

Table of Contents

Acknowledgement	v
Introduction.....	1
1. Laboratory session 1: Basics of T-SQL programming	2
Learning Objective:	2
1.1 Decalring variables and Parameters:.....	2
1.2 IF ... ELSE statemens.....	3
1.3 WHILE Loop With CONTINUE and BREAK Keywords	4
1.4.CASE... When expressions	6
1.5 Exercises	7
2. Laboratory session 2: User defined functions.....	8
Learning objective	8
2.1 Creating scalar user defined functions using SQL server:.....	8
2.2. Inline table valued functions.....	10
2.3 Multi statements table valued functions	10
2.4. Altering Functions	11
2.5. Dropping Functions	12
2.6. Exercise.....	12
2.7 Creating function using Oracle SQL.....	14
3. Laboratory session 3: Stored Procedures	15
Learning Objective.....	15
3.1 What is stored procedure?.....	15
3.2. Stored Procedures vs Functions:	15
3.3. Executing stored procedures	19
3.4 .Dropping Stored procedures	19
3.5.Modifying stored procedures	19
3.6 Exercise.....	19
Oracle syntax to create storedprocedure	20
4. Laboratory Session 4: Triggers.....	21
Learning Objective:	21
4.1. What is Trigger?	21
4.2 .DML Triggers:	21

4.3. DDL Triggers	23
4.3. Dropping a Trigger	24
4.4 Disabling and Enabling a Trigger	24
4.5. Creating trigger using Oracle syntax:	24
4.6 Calling Procedures within Triggers	25
4.7. Exercise.....	25
5. Laboratory Session 5: Configuring Server and Database security	26
Learning Objective:	26
5.1 Creating Logins.....	27
5.2 Creating and Managing database Users	28
5.3 Creating Schemas.....	29
5.4. Creating Roles.....	30
5.5 Granting Permissions	31
5.6 Revoking Permissions.....	31
5.7 Exercise.....	32
6. Laboratory Session 6: Configuring oracle database Security	34
6.1. Creating user accounts	34
6.2 Granting permission.....	34
6.3. Assigning membership to roles.....	34
6.4. Granting permission.....	34
6.5. Exercise.....	34
7. Laboratory Session7: Managing Transactions.....	35
Learning Objective.....	35
What a transaction is?	35
7.1. Transaction management modes.....	35
7.2. Transaction Isolation models	37
7.3 Exercise.....	39
8. Laboratory Session 8: Data Import and Export	40
Learning Objective.....	40
8.1. Importing from a text file.....	40
8.2. Exporting to a spreadsheet	41
8.2. Exercise.....	44
9. Laboratory Session 9: Backup and Restore configuration.....	45
Learning Objective.....	45

Introduction.....	45
9.2. Backing up data.....	45
9.3. Restoring a backup.....	47
9.4. Exercise.....	48
10. Laboratory Session10: Replication	49
Learning Objective.....	49
Introduction.....	49
10.1.Replication Architecture	50
10.2.Configuring the Distributor.....	51
10.3.Configuring a transactional Publisher.....	53
10.4. Exercise.....	61
11. Laboratory Session 11: Database Mirroring	62
Learning Objective:	62
11.1 Mirroring Prerequisites	62
11.2. Endpoint Firewall Rules	64
11.3. Database Mirroring Monitor	67
Exercise.....	68
12. Laboratory Session 12: Log shipping	69
Learning Objective:	69
12.1. Steps to configure log shipping.....	70
A. Using SQL Server Management Studio	70
B. Using Transact SQL	73
Exercise.....	73
Bibliography	74
Annexes.....	75
Annex 1	75
Annex 2	76

List of Acronyms

DDL	Data Definition Language
DML	Data Manipulation Language
DBMS	Database Management System
DB	Database
SSMS	SQL Server Management Studio
SQL	Structured Query Language
SSIS	SQL Server Integration Service
SSRS	SQL Server Reporting Service
T-SQL	Transact SQL

Introduction

The past few decades have seen a proliferation of organizations that rely heavily on information technology. These organizations store their data in databases. There are several database management system softwares(DBMS) available in the software market with primary differences in performance, platform support, security and functionalities they provide.

In this manual, Microsoft SQL server and oracle11g (where deemed necessary) are used for demonstrating some database concepts. The lessons covered in this manual cover advanced concepts from both database development and administration point of view.

This manual is primarily prepared for Bsc degree students who already have taken the course **“Fundamentals of Database Systems”** or equivalent. The manual covers advanced database topics such as functions, stored procedures, triggers, security, transaction, import/export, backup, replication, mirroring and log shipping. After explanation of each topic, examples are given. In addition, each session includes hands-on exercises to test the knowledge of students and challenge them how to apply that knowledge to solve real world practical business problems.

The database tables used in this material are taken from the hypothetical Logical data model of XYZ registrar and ABC Publishing databases which are shown in Annex 1 and 2 of this manual.

To evaluate student individual practical skills and performance on this course, there will be laboratory tests seriously monitored on regular basis.

