Laboratorio de Bases de Datos (EBB)

Unidad I - Introducción

Departamento de Electricidad, Electrónica y Computación
Facultad de Ciencias Exactas y Tecnología
Universidad Nacional de Tucumán

Primer Cuatrimestre 2020

Introducción [1 | 2]

- Bases de Datos
  - Tipos de Bases de Datos
  - Objetos de Bases de Datos
- Trabajo con un SGBDR
  - Diseño de aplicaciones
  - Implementación de BDs
  - Administración de BDs

Introducción [2 | 2]

- Creación de Bases de Datos
  - Almacenamiento y Transacciones
  - Creación y modificación de BDs
  - Tipos de datos y creación de tablas
  - Planificación de la capacidad

- Código: https://github.com/luisenieto/LBD2020.git

Tipos de Bases de Datos [1 | 5]

- **SGBD**
  - Sistema Gestor de Bases de Datos
  - Compuesto por un conjunto de aplicaciones que se encargan de la creación y acceso a las BDs
  - Si se soportan bases de datos relacionales, se habla de un SGBDR
  - Ej. de SGBDR: Oracle, SQL Server, MySQL, PostgreSQL, etc

- Las siglas equivales en inglés para SGBD y SGBDR son DBMS y RDBMS respectivamente.

Tipos de Bases de Datos [2 | 5]

- Al instalar un SGBDR, el mismo se ejecuta como servicio
- Ventajas:
  - Continúa la ejecución después que se desconecta el usuario
  - Comienza a ejecutarse antes que se conecte el usuario
  - Generalmente se ejecuta bajo una cuenta de sistema o una creada para este fin
  - Se puede administrar remotamente

Servicio de MySQL

- En Ubuntu, para administrar el servicio de mysql se puede ejecutar:

```
service mysql start | stop | status
```

Tipos de Bases de Datos [3 | 5]

- Según el motor y su versión, se deberán instalar o no herramientas gráficas para su administración
- Para MySQL, una de estas herramientas puede ser “Workbench”

Conexión a MySQL con Workbench

- Para MySQL, además de Workbench también se puede usar Navicat, Sequel Pro, HeidiSQL, phpMyAdmin, SQLyog, SQLWave, dbForge Studio, etc.

Tipos de Bases de Datos [4 | 5]

- **Base de Datos**
  - Conjunto de datos que pertenecen a un mismo contexto que se pueden guardar para ser usados posteriormente
  - Los SGBD permiten guardar y posteriormente acceder a los datos en forma rápida y estructurada

Tipos de Bases de Datos [5 | 5]

- **BDs del sistema**: guardan información del SGBDR (no hay que borrarlas)
  - `mysql`
  - `information_schema`
  - `performance_schema`
  - `sys`
- **BDs de usuario**: son las BDs de producción
  - El SGBDR puede manejar muchas de éstas

BDs de sistema y usuario en MySQL

- En el caso de MySQL, para ver las BDs de sistema se puede usar el comando "`show databases`", o desde Workbench editar las preferencias, entrada "SQL Editor", opción "Show metadata and internal schemas".

Objetos de Bases de Datos [1 | 7]

- **Tablas**: colección de filas con la misma cantidad de columnas
- **Tipos de datos**: valores permitidos para cada columna
- **Restricciones (*constraints*)**: reglas que rigen los valores permitidos para las columnas. Proporcionan el mecanismo de integridad de datos

- Una base de datos no es únicamente un conjunto de tablas y sus respectivos datos.

Objetos de Bases de Datos [2 | 7]

- **Índices**: estructura de almacenamiento que brinda un acceso rápido a los datos y fuerza la integridad de los mismos
- **Valores por defecto**: valores que toman las columnas cuando no se proporciona alguno
- **Reglas**: información que define los valores para las columnas

Objetos de Bases de Datos [3 | 7]

- **Vistas**: forma de ver los datos de una o más tablas
- **Procedimientos almacenados**: colección de sentencias SQL, con un nombre, que se ejecutan como un todo. Admiten parámetros de entrada y/o salida
- **Desencadenadores (*triggers*)**: procedimientos almacenados que se ejecutan automáticamente después de producida una acción

Objetos de Bases de Datos [4 | 7]

- **Funciones**: rutinas formadas por una o más sentencias SQL
- **Usuarios**: entidad que puede solicitar un recurso del servidor de BD
- **Roles (grupos)**: conjunto de usuarios de una misma BD que comparten los mismos permisos

Objetos de Bases de Datos [5 | 7]

- Los objetos se organizan en una determinada jerarquía:
  - **MySQL**: servidor - BD - objeto

Objetos de Bases de Datos [6 | 7]

- Sobre los nombres de objetos en MySQL:
  - Pueden ir entre comillas simples invertidas (' ` ')
  - En los nombres de BDs y tablas no se puede emplear '.', '\' ni '/', ya que las BDs se implementan como directorios y las tablas como archivos
  - Las palabras reservadas y los nombres de funciones son insensibles a mayúsculas/minúsculas, mientras que los nombres de BDs y tablas sí lo son

- El empleo de la comilla simple invertida permite emplear casi cualquier caracter en el nombre de los objetos (hasta se pueden especificar nombres reservados).

- Cada vez que se crea una BD en MySQL, en su directorio de datos se crea un directorio cuyo nombre es el de la BD. Por cada tabla que se cree en la BD, se crea un archivo con este nombre dentro del directorio que corresponde a la BD. Por esta razón, como MySQL se puede instalar en Linux, y el mismo es sensible a mayúsculas y minúsculas, se debe tener cuidado con los nombres de tablas y bases de datos.

