2008

Universidad
Nacional de El
salvador

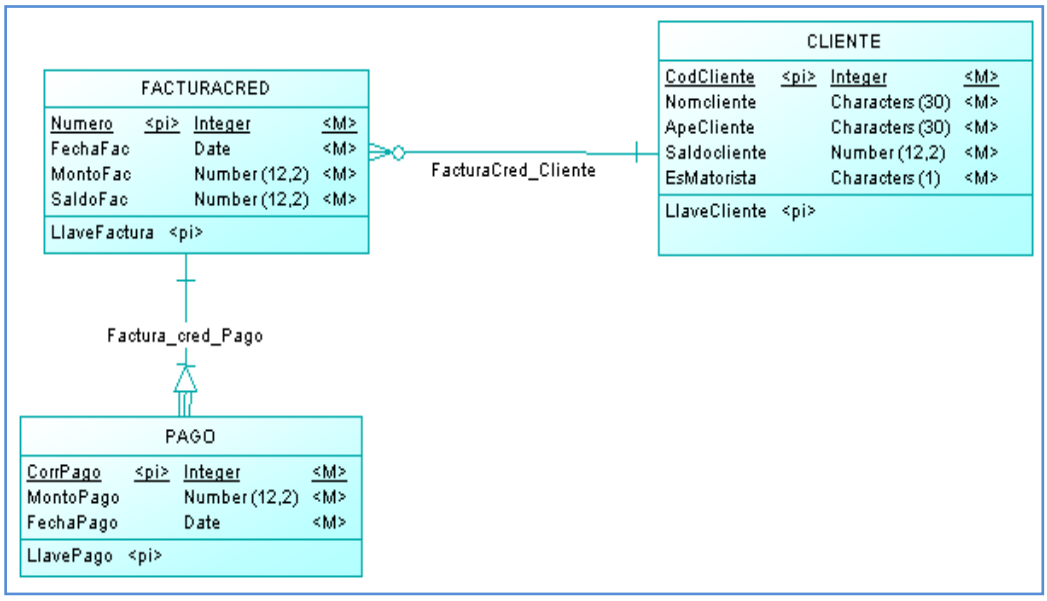
SOLUCION DEL PARCIAL III TRIGGERS Y PROCEDIMIENTOS.

El objetivo del tercer parcial es evaluar lo hecho en el proyecto, en el cual debemos aplicar triggers, procedimientos, funciones y sequence.

Indicaciones Preliminares:

- Haga una Conexión en el SQLDeveloper hacia su carnet igual como la hizo en Laboratorio con Nombre de Conexión, Usuario y Password con su **Carnet,** Hostname=**"servidor"** y Service Name=**"sistemas".**
- Elimine todos los objetos de su usuario.
- Si deja constancia de su trabajo en la PC, se le penalizara con 0.1 de nota.
- Analice el requerimiento aplicando todo lo visto en clase constraints, triggers, procedimientos y funciones almacenados en la base, y todo lo que considere necesario visto en clase.

1) Desarrolle un esquema de Base de Datos, en base al siguiente diagrama ER (30%)



2) Asuma que no se pueden meter Montos de Facturas de menos de 2 dólares, Si no se introducen Fechas en cualquier tabla se toma la fecha del Sistema, los saldos deberán ser mayor que cero. Si el cliente es Mayorista =M, si es Detalle =D únicamente.(10%)

3) Siempre que se añada un registro en la Tabla de FacturaRed, deberá de Incrementar el Saldo de Cliente con el monto de la factura (10%).

4) Siempre que se añada un registro en la Tabla de pagos, el valor del monto de pago, debe de bajar el saldo de la Factura y a su vez el saldo del cliente, y al eliminar un pago debe realizar la operación contraria (15%)

5) Haga un procedimiento almacenado que llene un Cursor Llamado "PagoClientesMayoristas" que contenga los Pagos hechos por clientes que son tipo ´M´. Y que muestre la suma total de ese tipo de pagos (20%).

6) Haga otro Procedimiento que sirva para introducir los registros mínimos para evaluar todas las situaciones que se definen en las preguntas anteriores. (15%)

Solución Clave: 1.

Primeramente se digita este código para poder ver los resultados impresos de los procedimientos y triggers.

```
set serveroutput on size unlimited;
begin
  dbms_output.ENABLE(100);
end;
```

Creación del Esquema.

Primero comenzamos haciendo las tablas.

```
CREATE TABLE CLIENTE
(
  CODCLIENTE INTEGER NOT NULL,
  NOMBRE VARCHAR2(20) NOT NULL,
  APELLIDO VARCHAR2(20) NOT NULL,
  DETALLE CHAR(1) NOT NULL,
  SALDOCLIENTE NUMBER(8,2)
);

CREATE TABLE FACTURA
(
  NUMERO INTEGER NOT NULL,
  CODCLIENTE INTEGER NOT NULL,
  SALDOFACT NUMBER (8,2) NOT NULL,
  MONTOFACT NUMBER (8,2) NOT NULL,
  FECHAFACT DATE NOT NULL
);

CREATE TABLE PAGO
(
  NUMERO INTEGER NOT NULL,
  CODPAGO INTEGER NOT NULL,
  MONTOPAGO NUMBER(8,2) NOT NULL,
  FECHAPAGO DATE NOT NULL
);
```

Hay que tomar en cuenta que el diagrama proporcionado por la cátedra es el modelo lógico y nosotros tenemos que crear las tablas en base al físico, es por eso que debemos manejar bien la teoría de las llaves primarias y foráneas.

La dependencia es un punto importante que se debe aplicar en estos ejercicios.

Para poder ejecutar estas sentencias solo hay que sombrear lo que queramos y luego presionar F5, o el botón: 

Es aconsejable que ejecutes uno por uno las sentencias no todas de una vez, para ver en cuál de ellas envía algún error.

