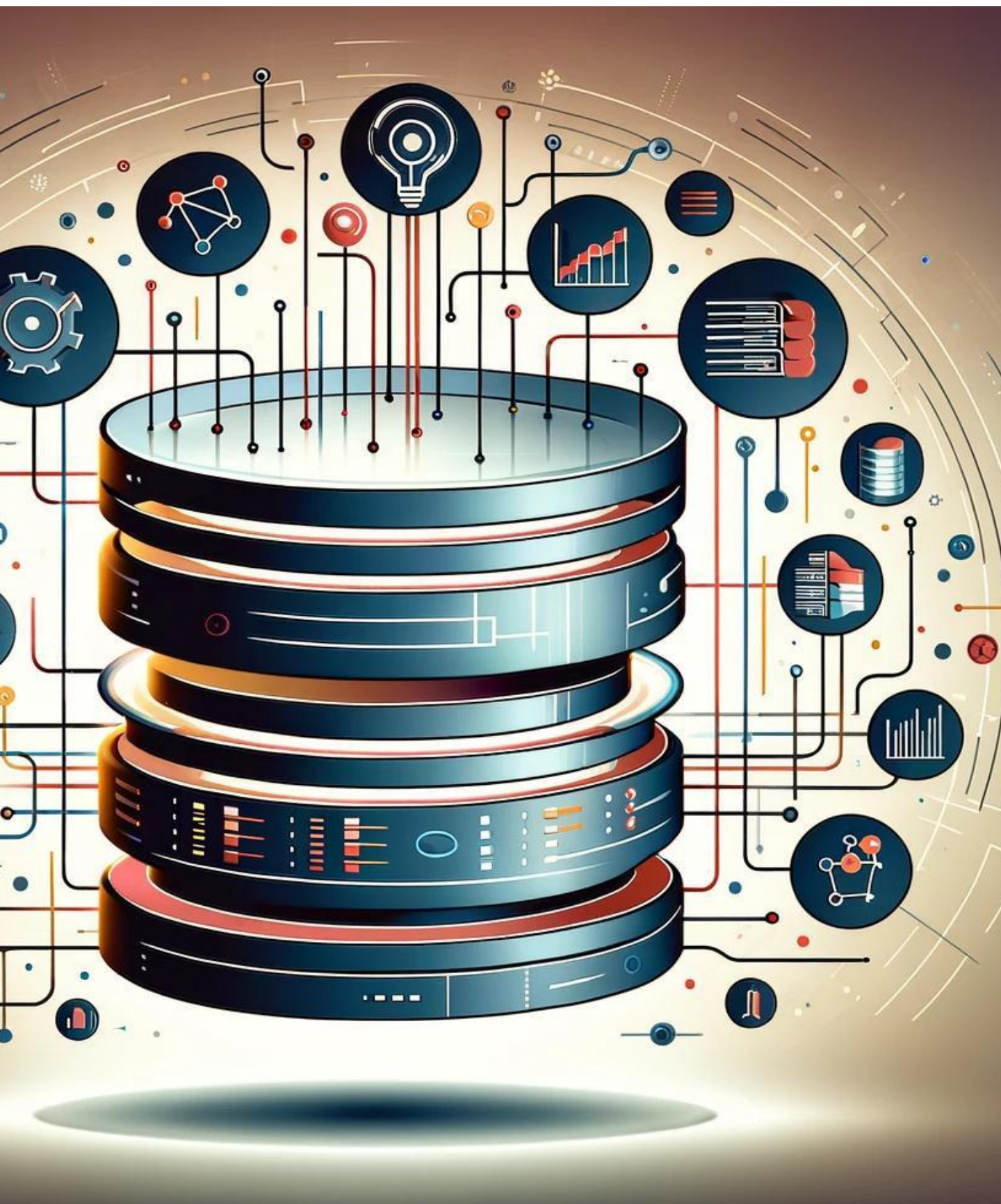




Bases de Datos 1



Alejandra Lliteras
Prof. Titular



Federico Orlando
Prof. Adjunto

TEMAS GENERALES

Bases de Datos 1

1

Modelo de
Datos

2

Teoría de
Diseño de
Bases de
Datos

3

Álgebra
Relacional

4

DBMS
Relacional
MySQL

5

Visualización
de Datos

TEMAS Y SUBTEMAS

Hoy veremos...

11.Funciones

12.Excepciones

13.Cursores

14.Transacciones

15.Triggers

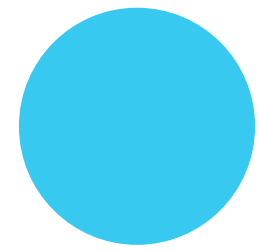
16.Índices

17.Optimización de consultas

18.Seguridad de los datos

Funciones





Una Función es similar a un SP pero se diferencia en:

- **Solo puede recibir parámetros de entrada y retorna siempre un solo valor.**
- **No pueden realizar operaciones que modifiquen datos en la BD.**
- **El valor de retorno puede ser utilizado directamente en consultas SQL**

DML

FUNCTION

DML

FUNCTION

● **LAST_INSERT_ID()** es una función en MySQL que se utiliza para recuperar el ultimo valor generado automáticamente por una columna autoincremental en una tabla.

● Es necesario redefinir el delimitador para evitar confusiones entre los delimitadores de la función y el estándar.

