UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES
Centro de e-Learning
Secretaría de Cultura y Extensión Universitaria

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Curso:

PROGRAMADOR WEB INICIAL

UTN.BA | Centro de e-Learning
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES
Secretaría de Cultura y Extensión Universitaria

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Módulo 2:

PROGRAMACIÓN WEB (PARTE 1)

Unidad 4:

INTRODUCCIÓN A MYSQL

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Presentación

En esta unidad aprenderemos a utilizar el sistema de gestión de base de datos MYSQL. Implementaremos sentencias de código para definir las Bases de Datos, tablas, vistas e índices (DDL).

También veremos cómo manipular datos empleando sentencias DML (seleccionar, insertar, eliminar y actualizar).

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Objetivos

Que los participantes logren...

- Identificar e implementar los componentes sintácticos de MYSQL.
- Crear, modificar y borrar tablas y dominios con SQL.
- Manipular datos; seleccionar, insertar, eliminar y actualizar datos empleando sentencias DML.

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Bloques temáticos

1. Tipos de campos empleados en las bases de datos
2. Un vistazo al SQL
    a. Tipos de campos en MySQL
    b. Componentes sintácticos
3. DDL (Data Definition Language)
    a. Crear una base de datos
    b. Crear una tabla
        i. Valores nulos
        ii. Valores por defecto
        iii. Claves primarias
        iv. Campos autoincrementales
        v. Comentarios
    c. Índice
        i. Claves primarias: PRIMARY KEY
        ii. Índices: KEY o INDEX
        iii. Claves únicas: UNIQUE
        iv. Claves foráneas
    d. DROP
    e. ALTER
4. DML (Data Manipulation Language)
    a. Selección de datos
    b. Cláusula HAVING

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Tipos de campos empleados en las bases de datos

Como hemos visto anteriormente, una base de datos está compuesta de tablas donde almacenamos registros con información de cada uno.
Debemos conocer los distintos tipos de datos que colocaremos en nuestras columnas, que pueden ser de diferentes tipos. Es importante detallar el tipo de valor que insertaremos, así facilitaremos la búsqueda posterior y optimizaremos los recursos de memoria.

No todas los sistemas de gestión de bases de datos (SGBD) utilizan los mismos tipos de datos en sus campos: hay algunos que son particulares en cada una. Igualmente, existe un conjunto de tipos de datos que están representados en la totalidad de estas bases.

Estos tipos comunes son los siguientes:

Alfanuméricos	Contienen cifras y letras. Presentan una longitud limitada (255 caracteres)
Numéricos	Existen de varios tipos, principalmente, enteros (sin decimales) y reales (con decimales).
Booleanos	Poseen dos formas: Verdadero y falso (Sí o No)
Fechas	Almacenan fechas facilitando posteriormente su explotación. Almacenar fechas de esta forma posibilita ordenar los registros por fechas o calcular los días entre una fecha y otra...
Memos	Son campos alfanuméricos de longitud ilimitada. Presentan el inconveniente de no poder ser indexados (veremos más adelante lo que esto quiere decir).
Autoincrementables	Son campos numéricos enteros que incrementan en una unidad su valor para cada registro incorporado. Su utilidad resulta más que evidente: Servir de identificador ya que resultan exclusivos de un registro.

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

La siguiente tabla recoge los sinónimos de los tipos de datos definidos:

Tipo de Dato	Sinónimos
BINARY	VARBINARY
BIT	BOOLEAN LOGICAL LOGICAL1 YESNO
BYTE	INTEGER1
COUNTER	AUTOINCREMENT
CURRENCY	MONEY
DATETIME	DATE TIME TIMESTAMP
SINGLE	FLOAT4 IEEESINGLE REAL
DOUBLE	FLOAT FLOAT8 IEEEDOUBLE NUMBER
SHORT	INTEGER2 SMALLINT
LONG	INT INTEGER INTEGER4
LONGBINARY	GENERAL OLEOBJECT
LONGTEXT	LONGCHAR MEMO NOTE
TEXT	ALPHANUMERIC CHAR - CHARACTER STRING - VARCHAR
VARIANT (No Admitido)	VALUE

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Un vistazo al SQL

Las aplicaciones en red son cada día más numerosas y versátiles. En muchos casos, el esquema básico de operación es una serie de scripts que rigen el comportamiento de una base de datos.
Debido a la diversidad de lenguajes y de bases de datos existentes, la manera de comunicar entre unos y otras sería realmente complicada de gestionar, pero contamos con estándares que nos permiten realizar las operaciones básicas de una forma universal. El **Structured Query Language** es un lenguaje estándar de comunicación con bases de datos, un lenguaje normalizado que nos permite trabajar con cualquier tipo de lenguaje (ASP o PHP) en combinación con cualquier tipo de base de datos (MS Access, SQL Server, MySQL.).
El hecho de que sea estándar no quiere decir que sea idéntico para cada base de datos. Determinadas bases de datos implementan funciones específicas que no tienen necesariamente que funcionar en otras.
Aparte de esta universalidad, el SQL posee otras dos características muy apreciadas. Por una parte, presenta una potencia y versatilidad notables, y su aprendizaje es accesible.

MySQL es un sistema de gestión de bases de datos.

Una base de datos define una estructura en la cual se guardará información. Los datos se guardan en tablas, en las cuales cada columna define el tipo de información que se guardará en ella y cada fila es un registro de la tabla. Cada base de datos tiene un nombre identificador, sus tablas tienen un nombre que las distingue y cada columna de una tabla tiene un tipo de dato asociado (por ejemplo, VARCHAR para cadenas de caracteres, INT para números enteros, etc).

El proceso de datos de las bases de datos se hace a través de consultas.

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Veamos un ejemplo: supongamos que en una Base de Datos (de ahora en más BD) tenemos un listado de todos los teléfonos de las personas que viven en Capital Federal, y le hacemos las siguientes preguntas:

> **¿Cuáles son los teléfonos de todas las personas de apellido Gómez?**
>
> Como respuesta, nos va a dar los teléfonos de todos los Gómez de Capital Federal que tenga ingresados.
>
> **¿Cuál es el nombre de la persona con el número de teléfono 0223-831-9268?**
>
> En este caso, como respuesta obtendremos "vacío" ya que la característica indicada es de Mar del Plata, y esta BD sólo tiene disponible información sobre Capital Federal, por lo tanto no puede encontrar dato alguno que responda a nuestra consulta.

Estas preguntas a la BD se realizan mediante un lenguaje llamado SQL (Structured Query Language – Lenguaje Estructurado de Consultas) y la BD nos va a responder con datos o "vacío" si es que no encontró ningún dato que respondiera a nuestra pregunta.

Tipos de campos en MySQL

Retomemos y profundicemos un tema muy importante al momento de crear nuestras bases de datos: la selección de los tipos de campos que albergarán los datos en nuestras tablas. MySQL soporta un número de tipos de campos (columnas) divididos en varias categorías. Una vez que hemos decidido qué información debemos almacenar, el siguiente paso consiste en elegir el tipo adecuado para cada atributo.

