

**CFP**  
**Programador**  
**full-stack**

# Base de Datos

## CFP Programador full-stack

*Seguridad y Transacciones en Bases de Datos*

# Introducción

- La seguridad de las bases de datos se refiere a la protección frente a **accesos malintencionados**
- Es posible controlar el acceso a la base de datos brindando la autorización adecuada
- Los datos guardados en la base de datos deben estar protegidos contra:
  - accesos no autorizados
  - destrucción o alteración malintencionadas
  - introducción accidental de inconsistencias

# Seguridad en Base de Datos

## *Tipos de Amenazas*



- **Pérdida de integridad.** La integridad se pierde si se realizan cambios no autorizados en los datos mediante acciones intencionadas o accidentales
- **Pérdida de disponibilidad.** Se refiere a que los objetos estén disponibles para un usuario humano o para un programa que tenga los derechos correspondientes
- **Pérdida de confidencialidad.** La confidencialidad de la base de datos tiene relación con la protección de los datos frente al acceso no autorizado

# Seguridad en Base de Datos

## *Niveles de Seguridad*

- **Sistema de bases de datos.** Autorización al acceso a una parte limitada de la base de datos
- **Sistema operativo.** La debilidad de la seguridad del sistema operativo puede servir como medio para el acceso no autorizado
- **Red.** La seguridad en el nivel del software de la red es tan importante en Internet y en las redes privadas de las empresas
- **Físico.** Los sitios que contienen los sistemas informáticos deben estar protegidos de intrusos
- **Humano.** Los usuarios deben ser autorizados cuidadosamente para reducir la posibilidad de intrusión



# Seguridad en Base de Datos

## *Autenticación y Autorización*

- La **autenticación** es el proceso por el cual se identifica un usuario como válida para posteriormente acceder a ciertos recursos definidos
  - Relacionado con la *gestión de usuarios y control de acceso* al DBMS
- La **autorización** es el proceso sobre el cual se establecen qué tipos de recursos están permitidos o denegados para cierto usuario o grupo de usuarios concreto
  - Relacionado con *permisos* (lectura, escritura) para cada usuario autenticado

# Seguridad en Base de Datos

## *Cifrado de Datos - Conceptos*

- El cifrado de datos se utiliza para proteger datos confidenciales como los números de las tarjetas de crédito y contraseñas
- El cifrado se puede utilizar también para proporcionar protección adicional a partes confidenciales de la base de datos
- Los datos se codifican utilizando algún algoritmo de codificación o cifrado
- Un usuario no autorizado que acceda a datos codificados tendrá dificultades para descifrarlos, pero a los usuarios autorizados se les proporcionarán claves para descifrar los datos

# Seguridad en Base de Datos

## *Mecanismos de Seguridad (MySQL)*

- Administración de cuentas de usuarios
  - Creación, modificación y borrado de cuentas de usuario
- Gestión de permisos
  - Otorgamiento, Modificación y Revocación de privilegios
- Conexiones seguras
  - Tipos de conexiones y soporte para SSL



# Mecanismos de Seguridad

## *Administración de Usuarios*

- Un usuario MySQL se define en términos de un nombre de usuario y una contraseña/password
- Por defecto, el motor de base de datos crea un usuario con permisos para todas las tablas de la base de datos
  - El superusuario se denomina ROOT
  - Se recomienda limitar el uso de ROOT a la gestión del DBMS y no usarlo en aplicaciones de producción

# Mecanismos de Seguridad

## *Creación de Usuarios*

- **CREATE USER** 'nombre\_usuario'@'host' **IDENTIFIED BY** 'tu\_contraseña';
- Seguido al nombre de usuario, se debe especificar la IP desde donde podrá realizar conexiones a la base de datos el usuario creado. Puede ser:
  - 'localhost' o '127.0.0.1', desde la misma PC en la que se encuentre instalado MySQL, es decir el host local
  - '192.168.1.100', sólo permite conexiones desde dicha IP (utilizada para identificar a un PC en un LAN)
  - '%' es un comodín que permite conexiones desde cualquier IP

# Mecanismos de Seguridad

## *Concesión de Privilegios a Usuarios*

- **GRANT** [**permiso**] **ON** [nombre de bases de datos]. [nombre de tabla] **TO** '[usuarioX]'@'host' [**WITH GRANT OPTION**];
- **Tipos de Permisos**
  - **ALL** : esta opción otorga todos los permisos
  - **CREATE**: permite crear nuevas tablas o bases de datos
  - **DROP**: permite eliminar tablas o bases de datos
  - **DELETE**: permite eliminar registros de tablas
  - **INSERT**: permite insertar registros en tablas
  - **SELECT**: permite leer registros en las tablas
  - **UPDATE**: permite actualizar registros seleccionados en tablas
  - **WITH GRANT OPTION**: permite que **usuarioX** maneje privilegios de otros usuarios



# Mecanismos de Seguridad

## *Revocar Privilegios a Usuarios*

- **REVOKE** [permisos] **ON** [nombre de base de datos] [nombre de tabla] **FROM** '[nombre de usuario]' @'host';

Una vez que se haya finalizado con la configuración de privilegios (GRANT o REVOKE) se deben refrescar todos los con el comando:

- **FLUSH PRIVILEGES;**

# Mecanismos de Seguridad

## *Privilegios a Usuarios - Ejemplos*

- **GRANT ALL ON** midb.usuarios **TO** 'juan'@'%'  
**WITH GRANT OPTION;**
- **GRANT ALL ON** midb.\* **TO** 'admin'@'%';
- **GRANT SELECT ON** midb.log **TO**  
'auditor'@'192.168.1.100';
- **REVOKE ALL ON** midb.\* **TO** 'admin'@'%';
- **FLUSH PRIVILEGES;**

# Base de Datos Relacionales

## *Características*

- Proveen almacenamiento de datos
- Organizan los datos en un modelo relacional, formulado con tablas de filas y columnas
- Permiten a todos los usuarios recuperar, editar, retornar y remover datos



