Stored Procedures

CMPE 2400

Databases

Stored Procedures

- Stored procedures are pre-compiled code that is stored on, and runs inside, the database server thus making it more efficient than when passed from another application across a connection
- A stored procedure may perform any task that a regular SQL statement may perform, as they are built using SQL

Stored Procedures

- Stored procedures may accept any number of parameters, either as literal values, or as variables passed from the caller.
- Stored procedures may deliver data to the user in three ways:
  - Via the return code
    - This code is always an integer and is often used to indicate the status of execution within the stored procedure.
  - Via results (displayed to the screen)
  - Via output parameters
    - These parameters may be viewed in the same light as a reference parameter passed to a method in C#
    - Output parameters also accept inputs in SQL Server

Stored Procedures

- May be called within other stored procedures
- May be grouped together to form libraries where the user may only be granted access to the top layer of procedures, which in turn may call other procedures during their execution
- As all execution happens on the database server, only data contained in the final result set, output parameters, and return code are passed back to the client or client application
- Different paths of execution may be created through proper use of branching

Modularization

- Stored procedures allow complex SQL coding to be broken up into logical units.
- Leads to more easily manageable SQL code
- Allows reusability if a stored procedure is created to perform a simple task rather than a series of tasks unique to a particular application

Security

- Using stored procedures, we may create a user interface for client applications
- This allows us to fully restrict access to the structure and data within our database, except through strict controls laid out within our stored procedures
- All updating, inserting, and deleting of records would be performed through stored procedures
- This will allow for server-side data validation and access restriction, thus protecting our database integrity

Object Existence

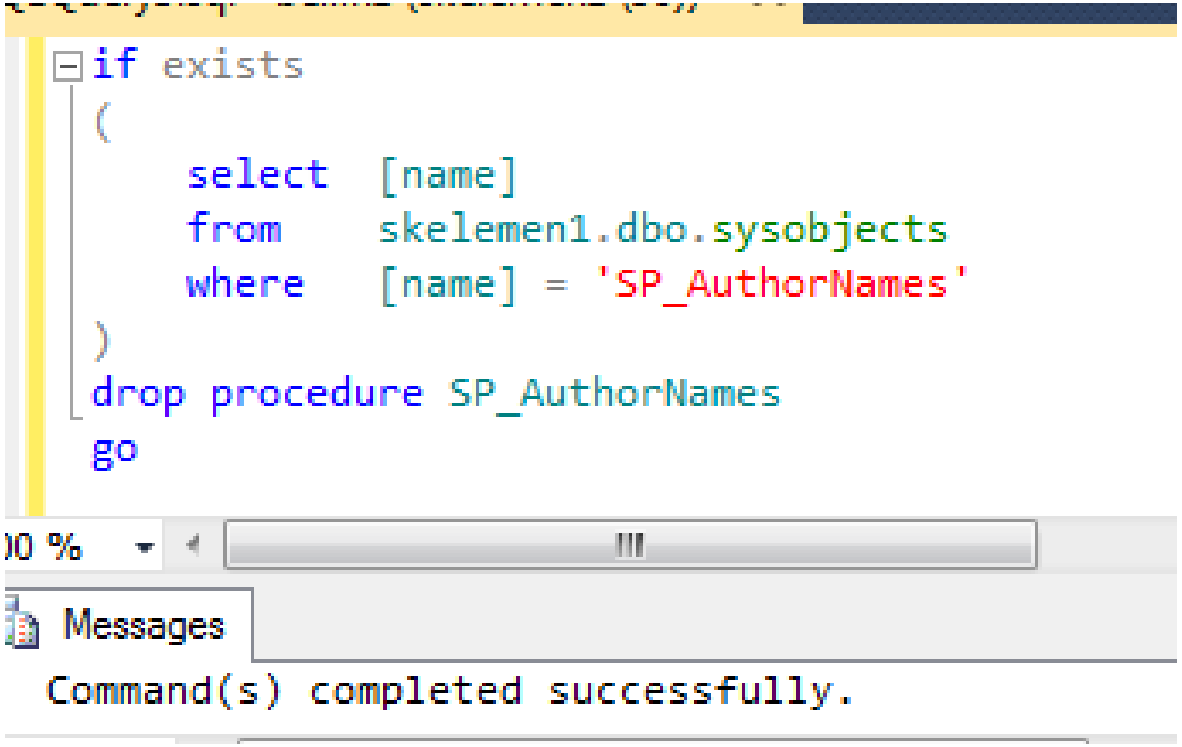
```
if exists
(       select  [name]
        from    sysobjects
        where   [name] = 'SP_ProcedureName'
)
drop procedure SP_ProcedureName
go
```

