BANKING SCENARIO

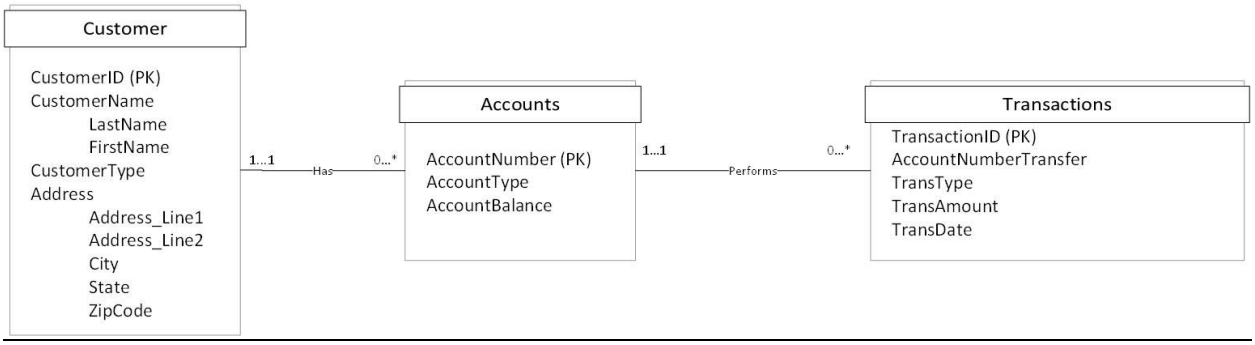
Oracle

DECEMBER 8, 2016

FRANK SMITH

TABLE OF CONTENTS

Bank Scenario



TABLES

In this section, the tables are created.

CUSTOMER TABLES

```
Create Table Customer (
  CustomerID NUMBER GENERATED ALWAYS AS IDENTITY Not Null,
  FirstName Varchar(30) Not Null,
  LastName Varchar(30) Not Null,
  CustomerType Varchar2(14) Not Null,
  Address_line1 Varchar2(60) Not Null,
  Address_line2 Varchar2(60) Null,
  City Varchar2(30) Not Null,
  State Char(2) Not Null,
  Zipcode Number(5) Not Null,
  Constraint C_PK1 Primary Key(CustomerID),
  Constraint CType_Check Check (CustomerType IN ('Large Business', 'Small Business', 'Personal'))
)
```

```
Table CUSTOMER created.
```

ACCOUNTS TABLE

```
Create Table Accounts (
  AccountNumber Number(12) Not Null,
  CustomerID Number Not Null,
  AccountType Varchar2(8) Not Null,
  AccountBalance Number(10,2) Null,
  Constraint A_PK1 Primary Key (AccountNumber),
  Constraint A_FK1 Foreign Key (CustomerID) References Customer(CustomerID),
  Constraint Balance_Check CHECK (AccountBalance>=0),
  Constraint CustomerID_Check CHECK (CustomerID>0),
  Constraint AType_Check Check (AccountType IN ('Savings', 'Checking', 'Business'))
)
```

```
Table ACCOUNTS created.
```

TRANSACTION TABLE

```
Create Table Transaction (
  TransactionID NUMBER GENERATED ALWAYS AS IDENTITY Not Null,
  AccountNumber Number(12) Not Null,
  AccountNumberTransfer Number(12) Null,
  TransType Varchar2(10) Not Null,
  TransAmount Number(10,2) Not Null,
  TransDate Date Not Null,
  Constraint T_PK1 Primary Key (TransactionID),
  Constraint T_FK1 Foreign Key (AccountNumber) References Accounts(AccountNumber),
  Constraint T_FK2 Foreign Key (AccountNumberTransfer) References Accounts(AccountNumber),
  Check (TransType IN ('Deposit', 'Transfer', 'Withdraw', 'Creation')),
  Constraint Correct_Values Check (TransType = 'Deposit' AND TransAmount>0 OR
  TransType = 'Withdraw' AND TransAmount<0 OR
  TransType = 'Transfer' AND AccountNumberTransfer is not null AND TransAmount !=0 OR
  TransType = 'Creation' AND TransAmount=0)
)
```

<u>I added a creation date for all new accounts.</u>

```
Table TRANSACTION created.
```

REQUIRED TRIGGERS, SEQUENCES, AND PROCEDURES

ACCOUNT CREATION DATE

```
Create or Replace Procedure Create_Tran_Accounts (u_Account Transaction.TransactionID%TYPE)

IS

BEGIN

INSERT INTO Transaction (AccountNumber, AccountNumberTransfer, TransType, TransAmount,
TransDate) VALUES (u_Account, '', 'Creation','0.00' ,SYSDATE);

END Create_Tran_Accounts;
```

```
Procedure CREATE_TRAN_ACCOUNTS compiled
```

TRANSACTION: NO UPDATING ACCOUNT NUMBERS

```
Create or Replace Trigger Trigger_Trans_No_Updating
Before Update on Transaction
For Each Row
Begin
  If Updating ('AccountNumber') THEN
    Raise_Application_Error(-4000, 'Error: Changing the Account Number is not allowed.');
  End If;
End Trigger_Trans_No_Updating;
```

```
Trigger TRIGGER_TRANS_NO_UPDATING compiled
```

ACCOUNTS: NO UPDATING ACCOUNT NUMBERS

```
Create or Replace Trigger Trigger_Acc_No_Updating
Before Update on Accounts
For Each Row
Begin
  If Updating ('AccountNumber') THEN
    Raise_Application_Error(-4000, 'Error: Changing the Account Number is not allowed.');
  End If;
End Trigger_Acc_No_Updating;
```

```
Trigger TRIGGER_ACC_NO_UPDATING compiled
```

ACCOUNTS SEQUENCE

```
CREATE SEQUENCE seq_A
INCREMENT BY 1
START WITH 100000000001
maxvalue 999999999999;
```

```
Sequence SEQ_A created.
```

PROCEDURES

PROCEDURE 1: ADD NEW CUSTOMERS

(Using a procedure) System should be able to add new customers ensuring that no duplicates are entered, and type is valid. A message should be returned with the success or failure of the action.

