ALUMNO: Cuesta Alario David **DNI: 20857516-N**

MySQL – Comandos en Xampp

Trabajar con Xampp

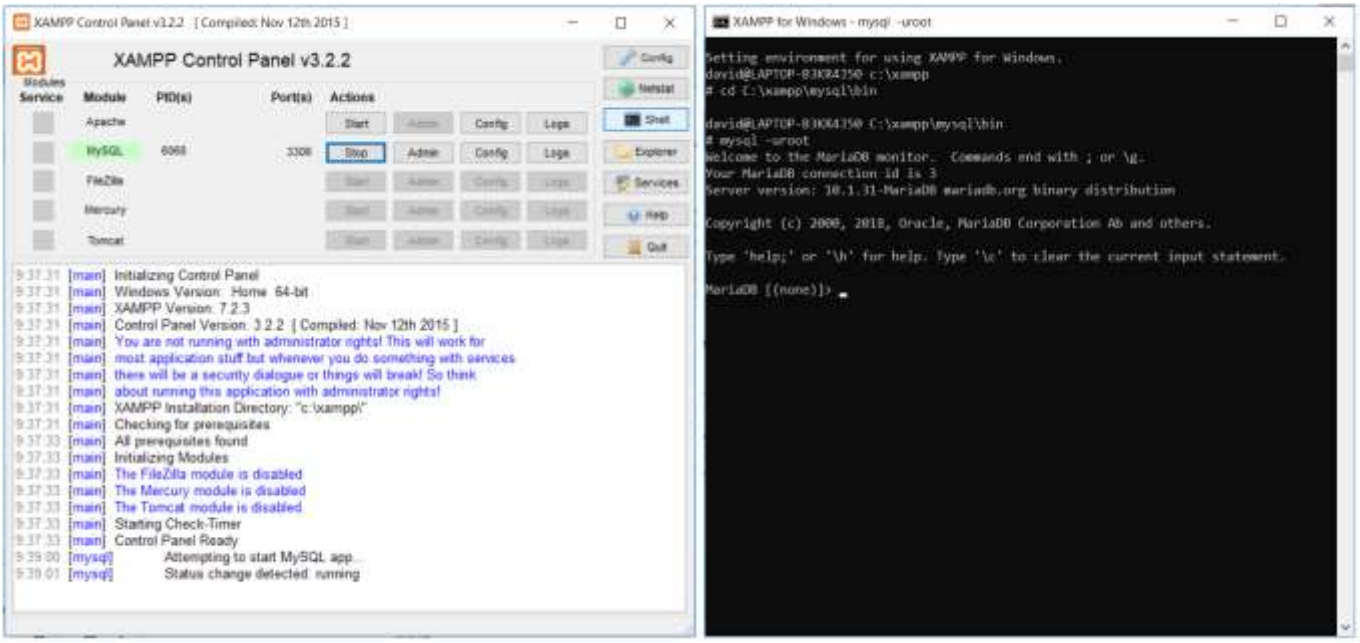
Una vez arrancado el servidor de MySQL con la acción Start se abre la ventana de comandos Shell y se introducen los siguientes comandos que permiten administrar las bases de datos como Root:

CD <dirección> **mysql –h**<host> **-u**<usuario> **-p**<contraseña>

- Donde la opción:
  - <dirección> representa el directorio en el que está instalado MySQL. Por defecto será → C:\xampp\mysql\bin
  - **–h**<host> corresponde al host (este campo no hace falta si estamos en la misma máquina que el servidor)
  - **-u**<usuario> indica el nombre del usuario que va a entrar
  - **-p**<contraseña> indica su contraseña

MySQL –uroot

Cuando se entra en MySQL, sale un mensaje de ayuda y el prompt de la línea de comandos cambia a **mysql >**



Para detener el servidor de MySQL desde el panel de XAMPP haremos clic en la acción stop del servidor MySQL

Cerrar la consola de MySQL:

mysql > QUIT;

Los siguientes comandos permiten acceder al manual de MySQL que incluye Ayuda para algunos comandos propios de:

mysql > HELP; los sitios locales y web

mysql > HELP CONTENTS; la sintaxis y funcionamiento de las instrucciones

Los siguientes comandos permiten moverse entre las diferentes tablas y bases de datos implementadas

mysql > STATUS;	Ver qué usuario somos y qué base de datos estamos usando
mysql > USE < Nombre Base Datos >**;**	Usar una base de datos existente
mysql > SHOW DATABASES;	Mostrar todas las bases de datos existentes
mysql > SHOW TABLES;	Mostrar las tablas de la base de datos actual
mysql > SHOW CREATE TABLE < Nombre Tabla >**;**	Mostrar la instrucción de creación de una tabla
mysql > DESCRIBE < Nombre Tabla >**;**	Mostrar los campos de una tabla

Grabar la sesión en un fichero

mysql> TEE < Nombre-Ruta Fichero.txt >

mysql> NOTEE (Si después ponemos simplemente TEE sigue añadiendo al fichero anterior)

Cambiar los permisos de root a permitido

Acedemos a la carpeta C:\xampp\apache\conf\extra

Abrimos el archivo httpd-xampp.conf

Y sustituimos el texto:

```
#
# New XAMPP security concept
#

Order deny,allow
Deny from all
Allow from ::1 127.0.0.0/8
```

→

```
#
# New XAMPP security concept
#
Order deny,allow
#Deny from all
Allow from all
```

Sintaxis de MySQL

Los comandos del estándar SQL siguen la sintaxis del estándar SQL92

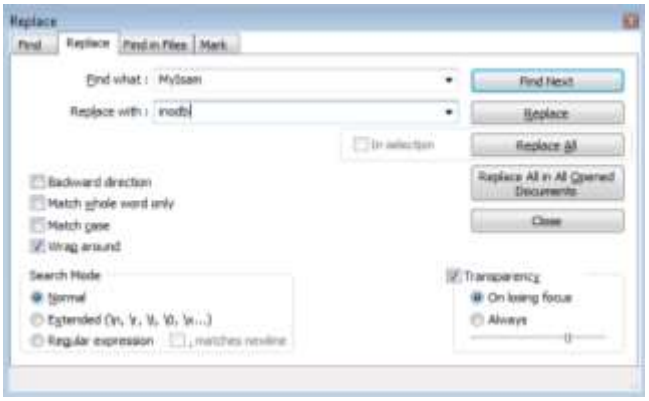
- Todas las instrucciones acaban con punto y coma ;
- A la hora de indicar la sintaxis de los distintos comandos de SQL se sigue la notación conocida como BNF:

SIMBOLO	SIGNIFICADO
MAYÚSCULAS	Palabra reservada de SQL
minúsculas	Nombre de un archivo, tabla, campo, etc…
Corchetes [ ]	El elemento que esta entre corchetes es opcional, apareciendo de 0 a 1 veces
Llaves { }	Se debe elegir una de las opciones que aparezcan entre llaves
Barra vertical |	OR lógico que separa las distintas opciones
Puntos suspensivos ...	Repetición de lo que aparezca justo antes de los puntos suspensivos

Comentarios en los comandos

- Dos guiones al principio de la línea: **mysql> --** Consulta 1
- También se admiten los comentarios al estilo de C y Java en cualquier parte de las instrucciones: /* .... */

Ejecutar scripts



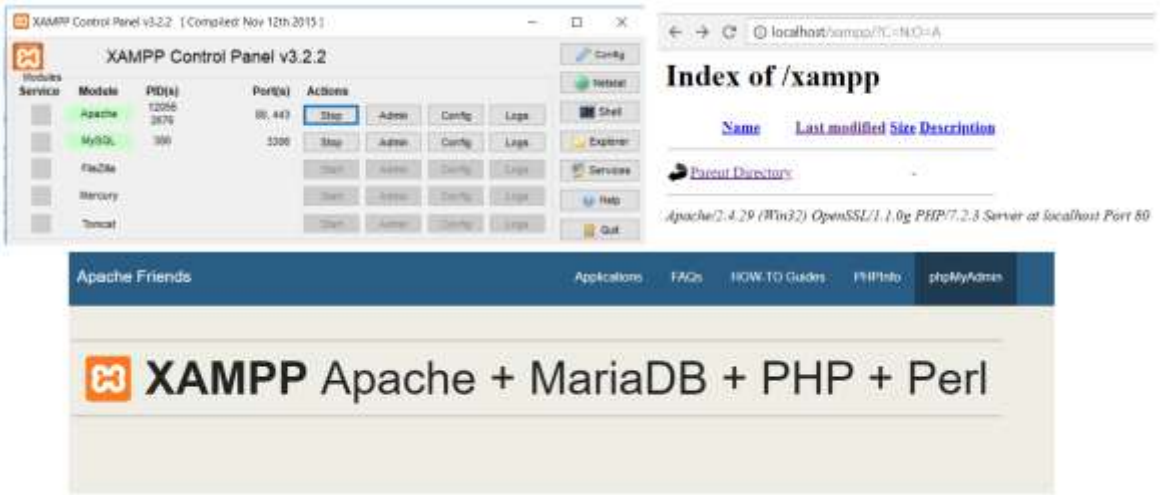
Los ficheros de script son ficheros de texto que contienen instrucciones en SQL. Para importarlos se debe sustituir las apariciones de la palabra MyIsam por inobd y ejecutar el siguiente comando:

mysql > SOURCE <directorio>****<nombre script.sql>

Crear una copia de seguridad

Se crea un script con todo lo referente a la BD – tablas y datos):