The following Database Management System (DBMS) softwares are needed to practice the concepts covered in this material:

- ✓ Microsoft SQL Server 2012 or Later
- ✓ Oracle 11g or Later

The introduction shall be page number one

1. Laboratory session 1: Basics of T-SQL programming

Learning Objective:

At the end of this lesson, students will be able to:

- Declare variables
- Understand the usage of If...Else statements
- Understand the usage of WHILE Loop With CONTINUE and BREAK Keywords
- Understand the usage of CASE...WHEN expressions
- Use control structures to solve problems

1.1 Declaring variables and Parameters:

In SQL Server, a variable is typically known as a local variable is only available in the batch, stored procedure or code block in which it is defined. A local variable is defined using the Transact-SQL (T-SQL) DECLARE statement. The name of the local variable needs to start with the @ symbol as the first character of its name. A local variable can be declared as any system or user defined data type. Here is a typical declaration for an integer variable named @Count.

```
DECLARE @Count INT
```

More than one variable can be defined with a single DECLARE statement. To define multiple variables, with a single DECLARE statement, you separate each variable definition with a comma, like this:

```
DECLARE @Count INT, @X INT, @Y INT, @Z CHAR (10)
```

In the above declaration, 4 local variables with a single DECLARE statement. A local variable is initially assigned a NULL value. A value can be assigned to a local variable by using the SET or SELECT statement. On the SET command you specify the local variable and the value you wish to assign to the local variable. Here is an example of where I have defined my @Count variable and then initialize the variable to 1.

```
DECLARE @Count INT  
SET @Count = 1
```

Here is an example of how to use the SELECT statement to initialize the value of a local variable with the value returned from a select that fetches the total number of rows in the employee table.

```
DECLARE @ROWCNT INT  
SELECT @ROWCNT= (SELECT COUNT (*) FROM employee)
```

One of the uses of a variable is to programmatically control the records returned from a SELECT statement. You do this by using a variable in the WHERE clause. Here is an example that returns

all the Customers records in the Sales database where the Customer's Country column is equal to 'Germany'

```
Declare @Country varchar(25)
set @Country = 'Germany'
select CompanyName from Sales.dbo.Customer where Country = @Country
```

1.2 IF ... ELSE statemens

T-SQL has the **IF** statement to help with allowing different code to be executed based on the results of a condition. The "IF" statement allows a T-SQL programmer to selectively execute a single line or block of code based upon a Boolean condition. There are two formats for the "IF" statement, both are shown below:

Format one: IF <condition> <then code to be executed when condition true>

Format two: IF <condition> <then code to be executed when condition true> ELSE < else code to be executed when condition is false>

In both of these formats, the <condition> is a Boolean expression or series of Boolean expressions that evaluate to true or false. If the condition evaluates to true, then the "then code" is executed. For format two, if the condition is false, then the "else code" is executed. If there is a false condition when using format one, then the next line following the IF statement is executed, since no else condition exists. The code to be executed can be a single TSQL statement or a block of code. If a block of code is used then it will need to be enclosed in a BEGIN and END statement.

Let's review how "Format one" works. This first example will show how the IF statement would look to execute a single statement, if the condition is true. Here, the if statement tests whether a variable is set to a specific value. If the variable is set to a specific value, then, print out the appropriate message.

```
Declare @x int
set @x = 29
if @x = 29 print 'The number is 29'
if @x = 30 print 'The number is 30'
```

The above code prints out only the phrase "The number is 29", because the first IF statement evaluates to true. Since the second IF is false, the second print statement is not executed. Now the condition statement can also contain a SELECT statement. The SELECT statement will need to return value or set of values that can be tested. If a SELECT statement is used the statement needs to be enclosed in parentheses.

```

if (select count(*) from Sales.dbo.Customer
    where fname like '[A-D]%) > 0
    print 'A-D Customers are Found '

```

The above examples only showed how to execute a single T-SQL statement if the condition is true. T-SQL allows you to execute a block of code as well. A code block is created by using a "BEGIN" statement before the first line of code in the code block, and an "END" statement after that last line of code in the code block. Here is any example that executes a code block when the IF statement condition evaluates to true.

```

if db_name() = 'master'
    begin
        Print 'We are in the Master Database'
        Print ''
        Print 'So whatever code you execute must be done carefully'
    End
else if db_name() = 'Sales'
    begin
        Print 'We are in the test Sales' database
        Print 'So Enjoy'
    End

```

Sometimes you want to not only execute some code when you have a true condition, but also want to execute a different set of T-SQL statements when you have a false condition. If this is your requirement then you will need to use the IF...ELSE construct, that I called format two above.

Example:

Suppose we want to determine whether to update or add a record to the employee table in the HR database. The decision is based on whether the employee already exists in the HR.dbo.Employee table. Here is the T-SQL code to perform this existence test for two different EmployeeId's.

```

if exists(select * from HR.dbo.Employee
    where CustomerId = 'Abebe123')
    Print 'Need to update Record 'Abebe123'
else
    Print 'Need to add Employee Record 'Abebe123'

```

1.3 WHILE Loop With CONTINUE and BREAK Keywords

BREAK keyword will exit the stop the while loop and control is moved to next statement after the while loop. CONTINUE keyword skips all the statement after its execution and control is sent to first statement of while loop.

Syntax:

```
WHILE Boolean_expression
{ sql_statement | statement_block | BREAK | CONTINUE }
{ sql_statement | statement_block }
```

BREAK: Causes an exit from the innermost WHILE loop. Any statements that appear after the END keyword, marking the end of the loop, are executed.

CONTINUE: Causes the WHILE loop to restart, ignoring any statements after the CONTINUE keyword.