Los tipos de datos más usados son:

- **Varchar:** recibe cadenas de palabras compuestas de letras, números y caracteres especiales.
- **Int:** números enteros con o sin signo.

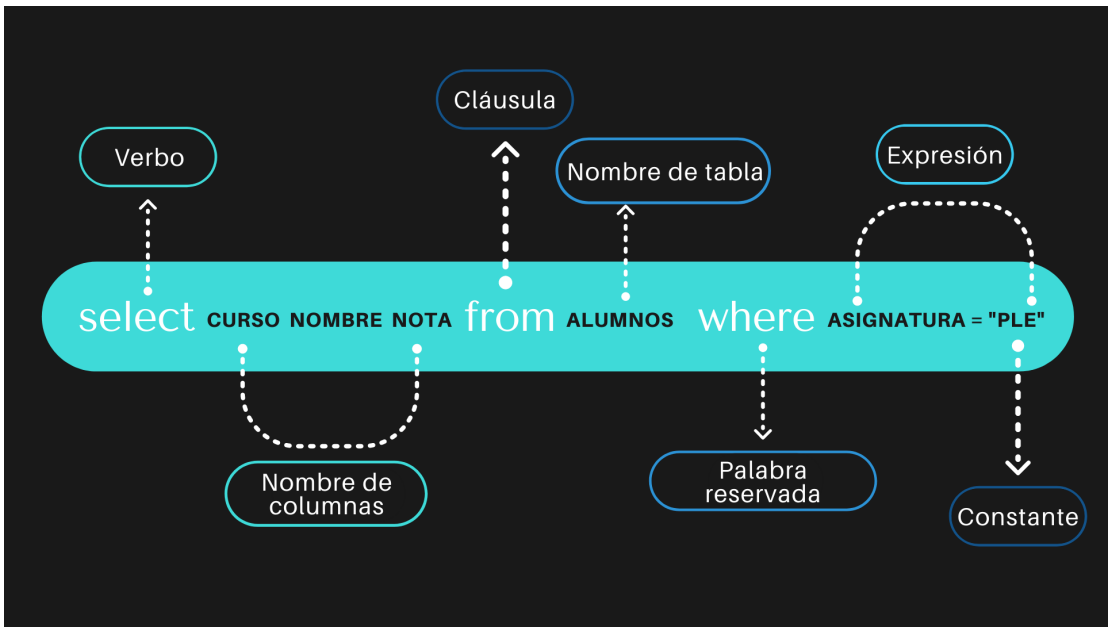
Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

- **Date:** una fecha de calendario que contiene el año (de cuatro cifras), el mes y el día. En formato AAAA/MM/DD.
- **Time:** La hora del día expresada en horas, minutos y segundos. El valor predeterminado es cero.

Existen más tipos de datos. Podrán conocerlos aquí:
https://dev.mysql.com/doc/refman/5.7/en/data-types.html

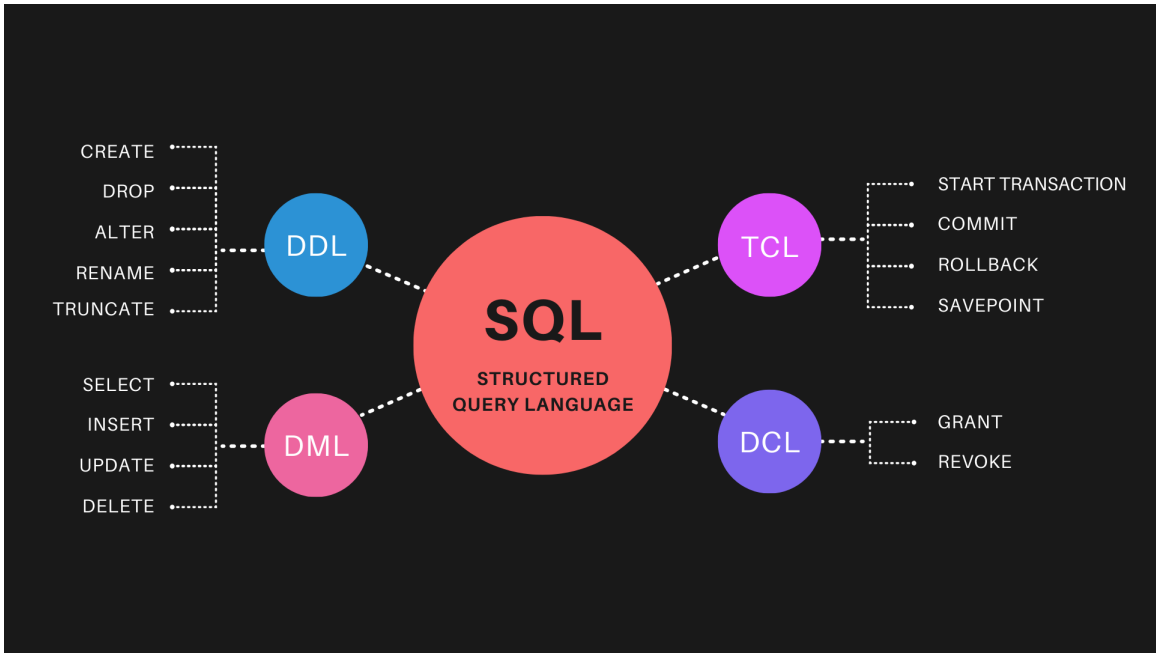
Componentes sintácticos

Las sentencias en SQL la mayoría de las veces tienen la misma estructura. Comienzan por un verbo que puede ser: select, insert, update, create. Luego, le sigue una o más cláusulas que anuncian los datos que van a ser intervenidos (from, where). Estas cláusulas pueden ser opcionales u obligatorias, según la acción a realizar (por ejemplo, from que es obligatoria en este caso).



Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Los mandatos, instrucciones o comandos de SQL se dividen en cuatro categorías:



Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

DDL (Data Definition Language)

El **DDL (Data Definition Language)** se encarga de definir las Bases de Datos como así también los elementos de nuestra BD: tablas, vistas e índices. Admite las siguientes sentencias de definición:

- **CREATE:** se utiliza esta sentencia para crear bases de datos, tablas, dominios, aserciones y vistas.
- **ALTER:** esta sentencia nos permite modificar tablas y dominios.
- **DROP:** se emplea para borrar bases de datos, tablas, dominios, aserciones y vistas.

Cada una de estas sentencias se puede aplicar a las tablas, vistas, procedimientos almacenados y triggers de la base de datos.

De este modo podemos:

- Añadir una nueva base de datos.
- Suprimir una base de datos.
- Añadir una nueva tabla a la base de datos.
- Suprimir una tabla de la base de datos.
- Modificar la estructura de una tabla existente.
- Añadir una nueva vista a la base de datos.
- Suprimir una vista de la base de datos.
- Construir un índice para una columna.
- Suprimir el índice para una columna.
- Definir un alias para un nombre de tabla.
- Suprime un alias para un nombre de tabla.