- **Opción 1:** descargar una copia
  Arrancar Apache desde XAMPP
  Poner en el navegador la URL: localhost/Xampp
  Parent Directory **-->** phpMyAdmin **-->** seleccionar la BD **-->** Exportar
- **Opción 2:** Desde el directorio de instalación de MySQL
  Desde la línea de comandos (fuera de mysql pero en el mismo directorio c:\xampp\mysql\bin) escribir:
  mysqldump –uroot < Nombre base de datos > **>** < Nombre Script.sql >**;**



Importar datos desde un fichero de texto

LOAD DATA INFILE < Nombre Fichero.txt >

INTO TABLE < Nombre Tabla > **(** < Nombre Campo 1 > **,** < Nombre Campo i > **);**

- El fichero de texto a importar ha de estar en el directorio de esa base de datos, por defecto:
  C:\dir_mysql\DATA\carpeta_con_el_nombre_de_la_BD\
- Si el fichero de texto a importar está en otro directorio, hemos de poner su ruta completa al estilo UNIX

Creación de una base de datos

CREATE DATABASE IF NOT EXISTS < Nombre Base Datos >

CREATE TABLE IF NOT EXISTS < Nombre Tabla > **(**

< Nombre Campo 1 > < Tipo Dato 1 > < Opciones 1 >, (Debe existir un mínimo de dos campos)

< Nombre Campo 2 > < Tipo Dato 2 > < Opciones 2 >,

< Nombre Campo N > < Tipo Dato N > < Opciones N >,

PRYMARY KEY < Campo Clave Primaria >

- Debe ser un campo de la tabla, Debe existir al menos una y ser única, pueden haber varias

UNIQUE < Campos Clave Alternativa >

- Deben ser campos de la tabla, Puede haber entre ninguna hasta varias
- Contendrá los campos que sean únicos pero no hayan sido escogidos como clave principal

FOREIGN KEY < Campos Clave Externa > **REFERENCES** < Nombre Tablas Referenciadas >

- Deben ser campos de la tabla, Puede haber entre ninguna hasta varias)
- Contendrá los campos de la tabla en el mismo orden que las tablas con las que se relacionen

ON UPDATE < Opciones Actualizar >

ON DELETE < Opciones Borrar > **);**

Tipos de datos:

Numéricos	**Enteros**	**INTEGER**	**INT**	
	Reales	**DECIMAL(i , j)**	**DEC(i , j)**	**NUMERIC(i , j)**
		i representa la cantidad total de dígitos enteros j representa la cantidad de dígitos decimales		
		DOUBLE PRECISION	**FLOAT**	**REAL**
Cadena de caracteres	**Longitud fija:**	**CHAR(n)**	**CHARACTER(n)**	
		n representa el número de caracteres de la cadena		
	Longitud variable	**VARCHAR(n)**	**CHARACTER VARYING(n)**	**CHAR VARYING(n)**
		n representa el número máximo de caracteres definidos		
Cadena de Bits	**Longitud fija:**	**BIT(n)**		
		n representa el número de caracteres de la cadena		
	Longitud variable	**BIT VARYING(n)**		
		n representa el número máximo de caracteres definidos		
Fecha Hora	**Fecha y hora**	**TIMEDATE**	YYYY-MM-DD HH:MM:SS	
	Sólo fecha	**DATE**	YYYY-MM-DD	
	Sólo hora	**TIME**	HH:MM:SS	

Un dominio permite crear un nuevo tipo de dato personalizado a partir de los que ya están definidos en el estándar SQL:
CREATE DOMAIN < Nombre nuevo Tipo > **AS** < Tipo Dato Referencia>

Opciones para los datos:

El estándar SQL permite especificar algunas opciones dentro de cada campo de la tabla.

DEFAULT permite inicializar automáticamente los valores del campo al valor por defecto para ese tipo de dato

NOT NULL permite indicar que un campo de la tabla nunca podrá estar vacío.
Esta restricción la han de cumplir obligatoriamente todos los campos que formen parte de la clave primaria.

Opciones para las restricciones de integridad:

El estándar SQL permite establecer como afectara al resto de la base de datos el eliminar o actualizar un campo

SET NULL dejar el campo vacío

SET DEFAULT dejar el valor por defecto en función del tipo de dato

CASCADE modifica todos los campos relacionados al modificarse la tabla con este parámetro

RESTRICT sólo se borra si no hay restricciones ni vistas que le hagan referencia.

Actualización de una base de datos

Permite cambiar el diseño de la tabla que se indique: tanto el diseño de sus atributos como de sus restricciones de integridad

ALTER TABLE < Nombre Base Datos > **.** < Nombre Tabla >

ALTER < Nombre Campo o Código Restricción > (permite cambiar algo que ya existe)

DROP < Código opción >**;** (Para eliminar una opción de un campo)

SET < Código opción >**;** (Para añadir una opción a un campo)

ADD < Nombre Campo o Código Restricción >**;** (permite añadir algo nuevo al diseño)

Después de ADD se pone lo mismo que se indicaría en la instrucción CREATE TABLE

DROP < Nombre Campo o Código Restricción >**;** (permite quitar algo del diseño)

Para borrar una tabla entera de una base de datos se emplea:

DROP TABLE < Nombre Tabla >

CASCADE modifica todos los campos relacionados al modificarse la tabla con este parámetro

RESTRICT sólo se borra si no hay restricciones ni vistas que le hagan referencia.

Para borrar una base de datos entera se emplea:

DROP DATABASE < Nombre Base Datos >

CASCADE modifica todos los campos relacionados al modificarse la tabla con este parámetro

RESTRICT sólo se borra si no hay restricciones ni vistas que le hagan referencia.

Definición de datos

Insertar nuevas filas de datos en los campos de una tabla

INSERT INTO < Nombre Tabla > **VALUES** < Valor Campo 1 >**,** < Valor Campo 2 >**,** < Valor Campo N >**;**

- Los campos deben estar en el mismo orden en el que se especificaron los atributos al crear la tabla
- Las cadenas de caracteres van entre comillas simples

INSERT INTO < Nombre Tabla > **(** < Nombre Campo i > **)** **VALUES** < Valor Campo i >**;**

- Permite añadir solo los campos indicados
- El orden de los campos de las instancias INSERT y VALUES debe coincidir
- Los campos con la restricción NOT NULL no pueden omitirse
- Las cadenas de caracteres van entre comillas simples

Modificar datos de una tabla

Sustituirá el valor indicado de los campos descritos por la cláusula SET de aquellos campos que cumplan las condiciones indicadas por la cláusula WHERE

UPDATE < Nombre Tabla >

SET < Nombre Campo i > **=** < Nuevo Valor Campo i >

- Permite seleccionar los campos que se deben sustituir, así como su nuevo valor
- Admite múltiples cambios simultáneos separados por comas
- Otra posibilidad es efectuar una operación sobre un determinado campo

  SET < Nombre Campo i > **=** < Nombre Campo i > < Operación a realizar >

WHERE < Nombre Campo > <OPERADOR>**;**

Borrar filas enteras de la tabla

DELETE FROM < Nombre Tabla > **WHERE** < Nombre Campo i > { = != , > , < } < Valor Campo i >**;**

- Sin la cláusula WHERE se borraran todas las tuplas de la tabla

Consultas de datos

CREATE VIEW < Nombre Vista > **AS SELECT** <SUBCLAUSULA> < Nombre Campo 1 > **,** < Nombre Campo i >

- Permite seleccionar los campos los datos que se quieren recuperar separados por comas
- Los resultados se mostraran como una tabla en el mismo orden
- Admite cualquier tipo de atributos, fórmulas, funciones agregadas, concatenación de cadenas de caracteres con "||"
- Tiene algunas subclausulas opcionales:

  DISTINCT permite que se eliminen los datos duplicados

FROM < Nombre Tabla 1 >

- Permite indicar las tablas de las que se deben obtener los campos seleccionados
- Las tuplas con valores nulos en el atributo de reunión no se incluyen en el resultado, salvo si es una reunión externa
- Se puede anidar más de una tabla mediante paréntesis sucesivos empleando las cláusulas de reunión

  FROM < Nombre Tabla 1 > **INNER JOIN** < Nombre Tabla 2 > **ON** < Nombre Campo 1 > **=** < Nombre Campo 2 >
  - Las reuniones internas permiten anidar dos tablas cuando tienen un campo común con un nombre distinto
  - Se ha de indicar cuales son los campos relacionados

  FROM < Nombre Tabla 1 > **NATURAL JOIN** < Nombre Tabla 2 >
  - Las reuniones naturales permiten anidar dos tablas cuando tienen un campo común con el mismo nombre
  - Se asume que los campos con el mismo nombre son los que están relacionados

  FROM <Nombre Tabla 1> <OPERADOR> **OUTER JOIN** <Nombre Tabla 2> **ON** < Nombre Campo 1 > **=** < Nombre Campo 2 >
  - Las reuniones externas permiten representar los campos de las tablas aunque no cumplan la cláusula de reunión
  - Aquellos campos que no se puedan emparejar aparecerán representados con el valor NULL
  - Podemos encontrar tres tipos de reuniones externas identificadas con los operadores **LEFT** , **RIGHT** y **FULL**

