**Ex: No: 07 WRITE A PL/SQL BLOCK THAT HANDLES**

**Date:15.03.24 ALL TYPES OF EXCEPTIONS.**

**AIM:**

To Write a PL/SQL block that handles all types of exceptions

**PROCEDURE:**

**Program to handle divide by zero exception:**

SQL> DECLARE

1. a NUMBER;
2. b NUMBER;
3. c NUMBER;
4. BEGIN
5. -- Prompt for user input
6. a := &a;
7. b := &b;

9

1. -- Perform division
2. c := a / b;

12

1. -- Output the result
2. dbms\_output.put\_line('Result of division: ' || to\_char(c));

15

1. EXCEPTION
2. WHEN ZERO\_DIVIDE THEN
3. dbms\_output.put\_line('Divisor cannot be zero');
4. WHEN VALUE\_ERROR THEN
5. dbms\_output.put\_line('Invalid input, please enter valid numbers');
6. END;
7. /

Enter value for a: 44 old 7: a := &a; new 7: a := 44; Enter value for b: 20 old 8: b := &b;

new 8: b := 20;

PL/SQL procedure successfully completed.

SQL> SET SERVEROUTPUT ON

SQL> /

Enter value for a: 44 old 7: a := &a; new 7: a := 44; Enter value for b: 0 old 8: b := &b; new 8: b := 0;

Divisor cannot be zero

PL/SQL procedure successfully completed.

SQL> /

Enter value for a: 99 old 7: a := &a; new 7: a := 99; Enter value for b: 33 old 8: b := &b; new 8: b := 33; Result of division: 3

PL/SQL procedure successfully completed.

**Program to accept sno from supplier table and print name of supplier if the status is greater than 20 else raise exception:**

SQL> CREATE TABLE supplier (

1. sno VARCHAR2(10),
2. sname VARCHAR2(20),
3. status NUMBER(3),
4. city VARCHAR2(20)
5. );

Table created.

SQL> INSERT INTO supplier (sno, sname, status, city) VALUES ('S1', 'Supplier 1', 10, 'Canada');

1 row created.

SQL> INSERT INTO supplier (sno, sname, status, city) VALUES ('S2', 'Supplier 2', 20, 'America');

1 row created.

SQL> INSERT INTO supplier (sno, sname, status, city) VALUES ('S3', 'Supplier 3', 30, 'Chicago');

1 row created.

SQL> INSERT INTO supplier (sno, sname, status, city) VALUES ('S4', 'Supplier 4', 15, 'UK');

1 row created.

SQL> INSERT INTO supplier (sno, sname, status, city) VALUES ('S5', 'Supplier 5',

35, 'Paris');

1 row created.

SQL> INSERT INTO supplier (sno, sname, status, city) VALUES ('S6', 'Supplier 6', 44, 'Britain');

1 row created.

SQL> INSERT INTO supplier (sno, sname, status, city) VALUES ('S7', 'Supplier 7', 50, 'Malasiya');

1 row created.

SQL> INSERT INTO supplier (sno, sname, status, city) VALUES ('S8', 'Supplier 8', 28, 'Russia');

1 row created.

SQL> INSERT INTO supplier (sno, sname, status, city) VALUES ('S9', 'Supplier 9', 12, 'Torando');

1 row created.

SQL> Select \* from supplier;

SNO SNAME STATUS CITY

---------- -------------------- ---------- -------------------- S1 Supplier 1 10 New York

S2 Supplier 2 20 Los Angeles

S3 Supplier 3 30 Chicago

S4 Supplier 4 15 UK

S5 Supplier 5 35 Paris

S6 Supplier 6 44 Britain

S7 Supplier 7 50 Malasiya

S8 Supplier 8 28 Russia

S9 Supplier 9 12 Torando

9 rows selected.

SQL> DECLARE

1. sn supplier.sno%TYPE;
2. snam supplier.sname%TYPE;
3. stat supplier.status%TYPE;
4. e1 EXCEPTION;
5. BEGIN
6. -- Prompt for user input
7. sn := '&serialno';

9

1. -- Select data from supplier table
2. SELECT sname, status INTO snam, stat FROM supplier WHERE sno = sn; 12
3. -- Check status and output name if greater than 20
4. IF stat > 20 THEN
5. dbms\_output.put\_line(snam);
6. ELSE
7. RAISE e1;
8. END IF;

19

1. EXCEPTION
2. WHEN e1 THEN
3. dbms\_output.put\_line('Status <= 20');
4. WHEN NO\_DATA\_FOUND THEN
5. dbms\_output.put\_line('No data found for the given serial number');
6. END;
7. /

Enter value for serialno: S4 old 8: sn := '&serialno';

new 8: sn := 'S4';

Status <= 20

PL/SQL procedure successfully completed.