# Bases de Datos Comerciales

## *Alternativas*

Rank			DBMS	Database Model	Score		
Mar 2017	Feb 2017	Mar 2016			Mar 2017	Feb 2017	Mar 2016
1.	1.	1.	Oracle +	Relational DBMS	1399.50	-4.33	-72.51
2.	2.	2.	MySQL +	Relational DBMS	1376.07	-4.23	+28.36
3.	3.	3.	Microsoft SQL Server +	Relational DBMS	1207.49	+4.04	+71.00
4.	4.	↑ 5.	PostgreSQL +	Relational DBMS	357.64	+3.96	+58.01
5.	5.	↓ 4.	MongoDB +	Document store	326.93	-8.57	+21.60
6.	6.	6.	DB2 +	Relational DBMS	184.91	-2.99	-3.02
7.	↑ 8.	7.	Microsoft Access	Relational DBMS	132.94	-0.45	-2.09
8.	↓ 7.	8.	Cassandra +	Wide column store	129.19	-5.19	-1.14
9.	9.	↑ 10.	SQLite	Relational DBMS	116.19	+0.88	+10.42
10.	10.	↓ 9.	Redis +	Key-value store	113.01	-1.03	+6.79

# MySQL

## *Características y Ventajas*

- Aprovecha la potencia de sistemas multiprocesador
- Soporta gran cantidad de tipos de datos
- Dispone de infinidad de bibliotecas y otras herramientas que permiten su uso a través de gran cantidad de lenguajes de programación
- Soporta hasta 32 índices por tabla





# MySQL

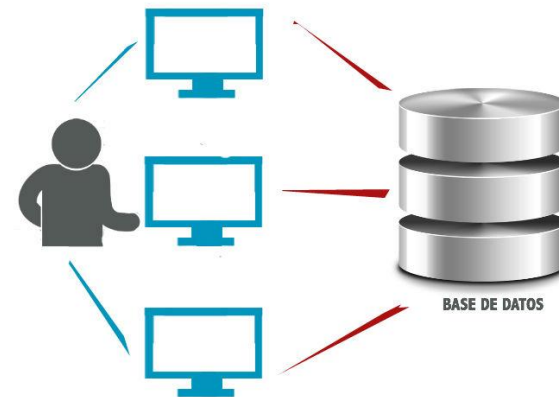
## *Características y Ventajas*

- Gestión de usuarios y passwords
- Gran popularidad entre proyectos de software Web
- Fácil instalación y configuración
- Escalabilidad y flexibilidad
- Alto rendimiento
- Alta disponibilidad
- Robusto soporte transaccional

# Transacciones en Base de Datos

## *Conceptos*

- Una transacción es un mecanismo para definir las unidades lógicas del procesamiento de una base de datos
- Una transacción se inicia por la ejecución de un programa escrito en un lenguaje en un lenguaje de programación
- Una transacción está delimitada por instrucciones de la forma *inicio transacción* y *fin transacción*



# Transacciones en Base de Datos

## *Fallos*

- Fallo de la computadora (caída del sistema)
- Un error de la transacción o del sistema
- Errores locales o condiciones de excepción detectados por la transacción
- Control de la concurrencia
- Fallo del disco rígido
- Problemas físicos y catástrofes



# Transacciones en Base de Datos

*ACID (AC--)*

## Atomicidad

- Requiere que cada transacción sea "todo o nada": si una parte de la transacción falla, todas las operaciones de la transacción fallan, y por lo tanto la base de datos no sufre cambios

## Consistencia

- Asegura que cualquier transacción llevará a la base de datos de un estado válido a otro estado válido. Cualquier dato que se escriba en la base de datos tiene que ser válido de acuerdo a todas las reglas definidas

# Transacciones en Base de Datos

*ACID (--ID)*

## Aislamiento

- Asegura que la ejecución concurrente de las transacciones resulte en un estado del sistema que se obtendría si estas transacciones fueran ejecutadas una detrás de otra

## Durabilidad

- Significa que una vez que se confirmó una transacción quedará persistida, incluso ante eventos como pérdida de alimentación eléctrica, errores y caídas del sistema

# Transacciones en Base de Datos

## *Estados*

- **Activa**, la transacción permanece en este estado durante su ejecución
- **Parcialmente confirmada**, después de ejecutarse la última instrucción
- **Fallida**, tras descubrir que no puede continuar la ejecución normal
- **Abortada**, después de haber retrocedido la transacción y restablecido la base de datos a su estado anterior al comienzo de la transacción
- **Confirmada**, tras completarse con éxito