Creación de las Constraints.

```
ALTER TABLE CLIENTE
ADD CONSTRAINT PK_CLIENTE
PRIMARY KEY (CODCLIENTE);

ALTER TABLE FACTURA
ADD CONSTRAINT PK_FACTURA
PRIMARY KEY (NUMERO);

ALTER TABLE PAGO
ADD CONSTRAINT PK_PAGO
PRIMARY KEY (NUMERO,CODPAGO);

ALTER TABLE FACTURA
ADD CONSTRAINT FK_FACTURA
FOREIGN KEY (CODCLIENTE)
REFERENCES CLIENTE;

ALTER TABLE PAGO
ADD CONSTRAINT FK_PAGO
FOREIGN KEY (NUMERO)
REFERENCES FACTURA;
```

En esta parte especificamos las llaves primarias y foráneas, la estructura es similar pero con algunas diferencias.

Algo importante es que en la tabla pago existe una llave foránea que forma parte de la llave primaria de esta, dando como resultado una llave compuesta gracias a la dependencia.

Algo que confunde es que a pesar de que numero ya se declaro como llave primaria debe declararse como foránea.

```
ALTER TABLE CLIENTE
ADD CONSTRAINT CK_CLIENTE_SALDO
CHECK (SALDOCLIENTE>=0);

ALTER TABLE FACTURA
ADD CONSTRAINT CK_FACTURA_SALDO
CHECK (SALDOFACT>=0);

ALTER TABLE CLIENTE
ADD CONSTRAINT CK_CLIENTE_DETALLE
CHECK (DETALLE='M' OR DETALLE='D');

ALTER TABLE FACTURA
ADD CONSTRAINT CK_FACTURA_MONTO
CHECK(MONTOFACT>=2);

ALTER TABLE PAGO
ADD CONSTRAINT CK_PAGO_MONTO
CHECK (MONTOPAGO>=2);

ALTER TABLE FACTURA
MODIFY(FECHAFACT DEFAULT SYSDATE);

ALTER TABLE PAGO
MODIFY (FECHAPAGO DEFAULT SYSDATE);
```

Ahora definiremos cada uno de las restricciones que se nos piden, como los saldos, los montos, las fechas por defecto, como por ejemplo el detalle del cliente que solo puede ser o mayorista ‘M’ o ‘D’.

Con los saldos de los clientes he considerado >=0 y no >0 ya que a la hora de cancelar todas las facturas su saldo llegara a 0.

Creación de Triggers.

El trigger **insertarfactura** debe de ejecutarse para actualizar el saldo del cliente, sumándole el saldo factura.

```
CREATE OR REPLACE TRIGGER INSERTFACTURA
AFTER INSERT ON FACTURA
REFERENCING NEW AS NEW OLD AS OLD
FOR EACH ROW
DECLARE
  v_CODCLIENTE CLIENTE.CODCLIENTE%TYPE;
  V_SALDOFACT FACTURA.SALDOFACT%TYPE;
  V_COUNT INTEGER;
BEGIN
  V_CODCLIENTE:=:NEW.CODCLIENTE;
  V_SALDOFACT:=:NEW.SALDOFACT;

  SELECT COUNT(*) INTO V_COUNT FROM CLIENTE WHERE CODCLIENTE=V_CODCLIENTE;
  IF(V_COUNT>0)THEN
    UPDATE CLIENTE SET SALDOCLIENTE=SALDOCLIENTE+ v_saldofact where codcliente=v_codcliente;
    dbms_output.put_line('CLIENTE ENCONTRADO');
  ELSE
    dbms_output.put_line('CLIENTE NO ENCONTRADO');
  END IF;
END;
```

El trigger **insertarpago** que tiene que dispararse cuando se ingrese un pago, es similar pero tiene que actualizar dos tablas, la de factura afectando al saldo de la factura como a la del cliente afectando de igual modo el saldo del cliente.

```
CREATE OR REPLACE TRIGGER INSERTPAGO
AFTER INSERT ON PAGO
REFERENCING NEW AS NEW OLD AS OLD
FOR EACH ROW
DECLARE
  V_CODCLIENTE CLIENTE.CODCLIENTE%TYPE;
  V_COUNT INTEGER;
  V_NUMERO FACTURA.NUMERO%TYPE;
  V_MONTOPAGO PAGO.MONTOPAGO%TYPE;
BEGIN
  V_NUMERO:=:NEW.NUMERO;
  V_MONTOPAGO:=:NEW.MONTOPAGO;
  SELECT COUNT(*) INTO V_COUNT FROM FACTURA WHERE NUMERO=V_NUMERO;
  IF(V_COUNT>0)THEN
    SELECT CODCLIENTE INTO V_CODCLIENTE FROM FACTURA WHERE V_NUMERO=NUMERO;
    UPDATE FACTURA SET SALDOFACT=SALDOFACT-V_MONTOPAGO WHERE NUMERO=V_NUMERO;
    UPDATE CLIENTE SET SALDOCLIENTE=SALDOCLIENTE-V_MONTOPAGO WHERE V_CODCLIENTE=CODCLIENTE;
    dbms_output.put_line('FACTURA ENCONTRADA');
  ELSE
    dbms_output.put_line('FACTURA NO ENCONTRADA');
  END IF;
END;
```

