19 JUNIO, 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 10
7 Disparadores 13
8 Agrupamiento 18
9 Ordenamiento 20
10 Desarrollo 22

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.

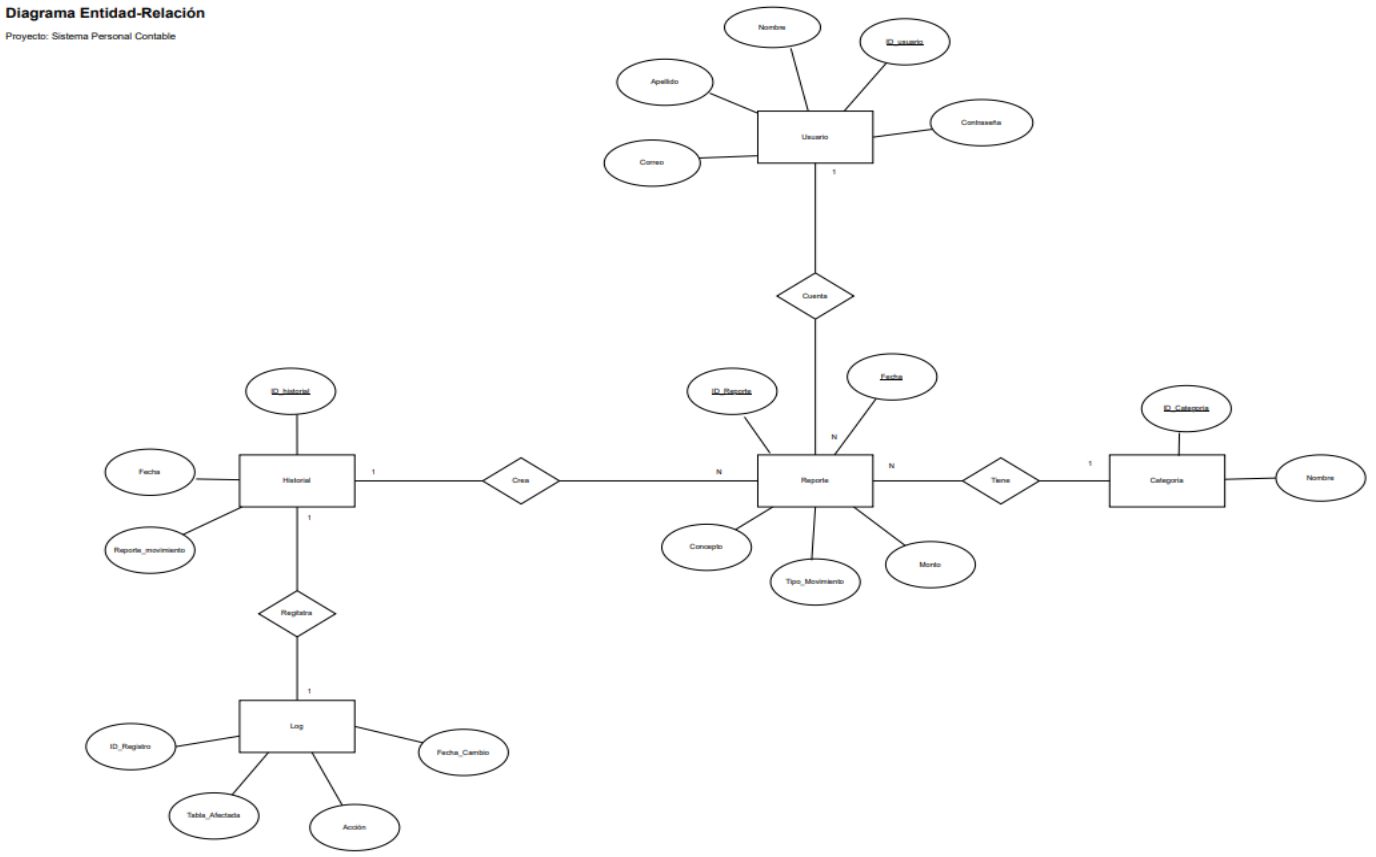
4 DISEÑO DE SOLUCIÓN

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. Reporte y sus atributos son id de reporte, monto, fecha, concepto, tipo movimiento.
3. Categoría y sus atributos son Id de categoría y nombre de categoría.
4. Historial y sus atributos son Id de Historial, fecha, Reporte de movimientos.
5. 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 termino de la siguiente manera:

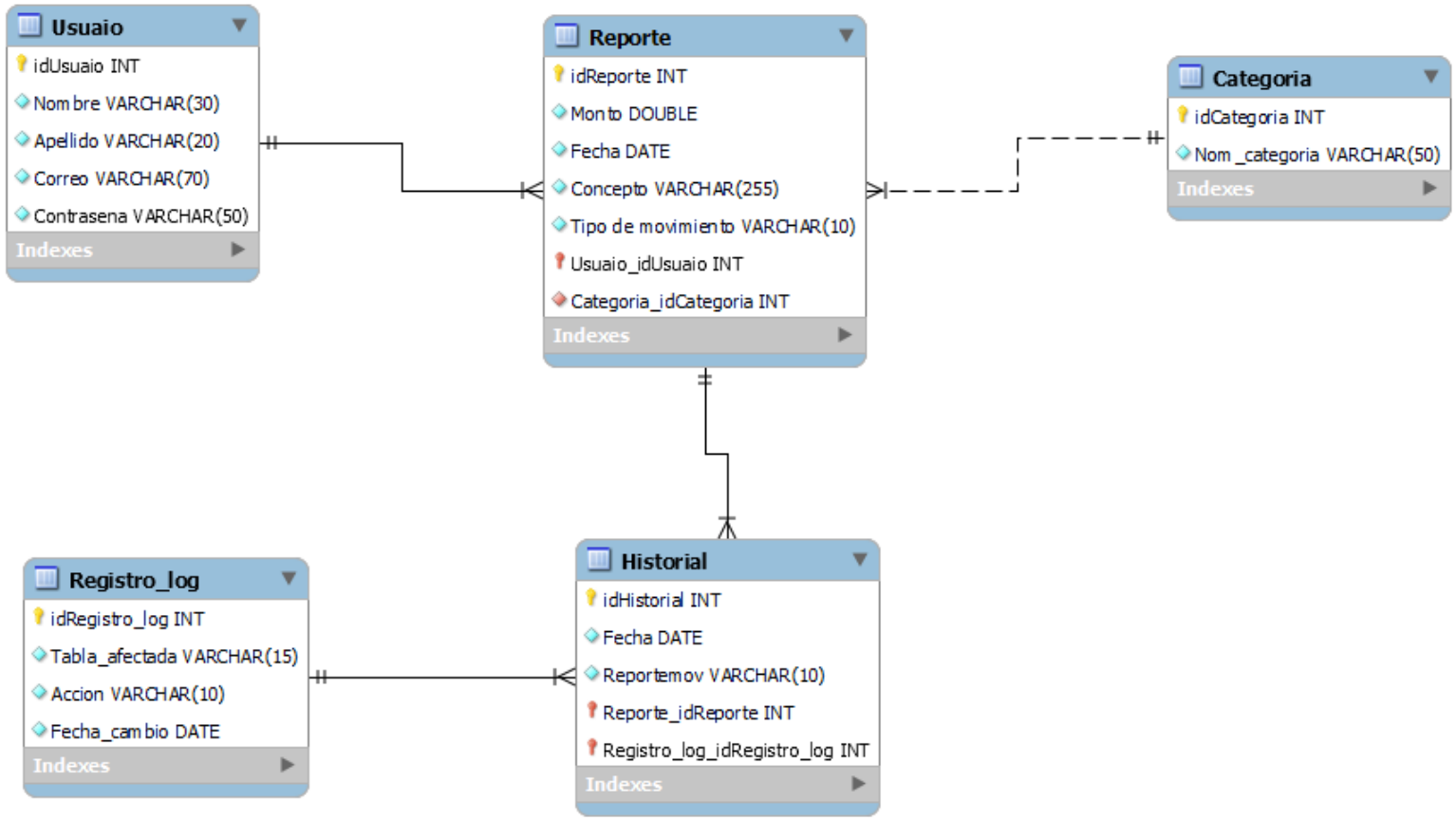


Modelo Relacional

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. 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 Categoria_IdCategoria.
3. Categoría y sus atributos son Id de categoría y nombre de categoría. Donde Id de reporte será la llave primaria.
4. 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.
5. ReporteLog y sus categorías son Id de registro, Tabla afectada, acción y Fecha. Donde Id de Regsitro será la llave primaria.

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



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

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;
```

- **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
registro_log.accion, #Tabla log
categoria.nombre AS Nombre_Categoria, #Tabla categroria
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;
```

- **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,nombre ASC;
```

- **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.

