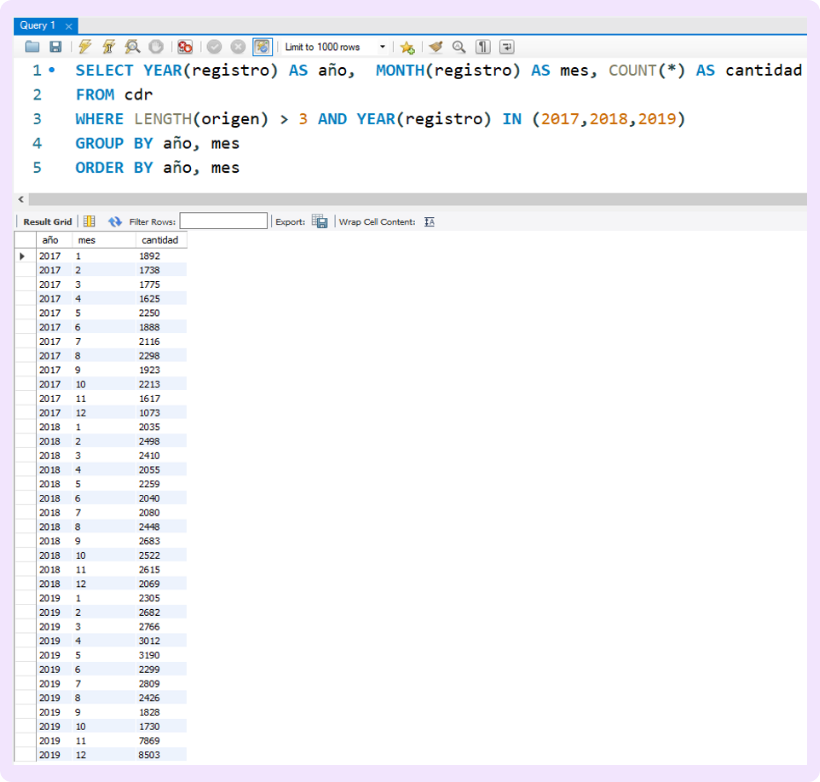
WHERE LENGTH(origen) > 3 AND YEAR(registro) IN (2017,2018,2019)

GROUP BY año, mes

ORDER BY año, mes

Es importante destacar que para incluir varios años en la consulta, se utilizó la cláusula IN. Además, los resultados se ordenan primero por año y, como segundo criterio, por mes.

1. Usó de cláusula IN



# Consulta multitablas – DML

Para explicar este tipo de ejercicios, crearemos una base de datos de empleados llamada ‘ehr’. En esta base de datos, las tablas ‘empleados’ y ‘áreas’ están relacionadas de la siguiente manera: un empleado pertenece a un área y un área puede tener varios empleados, como se presenta en el siguiente diagrama:

Note que area\_id en la tabla empleados puede ser nullable:

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 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

Los datos quedan así:

#### 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]>

#### 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)

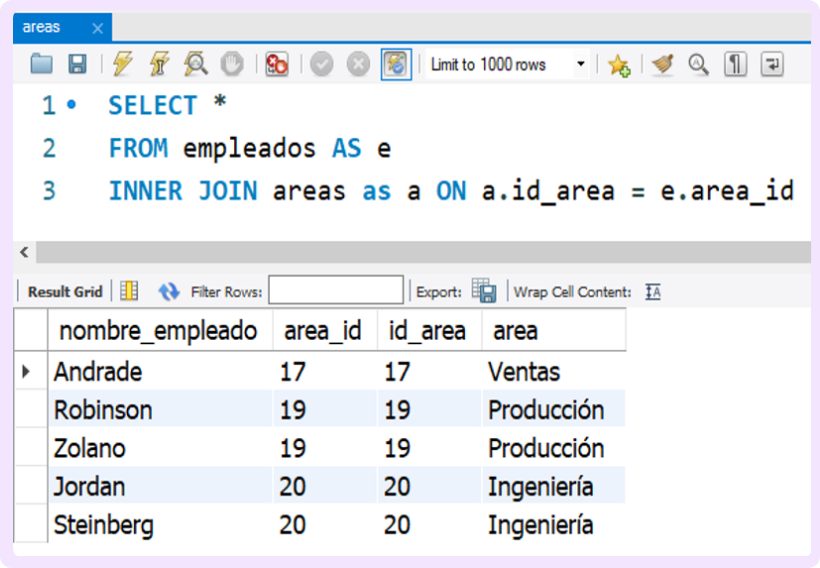
En esta operación, cada fila de la tabla 'empleados' se combina (o une) con las filas correspondientes de la tabla 'areas' que cumplen con la condición especificada en el predicado del JOIN. Cualquier fila en la tabla 'empleados' o en la tabla 'áreas' que no tenga una correspondiente en la otra tabla será excluida; solo aquellas con correspondencias aparecerán en el resultado. Este es el tipo de JOIN más utilizado y es considerado la combinación predeterminada.

