**FORMATO PARA EL DESARROLLO DE COMPONENTE FORMATIVO**

|  |  |
| --- | --- |
| PROGRAMA DE FORMACIÓN | ADSO |

|  |  |  |  |
| --- | --- | --- | --- |
| COMPETENCIA | 220501096. Desarrollar la solución de software de acuerdo con el diseño y metodologías de desarrollo. | RESULTADOS DE APRENDIZAJE | 220501096-02. Construir la base de datos para el software a partir del modelo de datos. |

|  |  |
| --- | --- |
| NÚMERO DEL COMPONENTE FORMATIVO | 14 |
| 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 | 2 - CIENCIAS NATURALES, APLICADAS Y RELACIONADAS |
| IDIOMA | Español |

1. **TABLA DE CONTENIDOS:**

1. Interfaz de línea de comandos de MySQL

1.1. Aplicación cliente de MySQL

1.2. Prueba de consultas básicas

1.3. SQL – Standard Query Language

2. SQL para la creación y eliminación de bases de datos – DDL

2.1. Claves foráneas

2.2. Mostrar y eliminar tablas y bases de datos

3. SQL para la creación y eliminación de bases de datos – DDL

3.1 Inserción de registros

3.2 Edición de registros

3.3 Borrado de registros

4. SQL para consulta de datos – DML

4.1 Funciones en MySQL

4.2. Operadores

4.3. Ordenar resultados

4.4. Listar y limitar resultados

4.5. Agrupar filas

5. Consulta multitablas – DML

5.1. Combinación interna (INNER JOIN)

5.2. Combinación externa.

1. **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 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.

1. **DESARROLLO DE CONTENIDOS:**
2. **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.

**1.1. Aplicación cliente de MySQL**

Se deben seguir las siguientes instrucciones para conectarse a la base de datos a través del cliente de línea de comandos

|  |
| --- |
| PASOS  CF014\_1.1. Aplicación cliente de MySQL |

* 1. **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 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 esta situación se ha consultado el resultado de la función **VERSION**, se sabe que es una función porque tiene paréntesis” ()”, y de la variable **CURRENT\_DATE.**

|  |  |
| --- | --- |
| Icono realista de campana 3d aislado sobre fondo blanco Representación vectorial de campana con nuevo mensaje de alerta de chat de notificación push | **¡Nota!**  Toda sentencia debe finalizar con punto y coma (;), luego debe dar enter. |

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

En nivel teórico, hay dos lenguajes para el uso de 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.** Ya no se empleará (por ahora), **MySQL Workbench**, y se CONOC

1. **SQL para la creación y eliminación de bases de datos – DDL**

**A person sitting at a desk working on a computer

Description automatically generated**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.

|  |  |
| --- | --- |
| Mano con altavoz Brazo de dibujos animados sosteniendo megáfono Fondo colorido con espacio de copia Dispositivo electrónico para aumentar el volumen de voz Mensaje de audio fuerte vector banner promocional | ¡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":**

|  |
| --- |
| PASOS  CF014\_2\_SQL para la creación y eliminación de bases de datos – DDL |

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> |

**A. 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.

|  |  |
| --- | --- |
| A black line drawing of a rectangle and a rectangle with arrows  Description automatically generated | 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> |

**B. 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:

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 5000);  Query OK, 0 rows affected (0.29 sec) |

**C. 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) |

|  |  |
| --- | --- |
| Icono realista de campana 3d aislado sobre fondo blanco Representación vectorial de campana con nuevo mensaje de alerta de chat de notificación push | **¡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.

|  |  |
| --- | --- |
| A hand holding a megaphone  Description automatically generated | 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. |

**D. 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)  mysql> |

**E. 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) |

**F. 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) |

**G. 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:

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**

**H. 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> |

**I. 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?**

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) |

**J. 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.**

La diferencia entre un índice único y un índice normal es que en los índices únicos no se admite la inclusión de filas con datos repetidos en la columna o columnas de la tabla. Una excepción es el valor **NULL,** que sí puede repetirse varias veces.

