

12 JULIO, 2023

INTEGRADORA

ASIGNATURA
BASE DE DATOS PARA
APLICACIONES

PROYECTO SPC

1 CONTENIDO

2	Introducción.....	3
3	Propuesta de solución	3
4	Diseño de solución	5
5	Vistas.....	7
6	Índices.....	11
7	Disparadores.....	14
8	Agrupamiento.....	19
9	Ordenamiento	21
10	Desarrollo	23
11	Procedimientos Almacenados.....	44
12	Desarrollo de procedimientos almacenados.....	51

2 INTRODUCCIÓN

La problemática en este sistema es que se necesita una base de datos que permita almacenar y organizar la información de las transacciones contables de un usuario de forma segura y accesible. Es necesario que el sistema tenga mecanismos de autenticación y registro de usuario para garantizar la privacidad y seguridad de la información.

La base de datos debe estar diseñada de tal manera que permita la fácil inserción y modificación de los datos de ingresos y egresos, y que proporcione una vista consolidada y clara de las transacciones realizadas por el usuario en un periodo de tiempo determinado.

Además, se necesita que el sistema cuente con un registro de actividad (log) que permita conocer en todo momento quién ha accedido al sistema y qué acciones han realizado, para poder detectar cualquier actividad sospechosa o inusual y tomar medidas de seguridad en consecuencia.

En resumen, se necesita una base de datos robusta y bien diseñada que maneje de forma segura y eficiente la información contable del usuario, permitiendo un fácil registro y consulta de la información, además de garantizar la privacidad y seguridad de los datos.

3 PROPUESTA DE SOLUCIÓN

En este reporte se presentará la solución para una base de datos para un sistema personal contable, en el cual se había contemplado inicialmente el uso de Oracle como gestor de base de datos (DBMS), sin embargo, debido a sus costos se decidió utilizar MySQL, que cuenta con ventajas que lo hacen una buena opción para este proyecto.

MySQL es un sistema de gestión de bases de datos relacional, que se caracteriza por ser de código abierto, lo que significa que es gratuito y se puede descargar desde su sitio web oficial. Además, es compatible con diversos sistemas operativos y lenguajes de programación, lo que facilita su integración con distintas aplicaciones.

Para el sistema personal contable se requiere de una base de datos que almacene la información de los usuarios, sus cuentas, transacciones y saldos. Para ello, se puede diseñar la estructura de la base de datos con MySQL Workbench, que es una herramienta gráfica que permite diseñar, modelar y generar la estructura de la base de datos.

MySQL permite la creación de las tablas con sus respectivas relaciones utilizando el lenguaje SQL, que es un estándar para la creación y manipulación de bases de datos relacionales.

Además, MySQL ofrece otras funciones y herramientas que pueden resultar útiles para un sistema de contabilidad personal, como, por ejemplo:

- La posibilidad de encriptar las contraseñas de los usuarios.
- La capacidad de realizar backups y restauraciones de la base de datos para garantizar la integridad de los datos.

- La capacidad de manejar grandes cantidades de datos de manera eficiente gracias a su arquitectura escalable.

En conclusión, aunque Oracle es una opción de alta calidad para la gestión de bases de datos, MySQL es una alternativa viable para un sistema personal contable, ya que es gratuito, fácil de usar, y cuenta con herramientas y funciones que permiten el manejo eficiente de la información financiera.

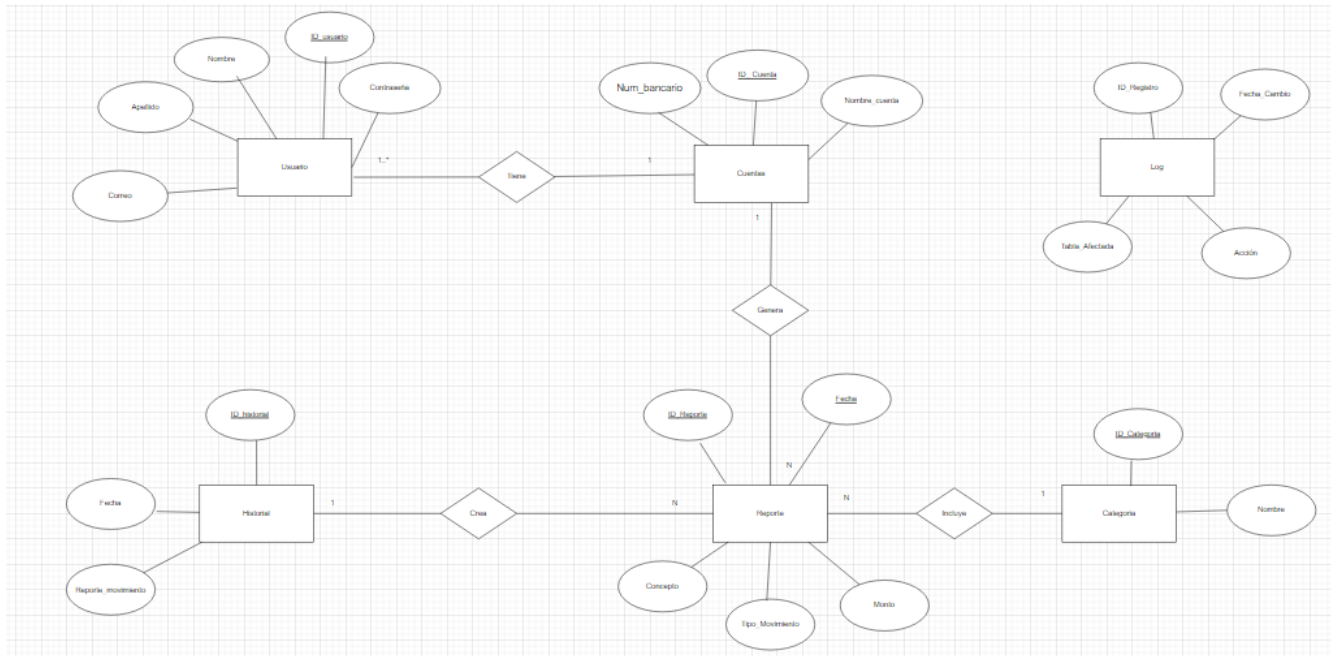
4 DISEÑO DE SOLUCIÓN

Modelo Entidad-Relación

El diseño de solución del modelo entidad-relación donde las entidades son:

1. Usuario y sus atributos son id de usuario, nombre, apellido, correo y contraseña.
2. Cuentas y sus atributos son idCuenta_usuario, nombre_cuenta y Num_bancario.
3. Reporte y sus atributos son id de reporte, monto, fecha, concepto, tipo movimiento.
4. Categoría y sus atributos son Id de categoría y nombre de categoría.
5. Historial y sus atributos son Id de Historial, fecha, Reporte de movimientos.
6. ReporteLog y sus categorías son Id de registro, Tabla afectada, acción y Fecha.

Estas entidades se dieron a base lo planteado en el proyecto por lo que el diagrama término de la siguiente manera:



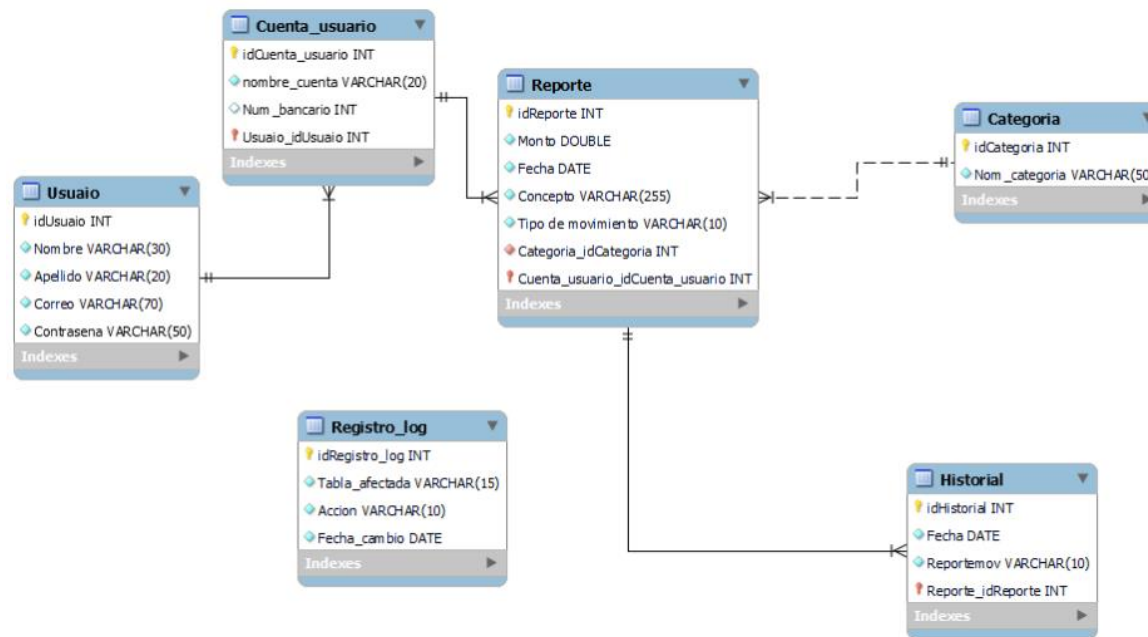
Nota: se agrego una nueva entidad llamada cuentas.

Modelo Relacional

El diseño de solución del modelo Relacional donde las entidades propuestas anteriormente fueron:

1. Usuario y sus atributos son id de usuario, nombre, apellido, correo y contraseña. Donde id de usuario será una llave primaria.
2. Cuenta_Usuario y sus atributos son idCuenta_usuario, nombre_cuenta, Num_bancario. Donde idCuenta_usuario será una llave primaria y la llave foránea será el id_usuario
3. Reporte y sus atributos son id de reporte, monto, fecha, concepto, tipo movimiento. Donde Id de reporte será la llave primaria y se agregarán dos campos más los cuáles serán las llaves foráneas Usuario_IdUsuario y Categoría_IdCategoría.
4. Categoría y sus atributos son Id de categoría y nombre de categoría. Donde Id de reporte será la llave primaria.
5. Historial y sus atributos son Id de Historial, fecha, Reporte de movimientos. Donde Id de Historial será la llave primaria y se agregarán dos campos más los cuáles serán las llaves foráneas Reporte_IdReporte y RegistroLog_IdRegistrolog.
6. ReporteLog y sus categorías son Id de registro, Tabla afectada, acción y Fecha. Donde Id de Registro será la llave primaria.

Por lo que el diseño en el gestor de la base de datos se encontró de la siguiente manera:



Nota: Se agrego una nueva tabla llamada cuenta_usuario.