Example 1:

```
WHILE (SELECT AVG(ListPrice) FROM Production.Product) < 300
BEGIN
    UPDATE Production.Product
        SET ListPrice = ListPrice * 2
    SELECT MAX(ListPrice) FROM Production.Product
    IF (SELECT MAX(ListPrice) FROM Production.Product) > 500
        BREAK
    ELSE
        CONTINUE
END
```

Example 2:

```
DECLARE @intFlag INT
SET @intFlag = 1
WHILE (@intFlag <=5)
BEGIN
    PRINT @intFlag
    SET @intFlag = @intFlag + 1
END
```

Example 3: Usage of WHILE Loop with BREAK keyword

```
DECLARE @intFlag INT
SET @intFlag = 1
WHILE (@intFlag <=5)
BEGIN
    PRINT @intFlag
    SET @intFlag = @intFlag + 1
    IF @intFlag = 4
        BREAK;
END
```

GO

1.4.CASE... When expressions

Evaluates a list of conditions and returns one of multiple possible result expressions.

The CASE expression has two formats:

- The simple CASE expression compares an expression to a set of simple expressions to determine the result.
- The searched CASE expression evaluates a set of Boolean expressions to determine the result. Within a SELECT statement, the searched CASE expression allows for values to be replaced in the result set based on comparison values.

Syntax for The simple CASE expression:

```
CASE input_expression
    WHEN when_expression THEN result_expression [ ...n ]
    [ELSE else_result_expression]
END
```

Example:

```
USE HR;
GO
SELECT FName, AcademicRank =
    CASE Qualification
        WHEN 'Bsc' THEN 'Graduate Assistant'
        WHEN 'Msc' THEN 'Lecturer'
        WHEN 'Phd' THEN 'Assistant Professor'
        ELSE 'Technical Assistant'
    END,
    salary
FROM employee
ORDER BY sex;
GO
```

Syntax for Searched CASE expression:

```
CASE
    WHEN Boolean_expression THEN result_expression [ ...n ]
    [ ELSE else_result_expression ]
END
```

Example:

```
USE HR;
GO
SELECT FName, LName, AcademicRank =
CASE
    WHEN 'Bsc' THEN 'Graduate Assistant'
    WHEN 'Msc' THEN 'Lecturer'
    WHEN 'Phd' THEN 'Assistant Professor'
    ELSE 'Technical Assistant'
END FROM employee ORDER BY sex;
GO
```

1.5 Exercises

1. Use if ...Else statement and write T-SQL program to convert Letter grades into their corresponding Number grades
2. Use while... loop and write T-SQL program to find the factorial of a number
3. Use CASE.....When and classify the payment category of employees as follows:
 - a. For salary ≥ 8000High paid
 - b. For salary ≥ 4000Midium paid
 - c. For salary < 400Low paid

2. Laboratory session 2: User defined functions

Learning objective

At the end of this lab session, students will be able to:

- Create and alter user defined functions
- Use functions to solve problems

User defined functions are functions which are developed by the users of the database system to supplement and extend the built-in functions. A user-defined function takes zero, or more, input parameters and returns either a scalar value or a table. Input parameters can be any data type except timestamp, cursor, or table. User defined functions may take the following forms:

- a. Scalar user defined functions
- b. Inline table valued functions
- c. Multi statements table valued functions

2.1 Creating scalar user defined functions using SQL server:

Syntax:

```
CREATE FUNCTION [schema_name.]function_name ([@parameter_name1  
parameter_data_type],[@parameter_name2 parameter_data_type], [@parameter_name3  
parameter_data_type],...[parameter_nameN parameter_data_type]  
)  
RETURNS return_data_type  
[ WITH<function_option>]  
AS  
BEGIN  
function_body  
RETURN scalar_expression  
END
```

Where

Schema_name = the name of the function owner

function_name = any valid sysname for the function

@ parameter_name1@ parameter_nameN= the name of the parameter for the function

return_data_type = any scalar data type data type like int, Varchar, Nvarcahr, date, char, Ncahr, etc.

function_option = specification of whether to encrypt the function definition or not, schema bound or not, etc

Example: create a function that calculates the net pay of Instructos based on the tax dedcution logic shown in the next table below.

Assume we have a table named employee with the follwong attributes
id int primary key, fname varchar(20), sex char, salary float

Tax deduction logic in a given range

```
create function netPay(@sal float)
returns float
as
begin
declare @np float
if @sal<=150
    set @np=@sal
else if @sal<650
    set @np=@sal-0.10*@sal
else if @sal<1400
    set @np=@sal-(50+0.15*(@sal-650))
else if(@sal<2350)
    set @np=@sal-(162.50+0.2*(@sal-1400))
else if(@sal<3500)
    set @np=@sal-(352.50+0.25*(@sal-2350))
else if(@sal<5000)
    set @np=@sal-(640+0.30*(@sal-3500))
else
    set @np=@sal-(1090+0.35*(@sal-5000))
return @np
end
```

Salary	Tax rate (%)
0-150	0 (free from tax)
151-650	10
651-1400	15
1401-2350	20
2351-3500	25
3500-5000	30
>5000	35

When referencing a scalar user defined function, specify the function owner and the function name in two-part syntax as shown next:

```
select dbo.netPay(Instructor.salary) from Instructor
```

Setting Permissions for User-defined Functions

- User must have CREATE FUNCTION permission to create, alter, and drop user defined functions
- Users other than the owner must be granted EXECUTE permission on a function before they can use it in a Transact-SQL statement

- If the function is being schema-bound, you must have REEERENCE permission on tables, views, and functions referenced by the function. REFERENCE permissions can be granted through the GRANT statement to views and user defined functions as well as tables.
- If a CREATE TABLE or ALTER TABLE S statement references a user-defined function in a CHECK constraint, DEFAULT clause, or computed column, the table owner must also own the function.