Objetos de Bases de Datos [7 | 7]

- Formas para referenciar los objetos:
  - **Absoluta**: se especifica toda la jerarquía a la cual pertenece un determinado objeto
    - Ej: `select * from TrabajosGraduacion.Alumnos`
  - **Relativa**: se especifica sólo una parte de la jerarquía a la cual pertenece un determinado objeto
    - Ej: `select * from Alumnos`

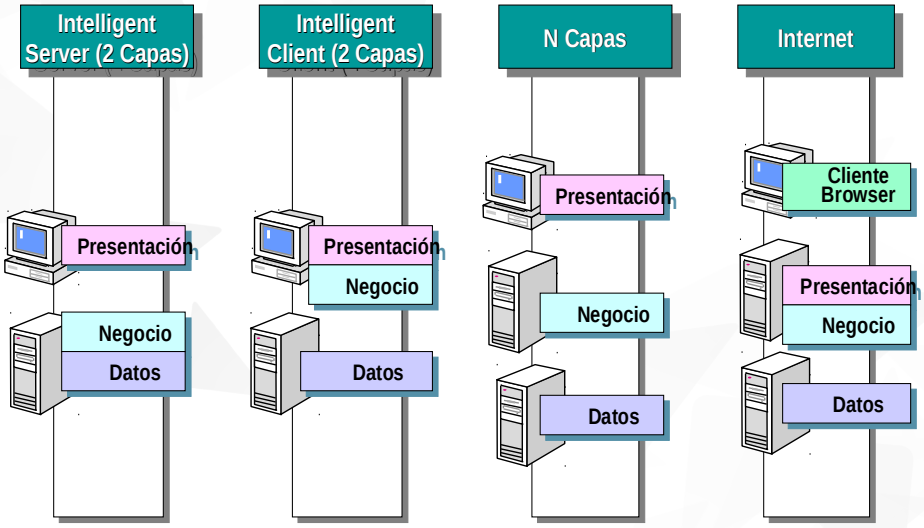
- Cada objeto creado tiene un nombre único. Cuando se emplean nombres completos se dice que el objeto está completamente cualificado.

- No siempre hay que especificar el nombre completo. Se pueden emplear nombres relativos. Si no se especifica la BD, se asume la que en ese momento sea la predeterminada.

Diseño de aplicaciones [1 | 4]

- Capas para implementar un modelo C/S:
  - **Presentación**: lógica para presentar los datos a los usuarios. Casi siempre la implementa el cliente
  - **Negocio**: lógica de la aplicación y reglas de negocio. El SGBDR puede estar involucrado en esta capa
  - **Datos**: definición de la BD, integridad, procedimientos almacenados, etc. El SGBDR se involucra estrechamente con esta capa

Diseño de aplicaciones [2 | 4]



Diseño de aplicaciones [3 | 4]

- Arquitecturas típicas:
  - **Intelligent Server (2 capas)**: la mayoría de los procesos ocurre en el servidor con la lógica de interfaz en el cliente. La lógica de negocios es implementada casi por completo en la BD
  - **Intelligent Client (2 capas)**: la mayoría de los procesos ocurre en el cliente con los datos residiendo en el servidor. El rendimiento es malo por el tráfico en la red

Diseño de aplicaciones [4 | 4]

- Arquitecturas típicas:
  - **N capas**: el procesamiento es dividido en un servidor de BD, uno o varios servidores de aplicación y los clientes. Es complejo pero muy escalable
  - **Internet**: el proceso es dividido en 3 capas con los servicios de negocio y de presentación residiendo en un servidor web, y los clientes usando simples navegadores

Implementación de BDs [1 | 4]

- Implementar una BD implica:
  - Diseñar la BD
  - Crear la BD y sus objetos
  - Probar y poner a punto la aplicación y la BD
  - Planear el desarrollo
  - Administrar la aplicación

Implementación de BDs [2 | 4]

- **Diseñar la BD**
  - Uso eficiente del HW que permita crecimiento futuro, identificando y modelando los objetos de la BD y la lógica de la aplicación, la información de cada objeto y sus relaciones
- **Crear la BD y sus objetos**
  - Tablas, integridad y entrada de datos, procedimientos almacenados, vistas, índices, seguridad, etc

Implementación de BDs [3 | 4]

- **Probar y poner a punto la aplicación y la BD**
  - Las tareas se deben ejecutar de forma rápida y correcta. Junto a un buen diseño, correcto uso de índices y RAID se consigue un buen rendimiento
- **Planear el desarrollo**
  - Analizar la carga de trabajo con el sistema en producción para recomendar una óptima configuración de índices

Implementación de BDs [4 | 4]

- **Administrar la aplicación**
  - Configurar los servidores y clientes, monitorizar el rendimiento en todo momento, gestionar los trabajos, alertas y operadores, manejar la seguridad y las copias de respaldo, etc

Administración de BDs [1 | 3]

- **Administrar una BD involucra:**
  - Instalar y configurar el SGBDR
  - Establecer la seguridad de la red
  - Construir las BDs
  - Manejar las actividades del día a día

Administración de BDs [2 | 3]

- **Construir las BDs:**
  - Reservar espacio en disco
  - Crear trabajos automatizados

Administración de BDs [3 | 3]

- **Manejar las actividades del día a día:**
  - Importación y exportación de datos
  - Copias de respaldo y restauraciones
  - Monitorizar y poner a punto la BD
  - Automatizar todo lo anterior

Almacenamiento y Transacciones [1 | 16]

- **Transacción:**
  - Una o más unidades de trabajo donde todas tienen éxito o ninguna
  - Puede ser un único cambio en la BD, o un conjunto de cambios relacionados que deben completarse todos o ninguno
  - Ejemplo: extracción de una cuenta y depósito en otra

- Cualquier cambio que se realice en una BD (agregado, modificación, eliminación de datos y/o estructura) se hace en el contexto de una transacción. Una transacción puede ser un único cambio en la BD, o una serie de cambios relacionados que deben completarse todos o ninguno.

- Todas las transacciones que se realizan sobre una BD se registran secuencialmente en el registro de transacciones (ver más adelante).

Almacenamiento y Transacciones [2 | 16]

- Propiedades de una transacción:
  - **A**tomicidad
  - **C**onsistencia
  - A**I**slamiento
  - **D**urabilidad

- **Atomicidad**: se completan todas las operaciones de la transacción, o ninguna. Esto evita que las modificaciones se realicen parcialmente.
- **Consistencia**: debe modificar los datos según las reglas definidas.
- **Aislamiento**: cualquier otra actividad concurrente con la transacción no tiene efecto sobre esta última.
- **Durabilidad**: cuando se completa una transacción, sus resultados se guardan.
- A estas propiedades se las conoce por el acrónimo ACID en inglés.