El trigger para **deletepago** debe de actualizar ambas tablas, Factura y Cliente es similar al de ingresar pago solo que esta vez se hace lo contrario, se resta el monto del pago al saldo de la Factura y al saldo del Cliente.

```
CREATE OR REPLACE TRIGGER DELETEPAGO
BEFORE DELETE ON PAGO
REFERENCING NEW AS NEW OLD AS OLD
FOR EACH ROW
DECLARE
  V_CODPAGO PAGO.CODPAGO%TYPE;
  V_CODCLIENTE CLIENTE.CODCLIENTE%TYPE;
  V_NUMERO FACTURA.NUMERO%TYPE;
  V_MONTOPAGO PAGO.MONTOPAGO%TYPE;
BEGIN
  V_NUMERO:=:OLD.NUMERO;
  V_MONTOPAGO:=:OLD.MONTOPAGO;
  SELECT CODCLIENTE INTO V_CODCLIENTE FROM FACTURA WHERE V_NUMERO=NUMERO;
  UPDATE FACTURA SET SALDOFACT=SALDOFACT+V_MONTOPAGO WHERE NUMERO=V_NUMERO;
  UPDATE CLIENTE SET SALDOCLIENTE=SALDOCLIENTE+V_MONTOPAGO WHERE CODCLIENTE=V_CODCLIENTE;
  dbms_output.put_line('PAGO ELIMINADO');
END;
```

Explicación de Trigger Insertar pago.

Se presenta una breve explicación del trigger de insertar pago ya que se asume que es el mayor grado de dificultad, debido a que los otros son similares.

Con este código podemos identificar:

```
CREATE OR REPLACE TRIGGER INSERTPAGO
AFTER INSERT ON PAGO
REFERENCING NEW AS NEW OLD AS OLD
FOR EACH ROW
```

- Estamos creando el trigger y asignándole un nombre, en este caso INSERTPAGO.
- Se define cuando debe de disparase, después de insertar en pago.
- Se referencia los registros nuevos como **NEW** y los viejos como **OLD**.
- Por último se define que la referencia es para cada registro.

```
DECLARE
  V_CODCLIENTE CLIENTE.CODCLIENTE%TYPE;
  V_COUNT INTEGER;
  V_NUMERO FACTURA.NUMERO%TYPE;
  V_MONTOPAGO PAGO.MONTOPAGO%TYPE;
```

Primero DECLARE, nos permite declarar variables que vayamos a ocupar dentro del código.

Se declara una variable del tipo del campo codcliente dentro de la tabla cliente.

Esta forma de declarar variables es muy segura ya que la variable es del mismo tipo, que el valor del campo que queremos guardar, y si se altera el tipo del campo no hay problema con el código.

Ahora lo que debemos hacer es recuperar los datos que nos sirvan para identificar tanto la factura como al cliente, con estos datos ya podemos actualizar las tablas necesarias.

```
  V_NUMERO:=:NEW.NUMERO;
  V_MONTOPAGO:=:NEW.MONTOPAGO;
```

Debemos igualar la variable correspondiente al campo que queremos del nuevo registro. Esto es posible por la sentencia NEW.

Luego de recupera esos valores podemos cerciorarnos de que la factura existe y esto puede hacerse con una consulta. Para eso tenemos la variable **v_count** en donde introducimos el resultado de contar a las facturas que poseen el **número factura** que recuperamos del nuevo registro.

```
BEGIN
  V_NUMERO:=:NEW.NUMERO;
  V_MONTOPAGO:=:NEW.MONTOPAGO;
  SELECT COUNT(*) INTO V_COUNT FROM FACTURA WHERE NUMERO=V_NUMERO;
  IF(V_COUNT>0)THEN
    SELECT CODCLIENTE INTO V_CODCLIENTE FROM FACTURA WHERE V_NUMERO=NUMERO;
    UPDATE FACTURA SET SALDOFACT=SALDOFACT-V_MONTOPAGO WHERE NUMERO=V_NUMERO;
    UPDATE CLIENTE SET SALDOCLIENTE=SALDOCLIENTE-V_MONTOPAGO WHERE V_CODCLIENTE=CODCLIENTE;
    dbms_output.put_line('FACTURA ENCONTRADA');
  ELSE
    dbms_output.put_line('FACTURA NO ENCONTRADA');
  END IF;
END;
```

Con la ayuda de un **IF** verificamos que exista, si existe actualizamos las tablas y si no imprimimos un mensaje que nos indique que acción no se efectuó ya que no se encontró la factura.

Asumiendo que la factura existe debemos recuperar el código cliente dentro de esa factura, entonces necesitamos otra consulta para recuperar el código cliente y por ende otra variable para guardarlo. Luego solo actualizamos las tablas usando estos valores, y imprimimos un mensaje para demostrar que la operación concluyo con éxito.

Importante, este trigger se disparo después (AFTER) de la inserción, pero para eliminar un pago debe hacerse antes, y en vez de **NEW usar OLD.**

Creación del cursor.

El cursor es una tabla temporal donde guardamos el resultado de una consulta para desplegarlo en pantalla, es por ello que utilizamos un for para recorrerlo e imprimimos con la variable que ayuda al for a contar.

```
CREATE OR REPLACE PROCEDURE MAYORISTAS
IS
  CURSOR c_mayo is select sum(montopago) TOTAL from cliente join factura
  using (codcliente) join pago using (numero)
  where detalle='M';
BEGIN
  for r in c_mayo loop
  dbms_output.put_line('MONTO TOTAL DE PAGOS DE MAYORISTA:'||r.TOTAL);
  END loop;
END MAYORISTAS;
```

La clave para mostrar los datos solicitados en el parcial se encuentra en la consulta SQL, que a estas alturas no debería ser un problema, hay muchas formas de aplicar cursores pero ahora nos decidimos por la más sencilla. También el nombre de nuestro cursor es mayoristas y el del parcial es otro pero la funcionalidad es igual.

Creación del Proceso de Llenado.

La creación del proceso es sencilla, solo usa un case para elegir que acción hacer primero ya sea insertar **clientes**, **facturas**, **pagos**. Los datos deben ayudarte a comprobar que todo esté bien, debes saber asociar los datos para saber el resultado que esperas.

