# SQL Stored Procedures and Triggers

Dr. Rudra Pratap Deb Nath [*]

Associate Professor
University of Chittagong, Chittagong

July 9, 2025

[*] Corresponding authors email: rudra@cu.ac.bd

# 1 Procedure

A procedure is a named block of code and is stored in the database for the re-usability purpose. This database object can be used to perform repeated execution. A procedure can include:

- SQL queries
- DDL, DML, DCL and TCL commands
- Collection types
- Cursors
- Loop and IF-Else statements
- Exception handling and so on.

## 1.1 Purpose of a procedure

Procedures give more power to SQL. Procedure can do things which SQL queries cannot. You can put multiple bundle of queries, put entire software business logics (data cleaning, validation, retrieval and much more) inside a procedure.

## 1.2 Syntax and examples to create procedures

```
SQL> create or replace procedure pr_name(p_1 varchar, p_2 int)
IS
        \\declare variables
begin
        \\business logic
end;
```

Let’s create a first simple test procedure

```
SQL> create or replace procedure test_proc
IS

begin
        dbms_output.put_line('First␣test␣procedure');
end;
```

Use the following command to see the output of printed by dbms_output.put_line() method.

```
SQL> set serveroutput on;
```

Execute the procedure with the following command.

```
SQL> execute test_proc();
```

The output will be First test procedure. Example with IN (*Actual* → *formal*), OUT (*Formal* → *Actual*), IN OUT (dual nature) parameter

```
SQL> create or replace procedure get_deptName
  (p_did IN departments.department_id%type,
  p_dname OUT departments.department_name%type)
  IS
  begin
    select department_name
    into p_dname
    from departments
    where department_id=p_did;
  end;
    /
```

Execute the procedure

```
SQL> declare
     d_name varchar(20);
     begin
        get_deptName(50,d_name);
        dbms_output.put_line(d_name);
     end;
     /
```

Example with IN OUT parameter

```
SQL> create or replace procedure format_phone_number
   (p_phone_no IN OUT varchar2) IS
   begin
    p_phone_no := ’(’|| SUBSTR(p_phone_no, 1, 3) ||’
       )’||
                       SUBSTR(p_phone_no, 4, 3) || ’-’
                           ||
                       SUBSTR(p_phone_no, 7);
  end;
  /
```

Execute the procedure

```
SQL> declare
     v_p_no varchar2(25);
     begin
        v_p_no:= ’8801778155342’
        format_phone_number(v_p_no)
        dbms_output.put_line(v_p_no);
     end;
     /
```

**A use case:** The owner of the business wants to automatically update its sales and product tables once items are sold.

```
SQL> create table products
(
        p_code varchar(10),
        p_name varchar(25),
        price float,
        quantity_remaining int,
        quantity_sold int,
        primary key(p_code)
);
```

```
SQL> create table sales
    (
       order_id int,
       order_date date,
       p_code varchar(10) references products(p_code),
       quantity_ordered int,
       sale_price      float,
       primary key (order_id)
    );
```

```
SQL>
insert into products values
    (’p1’, ’Note8’, ’110000’, 5, 195);
insert into sales
    values (1, sysdate, ’p1’, 100, ’11000000’);
```

```
insert into sales
    values  (2, sysdate-1, 'p1', 95,'10450000') ;
commit;
```

- For every Note8 sale, modify the database tables accordingly.

```
SQL>
create or replace procedure pr_sale (p_product_name varchar, p_quantity int)
IS
        v_product_code varchar(30);
        v_price float;
        v_cnt int;
        o_id int;

begin
        select count (*)
        into v_cnt
        from products
        where p_name=p_product_name
        and quantity_remaining >=p_quantity;



        if v_cnt > 0 then
                select p_code, price
                into v_product_code, v_price
                from products
                where p_name=p_product_name;

                select max(order_id)
                into o_id
                from sales;

                insert into sales values (o_id+1, sysdate, v_product_code, p_quantity, (v_price*p_quantity));

                update products
                set quantity_remaining= (quantity_remaining - p_quantity),
                        quantity_sold= (
                           quantity_sold +
                           p_quantity)
                where p_code=v_product_code;

                dbms_output.put_line('Product␣sold!'
                   );
        else
                dbms_output.put_line ('Out␣of␣Stock'
                   );
        end if;
end;
```

Execute the procedure

```
SQL>  execute pr_sale('Note8',1);
Product sold!
```

Creating a function in PLSQL

```
SQL>
create or replace function get_sal
(p_id in employees.employee_id%TYPE)
return number
IS
v_salary employees.salary%TYPE :=0;
begin
        select salary
        into v_salary
        from employees
        where employee_id = p_id;
        return v_salary;
end get_sal;
/
```

Invoking functions in SQL expressions:

```
SQL>
select employee_id, get_sal(employee_id)
from employees;
```

The user_procedures view lists all functions and procedures that are owned by the current user, along with their associated properties. We can run a query against this view and filter its results to just stored procedures:

```
SQL>  Select object_name
      from  user_procedures
      where  object_type = 'PROCEDURE';
```

The `all_procedures` view lists all functions and procedures that are accessible to the current user, along with associated properties:

```
SQL>  select owner,object_name
        from    all_procedures
        where  object_type = 'PROCEDURE';
```

# 2 Trigger

A trigger is a PL/SQL block associated with a table, view, schema, or the database. Unlike a stored procedure, you can enable and disable a trigger, but you cannot explicitly invoke it. While a trigger is enabled, the database automatically invokes it—that is, the trigger fires—whenever its triggering event occurs. While a trigger is disabled, it does not fire.