5 VISTAS

- **Vista para Saldo General Actual del Usuario**

La siguiente vista genera una tabla donde relaciona los datos del usuario con el saldo actual con el que cuenta y este se obtiene con la suma de sus ingresos y la resta de la suma de los egresos. Estos datos ayudaran a la consulta del usuario sobre su saldo general.

```
#Tabla usuario SELECT * FROM reporte

CREATE VIEW v_saldo_usuario AS SELECT
id_usuario,CONCAT(usuario.nombre,',',usuario.apellido) as Usuario,
ROUND((SELECT SUM(monto) FROM reporte WHERE tipo_movimiento='Ingreso')-
(SELECT SUM(monto) FROM reporte WHERE tipo_movimiento='Egreso'),2) AS Saldo
FROM usuario
inner join cuenta_usuario ON cuenta_usuario.fk_id_usuario=usuario.id_usuario
inner join reporte ON
cuenta_usuario.id_cuenta_usuario=reporte.fk_cuenta_usuario
INNER JOIN categoria ON reporte.fk_categoria= categoria.id_categoria
GROUP BY id_usuario;
#Vista que muestra el ID del usuario, su nombre completo y, restandole los
valores de "Egreso" a "Ingreso",
#se obtiene el valor total, que llamamos "saldo", además de limitar la resta a
dos decimales, finalmente
#agrupando todos los valores obtenidos por la ID del usuario
```

- **Vista para la creación del reporte del usuario**

La siguiente vista genera una tabla donde relaciona los datos del usuario donde se muestra el ingreso/egreso, el movimiento, la categoría seleccionada y la fecha en que se realizó.

```
#Tabla General Usuario (SOLO TIENE ORDER BY)
CREATE VIEW v_informacion_general_all AS SELECT
id_usuario,CONCAT(usuario.nombre,' ',usuario.apellido) AS Nombre_Usuario,
#Tabla usuarios
reporte.tipo_movimiento, #Tabla reporte
categoria.nombre AS Nombre_Categoria, #Tabla categoria
historial.fecha #Tabla historial
FROM usuario
inner join cuenta_usuario ON cuenta_usuario.fk_id_usuario=usuario.id_usuario
inner join reporte ON
cuenta_usuario.id_cuenta_usuario=reporte.fk_cuenta_usuario
INNER JOIN categoria ON reporte.fk_categoria= categoria.id_categoria
inner join historial ON historial.fk_reporte=reporte.id_reporte
ORDER BY historial.fecha desc;
#Tabla que junta el nombre completo del usuario, el movimiento que va a hacer,
```

- **Vista que muestra los nombres de las categorías y la fecha**

Esta vista muestra la lista de categorías creadas, así como su nombre y fecha de creación.

```
#Tabla categoria SOLO ORDER BY
CREATE VIEW v_nombres_categoria AS SELECT
nombre, reporte.fecha as Ultima_modificacion
FROM categoria INNER JOIN reporte ON reporte.fk_categoria=
categoria.id_categoria
ORDER BY reporte.fecha DESC;
#Vista que muestra los nombres de las categorias y la fecha escrita en la tabla
pedidos que marca,
#como se muestra en la vista, la fecha de la ultima modificacion de la
categoria,
#la fecha se ordena desde la mas reciente hasta la mas vieja
```

- **Vista que muestra el número de categorías creadas**

Esta vista muestra el número de categorías creadas de tal modo que el cliente pueda tener una mejor visualización.

```
#Tabla Categoria 2 SOLO GROUP BY
CREATE VIEW v_modificacion_categoria AS SELECT
COUNT(nombre) AS Modificacion_Categorias, reporte.fecha as Ultima_modificacion
FROM categoria INNER JOIN reporte ON reporte.fk_categoria=
categoria.id_categoria
GROUP BY reporte.fecha;
```

- **Vista que junta la acción y fecha de la acción de la tabla de log**

Esta vista muestra un “historial” donde se muestran las acciones por fecha y las acciones registradas.

```
#Tabla registro_log
CREATE VIEW v_fecha_accion_registro_log AS SELECT
accion, fecha_cambio AS Fecha FROM registro_log
ORDER BY fecha_cambio desc;
#vista que junta la acción y fecha de la accion (de mas reciente a mas
antigua) de la tabla de log
```

- **Vista que junta todas las acciones**

Esta vista muestra un “historial” donde se muestran todas las acciones registradas.

```
CREATE VIEW v_all_acciones_registro_log AS SELECT
COUNT(accion) AS Total_de_acciones, fecha_cambio AS Fecha FROM registro_log
GROUP BY fecha_cambio;
#Vista que junta todas las acciones que se hicieron en X dia
```


- **Vista que muestra el monto, y la fecha del reporte de ingresos**

Esta vista muestra un “historial” donde se muestra el reporte de ingresos realizados

```
##Tablas reporte
CREATE VIEW v_resumen_ingresos_reporte AS SELECT
id_reporte, monto, categoria.nombre AS Categoria, fecha FROM reporte
INNER JOIN categoria ON reporte.fk_categoria= categoria.id_categoria
WHERE tipo_movimiento='Ingreso'
ORDER BY fecha desc;
#Vista que muestra el monto, y la fecha del reporte (mas reciente a mas
antigua)
# unicamente de ingresos
```

- **Vista que muestra el monto, y la fecha del reporte de egresos**

Esta vista muestra un “historial” donde se muestra el reporte de egresos realizados

```
CREATE VIEW v_resumen_egresos_reporte AS SELECT
id_reporte, monto, categoria.nombre AS Categoria, fecha FROM reporte
INNER JOIN categoria ON reporte.fk_categoria= categoria.id_categoria
WHERE tipo_movimiento='Egreso'
ORDER BY fecha desc;
#Vista que muestra el monto, y la fecha del reporte (mas reciente a mas
antigua)
# unicamente de egresos
```

- **Vista que muestra la suma de todos los ingresos, dividido por categorías**

Esta vista muestra un “historial” donde se muestra el reporte de la suma todos los ingresos mostrando la categoría a donde fueron agregados.

```
CREATE VIEW v_ingresos_totales_categoria_reporte AS SELECT
SUM(monto) AS Ingreso_Total, categoria.nombre FROM reporte
INNER JOIN categoria ON reporte.fk_categoria= categoria.id_categoria
WHERE tipo_movimiento='Ingreso'
GROUP BY categoria.nombre;
#Vista que muestra la suma de todos los ingresos, dividido por categorias
```

- **Vista que muestra la suma de todos los egresos, dividido por categorías**

Esta vista muestra un “historial” donde se muestra el reporte de la suma todos los egresos mostrando la categoría a donde fueron agregados.

```
CREATE VIEW v_egresos_totales_categoria_reporte AS SELECT
SUM(monto) AS Egreso_Total,categoria.nombre FROM reporte
INNER JOIN categoria ON reporte.fk_categoria= categoria.id_categoria
WHERE tipo_movimiento='Egreso'
GROUP BY categoria.nombre;
#Vista que muestra la suma de todos los egresos, dividido por categorías
```

- **Vista que muestra el nombre de los bancos registrados**

Esta vista que muestra el nombre de los bancos registrados, los números de cuenta de los respectivos bancos y a nombre de que usuario está dicha cuenta bancaria

```
CREATE VIEW v_cuentas_bancarias_cuenta_usuario AS SELECT
nombre_banco AS Nombre_del_Banco, num_cuenta AS Numero_de_Cuenta,
CONCAT(usuario.nombre,' ',usuario.apellido) AS Nombre_Usuario
FROM cuenta_usuario
INNER JOIN usuario ON usuario.id_usuario=cuenta_usuario.fk_id_usuario
ORDER BY nombre_banco asc;
# Vista que muestra el nombre de los bancos registrados, los numeros de cuenta
de los respectivos bancos
# y a nombre de que usuario está dicha cuenta bancaria
```

- **Vista que muestra el nombre del propietario de las cuentas bancarias**

Esta vista muestra el nombre del propietario de las cuentas bancarias y cuantas cuentas tiene en total.

```
CREATE VIEW v_all_cuentas_bancarias_cuenta_usuario AS SELECT
CONCAT(usuario.nombre,' ',usuario.apellido) AS Nombre_Usuario,
COUNT(*) AS Numero_Cuentas_Bancarias
FROM cuenta_usuario
INNER JOIN usuario ON usuario.id_usuario=cuenta_usuario.fk_id_usuario
GROUP BY fk_id_usuario;
# Vista que muestra el nombre del propietario de las cuentas bancarias y
cuantas cuentas tiene en total
```

- **Vista que muestra el movimiento y la fecha del movimiento de historial**

Esta vista muestra un historial donde se muestra el reporte de movimientos hechos y la fecha en que fueron realizados.

```
#Tabla historial
CREATE VIEW v_all_historial AS SELECT
fecha,COUNT(reporte_mov) AS Transacciones_del_dia
FROM historial
GROUP BY fecha
ORDER BY fecha desc;
#Vista que muestra el movimiento y la fecha del movimiento de historial
```

6 ÍNDICES

- **Creación de índice simple en la tabla usuario**

Este índice que sirve para facilitar la búsqueda de la id de los usuarios

```
CREATE INDEX id_usuarios ON usuario ( id_usuario );
```

- **Creación de compuesto para la tabla usuario**

Este índice sirve para facilitar la búsqueda de los datos de la cuenta de un usuario

```
create index Datos_app_usuario on usuario (id_usuario,correo,contrasena);
```

- **Creación de índice único para la tabla usuario**

Este índice sirve para facilitar la búsqueda de los datos de un usuario junto con su id

```
CREATE UNIQUE INDEX Datos_usuario ON usuario (id_usuario,nombre,apellido);
-----
```

- **Creación de índice simple para la tabla categoría**

Este índice sirve para facilitar la búsqueda de la id de la categoría

```
create index id_categoria on categoria (id_categoria);
```

- **Creación de índice compuesto para la tabla categoría**

Este índice sirve para facilitar la búsqueda del nombre de la categoría junto con su id

```
create index datos_categoria on categoria (id_categoria,nombre);
```

- **Creación de índice simple para la tabla registro log**

Este índice sirve para facilitar la búsqueda de la id de los registros

```
create index registro_log on registro_log (id_registro_log);
```

- **Creación de índice compuesto para la tabla registro log**

Este índice sirve para facilitar la búsqueda de toda la información acerca de los cambios realizados en las tablas

```
create index cambios_tablas on registro_log  
(tabla_afectada, accion, fecha_cambio);
```

- **Creación de índice único para la tabla registro log**

Este índice sirve para facilitar la búsqueda de las fechas donde se realizaron los cambios y junto con el cambio realizado esto en caso de que solo se desee hacer una consulta de las fechas y lo que se realizó en dicho momento

```
CREATE UNIQUE INDEX fecha_cambios on registro_log (accion, fecha_cambio);
```