PROCEDURE 1

```
Create or Replace Procedure Create_Customer (u_Fname Customer.FirstName%TYPE, u_Lname
Customer.LastName%TYPE,u_CustomerType Customer.CustomerType%TYPE, u_Address_Line1
Customer.Address_Line1%TYPE, u_Address_Line2 Customer.Address_Line2%TYPE, u_City
Customer.City%TYPE, u_State Customer.State%TYPE, u_Zipcode Customer.Zipcode%TYPE)

IS

  Cust_Count INT;

 Status int := 0; Valid_Update EXCEPTION; Invalid_UpdateMore EXCEPTION; Invalid_UpdateMissing
EXCEPTION; Invalid_UpdateWrong_Type EXCEPTION;

BEGIN

   If (u_Fname IS NOT NULL) AND (u_Lname IS NOT NULL) AND (u_Address_Line1 IS NOT NULL) AND
(u_City IS NOT NULL) AND (u_State IS NOT NULL) AND (u_Zipcode IS NOT NULL) Then

     BEGIN

       select count(*)

       into Cust_Count

       from Customer

       Where u_Fname = FIRSTNAME AND u_Lname = LASTNAME AND Upper(u_CustomerType) =
Upper(CUSTOMERTYPE) AND u_Address_Line1 = ADDRESS_LINE1 AND u_City = CITY AND Upper(u_State)
= Upper(STATE) AND u_Zipcode = ZIPCODE;
```

```
if (Cust_Count < 1 ) THEN
  BEGIN
    If (Upper(u_CustomerType) = Upper('Personal')) THEN
      BEGIN
        INSERT INTO Customer (FirstName, LastName, CustomerType, Address_line1, Address_line2,
City, State, Zipcode) VALUES (u_Fname, u_Lname, 'Personal', u_Address_Line1, u_Address_Line2, u_City,
Upper(u_State), u_Zipcode);
        COMMIT;
        RAISE Valid_Update;
      END;
    ELSIF (Upper(u_CustomerType) = Upper('Small Business')) THEN
      BEGIN
        INSERT INTO Customer (FirstName, LastName, CustomerType, Address_line1, Address_line2,
City, State, Zipcode) VALUES (u_Fname, u_Lname, 'Small Business', u_Address_Line1, u_Address_Line2,
u_City, Upper(u_State), u_Zipcode);
        COMMIT;
        RAISE Valid_Update;
      END;
    ELSIF (Upper(u_CustomerType) = Upper('Large Business')) THEN
      BEGIN
        INSERT INTO Customer (FirstName, LastName, CustomerType, Address_line1, Address_line2,
City, State, Zipcode) VALUES (u_Fname, u_Lname, 'Large Business', u_Address_Line1, u_Address_Line2,
u_City, Upper(u_State), u_Zipcode);
        COMMIT;
        RAISE Valid_Update;
      END;
    ELSE
      BEGIN
        RAISE Invalid_UpdateWrong_Type;
      END;
    END IF;
```

```
          END;
       ELSE
         BEGIN
          RAISE Invalid_UpdateMore;
         END;
       END IF;
     END;
  ELSE
    BEGIN
     RAISE Invalid_UpdateMissing;
    END;
  END IF;



Exception


WHEN Valid_Update THEN
                    Status := 1;
    DBMS_OUTPUT.PUT_LINE('Success ' || Status || ': Customer Created');


WHEN Invalid_UpdateMore THEN
                    Status := 2;
    DBMS_OUTPUT.PUT_LINE('Error ' || Status || ': Duplicate User');


WHEN Invalid_UpdateMissing THEN
                    Status := 3;
    DBMS_OUTPUT.PUT_LINE('Error ' || Status || ': Missing Data');


WHEN Invalid_UpdateWrong_Type THEN
                    Status := 4;
```

```
    DBMS_OUTPUT.PUT_LINE('Error ' || Status || ': Wrong Type Entered.');

When OTHERS THEN
  Rollback;

  DBMS_OUTPUT.PUT_LINE( 'An error has occurred on: ' || SYSDATE || Status);

END Create_Customer;
```

```
Procedure CREATE_CUSTOMER compiled
```

PROCEDURE 1: INSERT

```
BEGIN
  Create_Customer('Patty' ,'May' , 'large Business','123 Fake Street' ,'' ,'Baltimore' ,'MD' ,'21117');
  Create_Customer('Frank' ,'Smith' , 'Small Business','555 Fake Street' ,'' ,'Towson' ,'md' ,'21286');
  Create_Customer('Mary' ,'Reed' , 'Personal','212 Fake Street' ,'' ,'Anchorage' ,'AK' ,'99507');
  Create_Customer('John' ,'Smith' , 'Personal','999 Fake Street' ,'' ,'Scranton' ,'PA' ,'18504');
  Create_Customer('John' ,'Duplicate' , 'Personal','999 Fake Street' ,'' ,'Scranton' ,'PA' ,'18504');
  Create_Customer('John' ,'Duplicate' , 'Personal','999 Fake Street' ,'' ,'Scranton' ,'PA' ,'18504');
  Create_Customer('John' ,'Blaylock' , 'Bad Type','999 Fake Street' ,'' ,'Scranton' ,'PA' ,'18504');
  Create_Customer('John' ,'Blaylock' , 'Personal','999 Fake Street' ,'' ,'Scranton' ,'PA' ,'18504');
  Create_Customer('Missing' ,'Value' , 'Personal','' ,'' ,'Scranton' ,'PA' ,'18504');
END;
```