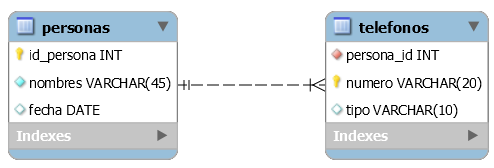
|  |
| --- |
| mysql> CREATE TABLE indices3 (  -> id INT,  -> nombre CHAR(19),  -> UNIQUE (nombre));  Query OK, 0 rows affected (0.09 sec) |

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.

**2.1. 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.

**Figura 1.** Claves foráneas



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.

|  |
| --- |
| La expresión **DELETE CASCADE** establece que, si se elimina una fila de la tabla 'persona' que tiene registros telefónicos relacionados a través de la columna **'persona\_id**' en la tabla **'telefonos**', estos registros también se eliminarán automáticamente. Este mecanismo es un ejemplo de restricción de 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'.**

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

|  |
| --- |
| Infografía interactiva  CF014\_2.2\_Claves foráneas |

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.**

Datos de la tabla **personas3.**

|  |
| --- |
| +------------+-----------+------------+  | 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 **telefonos3**.

|  |
| --- |
| +--------+-----------+  | numero | persona\_id|  +--------+-----------+  | 234279534 | 1 |  | 654345534 | 1 |  | 578819343 | 3 |  | 234294276 | 4 |  | 234293465 | 4 |  | 156096465 | 3 |  +--------+-----------+  7 rows in set (0.000 sec)  MariaDB [prueba]> |

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.

**2.2. 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> |

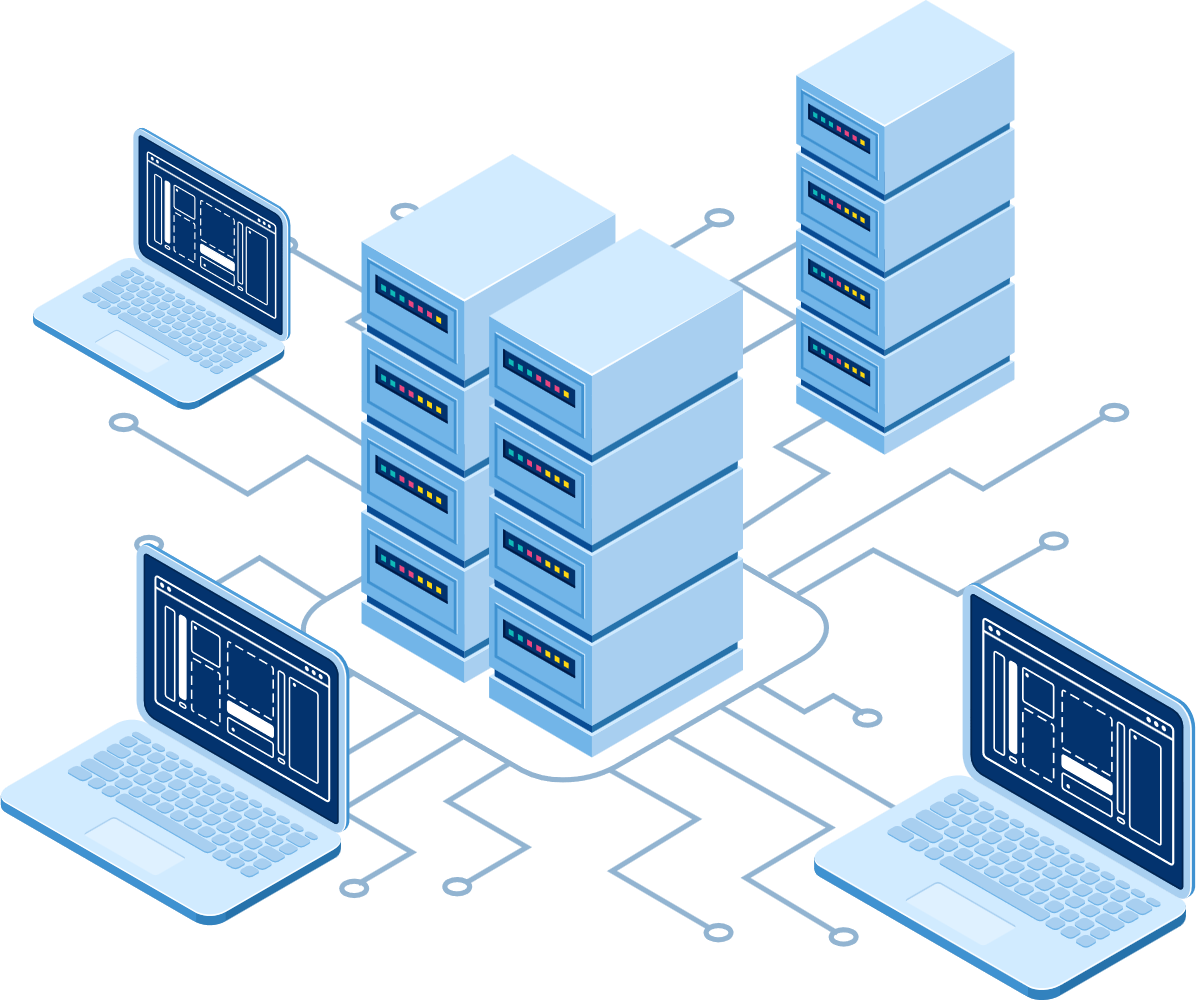
De manera similar, se puede borrar bases de datos enteras, utilizando la sentencia **DROP DATABASE.** La sintaxis asimismo es muy sencilla, a continuación, se creará una base de datos, una tabla y se borrará la base de datos.