2.2. Inline table valued functions

These type of functions are fucntions that return list of records/tables based on a logic defined using a single SQL statement.

Syntax:

```
CREATE FUNCTION [ schema_name. ] function_name
    ( [ { @parameter_name ] parameter_data_type
      [ ,...n ]
    ]
)
RETURNS TABLE
    [ WITH<function_option> [ ,...n ] ]
    [ AS ]
    RETURN [ ( [ select_stmt ] ) ]
[ ; ]
```

Example: create an inline table valued function to list the name of all female Instructors

```
create function fn_FemaleInstructors()
returns table
as
return select FirstName,MidleNamefrom Instructor where sex='F'
```

To call the function, run the function in following manner:

```
select * from dbo. fn_FemaleInstructors()
```

2.3 Multi statements table valued functions

These type of functions are fucntions that return list of records/tables based on a logic defined using many SQL statements that perform complex operations. In the same way that you use a view, you can usea table-valued function in the FROM clause of a Transact-SQL statement.

When using a multi-statement table-valued function, consider the following facts:

- The BEGIN and END delimit the body of the function
- The RETURNS clause specifies **table** as the data type returned.
- The RETUENS clause defines a name for the table and defines the format of the table.

The scope of the return variable name is local to the function.

Syntax:

```

CREATE FUNCTION [schema_name.] function_name
( [ @parameter_name parameter_data_type,
  [ ,...n ]
]
)
RETURNS @return_variable TABLE <table_type_definition>
[ WITH<function_option> [ ,...n ] ]
AS
BEGIN
    function_body
    RETURN
END

```

Example: create a multi-statement table-valued function that returns the name orFull names of employees, depending on the parameter provided.

Solution:

```

CREATE FUNCTION fn_Employees
(@length varchar(60))
RETURNS @fn_EmployeeTABLE(EmployeeIDint Primary Key Not Null,
EmployeeName varchar(16) Not Null)
AS
BEGIN
    IF (@length = 'ShortName')
        INSERT @fn_Employees SELECT EmployeeID, fNameFROM Employees
    ELSE IF(@length = 'FullName')
        INSERT @fn_Employees SELECT EmployeeID,fName + ' ' +MidName + ' ' +
        GfatherName FROM Employees
    RETURN
END

```

2.4. Altering Functions

You modify a user-defined function by using the ALTER FUNCTION statement

Syntax:

```

ALTER FUNCTION function_name
<New Function Content>

```

This example shows how to alter a function

```

ALTER FUNCTION dbo. fn_NetPay
<New Function Content here>

```

2.5. Dropping Functions

You can drop a user-defined function by using DROP FUNCTION statement.

Syntax: **DROP FUNCTION** *function_name*

Example: DROP FUNCTION dbo.fn_NetPay

2.6. Exercise

1. Create a function which gives you the maximum of two integers.
2. Suppose that we have two tables named as:
 - a. Student(fname, studID, sex)
 - b. Stud_Grade (studID, courseNo,CourseTitle, CrHr, Grade)
3. Based on the values that could be populated to these tables, create a function that returns the GPA of a student when you pass studID as an argument to the function.

The following table structures are taken from ABC publishing database. Refer these tables and write T-SQL statements to answer questions that follow.

Take the following assumptions:

- The attribute “**BookTitle**” in the book table can have values like C++, Database, Java, etc...
- The attribute “**City**” in the publisher table can have values like AA, Jimma, New York, London, etc...

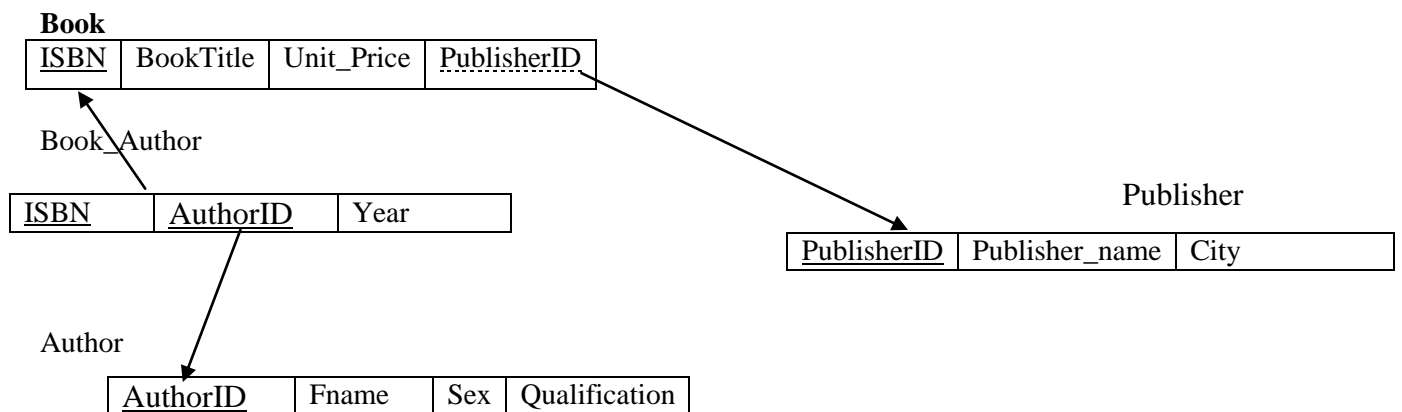


Table 2.1: Tables from ABC publishing database

1. Write a scalar function that returns the average price of Books published by male authors.
2. Write a scalar function that returns the total number of female Authors who published database books by **Jimma** publishers.
3. Write a scalar function that takes 'Publisher_Name' as an argument and return total number of authors who published books in the past ten years.
4. Create an inline table valued function that generate the list of all authors who published java books
5. Create a multi statement table valued function to find ID, name and sex of all authors who already published books. Call this function under a new scalar function that returns the total number of books published by female authors.

