ESTUDIANTE: Cristian Jurado

CARRERA: Software **PARALELO:** A **NIVEL:** 4

ASIGNATURA: Base de Datos

DOCENTE: Ing. Edwin Hernando Buenaño

FECHA: 30/06/2025

TALLER PRÁCTICO: Creación Base de datos:

```
SQL> CONNECT SYSTEM;
Introduzca la contraseña:
Conectado.
SQL> CREATE USER BANCO IDENTIFIED BY B123;

Usuario creado.

SQL> GRANT CONNECT, RESOURCE, UNLIMITED TABLESPACE TO BANCO;

Concesi¾n terminada correctamente.
```

```
SQL> CONNECT BANCO;
Introduzca la contraseña:
Conectado.
SQL> CREATE TABLE CLIENTES (
  2    ID_CLI      VARCHAR(10) PRIMARY KEY,
  3    NOM_CLI     VARCHAR(10),
  4    APE_CLI     VARCHAR(20),
  5    DIR_CLI     VARCHAR(100)
  6  );

Tabla creada.

SQL> CREATE TABLE CUENTAS (
  2    NUM_CUE     VARCHAR(10) PRIMARY KEY,
  3    TIP_CUE     VARCHAR(20),
  4    SAL_CUE     NUMBER,
  5    ANIO_APE    NUMBER,
  6    ID_CLI_PER  VARCHAR(10) NOT NULL REFERENCES CLIENTES(ID_CLI)
  7  );

Tabla creada.

SQL> CREATE TABLE CAJEROS_AUT (
  2    ID_CAJ      NUMBER PRIMARY KEY,
  3    UBI_CAJ     VARCHAR(100),
  4    FEC_CAL_CAJ DATE
  5  );

Tabla creada.

SQL> CREATE TABLE TRANSFERENCIAS (
  2    NUM_TRA     VARCHAR(10) PRIMARY KEY,
  3    FEC_HOR_TRA DATE,
  4    VALOR_TRA   NUMBER,
  5    CUE_ORI     VARCHAR(10) NOT NULL REFERENCES CUENTAS(NUM_CUE),
  6    CUE_DES     VARCHAR(10) NOT NULL REFERENCES CUENTAS(NUM_CUE)
  7  );

Tabla creada.
```

CREACIÓN DE LA SECUENCIA:

```
SQL> CREATE SEQUENCE SEQ_TRANSFERENCIAS START WITH 1 INCREMENT BY 1;

Secuencia creada.
```

CREACIÓN DEL PROCEDURE DEPOSITAR:

```
SQL> CREATE OR REPLACE PROCEDURE DEPOSITAR( P_NUM_CUE VARCHAR, P_VALOR_DEP NUMBER, P_ID_CAJ_DEP NUMBER)
  2  IS
  3    V_EXISTE NUMBER;
  4  BEGIN
  5    SELECT COUNT(*) INTO V_EXISTE FROM CUENTAS WHERE NUM_CUE = P_NUM_CUE;
  6    IF V_EXISTE = 0 THEN
  7      RAISE_APPLICATION_ERROR(-20011, 'La cuenta no existe');
  8    END IF;
  9
 10    UPDATE CUENTAS
 11    SET SAL_CUE = SAL_CUE + P_VALOR_DEP
 12    WHERE NUM_CUE = P_NUM_CUE;
 13
 14    INSERT INTO TRANSFERENCIAS (NUM_TRA, FEC_HOR_TRA, VALOR_TRA, CUE_ORI, CUE_DES)
 15                        VALUES (SEQ_TRANSFERENCIAS.NEXTVAL,SYSDATE,P_VALOR_DEP, NULL, P_NUM_CUE);
 16  COMMIT;
 17
 18  EXCEPTION
 19    WHEN OTHERS THEN
 20      ROLLBACK;
 21      RAISE_APPLICATION_ERROR(-20012, 'Error inesperado en DEPOSITAR: ' || SQLERRM);
 22  END DEPOSITAR;
 23  .
SQL> /

Procedimiento creado.
```

Justificación ACID:

Atomicidad:

En el procedure DEPOSITAR, todas las acciones que forman parte del depósito (actualizar el saldo y registrar la transferencia) se hacen como una sola unidad. Si ocurre un error en cualquier paso, se ejecuta un ROLLBACK y no se realiza ningún cambio en la base de datos. Así, nunca hay depósitos a medias.