|  |
| --- |
| 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> |

|  |  |
| --- | --- |
| **A person standing next to a magnifying glass and a book  Description automatically generated** | **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. |

1. **SQL para la creación y eliminación de bases de datos – DDL**

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]> |

**3.1 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.



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

**3.2 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]  [ORDER BY ...]  [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 lo 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]> |

**3.3 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]> |

1. A computer screen with a yellow and blue screen

   Description automatically generated**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.

**4.1 Funciones en MySQL**

A computer screen with text and blue squares

Description automatically generatedConsiderando 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:

**Tabla 1.** Funciones en 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, 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:**

A blue and yellow browser window

Description automatically generated

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

|  |  |
| --- | --- |
| A computer screen with a black background  Description automatically generated | **BASE DE DATOS**  Descargue aquí el archivo para crear la base de datos cdr. |

Conozca los datos de prueba con los que se trabajará.

**Tabla 2.** Diccionario de datos

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

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

**Figura 2.** MySQL WorkBench

|  |  |
| --- | --- |
| A screenshot of a computer  Description automatically generated | 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. 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:

|  |
| --- |
| PASOS  CF014\_4.1 Funciones en MySQL |

* 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]>**

|  |
| --- |
| Slides  CF014\_ 4.2\_Operadores |

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**

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

**Tabla 3.** 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'**

**Figura 3.** Operadores de desigualdad

A screenshot of a computer

Description automatically generated

* 1. **Ordenar resultados**

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

|  |
| --- |
| Acordeón  CF014\_4.3\_Ordenar resultados |

* 1. **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**.

|  |
| --- |
| Slides  CF014\_4.4\_ Listar y limitar resultados. |

* 1. **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**

**Figura 4.** Ejemplo llamadas entrantes 2017

A screenshot of a computer

Description automatically generated

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.

**Figura 5.** Usó de cláusula IN

A screenshot of a computer

Description automatically generated

1. **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 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#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**

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*).**

En esta operación, cada fila de la tabla **'empleados'** se combina (o une) con las filas correspondientes de la tabla **'áreas'** 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**

**Figura 6.** Uso de JOIN

A screenshot of a computer

Description automatically generated

|  |  |
| --- | --- |
| La condición de la relación, especificada como **𝑎.𝑖𝑑\_𝑎𝑟𝑒𝑎=𝑒.𝑎𝑟𝑒𝑎\_𝑖𝑑*a,*** 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. | A diagram of a venn diagram  Description automatically generated |

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.