El valor de la acción se la mandamos cuando ejecutamos el procedimiento, que se verá después.

```
CREATE OR REPLACE PROCEDURE INSERTAR(ACCION VARCHAR2)
IS
  EXCEPTION1 EXCEPTION;
BEGIN
  CASE ACCION
  WHEN 'CLIENTE'THEN
    INSERT INTO CLIENTE VALUES(1,'DANIEL','PEÑALVA','M',0);
    INSERT INTO CLIENTE VALUES(2,'DOUGLAS','TOBAR','M',0);
    INSERT INTO CLIENTE VALUES(3,'LUCIO','RAMIREZ','D',0);
    INSERT INTO CLIENTE VALUES(4,'CARLOS','PEREZ','M',0);
    INSERT INTO CLIENTE VALUES(5,'REBECCA','PEÑALVA','D',0);
  WHEN 'FACTURA'THEN
    INSERT INTO FACTURA VALUES(1,1,500,500,'10/06/2008');
    INSERT INTO FACTURA VALUES(2,2,500,500,'10/06/2008');
    INSERT INTO FACTURA VALUES(3,1,500,500,'10/06/2008');
    INSERT INTO FACTURA VALUES(4,3,500,500,'10/06/2008');
    INSERT INTO FACTURA VALUES(5,4,500,500,'10/06/2008');
  WHEN 'PAGO' THEN
    INSERT INTO PAGO VALUES(1,1,100,'24/6/2008');
    INSERT INTO PAGO VALUES(1,2,100,'24/6/2008');
    INSERT INTO PAGO VALUES(2,1,100,'24/6/2008');
    INSERT INTO PAGO VALUES(4,1,100,'24/6/2008');
    INSERT INTO PAGO VALUES(1,3,100,'24/6/2008');
  ELSE
    RAISE EXCEPTION1;
  END CASE;
  EXCEPTION
  WHEN EXCEPTION1 THEN
  dbms_output.put_line('ACCION NO VALIDA');
END INSERTAR;
```

Creación de un proceso para eliminar un pago especifico.

Esto no se solicita en el parcial solo se ha hecho para facilidad de ejecutar los procedimientos, y demostrar la utilidad de los procedimientos.

```
CREATE OR REPLACE PROCEDURE ELIMINARPAGO(COD PAGO.CODPAGO%TYPE,NUM PAGO.NUMERO%TYPE)
IS
BEGIN
  DELETE FROM PAGO WHERE CODPAGO=COD AND numero=NUM;
END ELIMINARPAGO;
```

A este procedimiento se le envían dos valores, los cuales es interesante ver como se declaran en el procedimiento, porque eso demuestra que es una forma práctica de declarar variables.

Controlando la Ejecución de todo.

```
--LLENADO DE TABLAS
EXECUTE INSERTAR('CLIENTE');
EXECUTE INSERTAR('FACTURA');
EXECUTE INSERTAR('PAGO');
--VER LAS TABLAS
SELECT * FROM CLIENTE;
SELECT * FROM FACTURA;
SELECT * FROM PAGO;
--ELINAR LAS TABLAS
DELETE FROM CLIENTE;
DELETE FROM FACTURA;
DELETE FROM PAGO;
--ELIMINAR UN PAGO
EXECUTE ELIMINARPAGO(1,1);
--CURSOR
EXECUTE MAYORISTAS;
--OTRAS
COMMIT;
ROLLBACK;
```

- Ejecución del proceso de llenado en el orden necesario.
- Ver el resultado del llenado de las tablas.
- Borrar los registros de las tablas.
- Ejecución del procedimiento eliminar un pago.
- Ejecución del cursor.
- Grabar y Deshacer.

Con este código será fácil controlar la ejecución de los procedimientos, si todas las acciones se efectúan bien debe de resultar. Primero ingresamos los clientes los cuales su saldo será cero por no haber introducido las facturas aun, pero ahora introducimos las facturas:

	NUMERO	CODCLIENTE	SALDOFACT	MONTOFACT	FECHAFACT
1	1	1	500	500	10/06/08
2	2	2	500	500	10/06/08
3	3	1	500	500	10/06/08
4	4	3	500	500	10/06/08
5	5	4	500	500	10/06/08

Ahora vemos los clientes los cuales se encuentran así:

	CODCLIENTE	NOMBRE	APELLIDO	DETALLE	SALDOCLIENTE
1	1	DANIEL	PEÑALVA	M	1000
2	2	DOUGLAS	TOBAR	M	500
3	3	LUCIO	RAMIREZ	D	500
4	4	CARLOS	PEREZ	M	500
5	5	REBECCA	PEÑALVA	D	0

Los datos deben concordar para saber que los triggers funcionan, ahora introducimos los pagos de cada cliente.

	NUMERO	CODPAGO	MONTOPAGO	FECHAPAGO
1	1	1	100	24/06/08
2	1	2	100	24/06/08
3	2	1	100	24/06/08
4	4	1	100	24/06/08
5	1	3	100	24/06/08

Ahora vemos que las facturas y los clientes deben haber sido actualizados:

	NUMERO	CODCLIENTE	SALDOFACT	MONTOFACT	FECHAFACT
1	1	1	200	500	10/06/08
2	2	2	400	500	10/06/08
3	3	1	500	500	10/06/08
4	4	3	400	500	10/06/08
5	5	4	500	500	10/06/08

Vemos que las actualizaciones de las facturas concuerdan con las de los clientes:

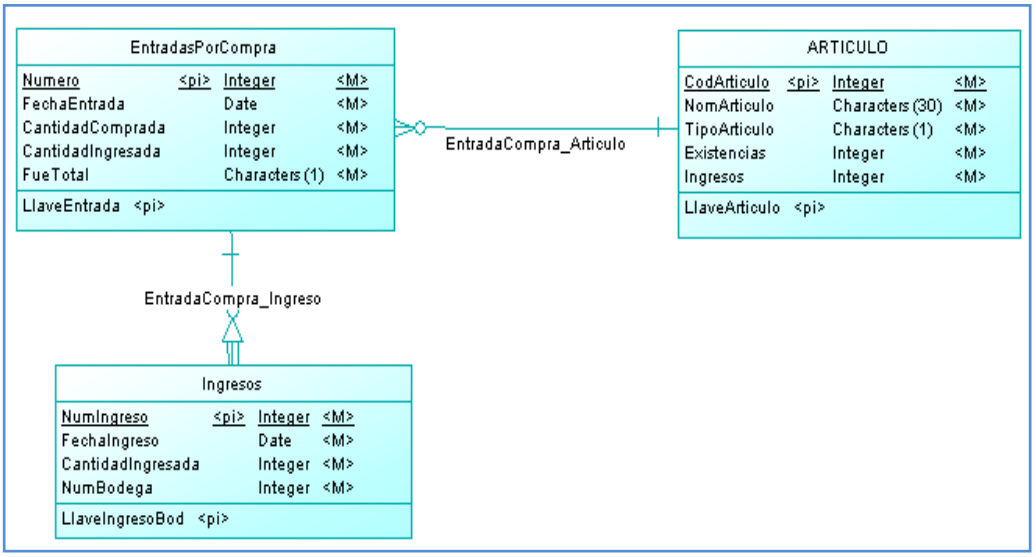
	CODCLIENTE	NOMBRE	APELLIDO	DETALLE	SALDOCLIENTE
1	1	DANIEL	PEÑALVA	M	700
2	2	DOUGLAS	TOBAR	M	400
3	3	LUCIO	RAMIREZ	D	400
4	4	CARLOS	PEREZ	M	500
5	5	REBECCA	PEÑALVA	D	0

Las otras claves son similares por lo que no se explicaran como esta, un punto importante es que para borrar todas las tablas debe seguirse el orden de dependencia. Es bueno saber resolver la mayor cantidad de enunciados posibles.

Indicaciones Preliminares:

- Haga una Conexión en el SQLDeveloper hacia su carnet igual como la hizo en Laboratorio con Nombre de Conexión, Usuario y Password con su **Carnet,** Hostname=**"servidor"** y Service Name=**"sistemas".**
- Elimine todos los objetos de su usuario.
- Si deja constancia de su trabajo en la PC, se le penalizara con 0.1 de nota.
- Analice el requerimiento aplicando todo lo visto en clase constraints, triggers, procedimientos y funciones almacenados en la base, y todo lo que considere necesario visto en clase.

1) Desarrolle un esquema de Base de Datos, en base al siguiente diagrama ER (30%)



2) Asuma que los artículos que no se les introduce existencia como mínimo deberá ser 10, Si no se introducen Fechas en cualquier tabla se toma la fecha del Sistema, los artículos pueden ser tipo A,B, C,D y E.(10%)

3) Siempre que se añada un registro en la Tabla de EntradaPorCompra, deberá de Incrementar el Valor de las Existencias con la CantidadComprada de dichoArticulo(10%).

4) Siempre que se añada un registro en la Tabla de Ingresos, el valor del campo CantidadIngresada, debe de incrementarse en EntradaPorCompra y a su vez el valor de Ingresos en el Artículo.Y al eliminar un Ingreso de dicha tabla, debe realizar la operación contraria (15%)

5) Haga un procedimiento almacenado que llene un Cursor Llamado "IngresosTipoA" que contenga los Ingresos hechos de Artículos Tipo "A". Y que muestre la suma total de ese tipo de pagos (20%).

6) Haga otro Procedimiento que sirva para introducir los registros mínimos para evaluar todas las situaciones que se definen en las preguntas anteriores.(15%)

Solución Clave: 2.

Primeramente se digita este código para poder ver los resultados impresos de los procedimientos y triggers.

```
set serveroutput on size unlimited;
begin
  dbms_output.ENABLE(100);
end;
```

Creación del Esquema.

Primero comenzamos haciendo las tablas.

```
CREATE TABLE ARTICULO
(
  codarticulo integer not null,
  nomarticulo character(30) not null,
  tipoarticulo character(1) not null,
  existencias integer not null,
  ingresos integer not null
);

CREATE TABLE ENTRADASPORCOMPRA
(
  numero integer not null,
  codarticulo integer not null,
  fechaentrada date not null,
  cantidadcomprada integer not null,
  cantidadingresada integer not null,
  fuetotal character(1) not null
);

CREATE TABLE INGRESOS
(
  numingreso integer not null,
  numero integer not null,
  fechaingreso date not null,
  cantidadingresada integer not null,
  numbodega integer not null
);
```

Creación de las Constraints.

Estas constraint definen las llaves primarias y foráneas.

```
ALTER TABLE ARTICULO
ADD CONSTRAINT PK_ARTICULO
PRIMARY KEY (CODARTICULO);

ALTER TABLE ENTRADASPORCOMPRA
ADD CONSTRAINT PK_ENTRADASPORCOMPRA
PRIMARY KEY (NUMERO);

ALTER TABLE INGRESOS
ADD CONSTRAINT PK_INGRESOS
PRIMARY KEY (NUMERO,NUMINGRESO);

ALTER TABLE ENTRADASPORCOMPRA
ADD CONSTRAINT FK_ENTRADASPORCOMPRA
FOREIGN KEY (CODARTICULO)
REFERENCES ARTICULO;

ALTER TABLE INGRESOS
ADD CONSTRAINT FK_INGRESOS
FOREIGN KEY (NUMERO)
REFERENCES ENTRADASPORCOMPRA;
```

Las restantes constraint son para garantizar que los datos a ser introducidos cumplan con las condiciones estipuladas en las constraint. Como lo es el caso de las fechas que al no introducirse debe tomar la del sistema, o el tipo de artículos.

```
ALTER TABLE ARTICULO
MODIFY (EXISTENCIAS DEFAULT 10);

ALTER TABLE ENTRADASPORCOMPRA
MODIFY (FECHAENTRADA DEFAULT SYSDATE);

ALTER TABLE INGRESOS
MODIFY (FECHAINGRESO DEFAULT SYSDATE);

ALTER TABLE ARTICULO
ADD CONSTRAINT CK_TIPOARTICULO
CHECK (TIPOARTICULO='A' OR TIPOARTICULO='B'
OR TIPOARTICULO='C' OR TIPOARTICULO='D' OR TIPOARTICULO='E');
```

Creación de Triggers.

