

# BANCO KINGDOM

Proyecto Base de Datos 5to HITO



## Integrantes:

- Hugo Brandon Chambi Quispe
- Karla Belen Diaz Flores
- Rivaldo Kari Laura
- Melanie Ingrid Villca Copa
- Iudwing Antoni Vargas Ibarra

## Grupo:

- The clup

## Objetivo General

Yo y mi equipo escogimos este tema para la simulación de un banco llamado Banco kingdom, mi persona y mis compañeros hemos trabajado en una base de datos de un banco para ver los aspectos administrativos y el orden de cada uno de los clientes.

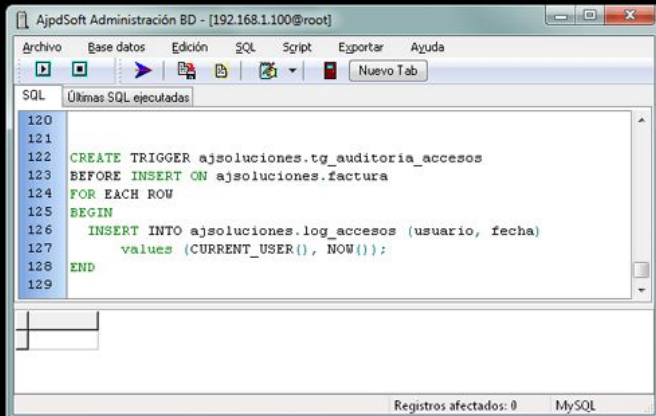
A Través del presente al elaborar este proyecto acudimos a los triggers, vistas y funciones para la resolución de problemas y de esta forma facilitar el trabajo en un banco.



# Objeto Específico

Como objetivo específico en todo el trabajo son los siguientes:

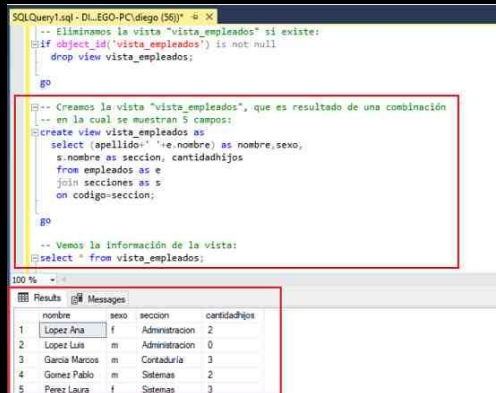
- Es el promover el ayudar al banco con los triggers para la asociación de las tablas con distintos ejercicios que ayuden de una forma en el proyecto.
- En cuanto a las funciones más nos servirán para hacer ejercicios lógicos o de selección dentro del programa y también en cuanto búsquedas de las tablas.
- Y por último las vistas que nos servirá para el control de los clientes dentro de la base de datos.



AjpdSoft Administración BD - [192.168.1.100@root]

```
120
121
122 CREATE TRIGGER ajsoluciones.tg_auditoria_accesos
123 BEFORE INSERT ON ajsoluciones.factura
124 FOR EACH ROW
125 BEGIN
126     INSERT INTO ajsoluciones.log_accesos (usuario, fecha)
127     values (CURRENT_USER(), NOW());
128 END
129
```

Registros afectados: 0 MySQL



```
-- Eliminamos la vista "vista_empleados" si existe:
-- If object_id('vista_empleados') is not null
drop view vista_empleados;

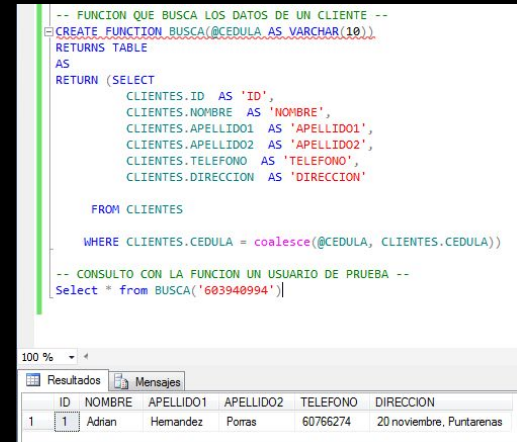
go

-- Creamos la vista "vista_empleados", que es resultado de una combinación
-- en la cual se muestran 5 campos:
create view vista_empleados as
select (apellido||' '||e.nombre) as nombre,sexo,
s.nombre as seccion, cantidadhijos
from empleados as e
join secciones as s
on codigo=seccion;

go

-- Vemos la información de la vista:
select * from vista_empleados;
```

