**Teradata Objects:**

It is a collection of objects known as tables, views, macros, triggers and stored procedures. Databases provide a logical grouping for information

1. Tables
2. Views
3. Macros
4. Triggers
5. Stored Procedures
6. Views

Views in Teradata can be easily created or modified. Views are very useful and are used for several purposes. A VIEW is a virtual table and don’t need any physical space to save data. Only VIEW definition is stored in DATA DICTIONARY and every time VIEW is queried; the SQL statement is executed and result is returned back to the user. VIEWS in Teradata can be created by using following syntax:

CREATE VIEWS IN TERADATA

CREATE VIEW VIEW\_NAME (COL1,COL2,COL3) AS SELECT COLUMN1, COLUMN2, COLUMN3

FROM BASE\_TABLE;

REPLACE/UPDATE/MODIFY VIEWS IN TERADATA

REPLACE VIEW VIEW\_NAME (COL1,COL2,COL3,COL4) AS SELECT COLUMN1, COLUMN2, COLUMN3, COLUMN4 FROM BASE\_TABLE;

ADVANTAGES OF VIEWS IN TERADATA

Adds extra level of security by avoiding access to base tables.

Helps in giving more proper column names than the original names present in base tables.

Allows to limit access to the records present in the base table by the use of WHERE clause.

Allows to limit access to the columns present in the base table by SELECTing only required columns.

Combine the result from multiple tables and provide single place to access the records.

You can use the VIEW to insert or update the data present in the base table.

**VIEWS WITH CHECK OPTION**

If you are creating a VIEW with CHECK option, then it will not only user to insert or update data in the base table if it violates the data constraints applied in the VIEW definition. Let see the example below:

BTEQ -- Enter your DBC/SQL request or BTEQ command:

.logon localtd/dbc

grant create view on financial to td06;

\*\*\* Grant accepted.

\*\*\* Total elapsed time was 1 second.

BTEQ -- Enter your DBC/SQL request or BTEQ command:

.logon localtd/td06

.logon localtd/td06

Password:

\*\*\* Logon successfully completed.

\*\*\* Teradata Database Release is 13.00.00.12

\*\*\* Teradata Database Version is 13.00.00.12

\*\*\* Transaction Semantics are BTET.

\*\*\* Character Set Name is 'ASCII'.

\*\*\* Total elapsed time was 1 second.

BTEQ -- Enter your DBC/SQL request or BTEQ command:

CREATE VIEW emp1\_vu AS SELECT emp\_no, first\_name, salary, dept\_no FROM emp1 WHERE salary > 40000 WITH CHECK OPTION;

\*\*\* View has been created.

\*\*\* Total elapsed time was 1 second.

Select \* from emp\_vu;

\*\*\* Query completed. 3 rows found. 4 columns returned.

\*\*\* Total elapsed time was 1 second.

EMP\_NO FIRST\_NAME SALARY DEPT\_NO

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

1115 Pritam 50000.00 10

1111 Ram 50000.00 10

1113 Mark 60000.00 30

BTEQ -- Enter your DBC/SQL request or BTEQ command:

Insert into emp\_vu values(1116, 'Rama',60000,20);

\*\*\* Insert completed. One row added.

\*\*\* Total elapsed time was 1 second.

BTEQ -- Enter your DBC/SQL request or BTEQ command:

Insert into emp\_vu values(1117, 'Deva',30000,30);

\*\*\* Failure 3564 Range constraint: Check error in field EMP1.SALARY.

Statement# 1, Info =0

\*\*\* Total elapsed time was 1 second.

Now if you try to insert into the base table via this view any data which has EMP\_SAL<40000, it will throw the following error:

\*\*\*\*Failure 3564 Range Constraint: Check error in field …

# Modifying Views

An existing view can be modified using REPLACE VIEW statement.

Following is the syntax to modify a view.

REPLACE VIEW <viewname>

AS

<select query>;

Example

The following example modifies the view Employee\_View for adding additional columns.

REPLACE VIEW emp1\_vu AS SELECT emp\_no, first\_name, last\_name, salary, dept\_no FROM emp1 WHERE salary > 40000 WITH CHECK OPTION;

\*\*\* View has been created.

\*\*\* Total elapsed time was 1 second.

BTEQ -- Enter your DBC/SQL request or BTEQ command:

Select \* from emp1\_vu;

\*\*\* Query completed. 4 rows found. 5 columns returned.