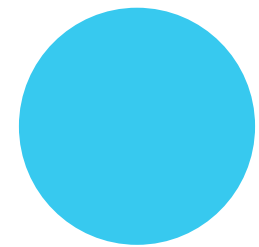
```
DELIMITER //
CREATE FUNCTION obtener_precio (IN p_descripcion VARCHAR) RETURNS FLOAT
BEGIN
    DECLARE FLOAT p_precio;
    SELECT precio INTO p_precio
    FROM bbdd1.tabla_ejemplo
    WHERE descripcion = p_descripcion;
    RETURN p_precio;
END //
DELIMITER ;
```

 Es necesario redefinir el delimitador para evitar confusiones entre los delimitadores de la función y el estándar.

```
DELIMITER //  
CREATE FUNCTION obtener_precio (IN p_descripcion VARCHAR) RETURNS FLOAT  
BEGIN  
    DECLARE FLOAT p_precio;  
    SELECT precio INTO p_precio  
    FROM bbdd1.tabla_ejemplo  
    WHERE descripcion = p_descripcion;  
    RETURN p_precio;  
END //  
DELIMITER ;
```

Excepciones



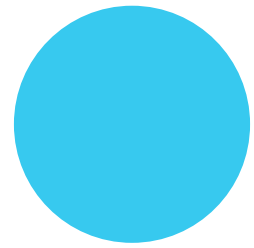


Una excepción es un evento que interrumpe el flujo normal de la ejecución de una consulta o un store procedure, indicando una condición inesperada o un problema al interactuar con la base de datos.

Estos eventos pueden ser causados por errores de sintaxis en las sentencias SQL, intentos de usar palabras reservadas de forma incorrecta o la falta de datos esperados, y se pueden manejar en store procedure para controlar y responder a estos errores de manera controlada.

DML

EXCEPCIONES



Una excepción en un store procedure se pueden manejar con la sentencia **DECLAER HANDLER**.

```
DECLARE handler_action HANDLER FOR condition_value [, condition_value] ...  
statement
```

```
handler_action: { CONTINUE | EXIT }
```

```
condition_value: {  
    mysql_error_code  
    | SQLSTATE [VALUE] sqlstate_value  
    | condition_name  
    | SQLWARNING  
    | NOT FOUND  
    | SQLException  
}
```

DML

EXCEPCIONES

```
DECLARE handler_action HANDLER FOR condition_value [, condition_value] ...  
statement
```

```
handler_action: { CONTINUE | EXIT }
```

- **CONTINUE** – El programa actual continuará la ejecución del procedimiento.
- **EXIT** – Esto finaliza la ejecución del procedimiento.

DML

EXCEPCIONES

DML

EXCEPCIONES

```
DECLARE handler_action HANDLER FOR condition_value [, condition_value] ...  
statement  
condition_value: {  
    mysql_error_code  
| SQLSTATE [VALUE] sqlstate_value  
| condition_name  
| SQLWARNING  
| NOT FOUND  
| SQLEXCEPTION  
}
```

● **mysql_error_code** : este es un literal entero que indica el código de error.

```
DECLARE CONTINUE HANDLER FOR 1051  
BEGIN  
    -- sentencias  
END;
```

En este caso 1051 significa tabla desconocida.

Listado de código de errores en:

<https://dev.mysql.com/doc/mysql-errors/9.4/en/server-error-reference.html>

DML

EXCEPCIONES

- **sqlstate_value** : es una cadena literal de 5 caracteres que especifica el valor **SQLSTATE**.

```
DECLARE CONTINUE HANDLER FOR SQLSTATE '42S02'  
BEGIN  
    -- sentencias  
END;
```

En este caso 42S02 significa tabla desconocida.

- **condition_name** – El nombre de la condición definida por el usuario especificada con **DECLARE ... CONDITION**.

```
DECLARE tabla_no_existente CONDITION FOR SQLSTATE '42S02';  
DECLARE CONTINUE HANDLER FOR tabla_no_existente  
BEGIN  
    -- sentencias  
END;
```

DML

EXCEPCIONES

- **SQLWARNING** – El valor abreviado para **SQLSTATE** comienza con '01'.

```
DECLARE CONTINUE HANDLER FOR SQLWARNING
BEGIN
    -- sentencias
END;
```

- **NOT FOUND** – El valor abreviado para **SQLSTATE** comienza con '02'.

```
DECLARE CONTINUE HANDLER FOR NOT FOUND
BEGIN
    -- sentencias
END;
```

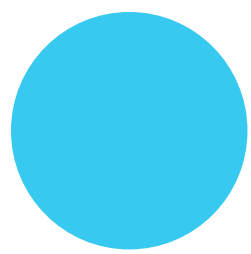
Esto es relevante en el uso de cursores.

● **SQLException** : valor abreviado para **SQLState** que no comienza con '00', '01', '02'.

```
DECLARE CONTINUE HANDLER FOR SQLException
BEGIN
    -- sentencias
END;
```

DML

EXCEPCIONES



Si ocurre una condición para la cual no se ha declarado ningún HANDLER, dependerá de la clase de condición la acción que se ejecutará:

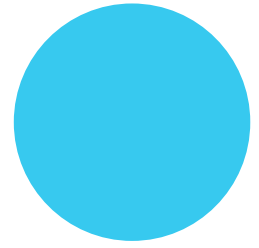
- **SQLException:** la ejecución termina en la declaración que generó la condición, como si hubiera un HANDLER EXIT.
- **SQLWarning:** continúa ejecutándose, como si hubiera un HANDLER CONTINUE.
- **NOT FOUND:** si la condición se generó normalmente, la acción es CONTINUE. Si fue planteado por SIGNAL o RESIGNAL, la acción es EXIT.

DML

EXCEPCIONES

Cursores



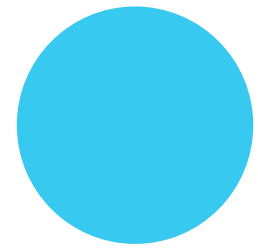


Un cursor es una estructura que se utilizar para almacenar las tuplas obtenidas al ejecutar una consulta SQL.

Se los puede recorrer e ir obteniendo de a una tupla por vez.

DML

CURSORES

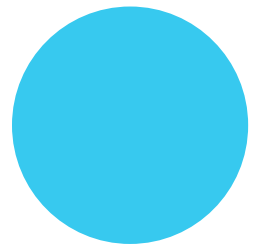


Sobre un cursor pueden realizarse las siguientes operaciones:

- **DECLARE:** declara un cursor
- **OPEN:** abre un cursor previamente declarado
- **FETCH:** recupera un valor de un cursor previamente abierto.
- **CLOSE:** cierra un cursor declarado previamente.