Almacenamiento y Transacciones [3 | 16]

- Las transacciones pueden ser:
  - **Explícitas**: el comienzo y el final están explícitamente definidos
  - **Implícitas**: algunos comandos SQL terminan implícitamente una transacción (`INSERT`, `ALTER`, `SELECT`, `UPDATE`, etc)

- Las transacciones implícitas terminan sin un `COMMIT` (ver más adelante).
- Por defecto, MySQL trabaja en el modo “autocommit”, con lo cual las transacciones implícitas se ejecutan sin necesidad de un `COMMIT`, y no pueden retrotraerse.

Almacenamiento y Transacciones [4 | 16]

Transacciones explícitas

```
START TRANSACTION
  sentencias
COMMIT | ROLLBACK
```

Almacenamiento y Transacciones [5 | 16]

- En MySQL una BD está formada por 2 tipos de archivos:
  - **Archivos de datos**
    - Datos e índices de la tabla (`*.ibd` para InnoDB)
  - **Registro de transacciones**
    - Estructura de datos en disco que permite corregir fallos debidos a transacciones incompletas

Archivos de datos y del registro de transacciones

- Al registro de transacciones MySQL lo llama “registro redo”.

- Los archivos de datos se encuentran dentro de la carpeta correspondiente a una BD, mientras que los archivos del registro de transacciones (comunes a todas las BD) se encuentran en la carpeta de datos de MySQL.

- En el caso de Workbench, en la solapa “Administration”, opción “Server status” se puede ver la carpeta que usa MySQL para los datos.

Almacenamiento y Transacciones [6 | 16]

- Funcionamiento del registro de transacciones:
  - Una aplicación envía un pedido de modificación (transacción) de datos
  - Cuando la transacción termina (`COMMIT`), MySQL la anota en el registro (se completaron con éxito todos los elementos de la transacción)
  - Luego MySQL lee el registro y aplica todas las transacciones terminadas en la BD

Almacenamiento y Transacciones [7 | 16]

- Funcionamiento del registro de transacciones (continuación):
  - Si se produce una falla en MySQL luego que la transacción está anotada en el registro de transacciones, cuando se vuelva a inicializar el servicio, y antes que se establezca la conexión, la misma se aplicará en la BD

Almacenamiento y Transacciones [8 | 16]

- Sobre el registro de transacciones:
  - Por defecto se implementa en disco por 2 archivos llamados `ib_logfile0` e `ib_logfile1`
  - MySQL se encarga de escribir en estos archivos de forma circular (escribe desde el comienzo y cuando llega al final vuelve al comienzo)
  - Por defecto, el tamaño de cada uno de estos archivos es de 5 MB

- Se puede cambiar la cantidad de archivos que forman el registro redo, y/o el tamaño de los mismos:
- 1. Detener el servicio de MySQL
- 2. Editar el archivo `my.cnf`. Para cambiar el tamaño de los archivos, configurar la opción `innodb_log_file_size`, y para aumentar la cantidad de archivos, configurar la opción `innodb_log_files_in_group`.
- 3. Reiniciar el servicio de MySQL

- También se puede cambiar el lugar donde se encuentran los archivos del registro redo, para que estén separados de los archivos de datos. Para esto, se modifica la opción `innodb_log_group_home_dir` en `my.cnf`.

Almacenamiento y Transacciones [9 | 16]

- **Motor de almacenamiento (MySQL):**
  - Componente que maneja las operaciones SQL para distintos tipos de tablas
  - MySQL permite cargar y descargar distintos motores de almacenamiento mientras se ejecuta el servicio
  - A los motores de almacenamiento también se los conoce como motores de tablas o tipos de tablas

Tipos de tablas en MySQL

- MySQL utiliza una arquitectura en la cual se pueden cargar y descargar motores de almacenamiento mientras se está ejecutando el servicio.

- Para determinar qué motores soporta el servidor, se puede usar:

```
SHOW ENGINES;
```

- Para la versión 8.0, el motor de almacenamiento por defecto es InnoDB.

Almacenamiento y Transacciones [10 | 16]

- Motores de almacenamiento en MySQL:
  - **InnoDB**: motor transaccional (conforme a ACID), con capacidades de aplicar y retrotraer transacciones, recuperación en caso de problemas con la BD. Guarda los datos usando índices agrupados, soporta integridad referencial mediante claves propagadas
  - **MyISAM**: usa bloqueos a nivel tabla, limitando las operaciones de lectura/escritura. Usado generalmente en escenarios de sólo lectura o *data warehousing*

Almacenamiento y Transacciones [11 | 16]

- Motores de almacenamiento en MySQL:
  - **Memory**: guarda todos los datos en RAM, permitiendo un acceso rápido. Antes tenía el nombre de HEAP. Se lo está dejando de usar
  - **CSV**: sus tablas son archivos de texto con valores separados por comas. Permite importar/exportar los datos en formato CSV. No emplea índices

Almacenamiento y Transacciones [12 | 16]

- Motores de almacenamiento en MySQL:
  - **Archive**: emplea tablas compactas, sin indexar. Pensado para guardar y recuperar grandes cantidades de datos históricos
  - **Blackhole**: acepta pero no guarda datos (similar al dispositivo /dev/null en Unix). Pensado para escenarios de replicación, donde se envían las sentencias DML a los servidores esclavos, pero el servidor principal no mantiene una copia de los datos

Almacenamiento y Transacciones [13 | 16]

- Motores de almacenamiento en MySQL:
  - **NDB (NDBCLUSTER)**: pensado para aplicaciones que requieren el mayor nivel de disponibilidad y uso
  - **Mrg_MyISAM (Merge)**: permite agrupar lógicamente un conjunto de tablas MyISAM idénticas y referenciarlas como un único objeto. Resulta útil en escenarios de "*data warehousing*"

Almacenamiento y Transacciones [14 | 16]

- Motores de almacenamiento en MySQL:
  - **Federated**: ofrece la capacidad de conectar servidores MySQL separados para crear una BD lógica a partir de varios servidores físicos. Muy bueno para entornos distribuidos
  - **Example**: pensado para quienes quieran escribir su propio motor de almacenamiento. Se pueden crear tablas, pero no se puede guardar ni recuperar nada