-OR-

```
drop procedure if exists SP_ProcedureName
go
```

Basic Example

- Get all of the authors first and last names formatted as ‘FirstName LastName’ from the authors table in the Publishers database
- First, make sure the procedure does not exist



Basic Example

- Second, build the regular SQL statement to accomplish the task, and test it to see the expected results

```
select  au_fname + ' ' + au_lname as 'Author Name'
from    Publishers.dbo.authors
```

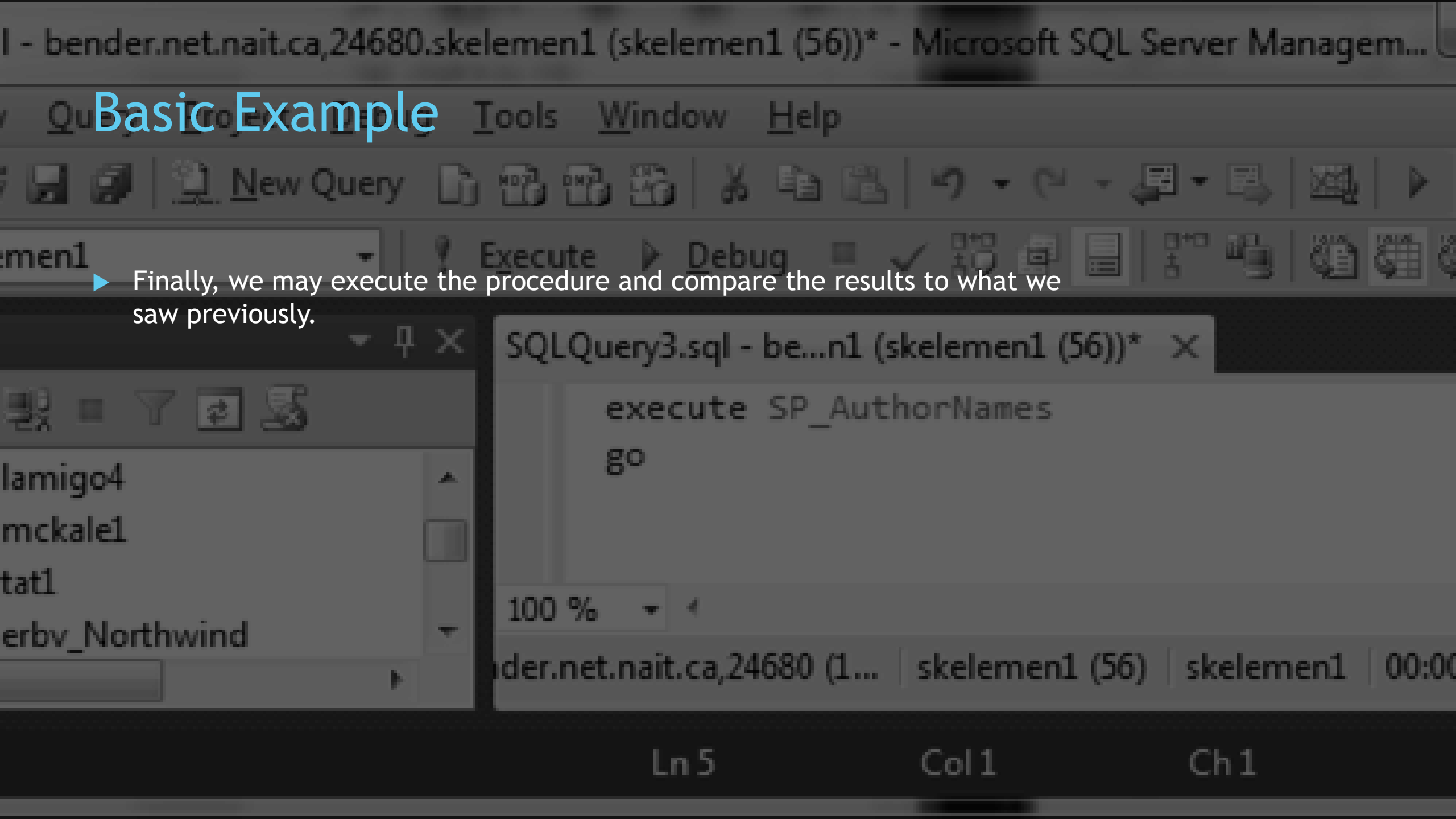
Basic Example

- Third, embed the SQL in a procedure creation batch

```
create procedure SP_AuthorNames
as
select  au_fname + ' ' + au_lname as 'Author Name'
from    Publishers.dbo.authors
go
```