Working and Non-Working Examples

```
Task completed in 0.269 seconds
PL/SQL procedure successfully completed.
Dbms Output
Buffer Size: 20000
AIT-732
Success 1: Customer Created
Success 1: Customer Created
Success 1: Customer Created
Success 1: Customer Created
Success 1: Customer Created
Error 2: Duplicate User
Error 4: Wrong Type Entered.
Success 1: Customer Created
Error 3: Missing Data
```

AIT-732 | CUSTOMER

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Sort.. | Filter:

	CUSTOMERID	FIRSTNAME	LASTNAME	CUSTOMERTYPE	ADDRESS_LINE1	ADDRESS_LINE2	CITY	STATE	ZIPCODE
1	1	Patty	May	Large Business	123 Fake Street	(null)	Baltimore	MD	21117
2	2	Frank	Smith	Small Business	555 Fake Street	(null)	Towson	MD	21286
3	3	Mary	Reed	Personal	212 Fake Street	(null)	Anchorage	AK	99507
4	4	John	Smith	Personal	999 Fake Street	(null)	Scranton	PA	18504
5	5	John	Duplicate	Personal	999 Fake Street	(null)	Scranton	PA	18504
6	6	John	Blaylock	Personal	999 Fake Street	(null)	Scranton	PA	18504

PROCEDURE 2: ADD NEW ACCOUNTS

(Using a procedure) System should be able to add a new account for an existing customer. Given a customer ID, and an account type, the system should verify that the customer id, and account type are valid, then create a new account. A message should be returned with the success or failure of the action.

PROCEDURE 2

```
Create or Replace Procedure Create_Accounts (u_ID Customer.CustomerID%TYPE, u_Account_Type
Accounts.AccountType%TYPE)

IS

--USED FOR ERROR CHECKING
Status int := 0; valid_AccountCreated EXCEPTION; Invalid_Customer EXCEPTION; Invalid_AccountType
EXCEPTION;

u_Count int;

BEGIN

Select Count(*) INTO u_Count
From Customer
Where u_ID = CustomerID;

IF (u_Count = 1) THEN
  BEGIN
   IF ((u_Account_Type = 'Savings') OR (u_Account_Type = 'Checking') OR (u_Account_Type = 'Business'))
THEN
     BEGIN
       Insert into Accounts (AccountNumber, CustomerID, AccountType, AccountBalance) values
(seq_A.nextval, u_ID , u_Account_Type , '0.00');
       BEGIN
       Create_Tran_Accounts(seq_A.currval);
```

```
      END;
      Commit;
      RAISE valid_AccountCreated;
    END;
  ELSE
    BEGIN
      RAISE Invalid_AccountType;
    END;
  END IF;
 END;
ELSE
 RAISE Invalid_Customer;
END IF;

Commit;

Exception

WHEN valid_AccountCreated THEN
                    Status := 1;
   DBMS_OUTPUT.PUT_LINE('Success ' || Status || ': Account Created');

WHEN NO_DATA_FOUND or TOO_MANY_ROWS THEN
                    Status := 2;
   DBMS_OUTPUT.PUT_LINE('Error ' || Status || ': Invalid Data Entered');

WHEN Invalid_Customer THEN
                    Status := 3;
   DBMS_OUTPUT.PUT_LINE('Error ' || Status || ': Customer ID is not Valid');
```

```
WHEN Invalid_AccountType THEN
                    Status := 4;
   DBMS_OUTPUT.PUT_LINE('Error ' || Status || ': Account Type is not Valid');

When OTHERS THEN
  Rollback;

  DBMS_OUTPUT.PUT_LINE('Error ' || Status || ': Wrong Type Entered.');

END Create_Accounts;
```

```
Procedure CREATE_ACCOUNTS compiled
```

PROCEDURE 2: INSERT

```
LOCK TABLE Accounts IN SHARE MODE NOWAIT;

BEGIN

  Create_Accounts('1' ,'Savings');

  Create_Accounts('1' ,'Checking');

  Create_Accounts('2' ,'Business');

  Create_Accounts('3' ,'Checking');

  Create_Accounts('4' ,'Savings');

  Create_Accounts('5' ,'Checking');

  Create_Accounts('70' ,'Savings');

  Create_Accounts('2' ,'Bad Type');

  Create_Accounts('2' ,'');

END;
```

Working and Non-Working Examples

```
Task completed in 0.076 seconds
Lock succeeded.

PL/SQL procedure successfully completed.
```

```
Dbms Output
Buffer Size: 20000
AIT-732
Success 1: Account Created
Success 1: Account Created
Success 1: Account Created
Success 1: Account Created
Success 1: Account Created
Success 1: Account Created
Error 3: Customer ID is not Valid
Error 4: Account Type is not Valid
Error 4: Account Type is not Valid
```

Start Page | AIT-732 | ACCOUNTS

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependen

Sort.. | Filter:

	ACCOUNTNUMBER	CUSTOMERID	ACCOUNTTYPE	ACCOUNTBALANCE
1	100000000001	1	Savings	0
2	100000000002	1	Checking	0
3	100000000003	2	Business	0
4	100000000004	3	Checking	0
5	100000000005	4	Savings	0
6	100000000006	5	Checking	0

I added "Creation" dates to the transaction table every time a new account is created.

AIT-732 | TRANSACTION

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Sort.. | Filter:

	TRANSACTIONID	ACCOUNTNUMBER	ACCOUNTNUMBERTRANSFER	TRANSTYPE	TRANSAMOUNT	TRANSDATE
1	1	100000000001	(null)	Creation	0	05-DEC-16
2	2	100000000002	(null)	Creation	0	05-DEC-16
3	3	100000000003	(null)	Creation	0	05-DEC-16
4	4	100000000004	(null)	Creation	0	05-DEC-16
5	5	100000000005	(null)	Creation	0	05-DEC-16
6	6	100000000006	(null)	Creation	0	05-DEC-16

PROCEDURE 3: TRANSACTIONS

(Using a procedure and trigger) System should be able to handle valid transactions on valid accounts for valid customers maintaining the integrity of the account balances. Specifically, the system should allow deposits to accounts (must have account number, amount), withdrawals (must have account number, amount) from accounts and transfers between accounts (must have 2 account numbers and an amount). When these transactions are made, the account balance must be updated (via trigger). No account balance may go below zero. A message should be returned with the success or failure of the action.