- **Creación de índice simple para la tabla reporte**

Este índice sirve para facilitar la búsqueda de la id del reporte

```
create index id_reporte on reporte (id_reporte);
```

- **Creación de índice compuesto para la tabla reporte**

Este índice sirve para facilitar la búsqueda de la información de los movimientos junto con la id de su respectivo reporte

```
create index fecha_movimientos on reporte (id_reporte, fecha, tipo_movimiento);
```

- **Creación de índice único para la tabla reporte**

Este índice sirve para facilitar la búsqueda de los movimientos realizados por los usuarios

```
CREATE UNIQUE INDEX usuario_movimientos on reporte  
(fk_usuario, tipo_movimiento);
```

- **Creación de índice simple para la tabla historial**

Este índice sirve para facilitar la búsqueda de la id del historial

```
create index id_historial on historial (id_historial);
```

- **Creación de índice compuesto para la tabla historial**

Este índice sirve para facilitar la búsqueda de la fecha en la cual ciertos registros comenzaron a formar parte del historial

```
create index fecha_registro on historial (fecha,fk_registro_log);
```

- **Creación de índice único para la tabla historial**

Este índice sirve para facilitar la búsqueda de los reportes y registros que forman parte del historial

```
CREATE UNIQUE INDEX historial_reporte_registro on historial  
(id_historial,fk_reporte,fk_registro_log);
```

7 DISPARADORES

- **Crea un disparador que no permite que las categorías se queden en blanco**

Esto delimita que el usuario deje las categorías en blanco y se asegura que la categoría contenga algo dentro.

```
DELIMITER $$
create trigger verificacion_categoria before insert on categoria
for each row
begin
declare categoria_blanco varchar(1);
set categoria_blanco=' ';
if new.nombre<=>categoria_blanco then
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No se admiten categorias en
blanco';
end if;
end; $$
```

- **Crea un disparador que no permite que el nombre sea el mismo al actualizar la categoría**

Este disparador tiene la finalidad de verificar que las categorías no tengan el mismo nombre y solo admita un único nombre por categoría.

```
DELIMITER $$
create trigger nom_repetido before update on categoria
for each row
begin
declare nom_viejo varchar(50);
set nom_viejo=old.nombre;
if new.nombre<=> nom_viejo then
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No hay ninun cambio de
nombre';
end if;
end; $$
```

- **Crea un disparador que evita borrar las categorías que se están usando**

Esto ayuda a evitar borrar categorías que contengan datos y así mandar un mensaje al usuario que notifique que no es posible la eliminación de esta.

```
DELIMITER $$
create trigger categoria_usada before delete on categoria
for each row
begin
declare num_categoria integer;
SELECT
    COUNT(*)
INTO num_categoria FROM
    reporte
WHERE
    fk_categoria = old.id_categoria;
if num_categoria>0 then
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Esta categoria se esta usando';
end if;
end; $$
```

- **Creación de disparadores para la inserción y actualización en la tabla log**

Inserción de datos en la tabla log sobre los datos de la tabla de categorías

```
#insert
DELIMITER $$
create trigger insert_log after insert on categoria
for each row
begin
insert into registro_log values (0,'Categoria','Insert',curdate());
end; $$
```

Actualización de datos en la tabla log e inserción de datos de la tabla de categorías

```
DELIMITER $$
create trigger update_log after update on categoria
for each row
begin
insert into registro_log values (0,'Categoria','Update',curdate());
end; $$
```

Eliminación de datos en la tabla log e inserción de datos de eliminación de la tabla de categorías

```
DELIMITER $$
create trigger Delete_log after delete on categoria
for each row
begin
insert into registro_log values (0,'Categoria','Delete',curdate());
end;$$
select * from registro_log;
```

- **Creación de disparador para evitar colocar una fecha que no sea la actual en historial**

Este disparador evita que el usuario ingrese una fecha que no sea la actual para el detalle.

```
DELIMITER $$
create trigger verificar_transaccion before insert on historial
for each row
begin
if new.fecha<>curdate() then
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Fecha erronea';
end if;
end;$$
```

- **Creación de disparador que evita la alteración de llaves foráneas en historial**

Este disparador alerta al usuario que lo que intenta ingresar es invalido y no permite la modificación de los datos.

```
DELIMITER $$
create trigger Confirmar_forana before update on historial
for each row
begin
declare nom_viejo varchar(50);
set nom_viejo=old.fk_reporte;
if new.fk_reporte<> nom_viejo then
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No se puede modificar este
valor';
end if;
end; $$
```


- **Creación de disparador que no permite la eliminación de una columna en el historial**

Este disparador ayuda al hecho de evitar de borrar columnas que contengan datos.

```
DELIMITER $$
create trigger fkrepote_uso before delete on historial
for each row
begin
declare epotek integer;
select count(*) into epotek from reporte where id_reporte=old.fk_reporte;
if epotek>0 then
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Este repoe se esta usando';
end if;
```

- **Creación de disparadores en la tabla log con los datos de la tabla historial**

Inserción de datos de la tabla historial en la tabla log

```
DELIMITER $$
create trigger insert_log_H after insert on historial
for each row
begin
insert into registro_log values (0,'Historial','Insert',curdate());
end;$$
```

Actualización de los datos de la tabla log

```
DELIMITER $$
create trigger update_log_H after update on Historial
for each row
begin
insert into registro_log values (0,'Historial','Update',curdate());
end;$$
```

Eliminación de los datos de historial e inserción de los mismos en la tabla log

```
DELIMITER $$
create trigger Delete_log_H after delete on Historial
for each row
begin
insert into registro_log values (0,'usuari','Delete',curdate());
end;$$
```

- **Creación de disparador que no permite la repetición del nombre de tablas ya existentes**

Este disparador tiene la finalidad de evitar que el usuario ingrese tablas inexistentes así que verifica el nombre de la tabla ingresado.

```
DELIMITER $$
create trigger tabla_verificacion before insert on registro_log
for each row
begin
if New.tabla_afectada <> 'usuario' or New.tabla_afectada <> 'reporte' or
New.tabla_afectada <> 'categoria'
or New.tabla_afectada <> 'historial' or New.tabla_afectada <> 'registro_log' then
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Tabla no valida';
end if;
end; $$
```

- **Creación de disparador que no permite la repetición del nombre de tablas ya existentes en la actualización de nombres**

Este disparador tiene la finalidad de evitar que el usuario ingrese tablas existentes en la actualización así que verifica el nombre de la tabla ingresado no se igual a otra.

```
DELIMITER $$
create trigger valores_seguimiento before update on registro_log
for each row
begin
if New.tabla_afectada <> 'usuario' or New.tabla_afectada <> 'reporte' or
New.tabla_afectada <> 'categoria'
or New.tabla_afectada <> 'historial' or New.tabla_afectada <> 'registro_log' then
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Las tablas no pueden ser
alteradas';
end if;
end; $$
```

- **Creación de disparador que evite la eliminación de datos con menos de un mes de creación**

Este disparador evita que el usuario elimine algún dato de las tablas menor a un mes de creación.

```
DELIMITER $$
create trigger Eliminar_mes before delete on registro_log
for each row
begin
declare un_mes date;
select DATE_ADD(old.fecha_cambio,interval 1 month) into un_mes from
registro_log where id_registro_log =old.id_registro_log;
if old.fecha_cambio< un_mes then
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No se puede borrar filas de
menos de un mes de antigüedad';
end if;
end;$$
```

8 AGRUPAMIENTO

- **Creación de agrupamiento de los datos por id de usuario**

Ayuda al usuario a entender mejor sus movimientos en base a su id previamente dado

```
#Tabla usuario

CREATE VIEW v_saldo_usuario AS SELECT
id_usuario,CONCAT(usuario.nombre,',',usuario.apellido) as Usuario,
ROUND((SELECT SUM(monto) FROM reporte WHERE tipo_movimiento='Ingreso')-
(SELECT SUM(monto) FROM reporte WHERE tipo_movimiento='Egreso'),2) AS Saldo
FROM REPORTE
INNER JOIN categoria ON reporte.fk_categoria= categoria.id_categoria
inner join usuario ON reporte.fk_usuario=usuario.id_usuario
GROUP BY id_usuario;
```

- **Creación de agrupamiento de datos por fecha**

El agrupamiento muestra los datos ordenados por fecha de realización de ingreso y hace mas accesible los datos al usuario.

```
CREATE VIEW v_modificacion_categoria AS SELECT
COUNT(nombre) AS Modificacion_Categorias,reporte.fecha as Ultima_modificacion
FROM categoria INNER JOIN reporte ON reporte.fk_categoria=
categoria.id_categoria
GROUP BY reporte.fecha;
```

- **Creación de agrupamiento de datos para el historial de registro**

Este agrupamiento ayuda a identificar los datos por fecha de cambio de los datos y así llevar un mejor control sobre ellos.

```
CREATE VIEW v_all_acciones_registro_log AS SELECT  
COUNT(accion) AS Total_de_acciones, fecha_cambio AS Fecha FROM registro_log  
GROUP BY fecha_cambio;
```

- **Creación de agrupamiento de datos por el nombre de la categoría**

Este agrupamiento toma los datos de la misma categoría y los une de tal manera que solo sea visible la categoría y el ingreso

```
CREATE VIEW v_ingresos_totales_reporte AS SELECT  
SUM(monto) AS Ingreso_Total, categoria.nombre FROM reporte  
INNER JOIN categoria ON reporte.fk_categoria= categoria.id_categoria  
WHERE tipo_movimiento='Ingreso'  
GROUP BY categoria.nombre;
```

- **Creación de agrupamiento de datos por el nombre de la categoría**

Este agrupamiento toma los datos de la misma categoría y los une de tal manera que solo sea visible la categoría y el egreso

```
CREATE VIEW v_egresos_totales_reporte AS SELECT  
SUM(monto) AS Egreso_Total, categoria.nombre FROM reporte  
INNER JOIN categoria ON reporte.fk_categoria= categoria.id_categoria  
WHERE tipo_movimiento='Egreso'  
GROUP BY categoria.nombre;
```

- **Creación de agrupamiento de datos por fecha en el historial**

Este agrupamiento ordena los datos por fecha de ingreso y hace un conteo de los datos y los muestra por orden ascendente.

```
#Tabla historial  
CREATE VIEW v_all_historial AS SELECT  
fecha, COUNT(reporte_mov) AS Transacciones_del_dia  
FROM historial  
GROUP BY fecha  
ORDER BY fecha asc;
```