nombre	sexo	seccion	cantidadhijos
Lopez Ana	f	Administración	2
Lopez Luis	m	Administración	0
Garcia Marcos	m	Contaduría	3
Gomez Pablo	m	Sistemas	2
Perez Laura	f	Sistemas	3



```
-- FUNCION QUE BUSCA LOS DATOS DE UN CLIENTE --
CREATE FUNCTION BUSCA(@CEDULA AS VARCHAR(10))
RETURNS TABLE
AS
RETURN (SELECT
    CLIENTES.ID AS 'ID',
    CLIENTES.NOMBRE AS 'NOMBRE',
    CLIENTES.APELLIDO1 AS 'APELLIDO1',
    CLIENTES.APELLIDO2 AS 'APELLIDO2',
    CLIENTES.TELEFONO AS 'TELEFONO',
    CLIENTES.DIRECCION AS 'DIRECCION'
FROM CLIENTES
WHERE CLIENTES.CEDULA = coalesce(@CEDULA, CLIENTES.CEDULA))

-- CONSULTA CON LA FUNCION UN USUARIO DE PRUEBA --
Select * from BUSCA('603940994');
```

ID	NOMBRE	APELLIDO1	APELLIDO2	TELEFONO	DIRECCION
1	Adrian	Hernandez	Pomras	60766274	20 noviembre, Puntarenas



## Marco Teórico

- Santander United Kingdom es uno de los bancos bolivianos y uno de los mayores proveedores del país más recientes.
- El banco tiene aproximadamente 1,000 empleados, 1 millón de clientes activos, con casi 50 sucursales y 10 centros de negocios corporativos.

## Prioridades estratégicas

- Aumentar la vinculación de los clientes, aportando una experiencia del cliente excelente
- Simplificar y digitalizar el negocio para mejorar la eficiencia y la rentabilidad
- Continuar integrando la sostenibilidad en nuestro negocio
- La implementación de estrategias que ayuden a nuestros clientes y faciliten el trabajo a nuestros empleados.





## Descripción

- El presente trabajo tiene la finalidad de realizar interacciones de tal modo que se reflejen algunos de los procesos que pueden realizarse dentro de una entidad financiera.
- En el cual se llegan a utilizar todos los conocimientos adquiridos durante la materia en el semestre.
- Se hace el uso de la aplicación datagrip y las funciones a utilizar son el manejo de vistas, funciones y manejo de triggers.



# Creacion de la Base de Datos

```
CREATE DATABASE Banco_Kingdom;
```

```
DROP DATABASE Banco_Kingdom;
```

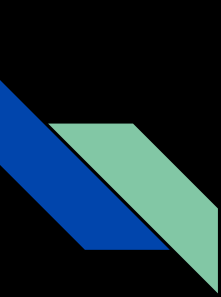
```
USE Banco_Kingdom;
```

```
CREATE TABLE operaciones
```

```
(  
    Codigo_operacion INTEGER(11) AUTO_INCREMENT PRIMARY KEY NOT NULL,  
    Descripcion      VARCHAR(50)  
);
```

```
CREATE TABLE transacciones
```

```
(  
    Codigo_transacciones INTEGER AUTO_INCREMENT PRIMARY KEY NOT NULL,  
    Numero_tarjeta      INT(11),  
    Codigo_operacion     INT(11),  
    fecha_de_transaccion Date,  
    cuenta_destino       VARCHAR(50),  
    monto                INTEGER,  
    FOREIGN KEY (Codigo_operacion) REFERENCES operaciones (Codigo_operacion),  
    FOREIGN KEY (Numero_tarjeta) REFERENCES Tarjeta (Numero_tarjeta)  
);
```

