**Informe MySQL2**

**Paso a Paso**

**para el desarrollo de la actividad**

**Daza Daza Andres Santiago**

**P1**

**CampusLands**

**2024**

1. **Análisis:** Se ve que lo que se solicita son:

5 tablas

* sucursales
* cliente
* empleado
* vehículos
* alquileres

1. Modelo conceptual: Se desarrolló el modelo conceptual mediante dibujos de google: <https://docs.google.com/drawings/d/1KrIpS9VhC5nUV7rXurmWhs3lNo0HQY3mQCjfvSmpiKA/edit?usp=sharing>
2. Modelo Lógico: Se desarrolló el diagrama para el modelo Lógico mediante dibujos de google: <https://docs.google.com/drawings/d/1XeKbWXPeej9sMFGR55KWRUv2ax_WI7JVpGb7FSPTGMY/edit?usp=sharing>

* cliente {

id int primary key

nombre1 varchar(25)

nombre2 varchar(25)

apellido1 varchar(25)

apellido2 varchar(25)

email varchar(25)

cedula int(15)

celular varchar(25)

ciudad\_resistencia varchar(25)

direccion varchar(25)

}

* sucursales {

id int primary key

email varchar(25)

celular varchar(25)

telefono varchar(25)

ciudad varchar(25)

direccion varchar(25)

}

* empleado {

id int primary key

nombre1 varchar(25)

nombre2 varchar(25)

apellido1 varchar(25)

apellido2 varchar(25)

ciudad\_residencia varchar(25)

celular varchar(25)

direccion varchar(25)

email varchar(25)

cedula int(15)

id\_sucursal int foreign key

}

* vehiculo {

id int primary key

color varchar(25)

motor varchar(25)

sunroof varchar(25)

capacidad varchar(25)

puertas int(3)

modelo varchar(25)

referencia varchar(25)

placa varchar(25)

tipo\_vehiculo varchar(25)

}

* alquileres {

id int primary key

id\_vehiculo int foreign key

id\_empleado int foreign key

id\_sucursal\_salida int foreign key

id\_sucursal\_llegada int foreign key

id\_cliente int foreign key

fecha\_salida date

fecha\_llegada date

fecha\_esperada\_llegada date

valor\_alquiler\_semana int(25)

valor\_alquiler\_dia int(25)

porcentaje\_descuento decimal(2,2)

valor\_cotizado int(40)

valor\_pagado int(40)

}

1. Normalización: se desarrolló la normalización mediante dibujos de google:  
   <https://docs.google.com/drawings/d/1plaLlebTUZknFSs70WY19-vAXL2Koxspv4mFjjHfWhc/edit?usp=sharing>
2. modelo físico: el modelo físico se desarrolló en MySQL, utilizando la creación de tablas y un mínimo de 100 inserts por tabla

Cree un total de 5 tablas, las cuales serán parte de la base de datos “AutoRental”, estas tablas tienen sus propios parámetros o valores, los cuales las hacen únicas

create table cliente (

id int primary key,

nombre1 varchar(25) not null,

nombre2 varchar(25),

apellido1 varchar(25) not null,

apellido2 varchar(25),

email varchar(50) not null,

cedula varchar(25) not null,

celular varchar(25) not null,

ciudad\_residencia varchar(25) not null,

direccion varchar(25) not null

);

create table sucursales(

id int primary key,

email varchar(25) not null,

celular varchar(25) not null,

telefono varchar(25) not null,

ciudad varchar(25) not null,

direccion varchar(25) not null

);

create table empleado(

id int primary key,

nombre1 varchar(25) not null,

nombre2 varchar(25),

apellido1 varchar(25) not null,

apellido2 varchar(25),

email varchar(50) not null,

cedula varchar(25) not null,

celular varchar(25) not null,

ciudad\_residencia varchar(25) not null,

direccion varchar(25) not null,

id\_sucursal int,

foreign key (id\_sucursal) references sucursales(id)