WHERE < Nombre Campo > <OPERADOR>

- Permite seleccionar de entre los campos indicados únicamente aquellos datos que cumplan las condiciones descritas
- Admite los siguientes operadores:
  - De comparación { **=** **!=** , **>** , **<** } < Valor Campo >
  - Lógicos **AND** , **OR** o **NOT** para separar varias condiciones
  - Comparación de subcadenas con **LIKE**
    - **_** sustituye a un carácter arbitrario
    - **%** a un número indeterminado de caracteres arbitrarios
    - Las cadenas de caracteres van entre comillas simples
  - Conjuntos de valores

    IN (< Rango de Valores o clausula SELECT >**)**

    NOT IN (<Rango de Valores o clausula SELECT >**)**

    BETWEEN <Numero entero> **AND** < Numero entero >
  - las comparaciones de igualdad no funciona sobre un campo vacío por lo que se emplean los operadores:

    IS NULL y **IS NOT NULL**
- La omisión de WHERE indica una selección de tuplas incondicional en la que se cogen todas las filas

GROUP BY < Nombre Campo para agrupación de las funciones agregadas >

- En esta cláusula deben aparecer todos los atributos que aparezcan en SELECT
- En esta cláusula pueden aparecer otros atributos de la tabla que no aparezcan en SELECT

HAVING < condiciones sobre resultados asociados a GROUP BY >

ORDER BY < Nombre Campo > <SUBCLAUSULA>

- Permite ordenar los datos en función del campo especificado de manera **ASC** Ascendente o **DESC** Descendente
- Admite varios campos separados por comas

Funciones agregadas

COUNT (<SUBCLAUSULA> < Nombre Campo >**)** Cuenta el número filas no vacías (aunque se repitan)

- Permite poner ***** como campo para contar todas las filas de la tabla
- Admite de forma opcional la subclausula: **DISTINCT** para contar solo los valores distintos

SUM (< Nombre Campo >**)** Devuelve la suma de los datos de esa columna

AVG (< Nombre Campo >**)** Devuelve la media de esa columna

MAX (< Nombre Campo >**)** Devuelve el mayor valor de esa columna

MIN (< Nombre Campo >**)** Devuelve el menor valor de esa columna

Eliminar consultas

DROP VIEW < Nombre Vista >**;**

Vistas

Actualizar vistas

CREATE OR REPLACE VIEW < Nombre Vista > **AS**
SELECT <SUBCLAUSULA> < Nombre Campo 1 > **,** < Nombre Campo i >
FROM < Nombre Tabla 1 >
WHERE < Nombre Campo > <OPERADOR>**;**

Eliminar vistas

DROP VIEW < Nombre Vista >**;**

Procedimientos Almacenados

CREATE PROCEDURE < Nombre procedimiento > **BEGUIN**
<PROCEDIMIENTO 1>**;**
<PROCEDIMIENTO 2>**;**
<PROCEDIMIENTO i>**;**
END

Los procedimientos son otras sentencias SQL

También admite el uso de:

- **IF** para establecer condiciones
- **FOR** para crear bucles

Índices

Un índice es un puntero a una fila de una determinada tabla de nuestra base de datos. Pero… ¿qué significa exactamente un puntero? Pues bien, un puntero no es más que una referencia que asocia el valor de una determinada columna (o el conjunto de valores de una serie de columnas) con las filas que contienen ese valor (o valores) en las columnas que componen el puntero.

Los índices mejoran el tiempo de recuperación de los datos en las consultas realizadas contra nuestra base de datos. Pero los índices no son todo ventajas, la creación de índices implica un aumento en el tiempo de ejecución sobre aquellas consultas de inserción, actualización y eliminación realizadas sobre los datos afectados por el índice (ya que tendrán que actualizarlo). Del mismo modo, los índices necesitan un espacio para almacenarse, por lo que también tienen un coste adicional en forma de espacio en disco.

La construcción de los índices es el primer paso para realizar optimizaciones en las consultas realizadas contra nuestra base de datos. Por ello, es importante conocer bien su funcionamiento y los efectos colaterales que pueden producir.

MySQL emplea los índices para encontrar las filas que contienen los valores específicos de las columnas empleadas en la consulta de una forma más rápida. Si no existiesen índices, MySQL empezaría buscando por la primera fila de la tabla hasta la última buscando aquellas filas que cumplen los valores establecidos para las columnas empleadas en la consulta. Esto implica que, cuanto más filas tenga la tabla, más tiempo tardará en realizar la consulta. En cambio, si la tabla contiene índices en las columnas empleadas en la consulta, MySQL tendría una referencia directa hacia los datos sin necesidad de recorrer secuencialmente todos ellos.

En general, MySQL emplea los índices para las siguientes acciones:

- Encontrar las filas que cumplen la condición WHERE de la consulta cuyas columnas estén indexadas.
- Para recuperar las filas de otras tablas cuando se emplean operaciones de tipo JOIN. Para ello, es importante que los índices sean del mismo tipo y tamaño ya que aumentará la eficiencia de la búsqueda. Por ejemplo: una operación de tipo JOIN sobre dos columnas que tengan un índice del tipo INT(10).
- Disminuir el tiempo de ejecución de las consultas con ordenación (ORDER BY) o agrupamiento (GROUP BY) si todas las columnas presentes en los criterios forman parte de un índice.
- Si la consulta emplea una condición simple cuya columna de la condición está indexada, las filas serán recuperadas directamente a partir del índice, sin pasar a consular la tabla.

A continuación, vamos a analizar los distintos tipos de índices que se pueden crear y las condiciones que deben cumplir cada uno de ellos:

- INDEX (NON-UNIQUE): este tipo de índice se refiere a un índice normal, no único. Esto implica que admite valores duplicados para la columna (o columnas) que componen el índice. No aplica ninguna restricción especial a los datos de la columna (o columnas) que componen el índice sino que se emplea simplemente para mejorar el tiempo de ejecución de las consultas.
- UNIQUE: este tipo de índice se refiere a un índice en el que todas las columnas deben tener un valor único. Esto implica que no admite valores duplicados para la columna (o columnas) que componen el índice. Aplica la restricción de que los datos de la columna (o columnas) deben tener un valor único.
- PRIMARY: este tipo de índice se refiere a un índice en el que todas las columnas deben tener un valor único (al igual que en el caso del índice UNIQUE) pero con la limitación de que sólo puede existir un índice PRIMARY en cada una de las tablas. Aplica la restricción de que los datos de la columna (o columnas) deben tener un valor único.
- FULLTEXT: estos índices se emplean para realizar búsquedas sobre texto (CHAR, VARCHAR y TEXT). Estos índices se componen por todas las palabras que están contenidas en la columna (o columnas) que contienen el índice. No aplica ninguna restricción especial a los datos de la columna (o columnas) que componen el índice sino que se emplea simplemente para mejorar el tiempo de ejecución de las consultas. Este tipo de índices sólo están soportados por InnoDB y MyISAM en MySQL 5.7.
- SPATIAL: estos índices se emplean para realizar búsquedas sobre datos que componen formas geométricas representadas en el espacio. Este tipo de índices sólo están soportados por InnoDB y MyISAM en MySQL 5.7.

Es importante destacar que todos estos índices pueden construirse empleando una o más columnas. Del mismo modo, el orden de las columnas que se especifique al construir el orden es relevante para todos los índices menos para el FULLTEXT (ya que este índice mira en TODAS las columnas que componen el índice).

Una vez hemos visto los tipos de índices, vamos a ver los distintos tipos de estructuras que se pueden crear para almacenar los índices junto con las características de cada uno de ellas:

- B-TREE: este tipo de índice se usa para comparaciones del tipo =, >, <, >=, <=, BETWEEN y LIKE (siempre y cuando se utilice sobre constantes que no empiecen por %). Para realizar búsquedas empleando este tipo de índice, se empleará cualquier columna (o conjunto de columnas) que formen el prefijo del índice. Por ejemplo: si un índice está formado por las columnas [A, B, C], se podrán realizar búsquedas sobre: [A], [A, B] o [A, B, C].
- HASH: este tipo de índice sólo se usa para comparaciones del tipo = o <=>. Para este tipo de operaciones son muy rápidos en comparación a otro tipo de estructura. Para realizar búsquedas empleando este tipo de índice, se emplarán todas las columnas que componen el índice.

Un índice puede ser almacenado en cualquier tipo de estructura pero, en función del uso que se le vaya a dar, puede interesar crear el índice en un tipo determinado de estructura o en otro. Por norma general, un índice siempre se creará con la estructura de B-TREE, ya que es la estructura más empleada por la mayoría de operaciones.

Como hemos visto en los apartados anteriores, los índices permiten optimizar las consultas sobre los elementos de la base de datos. Esto desemboca en una reducción considerable del tiempo de ejecución de la consulta. Esta reducción de tiempo es visible sobre tablas de gran tamaño. Sobre tablas pequeñas, el uso de índices no aporta una disminución drástica del tiempo de ejecución de la consulta ya que, prácticamente, MySQL tarda lo mismo en acceder al índice que en acceder de forma secuencial al contenido de la tabla para buscar la fila deseada.

No todas las soluciones son perfectas y tampoco lo iba a ser la creación de índices. La creación de índices también tiene efectos negativos. Estos efectos negativos es bueno conocerlos ya que pueden ocasionar efectos colaterales no deseados.