2.7 Creating function using Oracle SQL

Synax:

```
create [or replace] function [schema .] function
[( argumentdatatype
[, argument datatype]...
)]
returndatatype
{ is | as } { pl/sql_function_body };
```

If a function already exists, you may replace it via **create or replace function** command. If you use the **or replace** clause, any EXECUTE grants previously made on the function will remain in place.

Example:

Salary deduction in a given range

```
create or replace
function NetPay(sal float) return
is
Np float;
begin
if (sal<=150) then
Np:=sal;
elsif (sal<=650) then
NP:=sal-(sal-150)*.1;
Elsif(sal>=651) then
NP:=sal-(50+(sal-650)*.15)
else
NP:=sal-(112.50+(sal-1400)*.15);
end if;
return Np;
end;
```

Salary	Tax (%)
0-150	0
151-650	10
651-1400	15
>=1401	20

To call the above oracle function, use the following code

Select NetPay(NetPay) from dual;

N.B. Dual is a dummy table with one element in it. It is useful because Oracle doesn't allow statements without specifying the From clause.

Exercise

Show oracle equivalent SQL codes for the functions we created Using SQL server codes.

3. Laboratory session 3: Stored Procedures

Learning Objective

After completing this lesson, students will be able to

- Define what stored procedure is.
- Work with stored procedures.
- Create, execute, modify, and drop a stored procedure
- Use different types of parameters with stored procedures.

3.1 What is stored procedure?

Stored procedures are SQL codes consisting of declarative and procedural SQL statements stored in the catalog of a database that can be activated by calling it from a program, a trigger or another stored procedure. They are comparable to multi- statement functions, but they boast features and flexibilities that are not possible within functions. Some of the benefits of using stored procedures are:

- They offer improved performance because of compiled code.
- Stored procedures are pre-compiled, so the execution plan doesn't have to be figured out each time they're called. This can result in a significant performance improvement.
- Code reuse and abstraction
- Permissions can be granted for stored procedures while being restricted for the underlying tables. This allows the DBA to provide a method for SQL programmers and report writers to access and/or manipulate data in a safe way.

Since database operations can be performed inside the stored procedures, they provide a strong level of security. Instead of access being granted to the underlying object, permission can be granted only to the stored procedure. Essentially, stored procedures create a level of abstraction for permissions—instead of the user being granted SELECT, INSERT, UPDATE, or DELETE rights, the user can be granted EXECUTE rights to a stored procedure.

3.2. Stored Procedures vs Functions:

While stored procedures are very similar to functions, they differ in several ways. The most notable difference is that stored procedures can be used to affect/modify the state of the database while functions are always used to fetch/read the data stored in the database..

Functions can be used in a SELECT statement while stored procedures are not.

The result of Table-valued functions can be joined to tables, views, and even other functions.

Stored procedures, however, cannot be used in this way.

- Functions should always return a value while stored procedures may or may not return a value.

SQL Server Stored procedures are created with the CREATE PROCEDURE statement. The syntax is shown below:

Syntax:

```
CREATE PROCEDURE procedure_name
(@param1 data_type,@param2 data_type @param3 data_type.....)
AS
-- SQL statements
```

Example: Use the following schema to understand the solutions in this example:

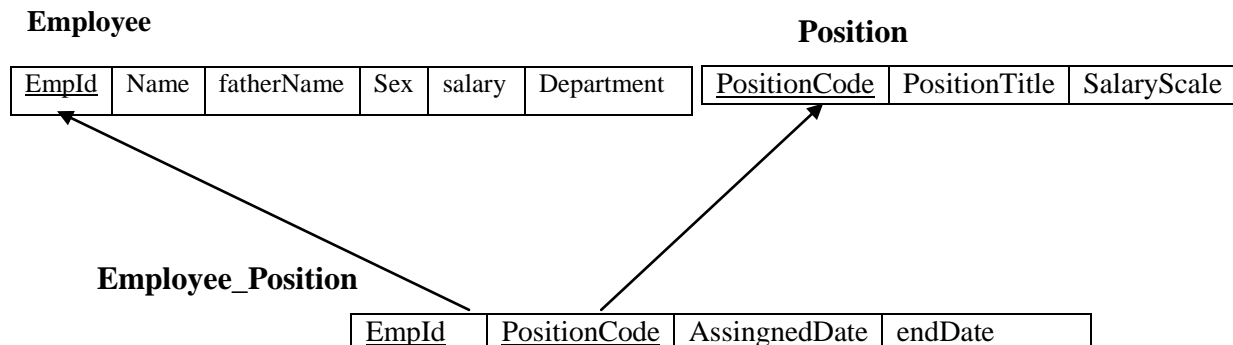


Table 3.1 Tables from HR database

Assume that we have several records in the above tables.

1. When PositionTitle is given as an argument, suppose that we want to see the list (name) of all employees together with the total number of years they have worked on that position. Write a stored procedure to achieve this purpose.