SELECT \*

FROM empleados AS e

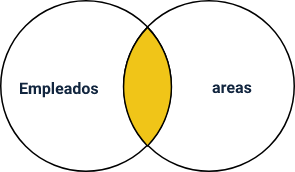
INNER JOIN areas as a ON a.id\_area = e.area\_id

1. Uso de JOIN



La condición de la relación, especificada como 𝑎.𝑖𝑑\_𝑎𝑟𝑒𝑎=𝑒.𝑎𝑟𝑒𝑎\_𝑖𝑑a, requiere que el valor esté presente en ambas tablas. Si un empleado tiene un valor NULL en 'area\_id', no se incluirá en los resultados, ya que no puede cumplir con esta condición. Esto se ilustra mediante un Diagrama de Venn, donde solo la intersección (conjunción) representa los registros que coinciden en ambas tablas.

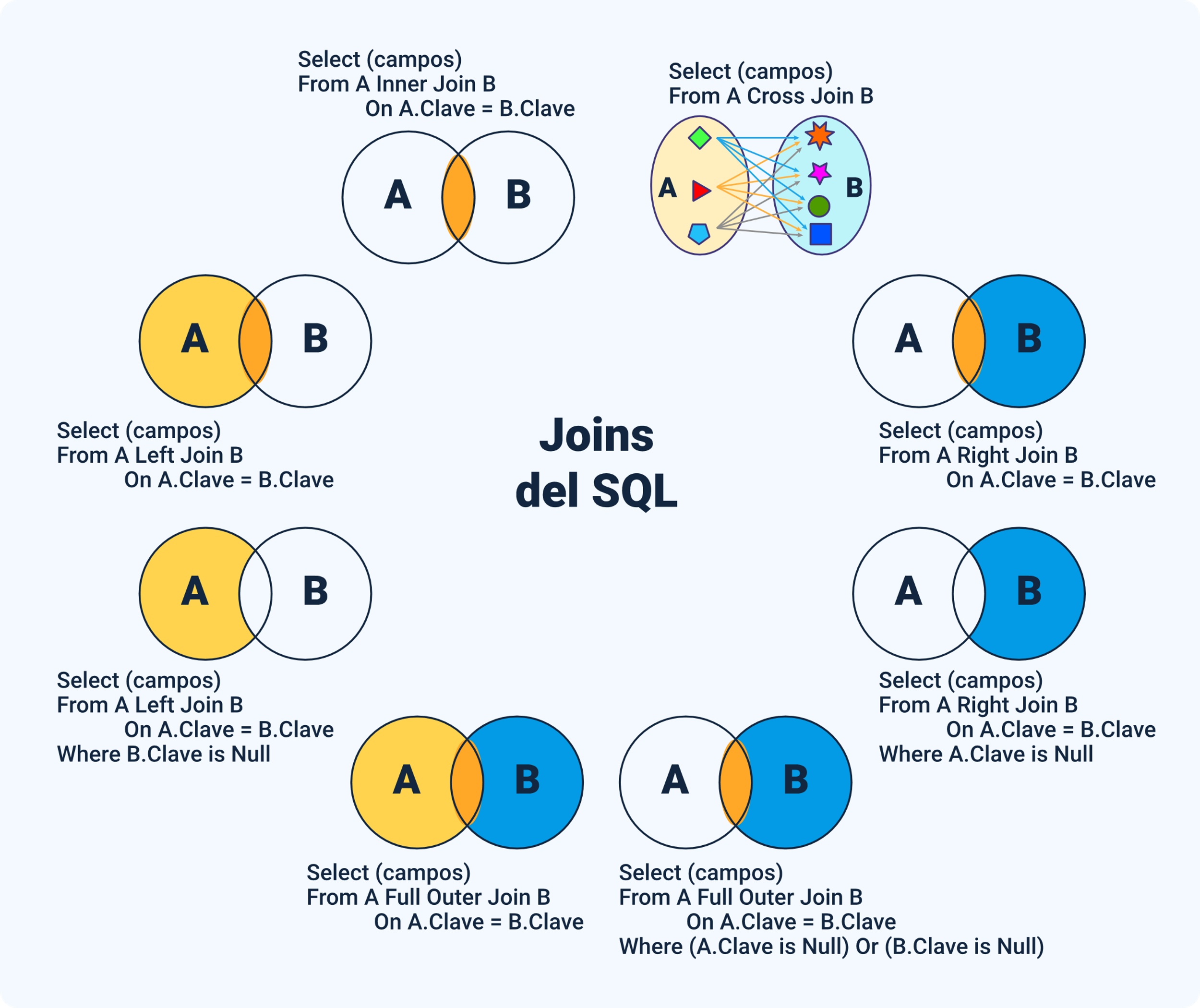
1. Diagramación de Empleados y áreas.