Uno de ellos es que las operaciones de inserción, actualización y eliminación que se realicen sobre tablas que tengan algún tipo de índice (o índices), verán aumentado su tiempo de ejecución. Esto es debido a que, después de cada una de estas operaciones, es necesario actualizar el índice (o los índices) presentes en la tabla sobre la que se ha realizado alguna de las operaciones anteriores.

Otro efecto negativo es que los índices deben ser almacenados en algún lugar. Para ello, se empleará espacio de disco. Por ello, el uso de índices aumenta el tamaño de la base de datos en mayor o menor medida.

Para crear un índice, se empleará la siguiente estructura:

CREATE <TIPO DE INDICE > **INDEX** < Nombre índice >
ON < Nombre Tabla 1 > **(** < Nombre columna índice > **)**
USING <METODO DE INDEXACION >

- Podemos encontrar los siguientes tipos de índice: UNIQUE | FULLTEXT | SPATIAL
- es el nombre de la tabla donde se va a crear el índice.
- El nombre de la columna (o columnas) que formarán el índice.
- Podemos encontrar dos tipos de indexación [BTREE | HASH].

Usuarios

CREATE USER < Nombre usuario > **IDENTIFIED BY** < Contraseña > ;

Ver todos los usuarios del sistema

SELECT USER , **HOST** authentication_string **FROM FROM** mysql. user ;

Otros conceptos

Calificación de atributos

Cuando coinciden los nombres de los atributos en varias tablas ha de indicarse de que tabla proviene cada campo mediante:

< Nombre Tabla > **.** < Nombre Campo >

El * selecciona todos los atributos de las relaciones

La cláusula **AS** que permite asociar dos campos consiguiendo:

- Hacer varias referencias a una misma relación mediante la asignación de un seudónimo
- Cambiar la representación de un campo en la tabla de resultados

```
SELECT E.NOMBRE AS NOMBRE_EMPLEADO,
       E.APELLIDO AS APEL_EMPLEADO,
       S.NOMBRE AS NOMBRE_SUPERVISOR,
       S.APELLIDO AS APEL_SUPERVISOR
FROM EMPLEADO AS E, EMPLEADO AS S
WHERE E.NSS SUPERV = S.NSS;
```

Funciones y Procedimientos

MySQL permite almacenan rutinas y programas del usuario para la base de datos en las que además de las instrucciones SQL, se pueden declarar variables, instrucciones de control, etc…

Funcionan de forma similar a cualquier lenguaje de programación.

Sintaxis de creación: **CREATE** { **FUNCTION** | **PROCEDURE** } < Nombre > **(**< Argumentos >**)** < Cuerpo >

Sintaxis de Llamada: **CALL** < Nombre > **(**< Argumentos >**)**

Triggers

Son procedimientos que se ejecuta automáticamente cuando se intentan insertar, modificar o borrar datos de una tabla.

Sintaxis de creación: **CREATE TRIGGER**

Sintaxis de borrado: **DROP TRIGGER**

Funciones incorporadas

Son funciones básicas adicionales que se ponen a disposición del usuario:

- Funciones matemáticas
- Funciones de fecha/hora
- Funciones de conversión entre tipos de datos
- Funciones avanzadas

Se pueden utilizar tanto en las instrucciones SQL como dentro del entorno de programación

Funciones sobre la fecha y hora

CURRENT_DATE devuelve la fecha actual en formato YYYY-MM-DD

CURRENT_TIME devuelve la hora actual en formato HH:MM:SS

NOW() devuelve la fecha y hora actual en formato YYYY-MM-DD HH:MM:SS

Extracción de partes de una fecha o hora:

YEAR (< Fecha >**)** devuelve el año de la fecha mediante un número entero

MONTH (< Fecha >**)** devuelve el mes de la fecha mediante un número entero

DAY (< Fecha >**)** devuelve el día de la fecha mediante un número entero

WEEKDAY (< Fecha >**)** devuelve el día de la semana de la fecha mediante un número entero del 1 al 7

DAYNAME (< Fecha >**)** devuelve el día de la semana de la fecha en texto

HOUR (< Hora >**)**

MINUTE (< Hora >**)**

SECOND (< Hora >**)**

Operaciones con fechas y horas:

DATE_ADD (< Fecha >**, INTERVAL (**< incremento > < tipo >**)**

- El incremento puede ser + para añadir ó – para quitar
- El tipo puede ser YEAR, MONTH, DAY, HOUR, MINUTE ó SECOND.

DATEDIFF (< Fecha más Moderna >**,** < Fecha más Antigua >**)**

- Días transcurridos entre dos fechas

Sistemas de seguridad para MySQL

La seguridad informática es la disciplina que se encarga de proteger la integridad y privacidad de la información almacenada en un sistema informático. Debe tenerse en cuenta que no existe ninguna técnica que permita asegurar la inviolabilidad de un sistema, por lo que para gestionar la seguridad de un sistema informático:

- Debe asegurarse de que los bienes a proteger:
  - Son utilizados como se debe
  - Sólo son accedidos por quien tiene permiso para ello
  - Cumplen la legislación vigente
- Para ello se establecen los siguientes objetivos:
  - Detectar los riesgos y amenazas para evitar que se produzcan o minimizar su efecto
  - Garantizar el uso adecuado de los bienes
  - Limitar las posibles pérdidas y asegurar la recuperación del sistema lo antes posible
  - Cumplir la legislación correspondiente
- Teniendo en cuenta:
  - El valor de los bienes que se van a proteger
  - La probabilidad con la que se puede ver expuesto a determinadas amenazas y el riesgo que ello supone.

Principios de seguridad CIDAN

Para alcanzar los objetivos es necesario contemplar una serie de servicios o principios de seguridad de la información

- **Confidencialidad:** Se garantiza que la información transmitida o almacenada en un sistema informático sólo podrá ser leída por su legítimo destinatario, por lo que si dicha información cae en manos de terceras personas no podrán acceder al contenido original.
- **Integridad:** Se garantiza que desde su creación la información almacenada, procesada o transmitida no ha sido modificada, o en su defecto permite detectar si se ha dañado, añadido o eliminado parte de la información.
- **Disponibilidad:** Mediante un diseño suficientemente robusto frente a ataques e interferencias se garantiza el correcto funcionamiento del sistema informático con la finalidad de que la información esté disponible en todo momento para sus legítimos usuarios y propietarios
- **Autenticidad:** Se puede comprobar la identidad del usuario que crea o accede a la información o a intenta acceder a una red o servicio.
- **No repudio:** Se demuestra la autoría de la información mediante un mecanismo que impida que el usuario que la ha creado y enviado pueda negar dicha circunstancia. Se aplica la misma situación al destinatario de la información

Derivados de los anteriores podemos definir otros principios de seguridad (pueden entrar en conflicto con los anteriores)

- **Autorización:** Permite controlar el acceso de los usuarios a los distintos equipos y servicios ofrecidos por el sistema.
- **Audibilidad:** Permite monitorizar el uso de los recursos del sistema por parte de los usuarios autorizados
- **Reclamación de origen:** Permite probar quién ha sido el creador de determinada información
- **Reclamación de propiedad:** Permite probar que un determinado documento o un contenido digital protegido por derechos de autor pertenece a un determinado usuario u organización que ostenta la titularidad de esos derechos
- **Anonimato:** Garantiza el anonimato de los usuarios que acceden determinados recursos o servicios
- **Protección a la réplica:** Impide la realización de “ataques de repetición” consistentes en la interceptación y posterior reenvío de mensajes para tratar de engañar al sistema y provocar operaciones no deseadas.
- **Confirmación:** Permite confirmar la realización de una operación reflejando los usuarios que han intervenido
- **Referencia temporal:** Permite garantizar la autenticación de las partes que intervienen así como el contenido e integridad de los mensajes y la constatación de la realización de una operación o comunicación en un determinado instante

Aspectos Legales

La seguridad informática es un tema amplio ya que abarca aspectos relacionados con el hardware, software, personas y datos en los que se deben tener en cuenta:

- Los aspectos legales, sociales y éticos relativos al derecho a la privacidad
- Las políticas gubernamentales, institucionales o corporativas relacionadas con la información privada

En la normativa **ISO 27002:2013** se establecen los estándares para la seguridad de la información

La **LOPD** Ley Orgánica de Protección de Datos de Carácter Persona tiene por objeto garantizar y proteger las libertades públicas y los derechos fundamentales de las personas físicas en lo que concierne al tratamiento de los datos personales.