);

create table vehiculo(

id int primary key,

color varchar(25) not null,

motor varchar(25) not null,

sunroof varchar(25) not null,

capacidad varchar(25) not null,

puertas int(3) not null,

modelo varchar(25) not null,

referencia varchar(25) not null,

placa varchar(25) not null,

tipo\_vehiculo varchar(25) not null

);

create table alquileres(

id int primary key,

fecha\_salida date not null,

fecha\_llegada date not null,

fecha\_esperada\_llegada date not null,

valor\_alquiler\_semana int(25) not null,

valor\_alquiler\_dia int(25) not null,

porcentaje\_descuento decimal(2,2) not null,

valor\_cotizado int(40) not null,

valor\_pagado int(40) not null,

id\_vehiculo int,

id\_empleado int,

id\_cliente int,

id\_sucursal\_salida int,

id\_sucursal\_llegada int,

foreign key (id\_vehiculo) references vehiculo(id),

foreign key (id\_empleado) references empleado(id),

foreign key (id\_cliente) references cliente(id),

foreign key (id\_sucursal\_salida) references sucursales(id),

foreign key (id\_sucursal\_llegada) references sucursales(id)

);

1. Inserts: Se realizaron los inserts independientes para cada tabla, con un mínimo de 100 inserciones.
2. se desarrollaron procedimientos y funciones para cada usuario:

**Procedimiento para registrar un cliente (**registrar\_cliente**):**

El primer procedimiento creado en el archivo MySQL\_Dia4\_Pro\_Fun.sql, sirve para insertar un cliente a la tabla de clientes, este procedimiento fue creado para permitirle al futuro usuario “empleado” el registro de nuevos clientes, este procedimiento recibe como parámetros, todos los datos del cliente que se desea registrar.

**delimiter //**

create procedure registrar\_cliente(cliente\_id int, cliente\_nombre1 varchar(25), cliente\_nombre2 varchar(25), cliente\_apellido1 varchar(25), cliente\_apellido2 varchar(25), cliente\_email varchar(50), cliente\_cedula varchar(25), cliente\_celular varchar(25), cliente\_ciudad\_residencia varchar(25), cliente\_direccion varchar(25))

begin

insert into cliente(id, nombre1, nombre2, apellido1, apellido2, email, cedula, celular, ciudad\_residencia, direccion)

values (cliente\_id, cliente\_nombre1, cliente\_nombre2, cliente\_apellido1, cliente\_apellido2, cliente\_email, cliente\_cedula, cliente\_celular, cliente\_ciudad\_residencia, cliente\_direccion);

end//

**delimiter** ;

Ejemplo de uso:

CALL registrar\_cliente('101', 'Alberto', 'Nicol', 'instantaneo', 'Apesta', 'AlbertEinstein@gmail.com', '1234567890', '123-123-123', 'Bucaramanga', 'Rincon de Giron 100#123-123');

**Procedimiento para actualizar información de un cliente (**actualizar\_cliente**):**

Este procedimiento, sirve para actualizar la información de un cliente ya registrado en la tabla clientes, este procedimiento fue creado para el futuro usuario “empleado”, de manera que este pueda actualizar la información, ingresando como primer parámetro el id del cliente que desea actualizar, para después añadir los demás datos.

**delimiter //**

create procedure actualizar\_cliente(id\_cliente int, n\_id\_cliente int, n\_cliente\_nombre1 varchar(25), n\_cliente\_nombre2 varchar(25), n\_cliente\_apellido1 varchar(25), n\_cliente\_apellido2 varchar(25), n\_cliente\_email varchar(50), n\_cliente\_cedula varchar(25), n\_cliente\_celular varchar(25), n\_cliente\_ciudad\_residencia varchar(25), n\_cliente\_direccion varchar(25))