SQL> /

Enter value for serialno: S2 old 8: sn := '&serialno'; new 8: sn := 'S2';

Supplier 2

**Program to accept sno from supplier table and print name of supplier if he resides in LONDON else raise exception:**

SQL> DECLARE

1. sn supplier.sno%TYPE;
2. snam supplier.sname%TYPE;
3. stat supplier.status%TYPE;
4. cit supplier.city%TYPE;
5. e1 EXCEPTION;
6. BEGIN
7. -- Prompt for user input
8. sn := '&sn';

10

1. -- Select data from supplier table
2. SELECT sname, status, city INTO snam, stat, cit FROM supplier WHERE sno = sn; 13
3. -- Debug statement to check retrieved values
4. dbms\_output.put\_line('Supplier Name: ' || snam);
5. dbms\_output.put\_line('City: ' || cit);

17

1. -- Check city and output name if it's 'LONDON'
2. IF UPPER(cit) = 'LONDON' THEN
3. dbms\_output.put\_line(snam);
4. ELSE
5. dbms\_output.put\_line('Supplier not in London');
6. END IF;

24

1. EXCEPTION
2. WHEN NO\_DATA\_FOUND THEN
3. dbms\_output.put\_line('No data found for the given serial number');
4. WHEN OTHERS THEN
5. RAISE e1;
6. END;

Enter value for sn: S4 old 9: sn := '&sn'; new 9: sn := 'S4';

Supplier Name: Supplier 4

City: London

Supplier 4

PL/SQL procedure successfully completed.

SQL> /

Enter value for sn: S3 old 9: sn := '&sn'; new 9: sn := 'S3';

Supplier Name: Supplier 3

City: Chicago

Supplier not in London

PL/SQL procedure successfully completed.

**RESULT:**

Thus the PL/SQL block that handles all types of exceptions has been verified and executedsuccessfully

**Ex: No: 08 CREATION OF PROCEDURES.**

# Date:22.04.24

**AIM:**

To work with PL / SQL Procedures

**PROCEDURE**

SQL> SET SERVEROUTPUT ON

SQL> CREATE OR REPLACE PROCEDURE pro

1. IS
2. a NUMBER;
3. b NUMBER;
4. c NUMBER;
5. d NUMBER;
6. BEGIN
7. -- Prompt for user input
8. a := &a;
9. b := &b;

11

1. IF a > b THEN
2. c := MOD(a, b);
3. IF c = 0 THEN
4. dbms\_output.put\_line('GCD is');
5. dbms\_output.put\_line(b);
6. ELSE
7. dbms\_output.put\_line('GCD is');
8. dbms\_output.put\_line(c);
9. END IF;
10. ELSE
11. d := MOD(b, a);
12. IF d = 0 THEN
13. dbms\_output.put\_line('GCD is');
14. dbms\_output.put\_line(a);
15. ELSE
16. dbms\_output.put\_line('GCD is');
17. dbms\_output.put\_line(d);
18. END IF;
19. END IF;
20. END;
21. /

SQL> SET SERVEROUTPUT ON;

SQL> /

Enter value for a: 212 old 9: a := &a; new 9: a := 212; Enter value for b: 12 old 10: b := &b;

new 10: b := 12;

Procedure created.

SQL> execute pro;

GCD is

8

PL/SQL procedure successfully completed.

**RESULT:**

Thus the implementation of PL/SQL procedure has been verified and executed successfully.

# Ex: No: 09 Creation of Database Triggers and functions

# Date:19.04.24

**AIM:**

To work with PL/SQL Triggers for the purpose of monitor the database object(table..etc) and functions .

**PROCEDURE:**

**Program to create a DB trigger before insert for each row on the spj table not allowing insertion for sno ‘s3’ and pno ‘p4’**

SQL> CREATE TABLE spj (

1. sno VARCHAR2(10),
2. pno VARCHAR2(10),
3. jno VARCHAR2(10)
4. );

Table created.