Lo primero que haremos para poder trabajar con bases de datos relacionales es definirlas.

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

1- Crear una base de datos

Desde el punto de vista de SQL, una base de datos es sólo un conjunto de relaciones (o tablas). Para organizarlas o visualizarlas se accede a ellas mediante su nombre. A nivel de sistema operativo, cada base de datos se guarda en un directorio diferente.

Cuando queramos crear una base de datos, no tendremos ninguna tabla, estará vacía nuestra BD, pero es una tarea muy simple agregar una tabla.. Empezaremos creando nuestra propia base de datos y manipulándola, mientras nos adentramos en el funcionamiento de MySQL

CREATE

Cuando utilicemos este comando se creará un objeto dentro de la base de datos. Dependiendo de lo que queramos crear, se colocará una palabra clave, Esto puede ser la propia base de datos, una tabla, vista, índice, trigger, función, procedimiento o cualquier otro objeto que el motor de la base de datos soporte. Empezaremos creando una base de datos utilizando la sentencia **CREATE DATABASE.**

DATABASE:

```
CREATE DATABASE nombre_de_la_base;
```

Se puede completar la orden con la cláusula:

```
CREATE DATABASE IF NOT EXISTS nombre_de_la_base;
```

Si no usamos IF NOT EXISTS y ya existe una base con ese nombre, MySQL nos avisará del error y no ejecutará acción alguna. Ya que la base de datos no podrá crearse si previamente existe una con el mismo nombre

Una vez que la base está creada, podemos seleccionarla como BD por defecto. De esta manera no tendremos que colocar el nombre de nuestra base de datos cada vez que necesitamos trabajar con ella. Para seleccionar una base de datos se usa el

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

comando USE, que no es exactamente una sentencia SQL, sino más bien una opción de MySQL.

```
USE nombre_de_la_base;
```

2- Crear una tabla

Ahora vamos a crear una tabla utilizando la sentencia: **CREATE TABLE**. Al momento de utilizarla poseemos diferentes características para crear nuestra tabla. La sentencia **CREATE TABLE** creará una tabla con las columnas que nosotros le indiquemos. Por ejemplo, crearemos una tabla que almacena nombres de personas y la fecha de nacimiento de cada uno. Colocaremos el nombre de la tabla y los nombres y tipos de las columnas.

```
CREATE TABLE nombre_de_tabla (nombre_del_campo tipo de campo, ...);
```

Por ejemplo:

```
CREATE TABLE personas (nombre VARCHAR(50), fecha DATE);
```

Utilizando esta sentencia hemos creado una nueva tabla llamada "personas" con dos columnas: una de ellas es "nombre" que puede contener cadenas de hasta 50 caracteres y la otra columna es "fecha" de tipo fecha.

Podemos realizar consultas, mostrando cuántas tablas hay y qué nombres tienen en una base de datos, usando la sentencia SHOW TABLES:

```
SHOW TABLES;
```

Como indicamos anteriormente, a la hora de definir los campos de nuestra tabla tendremos muchas más opciones. Además del tipo y el nombre, podemos definir valores por defecto, permitir o no que contengan valores nulos, crear una clave primaria, indexar...

La sintaxis para definir campos es:

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

```
nombre_col tipo [NOT NULL | NULL] [DEFAULT valor_por_defecto] [AUTO_INCREMENT]
[[PRIMARY] KEY] [COMMENT 'string'][definición_referencia]
```

Veamos cada una de las opciones por separado detallando cada una.

a- Valores nulos

El concepto del valor NULL muchas veces confunde a los recién llegados a SQL, ya que frecuentemente creen que NULL es lo mismo que una cadena de caracteres vacía ' ' o el valor cero, pero no lo es.

Los valores NULL indican que los datos son desconocidos, no aplicables o que se agregarán más adelante. Por ejemplo, la inicial de un cliente puede que no sea conocida en el momento en que éste hace un pedido. No hay dos valores NULL iguales. Es por esto que la comparación entre dos valores NULL, o entre un valor NULL y cualquier otro valor, tiene un resultado desconocido.

A la hora de definir una nueva columna podremos decidir si podrá contener valores nulos. Debemos tener en cuenta que, las columnas que son o forman parte de una clave primaria no pueden tener valores nulos. Al definir una columna como clave primaria, automáticamente se impide que pueda contener valores nulos.

La opción por defecto permite valores nulos, NULL. Para restringirlo se deberá usar NOT NULL para que no se permitan dichos valores nulos.

Por ejemplo:

```
CREATE TABLE provincia1 (nombre CHAR(20) NOT NULL, poblacion INT NULL);
```

b- Valores por defecto

Cada columna puede poseer un valor por defecto que se asignará de forma automática cuando no se especifique un valor determinado al añadir filas en la tabla. Se utilizará DEFAULT.

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Si una columna puede tener un valor nulo, y no se especifica un valor por defecto, se usará NULL como valor por defecto. En el ejemplo anterior, el valor por defecto para población es NULL

Por ejemplo, si queremos que el valor por defecto para población sea 5000, podemos crear la tabla como:

```
CREATE TABLE provincia2 (nombre CHAR(20) NOT NULL, poblacion INT NULL DEFAULT
5000);
```

c- Claves primarias

También se puede definir una clave primaria sobre una columna, usando la palabra clave KEY o PRIMARY KEY. Sólo puede existir una clave primaria en cada tabla, y la columna sobre la que se define una clave primaria no puede tener valores NULL. Si esto no se especifica de forma explícita, MySQL lo hará de forma automática.

Por ejemplo, si queremos crear un índice en la columna nombre de la tabla de provincias, crearemos la tabla así:

```
CREATE TABLE provincia3 (nombre CHAR(20) NOT NULL PRIMARY KEY, poblacion INT
NULL DEFAULT 5000);
```

Usar NOT NULL PRIMARY KEY equivale a PRIMARY KEY, NOT NULL KEY o sencillamente KEY.

d- Campos autoincrementales

Podremos también crear un campo que se incremente automáticamente usando AUTO_INCREMENT. Esta columna sólo podrá contener datos de tipo enteros. Este campo hará que si a la hora de agregar una fila se omite el valor de la columna autoincrementada o si se inserta un valor nulo, su valor se colocará automáticamente, tomando el valor más alto que hay y sumándole una unidad. Esto

permite crear, de una forma sencilla, una columna con un valor único para cada fila de la tabla.