begin

update cliente set id = n\_id\_cliente, nombre1 = n\_cliente\_nombre1, nombre2 = n\_cliente\_nombre2, apellido1 = n\_cliente\_apellido1, apellido2 = n\_cliente\_apellido2, email = n\_cliente\_email, cedula = n\_cliente\_cedula, celular = n\_cliente\_celular, ciudad\_residencia = n\_cliente\_ciudad\_residencia, direccion = n\_cliente\_direccion

where id = id\_cliente;

end//

**delimiter ;**

ejemplo de uso:

call actualizar\_cliente(12345, 102, 'Juancho','Felipe','Casillas','Vida','JuanchoVida@gmail.com','6333333-3', '633-33-33', 'Piedecuesta', 'barrio 232#232-232');

**Procedimiento para registrar empleados(**registrar\_empleado**):**

este procedimiento, sirve para permitirle al futuro usuario “Director”, añadir nuevos empleados a la tabla de empleados, este usuario está pensado para que sea algo así como un “superior” de los demás empleados, por ende tiene los permisos para agregar empleados nuevos, este procedimiento, recibe como parámetros, toda la información del nuevo empleado, incluida la sucursal en la que estará asignado, se pensó, que se podría llegar a equivocar al momento de asignar la sucursal, así que se maneja el caso de que se coloque un id de sucursal inexistente, mostrando un mensaje de error.

**delimiter //**

create procedure registrar\_empleado(empleado\_id int, empleado\_nombre1 varchar(25), empleado\_nombre2 varchar(25), empleado\_apellido1 varchar(25), empleado\_apellido2 varchar(25), empleado\_email varchar(25), empleado\_cedula varchar(25), empleado\_celular varchar(25), empleado\_ciudad\_residencia varchar(25), empleado\_direccion varchar(25), empleado\_sucursal int)

begin

declare n\_sucursales int;

select count(\*) into n\_sucursales from sucursales where sucursales.id = empleado\_sucursal;

if n\_sucursales > 0 then

insert into empleado(id, nombre1, nombre2, apellido1, apellido2, email, cedula, celular, ciudad\_residencia, direccion, id\_sucursal) values(empleado\_id, empleado\_nombre1, empleado\_nombre2, empleado\_apellido1, empleado\_apellido2, empleado\_email, empleado\_cedula, empleado\_celular, empleado\_ciudad\_residencia, empleado\_direccion, empleado\_sucursal);

else

select 'error: la sucursal no existe bro...' as mensaje;

end if;

end//

**delimiter ;**

ejemplo de uso:

call registrar\_empleado(102, 'Cristiano', 'nose', 'Ronaldo', 'nose', 'elBichoSiuu@gmail.com', '7777777777', '123432177', 'niidea', 'sisas', 100);

**Procedimiento para actualizar información de un empleado(**Actualizar\_empleado**):**

Este procedimiento sirve para actualizar la información de un empleado ya existente, recibiendo como primer parámetro el id del empleado que se busca actualizar, de manera que los demás parámetros, son el resto de la información del empleado en cuestión, este procedimiento está pensado para ser utilizado por el futuro usuario “director”.

**delimiter //**

create procedure actualizar\_empleado(empleado\_id int, n\_empleado\_id int, empleado\_nombre1 varchar(25), empleado\_nombre2 varchar(25), empleado\_apellido1 varchar(25), empleado\_apellido2 varchar(25), empleado\_email varchar(25), empleado\_cedula varchar(25), empleado\_celular varchar(25), empleado\_ciudad\_residencia varchar(25), empleado\_direccion varchar(25), empleado\_sucursal int)

begin

update empleado set id = n\_empleado\_id, nombre1 = empleado\_nombre1, nombre2 = empleado\_nombre2, apellido1 = empleado\_apellido1, apellido2 = empleado\_apellido2, email = empleado\_email, cedula = empleado\_cedula, celular = empleado\_celular, ciudad\_residencia = empleado\_ciudad\_residencia, direccion = empleado\_direccion, id\_sucursal = empleado\_sucursal

where id = empleado\_id;

end//

**delimiter ;**

ejemplo de uso:

call actualizar\_empleado(103, 102, 'Cristiano', null,'Ronaldo', 'siuu', 'elBichoSiuu@gmail.com', '7777777777', '123432177', 'niidea', 'sisas', 99);