9 ORDENAMIENTO

- **Creación de ordenamiento por fecha de manera ASC**

Este ordenamiento muestra los datos del usuario ordenados por fecha de registro y los manda a mostrar de manera ASC.

```
#Tabla General Usuario (SOLO TIENE ORDER BY)
CREATE VIEW v_informacion_general_all AS SELECT
id_usuario, CONCAT(usuario.nombre, ' ', usuario.apellido) AS Nombre_Usuario,
#Tabla usuarios
reporte.tipo_movimiento, #Tabla reporte
registro_log.accion, #Tabla log
categoria.nombre AS Nombre_Categoria, #Tabla categoria
historial.fecha #Tabla historial
FROM usuario inner join reporte ON usuario.id_usuario=reporte.fk_usuario
INNER JOIN categoria ON reporte.fk_categoria= categoria.id_categoria
INNER JOIN historial ON reporte.id_reporte=historial.fk_reporte
INNER JOIN registro_log ON historial.fk_registro_log =
registro_log.id_registro_log
ORDER BY historial.fecha asc;
```

- **Creación de ordenamiento en la tabla categorías**

Este ordenamiento tiene la finalidad de mostrar la fecha y el nombre de las categorías de forma ASC de tal manera que sea más entendible los datos.

```
#Tabla categoria SOLO ORDER BY
CREATE VIEW v_nombres_categoria AS SELECT
nombre, reporte.fecha as Ultima_modificacion
FROM categoria INNER JOIN reporte ON reporte.fk_categoria=
categoria.id_categoria
ORDER BY reporte.fecha, nombre ASC;
```

- **Creación de ordenamiento para la tabla log**

Este ordenamiento tiene la finalidad de mostrar las fechas de cambio en el registro de manera ASC de tal forma que quedan desde la más antigua a la más actual.

```
#Tabla registro_log
CREATE VIEW v_fecha_accion_registro_log AS SELECT
accion, fecha_cambio AS Fecha FROM registro_log
ORDER BY fecha_cambio asc;
```

- **Ordenamiento que muestra el monto, y la fecha del reporte de ingresos**

Este ordenamiento muestra un “historial” donde se muestra el reporte de ingresos realizados ordenados por el tipo de movimiento de manera ASC.

```
##Tablas reporte
CREATE VIEW v_resumen_ingresos_reporte AS SELECT
id_reporte,monto,categoria.nombre AS Categoria, fecha FROM reporte
INNER JOIN categoria ON reporte.fk_categoria= categoria.id_categoria
WHERE tipo_movimiento='Ingreso'
ORDER BY tipo_movimiento asc;
```

- **Ordenamiento que muestra el monto, y la fecha del reporte de egresos**

Esta vista muestra un “historial” donde se muestra el reporte de egresos realizados ordenados por el tipo de movimiento de manera ASC.

```
CREATE VIEW v_resumen_egresos_reporte AS SELECT
id_reporte,monto,categoria.nombre AS Categoria, fecha FROM reporte
INNER JOIN categoria ON reporte.fk_categoria= categoria.id_categoria
WHERE tipo_movimiento='Egreso'
ORDER BY tipo_movimiento asc;
```

- **Ordenamiento que muestra el movimiento y la fecha del movimiento de historial**

Esta vista muestra un historial donde se muestra el reporte de movimientos hechos y la fecha en que fueron realizados esto ordenado de forma ASC.

```
#Tabla historial
CREATE VIEW v_all_historial AS SELECT
fecha,COUNT(reporte_mov) AS Transacciones_del_dia
FROM historial
GROUP BY fecha
ORDER BY fecha asc;
```

10 DESARROLLO

create database SPC;

use SPC;

```
CREATE TABLE usuario (  
    id_usuario INTEGER primary key auto_increment,  
    nombre VARCHAR(30) NOT NULL,  
    apellido VARCHAR(20) NOT NULL,  
    correo VARCHAR(70) NOT NULL,  
    contrasena VARCHAR(50) NOT NULL  
);
```

```
CREATE TABLE categoria (  
    id_categoria INTEGER primary key auto_increment,  
    nombre VARCHAR(50) NOT NULL  
);
```

```
CREATE TABLE registro_log(  
    id_registro_log integer auto_increment primary key,  
    tabla_afectada varchar(15) not null,  
    accion varchar(10) not null,  
    fecha_cambio date not null  
);
```

```
CREATE TABLE cuenta_usuario(  
    id_cuenta_usuario integer primary key auto_increment,  
    nombre_banco varchar(20) not null,  
    num_cuenta integer not null,  
    fk_id_usuario INTEGER not null,  
    foreign key(fk_id_usuario) references usuario (id_usuario)  
);
```

```

CREATE TABLE reporte (
    id_reporte INTEGER primary key auto_increment,
    monto double not null,
    fecha date not null,
    concepto varchar(255),
    tipo_movimiento varchar(10) not null,
    fk_cuenta_usuario integer not null,
    fk_categoria integer,
    foreign key (fk_cuenta_usuario) references cuenta_usuario (id_cuenta_usuario),
    foreign key(fk_categoria) references categoria (id_categoria)
);

```

```

CREATE TABLE historial (
    id_historial INTEGER NOT NULL primary key,
    fecha DATE not null,
    reporte_mov VARCHAR(10),
    fk_reporte INTEGER NOT NULL,
    foreign key (fk_reporte) references reporte (id_reporte)
);

```

```

CREATE VIEW v_saldo_usuario AS SELECT
id_usuario,CONCAT(usuario.nombre,',',usuario.apellido) as Usuario,
ROUND((SELECT SUM(monto) FROM reporte WHERE tipo_movimiento='Ingreso')-
(SELECT SUM(monto) FROM reporte WHERE tipo_movimiento='Egreso'),2) AS Saldo FROM
usuario
inner join cuenta_usuario ON cuenta_usuario.fk_id_usuario=usuario.id_usuario
inner join reporte ON cuenta_usuario.id_cuenta_usuario=reporte.fk_cuenta_usuario
INNER JOIN categoria ON reporte.fk_categoria= categoria.id_categoria
GROUP BY id_usuario;

```



```

CREATE VIEW v_informacion_general_all AS SELECT
id_usuario,CONCAT(usuario.nombre,' ',usuario.apellido) AS Nombre_Usuario,
reporte.tipo_movimiento, #Tabla reporte
categoria.nombre AS Nombre_Categoria, #Tabla categoria
historial.fecha #Tabla historial
FROM usuario
inner join cuenta_usuario ON cuenta_usuario.fk_id_usuario=usuario.id_usuario
inner join reporte ON cuenta_usuario.id_cuenta_usuario=reporte.fk_cuenta_usuario
INNER JOIN categoria ON reporte.fk_categoria= categoria.id_categoria
inner join historial ON historial.fk_reporte=reporte.id_reporte
ORDER BY historial.fecha desc;

```

```

CREATE VIEW v_nombres_categoria AS SELECT
nombre,reporte.fecha as Ultima_modificacion
FROM categoria INNER JOIN reporte ON reporte.fk_categoria= categoria.id_categoria
ORDER BY reporte.fecha DESC;

```

```

CREATE VIEW v_modificacion_categoria AS SELECT
COUNT(nombre) AS Modificacion_Categorias,reporte.fecha as Ultima_modificacion
FROM categoria INNER JOIN reporte ON reporte.fk_categoria= categoria.id_categoria
GROUP BY reporte.fecha;

```

```

CREATE VIEW v_fecha_accion_registro_log AS SELECT
accion, fecha_cambio AS Fecha FROM registro_log
ORDER BY fecha_cambio desc;

```

```

CREATE VIEW v_all_acciones_registro_log AS SELECT
COUNT(accion) AS Total_de_acciones, fecha_cambio AS Fecha FROM registro_log
GROUP BY fecha_cambio;

```

```
CREATE VIEW v_resumen_ingresos_reporte AS SELECT
id_reporte, monto, categoria.nombre AS Categoria, fecha FROM reporte
INNER JOIN categoria ON reporte.fk_categoria= categoria.id_categoria
WHERE tipo_movimiento='Ingreso'
ORDER BY fecha desc;
```

```
CREATE VIEW v_resumen_egresos_reporte AS SELECT
id_reporte, monto, categoria.nombre AS Categoria, fecha FROM reporte
INNER JOIN categoria ON reporte.fk_categoria= categoria.id_categoria
WHERE tipo_movimiento='Egreso'
ORDER BY fecha desc;
```

```
CREATE VIEW v_ingresos_totales_categoria_reporte AS SELECT
SUM(monto) AS Ingreso_Total, categoria.nombre FROM reporte
INNER JOIN categoria ON reporte.fk_categoria= categoria.id_categoria
WHERE tipo_movimiento='Ingreso'
GROUP BY categoria.nombre;
```

```
CREATE VIEW v_egresos_totales_categoria_reporte AS SELECT
SUM(monto) AS Egreso_Total, categoria.nombre FROM reporte
INNER JOIN categoria ON reporte.fk_categoria= categoria.id_categoria
WHERE tipo_movimiento='Egreso'
GROUP BY categoria.nombre;
```

```
CREATE VIEW v_cuentas_bancarias_cuenta_usuario AS SELECT
nombre_banco AS Nombre_del_Banco, num_cuenta AS Numero_de_Cuenta,
CONCAT(usuario.nombre, ' ', usuario.apellido) AS Nombre_Usuario
FROM cuenta_usuario
INNER JOIN usuario ON usuario.id_usuario=cuenta_usuario.fk_id_usuario
ORDER BY nombre_banco asc;
```

```
CREATE VIEW v_all_cuentas_bancarias_cuenta_usuario AS SELECT
CONCAT(usuario.nombre,' ',usuario.apellido) AS Nombre_Usuario,
COUNT(*) AS Numero_Cuentas_Bancarias
FROM cuenta_usuario
INNER JOIN usuario ON usuario.id_usuario=cuenta_usuario.fk_id_usuario
GROUP BY fk_id_usuario;
```

```
CREATE VIEW v_all_historial AS SELECT
fecha,COUNT(reporte_mov) AS Transacciones_del_dia
FROM historial
GROUP BY fecha
ORDER BY fecha desc;
```

```
CREATE INDEX id_usuarios ON usuario ( id_usuario );
```

```
create index Datos_app_usuario on usuario (id_usuario,correo,contrasena);
```

```
CREATE UNIQUE INDEX Datos_usuario ON usuario (id_usuario,nombre,apellido);
```

```
create index id_categoria on categoria (id_categoria);
```

```
create index datos_categoria on categoria (id_categoria,nombre);
```

```
create index registro_log on registro_log (id_registro_log);
```

```
create index cambios_tablas on registro_log (tabla_afectada,accion,fecha_cambio);
```

```
CREATE UNIQUE INDEX fecha_cambios on registro_log (accion,fecha_cambio);
```

```
create index id_reporte on reporte (id_reporte);
```

```
create index fecha_movimientos on reporte (id_reporte,fecha,tipo_movimiento);
```

```
CREATE UNIQUE INDEX usuario_movimientos on reporte (fk_cuenta_usuario,tipo_movimiento);
```

```
create index id_historial on historial (id_historial);
```

```
create index fecha_registro on historial (fecha,fk_reporte );
```

```
CREATE UNIQUE INDEX historial_reporte_registro on historial (id_historial,fk_reporte);
```

```
DELIMITER $$
```

```
create trigger verificacion_categoria before insert on categoria
```

```
for each row
```

```
begin
```

```
declare categoria_blanco varchar(1);
```

```
set categoria_blanco=' ';
```

```
if new.nombre<=>categoria_blanco then
```

```
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No se admiten categorias en blanco';
```

```
end if;
```

```
end; $$
```