SQL> CREATE OR REPLACE TRIGGER t1

1. BEFORE INSERT ON spj
2. FOR EACH ROW
3. BEGIN
4. IF (:new.sno = 's3' AND :new.pno = 'p4') THEN
5. raise\_application\_error(-20000, 'Cannot insert s3, p4');
6. END IF; -- End IF statement
7. END; -- End trigger body
8. /

SQL> @p1001.sql

SP2-0310: unable to open file "p1001.sql"

SQL> INSERT INTO spj VALUES ('s3', 'p4', 'j2', 30);

INSERT INTO spj VALUES ('s3', 'p4', 'j2', 30)

\*

ERROR at line 1:

ORA-00913: too many values

**Program to create a DB trigger to update the qty if qty is greater than existing qty** SQL> CREATE OR REPLACE TRIGGER t2

1. BEFORE UPDATE ON spj
2. FOR EACH ROW
3. BEGIN
4. IF (:new.qty < :old.qty) THEN
5. raise\_application\_error(-20001, 'Cannot Update');
6. END IF;
7. END;
8. /

SQL>@p1002.sql

Trigger created

SQL>update spj set qty=10 where sno=’s1’;

update spj set qty=10 where sno=’s1’ \*

ERROR at line 1:

ORA-20001: Cannot update

ORA-06512: at “309038.T2”,line 4

ORA-04088: error during execution of trigger ‘309038.T2’

**Program to create a DB trigger not allowing deletion in supplier table**

SQL> CREATE OR REPLACE TRIGGER t3

1. BEFORE DELETE ON supplier
2. FOR EACH ROW
3. BEGIN
4. raise\_application\_error(-20002, 'Deletion Not allowed');
5. END;
6. /

Trigger created.

SQL> DELETE FROM supplier WHERE sno = 'S1';

DELETE FROM supplier WHERE sno = 'S1'

\*

ERROR at line 1:

ORA-20002: Deletion Not allowed

ORA-06512: at "SCOTT.T3", line 2

ORA-04088: error during execution of trigger 'SCOTT.T3'

**Create a PL/SQL trigger which prevents the insertion of new record into the employee table.**

SQL> create or replace trigger trig before insert on emp for each row 2 begin

1. raise\_application\_error (-20998, 'insertion not allowed');
2. end;
3. /

Trigger created.

SQL> INSERT INTO emp (empno, ename) VALUES (123, 'John');

INSERT INTO emp (empno, ename) VALUES (123, 'John')

\*

ERROR at line 1:

ORA-20998: insertion not allowed

ORA-06512: at "SCOTT.TRIG", line 2

ORA-04088: error during execution of trigger 'SCOTT.TRIG'

**Create a PL/SQL trigger which prevents all DML operations on the table account.**

SQL> CREATE OR REPLACE TRIGGER kkks

1. BEFORE INSERT OR DELETE OR UPDATE ON accounts
2. FOR EACH ROW
3. BEGIN
4. raise\_application\_error(-04098, 'Changes not allowed');
5. END;
6. /

Trigger created.

SQL> INSERT INTO accounts (account\_id, account\_name) VALUES (6, 'Mortgage');

INSERT INTO accounts (account\_id, account\_name) VALUES (6, 'Mortgage')

\*

ERROR at line 1:

ORA-21000: error number argument to raise\_application\_error of -4098 is out of range

ORA-06512: at "SCOTT.KKKS", line 2

ORA-04088: error during execution of trigger 'SCOTT.KKKS'

**Create a PL/SQL trigger to update the salary of employee if the salary is greater than the existing salary.**

SQL> CREATE OR REPLACE TRIGGER kkt

1. BEFORE UPDATE ON emp
2. FOR EACH ROW
3. BEGIN
4. IF :new.sal < :old.sal THEN
5. raise\_application\_error(-20002, 'Salary cannot be reduced');
6. END IF;
7. END;
8. /

Trigger created.

SQL> UPDATE emp SET sal = 3000 WHERE empno = 7844;

1 row updated.

SQL> UPDATE emp SET sal = 2000 WHERE empno = 7844;

UPDATE emp SET sal = 2000 WHERE empno = 7844

\*

ERROR at line 1:

ORA-20002: Salary cannot be reduced

ORA-06512: at "SCOTT.KKT", line 3

ORA-04088: error during execution of trigger 'SCOTT.KKT'

**FUNCTIONS**

**To write a PL/SQL block to find factorial of given number using function**

SQL> CREATE OR REPLACE FUNCTION fact(limit NUMBER) RETURN

NUMBER IS

1. ans NUMBER(3);
2. BEGIN
3. ans := 1;
4. FOR i IN 1..limit LOOP
5. ans := ans \* i;
6. END LOOP;
7. RETURN ans;
8. END;
9. /

Function created.