**Figura 7.** Representación Diagrama de Venn

A diagram of a group of circles with text

Description automatically generated

* 1. **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**

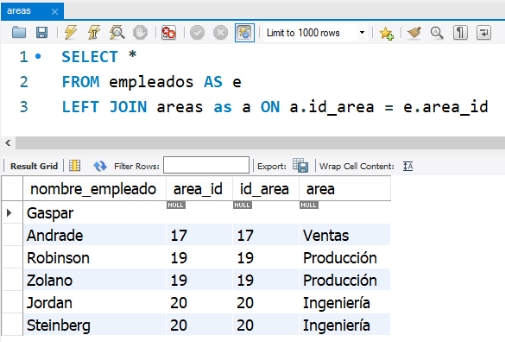
A diagram of a diagram

Description automatically generatedEl 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 (áreas), 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.

**Figura 8.** Ejemplo LEFT JOIN



**RIGTH JOIN**

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

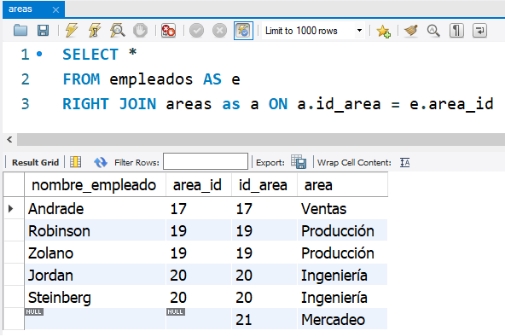
**SELECT \***

**FROM empleados AS e**

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

|  |  |
| --- | --- |
| 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 | A diagram of a diagram  Description automatically generated |

**Figura 9.** Ejemplo RIGTH JOIN



1. **SÍNTESIS**

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

A chart with green and blue squares

Description automatically generated

1. **ACTIVIDADES DIDÁCTICAS (Se debe incorporar mínimo 1, máximo 2)**

|  |  |
| --- | --- |
| DESCRIPCIÓN DE ACTIVIDAD DIDÁCTICA | |
| Nombre de la Actividad | Fundamentos de SQL |
| Objetivo de la actividad | Comprender los comandos fundamentales de SQL para la gestión eficaz de bases de datos |
| Tipo de actividad sugerida | Relacionar conceptos |
| Archivo de la actividad  (Anexo donde se describe la actividad propuesta) | *CF014\_Actividad didactica* |

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 |
| 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 Creating a Table*. MySQL*.* | Página web | <https://dev.mysql.com/doc/refman/8.0/en/creating-tables.html> |
| MySQL | MySQLTutorial. (2020). *MySQL 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> |

1. **GLOSARIO:**

|  |  |
| --- | --- |
| TÉRMINO | SIGNIFICADO |
| SQL: | Lenguaje de Consulta Estructurada (Structured Query Language), utilizado para gestionar y manipular bases de datos relacionales. |
| SGBD: | Sistema de Gestión de Bases de Datos. Software que permite la creación, modificación y administración de 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. |
| Clave primaria: | atributo o conjunto de atributos que identifica de manera única cada fila en una tabla. |
| 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. |
| 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. |
| Normalización: | proceso de organización de campos y tablas de una base de datos para minimizar la redundancia y la dependencia. |
| CRUD: | acrónimo de Crear, Leer, Actualizar y Eliminar (Create, Read, Update, Delete). Representa las operaciones básicas en una base de datos. |

1. **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/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>

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>

1. **CONTROL DEL DOCUMENTO**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Nombre | Cargo | Dependencia  *(Para el SENA indicar Regional y Centro de Formación)* | Fecha |
| Autor (es) | Henry Eduardo Bastidas Paruma | Experto Temático | Regional Cauca - Centro de Teleinformática y Producción Industrial | Noviembre 2022 |
| Peter Emerson Pinchao Solis | Experto Temático | Regional Cauca - Centro de Teleinformática y Producción Industrial | Noviembre 2022 |
| Paola Alexandra Moya | Evaluadora instruccional | Regional Antioquia - Centro de Servicios de Salud | Mayo 2024 |
|  | Olga Constanza Bermúdez Jaimes | Responsable Línea de Producción Antioquia | Regional Antioquia - Centro de Servicios de Salud | Mayo 2024 |

1. **CONTROL DE CAMBIOS**

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Nombre | Cargo | Dependencia | Fecha | Razón del Cambio |
| Autor (es) |  |  |  |  |  |