SQL Server

Arctech Info Private Limited

STORED PROCEDURES

SQL Server stored procedures are used to group one or more Transact-SQL statements into logical units. The stored procedure is stored as a named object in the SQL Server Database Server.

When you call a stored procedure for the first time, SQL Server creates an execution plan and stores it in the cache. In the subsequent executions of the stored procedure, SQL Server reuses the plan to execute the stored procedure very fast with reliable performance.

Executing a stored procedure –

```
EXECUTE sp_name;
```

OR

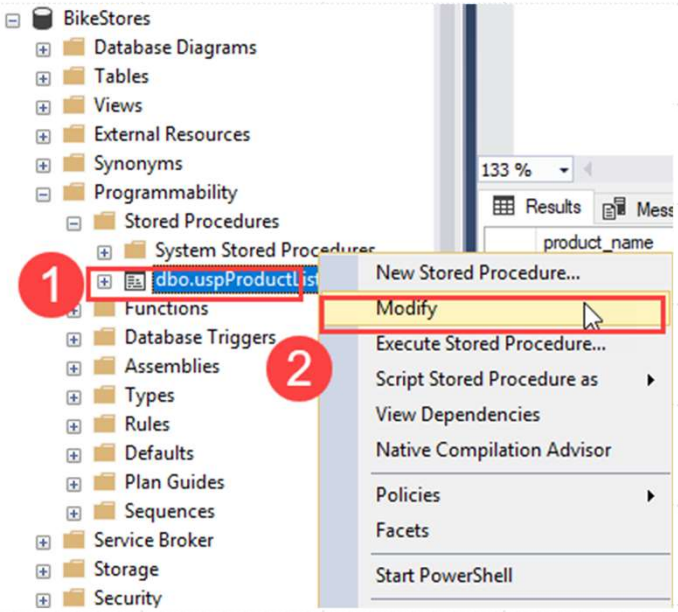
```
EXEC sp_name;
```

Stored Procedure

- The uspProductList is the name of the stored procedure.
- The AS keyword separates the heading and the body of the stored procedure.
- If the stored procedure has one statement, the BEGIN and END keywords surrounding the statement are optional. However, it is a good practice to include them to make the code clear.

```
CREATE PROCEDURE uspProductList
AS
BEGIN
    SELECT
        product_name,
        list_price
    FROM
        production.products
    ORDER BY
        product_name;
END;
```

Modifying stored procedure




```
ALTER PROCEDURE uspProductList
  AS
  BEGIN
     SELECT
        product_name,
        list_price
     FROM
        production.products
     ORDER BY
        list_price
  END;
```

Deleting a stored procedure

DROP PROCEDURE sp_name;

OR

DROP PROC sp_name;

- Example-
- DROP PROCEDURE uspProductList;

Stored procedure with parameter

- Every parameter must start with the @ sign.
- The AS DECIMAL keywords specify the data type of the @min_list_price parameter.
- The parameter must be surrounded by the opening and closing brackets.
- Executing stored procedure with parameter
- EXEC uspFindProducts 900,1000;
- To create an output parameter for a stored procedure, you use the following syntax:
- Syntax
  - parameter_name data_type OUTPUT

```
ALTER PROCEDURE uspFindProducts(
    @min_list_price AS DECIMAL
    ,@max_list_price AS DECIMAL
)
AS
BEGIN
    SELECT     product_name,  list_price
    FROM     production.products
    WHERE
        list_price >= @min_list_price AND
        list_price <= @max_list_price
    ORDER BY  list_price;
END;
```

Stored procedure with optional parameter

- Execute SP –

```
EXECUTE uspFindProducts
    @min_list_price = 6000,
    @name = 'Trek';
```

- OR

```
EXECUTE uspFindProducts
  @name = 'Trek';
```

```
ALTER PROCEDURE uspFindProducts(
    @min_list_price AS DECIMAL = 0
    ,@max_list_price AS DECIMAL = 999999
    ,@name AS VARCHAR(max)
)
AS
BEGIN
    SELECT    product_name,    list_price  FROM
production.products
    WHERE
        list_price >= @min_list_price AND
        list_price <= @max_list_price AND
        product_name LIKE '%' + @name + '%'
    ORDER BY   list_price;
END;
```

Variables

- A variable is an object that holds a single value of a specific type e.g., integer, date, or varying character string.
- Example –
- DECLARE @model_year SMALLINT;
- The DECLARE statement initializes a variable by assigning it a name and a data type. The variable name must start with the @ sign. In this example, the data type of the @model_year variable is SMALLINT.
- By default, when a variable is declared, its value is set to NULL.