PROCEDURE 3

```
Create or Replace Procedure Add_Transaction (u_AccountNumber Transaction.AccountNumber%TYPE,
u_AccountNumberTransfer Transaction.AccountNumberTransfer%TYPE, u_AccountType
Transaction.TransType%TYPE, u_Amount Transaction.TransAmount%TYPE)

IS

--USED FOR ERROR CHECKING

Status int := 0; Invalid_Transaction EXCEPTION; Valid_TransactionCreatedD EXCEPTION;
Valid_TransactionCreatedW EXCEPTION; Valid_TransactionCreatedT EXCEPTION;
Invalid_TransactionFailed EXCEPTION; Invalid_AccountNumber EXCEPTION;

ValidAcc INT;

BEGIN

Select Count(AccountNumber) into ValidAcc

From Accounts

Where u_AccountNumber = ACCOUNTNUMBER;

IF (ValidAcc = 1) THEN

  IF (Upper(u_AccountType) = Upper('Deposit')) THEN

    BEGIN

      IF (u_AccountNumber IS NOT NULL) AND (u_Amount > 0) THEN
```

```
      BEGIN
        INSERT INTO Transaction (AccountNumber, AccountNumberTransfer,TransType, TransAmount,
TransDate) VALUES (u_AccountNumber, '', 'Deposit', u_Amount, SYSDATE);
        Commit;
        RAISE Valid_TransactionCreatedD;
      END;
    ELSE
      BEGIN
        RAISE Invalid_TransactionFailed;
      END;
    END IF;
  END;
ELSIF (Upper(u_AccountType) = Upper('Withdraw')) THEN
  BEGIN
    IF (u_AccountNumber IS NOT NULL) AND (u_Amount < 0) THEN
      BEGIN
        INSERT INTO Transaction (AccountNumber, AccountNumberTransfer,TransType, TransAmount,
TransDate) VALUES (u_AccountNumber, '', 'Withdraw', u_Amount, SYSDATE);
        Commit;
        RAISE Valid_TransactionCreatedW;
      END;
    ELSE
      BEGIN
        RAISE Invalid_TransactionFailed;
      END;
    END IF;
  END;
ELSIF (Upper(u_AccountType) = Upper('Transfer')) THEN
  BEGIN
    IF (u_AccountNumber IS NOT NULL) AND (u_Amount < 0) AND (u_AccountNumberTransfer IS NOT
NULL) THEN
```

```
      BEGIN
        INSERT INTO Transaction (AccountNumber, AccountNumberTransfer,TransType, TransAmount,
TransDate) VALUES (u_AccountNumber, u_AccountNumberTransfer, 'Transfer', u_Amount, SYSDATE);
        INSERT INTO Transaction (AccountNumber, AccountNumberTransfer,TransType, TransAmount,
TransDate) VALUES (u_AccountNumberTransfer, u_AccountNumber, 'Transfer', -(u_Amount), SYSDATE);
        Commit;
        RAISE Valid_TransactionCreatedT;
      END;
    ELSE
      BEGIN
        RAISE Invalid_TransactionFailed;
      END;
    END IF;
  END;
 ELSE
  BEGIN
    RAISE Invalid_Transaction;
  END;
 END IF;
ELSE
 BEGIN
  RAISE Invalid_AccountNumber;
 END;
END IF;



Exception


WHEN Valid_TransactionCreatedD THEN
                    Status := 1;
   DBMS_OUTPUT.PUT_LINE('Success ' || Status || ': Deposit Transaction Created');
```

```
WHEN Valid_TransactionCreatedW THEN
    Status := 2;
            DBMS_OUTPUT.PUT_LINE('Success ' || Status || ': Withdraw Transaction Created');

WHEN Valid_TransactionCreatedT THEN
                        Status := 3;
    DBMS_OUTPUT.PUT_LINE('Success ' || Status || ': Transfer Transaction Created');

WHEN NO_DATA_FOUND or TOO_MANY_ROWS THEN
                        Status := 4;
    DBMS_OUTPUT.PUT_LINE('Error ' || Status || ': Invalid Data Entered');

WHEN Invalid_TransactionFailed THEN
                        Status := 5;
    DBMS_OUTPUT.PUT_LINE('Error ' || Status || ': Missing Data');

WHEN Invalid_Transaction THEN
                        Status := 6;
    DBMS_OUTPUT.PUT_LINE('Error ' || Status || ': Invalid Type Entered');

WHEN Invalid_AccountNumber THEN
                        Status := 7;
    DBMS_OUTPUT.PUT_LINE('Error ' || Status || ': Invalid Account Number Entered');

When OTHERS THEN
  Rollback;

DBMS_OUTPUT.PUT_LINE( 'An error has occurred on: ' || SYSDATE || Status);
```

END Add_Transaction;

```
Procedure ADD_TRANSACTION compiled
```

PROCEDURE 3: TRIGGER

```
create or replace trigger Trigger_Account_Balance
 After insert or update or delete on Transaction
 For Each Row

BEGIN

UPDATE Accounts SET AccountBalance = (AccountBalance + :New.TransAmount)
WHERE AccountNumber = :New.AccountNumber;
END Trigger_Account_Balance;
```

```
Trigger TRIGGER_ACCOUNT_BALANCE compiled
```