DELIMITER \$\$

```
create trigger nom_repetido before update on categoria
for each row
begin
declare nom_viejo varchar(50);
set nom_viejo=old.nombre;
if new.nombre<=> nom_viejo then
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No hay ninun cambio de nombre';
end if;
end; $$
```

DELIMITER \$\$

```
create trigger categoria_usada before delete on categoria
for each row
begin
declare num_categoria integer;
SELECT
    COUNT(*)
INTO num_categoria FROM
    reporte
WHERE
    fk_categoria = old.id_categoria;
if num_categoria>0 then
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Esta categoria se esta usando';
end if;
end; $$
```

DELIMITER \$\$

```
create trigger insert_log after insert on categoria
for each row
begin
insert into registro_log values (0,'Categoria','Insert',curdate());
end;$$
```

DELIMITER \$\$

```
create trigger update_log after update on categoria
for each row
begin
insert into registro_log values (0,'Categoria','Update',curdate());
end;$$
```

DELIMITER \$\$

```
create trigger Delete_log after delete on categoria
for each row
begin
insert into registro_log values (0,'Categoria','Delete',curdate());
end;$$
```

DELIMITER \$\$

```
create trigger verificar_transaccion before insert on historial
for each row
begin
if new.fecha<>curdate() then
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Fecha erronea';
end if;
end;$$
```

DELIMITER \$\$

```
create trigger Confirmar_forana before update on historial
for each row
begin
declare nom_viejo varchar(50);
set nom_viejo=old.fk_reporte;
if new.fk_reporte<> nom_viejo then
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No se puede modificar este valor';
end if;
end; $$
```

DELIMITER \$\$

```
create trigger fkrepote_uso before delete on historial
for each row
begin
declare epotek integer;
select count(*) into epotek from reporte where id_reporte=old.fk_reporte;
if epotek>0 then
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Este repoe se esta usando';
end if;
end; $$
```

DELIMITER \$\$

```
create trigger insert_log_H after insert on historial
for each row
begin
insert into registro_log values (0,'Historial','Insert',curdate());
end; $$
```

DELIMITER \$\$

```
create trigger update_log_H after update on Historial
for each row
begin
insert into registro_log values (0,'Historial','Update',curdate());
end;$$
```

DELIMITER \$\$

```
create trigger Delete_log_H after delete on Historial
for each row
begin
insert into registro_log values (0,'usuari','Delete',curdate());
end;$$
```

DELIMITER \$\$

```
create trigger tabla_verificacion before insert on registro_log
for each row
begin
if New.tabla_afectada <> 'usuario' or New.tabla_afectada <> 'reporte' or New.tabla_afectada <>
'categoria'
or New.tabla_afectada <> 'historial' or New.tabla_afectada <> 'registro_log' then
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Tabla no valida';
end if;
end; $$
```


DELIMITER \$\$

```
create trigger valores_seguimiento before update on registro_log
for each row
begin
if New.tabla_afectada <> 'usuario' or New.tabla_afectada <> 'reporte' or New.tabla_afectada <>
'categoria'
or New.tabla_afectada <> 'historial' or New.tabla_afectada <> 'registro_log' then
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Las tablas no pueden ser alteradas';
end if;
end; $$
```

DELIMITER \$\$

```
create trigger Eliminar_mes before delete on registro_log
for each row
begin
declare un_mes date;

select DATE_ADD(old.fecha_cambio,interval 1 month) into un_mes from registro_log where
id_registro_log =old.id_registro_log;

if old.fecha_cambio< un_mes then

SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No se puede borrar filas de menos de un
mes de antigüedad';

end if;
end;$$
```

```

DELIMITER $$

CREATE PROCEDURE insertar_usuario(nombre_new varchar(30),apellido_new
varchar(20), correo_new varchar(70), contrasena_new varchar(50))
begin
DECLARE EXIT HANDLER FOR SQLEXCEPTION
BEGIN
    ROLLBACK;
    SELECT 'Error al insertar usuario' AS mensaje;
END;

set autocommit = 0;
start transaction;

if exists(select * from usuario where nombre=nombre_new and
apellido=apellido_new and correo=correo_new and contrasena=contrasena_new )
THEN
select "Usuario ya existente" as mensaje;
rollback;
else
insert into usuario (nombre,apellido,correo,contrasena) values (nombre_new,
apellido_new, correo_new,contrasena_new);
end if;

set autocommit = 1;
end; $$

```

```

DELIMITER $$

create procedure actualizar_usuario(id_usq integer,nombre_new
varchar(30),apellido_new varchar(20), correo_new varchar(70), contrasena_new
varchar(50))
begin
DECLARE EXIT HANDLER FOR SQLEXCEPTION
BEGIN
    ROLLBACK;
    SELECT 'Error al actualizar usuario' AS mensaje;
END;

set autocommit = 0;
start transaction;

select * from usuario where id_usuario = id_usq FOR UPDATE;
if exists(select * from usuario where id_usuario = id_usq) then
update usuario set nombre=nombre_new, apellido=apellido_new, correo=correo_new,
contrasena=contrasena_new where id_usuario = id_usq;
commit;
else
select 'El usuario no existe ' as mensaje;
rollback;
end if;

set autocommit =1;
end; $$

```

DELIMITER \$\$

create procedure consultar_usuario (id_consultar varchar(255))

begin

 set autocommit = 0;

 if id_consultar="" then

 select * from usuario;

 else

 select * from usuario where id_usuario=id_consultar;

 end if;

 set autocommit = 1;

end; \$\$

DELIMITER \$\$

CREATE PROCEDURE eliminacion_usuario(id_usuario_new INTEGER)

BEGIN

 DECLARE EXIT HANDLER FOR SQLEXCEPTION

 BEGIN

 ROLLBACK;

 SELECT 'Error al eliminar el usuario' AS mensaje;

 END;

 SET autocommit = 0;

 START TRANSACTION;

 DELETE FROM usuario WHERE id_usuario = id_usuario_new;

 IF ROW_COUNT() = 0 THEN

 SELECT 'El usuario no existe' AS mensaje;

 ROLLBACK;

 ELSE

 SELECT 'Usuario eliminado correctamente' AS mensaje;

 COMMIT;

```

    END IF;

END; $$

DELIMITER $$

CREATE PROCEDURE insertar_reporte(p_monto DOUBLE, p_fecha DATE, p_concepto
VARCHAR(255), p_tipo_movimiento VARCHAR(10),
p_fk_cuenta_usuario INTEGER, p_fk_categoria INTEGER)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        SELECT 'Error al insertar el reporte' AS mensaje;
    END;

    SET autocommit = 0;
    START TRANSACTION;

    INSERT INTO reporte (monto, fecha, concepto, tipo_movimiento, fk_cuenta_usuario, fk_categoria)
    VALUES (p_monto, p_fecha, p_concepto, p_tipo_movimiento, p_fk_cuenta_usuario,
p_fk_categoria);

    SET autocommit = 1;
    COMMIT;
    SELECT 'Reporte insertado correctamente' AS mensaje;
END; $$

```

```

DELIMITER $$

CREATE PROCEDURE eliminar_reporte(p_id_reporte INTEGER)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        SELECT 'Error al eliminar el reporte' AS mensaje;
    END;
    SET autocommit = 0;
    START TRANSACTION;
    DELETE FROM reporte WHERE id_reporte = p_id_reporte;
    IF ROW_COUNT() = 0 THEN
        SELECT 'El reporte no existe' AS mensaje;
        ROLLBACK;
    ELSE
        SELECT 'Reporte eliminado correctamente' AS mensaje;
        COMMIT;
    END IF;
END; $$

```

```

DELIMITER $$

CREATE PROCEDURE cuenta_usada(num_cuenta_new INTEGER)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        SELECT 'Error al comprobar la cuenta de banco' AS mensaje;
    END;

```

```

SET autocommit = 0;
START TRANSACTION;

IF EXISTS(SELECT * FROM cuenta_usuario WHERE num_cuenta = num_cuenta_new) THEN
    SELECT 'Esa cuenta de banco ya ha sido registrada' AS mensaje;
    ROLLBACK;
ELSE
    SELECT 'La cuenta de banco no ha sido utilizada' AS mensaje;
    COMMIT;
END IF;
SET autocommit = 1;
END $$

DELIMITER $$
CREATE PROCEDURE eliminar_cuenta(id_cuenta_usuario_new INTEGER)
BEGIN
    SET autocommit = 0;
    START TRANSACTION;

    IF NOT EXISTS (SELECT * FROM cuenta_usuario WHERE id_cuenta_usuario =
id_cuenta_usuario_new) THEN
        SELECT 'La cuenta de usuario no existe' AS mensaje;
        ROLLBACK;
    ELSE
        DELETE FROM cuenta_usuario WHERE id_cuenta_usuario = id_cuenta_usuario_new;
        SELECT 'Cuenta de usuario eliminada correctamente' AS mensaje;
        COMMIT;
    END IF;
    SET autocommit = 1;
END; $$

```

DELIMITER \$\$

CREATE PROCEDURE categoria_repetida(id_categoria_new INTEGER, nombre_new VARCHAR(50))

BEGIN

DECLARE EXIT HANDLER FOR SQLEXCEPTION

BEGIN

ROLLBACK;

SELECT 'Error al comprobar la categoría' AS mensaje;

END;

SET autocommit = 0;

START TRANSACTION;

IF EXISTS(SELECT * FROM categoria WHERE nombre = nombre_new) THEN

SELECT 'Esta categoría ya existe' AS mensaje;

ROLLBACK;

ELSE

INSERT INTO categoria (id_categoria, nombre) VALUES (id_categoria_new, nombre_new);

COMMIT;

SELECT 'Categoría insertada correctamente' AS mensaje;

END IF;

SET autocommit = 1;

END; \$\$

DELIMITER \$\$

CREATE PROCEDURE eliminar_categoria(id_categoria_new INTEGER)

BEGIN

DECLARE EXIT HANDLER FOR SQLEXCEPTION

BEGIN

ROLLBACK;

SELECT 'Error al eliminar la categoría' AS mensaje;

