

UNIVERSIDAD TECNOLÓGICA NACIONAL

FACULTAD REGIONAL RESISTENCIA

INGENIERÍA EN SISTEMAS DE INFORMACIÓN

**BASES DE DATOS**

Guía de Trabajos Prácticos: Unidad 6

Carpeta de Entrega

**Grupo 4**

Integrantes:

* **Andres**, Aldo Omar.
* **Bravo Pérez**, Agustín Nicolás.
* **Brites,** Agustín.
* **Sáez Franci**, Juliana Carla Desiree.
* **Salomón**, Hilel Mauricio.
* **Machuain**, Ezequiel.

*2 de Julio de 2023*

Cuestionario Unidad VI: Almacenamiento de registros; organización de ficheros primarios; índices.

*• ¿Cómo almacena y accede a datos persistentes un SGBD?*

*• ¿Por qué el coste de E/S es tan importante para las operaciones de base de datos?*

*• ¿Cómo organiza un SGBD los archivos de registros de datos en disco para minimizar el coste de E/S?*

*• ¿Qué es un índice y por qué se usa?*

*• ¿Cuál es la relación entre un archivo de registros de datos y cualquiera de los índices sobre este archivo?*

*• ¿Qué propiedades importantes tienen los índices?*

*• ¿Cómo funciona un índice asociativo y cuándo es más efectivo?*

*• ¿Cómo funciona un índice basado en árboles y cuándo es más efectivo?*

*• ¿Cómo se pueden utilizar índices para optimizar el rendimiento para una carga de trabajo dada?*

*1. ¿Qué es una “instancia” en MySQL? ¿Cuál es la diferencia con una “base de datos” o “schema”?*

*2. Enumere y dé una breve descripción sobre clasificaciones y tipos de datos soportados por el gestor (capítulos 11.1 a 11.5 del manual de referencia).*

*3. ¿Qué son los “storage engines” en MySQL? Elabore un concepto y describa brevemente al menos tres de ellos.*

*4. ¿Dónde almacena físicamente las bases de datos MySQL?*

*5. Cree una base de datos, que como único requisito obligatorio debe llamarse “BDA2023”; para ésto investigue qué parámetros opcionales puede establecer y/o configurar para la creación. Escoja, y fundamente, los que considere adecuados. Documente cada paso con una copia de pantalla del asistente de creación o el comando utilizado. Haga lo mismo para la creación de una tabla “Tabla1” con cinco campos de distintos tipos.*

*6. Elija al menos cinco conceptos vistos en la clase de teoría de “Introducción a las bases de datos” e identifique y describa brevemente cómo son implementados por MySQL.*

*Manual de Referencia MySQL 8.0: https://dev.mysql.com/doc/refman/8.0/en/*

## ¿Cómo almacena y accede a datos persistentes un SGBD?

Un sistema de gestión de bases de datos (SGBD) almacena información de manera persistente utilizando dispositivos de almacenamiento externo, como discos y cintas. Los datos se organizan en páginas y, para accederlos, se cargan en memoria para su procesamiento mediante el gestor de la memoria intermedia. El gestor de espacio en disco administra el espacio libre y ocupado en el disco para los archivos de la base de datos.

## ¿Por qué el coste de E/S es tan importante para las operaciones de base de datos?

El coste de E/S, es decir, la demora de la transferencia de datos entre la memoria principal y los dispositivos de almacenamiento, es crucial en las operaciones de base de datos debido a su impacto en el tiempo de respuesta a una consulta, lo cual impacta a la escalabilidad y el rendimiento del SGBD. Casi toda sentencia SQL implica leer o modificar un archivo (datos en memoria secundaria), lo cual implica una operación de E/S.

Optimizar el coste de E/S es fundamental porque un coste más bajo permite un acceso más rápido a los datos, una mejor utilización de los recursos y un rendimiento más eficiente en general.

