**Stored Procedure and Triggers**

**Sem-IV**

**(2020-2021)**

**BY**

**MRS. PALLAVI V. MIRJKAR**

**Asst. Prof.**

**Dept. of Computer Science**

**Introduction**

A procedure (often called a stored procedure) is a subroutine like a subprogram in a regular computing language, stored in database. There are many useful applications of SQL procedures within a database or database application architecture. SQL procedures can be used to create simple scripts for quickly querying transforming, updating data, generating basic reports, improve application performance, modularizing applications, and improve overall database design, and database security.

**Features of SQL procedures**

* Can contain SQL Procedural Language statements and features which support the implementation of control-flow logic around traditional static and dynamic SQL statements.
* Easy to implement, because they use a simple high-level, strongly-typed language.
* SQL procedures are more reliable than equivalent external procedures.
* Support input, output, and input-output parameter passing modes.
* Support a simple, but powerful condition and error-handling model.
* Return multiple results sets to the caller or to a client application.
* Allow you to easily access the SQLSTATE and SQLCODE values as special variables.
* Reside in the database and are automatically backed up and restored.
* Can be invoked wherever the CALL statement is supported.
* Support nested procedure calls to other SQL procedures or procedures implemented in other languages.

**Disadvantages**

* Stored procedure languages are vendor specific. Therefore if you switch to another vendor's database, it requires to rewriting the existing stored procedures.
* Stored procedure languages from different vendors have different levels of sophistication. For example, Oracle's PL/SQL has more language features and built-in features than Microsoft's T-SQL.
* Tool support for writing and debugging stored procedures is often not as good as for other programming languages, though it depends on vendors and languages.

**Defining an SQL procedure**

The CREATE PROCEDURE statement for SQL procedures :

* Names the procedure
* Creates the stored procedure
* Defines the parameters and their attributes
* Provides other information about the procedure which will be used when the procedure is called
* Defines the procedure body

Here is the complete syntax of CREATE PROCEDURE

*CREATE [OR REPLACE] PROCEDURE proc\_name [list of parameters]*

*IS*

 *Declaration section*

*BEGIN*

 *Execution section*

*EXCEPTION*

 *Exception section*

*END;*

### Procedures: Passing Parameters

We can pass parameters to procedures in three ways.
1) IN-parameters
2) OUT-parameters
3) IN OUT-parameters

A procedure may or may not return any value.

**IS -** marks the beginning of the body of the procedure and is similar to DECLARE in anonymous PL/SQL Blocks. The code between IS and BEGIN forms the Declaration section.

The syntax within the brackets [ ] indicate they are optional. By using CREATE OR REPLACE together the procedure is created if no other procedure with the same name exists or the existing procedure is replaced with the current code.

### Procedures: Example

1. **Create an empty procedure or replace a procedure with Hello World and execute the same.**

1 CREATE PROCEDURE MANDY1

 2 IS

 3 BEGIN

 4 NULL;

 5\* END;

SQL> /

Procedure created.

PL/SQL BLOCK:

 1 CREATE OR REPLACE PROCEDURE MANDY1 IS

 2 BEGIN

 3 DBMS\_OUTPUT.PUT\_LINE('HELLO WORLD');

 4\* END;

 5 /

Procedure created.

**Output:-**

**SQL> EXEC MANDY1();**

**HELLO WORLD**

**PL/SQL procedure successfully completed.**

### How to execute a Stored Procedure?

There are two ways to execute a procedure.

1) From the SQL prompt.

 *EXECUTE [or EXEC] procedure\_name;*

2) Within another procedure – simply use the procedure name.

 *procedure\_name;*

**Triggers: Program**

**TRIGGER: creating and working with**

1. **Insert/update/Delete Trigger**

**Insert:**

SQL> select \* from employee;

 EMPNO ENAME SALARY

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

 1 Alice 5000

 2 Bob 6000

 3 john 7000

 4 jay 8000

 4 jay 8000

SQL> create or replace trigger intrigger

 2 before insert on employee

 3 begin

 4 if inserting then

 5 dbms\_output.put\_line('do you want to insert it');

 6 end if;

 7 end;

 8 /

Trigger created.

SQL> connect/as sysdba

Connected.

SQL> grant create trigger to system;

Grant succeeded.

SQL> connect

Enter user-name: system

Enter password:

Connected.

SQL> set serveroutput on;

SQL> insert into employee values(7,'jack',2000);

do you want to insert it

1 row created.

 **UPDATE**

SQL> create or replace trigger uptrigger

 2 before update on employee

 3 begin

 4 if updating then

 5 dbms\_output.put\_line('do you want to update it');

 6 end if;

 7 end;

 8 /

Trigger created.