Almacenamiento y Transacciones [15 | 16]

- Motores de almacenamiento en MySQL:
  - En una misma BD se pueden usar distintos tipos de tablas
  - El tipo de tabla se especifica en la sentencia `CREATE TABLE`. Para cambiar el tipo de tabla:

```
ALTER TABLE <tabla> ENGINE=<motor>;
```

Almacenamiento y Transacciones [16 | 16]

- Algunas diferencias entre InnoDB y MyISAM:

InnoDB	MyISAM
Soporta bloqueos a nivel fila	Soporta bloqueos a nivel tabla
Soporta claves propagadas	No soporta claves propagadas
Soporta transacciones (se puede hacer `commit` y `roll back`)	No soporta transacciones (no se puede hacer `commit` ni `roll back`)

Creación y modificación de BDs [1 | 2]

- Para crear una BD se puede especificar una sentencia SQL o usar alguna herramienta gráfica
- Se debe especificar el nombre
- Ejemplo:

```
CREATE DATABASE | SCHEMA [IF NOT EXISTS] LBD2020;
```

Creación de BD en MySQL

- Al crear una BD en MySQL, se crea un directorio con el nombre de la misma. En este ejemplo, se crearía el directorio `/var/lib/mysql/LBD2019`.

Creación y modificación de BDs [2 | 2]

- Para borrar una BD se puede usar la sentencia `DROP DATABASE`:

```
DROP DATABASE | SCHEMA [IF EXISTS] LBD2020;
```

- No se puede eliminar una BD cuando:
  - Está siendo restaurada
  - Un usuario está conectado a la misma
  - Se está publicando como parte de una replicación

Tipos de datos y creación de tablas [1 | 43]

- Al crear una tabla, se debe especificar:
  - Nombre (único dentro de la BD)
  - Nombres y tipos de datos de las columnas (únicos dentro de la tabla)
  - Si cada columna admite o no valores nulos (`NULL` o `NOT NULL`)

- La cantidad de tablas que se pueden crear dentro de una misma BD, como así también la cantidad de columnas por tabla, depende del motor de BD y su versión.

Tipos de datos y creación de tablas [2 | 43]

- Creación de tablas:

```
CREATE TABLE [IF NOT EXISTS] Socio (
  <col1> <tipo_dato> INT NOT NULL,
  <col2> <tipo_dato> NOT NULL,
  <col3> <tipo_dato> NOT NULL,
  <col4> <tipo_dato> NULL,
  ...
) ENGINE = <motor de almacenamiento>;
```

Tipos de datos y creación de tablas [3 | 43]

- MySQL soporta diferentes tipos de datos, agrupados en categorías:
  - Numéricos: enteros, con punto decimal fijo, con punto decimal flotante, bits
  - Fechas y horas
  - Cadenas
  - JSON
  - Espaciales

Tipos de datos y creación de tablas [4 | 43]

- **Tipos de datos numéricos:**
  - Enteros: *TINYINT*, SMALLINT, *MEDIUMINT*, INT[EGER], *BIGINT*
  - Con punto decimal fijo: DECIMAL, NUMERIC
  - Con punto decimal flotante: FLOAT, DOUBLE
  - Bit: BIT

- Se puede usar INT o INTEGER.

- Los tipos INTEGER y SMALLINT son del estándar SQL. MySQL también agrega los tipos TINYINT, MEDIUMINT y BIGINT.

Tipos de datos y creación de tablas [5 | 43]

Numéricos enteros:

Tipo	Tamaño	Valor mín. c/signo	Valor mín. s/signo	Valor máx. C/signo	Valor máx. S/signo
TINYINT	1	-128	0	127	255
SMALLINT	2	-32768	0	32767	65535
MEDIUMINT	3	-8388608	0	8388607	16777215
INT	4	-2147483648	0	2147483647	4294967295
BIGINT	8	-2^{63}	0	$2^{63} - 1$	$2^{64} - 1$

Tipos de datos y creación de tablas [6 | 43]

- **Numéricos con punto decimal fijo:**
  - Guardan valores numéricos exactos, por ejemplo, valores monetarios
  - Tipos: `DECIMAL`, `NUMERIC`
  - Se especifica la precisión (cantidad total de dígitos) y la escala (cantidad de dígitos para la parte decimal)
    - Ej: `salario DECIMAL(5, 2)`

- Para `DECIMAL` se puede usar `DEC` o `FIXED`.
- En el ejemplo mostrado, la columna `salario` podrá guardar desde -999.99 hasta 999.99.

Tipos de datos y creación de tablas [7 | 43]

- **Numéricos con punto decimal flotante:**
  - Guardan valores numéricos aproximados
  - Tipos: `FLOAT`, `DOUBLE`
  - También se especifica la precisión y la escala
  - MySQL redondea al guardar los valores
    - Ej: `salario FLOAT(7, 4)`
    - Valor que se guarda: 999.00009
    - Resultado que se obtiene: 999.0001

- Para `FLOAT` se usan 4 bytes y para `DOUBLE` 8 bytes.
- `DOUBLE` es un alias de MySQL para `DOUBLE PRECISION`. También se puede usar el alias `REAL`.

Tipos de datos y creación de tablas [8 | 43]

- **Numéricos para bits:**
  - Guardan bits
  - Tipo: `BIT`
  - Para especificar los valores, se puede usar la notación `b'valor'`:
    - Ej: `b'1'` y `b'0'`

Tabla con tipos de datos numéricos

Tipos de datos y creación de tablas [9 | 43]

- **Atributos para los tipos numéricos:**
  - **Ancho de visualización**: MySQL soporta una extensión para especificar, opcionalmente, el ancho con el cual se muestran los tipos enteros
    - Por ejemplo, `INT(4)` especifica que un entero se muestre con 4 dígitos
    - El ancho de visualización no restringe el rango de valores que se pueden guardar ni impide que valores más grandes que el mismo se muestren correctamente

- Una columna `SMALLINT(3)` tiene el rango de -32768 a 32767. Aquellos valores fuera del rango de visualización de 3 dígitos se muestran en forma completa.