## ***¿Cómo organiza un SGBD los archivos de registros de datos en disco para minimizar el coste de E/S?***

Los SGBD organizan estos archivos de registros utilizando técnicas como la agrupación de registros contiguos, compresión de datos, almacenamiento en búfer y algoritmos de recuperación eficientes. Estas técnicas ayudan a minimizar el coste de E/S al reducir la cantidad de operaciones de lectura y escritura que se deben ejecutar.

En particular, el uso de cachés en memoria principal almacenan temporalmente datos leídos de los archivos. Una consulta que requiera buscar unos datos de un archivo, se fijaría primero si están en el caché, para evitar una operación de E/S.

Hay distintas estructuras para organizar los archivos de registros en disco. Por ejemplo, los archivos de acceso directo dividen a los registros en bloques, con una dirección física para cada bloque, de manera que se pueda acceder a cualquier registro de forma directa, evitando un acceso secuencial. Podrían agregarse estructuras adicionales como un archivo índice, que guarda una colección de pares clave-dirección para hacer todavía más rápida la búsqueda de un registro particular, evitando recorrer todo el archivo buscando, por ejemplo, el primer nombre que comienza con 'M'.

## ¿Qué es un índice y por qué se usa?

Un índice es una estructura de datos que se utiliza para mejorar la eficiencia de las operaciones de búsqueda en una base de datos. El propósito principal es acelerar el acceso a datos, reduciendo el costo de E/S. La desventaja que tienen es que también ocupan espacio de almacenamiento, y se deben actualizar cada vez que los datos cambien, ya que los índices apuntan a los datos.

Se basan en la idea de que buscar un dato en una lista es mucho más rápido cuando la lista está ordenada. Debido a que el archivo físico solo puede estar ordenado en base a una columna (generalmente la clave primaria), se pueden crear archivos índices secundarios de las otras columnas que mantengan una colección de pares clave-dirección con valores de la columna indexada junto con punteros a las direcciones físicas correspondientes en el archivo de las filas. Esta estructura permite a la base de datos realizar búsquedas más rápidas en función de los valores del índice, en lugar de ir fila a fila hasta encontrar la que busca.

Resumiendo, los índices ofrecen las siguientes ventajas:

1. Mejora en el rendimiento de consultas.
2. Reducción de la carga en la base de datos.

## ¿Cuál es la relación entre un archivo de registros de datos y cualquiera de los índices sobre este archivo?

Los archivos de registros de datos y los índices asociados a esos datos tienen una relación de dependencia y complementariedad. Los archivos índices dependen del archivo de datos, ya que cuando los datos cambien los índices deben actualizarse para mantener la consistencia, y a la vez lo complementan, ya que proporcionan un acceso más eficiente a los datos.

La relación de complementariedad se nota cuando se ejecutan consultas de lectura, ya que el SGBD puede utilizar un índice para encontrar un registro particular sin recorrer toda la tabla. La dependencia se nota cuando una consulta de modificación de datos obliga a actualizar el índice. Por ejemplo, si hay un índice de los nombres de usuario donde la clave es la primer letra, y un usuario pasa de nombre 'jugador' a 'player', la primer letra cambió.

## ¿Qué propiedades importantes tienen los índices?

Los índices en los sistemas de gestión de bases de datos ofrecen propiedades importantes que los hacen (usualmente) efectivos. Estas son:

1. Eficiencia de búsqueda: Los índices están diseñados para mejorar la eficiencia de búsqueda en una base de datos. Permiten una recuperación más rápida de los datos al evitar la necesidad de recorrer todo el archivo de registros. En cambio, los índices proporcionan un acceso directo a los registros que cumplen con ciertos criterios de búsqueda.
2. Ordenación: Los índices mantienen los valores de las columnas ordenados, lo que facilita las búsquedas basadas en rangos y comparaciones. Al tener los datos ordenados, los índices permiten realizar operaciones como búsqueda de valores mínimos y máximos, o realizar consultas por igualdad o rango de valores.
3. Reducción de la carga de la base de datos: Al acelerar las búsquedas y consultas, los índices reducen la carga en la base de datos. Esto mejora el rendimiento general del sistema y permite manejar un mayor número de solicitudes simultáneas.
4. Flexibilidad en la creación de índices: Los SGBD permiten crear índices en una o varias columnas de una tabla, lo que brinda flexibilidad para optimizar diferentes tipos de consultas. Puedes crear índices en columnas individuales, en combinaciones de columnas o incluso utilizar índices especializados como índices de texto completo.
5. Mantenimiento y actualización: Los índices requieren un mantenimiento adecuado para mantener su eficiencia. Cada vez que se realizan modificaciones en los datos (inserciones, actualizaciones o eliminaciones), los índices deben actualizarse para reflejar esos cambios y mantener la coherencia con el archivo de registros de datos.
6. Sobrecarga de almacenamiento: Los índices ocupan espacio adicional en el disco, ya que almacenan estructuras de datos adicionales. Esto puede aumentar los requisitos de almacenamiento de la base de datos, especialmente en tablas grandes o con múltiples índices. La sobrecarga de almacenamiento debe tenerse en cuenta al decidir qué columnas indexar y cómo gestionar los recursos de almacenamiento.

## ¿Cómo funciona un índice asociativo y cuándo es más efectivo?

Un índice asociativo, también conocido como índice hash, es un tipo de índice utilizado en SGBD para acelerar la búsqueda de registros basados en la igualdad. A diferencia de otros tipos de índices que mantienen una ordenación de los valores, un índice asociativo utiliza una función hash para calcular una ubicación directa donde se almacenan los valores y sus correspondientes punteros.

El funcionamiento de estos índices se basa en tres pasos principales:

1. Cálculo del hash: Cuando se inserta un nuevo registro en la base de datos, se aplica una función hash a la columna indexada para calcular un valor hash. Este valor se utiliza para determinar la ubicación donde se almacenará el registro en la estructura del índice.
2. Almacenamiento: El registro se inserta en la ubicación calculada a partir del valor hash. En el índice asociativo, no hay una ordenación específica de los valores almacenados. Cada ubicación en la estructura del índice se conoce como "bucket" y puede contener múltiples registros si hay colisiones de hash (dos registros que generan el mismo valor hash).
3. Búsqueda: Cuando se realiza una consulta basada en igualdad en la columna indexada, se aplica la función hash al valor buscado y se busca en el bucket correspondiente. Si hay más de un registro en ese bucket debido a una colisión de hash, se pueden utilizar técnicas adicionales para identificar el registro correcto, como la comparación directa del valor buscado.

El índice asociativo es más efectivo en situaciones donde las búsquedas se realizan principalmente en base a igualdad exacta, es decir, cuando se busca un valor específico en la columna indexada. Esto se debe a que la función hash permite una búsqueda directa y no es necesario recorrer todo el índice. Además, tiende a ser eficiente incluso para tablas con un gran número de registros, ya que la búsqueda se reduce a un número limitado de buckets en lugar de recorrer una estructura jerárquica.

Sin embargo, el índice asociativo no es tan efectivo para consultas que involucran rangos o patrones de búsqueda más complejos. En esos casos, otros tipos de índices, como los índices de árbol B, pueden ser más apropiados. Además, el rendimiento del índice asociativo puede verse afectado por colisiones de hash, donde varios registros generan el mismo valor hash y se almacenan en el mismo bucket, lo que puede ralentizar la búsqueda.

## ¿Cómo funciona un índice basado en árboles y cuándo es más efectivo?

Un índice basado en árboles, específicamente un índice de árbol B, es una estructura de datos utilizada en sistemas de gestión de bases de datos para acelerar la búsqueda y recuperación de datos. Este tipo de índice utiliza una estructura de árbol balanceado para organizar y ordenar los valores de una columna indexada.