**Procedimiento para registrar vehículos(**registrar\_vehiculo**):**

Este procedimiento, sirve para registrar un nuevo vehículo en la tabla vehículo, recibe como parámetros la información del nuevo vehículo y está pensado para ser utilizado por el futuro usuario “empleado”, además, este procedimiento, tiene la funcionalidad para evitar que se registren vehículos los cuales su tipo de vehículo no haya sido ingresado nunca antes en la tabla de vehículos, y mostrará un mensaje de error en el caso de que se trate de ingresar un vehículo con el tipo de vehículo incorrecto.

**delimiter //**

create procedure registrar\_vehiculo(vehiculo\_id int, vehiculo\_color varchar(25), vehiculo\_motor varchar(25), vehiculo\_sunroof varchar(25), vehiculo\_capacidad varchar(25), vehiculo\_puertas int, vehiculo\_modelo varchar(25), vehiculo\_referencia varchar(25), vehiculo\_placa varchar(25), vehiculo\_tipo\_vehiculo varchar(25))

begin

declare tipo\_vehiculo\_e int;

select count(\*) into tipo\_vehiculo\_e from vehiculo where tipo\_vehiculo = vehiculo\_tipo\_vehiculo;

if tipo\_vehiculo\_e > 0 then

insert into vehiculo(id, color, motor, sunroof, capacidad, puertas, modelo, referencia, placa, tipo\_vehiculo)

values(vehiculo\_id, vehiculo\_color, vehiculo\_motor, vehiculo\_sunroof, vehiculo\_capacidad, vehiculo\_puertas, vehiculo\_modelo, vehiculo\_referencia, vehiculo\_placa, vehiculo\_tipo\_vehiculo);

select 'Vehículo registrado exitosamente.' as mensaje;

else

select 'Error: El tipo de vehículo especificado no es válido.' as mensaje;

end if;

end//

**delimiter ;**

ejemplo de uso:

call registrar\_vehiculo(101, 'Rojo', 'Gasolina', 'No', '5 personas', 4, 'Sedán', '2023', 'ABC123', 'Automóvil');

**Procedimiento para actualizar la información de un vehículo(**actualizar\_vehiculo**):**

Este procedimiento, sirve para actualizar la información de los vehículos ya registrados en la tabla vehículos, de manera que recibe como primer parámetro el id del vehículo que se busca actualizar, este procedimiento está pensado para el futuro usuario “empleado”.

**delimiter //**

create procedure actualizar\_vehiculo( vehiculo\_id int, nuevo\_vehiculo\_id int, vehiculo\_color varchar(25), vehiculo\_motor varchar(25), vehiculo\_sunroof varchar(25), vehiculo\_capacidad varchar(25), vehiculo\_puertas int, vehiculo\_modelo varchar(25), vehiculo\_referencia varchar(25), vehiculo\_placa varchar(25), vehiculo\_tipo\_vehiculo varchar(25))

begin

update vehiculo set id = nuevo\_vehiculo\_id, color = vehiculo\_color, motor = vehiculo\_motor, sunroof = vehiculo\_sunroof, capacidad = vehiculo\_capacidad, puertas = vehiculo\_puertas, modelo = vehiculo\_modelo, referencia = vehiculo\_referencia, placa = vehiculo\_placa, tipo\_vehiculo = vehiculo\_tipo\_vehiculo

where id = vehiculo\_id;

select 'Vehículo actualizado correctamente.' as mensaje;

end//

**delimiter ;**

ejemplo de uso:

call actualizar\_vehiculo(101, 102, 'Rojo', 'Gasolina', 'No', '5 personas', 4, 'Sedán', '2023', 'ABC123', 'Automóvil');

**Procedimiento para insertar sucursales(**insertar\_sucursal**):**

Este procedimiento, sirve para permitir al futuro usuario “empleado”, insertar nuevas sucursales en la tabla de sucursales, este procedimiento, recibe como parámetros toda la información de la nueva sucursal.