PROCEDURE 3: INSERT

```
BEGIN
  Add_Transaction('100000000001', '', 'deposit', '10000');
  Add_Transaction('100000000001', '', 'WITHDRAW', '-5000');
  Add_Transaction('100000000002', '', 'Withdraw', '-5000');
  Add_Transaction('100000000001', '100000000002', 'Transfer', '-5000');
  Add_Transaction('100000000002', '', 'Deposit', '10000');
  Add_Transaction('100000000003', '', 'Deposit', '10000');
  Add_Transaction('100000000004', '', 'Deposit', '10000');
  Add_Transaction('100000000005', '', 'Deposit', '10000');
  Add_Transaction('100000000001', '', 'Deposit', '1333');
  Add_Transaction('100000000090', '', 'Deposit', '1333');
END;
```

Working and Non-Working Examples

Task completed in 0.084 seconds

```
PL/SQL procedure successfully completed
```

Dbms Output

Buffer Size: 20000

AIT-732

```
Success 1: Deposit Transaction Created
Success 2: Withdraw Transaction Created
An error has occurred on: 05-DEC-160
Success 3: Transfer Transaction Created
Success 1: Deposit Transaction Created
Success 1: Deposit Transaction Created
Success 1: Deposit Transaction Created
Success 1: Deposit Transaction Created
Success 1: Deposit Transaction Created
Error 7: Invalid Account Number Entered
```

	ACCOUNTNUMBER	CUSTOMERID	ACCOUNTTYPE	ACCOUNTBALANCE
1	100000000001	1	Savings	1333
2	100000000002	1	Checking	15000
3	100000000003	2	Business	10000
4	100000000004	3	Checking	10000
5	100000000005	4	Savings	10000
6	100000000006	5	Checking	0

AIT-732 | TRANSACTION

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Sort.. | Filter:

	TRANSACTIONID	ACCOUNTNUMBER	ACCOUNTNUMBERTRANSFER	TRANSTYPE	TRANSAMOUNT	TRANSDATE
1	1	100000000001	(null)	Creation	0	05-DEC-16
2	2	100000000002	(null)	Creation	0	05-DEC-16
3	3	100000000003	(null)	Creation	0	05-DEC-16
4	4	100000000004	(null)	Creation	0	05-DEC-16
5	5	100000000005	(null)	Creation	0	05-DEC-16
6	6	100000000006	(null)	Creation	0	05-DEC-16
7	7	100000000001	(null)	Deposit	10000	05-DEC-16
8	8	100000000001	(null)	Withdraw	-5000	05-DEC-16
9	10	100000000001	100000000002	Transfer	-5000	05-DEC-16
10	11	100000000002	100000000001	Transfer	5000	05-DEC-16
11	12	100000000002	(null)	Deposit	10000	05-DEC-16
12	13	100000000003	(null)	Deposit	10000	05-DEC-16
13	14	100000000004	(null)	Deposit	10000	05-DEC-16
14	15	100000000005	(null)	Deposit	10000	05-DEC-16
15	16	100000000001	(null)	Deposit	1333	05-DEC-16

PROCEDURE 4: STATE BALANCE

(Using a procedure) Given a state, display the sum of the balances by state

PROCEDURE 4

```
Create or Replace Procedure Balance_Per_State (u_State Customer.State%TYPE)

IS

--USED FOR ERROR CHECKING
Status int := 0; NO_Data EXCEPTION; Valid_Data EXCEPTION;

u_Total Number(10,2);

BEGIN

IF (u_State IS NOT NULL) THEN
  BEGIN
    SELECT SUM(AccountBalance) into u_Total
    from Customer, Accounts
    WHERE Customer.CustomerID = Accounts.CustomerID AND Upper(Customer.State) = Upper(u_State);
      BEGIN
        RAISE Valid_Data;
      END;
      Commit;
  END;
ELSE
  BEGIN
    RAISE NO_Data;
  END;
```

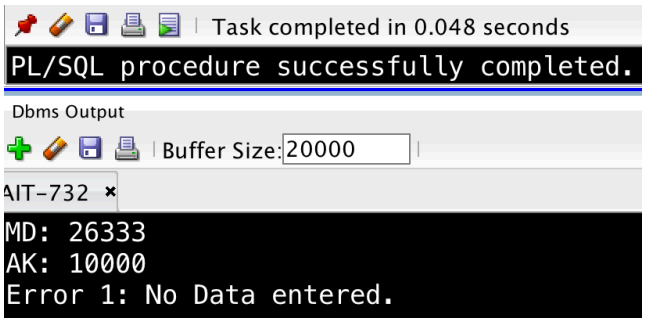
```
END IF;

Exception

WHEN NO_DATA THEN
                    Status := 1;
   DBMS_OUTPUT.PUT_LINE('Error ' || Status || ': No Data entered.');

WHEN Valid_Data THEN
                    Status := 2;
   DBMS_OUTPUT.PUT_LINE(TO_CHAR(Upper(u_State)) || ': ' || u_Total);

When OTHERS THEN
  Rollback;

  DBMS_OUTPUT.PUT_LINE( 'An Error has occured On: ' || SYSDATE);

END Balance_Per_State;
```

```
Procedure BALANCE_PER_STATE compiled
```

PROCEDURE 4: INPUT

```
BEGIN
Balance_Per_State ('md');
Balance_Per_State ('AK');
Balance_Per_State ('');
END;
```

Working and Non-Working Examples



PROCEDURE 5: CHANGE OF ADDRESS

(Using a procedure) Given a customer id and a new address, update the customer's address. A message should be returned with the success or failure of the action.