El funcionamiento de un índice de árbol B es el siguiente:

1. Organización de los valores: Los valores de la columna indexada se organizan en una estructura de árbol balanceado. Cada nodo del árbol contiene varios valores y punteros a otros nodos o registros en la base de datos.
2. División y equilibrio: A medida que se insertan nuevos valores en el índice, se realiza una división de nodos y reequilibrio del árbol para mantenerlo balanceado. Esto garantiza que la altura del árbol se mantenga relativamente baja y que las operaciones de búsqueda sean eficientes.
3. Búsqueda: Cuando se realiza una búsqueda en el índice, se inicia desde la raíz del árbol y se desciende a través de los nodos siguiendo una serie de comparaciones. Cada nivel del árbol proporciona información sobre qué dirección tomar para llegar al valor buscado. El proceso continúa hasta llegar a un nodo hoja que contiene el valor buscado o hasta que se determina que el valor no está presente en el índice.
4. Rangos y consultas adicionales: Además de la búsqueda de igualdad, los índices de árbol B también pueden admitir consultas basadas en rangos. Al seguir los punteros adecuados en el árbol, se pueden identificar rápidamente los valores dentro de un rango especificado.

Estos índices suelen ser más efectivos en estos casos:

1. Búsquedas basadas en igualdad y rangos: La estructura jerárquica del árbol permite realizar búsquedas eficientes, minimizando la cantidad de nodos que deben ser visitados.
2. Mantenimiento eficiente: La estructura balanceada del árbol permite minimizar las reorganizaciones necesarias cuando los datos son modificados.
3. Tablas grandes: A medida que aumenta el tamaño de la tabla, el rendimiento del índice de árbol B sigue siendo eficiente, ya que el número de niveles del árbol se mantiene relativamente bajo.

## ¿Cómo se pueden utilizar índices para optimizar el rendimiento para una carga de trabajo dada?

Es importante considerar que los índices se vuelven útiles cuanto más datos guardados uno tenga, ya que es en bases de datos grandes que los tiempos de respuesta se vuelven un problema. A cambio de reducir el tiempo de respuesta de una consulta, los índices también ocupan memoria, por lo que no se puede simplemente crear un índice para cada columna de cada tabla. Para utilizar índices de manera efectiva se puede tomar en consideración las siguientes sugerencias:

1. **Identificar las consultas más frecuentes**: Analizar las consultas más comunes en la carga de trabajo, que suelen ser candidatas para la creación de índices ya que suelen ser las más importantes para los usuarios. También hay que determinar qué consultas demoran mucho realizando un "full table scan" (es decir que no están aprovechando ningún índice ya existente), ya que se podría agregar un índice para solventar esa demora.
2. **Evaluar las columnas a indexar:** No todas las columnas necesitan ser indexadas. Se debe evaluar cuáles columnas son más relevantes para las consultas críticas y en qué columnas las consultas son más selectivas (es decir, devuelven un subconjunto más pequeño de registros). Indexar estas columnas puede mejorar significativamente el rendimiento.
3. **Considerar el equilibrio entre el rendimiento de lectura y escritura:** Los índices mejoran el rendimiento de las consultas de lectura, pero tienen un impacto en las operaciones de escritura, ya que los índices deben actualizarse. Conviene indexar cuando hay muchas consultas de lectura, y en lo posible evitarlo cuando hay muchas consultas de escritura afectando a la tabla que se desea indexar.
4. **Monitorear y ajustar:** Realizar un monitoreo regular del rendimiento de las consultas y ajustar los índices según sea necesario. A medida que cambia la carga de trabajo o se agregan nuevas consultas, es posible que se requieran modificaciones en los índices existentes o la creación de nuevos índices para adaptarse a los nuevos requisitos.

***1. ¿Qué es una “instancia” en MySQL? ¿Cuál es la diferencia con una “base de datos” o “schema”?***

Una instancia de MySQL es un servidor MySQL corriendo en algún computador, mientras que una base de datos o schema es un contenedor lógico para organizar conjuntos de datos relacionados, y no es un programa en ejecución.