END;


```

SET autocommit = 0;
START TRANSACTION;
DELETE FROM categoria WHERE id_categoria = id_categoria_new;
IF ROW_COUNT() = 0 THEN
    SELECT 'La categoría no existe' AS mensaje;
    ROLLBACK;
ELSE
    SELECT 'Categoría eliminada correctamente' AS mensaje;
    COMMIT;
END IF;
END; $$

DELIMITER $$

CREATE PROCEDURE actualizar_usuario_correo(id_usuario_new INTEGER, nuevo_correo_new
VARCHAR(70))
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        SELECT 'Error al actualizar el correo del usuario' AS mensaje;
    END;
    SET autocommit = 0;
    START TRANSACTION;
    UPDATE usuario
    SET correo = nuevo_correo_new
    WHERE id_usuario = id_usuario_new;

```

```

IF ROW_COUNT() = 0 THEN
    SELECT 'El usuario no existe' AS mensaje;
    ROLLBACK;
ELSE
    SELECT 'Correo actualizado correctamente' AS mensaje;
    COMMIT;
END IF;
END; $$

DELIMITER $$
CREATE PROCEDURE nombre_banco_indiv (nombre_b varchar(20))
BEGIN

    IF EXISTS(select nombre_banco from cuenta_usuario
where nombre_b=nombre_banco) THEN

        SELECT * FROM v_cuentas_bancarias_cuenta_usuario where
nombre_b=Nombre_del_Banco;

    ELSE

        SELECT "No se pudo encontrar el banco" as mensaje;

    END IF;

END;$$

```

DELIMITER \$\$

CREATE PROCEDURE usuario_unico_id (usuario_unico integer)

BEGIN

IF EXISTS(select id_usuario from usuario
where id_usuario=usuario_unico) THEN

SELECT * FROM v_saldo_usuario where usuario_unico=id_usuario;

ELSE

SELECT "No se pudo encontrar el usuario" as mensaje;

END IF;

END;\$\$

DELIMITER \$\$

CREATE PROCEDURE busqueda_movimientos (fecha_busc integer)

BEGIN

DECLARE EXIT HANDLER FOR SQLEXCEPTION

BEGIN

SELECT CONCAT(' El dato "',fecha_busc,'" no es valido, utilice un numero entre
01 a 12') as mensaje;

END;

IF EXISTS(SELECT fecha from reporte where EXTRACT(MONTH FROM fecha)=fecha_busc)
THEN

SELECT * FROM REPORTE WHERE EXTRACT(MONTH FROM fecha)=fecha_busc;

ELSE

SELECT "No hubo ningun movimiento en esa fecha" as mensaje;

END IF;

END;\$\$

11 PROCEDIMIENTOS ALMACENADOS

- **Procedimiento para inserción de usuarios**

El procedimiento creado tiene como fin la inserción de usuarios, por lo que la transacción utilizada valida que no exista un registro con exactamente la misma información y caso de que se encuentre muestra un mensaje de “Usuario ya existente”.

```
#Insert
DELIMITER $$
CREATE PROCEDURE insertar_usuario(nombre_new varchar(30),apellido_new
varchar(20), correo_new varchar(70), contrasena_new varchar(50))
begin
set autocommit = 0; #<-- esto siempre va
start transaction; #<-- esto siempre va
#todas las acciones de la transaccion
#validar que no exista un registro con exactamente la misma informacion
#Por lo general en los inserts siempre vamos a hacer una validacion asi
if exists(select * from usuario where nombre=nombre_new and
apellido=apellido_new and correo=correo_new and contrasena=contrasena_new )
THEN
select 'Usuario ya existente' as mensaje;
rollback;
ELSE
insert into usuario (nombre,apellido,correo,contrasena) values (nombre_new,
apellido_new, correo_new,contrasena_new);

END IF;|
set autocommit = 1; #<-- esto siempre va
end; $$
```

- **Procedimiento para actualización de usuarios**

El procedimiento creado tiene como fin la actualización de datos de los usuarios, por lo que la transacción utilizada valida que el usuario exista y caso de que no se encuentre muestra un mensaje de “El usuario no existe”.

```
#ACTUALIZAR
DELIMITER $$
create procedure actualizar_usuario(id_usq integer,nombre_new
varchar(30),apellido_new varchar(20), correo_new varchar(70), contrasena_new
varchar(50))
begin
set autocommit = 0;
start transaction;
select * from usuario where id_usuario = id_usq FOR UPDATE;
if exists(select * from usuario where id_usuario = id_usq) then
update usuario set nombre=nombre_new, apellido=apellido_new, correo=correo_new,
contrasena=contrasena_new where id_usuario = id_usq;
commit;
else
select 'El usuario no existe' as mensaje;
rollback;
end if;
set autocommit =1;
end; $$
```

- **Procedimiento para consulta de usuarios**

Este procedimiento tiene como fin la consulta del usuario

```
DELIMITER $$
create procedure consultar_usuario (id_consultar varchar(255))
begin
    set autocommit = 0;
    if id_consulta='' then
        select * from usuario;
    else
        select * from usuario where id_usuario=id_consultar;
    end if;
    set autocommit = 1;
end; $$
```

- **Procedimiento para eliminación de usuarios**

Este procedimiento verifica que el usuario exista para poder eliminar, en caso contrario manda un mensaje de “El usuario no existe” y caso de entrar en error manda mensaje de “Error al eliminar reporte, si esto no ocurre se manda un mensaje de “Usuario eliminado correctamente”.

```
#eliminación
#|create procedure
DELIMITER $$
CREATE PROCEDURE eliminacion_usuario(id_usuario_new INTEGER)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        SELECT 'Error al eliminar el usuario' AS mensaje;
    END;

    SET autocommit = 0;
    START TRANSACTION;

    DELETE FROM usuario WHERE id_usuario = id_usuario_new;

    IF ROW_COUNT() = 0 THEN
        SELECT 'El usuario no existe' AS mensaje;
        ROLLBACK;
    ELSE
        SELECT 'Usuario eliminado correctamente' AS mensaje;
        COMMIT;
    END IF;

END; $$
```

- **Procedimiento para la inserción de un nuevo reporte**

Este procedimiento se encarga de insertar el reporte en caso de que entre en error manda un mensaje de “Error al insertar reporte” si esto no ocurre se manda un mensaje notificando la inserción de datos con un mensaje “Reporte insertado correctamente”.

```
#inserta nuevo reporte
DELIMITER $$
CREATE PROCEDURE insertar_reporte(p_monto DOUBLE, p_fecha DATE, p_concepto VARCHAR(255), p_tipo_movimiento VARCHAR(10), p_fk_cuenta_usuario INTEGER, p_fk_categoria INTEGER)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        SELECT 'Error al insertar el reporte' AS mensaje;
    END;

    SET autocommit = 0;
    START TRANSACTION;

    INSERT INTO reporte (monto, fecha, concepto, tipo_movimiento, fk_cuenta_usuario, fk_categoria)
    VALUES (p_monto, p_fecha, p_concepto, p_tipo_movimiento, p_fk_cuenta_usuario, p_fk_categoria);

    SET autocommit = 1;
    COMMIT;
    SELECT 'Reporte insertado correctamente' AS mensaje;
END; $$
```

- **Procedimiento que elimina un reporte con su ID**

Este procedimiento verifica que el reporte exista en caso de que no sea así manda un mensaje de “El reporte no existe” y si este entra en error se mandara un mensaje de “Error al eliminar reporte”, si el reporte se inserta se mandara un mensaje que notifica que “Reporte eliminado correctamente”.

```
#eliminar un reporte con su id
DELIMITER $$
CREATE PROCEDURE eliminar_reporte(p_id_reporte INTEGER)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        SELECT 'Error al eliminar el reporte' AS mensaje;
    END;

    SET autocommit = 0;
    START TRANSACTION;

    DELETE FROM reporte WHERE id_reporte = p_id_reporte;

    IF ROW_COUNT() = 0 THEN
        SELECT 'El reporte no existe' AS mensaje;
        ROLLBACK;
    ELSE
        SELECT 'Reporte eliminado correctamente' AS mensaje;
        COMMIT;
    END IF;
END; $$
```

- **Procedimiento verifica si una cuenta de banco se usa en más de una transacción**

Este procedimiento verifica si la cuenta se usa en más de una transacción en caso de que no sea así manda un mensaje de “La cuenta de banco no ha sido utilizada” y si este entra en error se mandara un mensaje de “Error al comprobar la cuenta de banco”, si la cuenta existe se mandara un mensaje que notifica que “Esa cuenta de banco ya ha sido registrada”.

```
#cuenta_usuario
#sirve para ver si una cuenta de banco se usa en más de una cuenta
#inserción
DELIMITER $$
CREATE PROCEDURE cuenta_usada(num_cuenta_new INTEGER)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        SELECT 'Error al comprobar la cuenta de banco' AS mensaje;
    END;

    SET autocommit = 0;
    START TRANSACTION;

    #Validar si el número de cuenta ya ha sido registrado
    IF EXISTS(SELECT * FROM cuenta_usuario WHERE num_cuenta = num_cuenta_new) THEN
        SELECT 'Esa cuenta de banco ya ha sido registrada' AS mensaje;
        ROLLBACK;
    ELSE
        SELECT 'La cuenta de banco no ha sido utilizada' AS mensaje;
        COMMIT;
    END IF;

    SET autocommit = 1;
END $$
```

- **Procedimiento para eliminar cuenta de usuario (banco)**

Este procedimiento elimina la cuenta de usuario donde primero verifica que la cuenta exista y si esto es así se manda un mensaje “Cuenta de usuario eliminada correctamente” en caso contrario mandara un mensaje “La cuenta de usuario no existe”.

```
-- eliminación
SELECT * FROM cuenta_usuario;
DELIMITER $$
CREATE PROCEDURE eliminar_cuenta(id_cuenta_usuario_new INTEGER)
BEGIN
    SET autocommit = 0;
    START TRANSACTION;

    IF NOT EXISTS (SELECT * FROM cuenta_usuario WHERE id_cuenta_usuario = id_cuenta_usuario_new) THEN
        SELECT 'La cuenta de usuario no existe' AS mensaje;
        ROLLBACK;
    ELSE
        DELETE FROM cuenta_usuario WHERE id_cuenta_usuario = id_cuenta_usuario_new;
        SELECT 'Cuenta de usuario eliminada correctamente' AS mensaje;
        COMMIT;
    END IF;

    SET autocommit = 1;
END; $$
```