DML

CURSORES



Veamos un ejemplo:

```
DELIMITER //
CREATE PROCEDURE concatenar_tabla_ejemplo()
BEGIN
    DECLARE aux_descripcion VARCHAR(50);
    DECLARE aux_precio FLOAT;
    DECLARE fin INT DEFAULT 0;
    DECLARE cursor_ejemlo CURSOR FOR SELECT descripcion,precio
    FROM bbdd1.tabla_ejemplo te;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET fin = 1;
    OPEN cursor_ejemlo;
    loop_cursor : LOOP
        FETCH NEXT FROM cursor_ejemlo INTO aux_descripcion,aux_precio;
        IF fin = 1 THEN
            LEAVE loop_cursor;
        END IF;
        INSERT INTO bbdd1.ejemplo_concatenado(ej_concatenado)
        VALUES (CONCAT(aux_descripcion,' $', aux_precio));
    END LOOP;
    CLOSE cursor_ejemlo;
END//
DELIMITER ;
```

DML

CURSORES

```
DELIMITER //
CREATE PROCEDURE concatenar_tabla_ejemplo()
BEGIN
    DECLARE aux_descripcion VARCHAR(50);
    DECLARE aux_precio FLOAT;
    DECLARE fin INT DEFAULT 0;
    DECLARE cursor_ejemlo CURSOR FOR SELECT descripcion,precio
    FROM bbdd1.tabla_ejemplo te;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET fin = 1;
    OPEN cursor_ejemlo;
    loop_cursor : LOOP
        FETCH NEXT FROM cursor_ejemlo INTO aux_descripcion,aux_precio;
        IF fin = 1 THEN
            LEAVE loop_cursor;
        END IF;
        INSERT INTO bbdd1.ejemplo_concatenado(ej_concatenado)
        VALUES (CONCAT(aux_descripcion, ' $', aux_precio));
    END LOOP;
    CLOSE cursor_ejemlo;
END//
DELIMITER ;
```

Transacciones



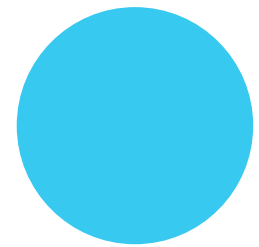
TRANSACCIONES

- **Dentro de un Store Procedure puedo manejar transacciones, que es un conjunto de sentencias que forman una unidad lógica de trabajo.**
- **Las transacciones deben cumplir con las propiedades ACID.**
- **Las transacciones se inician con las sentencias `START TRANSACTION`, y se finalizan con las sentencia `COMMIT` o `ROLLBACK`.**

TRANSACCIONES

Propiedades ACID

- **ATOMICIDAD:** todas las operaciones de la transacción se ejecutan o no lo hace ninguna de ellas.
- **CONSISTENCIA:** la ejecución aislada de la transacción conserva la consistencia de la BD.
- **AISLAMIENTO (ISOLATION):** cada transacción ignora el resto de las transacciones que se ejecutan concurrentemente en el sistema.
- **DURABILIDAD:** una transacción finalizada con éxito realiza cambios permanentes en la BD, incluso si hay fallos en el sistema.



Veamos un ejemplo

```
DELIMITER //
```

```
CREATE PROCEDURE actualizar_precio_con_tx (IN p_id INTEGER, IN nuevo_precio FLOAT)
```

```
BEGIN
```

```
    -- INICIAMOS LA TRANSACCIÓN
```

```
    START TRANSACTION;
```

```
    UPDATE bbdd1.tabla_ejemplo
```

```
    SET precio=nuevo_precio
```

```
    WHERE id_tabla_ejemplo = p_id;
```

```
    -- CONFIRMAMOS LA TRANSACCIÓN
```

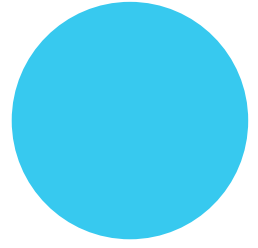
```
    COMMIT;
```

```
END //
```

```
DELIMITER ;
```

TRANSACCIONES

Propiedades ACID



Veamos un ejemplo

```
DELIMITER //
CREATE PROCEDURE actualizar_precio_con_tx (IN p_id INTEGER, IN nuevo_precio FLOAT)
BEGIN
    -- INICIAMOS LA TRANSACCIÓN
    START TRANSACTION;

    UPDATE bdd1.tabla_ejemplo
    SET precio=nuevo_precio
    WHERE id_tabla_ejemplo = p_id;

    -- CONFIRMAMOS LA TRANSACCIÓN
    COMMIT;
END //
DELIMITER ;
```

Triggers

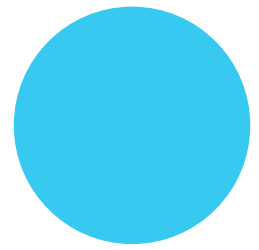


TRIGGERS

Un disparador(trigger) es un objeto con nombre dentro de una base de datos el cual se asocia con una tabla y se activa cuando ocurre en ésta un evento en particular, permitiendo automatizar acciones o aplicar una lógica ante el cambio producido.

```
CREATE TABLE cuenta (cuenta_id INT, cantidad DECIMAL(10,2));  
CREATE TRIGGER ins_sum  
BEFORE INSERT ON cuenta  
FOR EACH ROW SET @sum = @sum + NEW.cantidad;
```

No puede haber dos disparadores en una misma tabla que correspondan al mismo momento y sentencia.



Un disparador(trigger) es un objeto con nombre dentro de una base de datos el cual se asocia con una tabla y se activa cuando ocurre en ésta un evento en particular, permitiendo automatizar acciones o aplicar una lógica ante el cambio producido.