***2. Enumere y dé una breve descripción sobre clasificaciones y tipos de datos soportados por el gestor (capítulos 11.1 a 11.5 del manual de referencia).***

MySQL admite una variedad de clasificaciones y tipos de datos para almacenar y manipular información:

Números enteros:

* TINYINT: Un número entero de 1 byte con signo.
* SMALLINT: Un número entero de 2 bytes con signo.
* INT: Un número entero de 4 bytes con signo.
* BIGINT: Un número entero de 8 bytes con signo.

Números de punto flotante:

* FLOAT: Un número de punto flotante de precisión simple.
* DOUBLE: Un número de punto flotante de precisión doble.

Texto y cadenas de caracteres:

* CHAR: Una cadena de longitud fija con un tamaño máximo especificado.
* VARCHAR: Una cadena de longitud variable con un tamaño máximo especificado.
* TEXT: Un texto de longitud variable con un tamaño máximo más grande que VARCHAR.

Fechas y horas:

* DATE: Almacena una fecha en el formato 'YYYY-MM-DD'.
* TIME: Almacena una hora en el formato 'HH:MM:SS'.
* DATETIME: Almacena una fecha y hora en el formato 'YYYY-MM-DD HH:MM:SS'.
* TIMESTAMP: Almacena una marca de tiempo desde el inicio de la época (1970-01-01 00:00:00) hasta el presente.

Booleanos:

* BOOLEAN o BOOL: Almacena valores de verdadero o falso.

Binarios:

* BINARY: Una cadena de bytes de longitud fija.
* VARBINARY: Una cadena de bytes de longitud variable.

Enumeraciones y conjuntos:

* ENUM: Almacena uno de los valores enumerados definidos.
* SET: Almacena una colección de valores de conjunto definidos.

Decimales de punto fijo:

* DECIMAL: Un número decimal de precisión fija, adecuado para valores monetarios y cálculos precisos.

Conjuntos de caracteres:

* CHAR(n): Una cadena de caracteres de longitud fija que utiliza un conjunto de caracteres específico.
* VARCHAR(n): Una cadena de caracteres de longitud variable que utiliza un conjunto de caracteres específico.
* TEXT: Un texto de longitud variable que utiliza un conjunto de caracteres específico.

Tipos de datos espaciales:

* GEOMETRY: Un tipo de dato para representar datos de geometría plana.
* POINT: Representa un punto en un plano.
* LINESTRING: Representa una secuencia de segmentos de línea.
* POLYGON: Representa una forma cerrada formada por segmentos de línea.
* JSON: Almacena y manipula datos en formato JSON (JavaScript Object Notation)

***3. ¿Qué son los “storage engines” en MySQL? Elabore un concepto y describa brevemente al menos tres de ellos.***

Los "storage engines" o **motores de almacenamiento** en MySQL son componentes que determinan cómo se almacenan, organizan y acceden a los datos en la base de datos. Cada motor de almacenamiento tiene sus propias características, rendimiento, soporte para transacciones y capacidades de indexación. MySQL ofrece varios motores de almacenamiento.

Algunos de los más utilizados en MySQL son:

* **InnoDB**: Es el motor de almacenamiento predeterminado en MySQL desde la versión 5.5. Proporciona soporte para transacciones ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad), integridad referencial y bloqueo de nivel de fila. Es adecuado para aplicaciones que requieren concurrencia y alta confiabilidad.
* **MyISAM**: Es uno de los motores de almacenamiento más antiguos en MySQL. No proporciona soporte para transacciones ni para integridad referencial, pero ofrece un buen rendimiento en operaciones de lectura y escritura intensivas.
* **MEMORY** (también conocido como HEAP): Almacena los datos en memoria principal en lugar de en disco, lo que brinda acceso extremadamente rápido a los datos. Sin embargo, los datos almacenados en la memoria se pierden en caso de un reinicio del servidor o un fallo.