- **Procedimiento que inserta categoría**

Este procedimiento inserta categoría donde primero verifica que no haya una categoría llamada de la misma manera y si esto es así se manda un mensaje “Esta categoría ya existe” en caso contrario mandara un mensaje “Categoría insertada correctamente” pero si el procedimiento entra en error mandara un mensaje “Error al comprobar la categoría”.

```
-- categoria
-- asegurarse que no haya categorias con el mismo nombre
-- inserción
DELIMITER $$
CREATE PROCEDURE categoria_repetida(id_categoria_new INTEGER, nombre_new VARCHAR(50))
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        SELECT 'Error al comprobar la categoría' AS mensaje;
    END;

    SET autocommit = 0;
    START TRANSACTION;

    IF EXISTS(SELECT * FROM categoria WHERE nombre = nombre_new) THEN
        SELECT 'Esta categoría ya existe' AS mensaje;
        ROLLBACK;
    ELSE
        INSERT INTO categoria (id_categoria, nombre) VALUES (id_categoria_new, nombre_new);
        COMMIT;
        SELECT 'Categoría insertada correctamente' AS mensaje;
    END IF;

    SET autocommit = 1;
END; $$
```

- **Procedimiento para la eliminación de categoría**

Este procedimiento elimina la categoría donde primero verifica que la categoría exista y si esto es así se manda un mensaje “Categoría eliminada correctamente” en caso contrario mandara un mensaje “La categoría no existe” y si esto entra en error notifica con un mensaje “Error al eliminar la categoría”.

```
-- eliminación
-- eliminar una categoría
-- categoria
DELIMITER $$
CREATE PROCEDURE eliminar_categoria(id_categoria_new INTEGER)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        SELECT 'Error al eliminar la categoría' AS mensaje;
    END;

    SET autocommit = 0;
    START TRANSACTION;

    DELETE FROM categoria WHERE id_categoria = id_categoria_new;

    IF ROW_COUNT() = 0 THEN
        SELECT 'La categoría no existe' AS mensaje;
        ROLLBACK;
    ELSE
        SELECT 'Categoría eliminada correctamente' AS mensaje;
        COMMIT;
    END IF;
END; $$
```


- **Procedimiento para cambiar el correo del usuario**

Este procedimiento actualiza el correo donde primero verifica que el usuario exista y si esto es así se manda un mensaje “Correo actualizado correctamente” en caso contrario mandara un mensaje “El usuario no existe” y si esto entra en error notifica con un mensaje “Error al actualizar el correo del usuario”.

```
#usuario
#cambiar correo
DELIMITER $$
CREATE PROCEDURE actualizar_usuario_correo(id_usuario_new INTEGER, nuevo_correo_new VARCHAR(70))
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        SELECT 'Error al actualizar el correo del usuario' AS mensaje;
    END;

    SET autocommit = 0;
    START TRANSACTION;

    UPDATE usuario
    SET correo = nuevo_correo_new
    WHERE id_usuario = id_usuario_new;

    IF ROW_COUNT() = 0 THEN
        SELECT 'El usuario no existe' AS mensaje;
        ROLLBACK;
    ELSE
        SELECT 'Correo actualizado correctamente' AS mensaje;
        COMMIT;
    END IF;

END; $$
```

- **Procedimiento para consulta individual**

Este procedimiento consulta de manera individual a los usuarios mediante su ID, si la ID que se quiere consultar existe, muestra la vista v_saldo_usuario, si no existe la ID, lanza un error: "No se pudo encontrar el usuario".

```
DELIMITER $$
CREATE PROCEDURE usuario_unico_id (usuario_unico integer)
BEGIN
    IF EXISTS(select id_usuario from usuario
    where id_usuario=usuario_unico) THEN
        SELECT * FROM v_saldo_usuario where usuario_unico=id_usuario;
    ELSE
        SELECT "No se pudo encontrar el usuario" as mensaje;
    END IF;
END;$$
#Este procedimiento consulta de manera individual a los usuarios mediante su ID,
# si la ID que se quiere consultar existe, muestra la vista v_saldo_usuario,
# Si no existe la ID, lanza un error: "No se pudo encontrar el usuario".|
```

- **Procedimiento consulta el nombre individual del banco**

Procedimiento que en caso de existir el nombre del banco que se escribe, te mostrará la vista v_cuentas_bancarias_cuenta_usuario que incluye el nombre de banco, num. de cuenta y nombre del usuario, en caso de que no exista el banco escrito lanzara el mensaje "No se pudo encontrar el banco".

```
#Procedimientos
DELIMITER $$
CREATE PROCEDURE nombre_banco_indiv (nombre_b varchar(20))
BEGIN
    IF EXISTS(select nombre_banco from cuenta_usuario
where nombre_b=nombre_banco) THEN
        SELECT * FROM v_cuentas_bancarias_cuenta_usuario where nombre_b=Nombre_del_Banco;
    ELSE
        SELECT "No se pudo encontrar el banco" as mensaje;
    END IF;
END;$$
```

- **Procedimiento para búsqueda de movimientos**

Procedimiento que consulta los movimientos que se hicieron en X mes, primero comprueba si hay algún movimiento que se haya hecho en el número del mes escrito, si es así muestra todos los resultados en el que coincidan ese mes y año.

```
DELIMITER $$
CREATE PROCEDURE busqueda_movimientos (fecha_busc integer)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        SELECT CONCAT(' El dato "',fecha_busc,'" no es valido, utilice un numero entre 01 a 12') as mensaje;
    END;
    IF EXISTS(SELECT fecha from reporte where EXTRACT(MONTH FROM fecha)=fecha_busc) THEN
        SELECT * FROM REPORTE WHERE EXTRACT(MONTH FROM fecha)=fecha_busc;
    ELSE
        SELECT "No hubo ningun movimiento en esa fecha" as mensaje;
    END IF;
END;$$
```

12DESARROLLO DE PROCEDIMIENTOS ALMACENADOS

Pruebas

- Procedimiento para inserción de usuarios

```
DELIMITER $$
CREATE PROCEDURE insertar_usuario(nombre_new varchar(30),apellido_new
varchar(20), correo_new varchar(70), contrasena_new varchar(50))
begin
DECLARE EXIT HANDLER FOR SQLEXCEPTION
BEGIN
    ROLLBACK;
    SELECT 'Error al insertar usuario' AS mensaje;
END;
set autocommit = 0;
start transaction;

if exists(select * from usuario where nombre=nombre_new and
apellido=apellido_new and correo=correo_new and contrasena=contrasena_new )
THEN
select "Usuario ya existente" as mensaje;
rollback;
else
insert into usuario (nombre,apellido,correo,contrasena) values (nombre_new,
apellido_new, correo_new,contrasena_new);
end if;
set autocommit = 1;

end; $$

select * from usuario;
CALL insertar_usuario('John', 'Doe', 'john.doe@example.com', 'mypassword');
```

Inserción

```
80 select * from usuario;
81 CALL insertar_usuario('John', 'Doe', 'john.doe@example.com', 'mypassword');
82
83
84 #ACTUALIZAR
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

mensaje
Usuario ya existente

- Procedimiento para actualización de usuarios

```
DELIMITER $$
create procedure actualizar_usuario(id_usq integer,nombre_new
varchar(30),apellido_new varchar(20), correo_new varchar(70), contraseña_new
varchar(50))
begin
DECLARE EXIT HANDLER FOR SQLEXCEPTION
BEGIN
    ROLLBACK;
    SELECT 'Error al actualizar usuario' AS mensaje;
END;
set autocommit = 0;
start transaction;
select * from usuario where id_usuario = id_usq FOR UPDATE;
if exists(select * from usuario where id_usuario = id_usq) then
update usuario set nombre=nombre_new, apellido=apellido_new, correo=correo_new,
contrasena=contrasena_new where id_usuario = id_usq;
commit;
else
select 'El usuario no existe ' as mensaje;
rollback;
end if;
set autocommit =1;
end; $$
```

Prueba

	id_usuario	nombre	apellido	correo	contrasena
▶	1	John	Doe	john.doe@example.com	mypassword
*	NULL	NULL	NULL	NULL	NULL

Actualización

```
109 CALL actualizar_usuario(1, '1NuevoNombre', '2NuevoApellido', 'nuevo1@example.com', 'nuevacontrasena1');
110 select * from usuario;
111
```

Result Grid Filter Rows: Edit: Export/Import: Wrap Cell Content:					
	id_usuario	nombre	apellido	correo	contrasena
▶	1	1NuevoNombre	2NuevoApellido	nuevo1@example.com	nuevacontrasena1
*	NULL	NULL	NULL	NULL	NULL

- Procedimiento para consulta de usuarios

```
#CONSULTAR
DELIMITER $$
create procedure consultar_usuario (id_consultar varchar(255))
begin
    set autocommit = 0;
    if id_consultar='' then
        select * from usuario;
    else
        select * from usuario where id_usuario=id_consultar;
    end if;
    set autocommit = 1;
end; $$
```

Prueba

```
124
125 call consultar_usuario(1);
126
```

	id_usuario	nombre	apellido	correo	contrasena
▶	1	1NuevoNombre	2NuevoApellido	nuevo1@example.com	nuevacontrasena1

- Procedimiento para eliminación de usuarios

```
#eliminación
#create procedure
DELIMITER $$
CREATE PROCEDURE eliminacion_usuario(id_usuario_new INTEGER)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        SELECT 'Error al eliminar el usuario' AS mensaje;
    END;

    SET autocommit = 0;
    START TRANSACTION;

    DELETE FROM usuario WHERE id_usuario = id_usuario_new;

    IF ROW_COUNT() = 0 THEN
        SELECT 'El usuario no existe' AS mensaje;
        ROLLBACK;
    ELSE
        SELECT 'Usuario eliminado correctamente' AS mensaje;
        COMMIT;
    END IF;

END; $$
```

```

152
153  call eliminacion_usuario(1);
154
155  #Reportes

```

Result Grid		Filter Rows:	Export:
	mensaje		
▶	Usuario eliminado correctamente		


```

152
153  call eliminacion_usuario(1);
154
155  #Reportes

```

Result Grid		Filter Rows:	Export:
	mensaje		
▶	El usuario no existe		

- Procedimiento para la inserción de un nuevo reporte

```

#inserta nuevo reporte
DELIMITER $$
CREATE PROCEDURE insertar_reporte(p_monto DOUBLE, p_fecha DATE, p_concepto VARCHAR(255), p_tipo_movimiento VARCHAR(10),
p_fk_cuenta_usuario INTEGER, p_fk_categoria INTEGER)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        SELECT 'Error al insertar el reporte' AS mensaje;
    END;

    SET autocommit = 0;
    START TRANSACTION;




    INSERT INTO reporte (monto, fecha, concepto, tipo_movimiento, fk_cuenta_usuario, fk_categoria)
    VALUES (p_monto, p_fecha, p_concepto, p_tipo_movimiento, p_fk_cuenta_usuario, p_fk_categoria);

