ICA | INFORMATICA COMMUNICATIE ACADEMIE

Course Database Implementation

Stored Procedures

Errors

Execute twice:

```
CREATE TABLE OurIfTest (col int PRIMARY KEY)
```

Second execution:

```
Msg 2714, Level 16, State 6, Line 1
There is already an object named 'OurIfTest' in the
database.
```

Use TRY/CATCH blocks for error handling

Execute twice:

```
BEGIN TRY
  CREATE TABLE OurIfTest (col int PRIMARY KEY)
END TRY
BEGIN CATCH
  DECLARE  @ErrorNo int, @Message nvarchar(2048),
           @Severity int, @State int
  SELECT
    @ErrorNo = ERROR_NUMBER(), @Message = ERROR_MESSAGE(),
    @Severity   = ERROR_SEVERITY(), @State= ERROR_STATE()

  IF @ErrorNo = 2714
    RAISERROR ('WARNING: Skipping CREATE as table already
                exists.', @Severity, @State)
  ELSE
    RAISERROR (@Message, @Severity, @State)
END CATCH
```

Within the CATCH block

- ***When*** execution is passed to the `CATCH` block, the error information is also passed to the `CATCH` block.
  - Use system functions to retrieve this:

Error function	Returns
`Error_Message()`	The text of the error message
`Error_Number()`	The number of the error < 50000: system defined messages >= 50000; user defined messages
`Error_Procedure()`	The name of the stored procedure or trigger in which the error occurred
`Error_Severity()`	The severity of the error
`Error_State()`	The state of the error
`Error_Line()`	The line number within the batch or stored procedure that generated the error
`Xact_State()`	Whether the transaction can be committed

What to do in the CATCH block?

1. You might need to roll back the transaction (follows)

2. If the detected error is not a SQL Server error:
   raise the error message to inform the caller.

   Two options:
   - use the `RAISERROR()` functionality (like in the previous example)

   - use the `THROW` command

3. Optionally, log the error to an error table.

`RAISERROR()` and severity levels

- **Returns (custom) error messages to calling procedure or front-end**
- **Severity levels from 0 through 18 can be specified by any user**
- **RAISERROR() with a severity 1 – 10 in a TRY block: control is NOT transfered to the CATCH block!**
- **RAISERROR() with a severity 11 – 19 in a TRY block: control is transfered to the CATCH block!**
- **Severity levels 20 – 25 are considered fatal**
  - client connection is terminated after receiving the message, and the error is logged in the error and application logs
  - **control is NOT transfered to the CATCH block**

THROW

Information captured similar to RAISERROR()

Can be configured

- Except the severity level: is always 16

```
-- Try out now:
BEGIN TRY
    RAISERROR ('custom error', 9, 1)
    SELECT 1/0
END TRY
BEGIN CATCH
    THROW
END CATCH
```

Nice blog: Exception handling

http://sqlhints.com/2014/01/20/exception-handling-in-sql-server/

Stored Procedure

- **Named program compiled and stored IN the server as an independent database object**