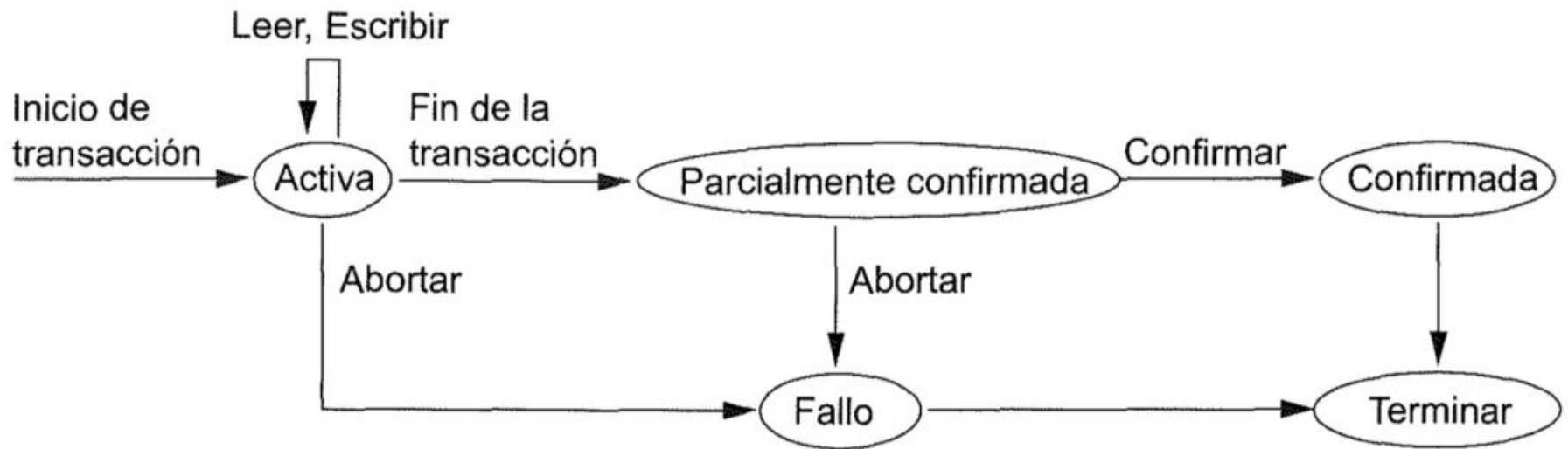
# Transacciones en Base de Datos

## *Operaciones*

- **INICIO DE TRANSACION.** Marca el inicio de la ejecución de una transacción
- **LEER o ESCRIBIR.** Especifican operaciones de lectura o escritura en los elementos de la base de datos que se ejecutan como parte de una transacción
- **FIN DE LA TRANSACION.** Especifica que las operaciones LEER y ESCRIBIR de la transacción han terminado y marca el final de la ejecución de la transacción. En este punto se comprueba si los cambios introducidos pueden confirmarse
- **CONFIRMAR (Commit).** Señala una finalización satisfactoria de la transacción, por lo que los cambios (actualizaciones) ejecutados se pueden enviar con seguridad a la base de datos
- **ABORTAR (Rollback).** Señala que la transacción no ha terminado satisfactoriamente, por lo que deben deshacerse los cambios

# Transacciones en Base de Datos

## *Diagrama de Transición de Estados*





# Transacciones en Base de Datos

## *Concurrencia de transacciones*

- **Enfoque de ejecución secuencial** de transacciones es más sencillo de implementar, pero menos eficiente. Para comenzar una transacción es necesario finalizar la anterior
- **Enfoque de ejecución concurrente** permite varias transacciones que actualizan concurrentemente los datos y puede provocar complicaciones en la consistencia de los mismos

El **esquema de control de concurrencia** de un DBMS controla la interacción entre las transacciones concurrentes para evitar que se destruya la consistencia de la base de datos

# Transacciones en Base de Datos

## *Ventajas de la Concurrencia*

- **Productividad y utilización de recursos mejorados**
  - El número de transacciones que puede ejecutar en un tiempo dado aumenta cuando varias transacciones se ejecutan en paralelo. Por ej., las operaciones de E/S, como uso de CPU y discos pueden trabajar en paralelo en una computadora
- **Tiempo de espera reducido**
  - Frente a transacciones cortas y largas, la ejecución concurrente reduce los retardos impredecibles en la ejecución de las transacciones
  - Se reduce también el tiempo medio de respuesta

# Transacciones en Base de Datos

## *Índices*

- Estructuras complementarias en DBMS que se asocian con los atributos de las tablas y agilizan la operaciones de búsqueda
- Juegan el mismo papel que los índices de los libros o los catálogos de fichas de las bibliotecas
- Por ejemplo, para recuperar un registro de cuenta dado su número de cuenta, el sistema de bases de datos buscaría en un índice para encontrar el bloque de disco en que se encuentra el registro correspondiente, y entonces extraería ese bloque de disco para obtener el registro cuenta



# Transacciones en Base de Datos

## *Tipos de Índices*

- **Índices ordenados.** Estos índices están basados en una disposición ordenada de los valores
- **Índices asociativos.** Estos índices están basados en una distribución uniforme de los valores a través de una serie de cajones (buckets). El valor asignado a cada cajón está determinado por una función, llamada función de asociación (hash function)

# Transacciones en Base de Datos

## *Índices Ordenados: Subtipos*

- **Índice primario o índices con agrupación.** La clave de búsqueda especifica el orden secuencial del archivo (archivos ordenados secuencialmente). La clave de búsqueda de un índice primario es normalmente la clave primaria (pero no necesariamente)
- **Índices secundarios o índices sin agrupación.** Las claves de búsqueda especifican un orden diferente del orden secuencial del archivo. Un archivo puede tener varios índices secundarios además de su método de acceso principal

# Transacciones en Base de Datos

## *Índices Multinivel*

- Solución para índices muy grandes
- El índice se trata como si fuese un archivo secuencial y se construye otro índice sobre el índice con agrupación
- Para localizar un registro se usa en primer lugar una búsqueda binaria sobre el índice más externo para buscar el registro con el mayor valor de la clave de búsqueda que sea menor o igual al valor deseado
- Los índices multinivel están estrechamente relacionados con la estructura de árbol, tales como los árboles binarios usados para la indexación en memoria

# Transacciones en Base de Datos

## *Índices en SQL*

**create index** <nombre-índice> **on** <nombre-tabla>  
(<lista-atributos>)

**drop index** <nombre-índice>

***lista-atributos*** es la lista de atributos de la tabla que constituye la clave de búsqueda del índice

### Ejemplo:

**create index** índice-s **on** sucursal (**nombre-sucursal**)

*índice llamado índice-s de la tabla sucursal con la clave de búsqueda nombre-sucursal*

# Transacciones en Base de Datos

## *Stored Procedures y Funciones*

- Comúnmente llamados procedimientos almacenados aunque pueden ser funciones o procedimientos
- Son módulos de programa de bases de datos que el DBMS almacena y ejecuta en el servidor de bases de datos
- Ayudan a mantener integridad de los datos y su consistencia. Incrementan la seguridad de las operaciones en el DBMS
- Si varias aplicaciones necesitan un mismo programa de base de datos, este último se puede almacenar en el servidor e invocarlo desde esas aplicaciones



# Stored Procedures y Funciones

## *Sintaxis*

**CREATE PROCEDURE** <nombre del procedimiento>  
«parámetros»  
<declaraciones locales>  
<cuerpo del procedimiento>;

**CREATE FUNCTION** <nombre de la función>  
«parámetros»  
**RETURNS** <tipo de devolución>  
<declaraciones locales>  
<cuerpo de la función>;

# Stored Procedures y Funciones

## *Elementos*

### Parámetros y tipos de retorno

```
CREATE PROCEDURE saldo_cuenta (IN dni INT, OUT saldo INT, OUT  
total_de_operaciones INT, OUT fecha_ultima_operacion INT)
```

```
BEGIN
```

```
 cuerpo del procedimiento
```

```
END
```