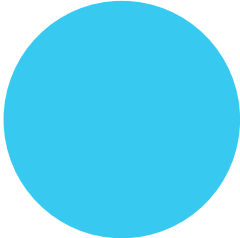
```
CREATE TABLE cuenta (cuenta_id INT, cantidad DECIMAL(10,2));  
CREATE TRIGGER ins_sum  
BEFORE INSERT ON cuenta  
FOR EACH ROW SET @sum = @sum + NEW.cantidad;
```


TRIGGERS

Un disparador(trigger) es un objeto con nombre dentro de una base de datos el cual se asocia con una tabla y se activa cuando ocurre en ésta un evento en particular, permitiendo automatizar acciones o aplicar una lógica ante el cambio producido.

```
CREATE TABLE cuenta (cuenta_id INT, cantidad DECIMAL(10,2));  
CREATE TRIGGER ins_sum  
BEFORE INSERT ON cuenta  
FOR EACH ROW SET @sum = @sum + NEW.cantidad;
```

No puede haber dos disparadores en una misma tabla que correspondan al mismo momento y sentencia.



```
CREATE TRIGGER nombre_trigger
{BEFORE | AFTER} {INSERT | UPDATE | DELETE}
ON nombre_tabla FOR EACH ROW
BEGIN ...
...
END;
```

- {BEFORE | AFTER} indica cuándo se activa el trigger, antes o después de la operación
- {INSERT | UPDATE | DELETE} es la operación que dispara el trigger
- FOR EACH ROW indica que se ejecutará una vez por cada tupla afectada
- BEGIN y END delimitan el cuerpo del trigger con las operaciones

TRIGGERS



Hay limitaciones sobre lo que puede aparecer dentro de la sentencia que el trigger ejecutará al activarse:

- **El trigger no puede referirse a tablas directamente por su nombre, incluyendo la misma tabla a la que está asociado. Sin embargo, se pueden emplear las palabras clave OLD y NEW. OLD se refiere a un registro existente que va a borrarse o que va a actualizarse antes de que esto ocurra. NEW se refiere a un registro nuevo que se insertará o a un registro modificado luego de que ocurre la modificación.**

TRIGGERS



Hay limitaciones sobre lo que puede aparecer dentro de la sentencia que el trigger ejecutará al activarse:



- **El trigger no puede invocar procedimientos almacenados utilizando la sentencia CALL. (Esto significa, por ejemplo, que no se puede utilizar un procedimiento almacenado para eludir la prohibición de referirse a tablas por su nombre).**
- **El disparador no puede utilizar sentencias que inicien o finalicen una transacción.**

TRIGGERS

TRIGGERS



Una columna OLD es de solo lectura y requiere privilegios de SELECT

-  **Una columna de NEW requiere privilegio de SELECT.**
-  **Es posible una columna NEW con BEFORE, para ello es necesario el privilegio de UPDATE.**



Para poder manipular un trigger, un usuario necesita el permiso: TRIGGER

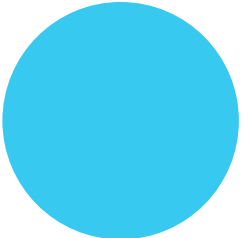
Índices



INDICES

- **Son archivos auxiliares que utiliza el DBMS para recuperar registros según algunos criterios de ordenación y facilitar las búsquedas. Generalmente es una estructura en forma de árbol que puede organizarse por más de una columna de la tabla.**
- **Un DBMS relacional utiliza los índices de acuerdo con los criterios que establece el optimizador de consultas.**

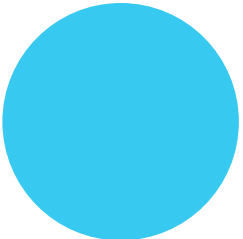
INDICES



Si no existe un índice, MySQL deberá empezar por el primer registro e ir leyendo registro a registro en forma secuencial toda la tabla para encontrar las filas relevantes. Cuanto más grande es la tabla, más costoso es este proceso.



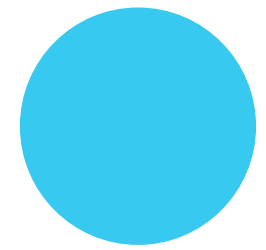
Si una tabla tiene 1000 registros y un índice, la búsqueda es por lo menos 100 veces más rápido que la lectura secuencial.



Si se requiere la mayor parte de la tabla, es más rápida la lectura secuencial porque requiere menos accesos a disco.

INDICES

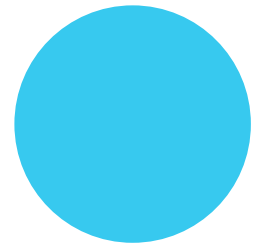
- **La principal ventaja es que aceleran las búsquedas, ordenaciones, selección, agrupación (totalización) según las claves de indexación.**
- **La principal desventaja es que penalizan las actualizaciones (hay que actualizar más archivos).**
- **Crear índices sobre una tabla NO siempre significa más velocidad en las consultas.**



Los tipos más comunes son:

- **Primary Key Index**
- **Unique Index**
- **Foreign Key Index**
- **Full Text Index**
- **Spatial Index**
- **Prefix Index**
- **Composite Index**
- **Memory Index**

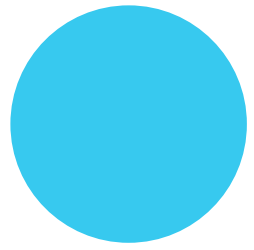
INDICES



Cuándo se crean los índices?

- **Automáticamente**
Un índice único se crea automáticamente cuando se define una restricción PRIMARY KEY o UNIQUE. El nombre del índice corresponde al nombre dado a la restricción
- **Manualmente**
Los usuarios pueden crear manualmente índices únicos y no únicos en las columnas de una tabla para mejorar la velocidad de acceso a las filas

INDICES

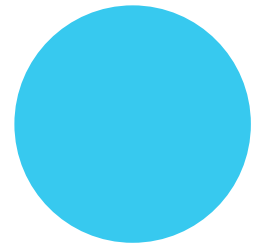


Para crear un índice existe la sentencia
CREATE INDEX

```
CREATE <tipo> INDEX nombre ON tabla (<columnas>);
```

```
CREATE INDEX idx_precio ON bbdd1.tabla_ejemplo (precio DESC);
```

INDICES






Para crear un índice existe la sentencia
CREATE INDEX

```
CREATE <tipo> INDEX nombre ON tabla (<columnas>);
```

```
CREATE INDEX idx_precio ON bbdd1.tabla_ejemplo (precio DESC);
```



INDICES

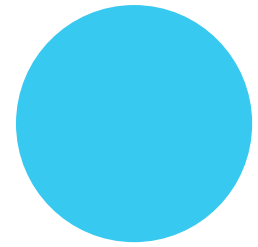
 Se debe crear un índice si:

-  La columna contiene un amplio rango de valores
-  La columna contiene una gran cantidad de valores NULL y las consultas buscan un valor determinado (\neq de NULL)
-  Una o más columnas son frecuentemente usadas en una cláusula WHERE o una condición de JOIN. Es normal crear índices en columnas con restricción de llave foránea.