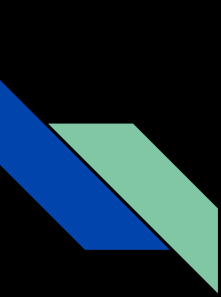


```
CREATE TABLE Tarjeta
(
    Numero_tarjeta    INTEGER AUTO_INCREMENT PRIMARY KEY NOT NULL,
    Codigo_cliente    INT(11),
    Codigo_cuenta      INT(11),
    fecha_afiliacion   Date,
    fecha_caducidad    Date,
    Saldo              INTEGER,
    FOREIGN KEY (Codigo_cliente) REFERENCES cliente (codigo_cliente),
    FOREIGN KEY (Codigo_cuenta) REFERENCES Tipo_cuenta (Codigo_cuenta)
);

CREATE TABLE Tipo_cuenta
(
    Codigo_cuenta      INTEGER AUTO_INCREMENT PRIMARY KEY,
    descripcion        VARCHAR(50)
);

CREATE TABLE cliente
(
    codigo_cliente     INTEGER AUTO_INCREMENT PRIMARY KEY NOT NULL,
    nombre             VARCHAR(50),
    apellido            VARCHAR(50)
);
```





```
INSERT INTO tarjeta (numero_tarjeta, Codigo_cliente, Codigo_cuenta, fecha_afiliacion,
fecha_caducidad, saldo)
VALUES (1001, 3025, 4012, '2021-10-03', '2024-10-03', 1000),
      (1002, 3026, 4013, '2021-01-04', '2024-01-04', 2000),
      (1003, 3027, 4014, '2021-09-12', '2024-09-12', 5002),
      (1004, 3028, 4015, '2021-01-01', '2024-01-01', 3008);
```

```
INSERT INTO cliente (codigo_cliente, nombre, apellido)
VALUES (3025, 'Juan Vargas', 'choque collque');
INSERT INTO cliente (codigo_cliente, nombre, apellido)
VALUES (3026, 'Richard Bautista', 'Sabedra castaño');
INSERT INTO cliente (codigo_cliente, nombre, apellido)
VALUES (3027, 'Vaneza Prado', 'Cuquimia Igor');
INSERT INTO cliente (codigo_cliente, nombre, apellido)
VALUES (3028, 'Palmer amidala', 'Cori Welsmayer');
```

```
INSERT INTO Tipo_cuenta (Codigo_cuenta, descripcion)
VALUES (4012, 'ESTANDAR');
INSERT INTO Tipo_cuenta (Codigo_cuenta, descripcion)
VALUES (4013, 'PREMIUM');
INSERT INTO Tipo_cuenta (Codigo_cuenta, descripcion)
VALUES (4014, 'ESTANDAR');
INSERT INTO Tipo_cuenta (Codigo_cuenta, descripcion)
VALUES (4015, 'PREMIUM');
```

```

INSERT INTO operaciones (Codigo_operacion, descripcion)
VALUES (6010, 'RETIRO');

INSERT INTO operaciones (Codigo_operacion, descripcion)
VALUES (7011, 'PRESTAMO');

INSERT INTO operaciones (Codigo_operacion, descripcion)
VALUES (8012, 'PRESTAMO');

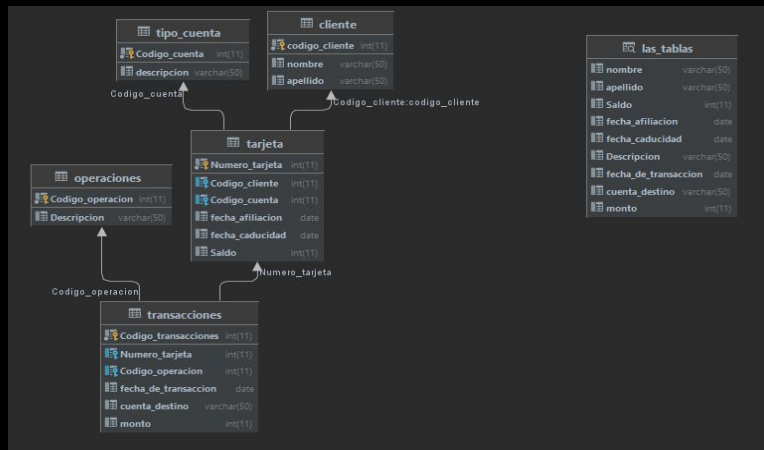
INSERT INTO operaciones (Codigo_operacion, descripcion)
VALUES (9013, 'RETIRO');