Consistencia:

Antes de cambiar el saldo, se valida que la cuenta existe. Esto protege la integridad de los datos, ya que solo se hacen depósitos en cuentas válidas y los registros siempre quedan correctos.

Aislamiento:

Si varias personas hacen depósitos al mismo tiempo, cada operación se maneja aparte y los cambios solo se ven cuando se hace el COMMIT. Esto evita que los procesos se mezclen y asegura que cada depósito sea independiente.

Durabilidad:

Cuando el depósito se completa y se hace el COMMIT, el cambio queda guardado en la base de datos. Aunque se apague el servidor, el saldo y el registro del depósito no se pierden.

CREACIÓN DEL PROCEDURE RETIRAR:

```
SQL> CREATE OR REPLACE PROCEDURE RETIRAR( P_NUM_CUE VARCHAR, P_VALOR_RET NUMBER, P_ID_CAJ NUMBER)
  2  IS
  3    V_SALDO NUMBER;
  4  BEGIN
  5    SELECT SAL_CUE INTO V_SALDO FROM CUENTAS WHERE NUM_CUE = P_NUM_CUE;
  6    IF V_SALDO < P_VALOR_RET THEN
  7      RAISE_APPLICATION_ERROR(-20001, 'Saldo insuficiente');
  8    END IF;
  9
 10    UPDATE CUENTAS
 11    SET SAL_CUE = SAL_CUE - P_VALOR_RET
 12    WHERE NUM_CUE = P_NUM_CUE;
 13
 14    INSERT INTO TRANSFERENCIAS (NUM_TRA, FEC_HOR_TRA, VALOR_TRA, CUE_ORI, CUE_DES)
 15                        VALUES (SEQ_TRANSFERENCIAS.NEXTVAL,SYSDATE,P_VALOR_RET, P_NUM_CUE,NULL);
 16  COMMIT;
 17
 18  EXCEPTION
 19    WHEN NO_DATA_FOUND THEN
 20      RAISE_APPLICATION_ERROR(-20002, 'La cuenta no existe');
 21    WHEN OTHERS THEN
 22      ROLLBACK;
 23      RAISE_APPLICATION_ERROR(-20010, 'Error en RETIRAR: ' || SQLERRM);
 24  END RETIRAR;
 25  .
SQL> /

Procedimiento creado.
```

Justificación ACID:

Atomicidad:

El procedure RETIRAR se asegura de que, si algo falla, como por ejemplo que no haya suficiente saldo, todo el proceso se cancela y la base de datos queda igual que antes. Ningún paso se ejecuta a medias.

Consistencia:

Siempre se verifica que la cuenta exista y tenga saldo suficiente antes de realizar el retiro. Así se evita que una cuenta quede con saldo negativo o que se retire de una cuenta que no existe**.**

Aislamiento:

Si dos personas intentan retirar al mismo tiempo de la misma cuenta, Oracle maneja cada retiro por separado, la primera que haga COMMIT, su saldo actualizado será visible para los demás, La segunda transacción, al intentar hacer su retiro, puede encontrar el saldo ya cambiado o incluso insuficiente.

Durabilidad:

Cuando el retiro se completa y se confirma con COMMIT, el cambio ya no se puede perder, aunque haya un corte de energía o se reinicie el sistema.