INDICES

 Se debe crear un índice si (cont):

-  La tabla es muy grande y se espera que la mayoría de las consultas recuperan menos del 15% de las filas de la tabla
-  Si se realiza un order by o group by de la tabla y el agrupado o ordenamiento es realizado sobre un prefijo de una clave (por ejemplo, ORDER BY parte1_clave, parte2_clave)



Normalmente NO es apropiado crear un índice si:

- **La tabla es pequeña (256/512 filas)**
- **La columna o columnas NO son usadas frecuentemente en una cláusula WHERE o una condición de JOIN o lo hacen parcialmente.**
- **La mayoría de consultas por la columna recuperan más del 15% de las filas de la tabla**
- **La tabla es modificada frecuentemente**

INDICES

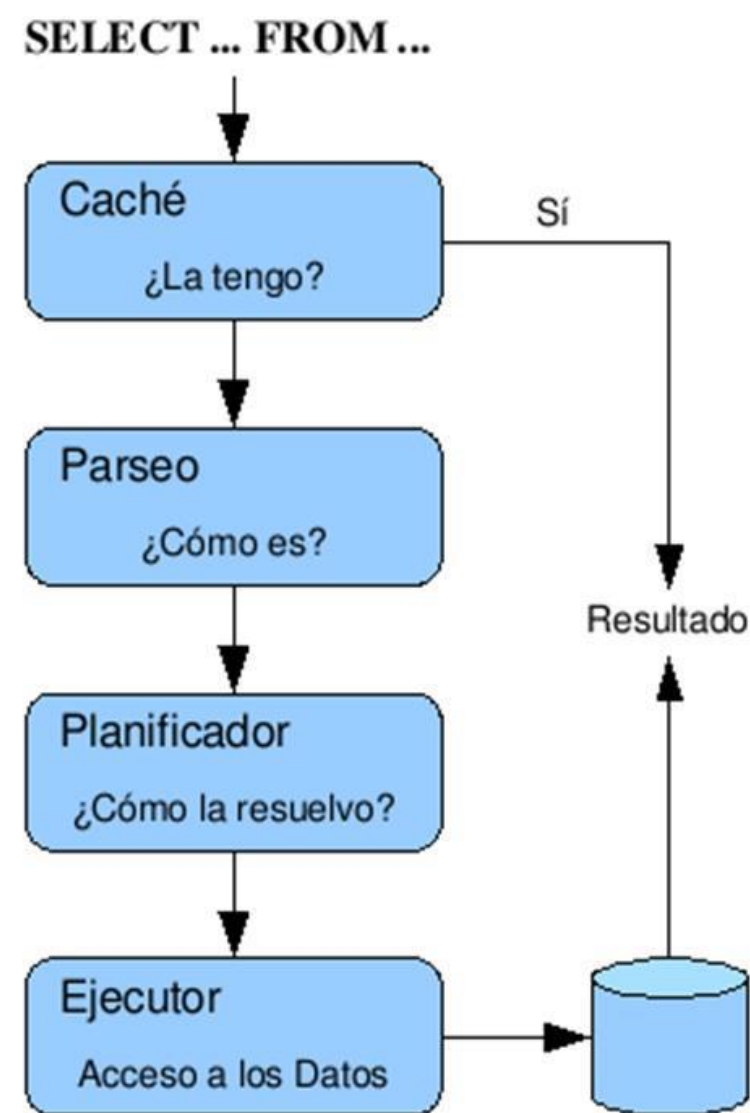
Descanso



Optimización de consultas



Cuando realizamos una consulta a la base de datos, el DBMS realiza un plan para su ejecución.