Variables

- To assign a value to a variable, you use the SET statement.
- SET statement to assign the query's result set to the variable –

```
SET @product_count = (
    SELECT
        COUNT(*)
    FROM
        production.products
);
```

```
DECLARE @model_year SMALLINT;

SET @model_year = 2018;


SELECT product_name,  model_year,  list_price
FROM  production.products
WHERE  model_year = @model_year
ORDER BY
    product_name;
```

SQL Constraints

- SQL constraints are used to specify rules for the data in a table.
- Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.
- Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

- The following constraints are commonly used in SQL:
- NOT NULL - Ensures that a column cannot have a NULL value
- UNIQUE - Ensures that all values in a column are different
- PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
- FOREIGN KEY - Prevents actions that would destroy links between tables
- CHECK - Ensures that the values in a column satisfies a specific condition
- DEFAULT - Sets a default value for a column if no value is specified
- CREATE INDEX - Used to create and retrieve data from the database very quickly

Views

- When you use the SELECT statement to query data from one or more tables, you get a result set.
- Next time, if you want to get the same result set, you can save this query into a text file, open it, and execute it again.
- SQL Server provides a better way to save this query in the database catalog through a view.
- A view is a named query stored in the database catalog that allows you to refer to it later.
- So the query above can be stored as a view using the CREATE VIEW statement

```
CREATE VIEW sales.product_info
AS
SELECT
    product_name,
    brand_name,
    list_price
FROM
    production.products p
INNER JOIN production.brands b
        ON b.brand_id = p.brand_id;
```

- Can write select on views similar to tables
  - SELECT * FROM sales.product_info;

Advantages of views

- Security
  - You can restrict users to access directly to a table and allow them to access a subset of data via views. For example, you can allow users to access customer name, phone, email via a view but restrict them to access the bank account and other sensitive information.
- Simplicity
  - A relational database may have many tables with complex relationships e.g., one-to-one and one-to-many that make it difficult to navigate. However, you can simplify the complex queries with joins and conditions using a set of views.
- Consistency
  - Sometimes, you need to write a complex formula or logic in every query. To make it consistent, you can hide the complex queries logic and calculations in views. Once views are defined, you can reference the logic from the views rather than rewriting it in separate queries.

Transaction

- A transaction is a single unit of work that typically contains multiple T-SQL statements.
- If a transaction is successful, the changes are committed to the database. However, if a transaction has an error, the changes have to be rolled back.
- When executing a single statement such as INSERT, UPDATE, and DELETE, SQL Server uses the autocommit transaction. In this case, each statement is a transaction.
- To start a transaction explicitly, you use the BEGIN TRANSACTION or BEGIN TRAN statement first
- Finally, commit the transaction using the COMMIT statement
- Or roll back the transaction using the ROLLBACK statement

Transaction example

```
CREATE TABLE invoices (
  id int IDENTITY PRIMARY KEY,
  customer_id int NOT NULL,
  total decimal(10, 2) NOT NULL DEFAULT 0 CHECK (total >= 0)
);

CREATE TABLE invoice_items (
  id int,
  invoice_id int NOT NULL,
  item_name varchar(100) NOT NULL,
  amount decimal(10, 2) NOT NULL CHECK (amount >= 0),
  tax decimal(4, 2) NOT NULL CHECK (tax >= 0),
  PRIMARY KEY (id, invoice_id),
  FOREIGN KEY (invoice_id) REFERENCES invoices (id)  ON UPDATE CASCADE   ON
DELETE CASCADE
);
```

```
BEGIN TRANSACTION;

INSERT INTO invoices (customer_id, total)
VALUES (100, 0);

INSERT INTO invoice_items (id, invoice_id, item_name, amount, tax)
VALUES (10, 1, 'Keyboard', 70, 0.08),
       (20, 1, 'Mouse', 50, 0.08);

UPDATE invoices
SET total = (SELECT  SUM(amount * (1 + tax)) FROM invoice_items
WHERE invoice_id = 1);

COMMIT;
```