- Si no se especifica un valor para el ancho de visualización, se toma 10 como valor predeterminado.

Tipos de datos y creación de tablas [10 | 43]

- **Atributos para los tipos numéricos (continuación):**
  - **`ZEROFILL`**: cuando se especifica un ancho de visualización junto con el atributo opcional (no estándar) `ZEROFILL`, el relleno por defecto de espacios se reemplaza con ceros
    - Por ejemplo, una columna `INT(4) ZEROFILL`, en la cual se guarda el número 5, se la muestra como 0005.

- El atributo `ZEROFILL` se ignora cuando la columna aparece en expresiones o consultas del tipo `UNION`.

- Si en una columna entera, con el atributo `ZEROFILL`, se guardan valores más grandes que el ancho de visualización, pueden aparecer problemas cuando MySQL genera tablas temporales en algunos *joins* complejos.

Tipos de datos y creación de tablas [11 | 43]

- **Atributos para los tipos numéricos (continuación):**
  - **`UNSIGNED`**: todos los tipos enteros pueden llevar este atributo opcional (no estándar), el cual permite sólo números positivos en una columna
    - Se puede usar cuando se necesite un rango numérico más grande para la columna: una columna `INT UNSIGNED`, sigue teniendo el mismo rango de valores, pero los límites se mueven de -2147483648 y 2147483647 a 0 y 4294967295

Atributos para los tipos numéricos

- Los tipos con punto decimal fijo y flotante también pueden ser `UNSIGNED`. Al igual que con los tipos enteros, se impide guardar números negativos, pero el rango superior se mantiene igual.

- Si se especifica el atributo `ZEROFILL`, MySQL automáticamente agrega el atributo `UNSIGNED`.

- Si se intenta guardar un valor negativo en una columna que no es `UNSIGNED`, se genera un error.

Tipos de datos y creación de tablas [12 | 43]

- **Atributos para los tipos numéricos (continuación):**
  - **`AUTO_INCREMENT`**: válido para los tipos enteros y con punto decimal flotante
    - La primera fila que se inserta toma el valor 1 en esta columna
    - La segunda fila toma el valor 2 y así sucesivamente
    - Cada tabla admite una sola columna con esta propiedad, y su valor no se especifica
    - Si se especifica un valor para esta columna, se inserta, y la siguiente fila toma el valor siguiente

- Si se inserta un `NULL` en una columna indexada con el atributo `AUTO_INCREMENT` se inserta el siguiente valor en la secuencia. Cuando se inserta cualquier otro valor, la columna toma ese valor y la secuencia se reinicializa para que el próximo valor siga a continuación de este último.

- Guardar un 0 en una columna `AUTO_INCREMENT` tiene el mismo efecto que guardar un `NULL`, a menos que esté habilitado el modo `NO_AUTO_VALUE_ON_ZERO`.

- Para determinar el valor más reciente se puede usar `LAST_INSERT_ID()`.

Tipos de datos y creación de tablas [13 | 43]

- **Atributos para los tipos numéricos (continuación):**
  - **`AUTO_INCREMENT`**: válido para los tipos enteros y con punto decimal flotante
    - MySQL exige que la columna con esta propiedad sea PK
    - No permite valores negativos
    - Requiere que la columna sea `NOT NULL`
    - No soporta restricciones `CHECK`

Atributo AUTO_INCREMENT

- Las restricciones tipo `CHECK` no pueden referenciar columnas `AUTO_INCREMENT`, ni se puede agregar este atributo a columnas existentes que se usen en restricciones `CHECK`.

Tipos de datos y creación de tablas [14 | 43]

- **Tipos de datos para fechas y horas:**
  - Tipos: `DATE`, `DATETIME`, `TIMESTAMP`, `TIME`, `YEAR`
  - El tipo `DATE` es para valores con sólo la fecha
  - Se recupera y visualiza en el formato 'AAAA-MM-DD' (por ejemplo: '`2020-02-21`')
  - Rango: '`1000-01-01`' a '`9999-12-31`'

- Se recomienda emplear 4 dígitos para los años para evitar problemas.

Tipos de datos y creación de tablas [15 | 43]

- **Tipos de datos para fechas y horas:**
  - El tipo `DATETIME` es para valores con fecha y hora
  - Se recupera y visualiza en el formato 'AAAA-MM-DD HH:MM:SS' (por ejemplo: '`2020-02-21 16:54:23`')
  - Rango: `'1000-01-01 00:00:00'` a `'9999-12-31 23:59:59'`

Tipos de datos y creación de tablas [16 | 43]

- **Tipos de datos para fechas y horas:**
  - El tipo `TIMESTAMP` es para valores con fecha y hora
  - Se recupera y visualiza en el formato 'AAAA-MM-DD HH:MM:SS' (por ejemplo: '`2020-02-21 16:54:23`')
  - Rango: `'1970-01-01 00:00:01'` UTC a `'2038-01-19 03:14:07'` UTC

Tipos de datos y creación de tablas [17 | 43]

- **Sobre los tipos `DATE`, `DATETIME` y `TIMESTAMP`:**
  - Las fechas siempre se expresan en el orden año-mes-día (por ejemplo: `'2020-02-21'`)
  - Se recomienda emplear 4 dígitos para los años para evitar problemas

Tipos de datos y creación de tablas [18 | 43]

- **Sobre los tipos `DATE`, `DATETIME` y `TIMESTAMP`: (continuación):**
  - MySQL permite guardar fechas donde el día, o el día y el mes, sean 0 (Ej: `'2020-01-00'`, `'2020-00-00'`)
  - Para evitar guardar `NULL` en una fecha, MySQL permite el valor `'0000-00-00'` (a este valor se lo conoce como "valor cero")

- Que se pueda guardar un cero para el día, o día y mes, resulta útil si por ejemplo se tienen que guardar fechas de nacimiento en las cuales no se conoce exactamente la fecha. En estos casos, funciones como `DATE_SUB()` y `DATE_ADD()`, las cuales requieren fechas completas, no darán resultados correctos. Para evitar que se puedan guardar estos valores, se debe habilitar el modo `NO_ZERO_IN_DATE`.