Las columnas con campos autoincrementales se suelen utilizar como claves primarias 'artificiales'.

```
CREATE TABLE provincia4 (clave INT AUTO_INCREMENT PRIMARY KEY, nombre CHAR(20)
NOT NULL, poblacion INT NULL DEFAULT 5000);
```

e- Comentarios

Cuando creamos la tabla, podemos añadir un comentario a cada columna como información adicional sobre alguna característica especial de la columna, utilizando COMMENT y entra en el apartado de documentación de la base de datos:

```
CREATE TABLE provincia5 (clave INT AUTO_INCREMENT PRIMARY KEY COMMENT 'Clave
principal', nombre CHAR(50) NOT NULL, poblacion INT NULL DEFAULT 5000);
```

Índice

Un índice (o KEY, o INDEX) es un grupo de datos que MySQL asociado con una o varias columnas de la tabla. En este grupo de datos aparece la relación entre el contenido y el número de fila donde está ubicado. Los índices, como los índices de los libros, sirven para agilizar y optimizar las consultas a las tablas, evitando que MySQL tenga que revisar todos los datos disponibles para devolver el resultado. Los índices se utilizan para buscar las filas con los valores de una columna específica rápidamente. Por ejemplo, si una tabla tiene 1000 filas utilizando INDEX es por lo menos 100 veces más rápido que realizando una lectura secuencial.

Si la tabla tiene un índice para las columnas, MySQL puede determinar rápidamente cuál es la posición del dato buscado sin tener que revisar toda la tabla.

Podemos crear el índice a la vez que creamos la tabla, usando la palabra INDEX seguida del nombre del índice a crear y columnas a indexar (que pueden ser varias):

```
INDEX nombre_indice (columna_indexada, columna_indexada2...)
```

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

La sintaxis es ligeramente distinta según la clase de índice:

```
PRIMARY KEY (nombre_columna_1 [,nombre_columna2...])
UNIQUE INDEX nombre_indice (columna_indexada1 [,columna_indexada2 ...])
INDEX nombre_index (columna_indexada1 [,columna_indexada2...])
```

Vamos a distinguir tres tipos de índices:

1. El primero tipo de índice corresponde a las claves primarias, o PRIMARY KEY, que como vimos, también se pueden crear en la parte de definición de columnas.
2. El segundo tipo de índice permite definir índices sobre una columna, sobre varias, o sobre partes de columnas. Para definirlos empleamos KEY o INDEX.
3. El tercero nos permite definir índices con claves únicas, también sobre una columna, sobre varias o sobre partes de columnas. Para definir índices con claves únicas se usa la opción UNIQUE.

A continuación los detallamos.

1- Claves primarias: PRIMARY KEY

La sintaxis para definir claves primarias es:

```
definición_columnas... PRIMARY KEY (index_nombre_col,...)
```

El ejemplo anterior que vimos para crear claves primarias, usando esta sintaxis, quedaría así:

```
CREATE TABLE provincia6 (nombre CHAR(20) NOT NULL, poblacion INT NULL DEFAULT
5000, PRIMARY KEY (nombre));
```

A su vez, podemos crear varias claves primarias en la misma sentencia colocándolas entre paréntesis, de la siguiente manera:

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

```
CREATE TABLE mitabla1 (id1 CHAR(2) NOT NULL, id2 CHAR(2) NOT NULL, texto
CHAR(30), PRIMARY KEY (id1, id2));
```

2- Índices: KEY o INDEX

Para definir estos índices se usan indistintamente las opciones **KEY o INDEX.**

```
CREATE TABLE mitabla2 (id INT, nombre CHAR(19), INDEX (nombre));
```

O su equivalente:

```
CREATE TABLE mitabla3 (id INT, nombre CHAR(19), KEY (nombre));
```

También podemos crear un índice sobre parte de una columna:

Este ejemplo usará sólo los cuatro primeros caracteres de la columna 'nombre' para crear el índice.

```
CREATE TABLE mitabla4 ( id INT, nombre CHAR(19), INDEX (nombre(4)));
```

3- Claves únicas: UNIQUE

Para definir índices con claves únicas se usa la opción **UNIQUE.**

Un índice único no permite la inserción de filas con claves repetidas. La excepción es el valor NULL, que sí se puede repetir.

```
CREATE TABLE mitabla5 (id INT, nombre CHAR(19), UNIQUE (nombre));
```

Tanto los índices normales como los de claves únicas pueden tomar valores NULL. A diferencia de las claves primarias, que no pueden contener valores nulos. Es por esto que las siguientes definiciones son equivalentes:

```
CREATE TABLE mitabla6 (id INT, nombre CHAR(19) NOT NULL, UNIQUE (nombre));
```

Y:

```
CREATE TABLE mitabla7 (id INT, nombre CHAR(19), PRIMARY KEY (nombre));
```

Claves foráneas

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

En MySQL, InnoDB es el único motor de almacenamiento que soporta claves foráneas en sus tablas. Sin embargo, esto no impide usarlas en otros tipos de tablas.

La diferencia consiste en que en esas tablas no se verifica si una clave foránea existe realmente en la tabla referenciada, como si se hace en tablas InnoDB y además que no se eliminan filas de una tabla con la definición de una clave foránea.

Hay dos modos de definir claves foráneas en bases de datos MySQL:

- El primero, sólo sirve para documentar. De esta manera, determinaremos una referencia a la vez que se define una columna:

```
CREATE TABLE personas (
       id INT AUTO_INCREMENT PRIMARY KEY,
       nombre VARCHAR(40),
       fecha DATE);

CREATE TABLE telefonos (
       numero CHAR(12),
       id INT NOT NULL REFERENCES personas (id)
       ON DELETE CASCADE ON UPDATE CASCADE);
```

  Hemos definido la referencia 'id' de la tabla 'telefonos', insertando una clave foránea a la columna 'id' en la tabla 'personas'. Pero esta definición no supone ningún comportamiento por parte de MySQL.

- La segunda manera de definirlas es mucho más útil, aunque sólo se aplica a tablas InnoDB. No se añade la referencia en la definición de la columna, se agrega luego de la definición de todas las columnas.

  El ejemplo anterior, usando tablas InnoDB y esta definición de claves foráneas quedará así:

```
CREATE TABLE personas2 (
    -id INT AUTO_INCREMENT PRIMARY KEY,
```

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

```
    -nombre VARCHAR(40),
    fecha DATE)
    ENGINE=InnoDB;

CREATE TABLE telefonos2 (
    numero CHAR(12),
    id INT NOT NULL,
    KEY (id),
    FOREIGN KEY (id) REFERENCES personas2 (id)
    ON DELETE CASCADE ON UPDATE CASCADE)
    ENGINE=InnoDB;
```

La columna que contiene la definición de clave foránea es necesario que esté indexada. MySQL lo hará de manera implícita. si es que no lo declaramos.

De esta manera se define la clave foránea en la columna 'id' de la tabla 'telefonos2' que se relaciona con la columna 'id' de 'personas2'. La definición a su vez fija las tareas a realizar en el caso de que se elimine una fila en la tabla 'personas'.

DROP

El comando DROP es utilizado para eliminar un objeto de la base de datos. Puede aplicarse a una tabla o un índice, como así también a una vista, función, procedimiento o cualquier otro objeto que la base de datos soporte.