**delimiter //**

create procedure insertar\_sucursal(sucursal\_id int, sucursal\_email varchar(25), sucursal\_celular varchar(25), sucursal\_telefono varchar(25), sucursal\_ciudad varchar(25), sucursal\_direccion varchar(25))

begin

insert into sucursales(id, email, celular, telefono, ciudad, direccion) values(sucursal\_id, sucursal\_email, sucursal\_celular, sucursal\_telefono, sucursal\_ciudad, sucursal\_direccion);

end//

**delimiter ;**

**Procedimiento para actualizar la información de sucursales ya registradas(**actualizar\_sucursales**):**

Este procedimiento, sirve para permitir al futuro usuario “empleado”, actualizar la información de las sucursales ya existentes, ingresando como primer parámetro el id de la sucursal en cuestión que se desea actualizar.

**delimiter //**

create procedure actualizar\_sucursal( sucursal\_id int, n\_sucursal\_id int, sucursal\_email varchar(25), sucursal\_celular varchar(25), sucursal\_telefono varchar(25), sucursal\_ciudad varchar(25), sucursal\_direccion varchar(25))

begin

update sucursales set id = n\_sucursal\_id, email = sucursal\_email, celular = sucursal\_celular, telefono = sucursal\_telefono, ciudad = sucursal\_ciudad, direccion = sucursal\_direccion

where id = sucursal\_id;

end//

**delimiter ;**

**Procedimiento para consultar disponibilidad de vehículos para alquiler por tipo de vehiculo, rango de precios y fechas de disponibilidad(**consultar\_vehiculos**):**

Este procedimiento sirve para consultar los vehículos que están disponibles para alquiler, teniendo en cuenta su tipo de vehículo, el rango de los precios y las fechas de disponibilidad. El procedimiento recibe como parámetros, El tipo de vehículo, el rango de las fechas que se solicita el vehículo, el rango de precio minimo y maximo por dia y el rango de precio minimo y maximo por semana.

Al recibir estos parámetros, el procedimiento lo primero que busca es seleccionar los datos de los vehículos, para luego hacer uso de un left join con la tabla alquileres para ingresar el rango de precios y verificar que el vehículo no esté siendo ya utilizado en algún otro alquiler, luego verifica y filtra los resultados, por el tipo de vehículo solicitado, y por las fechas solicitadas.

**delimiter //**

create procedure consultar\_vehiculos(con\_tipo\_vehiculo varchar(25), con\_fecha\_inicio date, con\_fecha\_fin date, con\_precio\_min\_dia int, con\_precio\_max\_dia int, con\_precio\_min\_semana int, con\_precio\_max\_semana int)

begin

select vehiculo.id as VehiculoID, vehiculo.color, vehiculo.motor, vehiculo.sunroof, vehiculo.capacidad, vehiculo.puertas, vehiculo.modelo, vehiculo.referencia, vehiculo.placa, vehiculo.tipo\_vehiculo, alquileres.valor\_alquiler\_semana, alquileres.valor\_alquiler\_dia

from vehiculo

left join alquileres on vehiculo.id = alquileres.id\_vehiculo

where vehiculo.tipo\_vehiculo = con\_tipo\_vehiculo and (alquileres.id is null or (alquileres.fecha\_salida > con\_fecha\_fin or alquileres.fecha\_llegada < con\_fecha\_inicio)) and ((ifnull(alquileres.valor\_alquiler\_dia, 0) between con\_precio\_min\_dia and con\_precio\_max\_dia) or (ifnull(alquileres.valor\_alquiler\_semana, 0) between con\_precio\_min\_semana and con\_precio\_max\_semana))

order by vehiculo.id;

end//

**delimiter ;**

ejemplo de uso:

call consultar\_vehiculos('Automóvil', '2024-10-01', '2024-10-10', 0, 0, 500000, 550000);

