

14 Stored Procedures

A group of statements grouped as a logical unit and stored in the database.

Accepts parameters and executes the T-SQL statements in the procedure, returns the result set if not just an update operation.

Benefits

1. **Performance:** Efficient as SPs are compiled and stored in executable form. The executable code is automatically cached.
2. **Reduced Network Traffic:** When SPs are used instead of T-SQL at the application level, only procedure name is passed.
3. **Reusable**
4. **Security:** Direct access to the tables can be removed, and SPs can also be encrypted so that the source code is not visible.

Modes of input Parameters

In MySQL, parameters in stored procedures can be declared with one of three modes: **IN**, **OUT**, or **INOUT**.

1. **IN:**

- The parameter is used to pass a value into the procedure.
- The value of the parameter is read-only within the procedure and cannot be modified.

2. **OUT:**

- The parameter is used to return a value from the procedure.
- The procedure can modify the parameter, and the modified value is returned to the caller.

3. **INOUT:**

- The parameter can both receive a value from the caller and return a value back to the caller.
- The procedure can read and modify the parameter.

Example:

```
DELIMITER //
```

```
CREATE PROCEDURE ExampleProcedure_2(  
    IN inParam INT,  
    OUT outParam INT,  
    INOUT inoutParam INT  
)  
BEGIN  
    -- Use inParam for input  
    SET outParam = inParam * 2; -- Set outParam based on inParam  
    SET inoutParam = inoutParam + 1; -- Modify inoutParam  
END //
```

```
DELIMITER ;
```

SQL

Calling the Example Procedure:

```
SET @in = 5;  
SET @out = 0;  
SET @inout = 10;  
  
CALL ExampleProcedure_2(@in, @out, @inout);  
  
SELECT @out AS outResult, @inout AS inoutResult;
```

SQL

- **@in** is passed to the procedure and is used but not modified.
- **@out** is set within the procedure and the modified value is returned.
- **@inout** is both passed to and modified by the procedure.

Writing Basic SPs

The default delimiter is `;` but in order to write SPs it needs to be changed.

1. Product By Name

```
DELIMITER $$  
  
CREATE PROCEDURE GetProductByNamePattern_2(  
    IN p_product_pattern VARCHAR(255)  
)  
BEGIN  
    SELECT product, brand, sale_price, market_price, rating FROM bb_products  
    WHERE product LIKE p_product_pattern;  
END $$  
  
DELIMITER ;
```

SQL

```
CALL GetProductByNamePattern_2('%apple%');
```

SQL

2. Top N products

```
DELIMITER //  
  
CREATE PROCEDURE GetTopNProductsByRating(  
    IN p_limit INT  
)  
BEGIN  
    SELECT * FROM bb_products  
    ORDER BY rating DESC  
    LIMIT p_limit;  
END //  
  
DELIMITER ;
```

SQL

3. 50% off!

```
DELIMITER //
```

SQL

```
CREATE PROCEDURE GetProductsMoreThan50PercentOff()
```

```
BEGIN
```

```
    SELECT *
```

```
    FROM BigBasket_Products
```

```
    WHERE sale_price < (0.5 * market_price);
```

```
END //
```

```
DELIMITER ;
```

```
CALL GetProductsMoreThan50PercentOff();
```

SQL

1. Insert a New Product

SQL

```
DELIMITER //
```



```
CREATE PROCEDURE InsertProduct(  
    IN p_product VARCHAR(255),  
    IN p_category VARCHAR(255),  
    IN p_sub_category VARCHAR(255),  
    IN p_brand VARCHAR(255),  
    IN p_sale_price INT,  
    IN p_market_price INT,  
    IN p_type VARCHAR(255),  
    IN p_rating FLOAT,  
    IN p_description TEXT  
)  
BEGIN  
    INSERT INTO BigBasket_Products (  
        product, category, sub_category, brand, sale_price, market_price, type,  
rating, description  
    ) VALUES (  
        p_product, p_category, p_sub_category, p_brand, p_sale_price,  
p_market_price, p_type, p_rating, p_description  
    );  
END //
```



```
DELIMITER ;
```

2. Update Product Details

SQL

```
DELIMITER //
```



```
CREATE PROCEDURE UpdateProduct(  
    IN p_index INT,  
    IN p_product VARCHAR(255),  
    IN p_category VARCHAR(255),  
    IN p_sub_category VARCHAR(255),  
    IN p_brand VARCHAR(255),  
    IN p_sale_price INT,  
    IN p_market_price INT,  
    IN p_type VARCHAR(255),  
    IN p_rating FLOAT,  
    IN p_description TEXT  
)  
BEGIN  
    UPDATE BigBasket_Products  
    SET product = p_product,  
        category = p_category,  
        sub_category = p_sub_category,  
        brand = p_brand,  
        sale_price = p_sale_price,  
        market_price = p_market_price,  
        type = p_type,  
        rating = p_rating,  
        description = p_description  
    WHERE `index` = p_index;  
END //
```



```
DELIMITER ;
```

3. Delete a Product

SQL

```
DELIMITER //
```

```
CREATE PROCEDURE DeleteProduct(  
    IN p_index INT  
)  
BEGIN  
    DELETE FROM BigBasket_Products  
    WHERE `index` = p_index;  
END //
```

```
DELIMITER ;
```

4. Get Product by Category

SQL

```
DELIMITER //
```

```
CREATE PROCEDURE GetProductsByCategory(  
    IN p_category VARCHAR(255)  
)  
BEGIN  
    SELECT * FROM BigBasket_Products  
    WHERE category = p_category;  
END //
```

```
DELIMITER ;
```

5. Get Products by Price Range

SQL

```
DELIMITER //
```

```
CREATE PROCEDURE GetProductsByPriceRange(  
    IN p_min_price INT,  
    IN p_max_price INT  
)  
BEGIN  
    SELECT * FROM BigBasket_Products  
    WHERE sale_price BETWEEN p_min_price AND p_max_price;  
END //
```

```
DELIMITER ;
```

These stored procedures allow for inserting, updating, deleting, and retrieving products based on specific criteria from the **BigBasket_Products** table.