### Variables

```
DECLARE <nombre_variable> tipo_de_dato (tamaño) DEFAULT valor_defecto.  
Ejemplos:
```

```
DECLARE x, y INT DEFAULT 0;
```

```
DECLARE total INT DEFAULT 0;
```

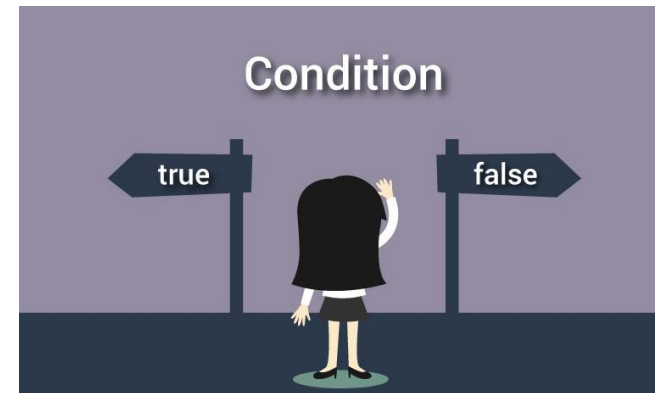
```
SET total= 10;
```

# Stored Procedures y Funciones

## *Elementos*

### Estructuras de control: IF

```
IF expression THEN
    statements;
ELSEIF elseif-expression THEN
    elseif-statements;
...
ELSE
    else-statements;
END IF;
```



### Estructuras de control: CASE

```
CASE case_expression
    WHEN when_expression_1 THEN commands
    WHEN when_expression_2 THEN commands
...
ELSE commands
END CASE;
```



# Stored Procedures y Funciones

## *Elementos*

### Estructuras de control: LOOP

**WHILE** expression **DO**

statements

**END WHILE**

**REPEAT**

statements;

**UNTIL** expression

**END REPEAT**



### Cursor

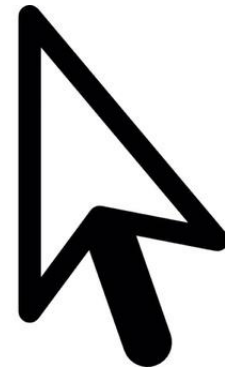
**DECLARE** cursor\_name **CURSOR FOR** select\_statement;

**OPEN** cursor\_name;

**FETCH** cursor\_name **INTO** variables list;

**CLOSE** cursor\_name;

**DECLARE CONTINUE HANDLER FOR NOT FOUND SET** finished = 1;



# Vistas en SQL

## *Conceptos*

Puede conceptualizarse como una tabla de sólo lectura o tabla virtual que deriva de otras tablas

- Las tablas de las que se deriva **pueden ser tablas base** (tablas propias de la base de datos, almacenadas físicamente en la base de datos) o **vistas definidas anteriormente**
- Se restringen las operaciones de actualización, **sólo puede consultarse**

**create view** <nombre-vista> **as** <expresión de consulta>

...donde <expresión de consulta> puede ser cualquier consulta válida

# Vistas en SQL

## *Utilidad*

- Permiten encapsular la lógica compleja de una consulta y simplificar la consulta desde una aplicación
- Frente a múltiples usuarios, permite esconder detalles de implementación de la DB y exponer sólo una parte de dichos datos
- En el caso de vistas indexadas, puede mejorarse el rendimiento general del sistema, sobre todo con consultas recurrentes

# Vistas en SQL

## *Ejemplo*

```
CREATE VIEW detalle_ultimos_10_movimientos AS  
SELECT CL.dni, MO.* FROM Clientes CL  
JOIN Cuentas CU ON (CL.Id = CU.Id_Cliente)  
JOIN Movimientos MO ON (CU.Id = MO.Id_Cuenta)  
ORDER BY MO.Fecha DESC  
LIMIT 10
```

### Desde la Aplicación u otra consulta:

1. `SELECT * FROM detalle_ultimos_10_movimientos;`
2. `SELECT SUM(monto) FROM (SELECT * FROM detalle_ultimos_10_movimientos WHERE dni='12345678');`

# Vistas en SQL

## *Actualización*

Una vista es **actualizable** si:

- Está definida a partir de una única tabla
- Conserva la clave (primaria o alternativa)
- No está definida mediante agrupación o funciones de agregación
- No incluye la cláusula
- No incluye sub-consultas
- No se define mediante unión, intersección, diferencia



# Vistas en SQL

## *Actualización (Ejemplos)*

```
CREATE VIEW Proveedor_ciudad (id_prov, ciudad ) AS  
SELECT id_prov, ciudad FROM PROVEEDORES;
```

→ **Actualizable**

Definida a partir de clave primaria

```
CREATE VIEW Situación_ciudad (situación, ciudad ) AS  
SELECT situación, ciudad FROM PROVEEDORES;
```

→ **No Actualizable**

No definida a partir de clave primaria

# Vistas en SQL

## *Actualización*

- Si la vista es actualizable y está definida con:
  - **WITH CHECK OPTION:** todos los INSERT y UPDATE sobre la vista son chequeados para asegurar que los datos satisfacen la condición de definición de la vista
  - **WITH LOCAL CHECK OPTION:** chequea la integridad sobre la vista
  - **WITH CASCADE CHECK OPTION:** chequea la integridad sobre la vista y cualquier vista dependiente

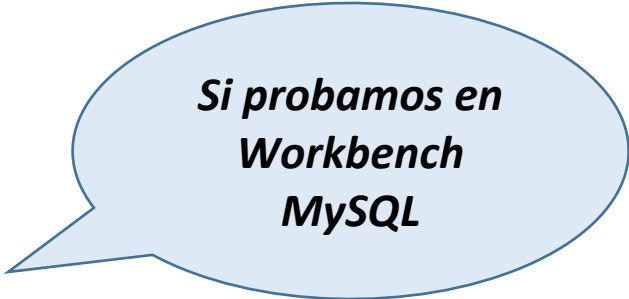
# Vistas en SQL

## Actualización - Ejemplos

1. **CREATE TABLE *t1*** (a INT);
2. **CREATE VIEW *v1* AS SELECT \* FROM *t1* WHERE *a* < 2**  
**WITH CHECK OPTION;**
3. **CREATE VIEW *v2* AS SELECT \* FROM *v1* WHERE *a* > 0**  
**WITH LOCAL CHECK OPTION;**
4. **CREATE VIEW *v3* AS SELECT \* FROM *v1* WHERE *a* > 0**  
**WITH CASCADED CHECK OPTION;**