**Procedimiento para hacer alquileres(**registrar\_alquileres**):**

Este procedimiento lo que busca, es permitirle al futuro usuario “cliente”, hacer un alquiler, recibiendo todos los datos del alquiler como parámetro, luego, verifica la existencia de los vehículos, de los empleados, del cliente, de la sucursal de salida y de la sucursal de llegada, de manera que en caso de que alguna de estas no esté presente, devolverá un mensaje de error, en caso de que si existan, realizará la inserción con total normalidad.

**delimiter //**

create procedure registrar\_alquiler(alquiler\_id int, fecha\_salida date, fecha\_llegada date, fecha\_esperada\_llegada date, valor\_alquiler\_semana int, valor\_alquiler\_dia int, porcentaje\_descuento decimal(5,2), valor\_cotizado int, valor\_pagado int, vehiculo\_id int, empleado\_id int, cliente\_id int, sucursal\_salida\_id int, sucursal\_llegada\_id int)

begin

declare existencia\_vehiculo int;

declare existencia\_empleado int;

declare existencia\_cliente int;

declare existencia\_sucursal\_salida int;

declare existencia\_sucursal\_llegada int;

select count(\*) into existencia\_vehiculo from vehiculo where id = vehiculo\_id;

select count(\*) into existencia\_empleado from empleado where id = empleado\_id;

select count(\*) into existencia\_cliente from cliente where id = cliente\_id;

select count(\*) into existencia\_sucursal\_salida from sucursales where id = sucursal\_salida\_id;

select count(\*) into existencia\_sucursal\_llegada from sucursales where id = sucursal\_llegada\_id;

if existencia\_vehiculo > 0 and existencia\_empleado > 0 and existencia\_cliente > 0 and existencia\_sucursal\_salida > 0 and existencia\_sucursal\_llegada > 0 then

insert into alquileres(id, fecha\_salida, fecha\_llegada, fecha\_esperada\_llegada, valor\_alquiler\_semana, valor\_alquiler\_dia, porcentaje\_descuento, valor\_cotizado, valor\_pagado, id\_vehiculo, id\_empleado, id\_cliente, id\_sucursal\_salida, id\_sucursal\_llegada)

values(alquiler\_id, fecha\_salida, fecha\_llegada, fecha\_esperada\_llegada, valor\_alquiler\_semana, valor\_alquiler\_dia, porcentaje\_descuento, valor\_cotizado, valor\_pagado, vehiculo\_id, empleado\_id, cliente\_id, sucursal\_salida\_id, sucursal\_llegada\_id);

else

select 'Error: Verifique la existencia de los elementos requeridos (vehículo, empleado, cliente, sucursal de salida y llegada).' as mensaje;

end if;

end//

**delimiter ;**

**Función para Consultar el historial de alquileres por cliente(**histrorial\_alqquileres**):**

Esta función, tiene como objetivo mostrarle al futuro usuario “ciente”, todos los alquileres que ha realizado los clientes con x id.

lo que hace, es recibir el id del cliente deseado como parámetro, para mostrar todos los resultados asociados a dicho cliente.

**delimiter //**

create function historial\_alquileres(cliente\_id INT)

returns text deterministic

begin

declare resultado TEXT;

select

group\_concat(

concat('alquiler id: ', alq.id,

', fecha salida: ', alq.fecha\_salida,

', fecha llegada: ', alq.fecha\_llegada,

', vehículo: ', veh.placa,

', empleado: ', emp.nombre1,

', valor cotizado: ', alq.valor\_cotizado

) separator '\n') into resultado

from alquileres alq

join vehiculo veh on alq.id\_vehiculo = veh.id

join empleado emp on alq.id\_empleado = emp.id

where alq.id\_cliente = cliente\_id;

if resultado is null then

set resultado = 'no se encontraron alquileres para este cliente.';

end if;

return resultado;

end //

**delimiter ;**

ejemplo de uso:

select historial\_alquileres(1) as Historial;