SQL> connect/as sysdba

Connected.

SQL> grant create trigger to system;

Grant succeeded.

SQL> connect

Enter user-name: system

Enter password:

Connected.

SQL> set serveroutput on;

SQL> update employee set ename=’joy’ where empno=3;

do you want to update it

1 row updated.

**DELETE**

SQL> create or replace trigger deltrigger

 2 before delete on employee

 3 begin

 4 if deleting then

 5 dbms\_output.put\_line('do you want to delete it');

 6 end if;

 7 end;

 8 /

Trigger created.

SQL> connect/as sysdba

Connected.

SQL> grant create trigger to system;

Grant succeeded.

SQL> connect

Enter user-name: system

Enter password:

Connected.

SQL> set serveroutput on;

SQL>delete from employee where empno=3;

do you want to delete it

1 row deleted.

**Update /delete/insert**

 SQL> CREATE OR REPLACE TRIGGER tr\_superheroes

 2  BEFORE INSERT OR DELETE OR UPDATE ON employee

 3  FOR EACH ROW

 4  ENABLE

 5  DECLARE

 6   v\_user VARCHAR2(15);

 7  BEGIN

 8  SELECT

 9   user INTO v\_user FROM dual;

 10  IF INSERTING THEN

 11   DBMS\_OUTPUT.PUT\_LINE('one line inserted by '||v\_user);

 12  ELSIF DELETING THEN

 13   DBMS\_OUTPUT.PUT\_LINE('one line Deleted by '||v\_user);

 14  ELSIF UPDATING THEN

 15   DBMS\_OUTPUT.PUT\_LINE('one line Updated by '||v\_user);

 16  END IF;

 17  END;

 18  /

**ROW LEVEL TRIGGER AFTER INSERT**

SQL> CREATE TABLE orders

 2 (order\_id number(5),

 3 quantity number(4),

 4 cost\_per\_itemnumber(20),

 5 total\_costnumber(8,2)

6 );

Table created.

SQL> CREATE OR REPLACE TRIGGERorders\_after\_insert

2 AFTER INSERT ON orders

3 FOR EACH ROW

4 DECLARE

5 v\_username varchar2(10);

6 BEGIN

7 SELECT user INTO v\_username FROM dual;

8 Dbms\_output.put\_line(v\_username||'records inserted');

9 END;

10 /

Trigger created.

SQL> insert into orders values(2,7,55,34);

1 row created.

SQL> select \* from orders;

 ORDER\_ID QUANTITY COST\_PER\_ITEM TOTAL\_COST

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

 2 7 55 34

**ALTER UPDATE**

SQL> create table orders\_audit(

2 order\_id number,

3 quantity\_before number,

4 quantity\_after number,

5 username varchar(50));

Table created.

SQL> CREATE OR REPLACE TRIGGER orders\_after\_update

 2 AFTER UPDATE ON orders

 3 FOR EACH ROW

 4 DECLARE

 5 v\_username varchar2(10);

 6 BEGIN

 7 SELECT user INTO v\_username

 8 FROM dual;

 9 INSERT INTO orders\_audit

 10 (order\_id,

 11 quantity\_before,

 12 quantity\_after,

 13 username)

 14 VALUES

 15 (:new.order\_id,

 16 :old.quantity,

 17 :new.quantity,

 18 v\_username);

 19 END;

 20 /

Trigger created.

SQL> update orders set quantity=15 where order\_id=2;

1 row updated.

SQL> select \* from orders\_audit;

 ORDER\_ID QUANTITY\_BEFORE QUANTITY\_AFTER USERNAME

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

 2 7 15 SYSTEM

**AFTER DELETE**

SQL> create table orders\_audit\_del

2 (

3 order\_id number,

4 quantity number,

5 delete\_date date,

6 deleted\_by varchar(50));

Table created.

SQL> CREATE OR REPLACE TRIGGER orders\_after\_delete

2 AFTER DELETE

3 ON orders

4 FOR EACH ROW

5 DECLARE

6 v\_username varchar2(20);

7 BEGIN

8 SELECT user INTO v\_username FROM dual;

9 INSERT INTO orders\_audit\_del(order\_id,quantity,delete\_date,deleted\_by)

10 VALUES(:old.order\_id,:old.quantity,sysdate,v\_username);

11 END;

12 /

Trigger created.

SQL> delete from orders where order\_id=2;

1 row deleted.

SQL> select \* from orders\_audit\_del;

 ORDER\_ID QUANTITY DELETE\_DA DELETED\_BY

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

 2 16 03-FEB-18 SYSTEM

**BEFORE INSERT**

SQL> CREATE TABLE orders

2 (order\_id number(5),

 3 quantity number(4),

 4 cost\_per\_itemnumber(6,2),

 5 total\_costnumber(8,2),

 6 create\_date date,

 7 created\_by varchar2(10)

8 );

Table created.