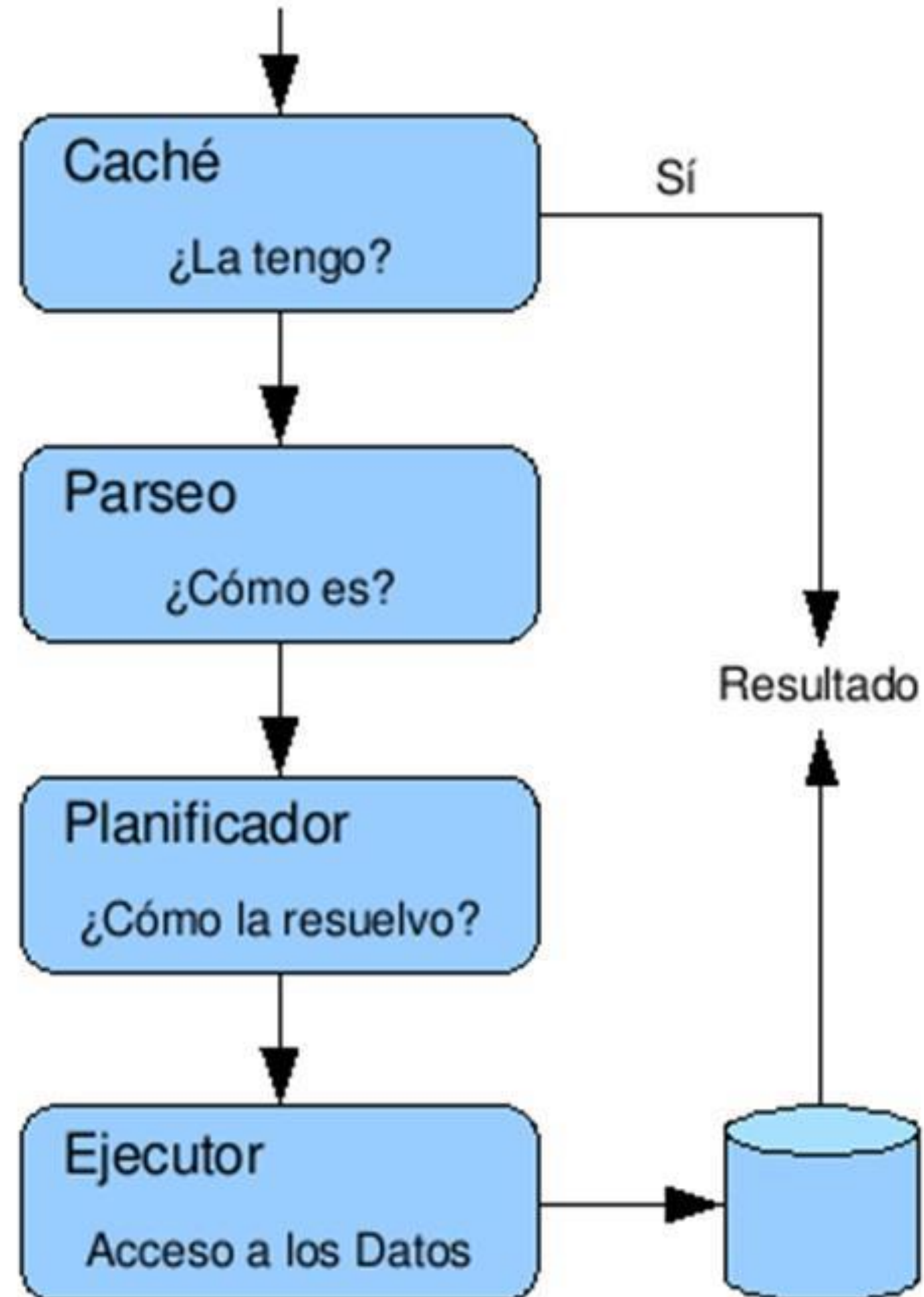


- **Este se realiza en el mismo proceso de compilación**
- **Analiza y optimiza la consulta a través de una serie de estadísticas que genera y recopila el mismo DBMS sobre los datos**

- **Se logra una secuencia de operaciones para realizar la consulta lo más eficiente posible**

Optimización de consultas

SELECT ... FROM ...

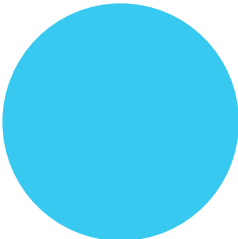


Optimización de consultas

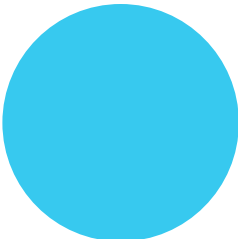
Optimización de consultas

- Utilizando el comando **EXPLAIN SELECT** se puede analizar el plan de ejecución. Es decir, MySQL explica cómo procesaría el **SELECT**, proporcionando también información acerca de cómo y en qué orden están unidas (join) las tablas.
- **EXPLAIN** es una ayuda para decidir qué índices agregar a las tablas, con el fin de que las sentencias **SELECT** encuentren registros más rápidamente.

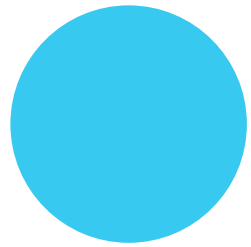
Optimización de consultas



Si un índice no está siendo utilizado por las sentencias **SELECT** cuando debería utilizarlo, debe ejecutarse el comando **ANALYZE TABLE**, a fin de actualizar las estadísticas de la tabla como la cardinalidad de sus claves, que pueden afectar a las decisiones que el optimizador toma.



Una buena aproximación sobre que tan bueno es el join se obtiene al multiplicar los valores del campo **rows** del **explain**. Esto proporciona una idea de cuantas filas debe examinar MySQL para ejecutar la consulta.



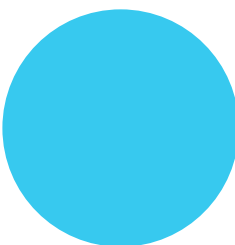
EXPLAIN retorna una tabla; cada línea de esta tabla muestra información acerca de una tabla, y tiene las siguientes columnas entre otras:

```
EXPLAIN SELECT *  
FROM bbdd1.tabla_para_join tpj RIGHT JOIN bbdd1.tabla_ejemplo te  
ON tpj.id_ejemplo = te.id_tabla_ejemplo;
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	te		ALL					5	100.0	
1	SIMPLE	tpj		ALL	fk_tabla_para_join_tabla_ejemplo				2	100.0	Using where; Using join buffer (hash join)

- id
- select_type
- table
- type
- possible_keys
- key
- rows
- filtered
- extra

Optimización de consultas



EXPLAIN retorna una tabla; cada línea de esta tabla muestra información acerca de una tabla, y tiene las siguientes columnas entre otras:

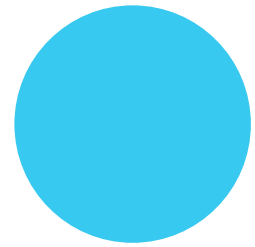
```
EXPLAIN SELECT *  
FROM bbdd1.tabla_para_join tpj RIGHT JOIN bbdd1.tabla_ejemplo te  
ON tpj.id_ejemplo = te.id_tabla_ejemplo;
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	te		ALL					5	100.0	
1	SIMPLE	tpj		ALL	fk_tabla_para_join_tabla_ejemplo				2	100.0	Using where; Using join buffer (hash join)

Optimización de consultas

select_type:

- **SIMPLE** (no contiene UNION o subconsultas)
- **PRIMARY** (select primario, más externo)
- **UNION**
- **DEPENDENT UNION** (depende de una consulta externa)
- **UNION RESULT**
- **SUBQUERY**
- **DEPENDENT SUBQUERY** (depende de una consulta externa en una subconsulta)
- **DERIVED** (subconsulta en un FROM)



type: tipos de joins.