• *mysql*> INSERT INTO v2 VALUES (2);  
Query OK, 1 row affected (0.00 sec)

• *mysql*> INSERT INTO v3 VALUES (2);  
ERROR 1369 (HY000): CHECK OPTION failed 'test.v3'



*Si probamos en  
Workbench  
MySQL*

# Triggers en SQL

## *Conceptos*

- Un trigger es un objeto de base de datos con nombre que se asocia a una tabla, y se activa cuando ocurre un evento en particular para la tabla
- Un uso puede ser verificar valores a ser insertados o llevar a cabo cálculos sobre valores involucrados en una actualización
- Se almacenan en la BD, por lo que son persistentes y accesibles para todas las operaciones de la base de datos

# Triggers en SQL

## *Sintaxis SQL*

- Los triggers tienen dos palabras clave, OLD y NEW. Son los valores que tienen las columnas antes y después de la modificación
- Los INSERT permiten NEW, los DELETE sólo OLD y los UPDATE ambas
- **CREATE TRIGGER** nombre del trigger  
    {BEFORE|AFTER}  
    {INSERT|UPDATE|DELETE}  
    ON nombre de la tabla  
    FOR EACH ROW  
    BEGIN  
        sentencia SQL  
    END;

# Triggers en SQL

## *Tipos de Triggers*

- **Triggers de nivel de fila:** se ejecutan una vez para cada fila afectada. Se crean utilizando la cláusula **for each row** en el comando **create trigger**
- **Triggers de nivel de instrucción:** se ejecutan una vez para cada instrucción. Se crean utilizando la cláusula **for each statement** en el comando **create trigger**
- **Triggers Before y After:** puesto que los triggers son ejecutados por sucesos, puede establecerse que se produzcan inmediatamente antes (before) o después (after) de dichos sucesos

# Triggers en SQL

## *Tipos de Triggers*

- **Triggers Instead Of:** Indican lo que se tiene que hacer en lugar de realizar las acciones que invoca el trigger
- **Triggers de esquema:** Son triggers sobre operaciones en el nivel de esquema tales como create table, alter table, drop table, audit, rename, truncate y revoke
- **Triggers en nivel de base de datos:** puede crear triggers que se activen al producirse sucesos de la base de datos, incluyendo errores, inicios de sesión, conexiones y desconexiones

# Triggers en SQL

## *Ejemplo*

- Supongamos la situación:

Cuando un usuario inserta un registro en la tabla **Prueba** cuyo ***Atributo2*** tenga un valor superior a 1000 se insertará automáticamente una fila en la tabla **ResultadoDisparador** con:

- la fecha de la inserción (*sysdate*) → ***Fecha actual del Sistema***
- el aviso “***Registro con Atributo2 superior a 1000***”
- el nombre de la tabla Prueba



# Triggers en SQL

## *Ejemplo*

- Para que el trigger funcione correctamente deberán existir las tablas **Prueba** y **ResultadoDisparador**:

```
CREATE TABLE PRUEBA(  
  "Atributo1" VARCHAR(10) NOT NULL,  
  "Atributo2" NUMBER)
```



```
CREATE TABLE RESULTADODISPARADOR (  
  "Fecha" DATE NOT NULL,  
  "Aviso" VARCHAR(100) NOT NULL,  
  "NombreTabla" VARCHAR(50) NOT NULL)
```

# Triggers en SQL

## *Ejemplo*

```
CREATE TRIGGER INSERCIONNUMEROMAYOR1000  
AFTER  
INSERT ON Prueba FOR EACH ROW WHEN  
(new.Atributo2 > 1000)  
BEGIN  
INSERT INTO ResultadoDisparador (fecha, aviso,  
tabla)  
VALUES  
(Sysdate, 'Registro con Atributo2 superior a 1000',  
'Prueba');  
END;
```