Existen otros tipos de consultas como FULL JOIN, FULL OUTER JOIN y CROSS JOIN que no se tratarán en detalle debido a su uso menos frecuente. Sin embargo, con una comprensión básica del concepto principal, es posible acceder a la documentación en línea para explorar cómo funcionan estos JOINs.

La siguiente imagen incluye Diagramas de Venn que representan diferentes tipos de relaciones. Estos diagramas son herramientas útiles para visualizar los resultados que cada tipo de sentencia JOIN puede retornar, ayudando así a comprender los resultados esperados con cada variante.

1. Representación Diagrama de Venn



La figura ilustra los diferentes tipos de joins en SQL: el Inner Join devuelve los registros con coincidencias en ambas tablas; el Cross Join genera el producto cartesiano de las dos tablas; el Left Join devuelve todos los registros de la tabla izquierda y los coincidentes de la tabla derecha; el Right Join devuelve todos los registros de la tabla derecha y los coincidentes de la tabla izquierda; el Full Outer Join devuelve todos los registros cuando hay coincidencias en una de las tablas; el Left Join con condición devuelve los registros de la tabla izquierda y los de la derecha que no tienen coincidencias; el Right Join con condición devuelve los registros de la tabla derecha y los de la izquierda que no tienen coincidencias; y el Full Outer Join con condición devuelve todos los registros con coincidencias nulas en cualquiera de las tablas.

## Combinación externa

Esta variedad de combinación de tablas permite seleccionar algunas filas de una tabla aunque no exista correspondencia con las filas de la otra tabla con la que se combina. Este método facilita la inclusión de datos que no cumplen con la condición de coincidencia especificada en la cláusula JOIN.