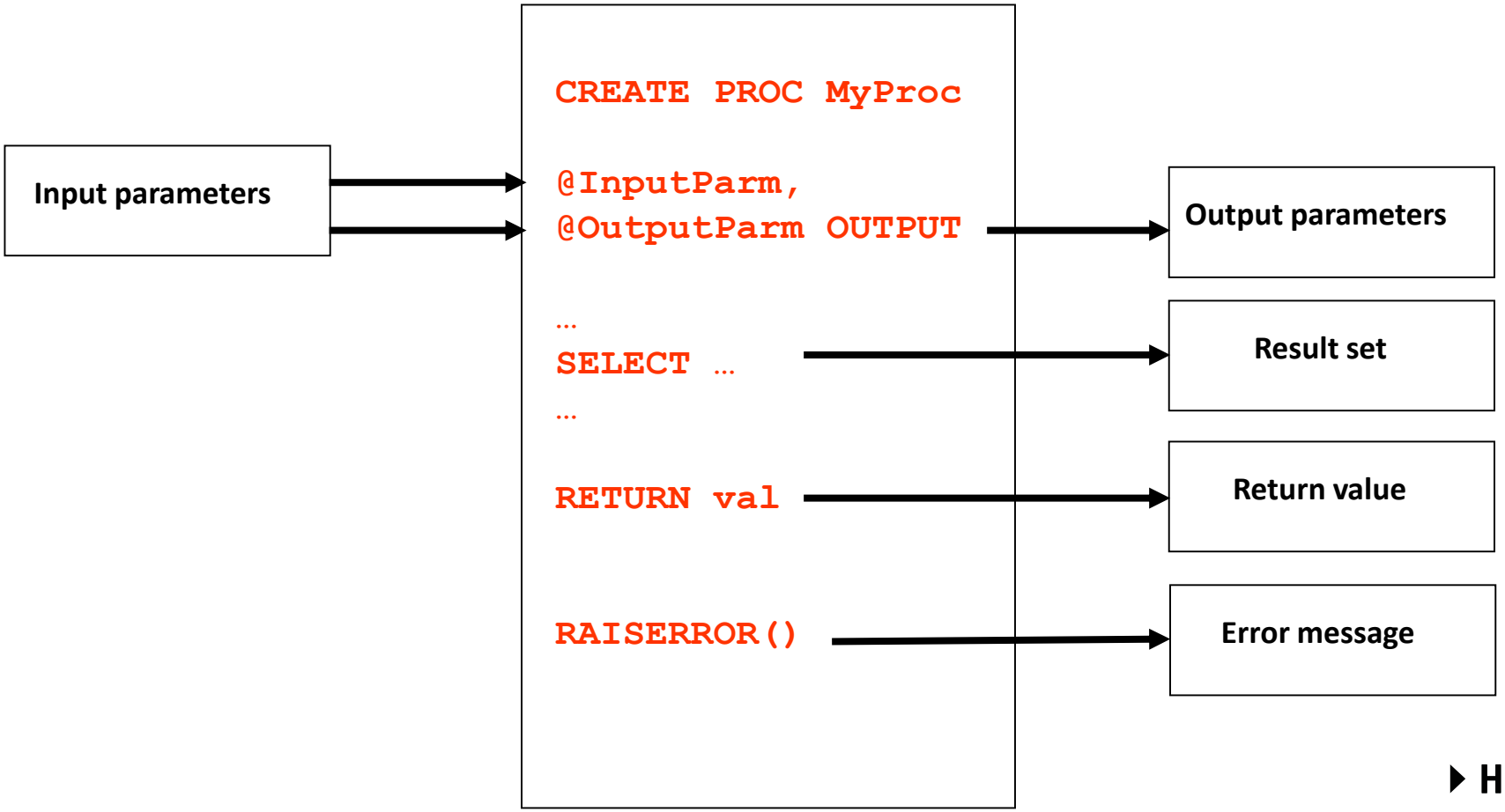
  Collection of:
  - SQL-statements
    and/or
  - procedural logic (if-statements, while-statements, etc.)
    and/or
  - calls of built-in functions (getdate(), etc.)

- **Can be called from a client**
  - or from another stored procedure
  - parameters may be passed and returned
  - returned error codes may be checked

T-SQL sp example

```
USE AdventureWorks

-- Definition
CREATE PROCEDURE spEmployee
AS
BEGIN
   SELECT * FROM HumanResources.Employee
END

-- Execution
EXECUTE spEmployee

-- Changing
ALTER PROCEDURE spEmployee ...

-- Dropping
DROP PROCEDURE spEmployee
```

Input parameters and information returned



Input parameters, with default

```
CREATE PROC spEmployee
    @LastName nvarchar(50) = NULL     -- Default NULL
AS
BEGIN
  IF @LastName IS NULL                -- EXEC spEmployee
    SELECT * FROM HumanResources.Employee
  ELSE                                -- EXEC spEmployee 'A'
    SELECT c.LastName, c.FirstName, e.*
    FROM Person.Person c
         INNER JOIN HumanResources.Employee e
           ON c.BusinessEntityID = e.BusinessEntityID
    WHERE c.LastName LIKE @LastName + '%'
END
```

Questions

```
-- What is the difference between
EXEC spEmployee
-- and
EXEC spEmployee @LastName = NULL

-- What is the difference between
EXEC spEmployee
-- and
EXEC spEmployee @LastName = ''
```

Code guideline

Use the parameter name in the call:

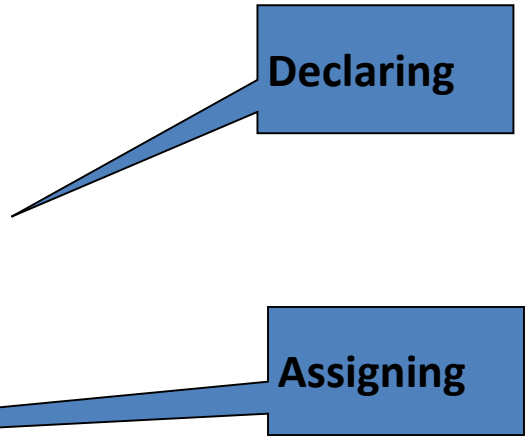
```
EXEC spEmployee @LastName = 'A'
```

Stored Procedure Example 2

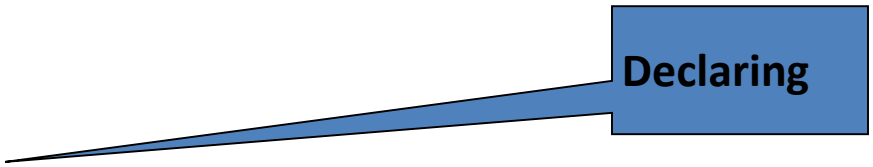
```
USE AdventureWorks2012
GO
CREATE PROC procUpdateMaritalStatus
            @BusinessEntityID INT,
            @newMaritalStatus CHAR(1)
AS
BEGIN
  UPDATE HumanResources.Employee
    SET MaritalStatus = @newMaritalStatus
    WHERE BusinessEntityID = @BusinessEntityID
END
GO
-- transaction management just for testing purposes
BEGIN TRANSACTION
EXEC procUpdateMaritalStatus
    @BusinessEntityID = 1,@newMaritalStatus = 'M'
ROLLBACK TRANSACTION
```

sp with an OUTPUT parameter

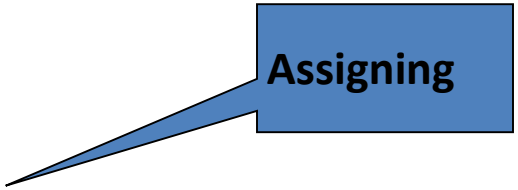
```
CREATE PROC procLookupTitle
   @BusinessEntityID INT,
   @JobTitle NVARCHAR(50) OUTPUT
AS
BEGIN
  SELECT @JobTitle = JobTitle
  FROM HumanResources.Employee
  WHERE BusinessEntityID = @BusinessEntityID
END
```