Basic Example

- Finally, we may execute the procedure and compare the results to what we saw previously.



Basic Example Results

Abraham Bennet	Reginald Blotchet-Halls	Cheryl Carson	Michel DeFrance	Innes del Castillo
Ann Dull	Marjorie Green	Morningstar Greene	Burt Gringlesby	Sheryl Hunter
Livia Karsen	Charlene Locksley	Stearns MacFeather	Heather McBadden	Michael O'Leary
Sylvia Panteley	Albert Ringer	Anne Ringer	Meander Smith	Dean Straight
Dirk Stringer	Johnson White	Akiko Yokomoto	*(23 row(s) affected)*	

Exercise 1

- In your own personal database create a procedure that will display the following information from NorthwindTraders:
  - ProductId, ProductName, UnitPrice and CategoryName
- Execute the procedure

Parameters

- SQL stored procedures start to show their true power with the addition of parameters
- Remember that in SQL Server, but not necessarily other DBMS', it is mandatory to preface all parameter names with the @ symbol

Parameters

- Logically, when we declare a parameter, we must also provide a data type for the parameter.
  - This type may be any type found in the section covered earlier regarding variable types.
- A default value may also be assigned to a parameter if no value is supplied.
- By default, parameters are input only, but we may apply the output modifier to allow the parameter value to be used for output **as well**
  - Consider this to be like a reference variable in C#

Extending the Syntax

- The only part of the creation syntax which must be added to is the parameter specification.

```
create proc[edure] SP_ProcedureName
[@Parameter1Name data type] [= DefaultValue] [output],
[@Parameter2Name data type] [= DefaultValue] [output]
as
SQL statement
go
```

Parameter Example

- A simple use of a parameter would be having an adjustable value in our where clause.
  - Let us adjust the procedure from the previous example to only retrieve author's names where the first name begins with a particular letter.
  - We will set a default value of 'M'
  - Note that because the procedure has already been created, you may either drop the procedure and recreate it, or change your create keyword to alter, as has been done on the following slide.

Parameter Example

```
alter procedure SP_AuthorNames
    @FirstLetter char = 'M'
as
select  au_fname + ' ' + au_lname as 'Author Name'
from    Publishers.dbo.authors
where   au_fname like (@FirstLetter + '%')
go

execute SP_AuthorNames
go
```

Default Parameter Results

Author Name

Michel DeFrance

Marjorie Green

Morningstar Greene

Michael O'Leary

Meander Smith

(5 row(s) affected)

Supplying a Parameter

```
execute SP_AuthorNames 'A'
go
```

- OR -

```
declare @FLetter char = 'A'

execute SP_AuthorNames @FLetter
go
```

New Results

Author Name

Abraham Bennet

Ann Dull

Albert Ringer

Anne Ringer

Akiko Yokomoto

(5 row(s) affected)

Exercise 2

- Create a procedure
- Parameter: CategoryName
- Display the product id, product name, total equivalent stock value (ie unit in stock X unit price) of all products of that category name from the NorthwindTraders database
- Execute the procedure passing the category ‘Beverages’ as parameter

Exercise 3

Declare	Declare a variable called @catName and assign the value ‘Beverages’ to it.
Execute	Execute the procedure in exercise 2, passing the variable @catName as parameter
Look	Look at the Categories table of the NorthwindTraders database. Change the value assigned to @catName to each of the category names and execute the procedure again.

Exercise 4

Make a copy of the procedure written in exercise2. Add a second parameter to represent a stock value. The procedure should display all the information for the category name and for stock values (Units in stock X Unit price) > than the stock value passed as parameter.

Declare 2 variables to be used as parameters when calling the procedure.

Set the declared variables to different values of your choice and pass them as parameters when calling the procedure.

