DatasetQuery Stored Procedures

Use stored procedures in dataset queries.

Reference **Modules** **Datasets** **UI** **DBs** DatasetQuery

---

DatasetQuery Stored Procedures (Referece) are precompiled SQL statements stored in the database that can be executed through Dataset Queries with parameters and return values. Stored Procedures enable:

- recompiled database logic
- Parameter passing (input/output)
- Return value handling
- Complex multi-step operations
- Transaction management
- Enhanced security through parameterization

Configuration in FrameworX
Execution Syntax
Execution Methods
Creating Stored Procedures
Parameter Types
Complete Example
Best Practices Checklist
Troubleshooting

- ## Configuration in FrameworX

Query Setup

1. Navigate to **Datasets** **Queries**
2. Create new query
3. Set **SqlStatement** to procedure call:

sql

```
EXEC ProcedureName @Param1 = {{Tag.Value1}}, @Param2 = {{Tag.Value2}}
```

Execution Syntax

Basic Execution

sql

```
-- Simple procedure
EXEC GetProductionData

-- With parameters
EXEC GetProductionByDate @Date = '{{Tag.QueryDate}}'

-- Multiple parameters
EXEC UpdateBatch
  @BatchID = {{Tag.Batch_ID}},
  @Status = {{Tag.Production_Status}},
  @Operator = '{{Tag.Operator_Name}}'
```

Output Parameters

sql

```
EXEC CalculateTotals
  @StartDate = '{{Tag.StartDate}}',
  @EndDate = '{{Tag.EndDate}}',
  @Total = {{Tag.TotalProduction}} OUTPUT,
  @Average = {{Tag.AverageRate}} OUTPUT
```

Return Values

sql

```
EXEC ValidateData
  @BatchID = {{Tag.Batch_ID}},
  @ReturnValue = {{Tag.ValidationResult}} RETURN_VALUE
```

Null Parameters

Use `@null@` for null values:

sql

```
EXEC UpdateRecord
  @ID = {{Tag.RecordID}},
  @Notes = @null@,
  @ModifiedBy = '{{Tag.Username}}'
```

Execution Methods

Using ExecuteCommand

csharp

```
// Execute stored procedure
@Dataset.Query.MyProcedure.ExecuteCommand();

// With status check
string status;
@Dataset.Query.MyProcedure.ExecuteCommandWithStatus(out status);

if (status == "OK")
{
    // Process success
}
```

Dynamic SQL Building

csharp

```
public void RunStoredProcedure()
{
    // Build procedure call
    string sql = string.Format(
        "EXEC GetDataByLine @Line = '{0}', @Date = '{1}'",
        @Tag.ProductionLine,
        DateTime.Now.ToString("yyyy-MM-dd")
    );

    @Dataset.Query.ProcedureQuery.SqlStatement = sql;

    // Execute
    string status;
    @Dataset.Query.ProcedureQuery.ExecuteCommandWithStatus(out status);

    if (status == "OK")
    {
        @Tag.ExecutionResult = "Success";
    }
}
```

Creating Stored Procedures

SQL Server Example

sql

```sql
CREATE PROCEDURE [dbo].[GetProductionSummary]
    @StartDate datetime,
    @EndDate datetime,
    @LineID int = NULL,
    @TotalCount int OUTPUT,
    @AvgRate float OUTPUT
AS
BEGIN
    SET NOCOUNT ON;

    -- Calculate totals
    SELECT @TotalCount = COUNT(*),
           @AvgRate = AVG(ProductionRate)
    FROM Production
    WHERE Timestamp BETWEEN @StartDate AND @EndDate
      AND (@LineID IS NULL OR LineID = @LineID);

    -- Return detailed data
    SELECT LineID,
           SUM(Count) as Total,
           AVG(Rate) as AvgRate
    FROM Production
    WHERE Timestamp BETWEEN @StartDate AND @EndDate
      AND (@LineID IS NULL OR LineID = @LineID)
    GROUP BY LineID;

    RETURN 0; -- Success
END
```

PostgreSQL Example

sql

```sql
CREATE OR REPLACE FUNCTION get_production_summary(
    start_date timestamp,
    end_date timestamp,
    line_id integer DEFAULT NULL
)
RETURNS TABLE(
    line_id integer,
    total_count bigint,
    avg_rate numeric
)
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN QUERY
    SELECT p.line_id,
           COUNT(*) as total_count,
           AVG(p.rate) as avg_rate
    FROM production p
    WHERE p.timestamp BETWEEN start_date AND end_date
      AND (line_id IS NULL OR p.line_id = line_id)
    GROUP BY p.line_id;
END;
$$;
```

Parameter Types

Type	SQL Server	PostgreSQL	MySQL
Input	`@param`	`param_name`	`IN param`
Output	`@param OUTPUT`	`OUT param`	`OUT param`
Input/Output	`@param OUTPUT`	`INOUT param`	`INOUT param`
Return	`RETURN`	`RETURNS`	N/A

Complete Example

1. Database Setup

sql

```sql
-- Create tables
CREATE TABLE Characteristic (
    Char_Id int PRIMARY KEY,
    Char_Desc varchar(25),
    OptionsChar varchar(25)
);

CREATE TABLE Specification (
    Spec_ID int PRIMARY KEY,
    Spec_Desc varchar(25),
    OptionsSpec varchar(25)
);

CREATE TABLE Active_Spec (
    AS_ID int PRIMARY KEY,
    Char_ID int,
    Spec_ID int,
    OptionsAS varchar(25)
);
```

2. Create Procedure

sql

```sql
CREATE PROCEDURE [dbo].[TestSP]
    @Char_Desc varchar(50)
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @Char_Id int;

    SELECT @Char_Id = Char_Id
    FROM Characteristic
    WHERE Char_Desc = @Char_Desc;

    SELECT c.Char_Desc,
           s.Spec_Id,
           s.Spec_Desc,
           c.OptionsChar,
           s.OptionsSpec,
           asp.OptionsAS
    FROM Specification s
    JOIN Active_Spec asp ON asp.Spec_Id = s.Spec_Id
    JOIN Characteristic c ON c.Char_Id = asp.Char_Id
    WHERE c.Char_Id = @Char_Id;
END
```

3. Configure in FrameworX

sql

```sql
-- Query configuration
EXEC TestSP @Char_Desc = '{{Tag.CharacteristicDesc}}'
```

4. Execute from Script

csharp

```csharp
public void ExecuteProcedure()
{
    @Tag.CharacteristicDesc = "C019";

    string status;
    @Dataset.Query.TestProcedure.ExecuteCommandWithStatus(out status);

    if (status == "OK")
    {
        // Results available in mapped tags
        DataTable results = @Tag.ProcedureResults;
    }
}
```

Best Practices Checklist

- **Use parameters** - Prevent SQL injection
- **Handle errors** - Check return values
- **Validate inputs** - Before execution

- **Use transactions** - For multi-step operations
- **Document procedures** - Clear parameter descriptions
- **Test thoroughly** - All parameter combinations
- **Monitor performance** - Execution plans

Troubleshooting

Procedure not found:

- Verify procedure exists
- Check schema/owner
- Confirm permissions

Parameter errors:

- Match parameter names exactly
- Check data types
- Verify nullable parameters

Output not returned:

- Confirm OUTPUT keyword
- Check tag mapping
- Verify execution method

Performance issues:

- Review execution plan
- Add indexes
- Optimize procedure logic

In this section...