El trigger **insertentradasporcompra** actualiza el valor de existencia de los artículos al realizar una compra que se registra en la tabla entradas por compra.

```
CREATE OR REPLACE TRIGGER INSERTENTRADASPORCOMPRA
AFTER INSERT ON ENTRADASPORCOMPRA
REFERENCING NEW AS NEW OLD AS OLD
FOR EACH ROW
DECLARE
  V_CODARTICULO ARTICULO.CODARTICULO% TYPE;
  V_CANTIDADCOMPRADA ENTRADASPORCOMPRA.CANTIDADCOMPRADA% TYPE;
  V_COUNT INTEGER;
BEGIN
  V_CODARTICULO:=:NEW.CODARTICULO;
  V_CANTIDADCOMPRADA:=:NEW.CANTIDADCOMPRADA;
  SELECT COUNT(*) INTO V_COUNT FROM ARTICULO WHERE codarticulo=V_CODARTICULO;
  IF(V_COUNT > 0) THEN
    UPDATE ARTICULO SET existencias= existencias+V_CANTIDADCOMPRADA WHERE V_CODARTICULO= codarticulo;
    dbms_output.put_line('ARTICULO ENCONTRADO');
  ELSE
    dbms_output.put_line('ARTICULO NO ENCONTRADO');
  END IF;
END;
```

El trigger **insertingresos** actualiza dos tablas Entradas por compra en el campo cantidad ingresada e ingresos en el campo ingresos, con estos campos puede llevarse control de las cantidades compradas y las ingresadas.

```
CREATE OR REPLACE TRIGGER INSERTINGRESOS
AFTER INSERT ON INGRESOS
REFERENCING NEW AS NEW OLD AS OLD
FOR EACH ROW
DECLARE
  V_NUMERO INGRESOS.NUMERO%TYPE;
  V_CODARTICULO ARTICULO.CODARTICULO%TYPE;
  V_CANTIDAD INGRESOS.CANTIDADINGRESADA%TYPE;
  V_COUNT INTEGER;
BEGIN
  V_CANTIDAD:=:NEW.CANTIDADINGRESADA;
  V_NUMERO:=:NEW.NUMERO;
  SELECT COUNT(*) INTO V_COUNT FROM ENTRADASPORCOMPRA WHERE NUMERO=V_NUMERO;
  IF(V_COUNT>0)THEN
    SELECT CODARTICULO INTO V_CODARTICULO FROM ENTRADASPORCOMPRA WHERE NUMERO=V_NUMERO;
    UPDATE ENTRADASPORCOMPRA SET CANTIDADINGRESADA=CANTIDADINGRESADA+V_CANTIDAD WHERE NUMERO=V_NUMERO;
    UPDATE ARTICULO SET INGRESOS=INGRESOS+V_CANTIDAD WHERE V_CODARTICULO=CODARTICULO;
    dbms_output.put_line('ENTRADA POR COMPRA ENCONTRADA');
  ELSE
    dbms_output.put_line('ENTRADA POR COMPRA NO ENCONTRADA');
  END IF;
END;
```

El trigger **deleteingresos** sirve para actualizar las dos tablas mencionadas en el trigger anterior, solo de manera diferente ya que media vez se elimine un ingreso deben retarse los valores.

```
CREATE OR REPLACE TRIGGER DELETEINGRESOS
BEFORE DELETE ON INGRESOS
REFERENCING NEW AS NEW OLD AS OLD
FOR EACH ROW
DECLARE
  V_NUMERO INGRESOS.NUMERO%TYPE;
  V_CODARTICULO ARTICULO.CODARTICULO%TYPE;
  V_CANTIDAD INGRESOS.CANTIDADINGRESADA%TYPE;
  V_COUNT INTEGER;
BEGIN
  V_CANTIDAD:=:OLD.CANTIDADINGRESADA;
  V_NUMERO:=:OLD.NUMERO;
  SELECT COUNT(*) INTO V_COUNT FROM ENTRADASPORCOMPRA WHERE NUMERO=V_NUMERO;
  IF(V_COUNT>0)THEN
    SELECT CODARTICULO INTO V_CODARTICULO FROM ENTRADASPORCOMPRA WHERE NUMERO=V_NUMERO;
    UPDATE ENTRADASPORCOMPRA SET CANTIDADINGRESADA=CANTIDADINGRESADA-V_CANTIDAD WHERE NUMERO=V_NUMERO;
    UPDATE ARTICULO SET INGRESOS=INGRESOS-V_CANTIDAD WHERE CODARTICULO=V_CODARTICULO;
    dbms_output.put_line('INGRESO ELIMINADO');
  ELSE
    dbms_output.put_line('INGRESO ELIMINADO');
  END IF;
END;
```

Creación del cursor.

Este cursor es similar al de la solución 1, pero la diferencia es que debemos consultar la información de artículos.

```
CREATE OR REPLACE PROCEDURE CURSORTIPOA
IS
  CURSOR INGRESOSTIPOA IS SELECT SUM(CANTIDADINGRESADA) TOTAL FROM ARTICULO
  JOIN ENTRADASPORCOMPRA USING (CODARTICULO)
  JOIN INGRESOS USING (NUMERO)
  WHERE TIPOARTICULO='A';
BEGIN
    FOR r in ingresostipoa loop
    dbms_output.put_line('TOTAL DE INGRESOS TIPO A:'||r.TOTAL);
    end loop;
END CURSORTIPOA;
```

Creación del Proceso de Llenado.