SQL> CREATE OR REPLACE TRIGGER orders\_before\_insert

2 BEFORE INSERT

 3 ON orders

 4 FOR EACH ROW

 5

6 DECLARE

 7 v\_username varchar2(10);

 8

9 BEGIN

 10 -- Find username of person performing INSERT into table

 11 SELECT user INTO v\_username FROM dual;

 12

 13 -- Update create\_date field to current system date

 14 :new.create\_date := sysdate;

 15

 16 -- Update created\_by field to the username of the person performingthe INSERT

 17 :new.created\_by := v\_username;

 18

19 END;

20 /

Trigger created.

SQL> set serveroutput on;

SQL> insert into orders( order\_id,quantity,cost\_per\_item,total\_cost) values(1,20,45,500);

1 row created.

SQL> select \* from orders;

 ORDER\_ID QUANTITY COST\_PER\_ITEM TOTAL\_COST CREATE\_DA CREATED\_BY

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

 1 20 45 500 09-FEB-18 DBSNMP

SQL>

**BEFORE UPDATE**

SQL> CREATE OR REPLACE TRIGGER orders\_before\_update

2 BEFORE UPDATE

 3 ON orders

 4 FOR EACH ROW

 5

6 DECLARE

 7 v\_username varchar2(10);

 8

9 BEGIN

 10 -- Find username of person performing INSERT into table

 11 SELECT user INTO v\_username FROM dual;

 12

 13 -- Update create\_date field to current system date

 14 :new.create\_date := sysdate;

 15

 16 -- Update created\_by field to the username of the person performingthe INSERT

 17 :new.created\_by := v\_username;

 18

19 END;

20 /

Trigger created.

SQL>

SQL> update orders set quantity=50 where order\_id=1;

1 row updated.

SQL> select \* from orders;

 ORDER\_ID QUANTITY COST\_PER\_ITEM TOTAL\_COST CREATE\_DA CREATED\_BY

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

 1 50 45 500 09-FEB-18 DBSNMP

SQL>

**BEFORE DELETE**

SQL> CREATE TABLE orders\_audit

2 (order\_id number(5),

 3 quantity number(4),

 4 cost\_per\_itemnumber(6,2),

 5 total\_costnumber(8,2),

 6 delete\_date date,

 7 deleted\_by varchar2(10)

8 );

Table created.

SQL> CREATE OR REPLACE TRIGGER orders\_before\_delete

2 BEFORE DELETE

 3 ON orders

 4 FOR EACH ROW

 5

6 DECLARE

 7 v\_username varchar2(10);

 8

9 BEGIN

 10 -- Find username of person performing the DELETE on the table

 11 SELECT user INTO v\_username FROM dual;

 12

 13 -- Insert record into audit table

 14 INSERT INTO orders\_audit( order\_id,quantity,cost\_per\_item,total\_cost,delete\_date,deleted\_by )

 15 VALUES( :old.order\_id,:old.quantity,:old.cost\_per\_item,:old.total\_cost,sysdate,v\_username );

 16

17 END;

18 /

Trigger created.

SQL> delete from orders where order\_id=1;

1 row deleted.

SQL> select \* from orders\_audit;

 ORDER\_ID QUANTITY COST\_PER\_ITEM TOTAL\_COST DELETE\_DA DELETED\_BY

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

 1 20 45 500 09-FEB-18 DBSNMP

SQL>

**STATEMENT LEVEL TRIGGER**

SQL> create table ordcrs(

2 order\_id number,

3 quantity number,

4 cost number);

Table created.

SQL> create table orders\_audit\_stat (

 2 order\_id number,

 3 username varchar(50));

Table created.

SQL> CREATE OR REPLACE TRIGGER orders\_after\_insert

 2 before INSERT

 3 ON ordcrs

 4 DECLARE

 5 v\_username varchar2(10);

 6 BEGIN

7 --Find username of person performing the INSERT into the table

 8 SELECT user INTO v\_username

9 FROM dual;

 10 -- Insert record into audit table

 11 INSERT INTO orders\_audit\_stat(order\_id,username)

 12 VALUES(100,v\_username);

 13 END;

 14 /

Trigger created.

SQL> insert into ordcrsvalues(3,77,900);

1 row created.

SQL> select \* from orders\_audit\_stat;

 ORDER\_ID USERNAME

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

 100 SYSTEM

**REMOVE TRIGGER**

SQL> drop trigger orders\_after\_insert;

Trigger dropped.