CREACIÓN DEL PROCEDURE TRANSFERIR:

```
SQL> CREATE OR REPLACE PROCEDURE TRANSFERIR(P_NUM_CUE_ORI VARCHAR,P_NUM_CUE_DES VARCHAR,P_VALOR NUMBER)
  2  IS
  3    V_SALDO_ORI NUMBER;
  4    V_EXISTE_DEST NUMBER;
  5  BEGIN
  6
  7    SELECT SAL_CUE INTO V_SALDO_ORI
  8    FROM CUENTAS WHERE NUM_CUE = P_NUM_CUE_ORI;
  9
 10    SELECT COUNT(*) INTO V_EXISTE_DEST FROM CUENTAS WHERE NUM_CUE = P_NUM_CUE_DES;
 11
 12    IF V_EXISTE_DEST = 0 THEN
 13      RAISE_APPLICATION_ERROR(-20003, 'Cuenta destino no existe');
 14    END IF;
 15
 16    IF V_SALDO_ORI < P_VALOR THEN
 17      RAISE_APPLICATION_ERROR(-20004, 'Saldo insuficiente en la cuenta origen');
 18    END IF;
 19
 20  INSERT INTO TRANSFERENCIAS ( NUM_TRA, FEC_HOR_TRA, VALOR_TRA, CUE_ORI, CUE_DES)
 21                      VALUES (SEQ_TRANSFERENCIAS.NEXTVAL, SYSDATE, P_VALOR, P_NUM_CUE_ORI,P_NUM_CUE_DES);
 22
 23    UPDATE CUENTAS
 24    SET SAL_CUE = SAL_CUE - P_VALOR
 25    WHERE NUM_CUE = P_NUM_CUE_ORI;
 26
 27    UPDATE CUENTAS
 28    SET SAL_CUE = SAL_CUE + P_VALOR
 29    WHERE NUM_CUE = P_NUM_CUE_DES;
 30
 31  COMMIT;
 32
 33  EXCEPTION
 34    WHEN NO_DATA_FOUND THEN
 35      RAISE_APPLICATION_ERROR(-20005, 'Cuenta origen no existe');
 36    WHEN OTHERS THEN
 37      ROLLBACK;
 38      RAISE_APPLICATION_ERROR(-20020, 'Error en Transferir: ' || SQLERRM);
 39  END TRANSFERIR;
 40  .
SQL> /

Procedimiento creado.
```

Justificación ACID:

Atomicidad:

En TRANSFERIR, si algún paso falla (por ejemplo, si la cuenta origen no tiene saldo o la cuenta destino no existe), todo se revierte y no se mueve ningún dinero. Así, no hay transferencias incompletas.

Consistencia:

Se valida que las dos cuentas existen y que la cuenta origen tiene saldo suficiente antes de transferir. Así, el dinero siempre “sale de un lado y entra al otro”, y nunca hay errores en los saldos.

Aislamiento:

Si varias transferencias se hacen al mismo tiempo entre cuentas diferentes o iguales, cada una es independiente y los cambios solo se ven al final, cuando todo termina correctamente.

Durabilidad: Cuando se confirma la transferencia con COMMIT, el movimiento de dinero y el registro de la operación quedan guardados en la base de datos para siempre.

PRUEBA DE DEPÓSITO:

Se creo una cuenta y un cliente:

```
SQL> INSERT INTO CLIENTES (ID_CLI, NOM_CLI, APE_CLI, DIR_CLI)
  2  VALUES ('1001', 'Ana', 'Pérez', 'Av. Principal 123');

1 fila creada.

SQL>
SQL> INSERT INTO CUENTAS (NUM_CUE, TIP_CUE, SAL_CUE, ANIO_APE, ID_CLI_PER)
  2  VALUES ('C123', 'AHORROS', 500, 2024, '1001');
```

Verificamos el saldo antes de hacer el depósito:

```
SQL> SELECT NUM_CUE, SAL_CUE FROM CUENTAS WHERE NUM_CUE = 'C123';

NUM_CUE       SAL_CUE
---------- ----------
C123              500
```

Antes de eso modificamos el campo CUE_ORI para que los registros se inserten en NULL:

```
SQL> ALTER TABLE TRANSFERENCIAS
  2  MODIFY (CUE_ORI VARCHAR(20) NULL);

Tabla modificada.
```

Depositamos a esa cuenta $200:

```
SQL> EXECUTE DEPOSITAR('C123', 200, 1);

Procedimiento PL/SQL terminado correctamente.
```