***4. ¿Dónde almacena físicamente las bases de datos MySQL?***

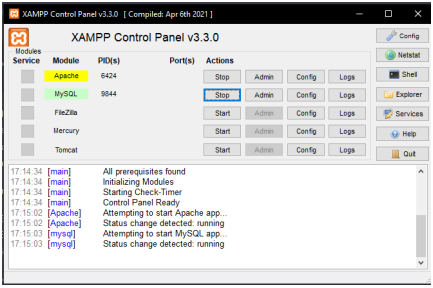
MySQL almacena los registros de las bases de datos en archivos dentro del sistema de archivos del sistema operativo del servidor en el que está instalado. El directorio principal donde se encuentran los archivos de datos de MySQL se conoce como **directorio de datos**. La ubicación del mismo puede variar según el sistema operativo y la configuración.

En general, este directorio suele estar ubicado en la siguiente ruta:

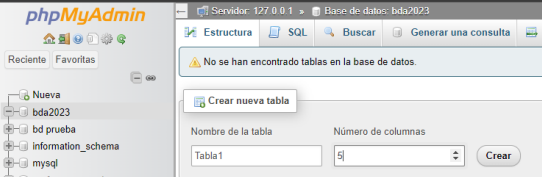
● En sistemas basados en Unix/Linux: /var/lib/mysql/

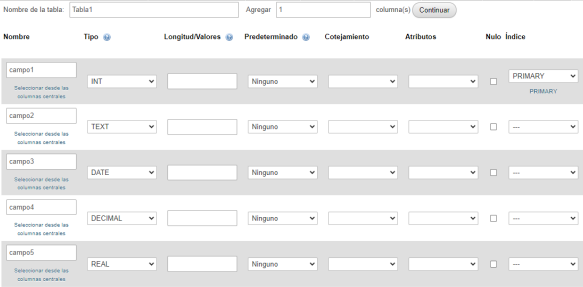
● En sistemas basados en Windows: C:\ProgramData\MySQL\MySQL Server X.X\data\ (donde X.X es la versión de MySQL instalada)

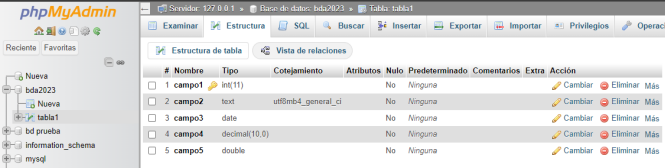
***5. Cree una base de datos, que como único requisito obligatorio debe llamarse “BDA2023”; para ésto investigue qué parámetros opcionales puede establecer y/o configurar para la creación. Escoja, y fundamente, los que considere adecuados. Documente cada paso con una copia de pantalla del asistente de creación o el comando utilizado. Haga lo mismo para la creación de una tabla “Tabla1” con cinco campos de distintos tipos.***

******

******

******

******

******

## 6. Elija al menos cinco conceptos vistos en la clase de teoría de “Introducción a las bases de datos” e identifique y describa brevemente cómo son implementados por MySQL.

## 

## 1. Cardinalidad: en el contexto de una base de datos relacional, la cardinalidad se utiliza para describir la relación entre tablas y la cantidad de registros relacionados entre sí. Existen tres tipos comunes de cardinalidad que se utilizan en MySQL:

## 

## Cardinalidad Uno a Uno (1:1): En una relación uno a uno, un registro en una tabla está relacionado con un solo registro en otra tabla y viceversa. Se puede llevar todos los atributos de una tabla a la otra y eliminarla, o usar restricciones de claves foráneas.

## 

## Cardinalidad Uno a Muchos (1:N): En una relación uno a muchos, un registro en una tabla está relacionado con varios registros en otra tabla, pero los registros en la segunda tabla solo están relacionados con un solo registro en la primera tabla. Se implementa utilizando restricciones de claves foráneas.

## 

## Cardinalidad Muchos a Muchos (N:M): se logra mediante una tabla de unión o tabla de enlace intermedia que relaciona los registros entre las dos tablas.

## 

## 2. Claves primarias: En MySQL, una clave primaria es un campo o combinación de campos que identifica de manera única cada registro en una tabla, y crea un índice primario en base a ese campo.

## En el motor de almacenamiento InnoDB, esto además implica que los registros se van a persistir en el disco ordenados según la columna que es clave primaria.