PROCEDURE 5

```
Create or Replace Procedure Update_Customer (u_ID Customer.CustomerID%TYPE, u_Address_Line1
Customer.Address_Line1%TYPE, u_Address_Line2 Customer.Address_Line2%TYPE, u_City
Customer.City%TYPE, u_State Customer.State%TYPE, u_Zipcode Customer.Zipcode%TYPE)

IS

--USED FOR ERROR CHECKING
Status int := 0; Invalid_Update EXCEPTION; Invalid_Customer EXCEPTION; Invalid_AllSame EXCEPTION;
```

```
u_Count int;
c_Add1 Varchar2(60);
c_Add2 Varchar2(60);
c_City Varchar(30);
c_State Char(2);
c_Zip Number(5);
c_Count INT;
BEGIN

Select Count(*) INTO u_Count
From Customer
Where u_ID = CustomerID;

Select ADDRESS_LINE1, ADDRESS_LINE2, CITY, STATE, ZIPCODE INTO c_Add1, c_Add2, c_City, c_State,
c_Zip
From Customer
Where u_ID = CustomerID;

c_Count := 0;

If (u_Count = 1) THEN
  BEGIN
    If (Upper(u_Address_Line1) <> Upper(c_Add1)) Then
      c_Count := c_Count + 1;
      BEGIN
      Update Customer set Address_Line1 = u_Address_Line1 where u_ID = CustomerID;
      DBMS_OUTPUT.PUT_LINE('UPDATED: Address 1');
      END;
    End if;
```

```
  If ((Upper(u_Address_Line2) <> Upper(c_Add2)) OR (u_Address_Line2 IS NOT NULL AND c_Add2 IS
NULL)) Then
    c_Count := c_Count + 1;
    BEGIN
    Update Customer set Address_Line2 = u_Address_Line2 where u_ID = CustomerID;
    DBMS_OUTPUT.PUT_LINE('UPDATED: Address 2');
    END;
  End if;

  If (Upper(u_City) <> Upper(c_City)) Then
    c_Count := c_Count + 1;
    Update Customer set City = u_City where u_ID = CustomerID;
    DBMS_OUTPUT.PUT_LINE('UPDATED: City');
  End if;

  If (Upper(u_State) <> Upper(c_State)) Then
    c_Count := c_Count + 1;
    Update Customer set State = u_State where u_ID = CustomerID;
    DBMS_OUTPUT.PUT_LINE('UPDATED: State');
  End if;

  If (u_Zipcode <> c_Zip) Then
    c_Count := c_Count + 1;
    Update Customer set Zipcode = u_Zipcode where u_ID = CustomerID;
    DBMS_OUTPUT.PUT_LINE('UPDATED: Zipcode');
  End if;

  If (u_Address_Line1 IS NULL) AND (u_Address_Line2 IS NULL) AND (u_City IS NULL) AND (u_State IS
NULL) AND (u_Zipcode IS NULL) Then
    c_Count := c_Count + 1;
```

```
      RAISE Invalid_Update;
    END IF;

    If (c_Count = 0) THEN
      RAISE Invalid_AllSame;
    END IF;

    Commit;
  END;
ELSE
  BEGIN
    RAISE Invalid_Customer;
  END;
END IF;



Exception

WHEN NO_DATA_FOUND or TOO_MANY_ROWS THEN
                        Status := 1;
     DBMS_OUTPUT.PUT_LINE('Error ' || Status || ': No Data entered, or to many rows.');

WHEN Invalid_Update THEN
                        Status := 2;
     DBMS_OUTPUT.PUT_LINE('Error ' || Status || ': No Data Entered');

WHEN Invalid_Customer THEN
                        Status := 3;
     DBMS_OUTPUT.PUT_LINE('Error ' || Status || ': Wrong Customer ID');
```

```
WHEN Invalid_AllSame THEN
                    Status := 4;
   DBMS_OUTPUT.PUT_LINE('Error ' || Status || ': No update, all information is the same.');



When OTHERS THEN
  Rollback;


  DBMS_OUTPUT.PUT_LINE( 'An Error has occurred on: ' || SYSDATE);


END Update_Customer;
```

```
Procedure UPDATE_CUSTOMER compiled
```

INSERT PROCEDURE 5

```
BEGIN
  Update_Customer('1' , '555 Coding Way' ,'Unit G' ,'Towson' ,'MD' ,'21286');
  Update_Customer('1' , '' ,'' ,'' ,'' ,'');
  Update_Customer('' , '555 Coding Way' ,'Unit G' ,'Towson' ,'MD' ,'21286');
END;
```

Working and Non-Working Examples

```
PL/SQL procedure successfully completed.
```

Dbms Output

Buffer Size: 20000

AIT-732

```
UPDATED: Address 1
UPDATED: Address 2
UPDATED: City
UPDATED: Zipcode
Error 2: No Data Entered
Error 1: No Data entered, or to many rows.
```

BEFORE

Start Page | AIT-732 | CUSTOMER

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Sort.. | Filter:

	CUSTOMERID	FIRSTNAME	LASTNAME	CUSTOMERTYPE	ADDRESS_LINE1	ADDRESS_LINE2	CITY	STATE	ZIPCODE
1	1	Patty	May	Large Business	123 Fake Street	(null)	Baltimore	MD	21117

AFTER

AIT-732 | CUSTOMER

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Sort.. | Filter:

	CUSTOMERID	FIRSTNAME	LASTNAME	CUSTOMERTYPE	ADDRESS_LINE1	ADDRESS_LINE2	CITY	STATE	ZIPCODE
1	1	Patty	May	Large Business	555 Coding Way	Unit G	Towson	MD	21286
2	2	Frank	Smith	Small Business	555 Fake Street	(null)	Towson	MD	21286
3	3	Mary	Reed	Personal	212 Fake Street	(null)	Anchorage	AK	99507
4	4	John	Smith	Personal	999 Fake Street	(null)	Scranton	PA	18504
5	5	John	Duplicate	Personal	999 Fake Street	(null)	Scranton	PA	18504
6	6	John	Blaylock	Personal	999 Fake Street	(null)	Scranton	PA	18504

PROCEDURE 6: AMOUNT GREATER THAN WHAT IS ENTERED

(Using a procedure) Given an amount, display the account number, last name, and amount of all transactions with an amount greater than the one passed into the procedure.