\*\*\* Total elapsed time was 1 second.

EMP\_NO FIRST\_NAME LAST\_NAME SALARY DEPT\_NO

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

1115 Pritam Singh 50000.00 10

1111 Ram Singh 50000.00 10

1113 Mark Taylor 60000.00 30

1116 Rama ? 60000.00 20

\*\*\* Null statement accepted.

Drop View

An existing view can be dropped using DROP VIEW statement.

Syntax

Following is the syntax of DROP VIEW.

DROP VIEW <viewname>;

DROP VIEW emp1\_vu;

\*\*\* View has been dropped.

\*\*\* Total elapsed time was 1 second.

CREATE TRIGGER/REPLACE TRIGGER

Required Privileges

To create a trigger, you must have the following privileges:

• CREATE TRIGGER on both of the following:

1. The database in which the trigger is created.
2. Either the subject table or its containing database.

• DROP TRIGGER on the subject table or the database.

Privileges Granted Automatically

None.

# Example 1 : Ensure That Parent Table Updates Propagate to Its Child

Suppose you want to ensure that all UPDATEs and DELETEs to a parent table are propagated to a child table. This example, which is designed to enforce referential integrity procedurally, defines two AFTER statement triggers on the parent table to ensure that primary key updates to the parent table propagate to the appropriate foreign key column in a child table.

These are the table definitions:

CREATE TABLE parents(

ParentID INTEGER,

Pname VARCHAR(10),

Age INTEGER)

UNIQUE PRIMARY INDEX (ParentID);

INSERT INTO parents VALUES(100, 'Ram',30);

INSERT INTO parents VALUES(101, 'Dev',40);

INSERT INTO parents VALUES(102, 'Jack',35);

INSERT INTO parents VALUES(103, 'Taylor',32);

INSERT INTO parents VALUES(104, 'Patric ',45);

INSERT INTO parents VALUES(105, 'Miller',50);

select \* from parents;

\*\*\* Query completed. 6 rows found. 3 columns returned.

\*\*\* Total elapsed time was 1 second.

ParentID Pname Age

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

101 Dev 40

104 Patric 45

102 Jack 35

100 Ram 30

105 Miller 50

103 Taylor 32

CREATE TABLE child(

ChildID INTEGER,

Cname VARCHAR(10),

Age INTEGER,

ParentID INTEGERREFERENCES WITH NO CHECK OPTIONparents(ParentID));

INSERT INTO child VALUES(1001, 'Adams',10,101);

INSERT INTO child VALUES(1002, 'Mark',05,102);

INSERT INTO child VALUES(1003, 'John',10,103);

INSERT INTO child VALUES(1004, 'Nency',15,104);

INSERT INTO child VALUES(1005, 'Marry',20,105);

SELECT \* FROM child;

\*\*\* Query completed. 5 rows found. 4 columns returned.

\*\*\* Total elapsed time was 1 second.

ChildID Cname Age ParentID

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

1005 Marry 20 105

1004 Nency 15 104

1002 Mark 5 102

1003 John 10 103

1001 Adams 10 101

.logon localtd/dbc

Password:

\*\*\* Logon successfully completed.

\*\*\* Teradata Database Release is 13.00.00.12

\*\*\* Teradata Database Version is 13.00.00.12

\*\*\* Transaction Semantics are BTET.

\*\*\* Character Set Name is 'ASCII'.

\*\*\* Total elapsed time was 1 second.

BTEQ -- Enter your DBC/SQL request or BTEQ command:

grant create trigger, drop trigger on financial to td06;

\*\*\* Grant accepted.

\*\*\* Total elapsed time was 1 second.

BTEQ -- Enter your DBC/SQL request or BTEQ command:

.logon localtd/td06

.logon localtd/td06

Password:

Now define the triggers on Parents , the subject table:

CREATE TRIGGER UpdateForKey

AFTER UPDATE OF ParentID ON parents

REFERENCING OLD TABLE AS OldTable NEW TABLE AS NewTable

FOR EACH STATEMENT(UPDATE childSET ParentID = NewTable.ParentID

WHERE child.ParentID = OldTable.ParentID;);

\*\*\* Trigger has been created.

\*\*\* Total elapsed time was 1 second.

CREATE TRIGGER DelForKey

AFTER DELETE ON parents

REFERENCING OLD TABLE AS OldTable

FOR EACH STATEMENT(UPDATE child SET ParentID=NULL

WHERE ParentID= OldTable.ParentID;);

\*\*\* Trigger has been created.