```

```

INSERT INTO transacciones (Codigo_transacciones, Numero_tarjeta, Codigo_operacion,
fecha_de_transaccion, cuenta_destino, monto)
VALUES (9110, 1001, 6010, '2021-05-04', 'Juanavarguera@gmail.com', 40),
(9111, 1002, 7011, '2021-04-01', 'Richardomalcolque@gmail.com', 100),
(9212, 1003, 8012, '2021-10-04', 'Thequinprado.com', 1000),
(9313, 1004, 9013, '2021-10-01', 'Palmerfiolder.com', 980);

```





## ¿Qué son las funciones?

- Las funciones integradas en SQL Server son una serie de rutinas almacenadas que reciben una serie de parámetros con los cuales realizan operaciones concretas para retornar un resultado específico.
- La sintaxis básica es: `create function NOMBRE (@PARAMETRO TIPO=VALORPORDEFEECTO) returns TIPO begin INSTRUCCIONES return VALOR end;` Luego del nombre se colocan (opcionalmente) los parámetros de entrada con su tipo. La cláusula "returns" indica el tipo de dato retornado.



## Ver quienes son PREMIUM y sus saldos del cliente

```
DROP FUNCTION IF EXISTS verquienessonPREMIUM
```

```
CREATE FUNCTION verquienessonPREMIUM(Codigo_cuenta VARCHAR(50) , descripcion VARCHAR(50))
```

```
RETURNS BOOL
```

```
BEGIN
```

```
RETURN Codigo_cuenta=descripcion;
```

```
end;
```

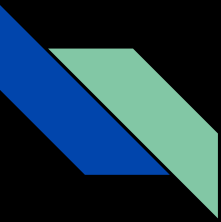
```
select cli.nombre,cli.apellido,tp.descripcion,tj.Saldo
```

```
from tarjeta AS tj
```

```
join cliente AS cli on tj.Codigo_cliente = cli.codigo_cliente
```

```
join Tipo_cuenta AS tp on tj.Codigo_cuenta = tp.Codigo_cuenta
```

```
where verquienessonPREMIUM(tp.descripcion,'PREMIUM');
```



## Buscar cliente con el codigo\_cliente

```
DROP FUNCTION IF EXISTS buscarcliente;
```

```
CREATE FUNCTION buscarcliente(Codigocuenta Integer , nombre VARCHAR(50))
```

```
RETURNS BOOL
```

```
BEGIN
```

```
RETURN Codigocuenta=nombre ;
```

```
end;
```

```
select tj.Codigo_cliente ,cli.nombre,cli.apellido,tj.Numero_tarjeta,tj.Saldo
```

```
from tarjeta AS tj
```

```
join cliente AS cli on tj.Codigo_cliente = cli.codigo_cliente
```

```
where buscarcliente(cli.codigo_cliente,3025);
```



# SALDO MINIMO

```
CREATE FUNCTION min_saldo() RETURNS int
```

```
BEGIN
```

```
return
```

```
(
```

```
SELECT min(tar.saldo)
```

```
FROM tarjeta AS tar
```

```
);
```

```
END;
```

```
SELECT min_saldo();
```

```
drop function min_saldo;
```



## función que verifica si existe un cliente o no

```
CREATE FUNCTION verificar(nombres varchar(50),apellidos varchar(50),  
nombres_comparar varchar(50),apellidos_comparar varchar(50))
```

```
returns bool
```

```
begin
```

```
    declare respuesta bool default false;
```

```
    set respuesta = (nombres =nombres_comparar and apellidos = apellidos_comparar);
```

```
    return respuesta;
```

```
end;
```

```
select
```

```
cli.codigo cliente,cli.nombre,cli.apellido,tar.fecha_afiliacion,tar.fecha_caducidad,tar.Sald  
o,tp.descripcion
```

```
from cliente as cli
```

```
INNER JOIN tarjeta AS tar ON cli.codigo cliente = tar.Codigo cliente
```

```
    INNER JOIN tipo_cuenta AS tp ON tar.Codigo_cuenta = tp.Codigo_cuenta
```

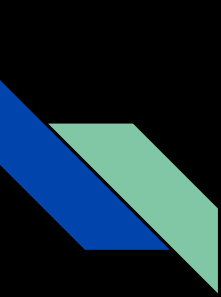
```
where verificar(cli.nombre, cli.apellido, 'Juan Vargas', 'choque collque');
```



## resta entre saldo y monto

```
create or replace function operaciones(num_tarjeta int, cod_transaccion int)
returns integer
begin
return
(
    select (sum(tar.Saldo) - sum(tra.monto)) as diferencia
    from tarjeta as tar,
    transacciones as tra
    where tar.Numero_tarjeta = num_tarjeta
    and tra.Codigo_transacciones = cod_transaccion
);
end;
select operaciones(1003,9212);
```





¿Qué son los triggers?