Solution:

```
CREATE PROCEDURE EMPLOYEEELIST(@PositionTitlevarchar(6))
AS
SELECT NAME,fatherName,dbo.fetchexperience(e.empId,p.positionCode) from employeee,
employee_positionep,
positon p where e.empId=ep.empId and
ep.positionCode= p.positionCode and p.positonTitle= @PositionTitle
```

The **fetchexperience** function used in this stored procedure can be defined as follows:

```
create function fetchexperience(@empIdint, @positionCodeint)
returnsfloat
as
begin
declare @activeExperienceint=0;
declare @pastExperienceint=0;
declare @totalExperienceint=0;
set @activeExperience=(select sum(YEAR(GETDATE()) - year(assignedDate))
fromemployee_position where assignEndDate is null and empId=@empId
andpositionCode=@positionCode)

set @pastExperience=(select sum(YEAR(endDate) - year(assignedDate))
fromemployee_position where endDate is not null and empId=@ empIdand
andpositionCode =@positionCode)

if(@activeExperience is null and @pastExperience is null)
set @totalExperience=0;
if(@activeExperience is null and @pastExperience is not null)
set @totalExperience=@pastExperience;
if(@activeExperience is not null and @pastExperience is null)
set @totalExperience=@activeExperience;
return @totalExperience;
end
```

2. Suppose that we decided to make a salary increment for each employee based on experience and sex. Based on the salary adjustment scale described below and write a stored procedure to effectively implement this scale. The stored procedure takes **sex** as parameter.

sex	Total Experience in year	Salary Increment
Female	>=3	20%
Female	<3	15%
Male	>=5	20%
Male	<5	15%

Assumption:

- Experience can be calculated based on assigned date and endDate
- An **Employee** may be assigned in different positions per year

Solution :

First, create a function to calculate the total experience of a person regardless of the type of position he/she has been assigned.

```

create function fetchTotalExperience(@empIdint)
returnsint
as
begin
declare @active int=0;
declare @past int=0;
declare @total int=0;

set @active=(select sum(YEAR(GETDATE()) - year(assignedDate))
fromemployee_position where assignEndDate is null and empId = @empId)

set @past=(select sum(YEAR(assignedDate) - year(endDate))
fromemployee_positionwhere endDate is not null and empId = @empId)

if(@active is null and @past is null)
    set @total=0;
if(@active is null and @past is not null)
    set @total=@past;

if(@active is not null and @past is null)
    set @total=@active;
    else
        set @total=@active+@past
return @total;
end

```

Then, create the stored procedure that will update the salary of all employees based on sex and experience

The above function will be called inside this procedure to check total experience and determine the amount of salary increment.

Solution:

Create procedure sp_increaseSal

as

Begin

```

updateEmployee set salary=salary+salary*.2 where
dbo.fetchTotalExperience(empId)>=3 and sex='F'

```

```

updateEmployee set salary=salary+salary*.15
where dbo.fetchTotalExperience(empId)<3 and sex='F'

```

```

updateEmployee set salary=salary+salary*.2
where dbo.fetchTotalExperience(empId)>=5 and sex='M'updateEmployee set
salary=salary+salary*.15 where dbo.fetchTotalExperience(empId)<5 and sex='M'

```

END;

3.3. Executing stored procedures

Syntax:

```
EXECUTE stored_procedureName Listofparameter_values
```

Example: EXECUTE fetchTotalExperience 11

3.4 .Dropping Stored procedures

```
DROP PROCEDURE stored_procedureName
```

3.5.Modifying stored procedures

```
ALTER PROCEDURE stored_procedureName  
AS New_Stored_procedure_definition
```

3.6 Exercise

Use the table structure shown below to answer question #1 and #2

Department

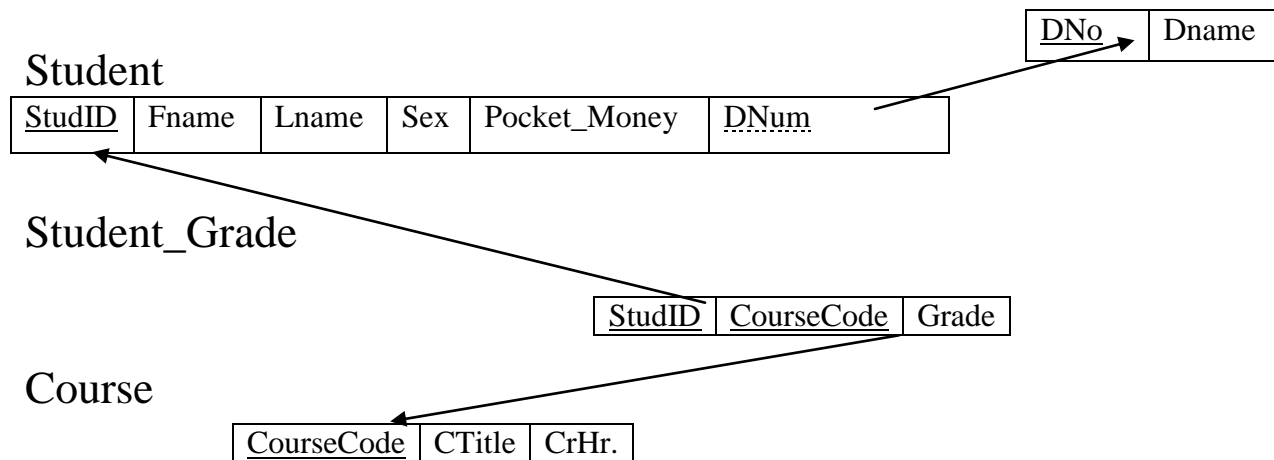


Table 3.2: Tables from Registrar database

1. Write a stored procedure that increment Pocket_Money by 50% for excellent students whose GPA is above 3.75. Use studentID as a parameter to the procedure
2. Write a stored procedure that display their name and department of all students who graduated with great distinction. Use the GPA threshold values given below to determine whether the status is distinction or not.
 - a. GPA ≥ 3.75 = Very Great Distinction
 - b. GPA ≥ 3.5 = Great Distinction
 - c. GPA ≥ 3.5 = Distinction
 - d. GPA ≥ 2.0 = Pass