- **system: la tabla tiene una fila. Son tablas de sistema.**
- **const: la tabla tiene una sola fila que coincide. Los valores de las columnas pueden verse como constantes por el optimizador. Es rápido porque son leídas una vez. Aparece cuando se compara todas las partes de una PK o un índice unique con valores constantes.**

Optimización de consultas

Optimización de consultas

type: tipos de joins (cont).

- **eq_ref:** Una fila es leída de la tabla para cada combinación de filas de las tablas anteriores. Esto es el mejor tipo de join posible. Es usado cuando todas las partes de un índice son usadas por el join y el índice es un índice PK o unique. Normalmente se da con el operador =.
- **ref:** Varias filas son leídas para cada combinación de filas de las tablas anteriores. Es usado cuando el join usa una parte del índice o el índice no es de pk o unique.

Optimización de consultas

type: tipos de joins (cont).

- **ref_or_null: ídem ref pero contemplando filas que contengan valores null.**
- **unique_subquery: Es de la forma valor in (subconsulta) usando una clave primaria.**
- **index_subquery: ídem anterior pero por un índice no unique.**



type: tipos de joins (cont).

- **range: compara una clave con una constante usando alguno de los siguientes operadores: =, <>, >, >=, <, <=, IS NULL, <=>, BETWEEN, or IN.**
- **index: examina todo el índice.**
- **all: examina toda la tabla.**

Optimización de consultas

Optimización de consultas

- **possible_keys:** indica que índices pueden ser usados para encontrar una fila en la tabla. Si no tiene valor, significa que no hay índices relevantes.
- **key:** indica que índice decidió usar. Es nulo si no eligió índice.
- **key_lenght:** indica la longitud del índice escogido.
- **ref:** indica que columnas o constantes fueron usadas con la clave para seleccionar las filas de la tabla.

Optimización de consultas

rows: indica la cantidad de filas estimadas necesarias en la ejecución.

extra: proporciona información adicional sobre como se ejecuta la consulta. Los posibles valores son: distinct, not exists, range checked for each record (index map: #), Using filesort, Using index, Using temporary, Using where, Using sort_union(...), Using union(...), Using intersect(...), Using index for group-by

Seguridad de los Datos



Triada de Seguridad

● C → Confidencialidad

● I → Integridad

● D → Disponibilidad

Consola

```
a@acme4:~$ sudo mysql -u root -p
[sudo] password for a:
Enter password: root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3
Server version: 5.7.29-0ubuntu0.18.04.1 (Ubuntu)

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

Creación de Base de Datos

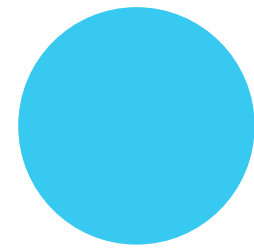
```
CREATE DATABASE produccion;  
GRANT ALL PRIVILEGES ON DATABASE produccion TO tomas;  
GRANT ALL PRIVILEGES ON DATABASE produccion TO juan;
```

```
CREATE TABLE user (  
    id INT NOT NULL AUTO_INCREMENT,  
    name VARCHAR(255) NOT NULL,  
    email VARCHAR(255) NOT NULL,  
    password VARCHAR(255) NOT NULL,  
    PRIMARY KEY (id)  
);
```

```
CREATE TABLE user (  
  id INT NOT NULL AUTO_INCREMENT,  
  name VARCHAR(255) NOT NULL,  
  email VARCHAR(255) NOT NULL,  
  password VARCHAR(255) NOT NULL,  
  PRIMARY KEY (id)  
);
```

USER			
id	name	email	password
1	Juan	juan@gmail.com	juancarlos123
2	Carmelo	carmelo@hotmail.com	43831
3	Timoteo	timoteo@yahoo.com.ar	enero2020!

Respaldo y restauración



**BACKUP DATABASE
produccion TO DISK =
'C:\Backups\produccion.bak';**

**RESTORE DATABASE prueba
FROM DISK =
'C:\Backups\produccion.bak';**

Mala configuración

Como pudimos ver en la ventana de acceso al servicio de MYSQL el usuario root utilizaba root como contraseña.