Se puede combinar con la sentencia ALTER (que veremos en breve). Ejemplo 1:

```
DROP TABLE TABLA_NOMBRE
```

Ejemplo 2:

```
ALTER TABLE TABLA_NOMBRE (DROP COLUMN CAMPO_NOMBRE1 )
```

ELIMINAR UNA TABLA

Algunas veces es necesario eliminar una tabla, ya sea porque es más sencillo crearla de nuevo que modificarla, o porque ya no es necesaria.

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Para eliminar una tabla se usa la sentencia DROP TABLE.

La sintaxis es simple:

```
DROP TABLE tbl_name [, tbl_name] ...
```

Por ejemplo:

```
DROP TABLE provincia6;
```

Se pueden añadir las palabras IF EXISTS para evitar errores si la tabla a eliminar no existe.

```
DROP TABLE [IF EXISTS] tbl_name [, tbl_name] ...
```

En nuestro ejemplo:

```
DROP TABLE provincia6;
ERROR 1051 (42S02): Unknown table 'provincia6'
```

Para evitar el mensaje de error:

```
DROP TABLE IF EXISTS provincia6;
```

ELIMINAR UNA BASE DE DATOS

De modo parecido, se pueden eliminar bases de datos completas, usando la sentencia DROP_DATABASE.

La sintaxis también es muy simple:

```
DROP DATABASE [IF EXISTS] db_name
```

Hay que tener cuidado, ya que al borrar cualquier base de datos se elimina también cualquier tabla que contenga.

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

ALTER

El comando ALTER se utiliza para modificar la estructura de un objeto. Se pueden agregar o quitar campos a una tabla, modificar el tipo de un campo, agregar índices o quitarlos, modificar un trigger, entre otras cosas..

Por Ejemplo: Agregar columna a una tabla

```
ALTER TABLE TABLA_NOMBRE (ADD NUEVO_CAMPO INT UNSIGNED)
ALTER TABLE Empleados (ADD Salario CURRENCY)
```

(Agrega un campo Salario de tipo Moneda a la tabla Empleados).
De la misma forma podemos eliminar una columna:

```
ALTER TABLE Empleados DROP Salario
```

(Elimina el campo Salario de la tabla Empleados).

Resumiendo, "ALTER TABLE" se usa para:

- añadir nuevos campos,
- eliminar campos existentes,
- modificar el tipo de dato de un campo,
- cambiar el nombre de un campo,
- agregar o quitar modificadores como "NULL", "UNSIGNED", "AUTO_INCREMENT",
- agregar o eliminar la clave primaria,
- agregar y eliminar índices,
- renombrar una tabla.

"ALTER TABLE" hace una copia temporal de la tabla original, realiza los cambios en la copia, luego borra la tabla original y renombra la copia.

Realizaremos un ejemplo modificando los campos de una tabla.

Para ello utilizamos la tabla "libros", definida con la siguiente estructura:

- código, INT UNSIGNED AUTO_INCREMENT, CLAVE PRIMARIA,

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

- titulo, VARCHAR(40) NOT NULL,
- autor, VARCHAR(30),
- editorial, VARCHAR (20),
- precio, DECIMAL(5,2) UNSIGNED.

Agregar campos:

Agregaremos el campo "cantidad", que será de tipo SMALLINT UNSIGNED NOT NULL.

La sentencia será de la siguiente manera: primero colocamos "ALTER TABLE" y luego el nombre de la tabla a modificar. Seguido por "ADD" y el nombre del campo, en este caso “cantidad” con su tipo y los modificadores.

```
ALTER TABLE libros ADD cantidad SMALLINT UNSIGNED NOT NULL;
```

Agreguemos otro campo a la tabla:

```
ALTER TABLE libros ADD edicion DATE;
```

- Si agregamos un campo que ya existe anteriormente, nos aparecerá un mensaje de error indicando que el campo ya existe en la tabla y la sentencia no se ejecuta.
- Si no especificamos la posición en donde deseamos agregar el campo, se colocará al final de los campos existentes. La manera de indicar la posición deseada será "AFTER" colocándolo así luego del campo especificado.

```
ALTER TABLE libros ADD CANTIDAD TINYINT UNSIGNED AFTER autor;
```

Eliminar campos:

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Como dijimos anteriormente, "ALTER TABLE" nos permite modificar la estructura de una tabla, podemos usarla entonces también para eliminar un campo, usaremos "DROP" y el nombre del campo a eliminar.

Continuamos con nuestra tabla de ejemplo llamada "libros".

Para eliminar el campo "edicion" tipeamos:

```
ALTER TABLE libros DROP edicion;
```

- Si tratamos de borrar un campo que no existe en nuestra tabla nos va a aparecer un mensaje de error y la acción no se ejecutará. A su vez podremos eliminar 2 campos en una misma sentencia de esta manera:

```
ALTER TABLE libros DROP editorial, DROP cantidad;
```

- Si eliminamos un campo que es parte de un índice, también se borrará el índice. Hay que tener precaución al eliminar un campo ya que éste puede ser una clave primaria, y si lo eliminamos no nos saldrá ninguna advertencia.
- Si una tabla tiene sólo un campo, este no podrá ser eliminado.
- Si eliminamos un campo clave, la clave también se elimina.

```
ALTER TABLE libros DROP codigo;
```

Modificar el tipo de dato:

Con "ALTER TABLE" podremos también modificar el tipo de campo, como así también sus atributos.

Seguimos utilizando nuestra tabla "libros", definida con la siguiente estructura:

- código, int unsigned,
- titulo, varchar(30) not null,
- autor, varchar(30),
- editorial, varchar (20),
- precio, decimal(5,2) unsigned,
- cantidad int unsigned.

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Queremos modificar el tipo del campo "cantidad", lo definiremos como SMALLINT UNSIGNED, ya que no utilizaremos valores mayores a 50000. Usaremos "ALTER TABLE" luego el nombre de la tabla y "MODIFY" seguido del nombre del campo con su tipo y los modificadores.

```
ALTER TABLE libros MODIFY cantidad SMALLINT UNSIGNED;
```

Modificaremos también el campo "titulo" para agregar campos de hasta 60 caracteres y a su vez, no permitir valores nulos, tipeamos:

```
ALTER TABLE libros MODIFY titulo VARCHAR(60) NOT NULL;
```

Debemos tener en cuenta que, si por ejemplo, tenemos un campo de texto de longitud 60 y lo modificamos a 40 caracteres, cambiarán los registros que ya están en la tabla, y si este supera el máximo definido, se cortará y quedará incompleto. Por lo cual hay que estar atentos a los cambios que realizamos en las tablas con datos cargados.

Por esto mismo. si un campo permite valores nulos, con registros ya cargados y se lo modifica a "NOT NULL", se modificarán todos los registros con valor nulo por el valor por defecto según el tipo (cadena vacía si es tipo texto y 0 si es numérico).

Otro caso podría ser, en un campo de tipo DECIMAL (5,2) con un registro "900.00" que lo modificamos a "decimal(4,2)". Se cambiará en su lugar, el valor límite más cercano, que sería en este caso "99.99".