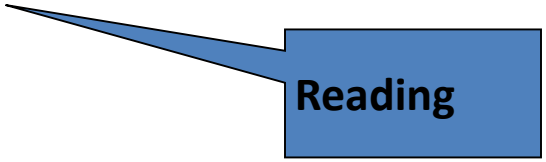


Call of sp with OUTPUT parameter

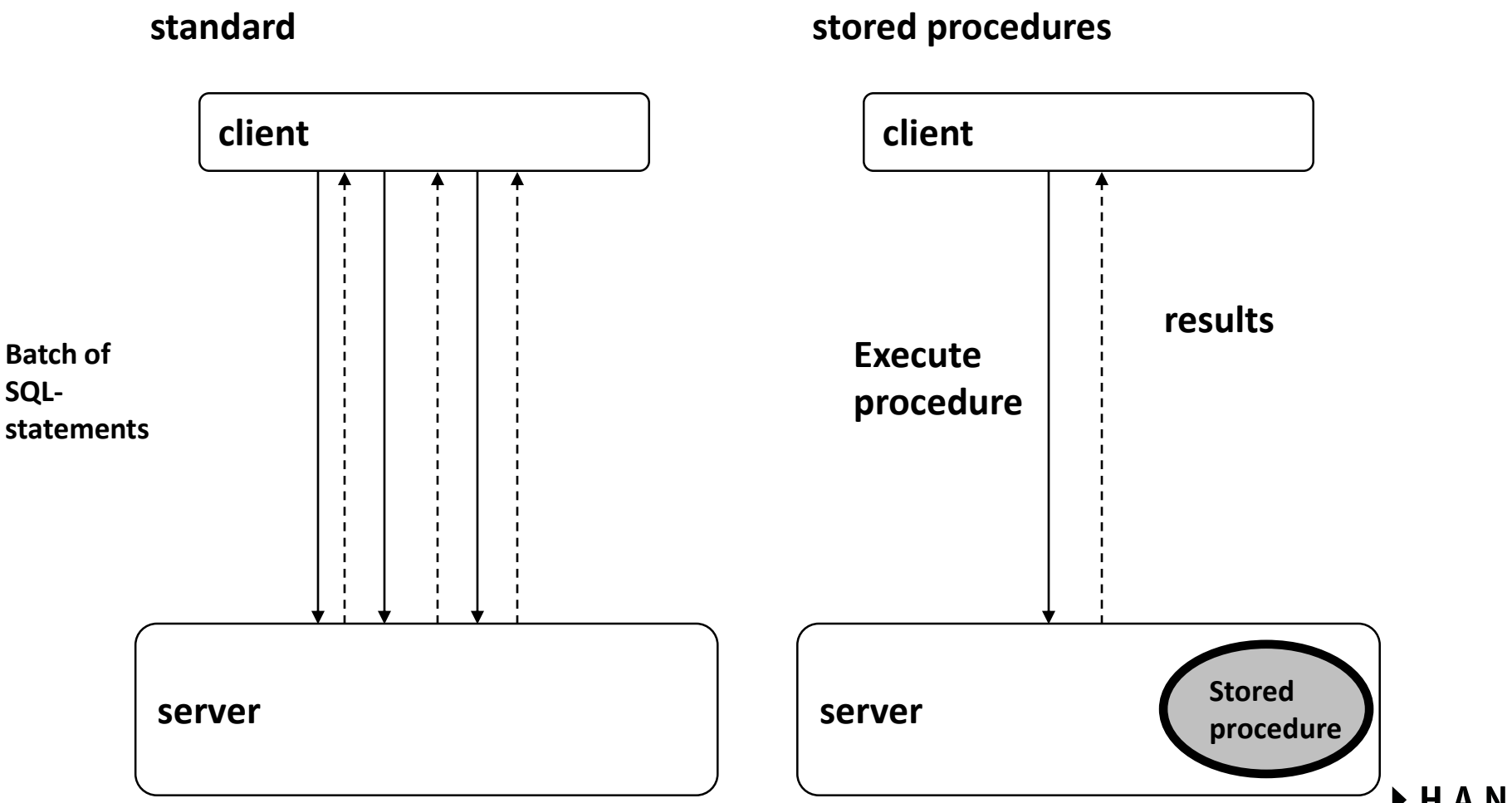



```
DECLARE @JobTitleOutput NVARCHAR(50)

EXEC procLookupTitle
   @BusinessEntityID = 3,
   @JobTitle = @JobTitleOutput OUTPUT

SELECT @JobTitleOutput AS 'Employee Title'
```