Exercise 5

- Create a procedure has as parameters 2 strings and a date. Each time the procedure is executed, it adds a record to the employee table of your username_DB database. The 2 string parameters will represent the first name and last name, respectively while the date parameter will represent the HireDate. It generates a random integer value in the range of 21 to 50 to represent the hourly rate. There is no need to provide values for the confirmation date and phone number at this point.
- Execute the procedure 10 times to add 10 new records to the employee table

Exercise 6

- Create a procedure to add confirmation dates to all records in the employee table.
- Confirmation dates will be based on hourly pay as per the table below:

Hourly Pay	Confirmation (Amount of time after hiring)
15-20	2 years
20+ -30	18 months
30+ - 5	1 year
35+ - 45	6 months
45+ - 50	12 weeks

Error checking

- A stored procedure may also return a value
  - This is generally used for returning error codes.
  - The built in system variable @@error may be used for this task.
    - If an error is encountered in the execution of the procedure, this value will be non-zero
  - As with other languages, the return value must be stored to be useful, otherwise it is lost to limbo.

Error Checking Example

- Modify your stored procedure to return the error code, call it, and store the returned value.

```
alter procedure SP_AuthorNames
    @FirstLetter char = 'M'
as
select  au_fname + ' ' + au_lname as 'Author Name'
from    Publishers.dbo.authors
where   au_fname like (@FirstLetter + '%')

return @@error
go

declare @FLetter char = 'A',
        @returnCode int

execute @returnCode = SP_AuthorNames @FLetter
go
```

Checking the code

- To see the return value of the execution along with the result set, use the following immediately after the execute statement, but before a go.

```
declare @FLetter char = 'A',
        @returnCode int

execute @returnCode = SP_AuthorNames @FLetter

print 'The error return code is: ' + cast(@ReturnCode as varchar)
go
```

Error Code Results

- Using text mode for your results will yield the following:

```
Author Name
----------------------------------------------------------------
Abraham Bennet
Ann Dull
Albert Ringer
Anne Ringer
Akiko Yokomoto

(5 row(s) affected)

The error return code is: 0
```

Using Error_Number() and Error_Message()

- The function Error_Number() also returns the value of @@error. Error_Message() gives a full description of the error.

Exercise 7

- Create a procedure that has as parameter an integer value to represent an id and a date value to represent a confirmation date. It returns the value of @@error
- The procedure must access the employee table of the ***username_DB*** database, set the confirmation date of the record with the given employee id (the first parameter) as the date being passed as the parameter.
- It should return the error number
- Try to run the procedure with a confirmation date which is after the hired date and one which is before the hired date.
- In each case, when the procedure is called you should check for the returned value and display a message accordingly for the user.

Using Try..Catch Blocks

In the previous exercise, the procedure executed the sql statement with a failure

We can avoid that by using a try-catch block

A try..catch block has the following form:

```
begin try
   statements
end try
begin catch
   statements
end catch
```

Using Try..Catch Blocks

Using try..catch, We place our SQL statement in the **try** block. If its successful, we return a message accordingly to the user.

All error-handling code are placed in the **catch** block.

Using the Try..Catch Block

```
use OM_DB
begin try
Insert into employee (FName,LName,hourlypay,HiredDate)
 values ('Hannah','Mission',10,'2022-03-04')

end try
begin catch
 print Error_message()
end catch
```

```
(0 rows affected)
The INSERT statement conflicted with the CHECK constraint "CK_hourlypay".
The conflict occurred in database "OM_DB", table "dbo.employee", column 'hourlypay'.
```

Exercise 8

Modify the procedure in Exercise 7 so that it uses a try-catch block. If the execution is successful, the procedure returns 1, otherwise it returns -1.

When the procedure is executed, you should catch the returned value and display the message “successful execution” or “execution failed” based on the returned value.

Review- General Syntax for Stored Procedures

```
create proc[edure] SP_ProcedureName
    [@Parameter1Name data type] [= DefaultValue] [output],
    [@Parameter2Name data type] [= DefaultValue] [output]
    as
    SQL statement
    go
```

Exercise 9

Write a procedure that has 1 input parameter and 3 output parameters. The input parameter is a product id and the 3 output parameters are the product name, unit price and the number of units of the product currently in stock. The procedure will use the product id to pull the other data values from the NorthwindTraders database.