Si intentamos definir "auto_increment" un campo que no es clave primaria, no se realizará la acción y nos mostrará un mensaje de error indicándonos que el campo debe ser clave primaria. Por ejemplo:

```
ALTER TABLE libros MODIFY codigo INT UNSIGNED AUTO_INCREMENT;
```

Agregar o quitar campos y atributos:

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

"ALTER TABLE" combinado con "MODIFY" permite agregar o quitar campos y atributos de campos. Si modificamos el valor por defecto ("DEFAULT") de un campo usando "MODIFY" vamos a tener que colocar su tipo y todos los modificadores otra vez. Si queremos definir únicamente el valor por defecto, podremos realizarlo de la siguiente manera::

```
ALTER TABLE libros ALTER autor SET DEFAULT 'Varios';
```

Para eliminar el valor por defecto podemos usar DROP:

```
ALTER TABLE libros ALTER autor DROP DEFAULT;
```

Cambiar nombre de los campos:

Con "ALTER TABLE" podremos cambiar el nombre de las columnas de una tabla. Usaremos "ALTER TABLE" seguido del nombre de la tabla y colocaremos "CHANGE" el nombre actual y luego el nombre nuevo con su tipo y los modificadores, ya que además del nombre podremos cambiar el tipo y sus modificadores.

Continuamos con nuestra tabla "libros", definida con la siguiente estructura:

- código, int unsigned auto_increment,
- nombre, varchar(60),
- autor, varchar(30),
- editorial, varchar (20),
- costo, decimal(5,2) unsigned,
- cantidad int unsigned,
- clave primaria: código.

Vamos a cambiar el campo "nombre" por "titulo" y no podrá contener valores nulos:

```
ALTER TABLE libros CHANGE nombre titulo VARCHAR(60) NOT NULL;
```

Cambiaremos también el campo "costo" por "precio":

```
ALTER TABLE libros CHANGE costo precio DECIMAL (5,2);
```

Agregar clave primaria:

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Con "ALTER TABLE" podremos agregar una clave primaria a una tabla existente. Usamos "ALTER TABLE" con "ADD PRIMARY KEY" y colocaremos el campo entre paréntesis.

Para agregar una clave primaria a una tabla existente usamos:

```
ALTER TABLE libros ADD PRIMARY KEY (codigo);
```

- Si deseamos que nuestra clave primaria sea autoincrementada, deberemos definir este atributo con "MODIFY" y "AUTO_INCREMENT", una vez que está creada.
  No podremos definirla como autoincrementada al mismo tiempo que la creamos. Por esto mismo debemos realizar los pasos en orden, ya que no podremos definir el "AUTO_INCREMENT" y luego convertirla en clave primaria.
- No hay que olvidar que en una tabla puede haber solamente una clave primaria, por lo que si queremos agregar una segunda clave primaria, nos saldrá un mensaje de error.
- A su vez si en un campo hay valores repetidos, no podremos seleccionarlo como clave primaria, nos saldrá un mensaje de error y la operación no se realiza.

Eliminar clave primaria:

Podremos eliminar la clave primaria utilizando "ALTER TABLE" seguido de "DROP PRIMARY KEY", no importa si fue definida al crear la tabla o definida luego.

```
ALTER TABLE libros DROP PRIMARY KEY;
```

Si tenemos una clave primaria que posee el atributo "AUTO_INCREMENT", no se podrá eliminar, y nos aparecerá un mensaje de error avisandonos. Lo que debemos

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

realizar primero es modificarlo y quitarle el atributo "AUTO_INCREMENT", para luego eliminar exitosamente la clave primaria..

Renombrar la tabla:

Podemos además cambiar el nombre de una tabla Usamos "ALTER TABLE" seguido del nombre actual, "RENAME" y el nuevo nombre.

Por ejemplo cambiaremos el nombre de la tabla "libros" por "ejemplares":

```
ALTER TABLE libros RENAME ejemplares;
```

También podemos cambiar el nombre a una tabla usando la siguiente sintaxis:

```
RENAME TABLE datos TO contactos;
```

Si queremos intercambiar el nombre de una tabla con otra, deberemos hacerlo de izquierda a derecha, utilizando un nombre temporal para el cambio. Por ejemplo intercambiaremos los nombres de la tabla “amigos” y la tabla “contactos”, utilizando “auxiliar”:

```
RENAME TABLE amigos TO auxiliar, contactos TO amigos, auxiliar TO contactos;
```

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

DML (Data Manipulation Language)

Su misión es la manipulación de datos: nos permite seleccionar, insertar, eliminar y actualizar datos. Por lo general, utilizaremos los siguientes comandos:

- SELECT
- INSERT
- UPDATE
- DELETE

Inserción de nuevas filas

Podremos insertar nuevas filas en nuestra tabla utilizando la sentencia INSERT. Debemos especificar el nombre de la tabla a la que añadiremos las nuevas filas, y los valores de cada columna. Las columnas de tipo cadena o fechas deben estar entre comillas sencillas o dobles, para las columnas numéricas esto no es indispensable, aunque pueden estarlo..

```
INSERT INTO personas VALUES (Pedro,'1977-04-19');
INSERT INTO personas VALUES (Juan,'1978-01-15');
```

- Si necesitamos agregar varios registros podríamos hacerlo en una sola sentencia:
  ```
  INSERT INTO personas VALUES (Mariano,'1972-09-07'), (Verónica,'1976-06-03');
  ```
- En el caso de no necesitar asignar un valor concreto para una columna, podemos asignarle, usando la palabra DEFAULT, el valor por defecto indicado para esa columna cuando se creó la tabla.

  Por ejemplo, habíamos definido el valor por defecto para población en 5000, por lo cual se asignará ese valor para la fila correspondiente a Buenos Aires.

  ```
  INSERT INTO provincia VALUES (Buenos Aires, DEFAULT);
  ```

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

- Otra opción será indicar una lista de columnas para las que se van a suministrar valores. A las columnas que no se nombren en esa lista se les asigna el valor por defecto.

  Este sistema, además, permite usar cualquier orden en las columnas, con la ventaja, con respecto a la anterior forma, de que no necesitamos conocer el orden de las columnas en la tabla para poder insertar datos:

  ```
  INSERT INTO provincia (poblacion, nombre) VALUES (7000000, 'Santa Fe'), (9000000, 'Córdoba'),(3500000,'Corrientes');
  ```

  Cuando creamos la tabla "provincia" definimos tres columnas: 'clave', 'nombre' y 'poblacion'. Ahora hemos insertado tres filas, en las que no hemos insertado la clave, ya que esta se calcula automáticamente, auto-incrementandose. Además hemos alterado el orden de 'nombre' y 'poblacion'.

- Existe otra sintaxis alternativa, que consiste en indicar el valor para cada columna:
  ```
  INSERT INTO provincia SET nombre='Mendoza', poblacion=8000000;
  ```

  Una vez más, a las columnas para las que no indiquemos valores se les asignarán sus valores por defecto.

Para que una fila se considere duplicada debe tener el mismo valor en una clave principal o clave única.