- Para evitar el valor `'0000-00-00'`, se debe habilitar el modo `NO_ZERO_DATE`. Este valor, al usarse a través de un conector ODBC se convierte a `NULL` ya que ODBC no lo permite. Este valor se puede emplear en cualquiera de los tipos para fechas y horas (por ejemplo: `'0000-00-00 00:00:00'`). También se pueden usar `'0'` o `0`.

Tipos de datos y creación de tablas [19 | 43]

- **Sobre los tipos `DATE`, `DATETIME` y `TIMESTAMP`: (continuación):**
  - Los tipos `DATETIME` y `TIMESTAMP` pueden llevar hasta 6 dígitos, los cuales representan microsegundos
  - Formato: '`YYYY-MM-DD HH:MM:SS[.fracción]`'
  - Al guardar valores `TIMESTAMP`, MySQL los convierte de la zona horaria actual a UTC, y al recuperarlos de UTC a la zona horaria actual

- Ejemplo: `col DATETIME(3)`. En este caso, `col` tiene hasta 3 dígitos para los microsegundos.

- Rango para `DATETIME` con microsegundos: '1000-01-01 00:00:00.000000' a '9999-12-31 23:59:59.999999'.

- Rango para `TIMESTAMP` con microsegundos: '1970-01-01 00:00:01.000000'' a '2038-01-19 03:14:07.999999'.

- Por defecto, la zona horaria actual para una conexión, es la hora del servidor (se puede configurar para cada conexión). Siempre que se mantenga constante la zona horaria, al recuperar un valor se obtiene el mismo que se guardó (no es el caso cuando hay un cambio de zona horaria). El valor de la zona horaria está disponible a través de la variable de sistema `time_zone`.

Tipos de datos y creación de tablas [20 | 43]

- **Sobre los tipos `DATE`, `DATETIME` y `TIMESTAMP`: (continuación):**
  - Valores inválidos para `DATE`, `DATETIME` o `TIMESTAMP` se convierten al valor cero del tipo apropiado (`'0000-00-00'` o `'0000-00-00 00:00:00'`)
  - Si se establece el modo `ALLOW_INVALID_DATES`, MySQL permite guardar fechas inválidas (por ejemplo: `'2020-11-31'`)

- MySQL soporta cualquier caracter como delimitador para la parte de la fecha u hora. Por ejemplo, el valor `'10:11:12'` aparenta ser una hora, pero si se lo usa para el tipo `DATE` se lo interpreta como `'2010-11-12'`. El valor `'10:45:15'` se convierte a `'0000-00-00'` ya que 45 no es un mes válido.

- La excepción es para los microsegundos (sólo se admite el punto).

- El modo `ALLOW_INVALID_DATES` resulta útil cuando se quiere permitir guardar fechas erróneas, las cuales pueden haber sido especificadas por un usuario (por ejemplo a través de un formulario web). En este modo, MySQL sólo verifica que los meses estén entre 1 y 12, y los días entre 1 y 31. Si el modo es deshabilitado, un valor inválido se convierte al valor cero.

Tipos de datos y creación de tablas [21 | 43]

- **Sobre los tipos `DATE`, `DATETIME` y `TIMESTAMP`: (continuación):**
  - Cualquier columna `DATETIME` y `TIMESTAMP` puede:
    - Tomar el valor por defecto de la fecha y hora actual (`DEFAULT CURRENT_TIMESTAMP`)
    - Tomar el valor de auto actualización (`ON UPDATE CURRENT_TIMESTAMP`)
    - Ambos

- Una columna que se auto inicializa toma la fecha y hora actual cuando no se le especifica un valor.

- Una columna que se auto actualiza toma la fecha y hora actual cuando se modifica cualquier otra columna de la fila (para evitar esto, se le debe especificar explícitamente el valor deseado).

- Para especificar estas propiedades automáticas se usan las cláusulas señaladas en la definición de la columna (no importa el orden).

- Si está deshabilitada la variable de sistema `explicit_defaults_for_timestamp`, una columna `TIMESTAMP` (no `DATETIME`) se puede inicializar o actualizar con la fecha y hora actual si se le asigna el valor `NULL` (a menos que la columna admita valores `NULL`).

Tipos de datos y creación de tablas [22 | 43]

- **Sobre los tipos `DATE`, `DATETIME` y `TIMESTAMP`: (continuación):**
  - En lugar de `CURRENT_TIMESTAMP` también se puede usar:
    - `CURRENT_TIMESTAMP()`
    - `NOW()`
    - `LOCALTIME`
    - `LOCALTIME()`
    - `LOCALTIMESTAMP`
    - `LOCALTIMESTAMP()`

- Una columna que se auto inicializa toma la fecha y hora actual cuando no se le especifica un valor.

- Una columna que se auto actualiza toma la fecha y hora actual cuando se modifica cualquier otra columna de la fila (para evitar esto, se le debe especificar explícitamente el valor deseado).

- Para especificar estas propiedades automáticas se usan las cláusulas señaladas en la definición de la columna (no importa el orden).

- Si está deshabilitada la variable de sistema `explicit_defaults_for_timestamp`, una columna `TIMESTAMP` (no `DATETIME`) se puede inicializar o actualizar con la fecha y hora actual si se le asigna el valor `NULL` (a menos que la columna admita valores `NULL`).

Tipos de datos y creación de tablas [23 | 43]

- **Sobre los tipos `DATE`, `DATETIME` y `TIMESTAMP`: (continuación):**
  - También se puede usar la cláusula `DEFAULT` para especificar un valor predeterminado constante (no automático):
    - Ejemplo: `DEFAULT 0` o `DEFAULT '2020-02-23 00:00:00'`

Tipos de datos y creación de tablas [24 | 43]

- **Tipos de datos para fechas y horas:**
  - El tipo `TIME` es para valores con sólo la hora
  - Se recupera y visualiza en el formato 'HH:MM:SS' (o 'HHH:MM:SS' para valores con horas grandes)
  - Rango: `'-838:59:59'` a `'838:59:59'`
  - Puede incluir una parte fraccional para representar microsegundos

- El tipo `TIME` no sólo se puede usar para representar la hora del día, sino también el tiempo transcurrido entre 2 eventos, el cual puede ser mucho más grande que 24 horas, o incluso negativo.