Oracle syntax to create storedprocedure

```
create [or replace] procedure [schema .] procedure
[( argument [ in | out | in out ] datatype
[, argument [ in | out | in out ] datatype...
)]
{ is | as }
pl/sql_subprogram_body
```

Example: Write a stored procedure that deletes all courses with less than 5 credit hours

Solution

```
create or replace procedure removeCourse(creditHr number)
as
beyond_Limit Exception;
begin
if(creditHr<=4) then
    delete from course where crHr = creditHr ;
else
    raisebeyond_Limit;
end if;
Exception
    whenbeyond_limit then
        raise_application_error
        (-20001,'you can not delete records with more than 4 credits');
end;
/
```


4. Laboratory Session 4: Triggers

Learning Objective: At the end of this lesson, students will be able to:

- Create, alter and delete a trigger
- Explain the different types of triggers
- Recognizes the use of a trigger in the database systems
- Describe the use of triggers
- Discuss some of the facts and guidelines that implementers must consider when determining whether to use triggers instead of other tools.

4.1. What is Trigger?

A Trigger is a special kind of stored procedure that executes to automate control of your several databases. There are three categories of triggers in SQL Server: Data Manipulation Language (DML) triggers, Data Definition Language (DDL) triggers, and logon triggers.

To control data modifications within your database, you use a DML trigger. To control modifications of objects (such as databases or tables), you use a DDL trigger. To audit or control logins, you create a logon trigger.

Unlike standard stored procedures, triggers cannot be called directly and do not pass oracle parameters.

We can create triggers by using the CREATE TRIGGER statement.

4.2 .DML Triggers:

Triggers are often referred to as a special type of stored procedure. Just as stored procedures can be programmed to perform just about anything within a database, so can triggers. Data Manipulation Language (DML) triggers are often used to

- ◆ Make changes to other columns (in the same table or other tables within the database) in response to a data modification.
- ◆ Perform auditing by recording who made a data modification and when this modifications had happened.
- ◆ Roll back or undo data modifications.
- ◆ Provide specialized error messages.

The statement specifies the table on which a trigger is defined, the events for which the trigger executes, and the particular instructions for the trigger.

DML triggers can be programmed to perform either after the data modification or instead of the data modification. When a trigger is created, one of these choices is specified:

AFTER: An AFTER trigger fires after the data modification. In other words, the data modification occurs, and then the trigger fires in response to the data modification. AFTER triggers can only be specified on tables; they can't be specified on views.

◆**INSTEAD OF:** An INSTEAD OF trigger fires before the data modification. In other words, the data modification is prevented from occurring, and instead, the trigger is executed.

Triggers use two special tables that are only accessible to triggers. The data in these tables is available immediately after an INSERT, UPDATE or DELETE statement.

◆**Inserted** table: The inserted table holds data from the last INSERT or UPDATE statement.

◆**Deleted** table: The deleted table holds data from the last DELETE or UPDATE statement.

The use of INSERT and DELETE statements in these tables is straight forward. If one or more rows are added with an INSERT statement, the row(s) are held in the inserted table. If one or more rows are deleted with a DELETE statement, the row(s) are held in the deleted table.

Syntax:

```
CREATE TRIGGER trigger_name
On table_name
{ AFTER|For|INSTEAD OF } { INSERT|UPDATE|DELETE }
AS
begin
[IF(UPDATE(column_name)....]
[(AND|OR) UPDATE (column_name)...]
Sql statements
End
```

Example 1: To create a trigger that prevents modification of salary values in the employee table, we use the following code:

```
go
create trigger trig_emp_update on employee
for update
as if(update(salary))
begin
    Raiserror('Salary information should not be modified',16,1)
    rollback transaction
end
go
```

Example 2: The following code is also used to prevent deletion of more than 5 books at a time from books table.

```
go
Create trigger tr_book_delete on Books for delete as
if (select count(*) from deleted ) >5
begin
Raiserror('You cannot delete more than 5 books at once',16,1)
Rollback transaction
end
go
```

Example 3:

The following trigger code is used to automatically increment (update) the salary of an employee by 50% when she/he is assigned to more than one project.

Take the Employee table with attributes (empId, FName, sex, salary) and

Emp_Project(empId,projectCode, assignedDate, endDate) tables for this problem.

Solution

```
CREATE TRIGGER trig_Salary_Update on Emp_Project for Insert AS
BEGIN
If(select count(*) from Emp_Projectep,insertedi where ep.empId=i.empId)>=2
    Update employee set salary=salary + salary*.5 where empid in(select empid from
inserted)
END
```

4.3. DDL Triggers

DDL triggers respond to DDL events. DDL events are created when DDL statements are executed. DDL statements that can fire triggers include CREATE, ALTER, DROP and GRANT.