## 

## Se puede crear una clave primaria de la siguiente manera:

## 

## CREATE TABLE clientes (

## id INT PRIMARY KEY,

## nombre VARCHAR(50),

## email VARCHAR(100) );

## 

## En este caso, la columna "id" se define como la clave primaria de la tabla "clientes". El tipo de datos de la columna puede variar según las necesidades. Al declararla como clave primaria, MySQL garantiza que cada valor en la columna "id" sea único y no nulo.

## Clave compuesta: Podemos definir una clave primaria compuesta utilizando la cláusula PRIMARY KEY en la creación de la tabla, incluyendo varias columnas separadas por comas:

## 

## CREATE TABLE pedidos (

## id\_cliente INT,

## id\_producto INT,

## cantidad INT,

## PRIMARY KEY (id\_cliente, id\_producto) );

## 

## En este caso, la clave primaria está compuesta por las columnas "id\_cliente" y "id\_producto". La combinación de los valores en ambas columnas debe ser única para cada registro en la tabla.

## 

## 3. Tipos de datos: el tipo de un dato refiere al conjunto de valores permitidos para una columna dada de una tabla dada. MySQL proporciona una amplia gama de tipos de datos primitivos que cubren diversos dominios, como números, cadenas de caracteres y fechas. El tipo de dato de un campo impacta cómo es persistido por el SGBD.

## 

## Cuando se crea una tabla en MySQL, se debe especificar el tipo de datos para cada columna:

## CREATE TABLE empleados (

## id INT,

## nombre VARCHAR(50),

## edad TINYINT,

## salario DECIMAL(10, 2),

## fecha\_contrato DATE );

## 

## 4. Transacciones: MySQL le deriva la mayoría de las dificultades de implementar transacciones a los motores de almacenamiento. Es por esto que algunos de ellos no soportan transacciones, y los detalles de cómo se implementan varían de motor a motor, aunque todos los motores deben respetar la misma interfaz para comunicarse con la capa superior del SGBD.

En particular, para mantener aisladas a las transacciones, InnoDB bloquea a nivel de fila. Comúnmente hay dos tipos de bloqueo:

* **Lock compartido**: permite a toda transacción que tenga el lock leer la fila, pero no puede modificarla. Esto permite que muchas transacciones lean la misma fila a la vez, ya que no hay riesgo de que ocurra un "data race".
* **Lock exclusivo**: permite que la transacción lea, modifique y/o elimine la fila, pero no permite que otras transacciones puedan tener el mismo lock. Así, si otra transacción quiere leer la fila que está bloqueada, debe esperar a que la transacción que tiene el lock exclusivo termine.

**5. Control de acceso:** un SGBD tiene varios tipos de usuarios, cada uno con un nivel de acceso distinto. El que más acceso suele tener es el Administrador de la Base de Datos.

MySQL otorga (y quita) privilegios administrativos (para gestionar el servidor MySQL) guardados en las tablas `user` y `global\_grants`, privilegios a nivel de una base de datos guardados en la tabla `db`, y privilegios dentro de una base de datos a nivel de tabla, columna o rutina, guardados en la tablas `tables\_priv`, `columns\_priv` y `procs\_priv`. El usuario "root" se crea al instalar el servidor y es tradicionalmente el que tiene todos los privilegios.

Al intentar conectarse al servidor, el servidor acepta la conexión si ésta trae las credenciales para validar la identidad de la cuenta y si la cuenta no está bloqueada. Esos controles de credenciales los realiza buscando la fila del usuario en una tabla `user`, validando que:

* El name y host recibido coincidan con valores de las columnas `name` y `host`.
* La columna `account\_locked` indique que la cuenta no está bloqueada.
* Si el cliente otorgó unas credenciales (ej: una contraseña), que las haya especificado de acuerdo con la columna `authentication\_string`.

Una vez que el servidor acepta la conexión, verifica cada petición recibida a través de esa conexión. Ante una petición, primero determina qué operación desea realizar el cliente y valida que sus privilegios sean suficientes.