```
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 asc;
```

- **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 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 tipo_movimiento asc;
```

- **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 tipo_movimiento asc;
```

- **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_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 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_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 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 asc;
```

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

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 ordenamos por fecha de realización de ingreso y hace mas accesible dos 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 ordenamos 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 categroria
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

```
#Creacion de la base de datos
create database SPC;
use SPC;
#Creacion de las tablas
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 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_usuario integer not null,
    fk_categoria integer,
    foreign key (fk_usuario) references usuario (id_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,
    fk_registro_log integer not null,
    foreign key (fk_reporte) references reporte (id_reporte),
    foreign key (fk_registro_log) references registro_log (id_registro_log)
);
```

#Creacion de vistas

```
#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;
#Vista que junta la id y nombre completo del usuario y saldo total actual
```

```
#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 categroria
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;
#Tabla que junta el nombre completo del usuario, el movimiento y acción que va a hacer,
#el nombre de la categoría
#a la que está haciendo la acción y la fecha en la que lo realizó

# Fin tabla usuario

#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;
#Vista que muestra los nombres de las categorías y la fecha escrita en la tabla pedidos

#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;
```

```
#Lo mismo que la de arriba, pero no muestra nombres, solo el número de categorías creadas

# Fin Tabla categoria

#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;
#vista que junta la acción y fecha de la acción de la tabla de log

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;
#Tabla que junta todas las acciones

########## Fin tabla registro_log

##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;
#Vista que muestra el monto, y la fecha del reporte de ingresos

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;
```

```
#Vista que muestra el monto, y la fecha del reporte de egresos

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;
#Vista que muestra la suma de todos los ingresos, dividido por categorías

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;
#Vista que muestra la suma de todos los egresos, dividido por categorías

#Fin tablas reporte

#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;
#Vista que muestra el movimiento y la fecha del movimiento de historial

########## Fin Tabla historial
```

#Indices

#Tablas usuario

#Indice que sirve para facilitar la búsqueda de la id de los usuarios

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

#Indice que 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);
```

#Indice que sirve para facilitar la búsqueda de los datos personales de un #usuario junto con su id

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

#Tabla categoría

#Indice que sirve para facilitar la búsqueda de la id de la categoría

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

#Indice que sirve para facilitar la búsqueda del nombre de la categoría junto con su id

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

#tabla registro_log

#Indice que sirve para facilitar la búsqueda de la id de los registros

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

#Indice que sirve para facilitar la búsqueda de tosa la información acerca de los cambios realizados en las tablas

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

#Indice que 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);

#tabla reporte

#Indice que sirve para facilitar la búsqueda de la id del reporte

create index id_reporte on reporte (id_reporte);

#Indice que 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);

#Indice que sirve para facilitar la búsqueda de los movimientos realizados por #los usuarios

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

#Tabla historial

#Indice que sirve para facilitar la búsqueda de la id del historial

create index id_historial on historial (id_historial);

#Indice que 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);

#Indice que sirve para facilitar la búsqueda de los reportes y registros qué forman parte del historial

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

#Disparadores

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

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

#Delete (No borra las categorías que se están usando)
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; $$

drop trigger categoria_usada;

#After (Todos los after solo dan registro a la tabla de cambios de sistema (registro_log))

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

#update
DELIMITER $$
create trigger update_log after update on categoria
for each row
begin
```

```
insert into registro_log values (0,'Categoria','Update',curdate());
end;$$

#Delete
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;

#Historial
#Before

#Insert (No deja poner otra fecha que no sea la actual)
DELIMITER $$
create trigger verificar_transaccion before insert on historial
for each row
begin
if new.fecha<>curdate() then
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Fecha erronea';
end if;
end;$$

drop trigger verificar_transaccion;

#Update (No deja modificar las llaves foreanas)
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; $$

drop trigger Confirmar_forana;

update historial set fk_reporte=1 where fk_reporte=2;

#Delete (No permite borrar la fila al menos que se desocupe)
DELIMITER $$
create trigger fkrepote_uso before delete on historial
for each row
begin
declare epotek integer;
select count(*) into epotek from reporte where id_reporte=old.fk_reporte;
if epotek>0 then
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Este repoe se esta usando';
end if;
end;$$
drop trigger fkrepote_uso;

delete from historial where fk_reporte=2;


#After
```

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

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

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

#Registro_log
#Before
#Insert (No permite que se ponga el nombre de otra tabla que no se las establecidas)
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; $$

drop trigger tabla_verificacion;

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

drop trigger valores_seguimiento;
select *from registro_log;
 update registro_log set tabla_afectada = 'Propaganda' where id_registro_log=2;

 #Delete (No permite borrar cambios del sistema que no sean mayores a un mes de publicacion)
 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 antiguedad';

 end if;

 end;$$


 delete from registro_log where id_registro_log=2;
```