\*\*\* Total elapsed time was 1 second.

select \* from parents;

\*\*\* Query completed. 6 rows found. 3 columns returned.

\*\*\* Total elapsed time was 1 second.

ParentID Pname Age

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

101 Dev 40

104 Patric 45

102 Jack 35

100 Ram 30

105 Miller 50

103 Taylor 32

update parents set **parentid=1001**where parentid=101;

\*\*\* Update completed. One row changed.

\*\*\* Total elapsed time was 1 second.

select \* from child;

\*\*\* Query completed. 5 rows found. 4 columns returned.

\*\*\* Total elapsed time was 1 second.

ChildID Cname Age ParentID

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

1005 Marry 20 105

1004 Nency 15 104

1002 Mark 5 102

1003 John 10 103

1001 Adams 10 **1001**

delete from parents where parentid=105;

\*\*\* Delete completed. One row removed.

\*\*\* Total elapsed time was 1 second.

select \* from parents;

\*\*\* Query completed. 5 rows found. 3 columns returned.

\*\*\* Total elapsed time was 1 second.

ParentID Pname Age

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

104 Patric 45

102 Jack 35

100 Ram 30

**1001** Dev 40

103 Taylor 32

select \* from child;

\*\*\* Query completed. 5 rows found. 4 columns returned.

\*\*\* Total elapsed time was 1 second.

ChildID Cname Age ParentID

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

1005 Marry 20 **?**

1004 Nency 15 104

1002 Mark 5 102

1003 John 10 103

1001 Adams 10 **1001**

# Example : Audit Log for Large Pay Raises

Triggers are particularly useful for audits of all kinds. This example shows an AFTER row trigger that inserts a log record whenever an employee gets a raise larger than ten percent.

These are the table definitions:

CREATE TABLE employee (

name CHARACTER(10),

dept\_Id INTEGER,

salary DECIMAL(10,2),

comments CHARACTER(20));

CREATE TABLE salary\_log (

user\_name CHARACTER(10),

emp\_name CHARACTER(10),

old\_salary DECIMAL(10,2),

new\_salary DECIMAL(10,2));

INSERT INTO employee VALUES('John Smith',10,30000,'');

INSERT INTO employee VALUES('Min Chan',20,40000,'');

INSERT INTO employee VALUES('Lev Ulyanov',30,20000,'');

INSERT INTO employee VALUES('Jack Smith',20,50000,'');

INSERT INTO employee VALUES('Adam Smith',30,60000,'');

select \* from employee;

\*\*\* Query completed. 5 rows found. 4 columns returned.

\*\*\* Total elapsed time was 1 second.

name dept\_Id salary comments

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

Adam Smith 30 60000.00

Min Chan 20 40000.00

Lev Ulyano 30 20000.00

John Smith 10 30000.00

Jack Smith 20 50000.00

select \* from salary\_log;

\*\*\* Query completed. No rows found.

\*\*\* Total elapsed time was 1 second.

Now define the trigger on the employee table:

CREATE TRIGGER RaiseTrig

AFTER UPDATE OF salary ON employee

REFERENCING OLD AS OldRow NEW AS NewRow

FOR EACH ROW

WHEN ((NewRow.salary-OldRow.salary)/OldRow.salary >.10)

INSERT INTO salary\_log

VALUES ('USER', NewRow.name, OldRow.salary, NewRow.salary);

When the following requests are processed, two inserts are made to salary\_log. The third update does not meet the WHEN condition of the trigger, so no corresponding row is inserted in salary\_log.

UPDATE employee

SET salary = salary\*1.5, comments = 'Employee of the Year'

WHERE name = 'John Smith';

UPDATE employee

SET salary = salary\*2, comments = 'Employee of the Decade'

WHERE name = 'Min Chan';

UPDATE employee

SET salary = salary\*1.05, comments = 'Normal midrange raise'

WHERE name = 'Lev Ulyanov';

select \* from salary\_log;

\*\*\* Query completed. 2 rows found. 4 columns returned.

\*\*\* Total elapsed time was 1 second.

user\_name emp\_name old\_salary new\_salary

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

USER John Smith 30000.00 45000.00

USER Min Chan 40000.00 80000.00

BTEQ -- Enter your DBC/SQL request or BTEQ command:

DROP TRIGGER RaiseTrig;

DROP TRIGGER RaiseTrig;

\*\*\* Trigger has been dropped.