You can user trigger for: security, auditing, data integrity, referential integrity, table replication, computing derived data automatically, event logging, and so on.

Trigger can be built on application level(Front end) or database level. Database triggers fire whenever a data event (such as DML) or system event (such as logon or shutdown) occurs on a schema or database. The excessive use of triggers can result in complex interdependencies, which may be difficult to maintain in large applications.

A DML triggering statement contains:

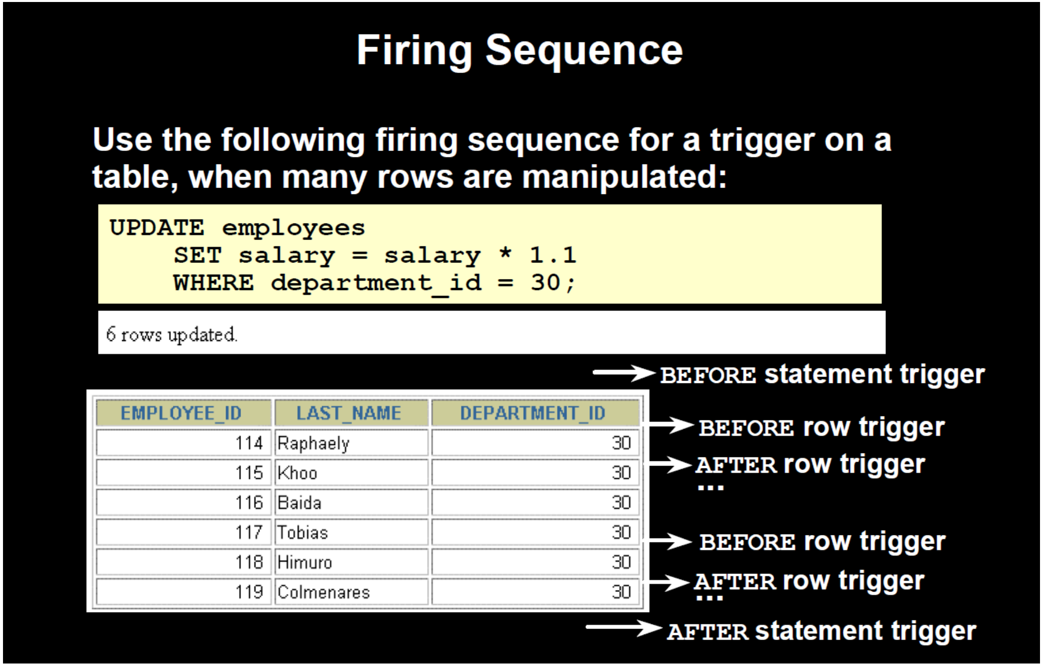
- Trigger timing
  - For table: Before, After
  - For view: Instead of
- Triggering event: Insert, update, or delete
- Table name: on table, view
- Trigger type: Row or statement
- When clause: Restricting condition
- Trigger body: A PL/SQL block.

Figure 1 shows the effect of row and statement trigger effects.

## 2.1 Syntax and examples of creating a DML level trigger

```
SQL> CREATE [OR REPLACE] TRIGGER trigger_name
timing
        event1 [OR event2 OR event3]
                ON table_name
trigger_body
```

Figure 1: Firing Sequence



## Firing Sequence

Use the following firing sequence for a trigger on a table, when many rows are manipulated:

```
UPDATE employees
    SET salary = salary * 1.1
    WHERE department_id = 30;
```

```
6 rows updated.
```

BEFORE statement trigger

| EMPLOYEE_ID | LAST_NAME | DEPARTMENT_ID |
|---|---|---|
| 114 | Raphaely | 30 |
| 115 | Khoo | 30 |
| 116 | Baida | 30 |
| 117 | Tobias | 30 |
| 118 | Himuro | 30 |
| 119 | Colmenares | 30 |

BEFORE row trigger
AFTER row trigger
...
BEFORE row trigger
AFTER row trigger
...
AFTER statement trigger