Execute the procedure passing the productid as input parameter value. Provide required variables for the output parameters

After you execute the procedure, use the output parameters to display: There are <num of units> units of the product <product name> in stock. The value of the stock is <value of stock>.

Note: The value of the stock is unitprice x No. of units in stock

System Stored Procedures

There are a number of stored procedures available on the server that can be used to obtain information about tables and databases.

We'll look at 2 of them:

Sp_Help	Sp_Statistics

Using Transactions in Stored Procedures

- Sometimes we may have several statements to be executed in a stored procedure and we may want either all of them to go through or none to go through.
- We have previously seen that we can use Transactions with rollback and commit to achieve the above.
- In stored procedures, you can choose to return different error values depending on where the rollback occurred.

Transaction Workflow

```
drop procedure if exists ProcX
--execute a check if required
--if statements to determine what to do
--time for some DML
begin transaction
    begin try
      --ONE DML operation
    end try
    begin catch
        rollback
        return -y
    end catch
    --repeat try/catch blocks as many times as needed
--if you get here, that means all the DML above succeeded
commit
```

- Create this procedure in your own copy of Northwind. What does this procedure do?

```
drop procedure if exists Buy
go
--create procedure to create a new order
create procedure Buy
@Customer varchar(50),
@Product varchar(50),
@HowMany int
as
--does the customer exist
declare @customerid char(5)

select @customerid = c.CustomerID
from Customers c
where c.CompanyName like '%' + @Customer + '%'
declare @numRows int = @@ROWCOUNT

begin transaction

if @numRows > 1
begin
    print 'Too many customers found'
    rollback
    return -1
end
else if @numRows = 0
begin
    --insert the new customer
    --because this could cause an error, use a try..catch
    begin try
        insert into customers (customerid, CompanyName)
        values(left(upper(@customer),5),@customer)
        set @customerid = left(upper(@customer),5)
    end try
    begin catch
        print 'Error inserting new customer: ' + ERROR_MESSAGE()
        rollback
        return -2
    end catch

    print @customerid
end
print @customerid
commit
return 0
```

- In the example that follows in the next 2 slides, we have a stored procedure that will set confirmation date for 2 employees (their ids and confirmation dates are passed as parameters), into the employee table of our username_DB database.
- Either both the confirmation dates will be set or non of them.
- The procedure will return a 1 on successful completion, a -1 if the first query was not successful and a -2 if the first query was successful but the second one was not.

Using Transactions in Stored Procedures

Using Transactions in Stored Procedures

```
use oveeyenm

drop procedure if exists SP_Transaction_Ex1
go

Create Proc SP_Transaction_Ex1
@empId_1 int,
@conf_date1 date,
@empId_2 int,
@conf_date2 date
as
begin Transaction
   Update OM_DB.dbo.employee
   Set confirmationDate =@conf_date1
   where employeeId=@empId_1
   if (@@error <>0)
     begin
        rollback
        return -1
     end

   Update OM_DB.dbo.employee
   Set confirmationDate =@conf_date2
   where employeeId=@empId_2
   if (@@error <>0)
     begin
        rollback
        return -2
     end
commit
return 1
 go
```

Using Transactions in stored procedures

```
declare @emp1 int=102
declare @date1 date='2022-05-01'
declare @emp2 int=110
declare @date2 date='2019-01-21'
declare @retVal int

Exec @retVal=SP_Transaction_Ex1 @emp1,@date1, @emp2,@date2

print
case @retVal
   when 1 then 'Successful Execution'
   when -1 then 'Failed on first update'
   when -2 then 'Failed on second update'
end

go
```

Exercise 10

- Consider the username_DB database
- When a new customer makes a purchase, you need to insert the customer in the customer table and the invoice into the CustomerInvoice table
- Write a stored procedure that has as parameters:
  - customer id, first name, last name
  - invoice id, employee id, invoice date
- It starts a transaction and first tries to insert the customer information into the customer table. If unsuccessful for any reason (customerId already exists, first name or last name is null) it rolls back and returns value -1
- If the above is successful, it attempts to add the invoice to the CustomerInvoice table. If for any reason, it's unsuccessful (invoice id already exists, date wrong string format), it rolls back and returns -2
- If everything goes well it commits and returns 0
- Write the code for declaring required variables, assigning values, executing the procedure and displaying appropriate messages for the user