\*\*\* Total elapsed time was 1 second.

BTEQ -- Enter your DBC/SQL request or BTEQ command:

# Example : Using a SET Clause

This example uses a SET clause as a triggered action statement in a BEFORE row trigger.This is the subject table definition. Assume that values are loaded into the amount column of the table from a source file employing a USING … INSERT request.

CREATE TABLE subject\_table (

entry\_date DATE,

amount INTEGER);

This is the trigger definition:

CREATE TRIGGER set\_trig

BEFORE INSERT ON subject\_table

REFERENCING NEW AS curr\_value

FOR EACH ROW

SET curr\_value.entry\_date = DATE;

-- Adds the current system date to the entry\_date column

Because of the SET clause assignment, the following triggering statement:

USING (amount INTEGER)

INSERT INTO subject\_table VALUES (NULL,:amount);

executes as if it were this statement:

USING (thisdate DATE, amount INTEGER)

INSERT INTO subject\_table VALUES (:thisdate, :amount);

# Example : Cascaded Triggers

This example demonstrates how one triggered action statement can cause another trigger to fire.These are the table definitions.

CREATE TABLE tab1 (

a INTEGER,

b INTEGER,

c INTEGER);

CREATE TABLE tab2 (

d INTEGER,

e INTEGER,

f INTEGER);

CREATE TABLE tab3 (

g INTEGER,

h INTEGER,

i INTEGER);

These are the trigger definitions.

CREATE TRIGGER trig1

AFTER INSERT ON tab1

REFERENCING NEW AS NewRow

FOR EACH ROW (

INSERT INTO tab2

VALUES (NewRow.a + 10, NewRow.b + 10, NewRow.c););

CREATE TRIGGER trig2

AFTER INSERT ON tab2

REFERENCING NEW AS NewRow

FOR EACH ROW (

INSERT INTO tab3

VALUES (NewRow.d + 100, NewRow.e + 100, NewRow.f););

Now, suppose the following INSERT request is submitted:

INSERT INTO tab1VALUES (1,2,3);

NOTE: This INSERT has fired trigger trig1 and that trigger has fired trig2, hence the remaining two table tab2 , tab3 got populated with the data.

select \* from tab1;

\*\*\* Query completed. One row found. 3 columns returned.

\*\*\* Total elapsed time was 1 second.

a b c

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

1 2 3

select \* from tab2;

\*\*\* Query completed. One row found. 3 columns returned.

\*\*\* Total elapsed time was 1 second.

d e f

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

11 12 3

select \* from tab3;

\*\*\* Query completed. One row found. 3 columns returned.

\*\*\* Total elapsed time was 1 second.

g h i

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

111 112 3

BTEQ -- Enter your DBC/SQL request or BTEQ command:

This triggeringevent fires a trigger (trig1) to insert into tab2. This operation is equivalent to the following INSERT request:

INSERT INTO tab2VALUES (11,12,3);

This triggering event fires a trigger (trig2) to insert into tab3. This operation is equivalent to the following INSERT request:

INSERT INTO tab3 VALUES (111,112,3);

# Example : Valid WHEN Clause

The following WHEN clause is valid because an aggregate appears on the right-hand side of the search condition, and the left-hand side of the inequality condition is a constant.

DROP TRIGGER Trig3;

CREATE TRIGGER Trig3

AFTER INSERT ON tab3

REFERENCING NEW AS NewRow

FOR EACH ROW

WHEN (10 > (SELECT a FROM tab1))

ABORT;

If you insert values into t1 and the WHEN condition is satisfied, then the triggered action statement, ABORT, performs and a failure message is returned.

INSERT INTO tab3 VALUES (222,333,444);

\*\*\* Failure 3514 User-generated transaction ABORT.

Statement# 1, Info =0

\*\*\* Total elapsed time was 1 second.

# Creating and Using Stored Procedures

# Database Privileges Required

You must have the following additional database privileges to create or use a stored procedure:

• CREATE PROCEDURE privilege

• Access to the appropriate database elements. (You cannot create a procedure that references a database table for which you do not have access permission.)

To replace or modify an existing procedure, you must have the DROP privilege on the procedure or DROP PROCEDURE privilege on the database containing it.

Also, after you have created a procedure, you must grant EXECUTE privilege on the procedure or EXECUTE PROCEDURE privilege on that database that contains it, to any other users before they can execute the procedure.