PROCEDURE 6

```
Create or Replace Procedure Amount_Greater_Than (u_Amount Transaction.TransAmount%TYPE)

AS

--USED FOR ERROR CHECKING
Status int := 0; Valid_Data EXCEPTION; Invalid_Data EXCEPTION;

rc sys_refcursor;
Acc_Number Number(12);
Trans_Number Number(12);
Cust_LName Varchar(30);
Trans_TranAmount Number(10,2);
u_Count NUMBER;
L_Counter NUMBER;

BEGIN

select count(*)
      into u_Count
      from Transaction
      Where TransAmount > u_Amount;

If (u_Amount IS NOT NULL) THEN
  BEGIN
   OPEN rc for SELECT Accounts.AccountNumber, Transaction.AccountNumber, LastName,
transaction.TransAmount INTO Acc_Number, Trans_Number, Cust_LName, Trans_TranAmount
```

```
   from Customer, Accounts, Transaction
   WHERE Customer.CustomerID = Accounts.CustomerID AND Accounts.AccountNumber =
Transaction.AccountNumber AND Transaction.TransAmount > u_Amount;
     BEGIN
       dbms_sql.return_result(rc);
     Commit;
     RAISE Valid_Data;
     END;
  END;
ELSE
  BEGIN
   RAISE Invalid_Data;
  END;
END IF;

Exception

WHEN Valid_Data THEN
                    Status := 1;
    DBMS_OUTPUT.PUT_LINE('Success ' || Status || ': Valid information entered.');

WHEN Invalid_Data THEN
                    Status := 2;
    DBMS_OUTPUT.PUT_LINE('Error ' || Status || ': No data entered.');

When OTHERS THEN
  Rollback;

  DBMS_OUTPUT.PUT_LINE( 'An error has occured on: ' || SYSDATE);
```

END Amount_Greater_Than;

```
Procedure AMOUNT_GREATER_THAN compiled
```

PROCEDURE 6: INSERT

BEGIN

Amount_Greater_Than ('100');

END;

```
ResultSet #1


ACCOUNTNUMBER ACCOUNTNUMBER LASTNAME                         TRANSAMOUNT
------------- ------------- ------------------------------ ------------
100000000001  100000000001  May                            10000
100000000002  100000000002  May                            5000
100000000002  100000000002  May                            10000
100000000003  100000000003  Smith                          10000
100000000004  100000000004  Reed                           10000
100000000005  100000000005  Smith                          10000
100000000001  100000000001  May                            1333

 7 rows selected


PL/SQL procedure successfully completed.
```

```
Dbms Output
Buffer Size: 20000
IT-732
Success 1: Valid information entered.
```

PROCEDURE 7: EXTRA CREDIT

[EXTRA CREDIT]: Write a trigger so that anytime a customer changes their address, the old address, new address, date, customer id and user ID are stored in a log.

CREATE LOG TABLE

```
Create Table Customer_Log (
  CustomerID INT Not Null,
  UserID Varchar2(10) Not Null,
  Change_Date DATE Not Null,
  NEW_Address_line1 Varchar2(60) Not Null,
  NEW_Address_line2 Varchar2(60) Null,
  NEW_City Varchar2(30) Not Null,
  NEW_State Char(2) Not Null,
  NEW_Zipcode Number(5) Not Null,
  OLD_Address_line1 Varchar2(60) Not Null,
  OLD_Address_line2 Varchar2(60) Null,
  OLD_City Varchar2(30) Not Null,
  OLD_State Char(2) Not Null,
  OLD_Zipcode Number(5) Not Null,
  Constraint Log_FK1 Foreign Key (CustomerID) References Customer(CustomerID)
)
```

```
Table CUSTOMER_LOG created.
```

YOU MAY OR MAY NOT NEED TO ALTER THE SESSION

Only if when adding the trigger, you get the following error:

ORA-00603: ORACLE server session terminated by fatal error

ORA-00600: internal error code, arguments: [kqlidchg0], [], [], [], [], [], [], [], [], [], [], []

ORA-00604: error occurred at recursive SQL level 1

ORA-00001: unique constraint (SYS.I_PLSCOPE_SIG_IDENTIFIER$) violated00603. 00000 - "ORACLE server session terminated by fatal error"*Cause: An Oracle server session was in an unrecoverable state.*Action: Log in to Oracle again so a new server session will be created automatically. Examine the session trace file for more information.

```
ALTER SESSION SET PLSCOPE_SETTINGS = 'IDENTIFIERS:NONE';
```

PROCEDURE 7: TRIGGER

```
create or replace trigger Trigger_CustomerUpdate

 After update on Customer

 For Each Row


DECLARE

  v_Username varchar2(10);

  n_Add1 Varchar2(60);

  n_Add2 Varchar2(60);

  n_City Varchar2(30);

  n_State Char(2);

  n_Zip Number(5);

  o_Add1 Varchar2(60);

  o_Add2 Varchar2(60);

  o_City Varchar2(30);

  o_State Char(2);

  o_Zip Number(5);
```

```
BEGIN

  -- Find username of person performing the DELETE on the table
  SELECT USER INTO v_Username
  FROM dual;

n_Add1 := :new.Address_Line1;
n_Add2 := :new.Address_Line2;
n_City := :new.City;
n_State := :new.State;
n_Zip := :new.Zipcode;
o_Add1 := :old.Address_Line1;
o_Add2 := :old.Address_Line2;
o_City := :old.City;
o_State := :old.State;
o_Zip := :old.Zipcode;

BEGIN
  Insert INTO Customer_Log (CustomerID, UserID, Change_Date, New_Address_Line1,
New_Address_Line2, New_City, New_State, New_Zipcode, Old_Address_Line1, Old_Address_Line2,
Old_City, Old_State, Old_Zipcode)
  VALUES (:new.CustomerID, v_Username, SYSDATE, n_Add1, n_Add2, n_City, n_State, n_Zip, o_Add1,
o_Add2, :old.City, o_State, o_Zip);
END;

END Trigger_CustomerUpdate;
```

```
Session altered.

Trigger TRIGGER_CUSTOMERUPDATE compiled
```

PROCEDURE 7: TEST INSERT

Instead of rewriting Procedure 5, which would require more debugging, please use the simple update which will place a single row on the “Customer_Log” table. If you use “Procedure 5”, with the way it was designed to show what was updated, it will insert multiple rows.