A test trigger with conditional predicates

```
SQL> CREATE OR REPLACE TRIGGER test
  BEFORE
    INSERT OR
    UPDATE OF salary, department_id OR
    DELETE
  ON employees
BEGIN
  CASE
    WHEN INSERTING THEN
      DBMS_OUTPUT.PUT_LINE('Inserting');
    WHEN UPDATING('salary') THEN
      DBMS_OUTPUT.PUT_LINE('Updating␣salary');
    WHEN UPDATING('department_id') THEN
      DBMS_OUTPUT.PUT_LINE('Updating␣department␣ID')
        ;
    WHEN DELETING THEN
      DBMS_OUTPUT.PUT_LINE('Deleting');
  END CASE;
END;
```

```
/
```

Controlling security with a database trigger.

```
SQL> CREATE OR REPLACE TRIGGER secure_emp
BEFORE INSERT OR UPDATE OR DELETE ON employees
BEGIN
        IF (TO_CHAR (SYSDATE,’DY’) IN (’SAT’,’SUN’))
           OR (TO_CHAR (SYSDATE, ’HH24’) NOT
          BETWEEN ’08’ AND ’18’)
        THEN
          IF DELETING THEN RAISE_APPLICATION_ERROR
             (-20502,’You␣may␣delete␣from␣EMPLOYEES␣
             table␣only␣during␣business␣hours.’);
          ELSIF INSERTING THEN
             RAISE_APPLICATION_ERROR (-20500,’You␣
             may␣insert␣into␣EMPLOYEES␣table␣only␣
             during␣business␣hours.’);
          ELSIF UPDATING (’SALARY’) THEN
             RAISE_APPLICATION_ERROR (-20503,’You␣
             may␣update␣SALARY␣only␣during␣business␣
             hours.’);
          ELSE
             RAISE_APPLICATION_ERROR (-20504,’You␣
                may␣update␣EMPLOYEES␣table␣only␣
                during␣normal␣hours.’);
          END IF;
        END IF;
END;
/
```

Creating a log entry for each update on salary

```
SQL> CREATE OR REPLACE TRIGGER log_salary_increase
  AFTER UPDATE OF salary ON employees
  FOR EACH ROW
BEGIN
  INSERT INTO Emp_log (Emp_id, Log_date, New_salary,
      Action)
  VALUES (:NEW.employee_id, SYSDATE, :NEW.salary, ’
     New␣Salary’);
END;
/
```

## 2.2 Triggers on system events

Syntax

```
SQL> CREATE [OR REPLACE] TRIGGER trigger_name
        timing
                [database_event1 [OR database_event2
                    OR ...]]
                ON {DATABASE|SCHEMA}
trigger_body
```

Logon and logoff trigger example

```
SQL> create table log_trig_table (
  user_id varchar2(50),
  log_date date,
  action varchar2(80),
  constraint log_pk primary key(user_id, log_date));

SQL>create or replace trigger logon_trig
  after logon on schema
  begin
  insert into log_trig_table values(user, sysdate, '
     logging␣on');
  end;
  /
SQL> create or replace trigger logoff_trig
  after logoff on schema
  begin
  insert into log_trig_table values(user, sysdate, '
     logging␣off');
  end;
  /
SQL> alter trigger logon_trig enable;
SQL> alter trigger logoff_trig enable;
SQL>connect test/test;
SQL>select * from sys.log_trig_table;

*** This will not work with sys user
```

## 2.3 How triggers and constraints differ

Both triggers and constraints can constrain data input, but they differ significantly.

A trigger always applies to new data only. For example, a trigger can prevent a DML statement from inserting a NULL value into a database column, but the column might contain NULL values that were inserted into the column before the trigger was defined or while the trigger was disabled.

A constraint can apply either to new data only (like a trigger) or to both new and existing data. Constraint behavior depends on constraint state, as explained in Oracle Database SQL Language Reference.

Constraints are easier to write and less error-prone than triggers that enforce the same rules. However, triggers can enforce some complex business rules that constraints cannot. Oracle strongly recommends that you use triggers to constrain data input only in these situations:

- To enforce referential integrity when child and parent tables are on different nodes of a distributed database
- To enforce complex business or referential integrity rules that you cannot define with constraints

Protecting data integrity with a trigger

```
SQL> CREATE OR REPLACE TRIGGER check_salary
  BEFORE UPDATE OF salary ON employees
  FOR EACH ROW
   WHEN (NEW.salary < OLD.salary)
   BEGIN
     RAISE_APPLICATION_ERROR (-20508, 'Do␣not␣
        decrease␣salary.');
END;
/
```

A trigger for ON UPDATE CASCADE functionality.

```
SQL> CREATE OR REPLACE TRIGGER cascade_updates
  AFTER UPDATE OF department_id ON departments
  FOR EACH ROW
BEGIN
  UPDATE employees
  SET employees.department_id=:NEW.department_id
  WHERE employees.department_id=:OLD.department_id;
  UPDATE job_history
  SET department_id=:NEW.department_id
  WHERE department_id=:OLD.department_id;
END;
/
```

Data dictionary for triggers: user_triggers,

**Acknowledgement:**

- https://www.youtube.com/watch?v=yLR1w4tZ36I&t=3367s
- PL/SQL slides by Oracle University.