Note: The EXECUTE privilege allows users to execute macros, while EXECUTE PROCEDURE is specific to stored procedures.

# Syntax:

CREATE PROCEDURE procedureName (IN/OUT/INOUT fields with datatypes)

BEGIN

SQL statements;

END;

# Types of Stored Procedures

BTEQ supports external and non-external stored procedures.

An external stored procedure is defined with a CREATE PROCEDURE or REPLACE PROCEDURE statement, along with EXTERNAL references for source files.

For example:

CREATE PROCEDURE xyz(

IN data1 INTEGER,

OUT data2 INTEGER)

LANGUAGE CPP NO SQL PARAMETER STYLE TD\_GENERAL

EXTERNAL NAME 'CS!xyz!source.cpp';

The C or C++ files can reside on the server or the client. The server will call for the transfer of client-resident files as needed to satisfy input requirements for the CREATE PROCEDURE or REPLACE PROCEDURE request. Source files must be encoded as ASCII (workstation) or EBCDIC (mainframe), no matter what the current session character set is.

A non-external stored procedure is created or replaced using the COMPILE command, which takes a single Stored Procedure Language (SPL) file as input. The encoding of this file must match that of the current session character set.

# Command Execution

Specify the following command to create stored procedures on the Teradata server:

.COMPILE FILE = <file-path>\example.spl;

As BTEQ processes the COMPILE command, it uses the specified file’s content as the stored procedure source text input. It sends the stored procedure source text to the Teradata server for processing the stored procedure definition.

# Creating or Replacing a Stored Procedure

To create a stored procedure, invoke the text editor on your system, create a file named example.txt, and enter the stored procedure source text:

CREATE PROCEDURE samplesp1 ()

BEGIN

/\* SPL Statements\*/

DECLARE V1 INTEGER;

SET V1 = 100;

END;

BTEQ -- Enter your SQL request or BTEQ command:

.COMPILE FILE = C:\Users\Administrator\Desktop\Teradata\_Capgemini\Procedure\example.txt

\*\*\* Procedure has been created.

\*\*\* Total elapsed time was 1 second.

BTEQ -- Enter your SQL request or BTEQ command:

CALL samplesp1 ();

\*\*\* Procedure has been executed.

\*\*\* Total elapsed time was 1 second.

BTEQ -- Enter your SQL request or BTEQ command:

The preceding example creates a procedure without any parameters.

To replace this with another stored procedure with parameters, create a file named example1.txt, and enter the stored procedure source text:

Create table saving\_account(

Acc\_No integer,

Amount decimal(10,2));

\*\*\* Table has been created.

\*\*\* Total elapsed time was 1 second.

INSERT INTO saving\_account VALUES(7777,30000);

INSERT INTO saving\_account VALUES(8888,50000);

INSERT INTO saving\_account VALUES(9999,80000);

REPLACE PROCEDURE samplesp1 (IN pAcc\_No INTEGER, OUT pAmount DECIMAL (10,2))

BEGIN

/\* SPL Statements\*/

SELECT Amount INTO :pAmount

FROM Saving\_Account

WHERE Acc\_No = :pAcc\_No;

END;

Submit the COMPILE command with this file name.

.COMPILE FILE = C:\Users\Administrator\Desktop\Teradata\_Capgemini\Procedure\example1.txt

\*\*\* Procedure has been replaced.

\*\*\* Total elapsed time was 1 second.

# Executing Stored Procedures

You can execute stored procedures either interactively, or include them in BTEQ scripts or input stream files.

Use the Teradata SQL CALL statement to execute a stored procedure interactively:

CALL samplesp1 (8888, pAmount);

\*\*\* Procedure has been executed.

\*\*\* Total elapsed time was 1 second.

pAmount

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

50000.00

BTEQ -- Enter your SQL request or BTEQ command:

If the procedure is created with parameters, you must specify the parameter arguments within the parenthesis.

# Including a Stored Procedure in a Script

Use the text editor on your system to include a stored procedure in a BTEQ script or input stream file. To execute the procedure samplesp1 from a script, for example, enter the Teradata SQL CALL statement after the LOGON statement in the script file named logon\_script.txt

.LOGON 10.74.161.248/td\_user1,td\_user1

CALL samplesp1 (8888, pAmount);

.LOGOFF

C:\Users\haryadav>bteq < D:\Users\haryadav\Downloads\logon\_script.txt

BTEQ 15.10.01.01 Fri Jan 27 09:30:35 2017 PID: 24020

+---------+---------+---------+---------+---------+---------+---------+--

.LOGON 10.74.161.248/td\_user1,

\*\*\* Logon successfully completed.