### 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

1. Representación dónde se encuentran los empleados y las áreas.

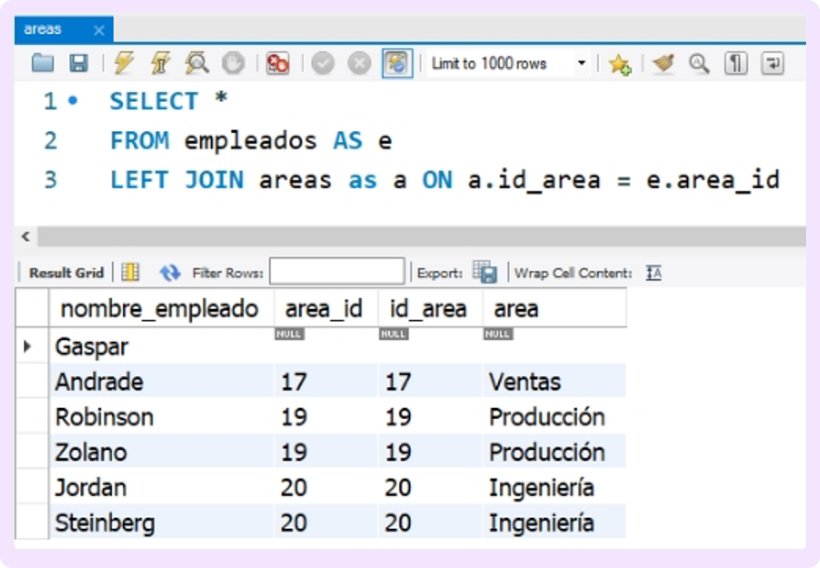


El resultado de este tipo de consulta incluye todos los registros de la tabla de la izquierda (la primera tabla definida en la cláusula FROM de la consulta), independientemente de si existe un registro correspondiente en la tabla de la derecha (la definida en el JOIN).

La sentencia LEFT JOIN empareja todos los valores de la tabla izquierda (empleados) con los valores correspondientes de la tabla derecha (areas), si existen. Si no hay correspondencia, retorna un valor nulo (NULL) en los campos de la tabla derecha.

A diferencia de lo que ocurre con un INNER JOIN, donde no se muestra un empleado cuya área no existe, el LEFT JOIN incluirá estos empleados. En el diagrama asociado, el empleado sin área asignada aparece en la región amarilla, mientras que los empleados con área asignada se muestran en la franja naranja, en la intersección de las tablas de empleados y áreas.

1. Ejemplo LEFT JOIN



### RIGTH JOIN

La sentencia RIGHT JOIN devuelve todos los valores de la tabla derecha (areas), emparejados con los valores correspondientes de la tabla izquierda (empleados), si existen. Si no hay correspondencia, se retorna un valor nulo (NULL) para los campos de la tabla izquierda.

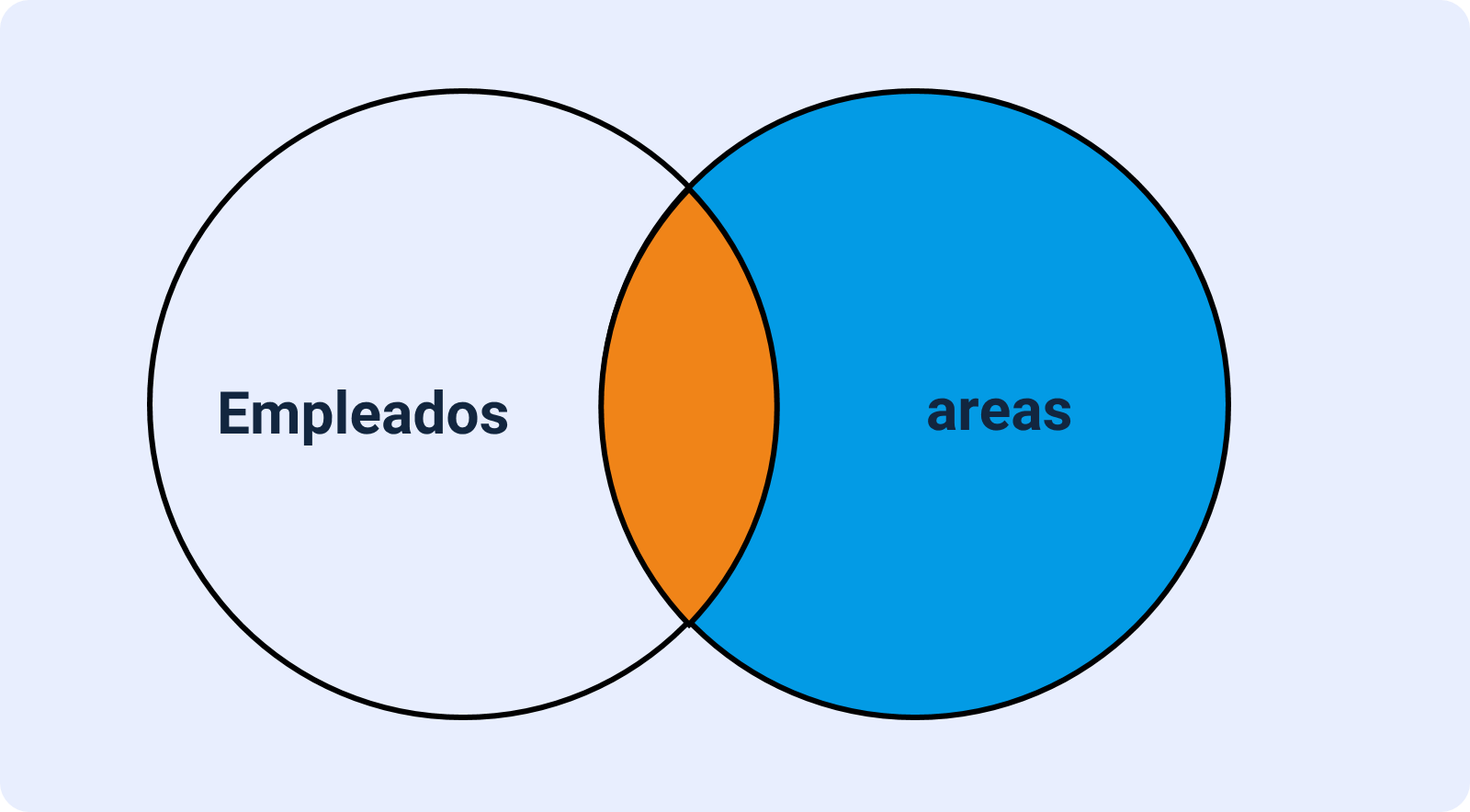
Esta operación garantiza que todos los registros de la tabla derecha estén presentes en el resultado, mientras que los registros de la tabla izquierda que no tengan correspondencia se representarán con valores nulos.

