جامعة طرابلس
كلية تقنية المعلومات

Advanced Databases قواعد البيانات المتقدمة
ITSE312
د. عبدالسلام منصور الشريف
a.abdoessalam@uot.edu.ly

المحاضرة الثامنة – الإجراءات المخزنة
Stored Procedures

Contents

- Programming Database
  - Stored Procedures

Stored Procedures

A stored procedure is an executable object stored in a database. SQL Server supports:

- Stored procedures:
  - One or more SQL statements precompiled into a single executable procedure.
- Extended stored procedures:
  - C or C++ dynamic-link libraries (DLL) written to the SQL Server Open Data Services API for extended stored procedures. The Open Data Services API extends the capabilities of stored procedures to include C or C++ code.

Stored Procedures

- Calling a stored procedure on the data source can provide:
  - Higher performance
    - SQL statements are parsed and compiled when procedures are created. This overhead is then saved when the procedures are executed.
  - Reduced network overhead
    - Executing a procedure instead of sending complex queries across the network can reduce network traffic.
  - Greater consistency
    - If an organization's rules are implemented in a central resource, such as a stored procedure, they can be coded, tested, and debugged once. Individual programmers can then use the tested stored procedures instead of developing their own implementations.
  - Greater accuracy
    - Because stored procedures are usually developed by experienced programmers, they tend to be more efficient and have fewer errors than code developed multiple times by programmers of varying skill levels.
  - Added functionality
    - Extended stored procedures can use C and C++ features not available in Transact-SQL statements.

Stored Procedures

```
CREATE [ OR ALTER ] { PROC | PROCEDURE }
    [schema_name.] procedure_name
    [ { @parameter_name [ type_schema_name. ] data_type }
        [ = default ] [ OUT | OUTPUT | [READONLY]
    ] [ ,...n ]
[ WITH <procedure_option> [ ,...n ] ]
AS { [ BEGIN ] sql_statement [;] [ ...n ] [ END ] }
[;]
<procedure_option> ::=
    [ ENCRYPTION ]
    [ RECOMPILE ]
    [ EXECUTE AS Clause ]
```

```
DROP { PROC | PROCEDURE } [ IF EXISTS ] { [ schema_name. ]
procedure } [ ,...n ]
```

Stored Procedures

```
CREATE PROC What_DB_is_this
AS
SELECT DB_NAME() AS ThisDB;
GO

EXECUTE What_DB_is_this
```

```
CREATE PROC What_DB_is_that @ID INT
AS
SELECT DB_NAME(@ID) AS ThatDB;
GO

EXECUTE What_DB_is_that 2
```

Stored Procedures

```
----- non paramertized proc
create proc dbo.getStudentSemesters09223333
as
select [Id],[SemesterType],[StudentId],[CourseId],
       [Mark],[IsSuspended],[TeachedBy],[EnrollmentOn]
  from [ITDatabase].[dbo].[Semesters]
  where StudentId = '09223333'
go
-- exec
exec dbo.getStudentSemesters09223333
```


Stored Procedures

```
----- paramertized proc
create proc dbo.getStudentSemesters
       @StudentId char(10)
as
select [Id],[SemesterType],[StudentId],[CourseId],
       [Mark],[IsSuspended],[TeachedBy],[EnrollmentOn]
  from [ITDatabase].[dbo].[Semesters]
  where StudentId = @StudentId
go
-- exec
exec dbo.getStudentSemesters '09223333'
```

Stored Procedures

```
-------- non data proc
create proc dbo.MathTutor
       @m1 smallint,
       @m2 smallint,
       @result smallint output
as
       set @result = @m1 * @m2;
go
-- exec
declare @answer smallint;
exec MathTutor 5,6, @answer output;
select 'The result is: ', @answer;
```


Procedural Language – TRY … CATCH

- Implements error handling for Transact-SQL. A group of Transact-SQL statements can be enclosed in a TRY block. If an error occurs in the TRY block, control is passed to another group of statements that is enclosed in a CATCH block.

```
BEGIN TRY
       { sql_statement | statement_block }
END TRY
BEGIN CATCH
     [ { sql_statement | statement_block } ]
END CATCH
[ ; ]
```

Procedural Language – TRY … CATCH

- In the scope of a CATCH block, the following system functions can be used to obtain information about the error that caused the CATCH block to be executed:
- ERROR_NUMBER() returns the number of the error.
- ERROR_SEVERITY() returns the severity.
- ERROR_STATE() returns the error state number.
- ERROR_PROCEDURE() returns the name of the stored procedure or trigger where the error occurred.
- ERROR_LINE() returns the line number inside the routine that caused the error.
- ERROR_MESSAGE() returns the complete text of the error message. The text includes the values supplied for any substitutable parameters, such as lengths, object names, or times.

Stored Procedures – Exception Handling

```
CREATE PROCEDURE usp_GetErrorInfo
AS
SELECT
    ERROR_NUMBER() AS ErrorNumber
    ,ERROR_SEVERITY() AS ErrorSeverity
    ,ERROR_STATE() AS ErrorState
    ,ERROR_PROCEDURE() AS ErrorProcedure
    ,ERROR_LINE() AS ErrorLine
    ,ERROR_MESSAGE() AS ErrorMessage;
GO

---exec
BEGIN TRY
    SELECT 1/0; -- Generate divide-by-zero error.
END TRY
BEGIN CATCH
    -- Execute error retrieval routine.
    EXECUTE usp_GetErrorInfo;
END CATCH;
```

Stored Procedures – Exception Handling

```
create table dbo.TableNoKey (ColA int, ColB int)
create table dbo.TableWithKey (ColA int primary key, ColB int)
go

create proc dbo.AddData @a int, @b int
as
  begin try
    insert dbo.TableNoKey values (@a, @b)
    insert dbo.TableWithKey values (@a, @b)
  end try
  begin catch
      select ERROR_NUMBER() ErrorNumber,
               ERROR_MESSAGE() [Message];
  end catch
GO

--exec
exec dbo.AddData 1, 1
exec dbo.AddData 2, 2
exec dbo.AddData 1, 3 --violates the primary key
```

Stored Procedures – Exception Handling

```
alter  proc dbo.AddData @a int, @b int
as
  begin try
    begin tran
    insert dbo.TableNoKey values (@a, @b)
    insert dbo.TableWithKey values (@a, @b)
    commit tran
end try
  begin catch
      rollback tran
      select ERROR_NUMBER() ErrorNumber,
              ERROR_MESSAGE() [Message]
  end catch
go


exec dbo.AddData 1, 1  -- will not exec duplicate and within same trans
```