- Se puede emplear una forma abreviada para asignar un valor al tipo `TIME`, pero se debe tener cuidado. Por ejemplo, el valor `'11:12'`, el cual sería interpretado como `'11:12:00'` por una persona, MySQL lo iterpreta como `'00:11:12'`.

- Ejemplo: `col TIME(6)`. En este caso, `col` tiene hasta 6 dígitos para los microsegundos.

Tipos de datos y creación de tablas [25 | 43]

- **Tipos de datos para fechas y horas:**
  - El tipo `YEAR` es para representar años
  - Se recupera y visualiza en el formato 'AAAA'
  - Rango: `'1901'` a `'2155'` (o `'0000'`)

Tabla con tipos de datos para fechas

- Los años se pueden especificar como una cadena (`'2019'`) o como número (`2019`). Si se especifican 2 dígitos, MySQL hace una conversión.

Tipos de datos y creación de tablas [26 | 43]

- **Tipos de datos para cadenas:**
  - De largo fijo y variable: `CHAR`, `VARCHAR`, `BINARY`, `VARBINARY`
  - Para grandes objetos: `TEXT`, `BLOB`
  - Para selección de un elemento: `ENUM`
  - Para selección de varios elementos: `SET`

Tipos de datos y creación de tablas [27 | 43]

- **Cadenas de largo fijo y variable:**
  - El largo de una columna `CHAR` o `BINARY`, entre 0 y 255, se mantiene fijo al valor que se especifica en la creación (se completa con espacios en blanco a la derecha)
  - El largo de una columna `VARCHAR` o `VARBINARY`, entre 0 y 65535, varía según el valor actualmente guardado

- En realidad, el valor de 65535 es el valor máximo que puede tener una fila, por lo que el rango superior dependerá de las otras columnas de la tabla.

Tipos de datos y creación de tablas [28 | 43]

- **Cadenas de largo fijo y variable:**
  - `CHAR`, `VARCHAR` y `TEXT` comparan cadenas usando una cierta secuencia de ordenamiento (*collation*), y `BINARY`, `VARBINARY` y `BLOB` comparan bytes:
    - Al comparar 2 cadenas con una secuencia de ordenamiento, como casi todas son insensibles a mayúsculas/minúsculas, esta comparación es igual
    - Al comparar 2 cadenas sin emplear una secuencia de ordenamiento, la comparación se basa en los valores numéricos que contienen

Tabla con tipos de datos CHAR y BINARY

- Para poder ver el valor de una columna BINARY en Workbench, en la propiedades seleccionar “SQL Execution” y seleccionar la opción “Treat Binary/Varbinary as nonbinary character string”.

Tipos de datos y creación de tablas [29 | 43]

- **Cadenas para grandes objetos:**
  - `TEXT`: resulta útil para guardar cadenas grandes, que van desde 1 byte a 4 GB de tamaño
  - No se guarda en memoria, siempre se lee del disco (mucho más lento que `CHAR`/`VARCHAR`)
  - Tiene 4 tipos:
    - `TINYTEXT`: 255 caracteres
    - `TEXT`: 65535 caracteres (64 kB)
    - `MEDIUMTEXT`: 16777215 caracteres (16 MB)
    - `LONGTEXT`: 4294967295 caracteres (4 GB)

- Igual que con `CHAR`/`VARCHAR`, las comparaciones emplean una secuencia de ordenamiento (*collation*).

Tipos de datos y creación de tablas [30 | 43]

- **Cadenas para grandes objetos:**
  - `BLOB`: similar a `TEXT`, pero las comparaciones se basan en los valores numéricos que contienen, no en una *collation*
  - Tiene 4 tipos: `TINYBLOB`, `BLOB`, `MEDIUMBLOB` y `LONGBLOB`

Tipos de datos y creación de tablas [31 | 43]

- **Cadenas para selección de un elemento:**
  - El tipo `ENUM` es una cadena con un valor elegido a partir de una lista de valores permitidos
  - Las cadenas que se especifican como valores de entrada se codifican como números
  - En las consultas, los números se traducen a sus cadenas correspondientes

Tipos de datos y creación de tablas [32 | 43]

- **Cadenas para selección de un elemento (continuación):**
  - Los valores listados tienen una posición: 1, 2, 3, ...
  - Si una columna `ENUM` vale `NULL`, la posición es `NULL`
  - Si una columna `ENUM` vale `''`, la posición es 0
  - Un `ENUM` puede tener hasta 65535 elementos

Tabla con tipo de dato Enum

- En el ejemplo, si la columna `tamanio` se hubiera declarado como `VARCHAR`, insertar un millón de filas con el valor '`medium`' requeriría 6 millones de bytes de almacenamiento. Al declararse como `ENUM` se requieren 1 millón.

- Si se inserta un valor que no sea uno de la lista en una columna `ENUM`, se inserta una cadena vacía (se distingue de una cadena vacía "normal" por el hecho que su posición vale 0). Si está habilitado el modo estricto, insertar un valor inválido genera un error.

- Si una columna `ENUM` permite valores `NULL`, este valor es válido para la columna, y el valor predeterminado es `NULL`. Si no puede ser `NULL`, su valor predeterminado es el primero de la lista de valores.

Tipos de datos y creación de tablas [33 | 43]

- **Cadenas para selección de varios elementos:**
  - El tipo `SET` es un objeto cadena que puede tener cero o más valores, elegidos a partir de una lista
  - Por ejemplo, una columna definida como `SET('uno', 'dos') NOT NULL` puede tener como valores:
    - `''`
    - `'uno'`
    - `'dos'`
    - `'uno,dos'`

Tabla con tipo de dato Set

- Si en la definición del `SET` se especifican valores duplicados se genera una advertencia, o un error en caso que se encuentre habilitado el modo SQL estricto.

- Un `SET` puede tener hasta 64 elementos distintos.