En tablas en las que no exista clave primaria ni índices de clave única no se hablará de filas duplicadas. Es más, en esas tablas seguramente existan filas con los mismos valores para todas las columnas.

Por ejemplo, en mitabla5 tenemos una clave única sobre la columna 'nombre':

```
INSERT INTO mitabla5 (id, nombre) VALUES (1, 'Carlos'), (2, 'Felipe'), (3, 'Antonio'), (4,'Carlos'), (5, 'Juan');
ERROR 1062 (23000): Duplicate entry 'Carlos' for key 1
```

Si intentamos insertar dos filas con el mismo valor de la clave única se produce un error y la sentencia no se ejecuta.

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Reemplazar filas

Existe una sentencia REPLACE, que es una alternativa para INSERT, que sólo se diferencia en que si existe algún registro anterior con el mismo valor para una clave primaria o única, se elimina el viejo y se inserta el nuevo en su lugar.

```
REPLACE [LOW_PRIORITY | DELAYED]
      [INTO] tbl_name
      [(col_name,...)]
      VALUES ({expr | DEFAULT},...) ,(...),...

REPLACE [LOW_PRIORITY | DELAYED]
      [INTO] tbl_name
SET col_name={expr | DEFAULT}, ...
```

Por ejemplo, sustituiremos las filas correspondientes a 'Mendoza' y 'Buenos Aires', que ya existían en la tabla y colocaremos 'San Luis' que no estaba.

```
REPLACE INTO provincia (nombre, poblacion) VALUES ('Mendoza', 7200000),
('Buenos Aires', 9200000), ('San Luis', 6000000);
```

Las mismas sintaxis que existen para INSERT, están disponibles para REPLACE:

```
REPLACE INTO provincia VALUES ('Buenos Aires', 9500000);
REPLACE INTO provincia SET nombre='Santa Fe', poblacion=10000000;
```

Actualizar filas

Podemos modificar valores de las filas de una tabla usando la sentencia UPDATE. En su forma más simple, los cambios se aplican a todas las filas, y a las columnas que especifiquemos.

```
UPDATE [LOW_PRIORITY] [IGNORE] tbl_name
      SET col_name1=expr1 [,col_name2=expr2 ...]
      [WHERE where_definition]
[ORDER BY ...]
[LIMIT row_count]
```

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Por ejemplo, podemos aumentar en un 10% la población de todas las ciudades de la tabla provincia usando esta sentencia:

```
UPDATE provincia SET poblacion= poblacion*1.10;
```

Podemos, del mismo modo, actualizar el valor de más de una columna, separándolas en la sección SET mediante comas.

En este ejemplo hemos incrementado el valor de la columna 'clave' en 10 y disminuido el de la columna 'poblacion' en un 3%, para todas las filas.

```
UPDATE provincia SET clave= clave+10, población = población *0.97;
```

Pero no tenemos por qué actualizar todas las filas de la tabla. Podemos limitar el número de filas afectadas de varias formas.

- La primera es mediante la cláusula WHERE. Usando esta cláusula podemos establecer una condición. Sólo las filas que cumplan esa condición serán actualizadas:

  ```
  UPDATE ciudad5 SET poblacion=poblacion*1.03 WHERE nombre='Mendoza';
  ```

  En este caso sólo hemos aumentado la población de las ciudades cuyo nombre sea 'Mendoza'. Las condiciones pueden ser más complejas. Existen muchas funciones y operadores que se pueden aplicar sobre cualquier tipo de columna, y también podemos usar operadores booleanos como AND u OR, desarrollaremos estos operadores más adelante.

- Otra forma de limitar el número de filas afectadas es usar la cláusula LIMIT. Esta cláusula permite especificar el número de filas a modificar. En este ejemplo decrementaremos en 10 unidades la columna clave de las dos primeras filas.

  ```
  UPDATE provincia SET clave=clave-10 LIMIT 2;
  ```

Esta cláusula se puede combinar con WHERE, indicando que sólo las 'n' primeras filas que cumplan una determinada condición se modifiquen. Esta combinación no se suele usar mucho, porque si no existen claves primarias o únicas, el orden de las filas va a ser arbitrario, no tiene sentido seleccionarlas usando sólo la cláusula LIMIT.

La cláusula LIMIT se suele asociar a la cláusula ORDER BY. Por ejemplo, si queremos modificar la fila con la fecha más antigua de la tabla 'personas', usaremos esta sentencia:

```
UPDATE personas SET fecha="1985-04-12" ORDER BY fecha LIMIT 1;
```

Si queremos modificar la fila con la fecha más reciente, usaremos el orden inverso, es decir, el descendente

```
UPDATE personas SET fecha="2001-12-02" ORDER BY fecha DESC LIMIT 1;
```

Cuando exista una clave primaria o única, se usará ese orden por defecto, si no se especifica una cláusula ORDER BY.

Eliminar filas

Para eliminar filas se usa la sentencia DELETE. La sintaxis es muy parecida a la de UPDATE:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM table_name
[WHERE where_definition] [ORDER BY...] [LIMIT row_count]
```

- La forma más simple es no usar ninguna de las cláusulas opcionales:

  ```
  DELETE FROM provincia;
  ```

  De este modo se eliminan todas las filas de la tabla. Pero es más frecuente que solo queramos eliminar ciertas filas que cumplan determinadas condiciones.

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

- La forma más normal de hacer esto es usar la cláusula WHERE:

```
DELETE FROM provincia WHERE clave=2;
```

- También podemos usar las cláusulas LIMIT y ORDER BY del mismo modo que en la sentencia UPDATE, por ejemplo, para eliminar las dos ciudades con más población:

```
DELETE FROM provincia ORDER BY poblacion DESC LIMIT 2;
```

- Si quisiéramos vaciar nuestra tabla y eliminar todas las filas podríamos utilizar DELETE sin condiciones, que borra secuencialmente todas las filas de la tabla. Pero existe otra sentencia que realiza la misma tarea de una forma mucho más rápida y eficiente: TRUNCATE. Esta sentencia borra la tabla y la vuelve a crear vacía.

```
TRUNCATE provincia;
```

Selección de datos

Ahora que ya tenemos nuestra base de datos y sabemos cómo añadir y modificar datos, aprenderemos a extraer los datos contenidos en ellas.

Utilizaremos la sentencia SELECT. Su sintaxis es un tanto compleja, pero veremos una forma más general:

```
SELECT [ALL | DISTINCT | DISTINCTROW] expresion_select,...
FROM referencias_de_tablas
WHERE condiciones [GROUP BY {nombre_col | expresion | posicion} [ASC | DESC],
... [WITH ROLLUP]]
[HAVING condiciones]
[ORDER BY {nombre_col | expresion | posicion} [ASC | DESC] ,...]
[LIMIT {[desplazamiento,] contador | contador OFFSET desplazamiento}]
```

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Forma incondicional

La manera más sencilla consiste en pedir todas las columnas y sin especificar ninguna condición.