1. Creación de usuarios y asignación de permisos:

Se crearon un total de 3 usuarios, “cliente”, “empleado”, “director”; cada uno con sus permisos exclusivos sobre las funciones, correspondientes, estos permisos permiten mantener una seguridad aceptable a la base de datos, ya que ninguno de ellos tiene permisos para eliminar ningún dato de la base de datos.

**Apartado para la creación de usuarios:**create user 'empleado'@'localhost' identified by 'contraseña\_empleado';

create user 'director'@'localhost' identified by 'contraseña\_director';

create user 'cliente'@'localhost' identified by 'contraseña\_cliente';

**Apartado para la asignación de permisos:**

grant execute on function AutoRental.historial\_alquileres to 'cliente'@'localhost';

grant execute on procedure AutoRental.registrar\_alquiler to 'cliente'@'localhost';

grant execute on procedure AutoRental.consultar\_vehiculos to 'cliente'@'localhost';

grant execute on procedure AutoRental.insertar\_sucursal to 'empleado'@'localhost';

grant execute on procedure AutoRental.actualizar\_sucursal to 'empleado'@'localhost';

grant execute on procedure AutoRental.registrar\_vehiculo to 'empleado'@'localhost';

grant execute on procedure AutoRental.actualizar\_vehiculo to 'empleado'@'localhost';

grant execute on procedure AutoRental.registrar\_cliente to 'empleado'@'localhost';

grant execute on procedure AutoRental.actualizar\_cliente to 'empleado'@'localhost';

grant execute on procedure AutoRental.registrar\_empleado to 'director'@'localhost';

grant execute on procedure AutoRental.actualizar\_empleado to 'director'@'localhost';

**Apartado para verificar los privilegios de cada usuario:**SHOW GRANTS FOR 'cliente'@'localhost';

SHOW GRANTS FOR 'empleado'@'localhost';

SHOW GRANTS FOR 'director'@'localhost';

1. creación de triggers y eventos:

Se crearon un trigger y un evento, after\_cliente\_insert y daily\_city\_backup

**after\_cliente\_insert**: Este trigger, está creado con el objetivo de que en el momento en el que se inserte un cliente nuevo en la tabla cliente, se inserta ese nuevo cliente a la tabla inserciones\_nuevas\_clientes, añadiendo el tipo de acción y la fecha y hora en la que se realizó la inserción.

delimiter //

create trigger after\_cliente\_insert

after insert on cliente

for each row

begin

insert into inserciones\_nuevas\_clientes(id, nombre1, nombre2, apellido1, apellido2, email, cedula, celular, ciudad\_residencia, direccion, action) values(new.id, new.nombre1, new.nombre2, new.apellido1, new.apellido2, new.email, new.cedula, new.celular, new.ciudad\_residencia, new.direccion, "INSERT");

end//

delimiter ;

Se espera que en el momento en el que se realice un insert en la tabla cliente, se accione inmediatamente este trigger.

**daily\_city\_backup:** Este evento busca crear una backup diaria de los clientes o de los datos registrados en la tabla de cliente, en la tabla backup\_clientes. de manera que diariamente, se realizará automáticamente esta back up.

delimiter //

create event daily\_city\_backup

on schedule every 1 day

do

begin

truncate table backup\_clientes;

insert into backup\_clientes(id, nombre1, nombre2, apellido1, apellido2, email, cedula, celular, ciudad\_residencia, direccion) select id, nombre1, nombre2, apellido1, apellido2, email, cedula, celular, ciudad\_residencia, direccion

from cliente;

end//

delimiter ;

Se espera que diariamente se desarrollé la back up, eliminando la backup anterior, de manera que limite el consumo y este no sea tan extenso.