- **Artículo 9** El responsable del fichero y el encargado del tratamiento, deberán adoptar las medidas de índole técnica y organizativas necesarias que garanticen la seguridad de los datos de carácter personal y eviten su alteración, pérdida, tratamiento o acceso no autorizado, habida cuenta del estado de la tecnología, la naturaleza de los datos almacenados y los riesgos a que están expuestos, ya provengan de la acción humana o del medio físico o natural

La **AEPD** Agencia Española de Protección de Datos es el organismo de control público e independiente encargado de velar por el cumplimiento de la LOPD

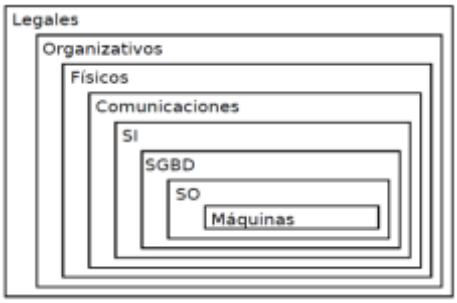
Algunas bases de datos contienen información confidencial a nivel individual pero permiten el acceso estadístico.
Para evitar la inferencia de datos confidenciales:

- No se permite realizar consultas estadísticas en las siguientes circunstancias:
  - Si el número de instancias es inferior a un umbral previamente fijado
  - Si los atributos de la consulta se han utilizado anteriormente de forma repetida
- Se introducen pequeños errores al redondear los resultados estadísticos

Control de Acceso

https://www.monografias.com/trabajos19/administracion-base-datos/administracion-base-datos.shtml

El control de acceso ofrece la posibilidad de acceder a recursos físicos o lógicos a través de un proceso de verificación que permite comprobar si l entidad que está solicitando el acceso tiene los derechos necesarios para hacerlo



Los controles deben garantizar la seguridad regulando:

- El acceso a las maquinas físicas donde se encuentra la información
- El acceso a los ficheros que gestiona el Sistema operativo
- La encriptación de las transacciones y comunicaciones efectuadas en la Red
- La información disponible para cada usuario en el SGBD

Para ello el Sistema de Gestión de base de datos debe:

- Conocer las restricciones especificadas para cada usuario
  - Datos a los que se tiene acceso
  - Operaciones permitidas sobre dichos datos
- Controla el cumplimiento de las restricciones supervisando la interacción con la Base de Datos mediante un log que almacena todas las operaciones que realizan los usuarios en cada sesión de trabajo

El Subsistema de Seguridad y Autorización es el encargado de garantizar la seguridad de la BD contra accesos no autorizados

El administrador debe tener una cuenta de Súper-Usuario que se encargara de llevar a cabo las siguientes acciones

- Dar de alta/baja de cuentas
- Conceder/revocar privilegios
- Asignar niveles de seguridad

Las cuentas y contraseñas se almacenan en una tabla de dos campos

Sistema de control de acceso de MySQL

El sistema de control de acceso de MySQL se basa en 2 aspectos:

- El identificador utilizado para el inicio de la sesión
- La ubicación desde la cual se inicia la conexión.
  - **"localhost"** cuando la conexión se realiza desde la propia máquina donde se está ejecutando MySQL
    - Solo permitiría el acceso desde localhost
  - **"%"** cuando la conexión se realiza desde cualquier máquina.
    - Permitiría el acceso desde todas las ubicaciones

A la hora de asignar privilegios en MySQL se ha de especificar un identificador y una ubicación (se suele considerar "%"por defecto si no se especifica una ubicación) Esto implica que en realidad no se trataría de un único usuario, si no que se estaría definiendo un usuario por cada ubicación desde la cual se conecte. De este modo:

- Podemos asignar privilegios distintos a un único identificador en función de la ubicación desde la cual se conecte.
- Pueden configurarse con la misma contraseña pero no están vinculadas por lo que actualiza una no actualizará la otra

MySQL ordena las cuentas de usuario siguiendo un criterio prefijado y solo trata de autenticar al usuario con la primera cuenta que se ajusta a dicho orden. Dichos criterios de ordenación son los siguientes:

- **Ubicación de las cuentas:** aquellas cuentas con una ubicación más específica preceden a las que tienen una ubicación más genérica.
  - Las direcciones DNS y las direcciones IP se consideran como completamente específicas
  - Las direcciones Localhost
  - La ubicación "%" se considera como genérica.
  - La ubicación "" se interpreta de la misma manera que "%" pero se consideran más genérica
- **El identificador**: aquellas cuentas con un identificador más específico preceden a las que tienen un identificador más genérico.
  - El resto son igual de específicos.
  - "" los usuarios anónimo son los más genéricos más genéricos
- Las cuentas de usuario con identificadores y ubicaciones igual de específicas son ordenadas de forma indeterminada.

Control Obligatorio

Permite clasificar los datos y usuarios en cuatro niveles de seguridad

- Máximo Secreto TS
- Secreto S
- Confidencial C
- Publico U

	Ventajas	Inconvenientes
Discrecional	Flexible	Vulnerable
Obligatorio	Muy seguro	Rígido

- De este modo cada objeto de datos solo puede ser accedido por usuarios con la acreditación apropiada:
  - Lectura: Un usuario solo puede leer objetos que pertenezcan a su mismo nivel de seguridad o inferior
  - Escritura: Un usuario solo puede escribir en objetos de su misma clase o superior
- Con este sistema se consigue que ningún usuario pueda:
  - Leer información que pertenezca a un nivel de seguridad superior
  - Transcribir información de un objeto a otro con un menor nivel de seguridad
- La mayor parte de SGBD sólo proporcionan mecanismos de control de acceso discrecionales pero ciertas aplicaciones requieren sistemas de seguridad multinivel

Control Discrecional

Permite conceder y revocar privilegios a los usuarios para realizar determinadas operaciones sobre ciertos datos

- Un usuario puede tener diversos privilegios sobre distintos objetos de la BD
- Diferentes usuarios pueden tener privilegios distintos sobre un mismo objeto
- Un permiso puede ser recibido por más de un camino
- Un permiso solo se pierde si es revocado por todos los caminos
- La revocación de un permiso puede incluir
  - **CASCADE** para la propagación de la revocación
  - **RESTRICT**: no cancela si afecta a privilegios propagados Esta opción no está disponible en MySQL

Podemos distinguir dos tipos de privilegios

A nivel de cuenta

Se asignan privilegios al margen de los sujetos u objetos de la base de datos

Estos privilegios sirven para operaciones de creación, modificación o eliminación de sujetos/objetos

El propietario de una clase:

- Tiene todos los privilegios sobre dicha clase
- Puede transferir o denegar cualquier privilegio sobre dicha clase a segundos usuarios
- Puede propagar la capacidad de transferir o denegar los privilegios que se posean sobre dicha clase a terceros usuarios

Concesión de privilegios

GRANT < Privilegios > **ON** *.* **TO** < Usuarios > **WITH GRANT OPTION;**

- Si se concede más de un privilegio se listan separándolos por comas
- Si se incluye a más de un usuario se listan separándolos por comas
- La subclausula WITH GRANT OPTION es opcional y otorga al usuario la capacidad de propagar dichos privilegios

Cancelación de privilegios

REBOKE GRANT OPTION FOR < Privilegios > **ON** *.* **FROM** < Usuarios >**;**

- Si se revoca más de un privilegio se listan separándolos por comas
- Si se incluye a más de un usuario se listan separándolos por comas
- La subclausula GRANT OPTION FOR es opcional y retirara únicamente la capacidad de propagar dichos privilegios

Privilegios disponibles

CREATE <SUBCLAUSULA>	Concede el privilegio de crear bases de datos y tablas
	- Disponemos de las siguientes Subclausulas
	  ○ **VIEW** Concede el privilegio de crear vistas
	  ○ **ROUTINE** Concede el privilegio de crear procedimientos almacenados
	  ○ **USER** Concede el privilegio de crear, eliminar y renombrar usuarios

ALTER <SUBCLAUSULA>	Concede el privilegio de editar bases de datos y tablas - Disponemos de las siguientes Subclausulas ○ **ROUTINE** concede el privilegio de crear y modificar procedimientos almacenados
DROP	Concede el privilegio de eliminar bases de datos, relaciones o vistas
INDEX	Concede el privilegio de crear y eliminar índices

A nivel de objeto

Se asignan privilegios para cada objeto de la base de datos
Estos privilegios sirven para:

- Consultar o actualizar la información almacenada en los objetos
- Manipular los objetos mediante los procedimientos almacenados

Concesión de privilegios

GRANT < Privilegios > **(**<Atributos>**)**
ON PROCEDURE < Nombre Base de Datos >**.** < Nombre Tabla >
TO < Usuarios > **WITH GRANT OPTION;**

- Si se concede más de un privilegio se listan separándolos por comas
- Si se incluye a más de un usuario se listan separándolos por comas
- Se pueden conceder los privilegios únicamente sobre un subconjunto de los atributos de la tabla
- La subclausula WITH GRANT OPTION es opcional y otorga al usuario la capacidad de propagar dichos privilegios
- La subclausula PROCEDURE permite dar privilegios sobre un procedimiento almacenado

Cancelación de privilegios

REBOKE GRANT OPTION FOR < Privilegios >
ON PROCEDURE < Nombre Base de Datos >**.** < Nombre Tabla >
FROM < Usuarios >**;**

- Si se revoca más de un privilegio se listan separándolos por comas
- Si se incluye a más de un usuario se listan separándolos por comas
- Se pueden revocar los privilegios únicamente sobre un subconjunto de los atributos de la tabla
- La subclausula GRANT OPTION FOR es opcional y retirara únicamente la capacidad de propagar dichos privilegios
- La subclausula PROCEDURE permite eliminar los privilegios sobre un procedimiento almacenado

Privilegios disponibles

SELECT	Concede el privilegio de consultar registros o atributos de una tabla
INSERT	Concede el privilegio de añadir registros o atributos en una relación
UPDATE	Concede el privilegio de modificar registros o atributos en una relación
DELETE	Concede el privilegio de eliminar registros de una relación
EXECUTE	Concede el privilegio de ejecutar procedimientos almacenados
REFERENCES	Concede el privilegio de referenciar la relación para especificar restricciones de integridad
ALL PRIVILEGES	Concede todos los privilegios de un determinado nivel La opción PRIVILEGES permite “””