SQL>

SQL> DECLARE

1. n NUMBER(3);
2. f NUMBER(3);
3. BEGIN
4. n := &limit;
5. f := fact(n);
6. dbms\_output.put\_line(n || '! = ' || f);
7. END;
8. /

Enter value for limit: 5 old 5: n := &limit; new 5: n := 5;

5! = 120

PL/SQL procedure successfully completed.

**Create a function which count total no.of employees having salary less than 6000.**

SQL> CREATE OR REPLACE FUNCTION count\_emp(esal NUMBER) RETURN

NUMBER AS

1. CURSOR vin\_cur IS SELECT empno, sal FROM emp;
2. Xno emp.empno%TYPE;
3. Xsal emp.sal%TYPE;
4. C NUMBER := 0;
5. BEGIN
6. OPEN vin\_cur;

8

1. LOOP
2. FETCH vin\_cur INTO Xno, Xsal;
3. EXIT WHEN vin\_cur%NOTFOUND;

12

1. IF Xsal < esal THEN
2. C := C + 1;
3. END IF;
4. END LOOP;

17

18 CLOSE vin\_cur;

19

1. RETURN C;
2. END;
3. /

Function created.

SQL> /\* Function specification \*/

SQL> DECLARE

1. Ne NUMBER;
2. Xsal NUMBER := 3000; -- Example salary value
3. BEGIN
4. Ne := count\_emp(Xsal);
5. DBMS\_OUTPUT.PUT\_LINE('Number of employees with salary less than ' ||

Xsal || ': ' || Ne);

1. END;
2. /

Number of employees with salary less than 3000: 9

PL/SQL procedure successfully completed.

Program to accept pno from parts table and print name of parts (using function):

SQL> -- Create the parts table

SQL> CREATE TABLE parts (

1. pno VARCHAR2(10),
2. pname VARCHAR2(100),
3. CONSTRAINT pk\_parts PRIMARY KEY (pno)
4. );

Table created.

SQL>

SQL> -- Insert sample values into the parts table

SQL> INSERT INTO parts (pno, pname) VALUES ('P1', 'Engine Assembly');

1 row created.

SQL> INSERT INTO parts (pno, pname) VALUES ('P2', 'Chassis Frame');

1 row created.

SQL> INSERT INTO parts (pno, pname) VALUES ('P3', 'Brake System');

1 row created.

SQL> INSERT INTO parts (pno, pname) VALUES ('P4', 'Transmission Unit');

1 row created.

SQL> INSERT INTO parts (pno, pname) VALUES ('P5', 'Suspension Kit');

1. row created.

SQL> CREATE OR REPLACE FUNCTION findname(p parts.pno%TYPE) RETURN

VARCHAR2 IS

1. a parts.pname%TYPE;
2. BEGIN
3. SELECT pname INTO a FROM parts WHERE pno = p;
4. RETURN a;
5. END;
6. /

Function created.

SQL> SELECT findname('P1') AS part\_name FROM DUAL;

PART\_NAME

--------------------------------------------------------------------------------

Engine Assembly

**RESULT:**

Thus the implementation of functions and database triggers has been executed successfully.

**EX.NO:10 DATABASE CONNECTIVITY WITH FRONT END TOOLS**

**DATE:26.04.24**

**EB BILL PREPARATION**

**AIM:**

To prepare a form in VB to generate EB bill and connect it SQL back end.

**PROCEDURE:**

Dim su As Integer

Dim eu As Integer

Dim consumed As Integer

Dim amount As Integer

Dim var1 As Integer

Dim cn As New ADODB.Connection

Dim rs As New ADODB.Recordset

Private Sub clear\_Click()

' Clear all text boxes

Text1.Text = ""

Text2.Text = ""

Text3.Text = ""

Text4.Text = ""

Text5.Text = ""

Text6.Text = ""

Text7.Text = ""

Text8.Text = ""

Text9.Text = ""

End Sub

Private Sub Delete\_Click()

' Delete the current record in the recordset rs.Delete

' Clear all text boxes after deleting

Text1.Text = ""

Text2.Text = ""

Text3.Text = ""

Text4.Text = ""

Text5.Text = "" ' Update the recordset rs.Update

' Display a message box indicating the record is deleted

MsgBox "Record Deleted"

End Sub

Private Sub eb\_Click()

' Show the DataReport1

DataReport1.Show

End Sub

Private Sub first\_Click()

