# Procedure and Privilege Management in MySQL and PostgreSQL

## MySQL

### Scenario

In MySQL, we have two users:

1. **procedure\_user**: This user has the privileges to create procedures and insert into a specific table.
2. **tester\_user**: This user does not have direct insert privileges on the table but can execute the procedure created by procedure\_user.

### Steps to Implement

1. **Create Users and Grant Privileges**

* CREATE USER 'procedure\_user'@'localhost' IDENTIFIED BY 'password';  
  CREATE USER 'tester\_user'@'localhost' IDENTIFIED BY 'password';  
    
  GRANT CREATE PROCEDURE, INSERT ON `your\_database`.`your\_table` TO 'procedure\_user'@'localhost';

1. **Create the Procedure**

* DELIMITER //  
    
  CREATE PROCEDURE your\_database.insert\_procedure(IN param1 INT, IN param2 VARCHAR(255))  
  BEGIN  
   INSERT INTO your\_table (column1, column2) VALUES (param1, param2);  
  END //  
    
  DELIMITER ;

1. **Grant Execution Privilege to tester\_user**

* GRANT EXECUTE ON PROCEDURE your\_database.insert\_procedure TO 'tester\_user'@'localhost';

1. **Test the Procedure Execution**

* -- As tester\_user  
  CALL your\_database.insert\_procedure(1, 'test');

With these steps, tester\_user can call the procedure but cannot insert directly into the table.

## PostgreSQL

### Scenario

In PostgreSQL, we have similar users:

1. **procedure\_user**: This user creates the procedure and has insert privileges on the table.
2. **tester\_user**: This user can execute the procedure but does not have direct insert privileges on the table.

### Concepts of Security Definer and Security Invoker

* **Security Definer**: The procedure runs with the privileges of the user who created it (the definer).
* **Security Invoker**: The procedure runs with the privileges of the user who invokes it (the invoker).

### Steps to Implement

1. **Create Users and Grant Privileges**

* CREATE USER procedure\_user WITH PASSWORD 'password';  
  CREATE USER tester\_user WITH PASSWORD 'password';  
    
  GRANT INSERT ON TABLE your\_table TO procedure\_user;

1. **Create the Procedure with Security Definer**

* CREATE OR REPLACE PROCEDURE insert\_procedure(param1 INT, param2 VARCHAR)  
  LANGUAGE plpgsql  
  SECURITY DEFINER  
  AS $$  
  BEGIN  
   INSERT INTO your\_table (column1, column2) VALUES (param1, param2);  
  END;  
  $$;  
    
  GRANT EXECUTE ON PROCEDURE insert\_procedure TO tester\_user;

With SECURITY DEFINER, tester\_user can execute the procedure using procedure\_user’s privileges.

1. **Create the Procedure with Security Invoker**

* CREATE OR REPLACE PROCEDURE insert\_procedure\_invoker(param1 INT, param2 VARCHAR)  
  LANGUAGE plpgsql  
  SECURITY INVOKER  
  AS $$  
  BEGIN  
   INSERT INTO your\_table (column1, column2) VALUES (param1, param2);  
  END;  
  $$;  
    
  GRANT INSERT ON TABLE your\_table TO tester\_user;

With SECURITY INVOKER, the procedure will use the privileges of the user executing the procedure. Therefore, tester\_user needs direct insert privileges on the table and tester\_user doesn’t need execution privilege on insert\_procedure.

### Detailed Explanation

#### Creating Users and Granting Privileges

* **MySQL**:
  + We create two users: procedure\_user and tester\_user.
  + procedure\_user is granted the privileges to create procedures and insert into the table.
  + tester\_user does not have direct insert privileges.
* **PostgreSQL**:
  + We create two users: procedure\_user and tester\_user.
  + procedure\_user is granted the privilege to insert into the table.
  + tester\_user does not have direct insert privileges initially.

#### Creating Procedures

* **MySQL**:
  + A procedure insert\_procedure is created by procedure\_user that inserts data into the table.
  + We use DELIMITER to change the default delimiter temporarily to allow the procedure creation.
* **PostgreSQL**:
  + Two procedures are created: one with SECURITY DEFINER and one with SECURITY INVOKER.
  + SECURITY DEFINER allows tester\_user to execute the procedure using procedure\_user’s privileges.
  + SECURITY INVOKER requires tester\_user to have direct insert privileges to execute the procedure.

#### Granting Execution Privileges

* **MySQL**:
  + tester\_user is granted the privilege to execute the procedure.
* **PostgreSQL**:
  + For SECURITY DEFINER, tester\_user is granted the privilege to execute the procedure.
  + For SECURITY INVOKER, tester\_user is granted the insert privilege on the table.

### Summary

* **MySQL**: Use GRANT EXECUTE to allow tester\_user to execute the procedure without direct insert privileges.
* **PostgreSQL**:
  + Use SECURITY DEFINER to allow tester\_user to execute the procedure using procedure\_user’s privileges.
  + Use SECURITY INVOKER to require tester\_user to have direct insert privileges to execute the procedure.

This comprehensive documentation covers the step-by-step implementation of procedures and privilege management in both MySQL and PostgreSQL, providing a clear understanding of the concepts and their practical applications.