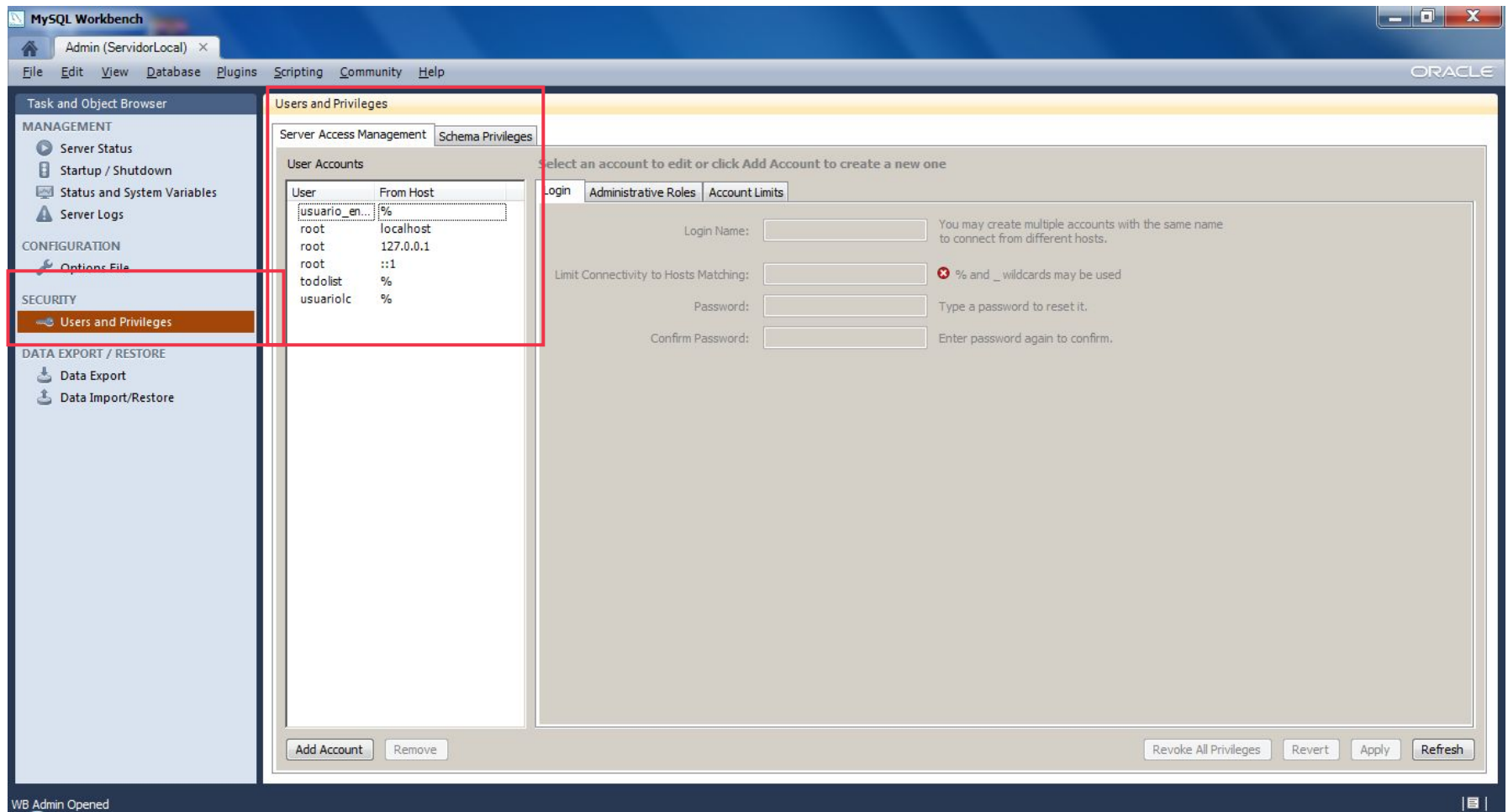
Base de Datos

**CFP**  
**Programador**  
**full-stack**

*Seguridad y Transacciones en Base de Datos(Ejercicios)*

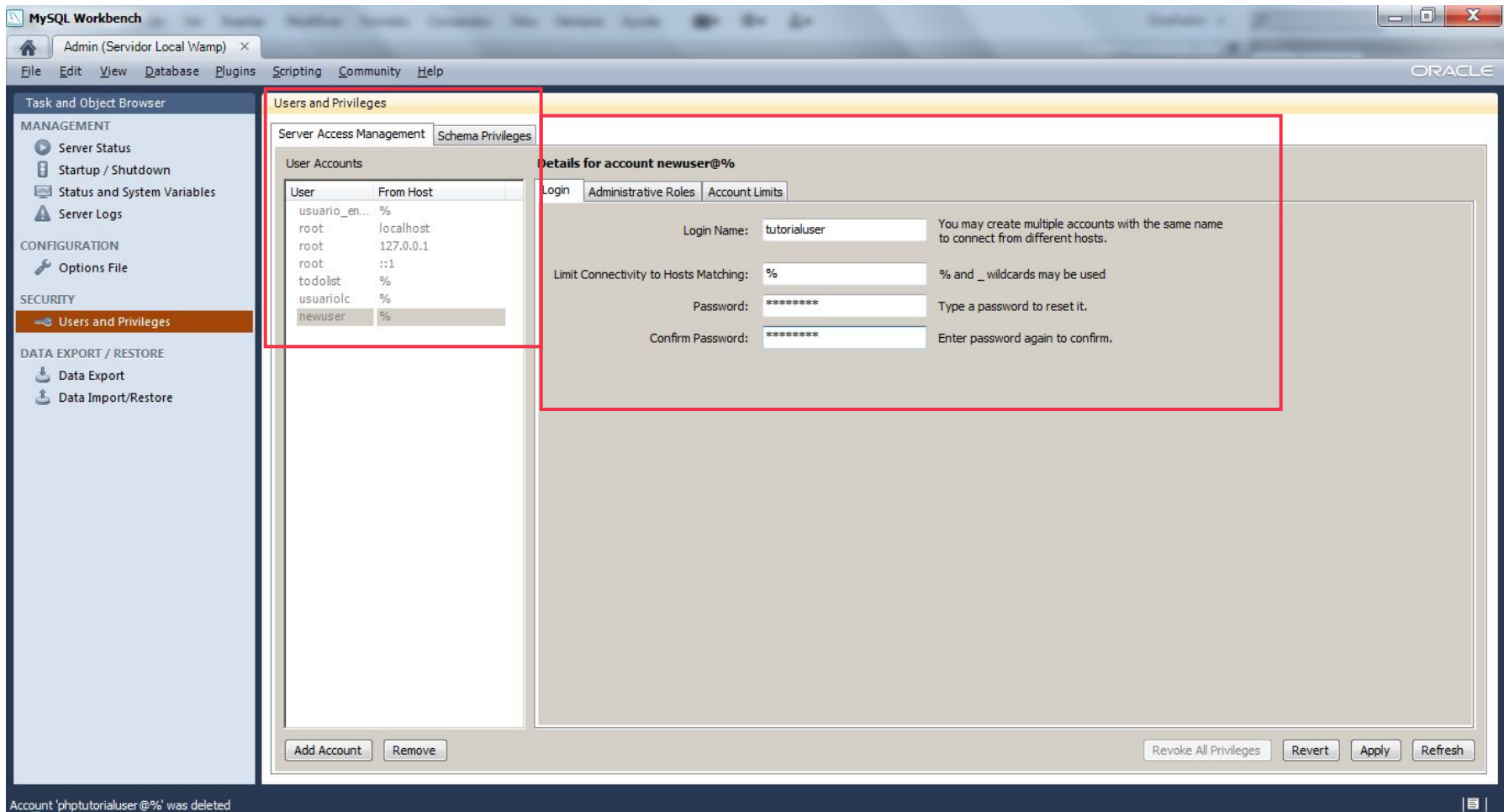
# Seguridad en Base de Datos

## *Gestión de usuarios en MySQL Workbench*



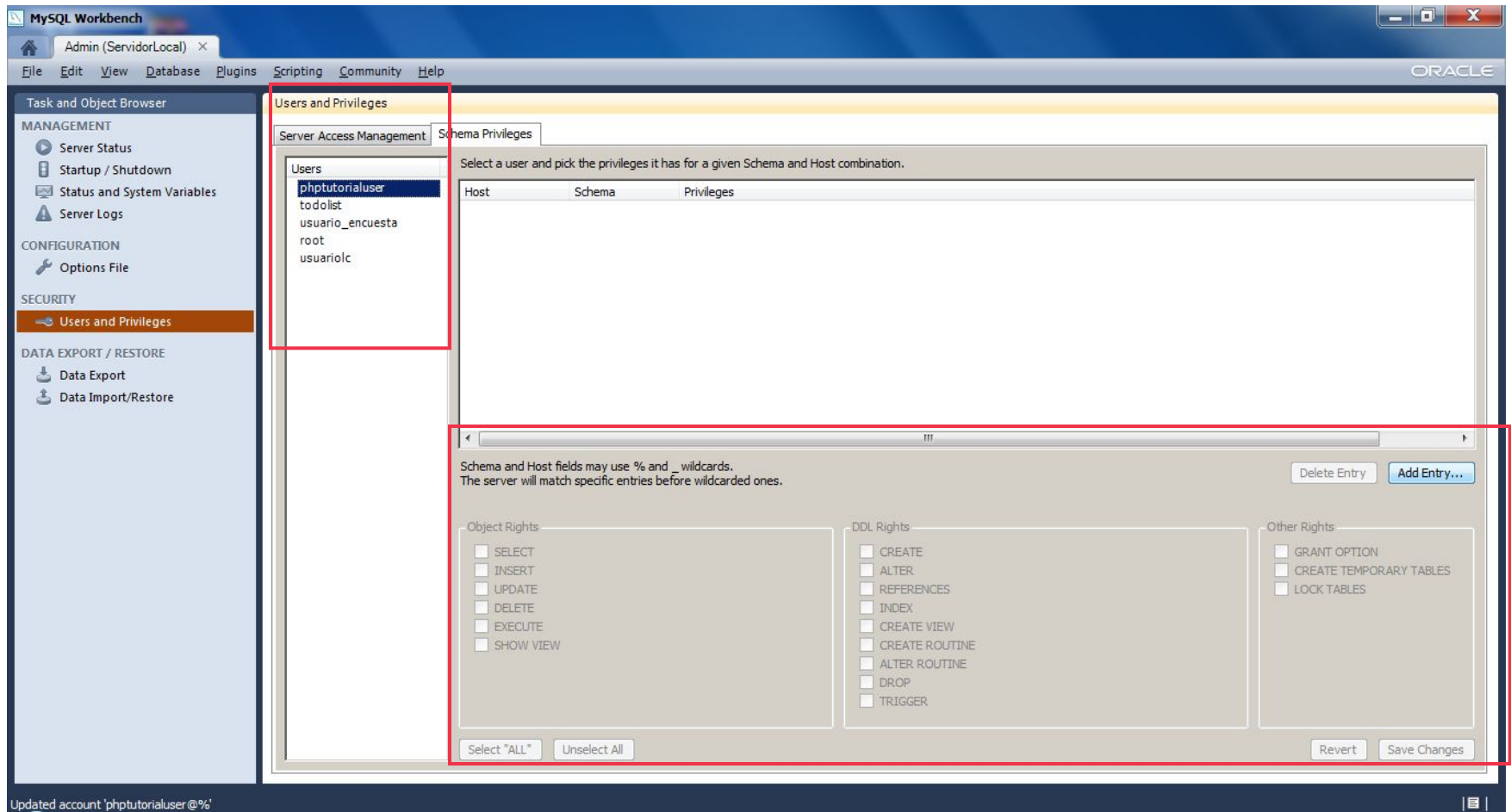
# Seguridad en Base de Datos

## *Gestión de usuarios en MySQL Workbench*



# Seguridad en Base de Datos

## *Gestión