```
a@acne4:~$ sudo mysql -u root -p
[sudo] password for a:
Enter password: root
```

Uno de los errores más comunes a la hora de generar un ambiente de trabajo con bases de datos es no utilizar contraseñas fuertes para el personal que lo administrará.

De todas formas, el usuario root jamás debe ser utilizado salvo en casos de emergencia. Cada uno de los técnicos que administre el ambiente deberá utilizar su propio usuario.

Roles y permisos

¿Es realmente necesario que tanto el usuario de Tomás como el usuario de Juan tengan todos los derechos sobre la base de datos?

MySQL provee roles y permisos que nos permiten otorgar solamente las funcionalidades necesarias a los usuarios.

El gran problema de otorgar todos los permisos a todos los usuarios de forma indiscriminada es que cualquiera podría borrar la base o información allí almacenada.

Por tiempo muchas veces se otorgan todos los permisos a todos los usuarios

Encriptación

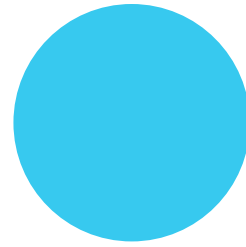
¿No resulta extraño que se pueda visualizar la contraseña de todos los usuarios en texto plano?

Uno de los grandes desafíos es entender la necesidad de contar con la información encriptada para evitar de esta manera que ningún usuario pueda acceder a información que no le pertenece.

USER			
id	name	email	<u>password</u>
1	Juan	juan@gmail.com	2b9965262771630d3d0567b191539560a66565708a54101658e62016b857804c
2	Carmelo	carmelo@hotmail.com	9013a67662556319008460053649934c4716043e1989134647763924818368f
3	Timoteo	timoteo@yahoo.com.ar	605032013c17479541771e5208e65280402046e534509931250537915091388

Encriptar información sensible también es cuidar a los profesionales que acceden a la base de datos

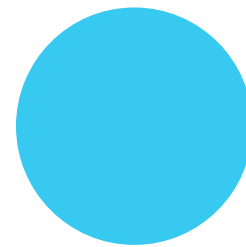
Encriptación



AES

Operaciones con matrices donde se realizan una cierta cantidad de rondas donde los datos son modificados en base a una clave de encriptación.

Las operaciones que se realizan sobre estas matrices son SUSTITUCION, CAMBIO DE FILAS, MEZCLA entre otras.



SHA-2

Conjunto de funciones hash criptográficas.

Existen varias subcategorías de SHA-2 y esto depende de la extensión del hash resultante.

Ambiente de desarrollo y prueba

El usuario que administra la base de datos realizó un respaldo y luego le pidieron que genere una base de datos de prueba con información.

Para no perder tiempo restauró un respaldo que tenía de la base de datos de producción en un ambiente de prueba.

¿El ambiente de prueba está igual de protegido que el de producción?

¿La información almacenada sigue encriptada?

Ambiente de desarrollo y prueba

Uno de los objetivos preferidos son las bases de pruebas ya que muchas veces se llenan con información que se extrae de las bases de producción. Esta práctica se realiza para ahorrar tiempo y “ser lo más realista posible” pero sin todas las medidas de protección que tiene la infraestructura de producción.

Si se va a utilizar información de producción deberá tener el mismo tratamiento por más que sea de prueba, si es posible siempre es preferible llenar las bases de datos con información falsa.

Respaldo

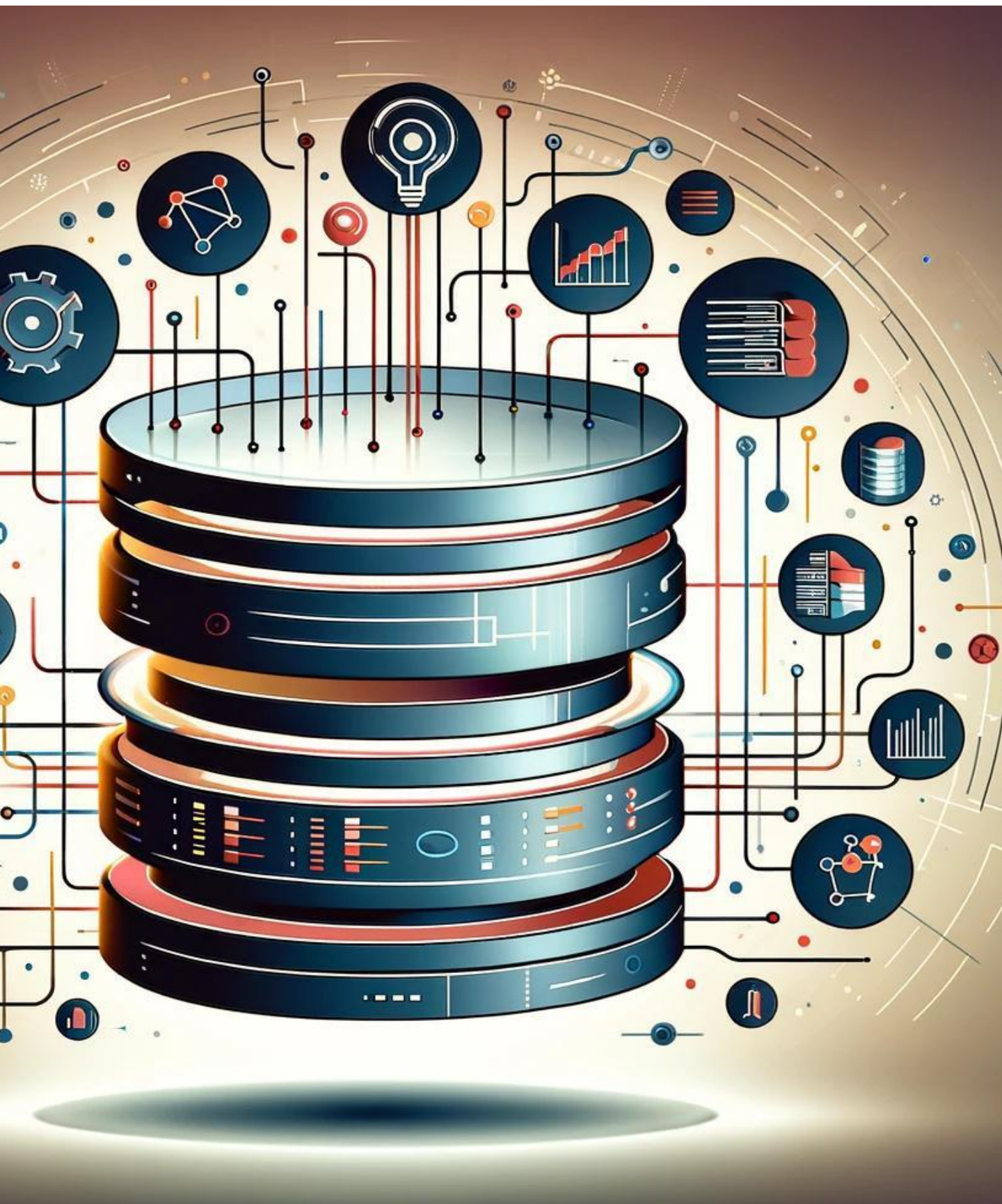
En una diapositiva anterior se vio como realizar un backup utilizando comandos de MYSQL.

Pero ¿está del todo bien?

Como medida principal está bien la realización de respaldo de las distintas bases de datos, pero no es suficiente.

¿Qué pasa si el equipo físico o virtual donde están las bases y los respaldos se infecta?

Una medida adicional para la protección de información podría ser almacenar el respaldo en otro equipo o también en algún medio fuera de línea. Como medida recomendada deberán ser las dos opciones anteriores las que se deban desarrollar.



Bibliografía de la clase

Bibliografía

- <https://dev.mysql.com/doc/refman/9.4/en/>

Importante!



Los slides usados en las clases teóricas de esta materia, no son material de estudio por sí solos en ningun caso.