' Move to the first record in the recordset rs.MoveFirst

' Update Text1 to Text5 with the corresponding field values

Text1.Text = rs.Fields(0)

Text2.Text = rs.Fields(1)

Text3.Text = rs.Fields(2)

Text4.Text = rs.Fields(3)

Text5.Text = rs.Fields(4)

' Display a message box indicating this is the first record

MsgBox "This is the first record"

End Sub

Private Sub gm\_Click()

' Calculate consumed and amount based on Text6 and Text7 values su = Text6.Text eu = Text7.Text consumed = eu - su Text8.Text = consumed If (consumed > 400) Then

amount = consumed \* 2 Else

amount = consumed \* 1.5

End If

Text9.Text = amount

End Sub

Private Sub insert\_Click()

' Add a new record to the recordset based on Text1 to Text5 values rs.AddNew rs.Fields(0) = Text1.Text rs.Fields(1) = Text2.Text rs.Fields(2) = Text3.Text rs.Fields(3) = Text4.Text rs.Fields(4) = Text5.Text ' Update the recordset rs.Update

' Display a message box indicating data was successfully added

MsgBox "Data was successfully added"

End Sub

Private Sub last\_Click()

' Move to the last record in the recordset rs.MoveLast

' Update Text1 to Text5 with the corresponding field values

Text1.Text = rs.Fields(0)

Text2.Text = rs.Fields(1)

Text3.Text = rs.Fields(2)

Text4.Text = rs.Fields(3)

Text5.Text = rs.Fields(4)

' Display a message box indicating this is the last record

MsgBox "This is the last record"

End Sub

Private Sub next\_Click()

' Move to the next record in the recordset

rs.MoveNext

' If at the end of the recordset, display a message box

If (rs.EOF) Then

MsgBox "This is the last record"

Else

' Update Text1 to Text5 with the corresponding field values

Text1.Text = rs.Fields(0)

Text2.Text = rs.Fields(1)

Text3.Text = rs.Fields(2)

Text4.Text = rs.Fields(3)

Text5.Text = rs.Fields(4)

End If

End Sub

Private Sub previous\_Click()

' Move to the previous record in the recordset rs.MovePrevious

' If at the beginning of the recordset, display a message box

If (rs.BOF) Then

MsgBox "This is the first record"

Else

' Update Text1 to Text5 with the corresponding field values

Text1.Text = rs.Fields(0)

Text2.Text = rs.Fields(1)

Text3.Text = rs.Fields(2)

Text4.Text = rs.Fields(3)

Text5.Text = rs.Fields(4)

End If

End Sub

Private Sub Form\_Load()

' Establish connection to the database

Set cn = New ADODB.Connection Set rs = New ADODB.Recordset cn.Open "dsn=eb;UserId=96017;Password=96017;" rs.Open "eb", cn, adOpenDynamic, adLockOptimistic

End Sub

Private Sub report\_Click()

' Show the DataReport1

DataReport1.Show

End Sub

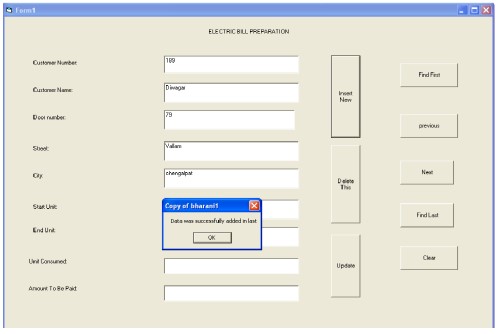
Private Sub update\_Click()

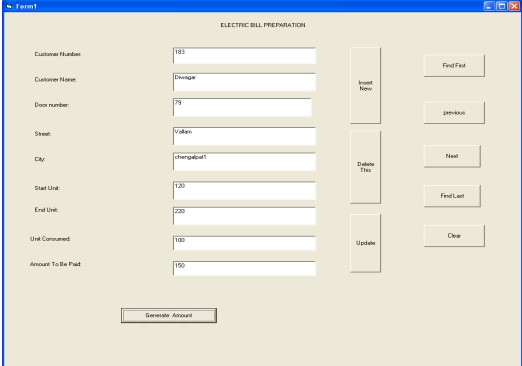
' Update the current record in the recordset with Text1 to Text5 values rs.Fields(0) = Text1.Text rs.Fields(1) = Text2.Text rs.Fields(2) = Text3.Text rs.Fields(3) = Text4.Text rs.Fields(4) = Text5.Text ' Update the recordset rs.Update