Tipos de datos y creación de tablas [34 | 43]

- **Tipo de datos JSON:**
  - JSON (JavaScript Object Notation): formato texto para representar datos estructurados
  - Muy usado para el intercambio de datos
  - Es un subconjunto de JavaScript, totalmente independiente del mismo

- JSON nace como alternativa a XML, requiriendo mucho menos formato que este último. Una de sus mayores ventajas es que se puede usar con cualquier lenguaje de programación (Python, Java, PHP, C#, etc).

Tipos de datos y creación de tablas [35 | 43]

- **Sintaxis JSON**
  - Un objeto JSON puede tomar 2 formas:
    - Una colección desordenada de pares clave:valor entre `{ }`
    - Una lista ordenada de valores (vector) entre `[ ]`
  - Ejemplo:

```
{ }
{"user":"Sammy",  "online":true, "followers":987}
[ ]
[1, 2, true, false]
```


- En JSON no se permiten 2 claves con el mismo nombre.
- El primer caso representa una colección vacía de pares clave:valor.
- El segundo caso representa una colección con 2 pares clave:valor. Cada par se separa del otro mediante una coma.
- El tercer caso representa un vector vacío de valores.
- El cuarto caso representa un vector con 4 valores. Cada valor se separa del otro mediante una coma.

Tipos de datos y creación de tablas [36 | 43]

- **Sintaxis JSON**
  - Un valor puede ser:
    - Una cadena entre comillas dobles
    - Un número
    - `true`
    - `false`
    - `null`
    - Otro objeto JSON
    - Un vector
  - Estas estructuras se pueden anidar

Tipos de datos y creación de tablas [37 | 43]

Sintaxis JSON (objetos anidados):

```
{
   "sammy" : {
      "username"  : "SammyShark",
      "followers" : 987
   },
   "jesse" : {
      "username"  : "JesseOctopus",
      "followers" : 432
   }
}
```


- En el ejemplo se puede ver que ahora, el valor para la clave "`sammy`" en lugar de ser una cadena, un número, etc, es otro objeto JSON, el cual tiene su propio conjunto de pares clave:valor (en rojo aparece el primero de estos objetos).
-
- JSON es sensible a mayúsculas y minúsculas. Para los valores booleanos se toma `true` o `false`, y para el valor nulo `null`.

Tipos de datos y creación de tablas [38 | 43]

Sintaxis JSON (vectores anidados):

```
{
    "username" : "SammyShark",
    "followers" : 987,
    "websites" : [
        {
            "description" : "work",
            "URL" : "https://www.digit.com/"
        },
        {
            "desciption" : "tutorials",
            "URL" : "https://www.digit.com/tutorials"
        }
    ]
}
```


- En el ejemplo anterior, cada valor era un objeto JSON. Ahora, cada valor es un vector de objetos JSON.

Tipos de datos y creación de tablas [39 | 43]

Sintaxis JSON (atributos multivaluados)

```
{
    "username" : "SammyShark",
    "followers" : 987,
    "websites" : ["https://www.sitio1.com/",
                  "https://www.sitio2.com/",
                  "https://www.sitio3.com/",
                  "https://www.sitio4.com/"
                 ]
}
```


- En este ejemplo, el valor de la clave “websites” es un vector de valores.

Tipos de datos y creación de tablas [40 | 43]

Comparación XML vs. JSON:

```
<users>
 <user>
  <username>SammyShark</username> <followers>987</followers>
 </user>
 <user>
  <username>JesseOctopus</username> <followers>432</followers>
 </user>
</users>
```

Tipos de datos y creación de tablas [41 | 43]

Comparación XML vs. JSON (continuación):

```
{"users": [
 {"username" : "SammyShark", "followers" : 987},
 {"username" : "JesseOctopus", "followers" : 432}
]}
```

Tipos de datos y creación de tablas [42 | 43]

- **¿Por qué la necesidad del tipo JSON?**
  - Suponer una aplicación donde se deban guardar las preferencias de los usuarios
    - Opción 1: crear una tabla con las columnas `id`, `idUsuario`, `preferencia` y `valor`
    - Opción 2: crear una tabla con las columnas `idUsuario` y `preferencia` (del tipo `JSON`)

- La primera opción funciona bien cuando se tienen pocos usuarios. Si por ejemplo se tienen 1000 usuarios, y cada uno con 5 preferencias, se tendrá una tabla con 5000 filas (sólo para esta pequeña funcionalidad).

- Con la segunda opción, para los 1000 usuarios la tabla tendría 1000 filas.

Tipos de datos y creación de tablas [43 | 43]

- Borrado de tablas:

```
DROP TABLE [IF EXISTS] Socio;
```

- Agregado / borrado de columnas:

```
ALTER TABLE Socio
  ADD domicilio VARCHAR(30) null[;]

ALTER TABLE Socio
  DROP COLUMN foto[;]
```

Planificación de la capacidad [1 | 1]

- Al planificar una BD se debe tener en cuenta el espacio que ocupará en disco en forma de archivos:
  - Tablas (de usuario y sistema)
    - Para las tablas de usuario, se puede estimar para cada tabla la cantidad de filas y el tamaño de cada una
  - Tamaño del registro de transacciones
  - Índices (número y tamaño)

- También ocupan espacio otros objetos que se crean en la BD: procedimientos almacenados, desencadenadores, vistas, etc).

Resumen [1 | 1]

- Tipos de Bases de Datos
- Objetos de Bases de Datos
- Arquitecturas de diseño de aplicaciones
- Actividades para implementar una BD
- Almacenamiento y Transacciones
- Creación y modificación de BDs
- Creación de tipos datos y tablas
- Planificación de la capacidad

Otro material [1 | 2]

- **Registro de transacciones en MySQL**:
  - https://dev.mysql.com/doc/refman/8.0/en/innodb-redo-log.html
  - https://dba.stackexchange.com/questions/72904/difference-between-transaction-log-and-redo-log-in-mysql
- **Tipos de datos**
  - https://dev.mysql.com/doc/refman/8.0/en/data-types.html

Otro material [2 | 2]

- **JSON**
  - https://www.digitalocean.com/community/tutorials/an-introduction-to-json
  - https://ed.team/blog/como-trabajar-con-json-en-mysql
  - https://scotch.io/tutorials/working-with-json-in-mysql
  - http://www.mysqltutorial.org/mysql-json/