Simple Update:

```
BEGIN

  UPDATE Customer SET Address_Line1 = '111 SQL Drive', Address_Line2 = 'APT A', City = 'Long Island',
State = 'NY', Zipcode = '11101'

  WHERE CustomerID = '1';

END;
```

Insert That Writes Multiple Rows

```
BEGIN

   Update_Customer('1' , '111 SQL Drive' ,'APT A' ,'Long Island' ,'ny' ,'11101');

END;
```

Working Examples

Before Insert

Start Page | AIT-732 | CUSTOMER

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Sort.. | Filter:

	CUSTOMERID	FIRSTNAME	LASTNAME	CUSTOMERTYPE	ADDRESS_LINE1	ADDRESS_LINE2	CITY	STATE	ZIPCODE
1	1	Patty	May	Large Business	555 Coding Way	Unit G	Towson	MD	21286

After Insert

Start Page | AIT-732 | CUSTOMER_LOG

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Partitions | Indexes | SQL

Sort.. | Filter: | Actions...

	CUSTOMERID	USERID	CHANGE_DATE	NEW_ADDRESS_LINE1	NEW_ADDRESS_LINE2	NEW_CITY	NEW_STATE	NEW_ZIPCODE	OLD_ADDRESS_LINE1	OLD_ADDRESS_LINE2	OLD_CITY	OLD_STATE	OLD_ZIPCODE
1	1	USER69	05-DEC-16	111 SQL Drive	APT A	Long Island	NY	11101	123 Fake Street	APTe A	Baltimore	MD	21117

ALL DATA FROM ALL TABLES

Select *

From Customer, Accounts, Transaction

Where (Customer.CustomerID = Accounts.CustomerID) AND (Transaction.AccountNumber = Accounts.AccountNumber);

Moving Left to Right: Picture #1

CUSTOMERID	FIRSTNAME	LASTNAME	CUSTOMERTYPE	ADDRESS_LINE1	ADDRESS_LINE2
1	Patty	May	Large Business	111 SQL Drive	APT A
1	Patty	May	Large Business	111 SQL Drive	APT A
2	Frank	Smith	Small Business	555 Fake Street	
3	Mary	Reed	Personal	212 Fake Street	
4	John	Smith	Personal	999 Fake Street	
5	John	Duplicate	Personal	999 Fake Street	
1	Patty	May	Large Business	111 SQL Drive	APT A
1	Patty	May	Large Business	111 SQL Drive	APT A
1	Patty	May	Large Business	111 SQL Drive	APT A
1	Patty	May	Large Business	111 SQL Drive	APT A
1	Patty	May	Large Business	111 SQL Drive	APT A

CUSTOMERID	FIRSTNAME	LASTNAME	CUSTOMERTYPE	ADDRESS_LINE1	ADDRESS_LINE2
2	Frank	Smith	Small Business	555 Fake Street	
3	Mary	Reed	Personal	212 Fake Street	
4	John	Smith	Personal	999 Fake Street	
1	Patty	May	Large Business	111 SQL Drive	APT A

15 rows selected

Moving Left to Right: Picture #2

CITY	ST	ZIPCODE	ACCOUNTNUMBER	CUSTOMERID	ACCOUNTT	ACCOUNTBALAN	TRANSACTIONID	ACCOUNTNUMBER	ACCOUNTNUMBER	TRANSTYPE
Long Island	NY	11101	1.0E+11	1	Savings	1333	1	1.0E+11		Creation
Long Island	NY	11101	1.0E+11	1	Checking	15000	2	1.0E+11		Creation
Towson	MD	21286	1.0E+11	2	Business	10000	3	1.0E+11		Creation
Anchorage	AK	99507	1.0E+11	3	Checking	10000	4	1.0E+11		Creation
Scranton	PA	18504	1.0E+11	4	Savings	10000	5	1.0E+11		Creation
Scranton	PA	18504	1.0E+11	5	Checking	0	6	1.0E+11		Creation
Long Island	NY	11101	1.0E+11	1	Savings	1333	7	1.0E+11		Deposit
Long Island	NY	11101	1.0E+11	1	Savings	1333	8	1.0E+11		Withdraw
Long Island	NY	11101	1.0E+11	1	Savings	1333	10	1.0E+11	1.0E+11	Transfer
Long Island	NY	11101	1.0E+11	1	Checking	15000	11	1.0E+11	1.0E+11	Transfer
Long Island	NY	11101	1.0E+11	1	Checking	15000	12	1.0E+11		Deposit

CITY	ST	ZIPCODE	ACCOUNTNUMBER	CUSTOMERID	ACCOUNTT	ACCOUNTBALAN	TRANSACTIONID	ACCOUNTNUMBER	ACCOUNTNUMBER	TRANSTYPE
Towson	MD	21286	1.0E+11	2	Business	10000	13	1.0E+11		Deposit
Anchorage	AK	99507	1.0E+11	3	Checking	10000	14	1.0E+11		Deposit
Scranton	PA	18504	1.0E+11	4	Savings	10000	15	1.0E+11		Deposit
Long Island	NY	11101	1.0E+11	1	Savings	1333	16	1.0E+11		Deposit

Moving Left to Right: Picture #3

TRANSAMOUNT	TRANSDATE
0	06-DEC-16
0	06-DEC-16
0	06-DEC-16
0	06-DEC-16
0	06-DEC-16
0	06-DEC-16
10000	06-DEC-16
-5000	06-DEC-16
-5000	06-DEC-16
5000	06-DEC-16
10000	06-DEC-16

TRANSAMOUNT	TRANSDATE
10000	06-DEC-16
10000	06-DEC-16
10000	06-DEC-16
1333	06-DEC-16