- Un disparador o trigger es una funcionalidad que la base de datos ejecuta de forma automática cuando se realiza una operación de tipo Insert, Update o Delete en una tabla o vista, o cuando se ejecuta una consulta SQL sobre una tabla o vista.
- Esto nos permite realizar acciones cuando se realiza una inserción, modificación o eliminación de un registro.



## CUENTAS ACTIVAS SEGÚN EL ADMINISTRADOR

```
create trigger tip_cuenta
before update
on tipo_cuenta
for each row
begin
    if new.descripcion =
        'ESTANDAR' or new.descripcion =
            'PREMIUM' then
        set new.estado = 'activo';
    else
        set new.estado = 'inactivo';
    end if;
end;
```

```
update tipo_cuenta
set descripcion =
    'PREMIUM'
where Codigo_cuenta = 4013
SELECT * from Tipo_cuenta ;
```



## añadir clientes

```
create or replace trigger before_agregar_cliente_update
```

```
before update
```

```
on cliente
```

```
for each row
```

```
begin
```

```
insert into cliente(codigo_cliente, nombre,  
apellido)
```

```
values  
(OLD.codigo_cliente, OLD.nombre, OLD.apellido);
```

```
end;
```

```
insert into cliente( nombre, apellido)
```

```
values ('Yeami Yanitsa', 'Sanchez Pisfil');
```

```
update cliente set nombre='Ludwing' where  
apellido ='Ibarra';
```

```
select *from cliente;
```

```
insert into cliente( nombre, apellido)
```

```
values ('Saul Elias', 'Canaza Herrera');
```

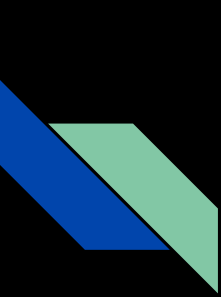
```
update cliente set nombre='Ludwing' where  
apellido ='Ibarra';
```

```
select *from cliente;
```



## ¿Qué son las vistas?

- Una vista es una tabla virtual cuyo contenido está definido por una consulta. Al igual que una tabla, una vista consta de un conjunto de columnas y filas de datos con un nombre.
- Las vistas suelen usarse para centrar, simplificar y personalizar la percepción de la base de datos para cada usuario. Las vistas pueden emplearse como mecanismos de seguridad, que permiten a los usuarios obtener acceso a los datos por medio de la vista, pero no les conceden el permiso de obtener acceso directo a las tablas base subyacentes de la vista.



## vista de todas las tablas juntas

```
create view las_tablas as
```

```
select cli.nombre,
```

```
cli.apellido,
```

```
tarje.Saldo,
```

```
tarje.fecha_afiliacion,
```

```
tarje.fecha_caducidad,
```

```
oper.Descripcion,
```

```
trans.fecha_de_transaccion,
```

```
trans.cuenta_destino,
```

```
trans.monto
```

```
from tarjeta as tarje
```

```
inner join transacciones as trans on  
tarje.Numero_tarjeta = trans.Numero_tarjeta
```

```
inner join cliente as cli on tarje.Codigo_cliente =  
cli.codigo_cliente
```

```
inner join tipo_cuenta as tc on tarje.Codigo_cuenta =  
tc.codigo_cuenta
```

```
inner join operaciones as oper on  
trans.Codigo_operacion = oper.Codigo_operacion;
```

```
select * from las_tablas;
```



## unir dos tablas

```
CREATE VIEW Registro
```

```
AS
```

```
SELECT cli.nombre, cli.apellido, Tar.Numero_tarjeta
```

```
FROM Cliente as cli
```

```
inner join tarjeta as tar on tar.Codigo_cliente = cli.codigo_cliente;
```

```
SELECT * FROM Registro;
```