de usuarios en MySQL Workbench*



# Transacciones en Base de Datos

## *Ejercicio de Triggers*

Observe el siguiente trigger y analice cuántas veces se ejecuta si dicho trigger es:

- *For each row*
- *For each statement*

	DNI	Nombre	Cuota
	1	Juan	133
	2	Gustavo	222
→	3	Pedro	452
→	4	Mariana	164
→	5	Silvina	223

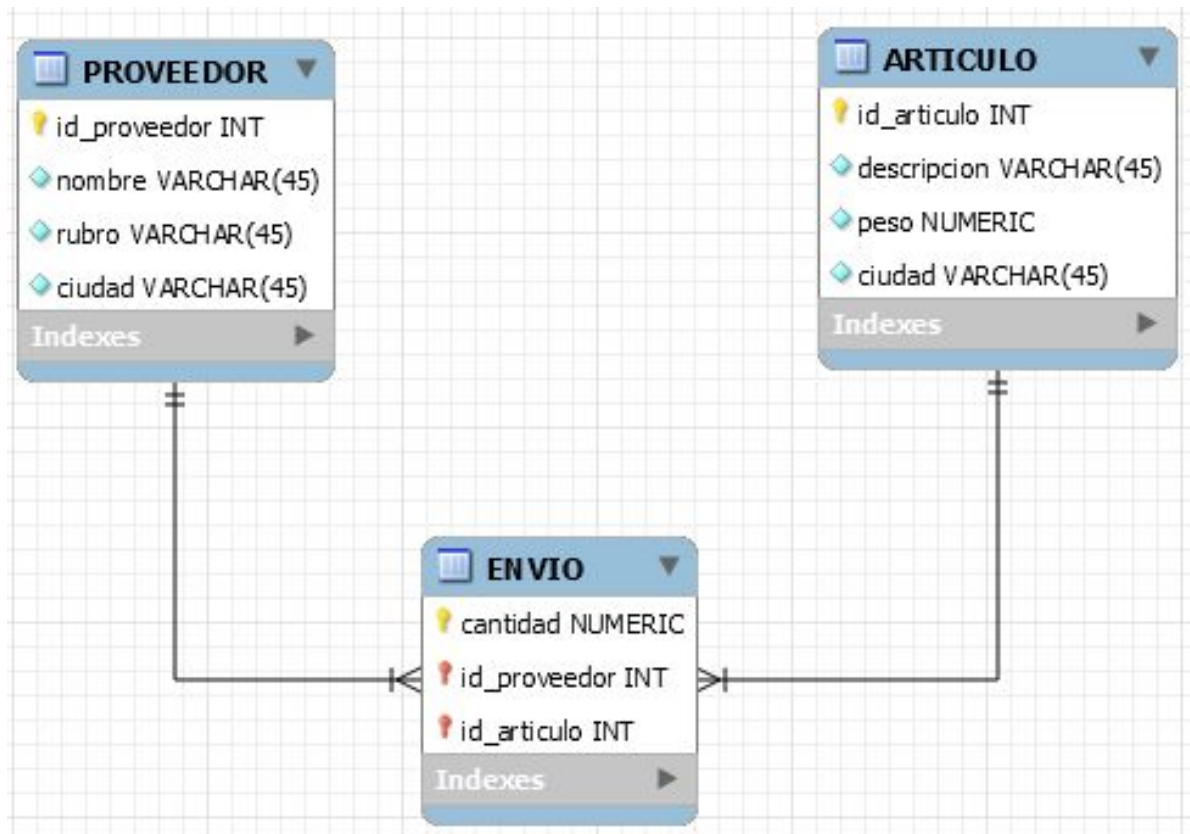
update cliente  
    set cuota = cuota \* 1.2  
    where dni > 2;