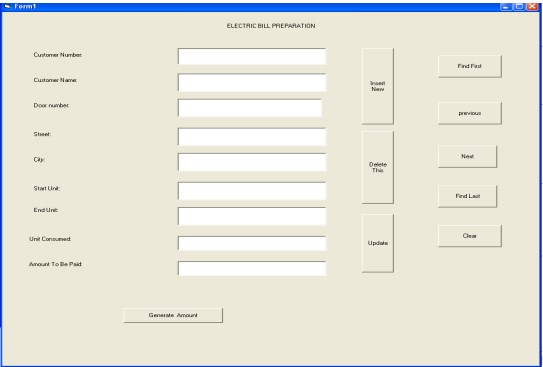
' Display a message box indicating data was updated

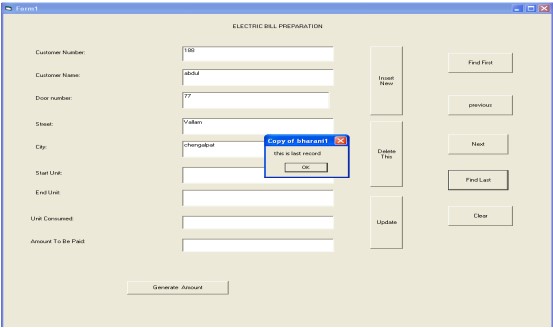
MsgBox "Data Updated"

End Sub









**RESULT**:

Thus the mini project for eb bill with sql back end was created Successfully

# Exp No. 11 Implementation of Decision tree

# Date:3.05.24

**AIM:**

To create a program to implement Decision Tree in Python sklearn

**PROCEDURE:**

import pandas as pd import numpy as np Import matplotlib.pyplot as plt from sklearn import metrics import seaborn as sns from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn import tree

iris = load\_iris()

data = pd.DataFrame(data = iris.data, columns iris.feature\_names) data['Species'] = iris.target target = np.unique(iris.target)

target\_n = np.unique(iris.target\_names) target\_dict = dict(zip(target, target\_n)) data['Species']=data['Species'].replace(target\_dict) x =data.drop(columns = "Species") y = data["Species"] names\_features = x.columns target\_labels = y.unique()

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.3, random\_state =93)

from sklearn.tree import DecisionTreeClassifier dtc = DecisionTreeClassifier(max\_depth = 3,random\_state = 93) dtc.fit(x\_train, y\_train) plt.figure(figsize = (30, 10), facecolor = 'b')

Tree = tree.plot\_tree(dtc, feature\_names = names\_features, class\_names = target\_labels, rounded = True, filled = True, fontsize = 14) plt.show() y\_pred = dtc.predict(x\_test)

confusion\_matrix = metrics.confusion\_matrix(y\_test, y\_pred) matrix = pd.DataFrame(confusion\_matrix) axis = plt.axes() sns.set(font\_scale = 1.3) plt.figure(figsize = (10,7))

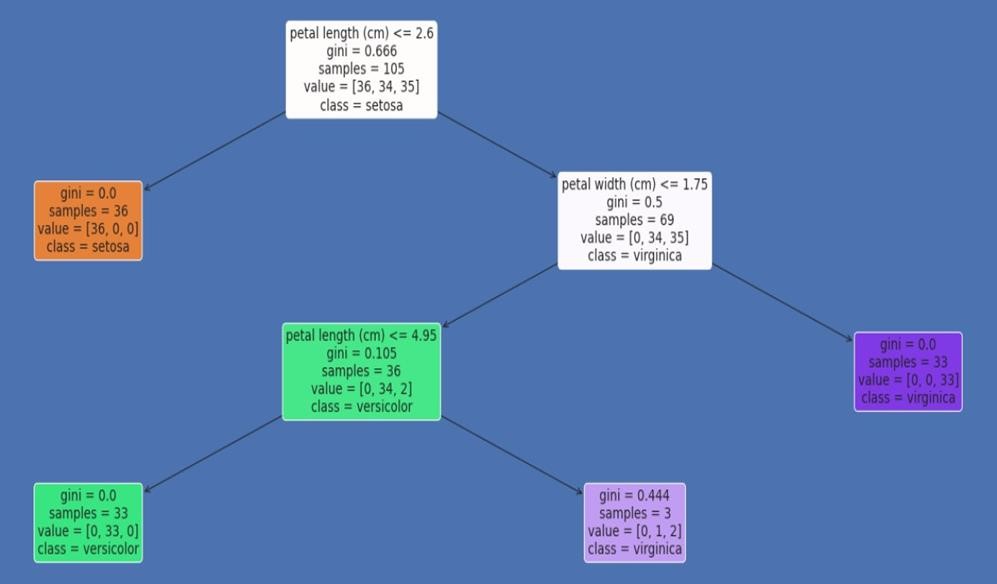
sns.heatmap(matrix, annot = True, fmt = "g", ax = axis, cmap = "magma") axis.set\_title('Confusion Matrix')