    SET autocommit = 1;
    COMMIT;
    SELECT 'Reporte insertado correctamente' AS mensaje;
END; $$

```

Pruebas




```
179
180 Call insertar_reporte(200,'2023-07-10 00:00:00','Cita veterinario','Egreso',2,1);
181
```

Result Grid |  Filter Rows: | Export:  | Wrap Cell Content: 

mensaje

► Error al insertar el reporte


```
177
180 Call insertar_reporte(200,'2023-07-10','Cita veterinario','Egreso',1,1);
181 select * from cuenta usuario;
```

Result Grid |  Filter Rows: | Export:  | Wrap Cell Content: 

mensaje

► Reporte insertado correctamente

- Procedimiento que elimina un reporte con su ID

```
DELIMITER $$
CREATE PROCEDURE eliminar_reporte(p_id_reporte INTEGER)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        SELECT 'Error al eliminar el reporte' AS mensaje;
    END;

    SET autocommit = 0;
    START TRANSACTION;

    DELETE FROM reporte WHERE id_reporte = p_id_reporte;

    IF ROW_COUNT() = 0 THEN
        SELECT 'El reporte no existe' AS mensaje;
        ROLLBACK;
    ELSE
        SELECT 'Reporte eliminado correctamente' AS mensaje;
        COMMIT;
    END IF;
END; $$
```

Pruebas

```
208
209 call eliminar_reporte(1);
210
```

Result Grid | Filter Rows: | Export

mensaje
El reporte no existe

```
---
211 call eliminar_reporte(3);
212 select * from reporte;
213
```

Result Grid | Filter Rows: | Export

mensaje
Reporte eliminado correctamente

- Procedimiento verifica si una cuenta de banco se usa en más de una transacción

```
#inserción
DELIMITER $$
CREATE PROCEDURE cuenta_usada(num_cuenta_new INTEGER)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        SELECT 'Error al comprobar la cuenta de banco' AS mensaje;
    END;

    SET autocommit = 0;
    START TRANSACTION;

    #Validar si el número de cuenta ya ha sido registrado
    IF EXISTS(SELECT * FROM cuenta_usuario WHERE num_cuenta = num_cuenta_new) THEN
        SELECT 'Esa cuenta de banco ya ha sido registrada' AS mensaje;
        ROLLBACK;
    ELSE
        SELECT 'La cuenta de banco no ha sido utilizada' AS mensaje;
        COMMIT;
    END IF;

    SET autocommit = 1;
END $$
```


Pruebas

```
232
233  call cuenta_usada(123456);
234
235  #eliminación
236  SELECT * FROM cuenta_usuario;
```

Result Grid		Filter Rows:	Export
	mensaje		
▶	La cuenta de banco no ha sido utilizada		

```
236
237  call cuenta_usada(123456);
238
```

Result Grid		Filter Rows:	Export
	mensaje		
▶	Esa cuenta de banco ya ha sido registrada		

- Procedimiento para eliminar cuenta de usuario (banco)

```
#eliminación
SELECT * FROM cuenta_usuario;

DELIMITER $$
CREATE PROCEDURE eliminar_cuenta(id_cuenta_usuario_new INTEGER)
BEGIN
    SET autocommit = 0;
    START TRANSACTION;

    IF NOT EXISTS (SELECT * FROM cuenta_usuario WHERE id_cuenta_usuario = id_cuenta_usuario_new) THEN
        SELECT 'La cuenta de usuario no existe' AS mensaje;
        ROLLBACK;
    ELSE
        DELETE FROM cuenta_usuario WHERE id_cuenta_usuario = id_cuenta_usuario_new;
        SELECT 'Cuenta de usuario eliminada correctamente' AS mensaje;
        COMMIT;
    END IF;

    SET autocommit = 1;
END; $$
```

Pruebas

```
266
267 call eliminar_cuenta(1);
268
```

Result Grid | Filter Rows: | Export:

	mensaje
▶	Cuenta de usuario eliminada correctamente

- Procedimiento que inserta categoría

```
DELIMITER $$
CREATE PROCEDURE categoria_repetida(id_categoria_new INTEGER, nombre_new VARCHAR(50))
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        SELECT 'Error al comprobar la categoría' AS mensaje;
    END;

    SET autocommit = 0;
    START TRANSACTION;

    IF EXISTS(SELECT * FROM categoria WHERE nombre = nombre_new) THEN
        SELECT 'Esta categoría ya existe' AS mensaje;
        ROLLBACK;
    ELSE
        INSERT INTO categoria (id_categoria, nombre) VALUES (id_categoria_new, nombre_new);
        COMMIT;
        SELECT 'Categoría insertada correctamente' AS mensaje;
    END IF;

    SET autocommit = 1;
END; $$
```

Pruebas

```
280
281 call categoria_repetida (1, 'Gatos');
282
```

Result Grid | Filter Rows: | Export: | Wrap

	mensaje
▶	Categoría insertada correctamente

```

280
281  call categoria_repetida (1,'Gatos');
282

```

Result Grid		Filter Rows:	Export:	Wrap
	mensaje			
▶	Esta categoría ya existe			

- Procedimiento para la eliminación de categoría

```

#eliminación
#eliminar una categoría
#categoria
DELIMITER $$
CREATE PROCEDURE eliminar_categoria(id_categoria_new INTEGER)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        SELECT 'Error al eliminar la categoría' AS mensaje;
    END;

    SET autocommit = 0;
    START TRANSACTION;

    DELETE FROM categoria WHERE id_categoria = id_categoria_new;

    IF ROW_COUNT() = 0 THEN
        SELECT 'La categoría no existe' AS mensaje;
        ROLLBACK;
    ELSE
        SELECT 'Categoría eliminada correctamente' AS mensaje;
        COMMIT;
    END IF;

END; $$

```

Prueba

```

325
326  call eliminar_categoria(1);
327

```

Result Grid		Filter Rows:	Export:
	mensaje		
▶	Categoría eliminada correctamente		

```

325
326 call eliminar_categoria(1);
327

```

Result Grid | Filter Rows: | Export:

	mensaje
▶	La categoría no existe

- Procedimiento para cambiar el correo del usuario

```

#cambiar correo
DELIMITER $$
CREATE PROCEDURE actualizar_usuario_correo(id_usuario_new INTEGER, nuevo_correo_new VARCHAR(70))
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        SELECT 'Error al actualizar el correo del usuario' AS mensaje;
    END;

    SET autocommit = 0;
    START TRANSACTION;

    UPDATE usuario
    SET correo = nuevo_correo_new
    WHERE id_usuario = id_usuario_new;

    IF ROW_COUNT() = 0 THEN
        SELECT 'El usuario no existe' AS mensaje;
        ROLLBACK;
    ELSE
        SELECT 'Correo actualizado correctamente' AS mensaje;
        COMMIT;
    END IF;
END; $$

```

Pruebas

```

356
357 call actualizar_usuario_correo(2, 'SPC@utez.edu.mx');
358 select * from usuario;
359

```






Result Grid | Filter Rows: | Export: | Wrap Cell Content:

	mensaje
▶	Correo actualizado correctamente

```

357 call actualizar_usuario_correo(2,'SPC@utez.edu.mx');
358 select * from usuario;
359
360

```

Result Grid					
Filter Rows: <input type="text"/>					
Edit:   					
Export/Import:  					
	id_usuario	nombre	apellido	correo	contrasena
▶	2	John	Doe	SPC@utez.edu.mx	mypassword
*	NULL	NULL	NULL	NULL	NULL

- Procedimiento para consulta individual

```

DELIMITER $$
CREATE PROCEDURE usuario_unico_id (usuario_unico integer)
BEGIN
    IF EXISTS(select id_usuario from usuario
    where id_usuario=usuario_unico) THEN
        SELECT * FROM v_saldo_usuario where usuario_unico=id_usuario;
    ELSE
        SELECT "No se pudo encontrar el usuario" as mensaje;
    END IF;
END;$$

```

Pruebas

```

573
574 call usuario_unico_id(2);
575 select * from usuario;
576


```

Result Grid			
Filter Rows: <input type="text"/>			
Expo			
	id_usuario	Usuario	Saldo

```

573
574 call usuario_unico_id(3);
575 select * from usuario;
576

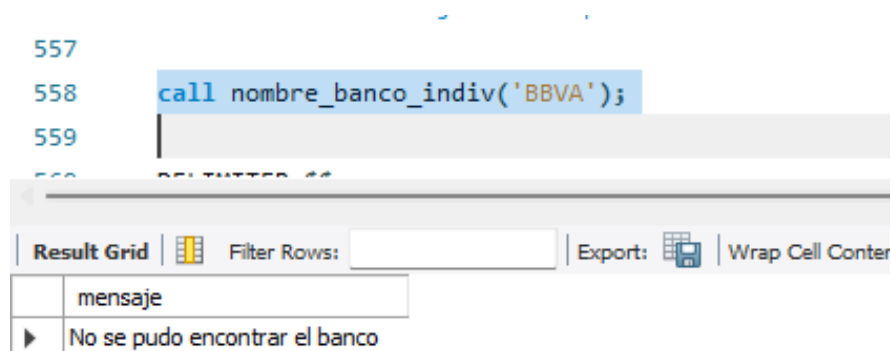
```

Result Grid	
Filter Rows: <input type="text"/>	
Export: 	
	mensaje
▶	No se pudo encontrar el usuario

- Procedimiento consulta el nombre individual del banco

```
DELIMITER $$
CREATE PROCEDURE nombre_banco_indiv (nombre_b varchar(20))
BEGIN
    IF EXISTS(select nombre_banco from cuenta_usuario
    where nombre_b=nombre_banco) THEN
        SELECT * FROM v_cuentas_bancarias_cuenta_usuario where nombre_b=Nombre_del_Banco;
    ELSE
        SELECT "No se pudo encontrar el banco" as mensaje;
    END IF;
END;$$
```

Pruebas





- Procedimiento para búsqueda de movimientos

```
DELIMITER $$
CREATE PROCEDURE busqueda_movimientos (fecha_busc integer)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        SELECT CONCAT(' El dato "', fecha_busc, '" no es valido, utilice un numero entre 01 a 12') as mensaje;
    END;
    IF EXISTS(SELECT fecha from reporte where EXTRACT(MONTH FROM fecha)=fecha_busc) THEN
        SELECT * FROM REPORTE WHERE EXTRACT(MONTH FROM fecha)=fecha_busc;
    ELSE
        SELECT "No hubo ningun movimiento en esa fecha" as mensaje;
    END IF;
END;$$
```

Pruebas

```
409
410 • call busqueda_movimientos('2023-07-10');
...
```

Result Grid |  Filter Rows: | Export:  | Wrap Cell Content

	mensaje
▶	No hubo ningun movimiento en esa fecha