Cantidad de tuplas Afectadas ?

# Transacciones en Base de Datos

## *Ejercicios de Vistas*

Observe el siguiente DER





# Transacciones en Base de Datos

## *Ejercicios de Vistas*

1. ENVIOS500 con los envíos de más de 500 unidades de algún artículo (a partir de ENVIO)
2. PRODUCTOS\_MAS\_PEDIDOS con los diferentes artículos que han tenido al menos un envío de más de 500 unidades (a partir de ENVIOS500)
3. ENVIOS\_PROV con los diferentes id de proveedor y la cantidad total de unidades enviadas (a partir de Envio)
4. DETALLE\_ENVIOS que contenga, para cada envío de más de 500 unidades, la descripción y el peso del artículo, el nombre del proveedor y la cantidad enviada (a partir de Proveedor, Artículo, ENVIOS500)

# Transacciones en Base de Datos

## *Ejercicios de Vistas*

Dada la siguiente vista:

```
CREATE VIEW Buenos_proveedores Buenos_proveedores (prov, sit,  
ciudad ) , ciudad ) AS SELECT id_prov id_prov, situación, ciudad ,  
situación, ciudad FROM PROVEEDORES WHERE situación > 15  
WITH CHECK OPTION;
```

Pruebe las siguientes transacciones:

1. Insert into Buenos\_proveedores (prov, sit, ciudad ) values (10, 20, 'Paris').
2. Update Buenos\_proveedores set situacion = 5 where prov = 20;
3. Insert into Buenos\_proveedores (prov, sit, ciudad ) values (8, 5, 'Roma');

# Seguridad en Bases de Datos

## *Ejercicios de Privilegios*

El usuario **A** ha creado la tabla **Usuario** (nro\_u, nombre, tarea), y luego ejecuta los siguientes comandos SQL:

1. GRANT INSERT ON Usuario TO B WITH GRANT OPTION;
2. GRANT SELECT ON Usuario TO B WITH GRANT OPTION;
3. GRANT SELECT ON Usuario TO C;

Indique quienes pueden ejecutar exitosamente los siguientes comandos:

1. SELECT \* FROM A.Usuario WHERE nro\_u='C';
2. INSERT INTO A.Usuario VALUES ('C','Gerente', 'Control');
3. GRANT SELECT ON A.Usuario TO D;

# Base de Datos

## CFP Programador full-stack

***Seguridad y Transacciones en Base de Datos(Repaso)***

# Seguridad en Base de Datos

## *Repaso*

- La seguridad se refiere a la protección de los datos contra accesos no autorizados
- Diferentes tipos de datos precisan diferentes niveles de seguridad
- El costo de la pérdida de los datos determinará el tipo de seguridad requerida
- No serán útiles los sistemas de seguridad que impidan el acceso a los datos a un usuario legítimo o de seguridad requerida
- El DBMS provee un subsistema de seguridad y autorización de la BD
- Para acceder a la BD se requiere una cuenta de usuario y contraseña

# Seguridad en Base de Datos

## *Repaso*

- El Administrador de la Base de Datos (DBA) -posee cuenta privilegiada *ROOT*- debe asegurar una política de acceso clara y consistente:
  - decidir quién entra a la BD y qué puede hacer sobre los objetos a los que puede acceder (limitado a lo que tienen acceso)
  - garantizar la seguridad de partes de la BD contra accesos no autorizados(sin derecho de acceso)
  - no impedir el acceso a los datos por usuarios habilitados (disponibilidad)

# Seguridad en Base de Datos

## *Repaso*

- El acceso a la BD se basa en otorgar y revocar privilegios sobre los objetos de la BD, dando acceso selectivo a otros usuarios (a discreción):
  - concesión de derechos (GRANT)
  - revocación de derechos (REVOKE)

# Transacciones en Base de Datos

## *Repaso*

- Una transacción es una unidad lógica de trabajo atómica
- No hay una sentencia explícita de inicio, implícitamente comienza por una sentencia SQL
- Finalización: con COMMIT o ROLLBACK
- Cada transacción tiene ciertas características, que se pueden especificar en la sentencia SET TRANSACTION



# SQL Procedural

## *Repaso*

- **Trigger:** Es un procedimiento que es invocado automáticamente por el DBMS en respuesta a un evento específico de la Base de Datos
- **Stored Procedure:** Es un procedimiento que es invocado explícitamente por el usuario
- **Función:** Puede ser predefinida o definida por el usuario para realizar operaciones específicas sobre los datos, y pueden ser invocadas desde un trigger, stored procedure o explícitamente

# Vistas en SQL

## *Repaso*

- Tabla virtual (habitualmente no materializada) donde las filas se generan al operar sobre la vista, según su definición (no necesariamente existen en forma física)
- Relación derivada que se define dando un nombre a una expresión de consulta
- Su contenido está definido como una consulta sobre una o más tablas (u otras vistas)