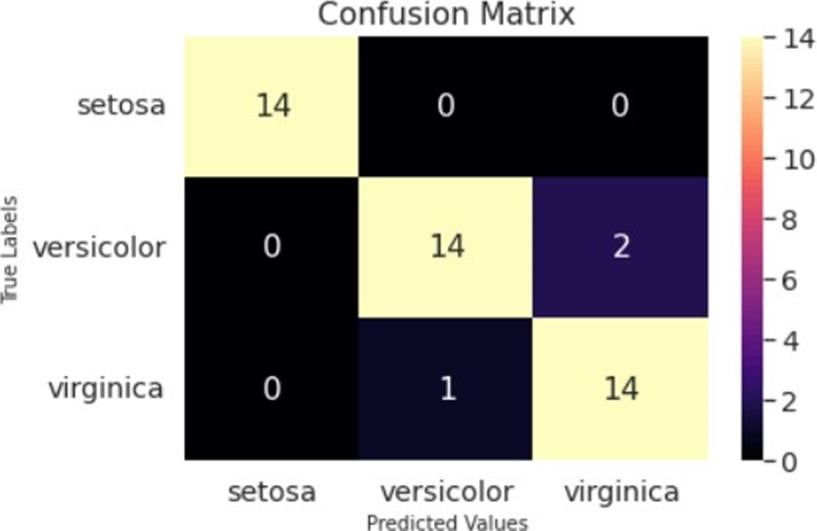
axis.set\_xlabel("Predicted Values", fontsize = 10) axis.set\_xticklabels([''] + target\_labels)

axis.set\_ylabel( "True Labels", fontsize = 10) axis.set\_yticklabels(list(target\_labels), rotation = 0) plt.show()

# 

**OUTPUT:**





**RESULT:**

Thus the program to implement the decision tree is implemented and the output is obtained.

# EX:NO: 12 Implementation of Apriori Algorithm

# Date: 3.05.24

**Aim:**

To implement the apriori algorithm in python

Procedure:

**import** numpy as np

**import** pandas as pd

**from** mlxtend.frequent\_patterns **import** apriori, association\_rules

# Now, we will load the Data data1 = pnd.read\_excel('Online\_Retail.xlx') data1.head()

**Output:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Invoice  No | Stock  Code | Description | Q  ua nti ty | InvoiceDa  te | Uni  tP rice | Custo m erID | Country |
| 0 | 536365 | 85123A | WHITE  HANGING | 6 | 2010-1201  08:26:00 | 2.55 | 17850.0 | United Kingdom |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 536365 | 71053 | WHITE  METAL  LANTERN | 6 | 2010-12-01  08:26:00 | 3.39 | 17850.0 | United Kingdom |
| 2 | 536365 | 84406B | CREAM  CUPID  HEARTS  COAT  HANGER | 8 | 2010-12-01  08:26:00 | 2.75 | 17850.0 | United Kingdom |
| 3 | 536365 | 84029G | KNITTED  UNION  FLAG  HOT  WATER  BOTTLE | 6 | 2010-12-01  08:26:00 | 3.39 | 17850.0 | United Kingdom |
| 4 | 536365 | 84029E | RED  WOOLLY  HOTTIE WHITE  HEART. | 6 | 2010-12-01  08:26:00 | 3.39 | 17850.0 | United Kingdom |

**Input:**

# here, we will explore the columns of the data data1.columns

**Output:**

Index(['InvoiceNo', 'StockCode', 'Description',

'Quantity', 'InvoiceDate', 'UnitPrice',

'CustomerID', 'Country'],

Dtype = 'object') **Input:**

# Now, we will explore the different regions of transactions data1.Country.unique()

**Output:**

array(['United Kingdom', 'France', 'Australia',

'Netherlands', 'Germany', 'Norway',

'EIRE', 'Switzerland', 'Spain', 'Poland',

'Portugal',

'Italy', 'Belgium', 'Lithuania', 'Japan', 'Iceland',

'Channel Islands', 'Denmark', 'Cyprus',

'Sweden', 'Austria', 'Israel', 'Finland',

'Bahrain', 'Greece', 'Hong Kong',

'Singapore', 'Lebanon', 'United Arab

Emirates', 'Saudi Arabia',

'Czech Republic', 'Canada',

'Unspecified', 'Brazil', 'USA',

'European Community', 'Malta',

'RSA'], dtype = object)