La creación del proceso es sencilla, solo usa un case para elegir que acción hacer primero ya sea insertar **artículos**, **entradasporcompra**, **ingresos**. Los datos deben ayudarte a comprobar que todo esté bien, debes saber asociar los datos para saber el resultado que esperas.

El valor de la acción se la mandamos cuando ejecutamos el procedimiento, que se verá después.

```
CREATE OR REPLACE PROCEDURE INSERTAR(ACCION VARCHAR2)
IS
  EXCEPTION1 EXCEPTION;
BEGIN
  CASE ACCION
  WHEN 'ARTICULO' THEN
    INSERT INTO ARTICULO VALUES(1,'DISCOS DUROS 500 G','A',10,0);
    INSERT INTO ARTICULO VALUES(2,'DISCOS DUROS 250 G','B',10,0);
    INSERT INTO ARTICULO VALUES(3,'DISCOS DUROS 160 G','C',10,0);
    INSERT INTO ARTICULO VALUES(4,'DISCOS DUROS 120 G','D',10,0);
    INSERT INTO ARTICULO VALUES(5,'DISCOS DUROS 80 G','E',10,0);
    INSERT INTO ARTICULO VALUES(6,'DISCOS DUROS 350 G','A',10,0);
  WHEN 'ENTRADASPORCOMPRA' THEN
    INSERT INTO ENTRADASPORCOMPRA VALUES(1,1,'07/07/2008',100,0,'N');
    INSERT INTO ENTRADASPORCOMPRA VALUES(2,3,'10/07/2008',100,0,'N');
    INSERT INTO ENTRADASPORCOMPRA VALUES(3,4,'15/07/2008',100,0,'N');
    INSERT INTO ENTRADASPORCOMPRA VALUES(4,5,'09/07/2008',100,0,'N');
    INSERT INTO ENTRADASPORCOMPRA VALUES(5,6,'15/07/2008',100,0,'N');
  WHEN 'INGRESOS' THEN
    INSERT INTO INGRESOS VALUES(1,1,'08/07/2008',50,1);
    INSERT INTO INGRESOS VALUES(2,2,'11/07/2008',60,2);
    INSERT INTO INGRESOS VALUES(3,3,'16/07/2008',70,3);
    INSERT INTO INGRESOS VALUES(4,4,'09/07/2008',75,4);
    INSERT INTO INGRESOS VALUES(5,5,'08/07/2008',25,1);
  ELSE
  RAISE EXCEPTION1;
  END CASE;
  EXCEPTION
  WHEN EXCEPTION1 THEN
    dbms_output.put_line('LA ACCION NO ES VALIDA');
END INSERTAR;
```

Creación de un proceso para eliminar un ingreso especifico.

Esto no se solicita en el parcial solo se ha hecho para facilidad de ejecutar los procedimientos, y demostrar la utilidad de los procedimientos.

```
CREATE OR REPLACE PROCEDURE ELIMINARINGRESO(NUM INGRESOS.NUMERO% TYPE, ING INGRESOS.NUMINGRESO% TYPE)
IS
BEGIN
  DELETE FROM INGRESOS WHERE num=NUMERO AND ING= numingreso;
END ELIMINARINGRESO;
```

A este procedimiento se le envían dos valores, los cuales es interesante ver como se declaran en el procedimiento, porque eso demuestra que es una forma práctica de declarar variables.

Controlando la Ejecución de todo.

```
--LLENADO DE TABLAS
EXECUTE INSERTAR('ARTICULO');
EXECUTE INSERTAR('ENTRADASPORCOMPRA');
EXECUTE INSERTAR('INGRESOS');
--VER LAS TABLAS
SELECT * FROM ARTICULO;
SELECT * FROM ENTRADASPORCOMPRA;
SELECT * FROM INGRESOS;
--ELINAR LAS TABLAS
DELETE FROM INGRESOS;
DELETE FROM ENTRADASPORCOMPRA;
DELETE FROM ARTICULO;
--ELIMINAR UN INGRESO
EXECUTE ELIMINARINGRESO(1,1);
--CURSOR
EXECUTE CURSORTIPOA;
--OTRAS
COMMIT;
ROLLBACK;
```

Este código es el mismo visto para controlar la ejecución de las diferentes sentencias que componen todo el parcial.

Ahora veremos los datos que debemos obtener si la ejecución de todas las sentencias fue correcta. Primeramente hay que introducir los artículos, claro que la existencia será de 10 por defecto pero los ingresos serán cero ya que no se han realizado ninguno.

	NUMERO	CODARTICULO	FECHAENTRADA	CANTIDADCOMPRADA	CANTIDADINGRESADA	FUETOTAL
1	1	1	07/07/08	100	0	N
2	2	3	10/07/08	100	0	N
3	3	4	15/07/08	100	0	N
4	4	5	10/07/08	100	0	N
5	5	6	15/07/08	100	0	N

Ahora vemos los artículos los cuales se encuentran así:

	CODARTICULO	NOMARTICULO	TIPOARTICULO	EXISTENCIAS	INGRESOS
1	1	DISCOS DUROS 500 G	A	110	0
2	2	DISCOS DUROS 250 G	B	10	0
3	3	DISCOS DUROS 160 G	C	110	0
4	4	DISCOS DUROS 120 G	D	110	0
5	5	DISCOS DUROS 80 G	E	110	0
6	6	DISCOS DUROS 350 G	A	110	0

Como se ve en los datos las existencias de los artículos han variado en 100, lo cual indica que el trigger realizo los cambios, ahora introducimos los ingresos para los artículos.