\*\*\* Teradata Database Release is 13.10.07.12

\*\*\* Teradata Database Version is 13.10.07.12

\*\*\* Transaction Semantics are BTET.

\*\*\* Session Character Set Name is 'ASCII'.

\*\*\* Total elapsed time was 1 second.

+---------+---------+---------+---------+---------+---------+---------+--

CALL samplesp1 (8888, pAmount);

\*\*\* Procedure has been executed.

\*\*\* Total elapsed time was 1 second.

pAmount

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

50000.00

+---------+---------+---------+---------+---------+---------+---------+--

.LOGOFF

\*\*\* You are now logged off from the DBC.

+---------+---------+---------+---------+---------+---------+---------+--

\*\*\* BTEQ exiting due to EOF on stdin.

\*\*\* Exiting BTEQ...

\*\*\* RC (return code) = 0

# Creating and Executing a Stored Procedure from a Script

The following example shows how to create and execute a stored procedure from the script file, assuming that the script is running in ANSI mode. The example.spl file contains the stored procedure source text.

.set session transaction ansi;

.LOGON 10.74.161.248/td\_user1,td\_user1

.COMPILE FILE = C:\Users\Administrator\Desktop\Teradata\_Capgemini\Procedure\example1.txt

COMMIT;

CALL samplesp1 (8888, pAmount);

.LOGOFF

C:\Users\haryadav>bteq < D:\Users\haryadav\Downloads\logon\_script.txt

BTEQ 15.10.01.01 Fri Jan 27 09:45:39 2017 PID: 1068

+---------+---------+---------+---------+---------+---------+---------+--

.set session transaction ansi;

+---------+---------+---------+---------+---------+---------+---------+--

.LOGON 10.74.161.248/td\_user1,

\*\*\* Logon successfully completed.

\*\*\* Teradata Database Release is 13.10.07.12

\*\*\* Teradata Database Version is 13.10.07.12

\*\*\* Transaction Semantics are ANSI.

\*\*\* Session Character Set Name is 'ASCII'.

\*\*\* Total elapsed time was 1 second.

+---------+---------+---------+---------+---------+---------+---------+--

.COMPILE FILE = C:\Users\Administrator\Desktop\Teradata\_Capgemini\Procedure\example1.txt

\*\*\* Procedure has been replaced.

\*\*\* Total elapsed time was 1 second.

+---------+---------+---------+---------+---------+---------+---------+--

COMMIT;

\*\*\* COMMIT done.

\*\*\* Total elapsed time was 1 second.

+---------+---------+---------+---------+---------+---------+---------+--

CALL samplesp1 (8888, pAmount);

\*\*\* Procedure has been executed.

\*\*\* Total elapsed time was 1 second.

pAmount

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

50000.00

+---------+---------+---------+---------+---------+---------+---------+--

.LOGOFF

\*\*\* You are now logged off from the DBC.

+---------+---------+---------+---------+---------+---------+---------+--

\*\*\* BTEQ exiting due to EOF on stdin.

\*\*\* Exiting BTEQ...

\*\*\* RC (return code) = 0

# STORED PROCEDURE in Teradata

Assume you want to define a stored procedure 'SP\_Employee' to add new employees to the Employee table and retrieve the name of the department to which the employee belongs.

You can also report an error, in case the row that you are trying to insert already exists, and handle that error condition.

# Example 1.

The CREATE PROCEDURE statement looks like this:

CREATE TABLE department(

DeptNo integer,

DeptName char(20),

Location char(20));

CREATE PROCEDURE InsertDepartment(

IN DeptNo integer,

IN DeptName char(20),

IN Location char(20))

BEGIN

INSERT INTO Department(DeptNo, DeptName, Location)

VALUES ( :DeptNo, :DeptName, :Location);

END;

.set width 500

.compile file=C:\Users\Administrator\Desktop\Teradata\_Capgemini\Procedure\Procedure.txt

\*\*\* One Error/Warning.

\*\*\* Warning: 5526 Stored Procedure is not created/replaced due to error(s).

\*\*\* Total elapsed time was 1 second.