Revisamos si nos llegó el depósito:

```
SQL> SELECT NUM_CUE, SAL_CUE FROM CUENTAS WHERE NUM_CUE = 'C123';

NUM_CUE       SAL_CUE
---------- ----------
C123              700
```

Chekeamos el registro en Transferencias:

```
SQL> SELECT * FROM TRANSFERENCIAS WHERE CUE_DES = 'C123' ORDER BY FEC_HOR_TRA DESC;

NUM_TRA    FEC_HOR_TR  VALOR_TRA CUE_ORI              CUE_DES
---------- ---------- ---------- -------------------- ----------
2          30/06/2025        200                      C123
```

PRUEBA DE RETIRO:

Verificamos el saldo antes del retiro:

```
SQL> SELECT NUM_CUE, SAL_CUE FROM CUENTAS WHERE NUM_CUE = 'C123';

NUM_CUE       SAL_CUE
---------- ----------
C123              700
```

Antes ejecutar el PROCEDURE modificamos el campo CUE_ORI para que los registros se inserten en NULL:

```
SQL> ALTER TABLE TRANSFERENCIAS
  2  MODIFY (CUE_DES VARCHAR(20) NULL);

Tabla modificada.
```

Se realiza el retiro:

```
SQL> EXECUTE RETIRAR('C123', 100, 1);

Procedimiento PL/SQL terminado correctamente.
```

Verifcamos el saldo que le quedó después del retiro:

```
SQL> SELECT NUM_CUE, SAL_CUE FROM CUENTAS WHERE NUM_CUE = 'C123';

NUM_CUE       SAL_CUE
---------- ----------
C123              600
```

Chekamos el registro de retiro en Transferencias:

```
SQL> SELECT NUM_TRA, VALOR_TRA, CUE_ORI, CUE_DES
  2  FROM TRANSFERENCIAS
  3  WHERE CUE_ORI = 'C123'
  4  ORDER BY FEC_HOR_TRA DESC;

NUM_TRA     VALOR_TRA CUE_ORI              CUE_DES
---------- ---------- -------------------- --------------------
4                 100 C123
```

PRUEBA DE TRANSFERENCIA:

Creamos otra cuenta de un cliente:

```
SQL> INSERT INTO CLIENTES (ID_CLI, NOM_CLI, APE_CLI, DIR_CLI)
  2  VALUES ('1002', 'Luis', 'Gómez', 'Calle Secundaria 456');

1 fila creada.

SQL>
SQL> INSERT INTO CUENTAS (NUM_CUE, TIP_CUE, SAL_CUE, ANIO_APE, ID_CLI_PER)
  2  VALUES ('C456', 'AHORROS', 300, 2024, '1002');

1 fila creada.
```

Revisamos saldos en las dos cuentas antes de hacer la transferencia:

```
SQL> SELECT NUM_CUE, SAL_CUE FROM CUENTAS WHERE NUM_CUE IN ('C123', 'C456');

NUM_CUE       SAL_CUE
---------- ----------
C123              600
C456              300
```

Realizamos la transferencia de la cuenta de Ana a la cuenta de Luis:

```
SQL> EXECUTE TRANSFERIR('C123', 'C456', 150);

Procedimiento PL/SQL terminado correctamente.
```

Revisamos los saldos después de transferir en las dos cuentas: Es correcto.

```
SQL> SELECT NUM_CUE, SAL_CUE FROM CUENTAS WHERE NUM_CUE IN ('C123', 'C456');

NUM_CUE       SAL_CUE
---------- ----------
C123              450
C456              450
```

Verificamos el registro en Transferencias: Es correcto.

```
SQL> SELECT NUM_TRA, VALOR_TRA, CUE_ORI, CUE_DES, FEC_HOR_TRA
  2  FROM TRANSFERENCIAS
  3  WHERE CUE_ORI = 'C123' AND CUE_DES = 'C456'
  4  ORDER BY FEC_HOR_TRA DESC;

NUM_TRA     VALOR_TRA CUE_ORI             CUE_DES             FEC_HOR_TRA
---------- ---------- -------------------- -------------------- -------------------
5                 150 C123                 C456                 30/06/2025 18:24:30
```