SELECT \*

FROM empleados AS e

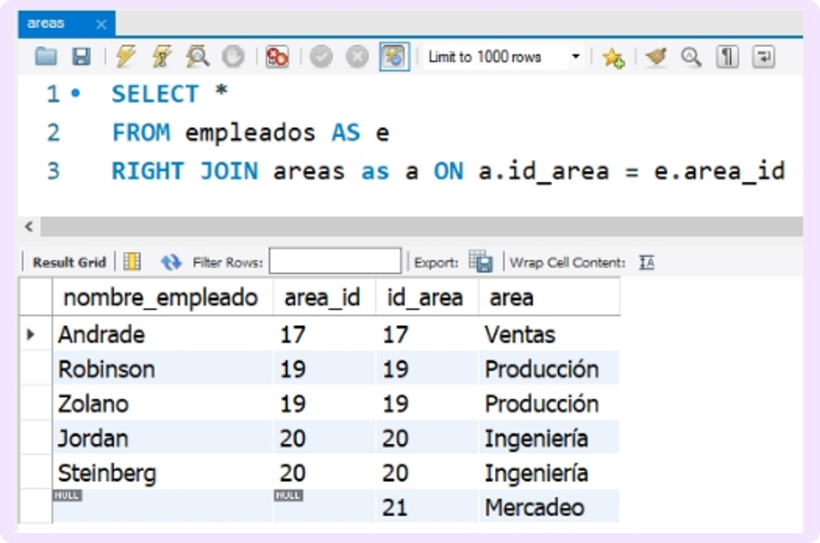
RIGHT JOIN areas as a ON a.id\_area = e.area\_id