Alternativas

Para proporcionar determinados permisos sólo para algunos atributos de una tabla podemos

- Utilizar el método de selección de atributos
- Crear una vista que incluya los atributos y dar los permisos para esa vista
  Si los permisos pueden modificar la tabla hemos de estar seguros de que la vista es actualizable

Auditoria

Se trata de un proceso que permite recoger, agrupar y evaluar evidencias para determinar si un sistema de información:

- Mantiene la integridad de los datos
- Utiliza eficientemente los recursos
- Cumple con las leyes y regulaciones establecidas
- Salvaguarda el activo empresarial
- Lleva a cabo eficazmente los fines de la organización

Para llevar a cabo su función permite medir, monitorear y registrar los accesos a la información almacenada en la base de datos informando al administrador acerca de:

- Los usuarios han intentado acceder
- Cuando se han producido el intento de acceso
- Información sobre el dispositivo, la aplicación y la dirección de red desde la que se llevaba a cabo la conexión
- Cuáles eran los recursos a los que se intentaba acceder
- El efecto sobre la base de datos que tuvo dicho acceso

Mediante un sistema de auditoria se consigue garantizar el cumplimiento de la legislación vigente respecto a la **integridad de la información** almacenada debido a que queda registrado cualquier intento de acceso y modificación de la información así como su autor mitigando los riesgos asociados a la perdida de información y resguardando información confidencial

Requisitos

No puede comprometer el funcionamiento de la base de datos sobre la que opera

El auditor no puede ser un usuario de la base de datos debido a que nadie podría controlar sus acciones

Debe facilitar información valiosa en cuanto a

- Seguridad
- Apoyo en la toma de decisiones
- Mejora en el funcionamiento de la organización

Inconvenientes

Consumen muchos recursos del sistema en cuanto a almacenamiento

Cuando el sistema de almacenamiento se agota puede producirse un bloqueo del sistema de auditoria para impedir el colapso de las funciones principales del sistema de almacenamiento. Esto se puede evitar

- Ampliando el sistema de almacenamiento si el funcionamiento del sistema se ve comprometido de forma repetida
- Almacenar las tablas del sistema que almacenan información acerca de la seguridad en un disco independiente e inactivo

Acciones auditables

- Todas las conexiones y desconexiones aceptadas y fallidas
- Reinicios del servidor
- Todas las funcionalidades SELECT, INSERT, UPDATE y DELETE así como procedimientos almacenados
    - Que sean ejecutadas por usuarios con permisos de administrador
    - Que provocan intentos de violación de restricciones de integridad o falta de autorización
    - Que provocan cambios en las tablas de sistema

Herramientas

Software específico de auditoría que facilita:

- La extracción de información
- El seguimiento de las transacciones
- La utilización de datos de prueba

Combinan información de diferentes Sistemas

- **Sistema de monitorización y ajuste (Tunning)** Se trata de un servicio permite:
    - Obtener información de la auditoría
    - Definir el nivel de auditoría óptimo
- **Sistema operativo** Se trata de un servicio permite
    - Controlar el acceso a la memoria y a los Ficheros del sistema o buffers
    - Controlar situaciones de error
- **Monitor de transacciones**
- **Protocolos y Sistemas Distribuidos**
- **Paquetes de Seguridad**

Recuperación

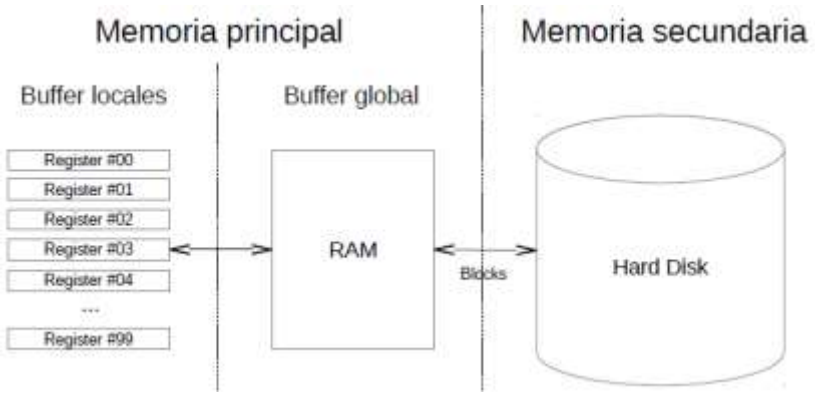
Un Sistema de Gestión de Base de Datos debe proporcionar alta disponibilidad, minimizando el tiempo de la incidencia

Los mecanismos de recuperación permiten restaurar el último estado consistente de un Sistema de Gestión de Base de Datos asegurando su disponibilidad frente a fallos lógicos o físicos que destruyan toda o parte de la información almacenada

Unidades lógicas de información

Para definir un sistema de recuperación es necesario agrupar la información en bloques consistentes

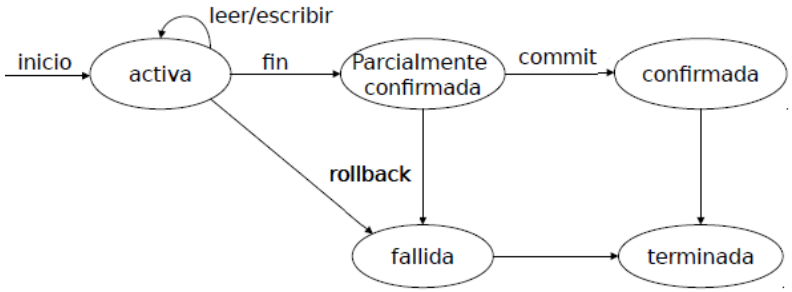
- **Unidad lógica de procesamiento:** Secuencia de instrucciones que incluye operaciones de acceso (inserción, borrado, modificación o consulta) a la base de datos
- **Unidad lógica de integridad:** Secuencia de operaciones que llevan la BD de un estado consistente a otro estado consistente
- **Unidad lógica de concurrencia:** Secuencia de operaciones que pueden ejecutarse independientemente de la ejecución de otras transacciones
- **Unidad lógica de recuperación:** Secuencia de transacciones cuyos cambios es posible garantizar su consistencia frente a posibles fallos



Consistencia de las operaciones

Para garantizar la consistencia de las operaciones los accesos a la información deben cumplir un conjunto de reglas:

- Una transacción no puede ejecutarse a medias o bien se ejecuta íntegramente o no se ejecuta.
- La ejecución de una transacción no puede interferir en la ejecución de otra transacción
- Los cambios realizados por una transacción confirmada deben persistir en la base de datos
- Una transacción no puede ejecutarse más veces de las que estaban previstas



Transacciones

Para garantizar su cumplimiento el gestor de la recuperación monitoriza las siguientes operaciones

- $\boldsymbol{START\ TRANSACTION}\ (< \boldsymbol{T} > < \boldsymbol{time} >)$
  Indica que la transacción T ha comenzado su ejecución
- $\boldsymbol{READ}(< \boldsymbol{T} > < \boldsymbol{X} > < \boldsymbol{valor} > < \boldsymbol{time} >)$
  Indica que la transacción T ha leído el valor del elemento X de la Base de Datos
- $\boldsymbol{WRITE}(< \boldsymbol{T} > < \boldsymbol{X} > < \boldsymbol{valorAntiguo} > < \boldsymbol{valorNuevo} > < \boldsymbol{time} >)$
  Indica que la transacción T ha modificado el valor del elemento X
- $\boldsymbol{COMMIT}(< \boldsymbol{T} > < \boldsymbol{time} >)$
  Permite que las modificaciones completamente realizadas se confirmen en la Base de datos
- $\boldsymbol{ROLLBACK}(< \boldsymbol{T} > < \boldsymbol{time} >)$
  Permite que las modificaciones que no han podido terminar deshagan su proceso y se reinicien sin que sus operaciones tengan efecto sobre la Base de Datos
- $\boldsymbol{END\_TRANSACTION}(< \boldsymbol{T} > < \boldsymbol{time} >)$
  Indica que la transacción T ha finalizado su ejecución
- $\boldsymbol{CHECKPOINT}(< \boldsymbol{time} >)$
  Permite forzar un punto de recuperación

Tipos de fallos

Los fallos pueden afectar a la ejecución de las transacciones, impidiendo que se cumplan algunas de sus propiedades ACID:

Fallos que provocan la pérdida de la memoria volátil (memoria principal)
Fallos que provocan la pérdida de la memoria estable (memoria secundaria)

Fallo por imposición del control de concurrencia

- Violación de la seriabilización, bloqueo mortal

Fallo en la transacción

- Overflow, división por cero, violaciones de restricciones, …
- Fallos locales (sólo afectan a la transacción fallida)

Anular las transacciones que provocaron la inconsistencia: deshacer algunas operaciones (requiere el uso del diario)
- Si es necesario, asegurar cambios correctos: rehacer algunas operaciones (requiere el uso del diario)

Fallo software

- SGBD, SO, comunicaciones, …
- Fallos globales

Fallo hardware

- Fallos en discos, máquinas, redes, …
- Fallos globales

Fallos catastróficos

- Corte de suministro eléctrico, robo,

incendio, inundación, sabotaje,
borrado/sobreescritura accidental, …

Restaurar copia de seguridad de la BD
- Reconstruir el estado consistente más reciente: rehacer algunas operaciones (requiere el uso del diario)

Técnicas de recuperación

Redundancia física

Se utilizan particiones de disco distintas

Se realiza una copia de seguridad periódica

Se obliga a que los datos que se modifican se escriban en disco antes que en la Base de Datos

Se utilizan diarios para monitorizar la ejecución de las transacciones operaciones realizadas sobre los datos

Un diario es un fichero almacenado en disco que almacena todas las operaciones que se ejecutan en la base de datos

Recuperar un fallo consistirá en deshacer o rehacer algunas de las operaciones a partir del contenido del diario.