```
SELECT * FROM personas;
```

Limitar las columnas: proyección

Una de las operaciones del álgebra relacional es la proyección, que consistía en seleccionar atributos concretos de una relación.

Utilizando la sentencia SELECT podemos realizar una proyección de una tabla, seleccionando únicamente las columnas de las que queremos obtener datos.

En el ejemplo, hemos reemplazado donde dice "expresion_select", por '*', que significa que se mostrarán todas las columnas. A su vez, podremos usar una lista de columnas, y de ese modo sólo se mostrarán esas:

```
SELECT nombre FROM personas;
```

Mostrar filas repetidas

Debido a que hemos excluido las columnas únicas, es posible que se puedan repetir filas,

Por ejemplo, añadamos las siguientes filas a nuestra tabla:

```
INSERT INTO personas VALUES ('Alicia', '1972-09-07'), ('Antonio', '1976-06-03');
```

Vemos que existen dos valores de filas repetidos, Mariano, para la fecha '1972-09-07' y Verónica para '1976-06-03'.

La sentencia que hemos usado asume el valor por defecto (ALL) para el grupo de opciones ALL, DISTINCT y DISTINCTROW, estas dos últimas siendo sinónimos.

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Si usamos DISTINCT, hará que sólo se muestren las filas diferentes.

Limitar las filas: selección

Otra de las operaciones del álgebra relacional era la selección, que seleccionará filas de una relación mientras cumplan determinadas condiciones, esto es lo que más destacamos de las BD, sus consultas.

SELECT nos permite usar condiciones como parte de su sintaxis, es decir, para hacer selecciones. Lo utilizaremos junto a la cláusula WHERE:

```
SELECT * FROM personas WHERE nombre="Mariano";
SELECT * FROM personas WHERE fecha>="1976-06-03";
SELECT * FROM personas WHERE fecha>="1972-09-07" AND fecha <="2011-01-01";
```

Cuando utilizamos WHERE podremos usar todas las funciones disponibles en MySQL, a excepción de las de resumen o reunión, que veremos a continuación, ya que están diseñadas para usarse exclusivamente con GROUP BY.

A su vez, se puede aplicar lógica booleana para crear expresiones complejas, estos operadores son AND, OR, XOR Y NOT.

Agrupar filas

Utilizando la sentencia SELECT podremos agrupar filas según los distintos valores de una columna, usando la cláusula GROUP BY.

Es similar a DISTINCT, pero, la cláusula GROUP BY es más potente, a continuación veremos sus diferencias:

- La primera diferencia que observamos utilizando GROUP BY es la manera en que se ordenan los valores de la columna indicada. En este caso, las columnas aparecerán ordenadas por fechas.

  ```
  SELECT fecha FROM personas GROUP BY fecha;
  ```

- Otra diferencia es que se eliminan los valores duplicados aún si la proyección no contiene filas duplicadas, por ejemplo:

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

```
SELECT nombre, fecha FROM personas GROUP BY fecha;
```

- La principal diferencia es que nos permite usar funciones de resumen o reunión. Por ejemplo, la función COUNT(), que sirve para contar las filas de cada grupo, esta sentencia muestra todas las fechas diferentes y el número de filas para cada fecha.

```
SELECT fecha, COUNT(*) AS cuenta FROM personas GROUP BY fecha;
```

Existen otras funciones de resumen o reunión, como MAX(), MIN(), SUM(), AVG(), STD(), VARIANCE(), etc. Estas funciones también se pueden usar sin la cláusula GROUP BY siempre que no se proyecten otras columnas

Por ejemplo, esta sentencia muestra el valor más grande de 'nombre' de la tabla 'personas', es decir, el último por orden alfabético.

```
SELECT MAX(nombre) FROM personas;
```

Cláusula HAVING

La cláusula HAVING permite hacer selecciones en situaciones en las que no es posible usar WHERE.

La cláusula WHERE no se puede aplicar a columnas calculadas mediante funciones de reunión, como en este ejemplo:

```
CREATE TABLE muestras (ciudad VARCHAR(40), fecha DATE, temperatura TINYINT);

INSERT INTO muestras (ciudad,fecha,temperatura) VALUES ('Madrid',
'2005-03-17', 23), ('París','2005-03-17', 16), ('Berlín', '2005-03-17', 15),
('Madrid', '2005-03-18', 25), ('Madrid', '2005-03-19', 24), ('Berlín',
'2005-03-19', 18);

SELECT ciudad, MAX(temperatura) FROM muestras GROUP BY ciudad HAVING
MAX(temperatura)>16;
```

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Nos daría como resultado:

| BERLÍN | 18 |

| MADRID | 25 |

Ordenar resultados

Podemos añadir una cláusula de orden ORDER BY para obtener resultados ordenados por la columna que queramos:

```
SELECT * FROM personas ORDER BY fecha;
```

Podremos elegir el orden en el que se muestran las filas. Podrá ser ascendente ASC o descendente DESC. Por defecto se usa el orden ascendente, de modo que el modificador ASC es opcional.

```
SELECT * FROM personas ORDER BY fecha DESC;
```

Podremos limitar el número de filas que queremos que devuelva, utilizando LIMIT:

```
SELECT * FROM personas LIMIT 3;
```

Esta cláusula se suele usar para obtener filas por grupos, y no sobrecargar demasiado al servidor, o a la aplicación que recibe los resultados. Para poder hacer esto la cláusula LIMIT admite dos parámetros. Cuando se usan los dos, el primero indica el número de la primera fila a recuperar, y el segundo el número de filas a recuperar. Podemos, por ejemplo, recuperar las filas de dos en dos:

```
SELECT * FROM personas LIMIT 0,2;
SELECT * FROM personas LIMIT 2,2;
SELECT * FROM personas LIMIT 4,2;
SELECT * FROM personas LIMIT 6,2;
```

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso
(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

En resumen

En esta unidad nos introdujimos a MYSQL como sistema de administración relacional de base de datos.
Analizamos sentencias SQL para definir datos (DDL) y para manipularlos (DML).
MySQL, junto con PHP, nos permitirá generar aplicaciones web dinámicas que puedan responder a peticiones de los usuarios y gestionar los datos que se introduzcan.

Bibliografía utilizada y sugerida

- Casillas Santillán, G. G. y Pérez Mora. Bases de datos MySQL. Recuperado de: http://ual.dyndns.org/biblioteca/Bases%20de%20Datos%20Avanzado/Pdf/05%20Bases%20de%20datos%20en%20MySQL.pdf
- Castán Salinas, A. Guía rápida de administración de MySQL. Recuperado de: http://www.xtec.cat/~acastan/textos/Administracion%20de%20MySQL.html
- MySQL 5.7 Reference Manual - Data Types. Recuperado de: https://dev.mysql.com/doc/refman/5.7/en/data-types.html
- Vélez de Guevara, L. Gestión de Bases de Datos. Recuperado de: https://readthedocs.org/projects/gestionbasesdatos/downloads/pdf/lates