1. Diagrama que representa las áreas que tienen y no empleados



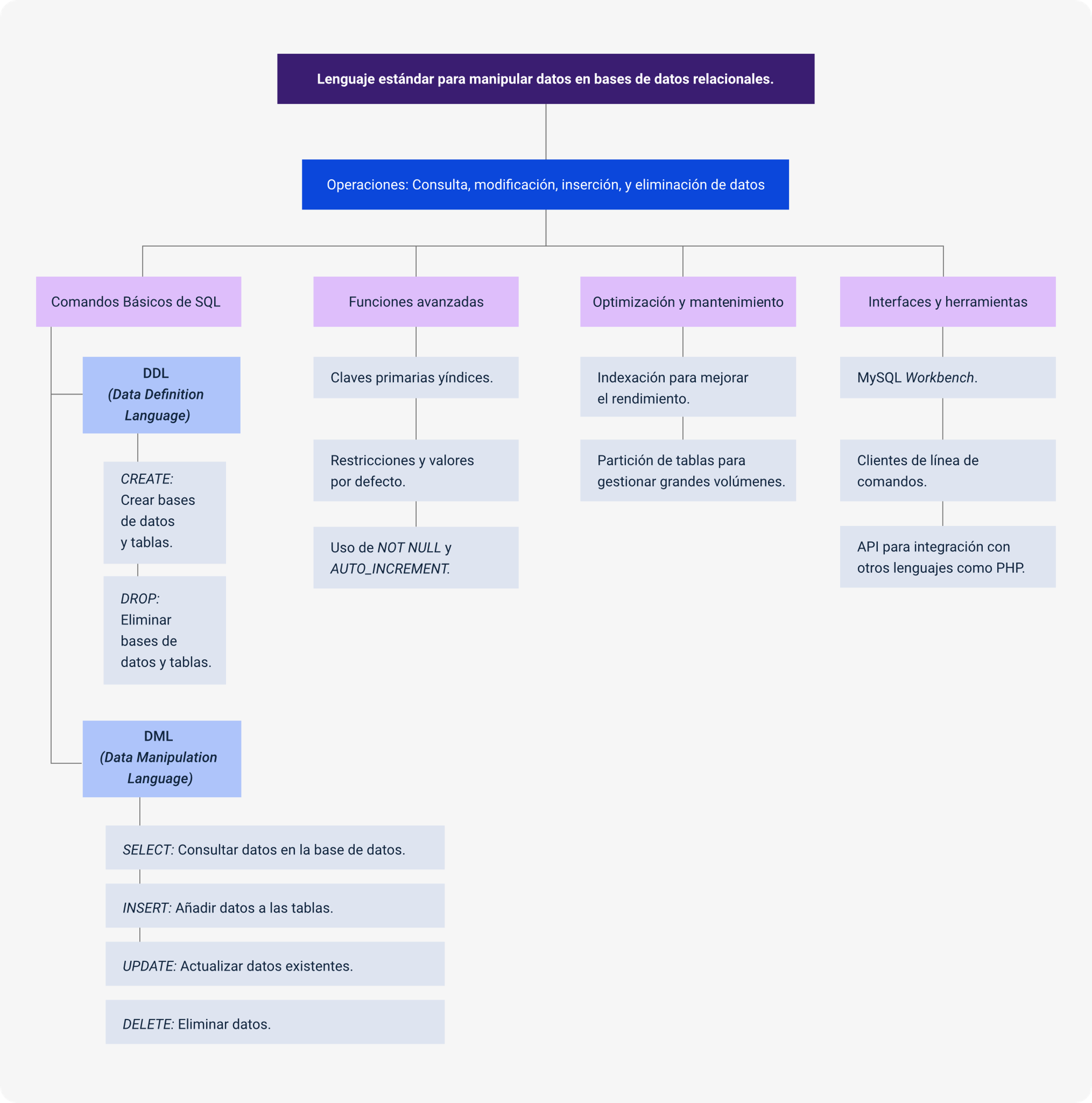
En el diagrama, las áreas que no tienen empleados se encuentran en el área verde, mientras que las áreas con empleados están en el área anaranjada, que representa la intersección de las tablas de empleados y áreas.