En el momento en el que ocurre un fallo:

- Aquellas transacciones que estaban en ejecución deben deshacerse debido a que no llegaron a su punto de confirmación
  Esto implica deshacer cada una de las operaciones a partir de las anotaciones en el diario, empezando por la última y en orden inverso
- Aquellas transacciones que ya han sido confirmadas deben rehacerse debido a que no podemos garantizar que todos los cambios se hallan escrito correctamente en la base de Datos
  Esto implica rehacer cada una de las operaciones a partir de las anotaciones en el diario, empezando por la primera y en el mismo orden

Cuando una transacción realiza un COMMIT significa que:

- La transacción ha llegado a un punto de confirmación
- El efecto de dichas operaciones se anotó en la diario (incluyendo el COMMIT)
- Todas las operaciones se han ejecutado con éxito
- La transacción queda confirmada y sus cambios son permanentes
- Se liberan los bloqueos, buffers, etc…

Cuando una transacción realiza un ROLLBACK significa que:

- La transacción ha llegado a un punto de confirmación
- El efecto de dichas operaciones se anotó en el diario (incluyendo el ROLLBACK)
- Se ha producido un fallo durante la transacción
- La transacción queda cancelada y sus operaciones no deben tener efecto
- Se liberan los bloqueos, buffers, etc…

Puntos de recuperación

Marcar un punto de recuperación implica

- Suspender la ejecución de todas las transacciones
- Escribir en el buffer del diario el registro de validación y forzar la escritura del buffer del diario en disco
- Forzar la escritura en disco de todo bloque del buffer global
- Reanudar la ejecución de las transacciones

Los puntos de validación permiten

- Recorrer el diario a partir del último punto de validación en vez de comenzar desde el principio
- Ignorar las transacciones confirmadas antes del último punto de validación (no es necesario rehacer todas las transacciones confirmadas)

Estos puntos de validación contienen:

- La lista de identificadores de las transacciones activas
- La dirección en el diario de la primera y última entrada de cada transacción activa

El Sistema de Gestión de Base de Datos marca automáticamente un punto de validación cada cierto tiempo tras escribir un determinado número de entradas en el diario.

Escritura inmediata en el diario

Cada operación ejecutada en la base de datos se anota inmediatamente en el diario del disco

Escritura diferida en el diario

Se utiliza un buffer local del diario que contiene un bloque con las últimas entradas del diario.

Cuando dicho buffer se llena de entradas se copia todo el bloque al disco mediante una única escritura por bloque

Cuando ocurre un fallo algunas entradas pueden no haber sido llevadas al diario en disco por lo que dichas entradas no serán consideradas en el proceso de recuperación

Diario adelantado

No se pueden actualizar los cambios realizados por una transacción en los registros de la Base de Datos hasta que se haya llevado a disco la anotación del diario que lo representa operación

Además el COMMIT de la transacción no se puede completar hasta que se haya escrito en disco cualquier entrada de diario para dicha transacción (Incluyendo el propio COMMIT)

Con el diario adelantado se garantiza que cuando se produzca un fallo en la base de datos todas las operaciones que se hayan llevado a la base de datos estarán también en el diario debido a que si el fallo sucede antes de que la operación se anote en el diario, dicha operación tampoco se habrá llevado a la memoria principal

Actualización diferida - no-deshacer / rehacer

Se difiere la consolidación de los cambios realizados por la transacción hasta después de confirmarse

- Si el fallo ocurre antes de alcanzar su punto de confirmación, no es necesario deshacer sus operaciones
- Si el fallo ocurre después de su punto de confirmación, es necesario rehacer sus operaciones

Este sistema implica

- Las entradas de escritura sólo necesitan guardar el valor nuevo dado que pueden rehacerse, pero nunca deshacerse
- Reiniciar una transacción es reintroducirla en el sistema como si fuera nueva
- Las operaciones se rehacen una a una en el orden en que aparecen anotadas en el diario

Actualización inmediata - deshacer / rehacer

Se pueden actualizar inmediatamente los cambios realizados por la transacción antes de confirmarlos

- Si el fallo ocurre antes de alcanzar su punto de confirmación, es necesario deshacer sus operaciones
- Si el fallo ocurre después de su punto de confirmación, es necesario rehacer sus operaciones

Este sistema implica

- Las entradas de escritura necesitan guardar el valor anterior y nuevo dado que pueden rehacerse o deshacerse
- Se debe deshacer primero y rehacer después
- Las operaciones se deshacen una a una en el orden inverso en que aparecen anotadas en el diario
- Las operaciones se rehacen una a una en el orden en que aparecen anotadas en el diario

Control de concurrencia

El acceso simultáneo a los datos no lleva necesariamente a estados consistentes de la base de datos debido a que varias transacciones pueden leer o escribir el mismo registro al mismo tiempo.

Los principales problemas que genera la concurrencia son:

- **Perdida de actualizaciones**: se produce cuando una transacción modifica el valor de un registro y seguidamente otra transacción lo sobrescribe
- **Actualizaciones asumidas**: se produce cuando una transacción lee un valor que seguidamente es modificado por otra transacción
- **Lecturas inconsistentes**:

Planificación de transacciones

Un plan de transacciones es una serie intercalada de acciones de distintas transacciones que se ejecutan de manera concurrente y donde se respeta el orden relativo de la acción dentro de la transacción

- Si se permite la intercalación de las operaciones de varias transacciones, existen muchos órdenes posibles

Serialización de planificaciones

Una planificación es serializable si produce un resultado equivalente a una planificación en serie de las mismas transacciones

- Es posible crear un plan en serie equivalente a la planificación mediante una ordenación en serie de las transacciones

La serialización permite:

- Decidir que planificaciones son correctas
- Definir las técnicas de control de concurrencia que eviten planes incorrectos

Los resultado de una planificación serializable son equivalente si

- Para cualquier estado inicial producen el mismo estado final en la base de datos
- Las operaciones se aplican a todos los elementos afectados por la planificación en el mismo orden

Dos operaciones de una planificación serializable están en conflicto si:

- Pertenecen a diferentes transacciones
- Tienen acceso al mismo elemento X
- Al menos una de ellas es WRITE

Dos planes son equivalentes por conflictos si:

- El orden de cualesquiera dos operaciones en conflicto es el mismo en ambos planes

Un plan P es serializable por conflictos si

- Es equivalente por conflictos a algún plan en serie
- El orden de las operaciones en conflicto coincide con el orden en que se ejecutarían en algún plan en serie

Podemos Construir un grafo de precedencia dirigido

- Si hay un ciclo en el grafo la planificación no es serializable por conflictos
- Si no hay ciclos el plan es serializable por lo que

No es posible crear un sistema automático que detecte y resuelva los grafos de precedencia para todas las transacciones que se ejecuten en el sistema debido a que es computacionalmente muy costoso

- El grafo se modificaría cada vez que aparezca una transacción nueva o se resuelva alguna ya existente
- Para analizar cada transacción hay que analizar cada una de las transacciones activas al menos una vez (coste cuadrático)

Protocolos de serialización

Es un conjunto de reglas basado en la teoría de la serialización que permite simplificar considerablemente el coste computacional realizando algunas asunciones

El plan de transacciones es serializable si todas las transacciones cumplen las reglas establecidas por el Sistema de Gestión de Base de Datos

Protocolos basados en reservas

La forma de asegurar la serialización es proporcionar protocolos de acceso a los datos mediante reservas.

- Cada transacción debe realiza una petición de reserva antes de acceder a un dato
- Si la petición no puede ser atendida, la transacción tendrá que esperar hasta que el dato se libere.
- Después de haber completado las operaciones pertinentes la transacción deberá liberal el dato
- Ninguna transacción puede reservar un dato que ya este reservado
- Ninguna transacción puede liberar un dato que no tenga previamente reservado

A cada registro de la base de datos se le asocia una variable que:

- Describe su estado respecto a las operaciones aplicables
- Sincroniza el acceso al elemento por parte de transacciones concurrentes
- Podemos encontrar dos tipos de candados
    - **Candados binarios** Es un método bastante restrictivo porque solo pueden ofrecen acceso exclusivo a los datos
    - **Candados ternaros**
        - Pueden ofrecer acceso compartido a un dato solo si todas las transacciones quieren consultarlo
        - Si alguna transacción quiere actualizar entonces tiene que reservar de forma exclusiva
        - Si una transacción tiene acceso exclusivo a un dato ninguna otra puede reservarlo en ninguna modalidad
        - Ninguna transacción podrá reservar en modo exclusivo un dato que se encuentra reservado en compartido

Protocolo de reserva de dos fases

El uso de reservas para la programación de transacciones no garantiza la serialización de los planes

Es necesario seguir un protocolo adicional que indique donde colocar las operaciones de bloqueo y desbloqueo de reservas dentro de las transacciones

El protocolo de reservas en dos fases:

- Añade una regla adicional al protocolo de reservas:

    "En toda transacción, todas las peticiones de reserva deben preceder a cualquier petición de liberación"

- Dentro de toda transacción se distinguen dos fases:
    - Fase de expansión o crecimiento donde la transacción
        - Puede realizar o promover reservas
        - No puede liberar ninguna reserva
    - Fase de contracción donde la transacción
        - No puede realizar ninguna reserva
        - Puede liberar o degradar reservas
- Toda transacción que sigue este protocolo es serializable por lo que ya no es necesario comprobar la serialización
- Puede limitar el grado de concurrencia

Este protocolo tiene cuatro variantes:

Protocolo conservador o estático

La transacción debe reservar todos los elementos a los que tendrá acceso antes de comenzar a ejecutarse

- Si no es posible reservar algún elemento no se reservara ninguno y esperara para reintentarlo más adelante
- Protocolo libre de bloqueo mortal

Protocolo estricto

La transacción no libera ninguna reserva de escritura hasta haber terminado su procedimiento

- Ninguna otra transacción puede leer o escribir un elemento modificado por la transacción salvo si esta se ha completado
- No es libre de bloqueo mortal (excepto si se combina con protocolo conservador)

Protocolo riguroso

La transacción no libera ninguna reserva ni de escritura ni de lectura hasta haber terminado su procedimiento

Bloqueo mortal

Se produce cuando una transacción está bloqueada por otra transacción que a su vez está bloqueada por la primera.

Condiciones necesarias para que se produzca el bloqueo mortal:

- **Exclusión mutua:** todas las transacciones bloqueadas están esperando la liberación de un registro por parte de otra
- **Reserva parcial:** todas las transacciones tienen al menos una reserva pero ninguna tiene todas las reservas necesarias para resolver su proceso
- **No expropiación:** Ninguna transacción tiene la capacidad de expropiar los datos que le faltan a las demás transacciones
- **Espera circular:** cada una de las transacciones bloqueadas está esperando a otra transacción de forma que la última de ellas estará esperando a la primera

Soluciones para el problema del bloqueo mortal

- **Protocolos de prevención**
  - **Reserva por adelantado**
    Cada transacción reserva desde el principio todos los elementos de lectura y escritura que necesita
    Si no puede reservarlos todos se cancela la transacción y lo reintenta después

    Inconvenientes:
    - Los datos están reservados durante más tiempo
    - Requiere conocer todos los datos que van a usarse al principio
    - Concurrencia muy limitada
  - **Reserva ordenada**
    Se ordenan los elementos de la base de datos conforme a un criterio común y se asegura que si una transacción necesita varios elementos, los reservara en ese orden.
    Si no puede reservarlos todos se cancela la transacción y lo reintenta después

    Inconvenientes:
    - Los programadores deben conocer y respetar el criterio de orden
    - Concurrencia muy limitada
  - **Uso de marcas de tiempo de transacción**
    Se asigna a cada transacción un identificador único en función del momento en el que se inició la su proceso
    Cuando a una transacción se le deniega el permiso para acceder a un dato se determinara si la transacción debe esperar o terminar en función de las marcas de tiempo de las transacciones involucradas
    - **Esperar – Morir**
      Si la transacción solicitante es más antigua que la que tiene el dato reservado
      La transacción solicitante espera a que termine la transacción más joven

      Si la transacción solicitante es más joven que la que tiene el dato reservado
      La transacción solicitante abortara el proceso y se reiniciara
    - **Herir – Esperar**
      Si la transacción solicitante es más antigua que la que tiene el dato reservado
      La transacción solicitante expropiara el dato a la transacción más joven que deberá terminar y reiniciarse

      Si la transacción solicitante es más joven que la que tiene el dato reservado
      La transacción solicitante espera a que termine la transacción más antigua
  - **No Esperar**
    Cuando a una transacción se le deniega el permiso para acceder a un dato esta abortara el proceso y se reiniciara

    Inconvenientes:
    - se producen reinicios innecesarios debido a que no se comprueba si realmente se está produciendo un bloqueo mortal
  - **Espera cautelosa**
    Cuando a una transacción se le deniega el permiso para acceder a un dato se le permitirá esperar siempre y cuando la transacción por la que esta esperando no se quede también bloqueada
  - **Uso de tiempos predefinidos**
    Cuando a una transacción se le deniega el permiso para acceder a un dato se le permite esperar durante un tiempo predeterminado. Cuando dicho intervalo se agota sin que se haya accedido al registro bloqueado la transacción solicitante abortara el proceso y se reiniciara

    Inconvenientes:
    - se producen reinicios innecesarios debido a que no se comprueba si realmente se está produciendo un bloqueo mortal

- **Protocolos de detección**
  El sistema comprueba periódicamente si se han producido bloqueos entre las transacciones y determina cuál de las transacciones bloqueadas se reinicia

  Requeriría la construcción de grafos de espera
  - Se crea un nodo por cada transacción en ejecución
  - Se crea un desde cada transacción bloqueada hasta la transacción que tiene el registro reservado
  - Cuando un registro reservado se libera se eliminan los arcos afectados

  Si se detecta un ciclo en el grafo de espera significa que existe un problema de bloqueo mortal entre las transacciones

  El periodo de detección se puede determinar en función de
  - El número de transacciones bloqueadas
  - El número de transacciones en ejecución
  - El tiempo que lleven esperando las transacciones bloqueadas

  Para determinar las transacciones que se deben reiniciar se pueden aplicar los siguientes criterios:
  - No elegir las transacciones que lleven mucho tiempo en ejecución o hayan realizado muchas modificaciones
  - Elegir las transacciones que participan en varios ciclos de bloqueos mortales

Espera indefinida

Se produce cuando una transacción no puede ejecutarse, mientras todas las demás se procesan con normalidad.

Este problema se debe a un esquema de espera injusto en el que se realiza una mala selección de la prioridad en el que las transacciones deben reiniciarse cuando se detecta un problema de bloqueo mortal.

Para solucionar este problema se puede

- Aumentar la prioridad de cada transacción que se reinicia
- Esperar–Morir y Herir–Esperar evitan la espera indefinida

Protocolos basados en marcas de tiempo

Cada transacción tiene asociada una marca de tiempo asignada por el sistema al empezar su ejecución $T(Transaccion)$

Se establece una planificación en serie que resulta de ordenar las transacciones por su marca de tiempo

Cada elemento tiene dos marcas de tiempo:

- Marca de tiempo de lectura más reciente $T(Lectura)$
- Marca de tiempo de escritura más reciente $T(Escritura)$

Cuando una transacción accede a un dato se compara la marca de tiempo de la transacción con las marcas de tiempo de las operaciones con las que se encuentra en conflicto.

- **Operaciones de lectura**
  Solo pueden estar en conflicto con las operaciones escritura
  Se compara la marca de tiempo de la transacción con la marca de tiempo escritura del elemento
  - Si $T(Transaccion) \geq T(Escritura)$ entonces $T(Lectura) = Max[\ T(Lectura)\ ,\ T(Transaccion)\ ]$
  - Si $T(Transaccion) < T(Escritura)$ entonces la transacción deberá abortar el proceso y reiniciarse
- **Operaciones de escritura**
  Pueden estar en conflicto con las operaciones escritura y de lectura
  Se compara la marca de tiempo de la transacción con la marca de tiempo de lectura y escritura del elemento
  - Si $T(Transaccion) \geq T(Escritura)$ y Si $T(Transaccion) \geq T(Escritura)$
    Entonces $T(Escritura) = Max[\ T(Escritura)\ ,\ T(Transaccion)\ ]$
  - Si $T(Transaccion) < T(Escritura)$ o $T(Transaccion) < T(Lectura)$
    Entonces la transacción deberá abortar el proceso y reiniciarse

Siempre que el algoritmo detecte dos operaciones conflictivas rechazara la última de las operaciones abortando la transacción que la realizo

Protocolos de concurrencia multiversión

Para almacenar versiones distintas de un mismo dato se requiere muchisimo mas espacio

Sin embargo, de todos modos debemos guardar esas versiones

- Por ejemplo, para realizar recuperaciones

El caso extremo son las BD temporales

- Mantienen todos los cambios y los instantes en que ocurrieron
- No se requiere mas espacio que el que ya se consume

Protocolos optimistas

Llamadas también técnicas de validación o certificación

- No se realiza ninguna comprobación mientras se realiza la transacción
- Las modificaciones no se realizan directamente sobre la BD: todas las modificaciones se realizan sobre copias locales
- Las modificaciones no se realizan hasta el final de la transaccion
- La transaccion finaliza correctamente si no se viola la serializacion
- En caso contrario se aborta la transaccion

La transaccion se ejecuta en tres fases:

- Fase de lectura: una transaccion puede leer valores y almacenarlos en variables locales. Las modificaciones se realizan sobre las variables locales
- Fase de validacion: se verifica la serializacion de la transaccion
- Fase de escritura: si la transaccion es serializable, se modifica la BD con los datos locales. En otro caso, rollback (la transaccion no ha tenido repercusion)

- Enfoque similar a la marca de tiempo
- Sin embargo, aqui la marca de tiempo es su validacion
- Por tanto, la planificacion en serie equivalente es la que resulta de ordenar las transacciones en funcion del comienzo de la fase de validacion
- Se llama tecnica optimista porque asume poca interferencia entre las transacciones