Stored Procedures vs SQL



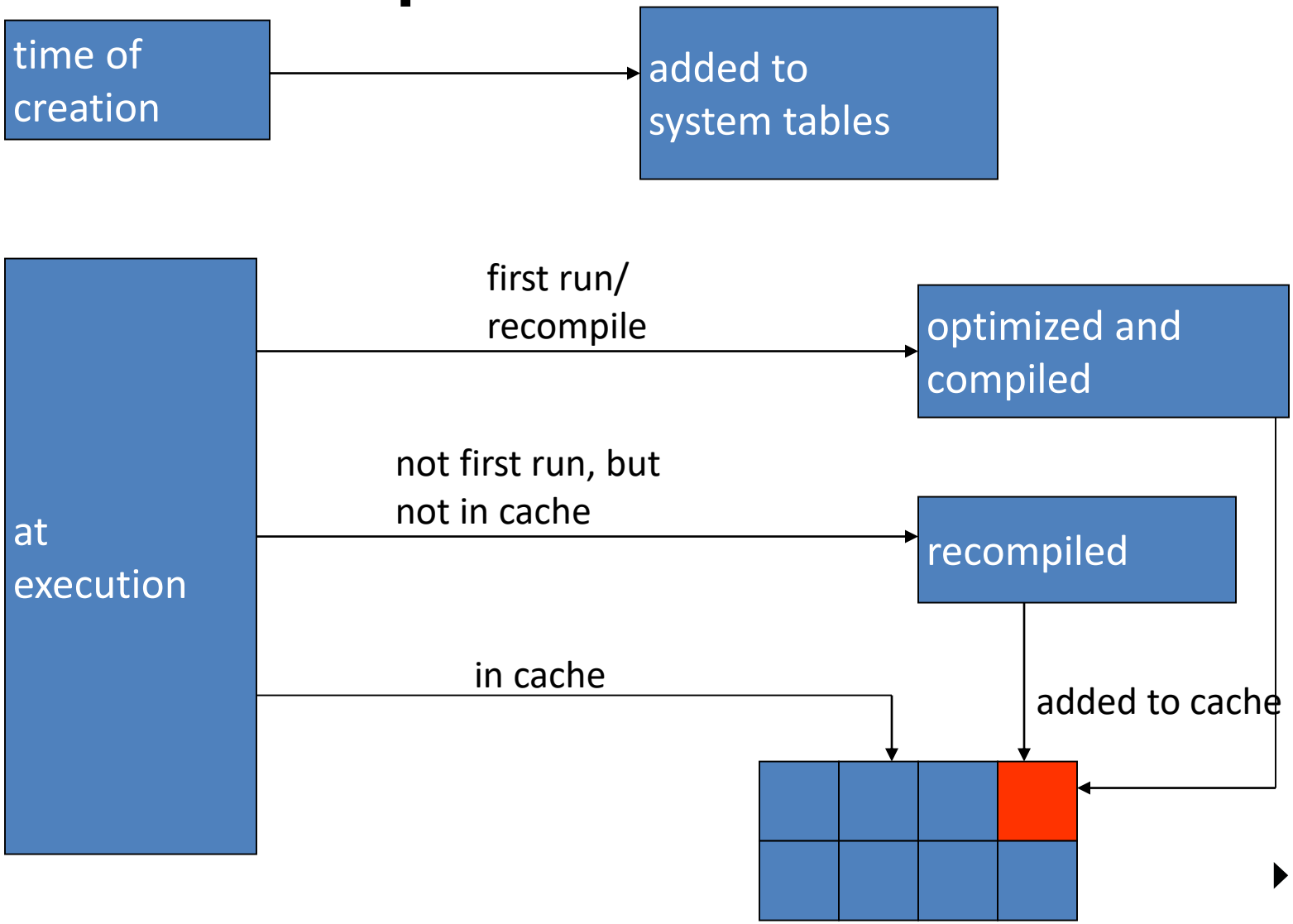
Exercises

Workbook

Theme Stored Procedures

Exercises 1 - 6

Sprocs and Performance



Benefits and Drawbacks sp's

Benefits

- **Faster execution**
  - Precompiled and optimized
- **Reduced network traffic**
- **Restricted, function-based access to tables**
- **Automation of complex transactions**

Drawbacks

- **Non-standard**
  - not portable across platforms
  - no standard way to pass or describe the parameters
  - no good support by tools
- **Complex coding**
- **Performance may be poor if the execution plan is not refreshed**

Exercise

Give a stored procedure with one parameter `@sortParameter` with data type `varchar` that:

1. Returns all records from table `Person.Person` from `AdventureWorks` if `@sortParameter` equals `'LastName'`, sorted on column `LastName`.

2. Returns all records from `Person.Person` if `@sortParameter` equals `'FirstName'`, sorted on column `FirstName`.

3. Raises an error in all other cases (use `RAISERROR`).

4. At last, use `TRY/CATCH`

sp with OUTPUT parm and return value

```
CREATE PROC procLookupTitle
    @BusinessEntityID INT,
    @JobTitle  NVARCHAR(50)    OUTPUT
AS
BEGIN
  SELECT @JobTitle = JobTitle
  FROM HumanResources.Employee
    WHERE BusinessEntityID = @BusinessEntityID
  IF @@ROWCOUNT = 0   -- row not found
    BEGIN RETURN 1 END    -- RETURN terminates the proc
  RETURN 0
END
```

SP with OUTPUT parm and return value

```
CREATE PROC procLookupTitle
    @BusinessEntityID INT,
    @JobTitleNVARCHAR(50) OUTPUT
AS
BEGIN
  SELECT @JobTitle = JobTitle
  FROM HumanResources.Employee
  WHERE BusinessEntityID = @BusinessEntityID
  IF @@ROWCOUNT = 0 -- row not found
    BEGIN RETURN 1 END -- RETURN terminates the proc
  RETURN 0
END
```

Call of sp with OUTPUT parm and return value

```
DECLARE @returnValue int
DECLARE @JobTitleOutput    nvarchar(50)


EXEC @returnValue = procLookupTitle
                        @BusinessEntityID = 3,
                        @JobTitle = @JobTitleOutput OUTPUT


SELECT @JobTitleOutput AS 'Employee title‘
SELECT @returnValue AS 'Return value from proc'
```

Debugging Sprocs

- execute parts of SQL separately
- use print statements
- use temporary tables
- debugger SQL Server

ICA | INFORMATICA COMMUNICATIE ACADEMIE

System Stored Procedures

name starts with `sp_`

in the `master` database