	NUMINGRESO	NUMERO	FECHAINGRESO	CANTIDADINGRESADA	NUMBODEGA
1	1	1	08/07/08	50	1
2	2	2	11/07/08	60	2
3	3	3	16/07/08	70	3
4	4	4	10/07/08	75	4
5	5	5	08/07/08	25	1

Ahora vemos las entradas por compra y artículo deben de haber sido actualizados:

	NUMERO	CODARTICULO	FECHAENTRADA	CANTIDADCOMPRADA	CANTIDADINGRESADA	FUETOTAL
1	1	1	07/07/08	100	50	N
2	2	3	10/07/08	100	60	N
3	3	4	15/07/08	100	70	N
4	4	5	10/07/08	100	75	N
5	5	6	15/07/08	100	25	N

Vemos que las actualizaciones de entradas por compra concuerdan con la de los artículos.

	CODARTICULO	NOMARTICULO	TIPOARTICULO	EXISTENCIAS	INGRESOS
1	1	DISCOS DUROS 500 G	A	110	50
2	2	DISCOS DUROS 250 G	B	10	0
3	3	DISCOS DUROS 160 G	C	110	60
4	4	DISCOS DUROS 120 G	D	110	70
5	5	DISCOS DUROS 80 G	E	110	75
6	6	DISCOS DUROS 350 G	A	110	25

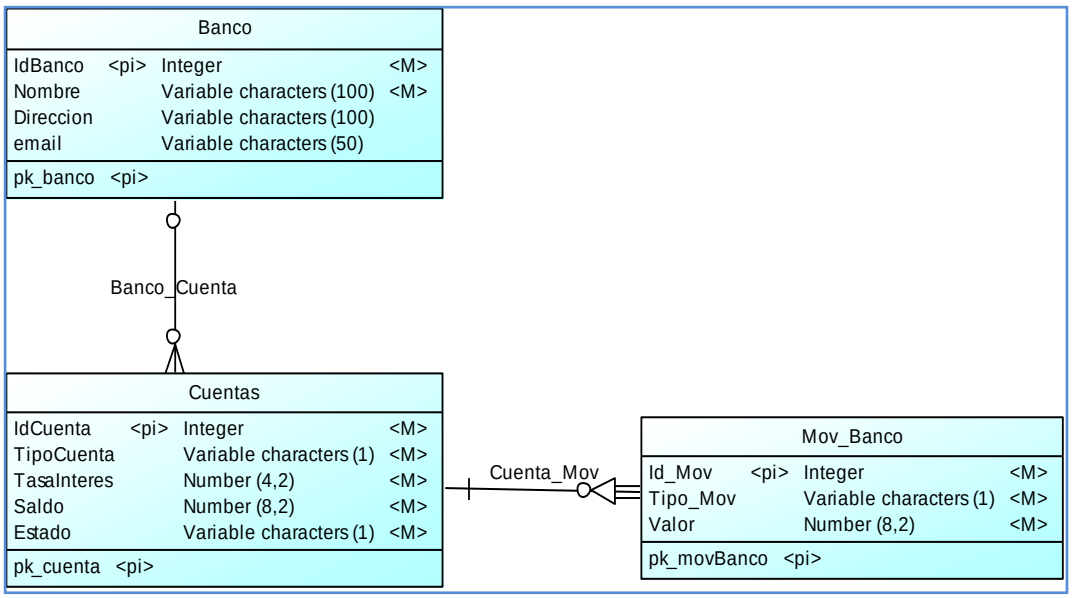
Los datos de la ejecución del cursor deben de dar la suma de los ingresos para los únicos dos artículos de tipo 'A', el total es de: 75.

Podemos ver que los datos cambian cuando eliminamos un ingreso, esto con ayuda del proceso para eliminar un ingreso especifico.

Indicaciones Preliminares:

- Haga una Conexión en el SQLDeveloper hacia su carnet igual como la hizo en Laboratorio con Nombre de Conexión, Usuario y Password con su **Carnet,** Hostname=**"servidor"** y Service Name=**"sistemas".**
- Elimine todos los objetos de su usuario.
- Si deja constancia de su trabajo en la PC, se le penalizara con 0.1 de nota.
- Analice el requerimiento aplicando todo lo visto en clase constraints, triggers, procedimientos y funciones almacenados en la base, y todo lo que considere necesario visto en clase.

1) Desarrolle un esquema de Base de Datos, en base al siguiente diagrama ER (30%)



2) El email del banco deberá ser único por cada banco ingresado, El tipo de cuenta deberá ser A = Ahorro, C=Corriente únicamente, La tasa de interés debe ser mayor o igual a .25 y si no se introduce algún valor deberá tomar 1.00, El saldo deberá ser mayor o igual a cien dólares, y el estado de la cuenta es Activo=A o Inactivo=I. El tipo de Movimiento bancario es entrada=E o salida=S y el valor deberá ser mayor que cero. (10%)

3) Desarrolle un proceso en el cual modifique el saldo de la cuenta si al añadir entrada de movimiento bancario sume al saldo de la cuenta o reste si es salida, tomando en cuenta que si es ingreso deberá aplicarle la tasa de interés de la cuenta automáticamente.(20%)

4) Desarrolle otro proceso en el cual si el saldo de la cuenta llega a 100, ponga en estado inactivo la cuenta y elimine todos los movimientos bancarios de dicha cuenta de manera automática. Antes de eliminarlo deberá guardar la cantidad de movimientos de esa cuenta y el usuario que inactivo la cuenta en una tabla denominada historia (25%)

5) Desarrolle un procedimiento que permita ingresar dos bancos, dos cuentas y dos movimientos bancarios por cuenta evaluando las situaciones de las preguntas anteriores (15%)