Errors reported during compilation

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

**SPL1040:E(L2), Invalid definition for stored procedure 'E(3524):The user does not have CREATE PROCEDURE access to database financial.'.**

select user;

\*\*\* Query completed. One row found. One column returned.

\*\*\* Total elapsed time was 1 second.

User

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

TD06

BTEQ -- Enter your DBC/SQL request or BTEQ command:

.logon localtd/dbc

Password:

\*\*\* Logon successfully completed.

\*\*\* Teradata Database Release is 13.00.00.12

\*\*\* Teradata Database Version is 13.00.00.12

\*\*\* Transaction Semantics are BTET.

\*\*\* Character Set Name is 'ASCII'.

\*\*\* Total elapsed time was 1 second.

BTEQ -- Enter your DBC/SQL request or BTEQ command:

grant create procedure, drop procedure, execute procedure on financial to td06;

\*\*\* Grant accepted.

\*\*\* Total elapsed time was 1 second.

.logon localtd/td06

Password:

\*\*\* Logon successfully completed.

\*\*\* Teradata Database Release is 13.00.00.12

\*\*\* Teradata Database Version is 13.00.00.12

\*\*\* Transaction Semantics are BTET.

\*\*\* Character Set Name is 'ASCII'.

\*\*\* Total elapsed time was 1 second.

.compile file C:\Users\Administrator\Desktop\Teradata\_Capgemini\Teradata\_Doc\Procedure.txt

\*\*\* Error 5862 C/C++ compiler is not installed.

Statement# 29958, Info =224

\*\*\* Total elapsed time was 1 second.

CALL InsertDepartment(10, 'ACCOUNTING', 'PUNE');

CALL InsertDepartment(20, 'SALES', 'CHENNAI');

CALL InsertDepartment(30, 'MARKETING', 'BANGALORE');

CALL InsertDepartment(40, 'DEVELOPMENT', 'PUNE');

CALL InsertDepartment(50, 'TESTING', 'PUNE');

SELECT \* FROM department ORDER BY 1;

\*\*\* Query completed. 5 rows found. 3 columns returned.

\*\*\* Total elapsed time was 1 second.

DeptNo DeptName Location

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

10 ACCOUNTING PUNE

20 SALES CHENNAI

30 MARKETING BANGALORE

40 DEVELOPMENT PUNE

50 TESTING PUNE

CREATE TABLE employee(

EmpName char(20),

EmpNo integer,

DeptNo integer);

CREATE PROCEDURE SP\_Employee(

IN name CHAR(20),

IN id INTEGER,

IN dept\_no INTEGER,

OUT dname CHAR(20),

INOUT errstr VARCHAR(30))

BEGIN

DECLARE CONTINUE HANDLER FOR SQLSTATE VALUE '23505'

SET errstr = 'Duplicate Row.';

INSERT INTO Employee (EmpName, EmpNo, DeptNo )

VALUES (:name, :id, :dept\_no);

SELECT DeptNameINTO dname FROM Department

WHERE DeptNo = :dept\_no;

END;

CALL <procedure name> [(parameter values)];

CALL SP\_Employee('DEV',1111,20,DNAME,ERRSTR);

# Example 2.

DROP TABLE us\_states;

CREATE MULTISET VOLATILE TABLE us\_states , NO JOURNAL, NO LOG,NO FALLBACK

(

STATE\_NAME VARCHAR(13),

STATE\_ABBR CHAR(2),

STATE\_CAPITAL VARCHAR(13),

STATE\_CITY VARCHAR(13)

)

PRIMARY INDEX(STATE\_NAME)

ON COMMIT PRESERVE ROWS

;

Now let’s create a Stored Procedure which will insert some rows into the table.

CREATE PROCEDURE US\_STATE\_1 ()

BEGIN

INSERT INTO US\_STATES VALUES('Alabama','AL','Montgomery','Birmingham');

INSERT INTO US\_STATES VALUES('Alaska','AK','Juneau','Anchorage');

INSERT INTO US\_STATES VALUES('Arizona','AZ','Phoenix','Phoenix');

INSERT INTO US\_STATES VALUES('Arkansas','AR','LittleRock','LittleRock');

INSERT INTO US\_STATES VALUES('California','CA','Sacramento','LosAngeles');

END;

To run the proc:

CALL US\_STATE\_1();

Once done, you can see message : CALL completed.

Now check the data in the table.

SELECT \* FROM US\_STATES;

If your proc was successfully compiled and successfully completed, then you should be able to see 5 rows in the table.