1. Ejemplo RIGTH JOIN



Síntesis

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



Material complementario

| Tema | Referencia | Tipo de material | Enlace del recurso |
| --- | --- | --- | --- |
| Funciones en MySQL | Shakmuria. (2023, abril 5). Introducción al SQL: Comandos básicos y ejemplos prácticos [Archivo de video]. YouTube. | Video | <https://www.youtube.com/watch?v=SNjTk5GjbTw&ab_channel=Shakmuria> |
| Consulta multitablas – DML | Códigos de Programación - MR. (2023, abril 10). Aprende SQL en 20 minutos [Archivo de video]. YouTube. | Video | <https://www.youtube.com/watch?v=E9m_ii37pNY&ab_channel=C%C3%B3digosdeProgramaci%C3%B3n-MR> |
| Combinación interna (INNER JOIN) | TodoCode. (2023, marzo 15). Tutorial Completo de SQL para Principiantes [Archivo de video]. YouTube. | Video | <https://www.youtube.com/watch?v=0BstRqp6Svg&ab_channel=TodoCode> |
| MySQL | MySQL 8.0 Reference Manual. (2021). 3.3.2 #[em Creating a Table]. MySQL. | Página web | <https://dev.mysql.com/doc/refman/8.0/en/creating-tables.html> |
| MySQL | MySQLTutorial. (2020). MySQL #[em AND Operator.] MySQL. | Página web | <https://www.mysqltutorial.org/mysql-and> |
| MySQL | MySQLTutorial. (2020). MySQL OR Operator. MySQL. | Página web | <https://www.mysqltutorial.org/mysql-or> |

Glosario

**Clave primaria**: atributo o conjunto de atributos que identifica de manera única cada fila en una tabla.

**CRUD**: acrónimo de Crear, Leer, Actualizar y Eliminar (Create, Read, Update, Delete). Representa las operaciones básicas en una base de datos.

**DDL**: lenguaje de Definición de Datos (Data Definition Language). Subconjunto de comandos en SQL usados para definir la estructura de la base de datos. Incluye comandos como CREATE y DROP.

**DML**: lenguaje de Manipulación de Datos (Data Manipulation Language). Subconjunto de comandos en SQL usados para el manejo de datos, incluyendo SELECT, INSERT, UPDATE y DELETE.

**FOREIGN KEY**: clave foránea, un campo en una tabla que es clave primaria en otra, utilizada para establecer y hacer cumplir un vínculo entre los datos en dos tablas.

**Índice**: estructura de datos que mejora la velocidad de las operaciones de la base de datos.

**JOIN**: operación en SQL que permite combinar filas de dos o más tablas basadas en una relación común entre ellas.

**Normalización**: proceso de organización de campos y tablas de una base de datos para minimizar la redundancia y la dependencia.

**SGBD**: sistema de Gestión de Bases de Datos. Software que permite la creación, modificación y administración de una base de datos.

**SQL**: lenguaje de Consulta Estructurada (Structured Query Language), utilizado para gestionar y manipular bases de datos relacionales.

Referencias bibliográficas

Asale y Rae. (2021). Base. Diccionario de la lengua española. <https://dle.rae.es/base#CiiosqO>

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. <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>

Microsoft. (2021). Descripción de los conceptos básicos de normalización de la base de datos. <https://docs.microsoft.com/es-es/office/troubleshoot/access/database-normalization-description>

MySQL 8.0 Reference Manual. (2021).13.1.20.5 FOREIGN KEY Constraints. 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>

Navarro, J., y Lafuente, M. (2010). Ilustración de moda: dibujo plano. Sistema de Bibliotecas SENA. <https://sena-primo.hosted.exlibrisgroup.com/permalink/f/q6j6k0/sena_aleph000023241>

Créditos

| Nombre | Cargo | Centro de Formación y Regional |
| --- | --- | --- |
| 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 |
| Henry Eduardo Bastidas Paruma | Experto Temático | Centro de Teleinformática y Producción Industrial - Regional Cauca |
| Peter Emerson Pinchao | Experto Temático | Centro de Teleinformática y Producción Industrial - Regional Cauca |
| Paola Alexandra Moya Peralta | Evaluadora Instruccional | Centro de Servicios de Salud - Regional Antioquia |
| Carlos Julián Ramírez Benítez | Diseñador de Contenidos Digitales | Centro de Servicios de Salud - Regional Antioquia |
| Edgar Mauricio Cortés García | Desarrollador Fullstack | Centro de Servicios de Salud - Regional Antioquia |
| Edgar Mauricio Cortés García | Actividad Didáctica | Centro de Servicios de Salud - Regional Antioquia |
| Luis Gabriel Urueta Álvarez | 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 |