# here, we will strip the extra spaces in the description data1['Description']

=data1['Description'].str.strip()

# Now, drop the rows which does not have any invoice number data1.dropna(axis = 0, subset = ['InvoiceNo'], inplace = True) data1['InvoiceNo']=data1['InvoiceNo'].astype('str') # Now, we will drop all transactions which were done on credit\_data1=data1[~data1['InvoiceNo'].str.contains('C')]

# Transactions done in France basket1\_France=(data1[data1['Country']=="France"].groupby(['InvoiceNo'

, 'Description'])['Quantity'].sum().unstack().reset\_index().fillna(0)

.set\_index('InvoiceNo'))

# Transactions done in the United Kingdom

basket1\_UK = (data1[data1['Country'] == "UnitedKingdom"].groupby(['InvoiceNo', 'Description'])['Quantity'].sum().unstack().reset\_index().fillna(0).set\_index('Invoice No'))

# Transactions done in Portugal

basket1\_Por = (data1[data1['Country'] =="Portugal"].groupby(['InvoiceNo', 'Description'])['Quantity'].sum().unstack().reset\_index().fillna(0).set\_index('In voiceNo'))

basket1\_Sweden=(data1[data1['Country']=="Sweden"].groupby(['InvoiceN o',

'Description'])['Quantity'].sum().unstack().reset\_index().fillna(0).set\_index( 'InvoiceNo'))

# Here, we will define the hot encoding function # for making the data suitable encode the datasets

basket1\_encoded = basket1\_France.applymap(hot\_encode1) basket1\_France = basket1\_encoded# for the concerned libraries

**def** hot\_encode1(P): **if**(P<= 0):

**return** 0 **if**(P>= 1):

# return 1

# Here, we will

basket1\_encoded = basket1\_UK.applymap(hot\_encode1) basket1\_UK = basket1\_encoded basket1\_encoded = basket1\_Por.applymap(hot\_encode1) basket1\_Por = basket1\_encoded basket1\_encoded = basket1\_Sweden.applymap(hot\_encode1) basket1\_Sweden = basket1\_encoded

**France:**

# Build the model frq\_items1 = AP(basket1\_France, min\_support = 0.05, use\_colnames = True) # Collect the inferred rules in a dataframe rules1 = AR(frq\_items1, metric = "lift", min\_threshold = 1) rules1 = rules1.sort\_values(['confidence', 'lift'], ascending = [False, False])

**print**(rules1.head() **Output:**

|  |
| --- |
| antecedents \ |
| 45 (JUMBO BAG WOODLAND ANIMALS) |
| 260 (PLASTERS IN TIN CIRCUS PARADE, RED TOADSTOOL ... |

|  |
| --- |
| 272 (RED TOADSTOOL LED NIGHT LIGHT, PLASTERS IN TI... |
| 302 (SET/6 RED SPOTTY PAPER CUPS, SET/20 RED RETRO... |
| 301 (SET/6 RED SPOTTY PAPER PLATES, SET/20 RED RET... |
|  |
| consequents antecedent support consequent support \ |
| 45 (POSTAGE) 0.076531 0.765306 |
| 260 (POSTAGE) 0.051020 0.765306 |
| 272 (POSTAGE) 0.053571 0.765306 |
| 302 (SET/6 RED SPOTTY PAPER PLATES) 0.102041s 0.127551 |
| 301 (SET/6 RED SPOTTY PAPER CUPS) 0.102041 0.137755 |
|  |
| support confidence lift leverage conviction |
| 45 0.076531 1.000 1.306667 0.017961 inf |
| 260 0.051020 1.000 1.306667 0.011974 inf |
| 272 0.053571 1.000 1.306667 0.012573 inf |
| 302 0.099490 0.975 7.644000 0.086474 34.897959 |
| 301 0.099490 0.975 7.077778 0.085433 34.489796 |

It can be seen that paper cups, paper and plates are brought together in France.

**RESULT:**

Thus the python program to implement the apriori algorithm is executed and the output is obtained succesfully.