Syntax

```
CREATE TRIGGER trigger_name
ON {DATABASE | SERVER}
FOR event_type
AS
Begin
Trigger code
End
```

Example :

The following DDL trigger is used to prevent the modification of any tables within the Current database:

```
CREATE TRIGGER trgNoTableChange
ON DATABASE
FOR ALTER_TABLE
AS
PRINT 'Tables may not be modified'
ROLLBACK;
```

4.3. Dropping a Trigger

You can remove a trigger by dropping it. Triggers are dropped automatically whenever their associated tables are dropped.

Syntax: `DROP TRIGGER trigger_name`

4.4 Disabling and Enabling a Trigger

You can enable or disable a specific trigger, or all triggers on a table. When a trigger is disabled, it is still defined for the table, however, when an INSERT , UPDATE, or DELETE statements is executed against the table, the actions in the trigger are not performed until the trigger is re-enabled.

SYNTAX:

```
ALTER TABLE TABLE_NAME
{ENABLE|DISABLE} TRIGGER
TRIGGER_NAME
```

Example: to disable the trig_emp_update trigger we created in the employee table:

```
ALTER TABLE Employee DISABLE TRIGGERtrig_emp_update
```

4.5. Creating trigger using Oracle syntax:

Syntax:

```
create [or replace] trigger [schema .] trigger
{ before | after | instead of }
{ dml_event_clause
| { ddl_event...
| database_event]...
}
on { [schema .] schema | database
[when ( condition ) ]
{ pl/sql_block
```

Example: Create a trigger that can perform the following task:

When the credit hour value of a record at the course table is modified, insert the copy of the modified record into another table called course_audit table.

```
create or replace trigger Course_BEF_UPD_ROW
```

```

before update on Course
for each row
begin
insert into COURSE_AUDIT
(Cno,CTitle, CrHr, Audit_Date)
values
(:old.Cno,:old.CTitle,:old.CrHr, Sysdate);
end;
/

```

4.6 Calling Procedures within Triggers

Rather than creating a large block of code within a trigger body, you can save the code as a stored procedure and call the procedure from within the trigger, by using the **call** command.

Example:

If you create an INSERT_STUDENT_AUDIT_DUP procedure that inserts rows into STUDENT_AUDIT_DUP table, you can call it from a trigger on the STUDENT table, as below:

```

CREATE OR REPLACE TRIGGER student_aft_ins_ROW
after insert on STUDENT
for each row
begin
call INSERT_STUDENT_AUDIT_DUP(:new.STUFID, :new.FNAME,:new.SEX);
end;

```

4.7. Exercise

Use the table structure shown on Table 3.1 from HR database to provide solutions in this exercise

1. Create a trigger that transfers data from one table to another table.
2. Create a trigger that guarantees that the total number of positions that an employee actively works on is not more than 2 positions.
3. Create a trigger called **AgeLimit** that prevent assignment of a candidate employee to any position if the person is within the age of 50 or more years
4. Disable the AgeLimit trigger
5. Remove the AgeLimittrigger.

5. Laboratory Session 5: Configuring Server and Database security

Learning Objective:

At the end of this lesson, students will be able to:-

- Describe the security options available
- Create accounts and roles
- Granting Permissions to users and roles
- Revoke permissions

Security configuration primarily focuses on prevention of unauthorized access of Server and Database resources.

To give you an idea of how many security options you have, the following list is categorized by SQL Server's three securable scopes:

1. **Server:** refers to objects at the server level such as login
2. **Database:** refers to objects at the database level such as users and schemas
3. **Schema :** refers to objects at the schema such as tables, views and functions

SQL Server supports the pre-packaged permissions concept. These are known as roles, you can think of them as a one-stop shop that allows you to grant permissions. The following table lists fixed server roles and fixed database-level roles along with the permissions they provide.

SQL Server Fixed Server Roles

Name	Permission Available
Bulkadmin	Run the BULK INSERT command
Dbcreator	Create, change, restore, or drop a database
Diskadmin	Administer disk files
Processadmin	End SQL Server processes
securityadmin	Set server and database-level permissions; set password
Serveradmin	Shut down the server; modify server configuration values
Setupadmin	Manage linked servers; run system stored procedures
Sysadmin	Perform any administrative task on the server

SQL Server Fixed Database Roles

Name	Permission Available
Public	Default role for all database users
db_accessadmin	Maintain access permissions to the database
db_backupoperator	Archive the database
db_datareader	Read all data from any user table
db_datawriter	Make any modifications to any user's data
db_ddladmin	Execute any DDL command in any database
db_denydatareader	Blocked from reading data in a database
db_denydatawriter	Prevented from making any data modifications
db_owner	Perform all setup and database maintenance
db_securityadmin	Administer permissions and roles

5.1 Creating Logins

Most Windows users need SQL Server login account to connect to SQL Server. This topic shows how to create a SQL Server login account.

To create a SQL Server login that uses Windows Authentication using SQL Server Management Studio (SSMS)

1. In SQL Server Management Studio, open Object Explorer and expand the folder of the server instance in which to create the new login.
 2. Right-click the Security folder, point to New, and then click Login.
 3. On the General page, enter the name of a Windows user in the Login name box.
 4. Select Windows Authentication.
 5. Click OK.
6. To create a SQL Server login that uses SQL Server Authentication using SSMS
1. In SQL Server Management Studio, open Object Explorer and expand the folder of the server instance in which to create the new login.
 - 1) Right-click the Security folder, point to New, and then Click Login.
 - 2) On the General page, enter a name for the new login in the Login name box.

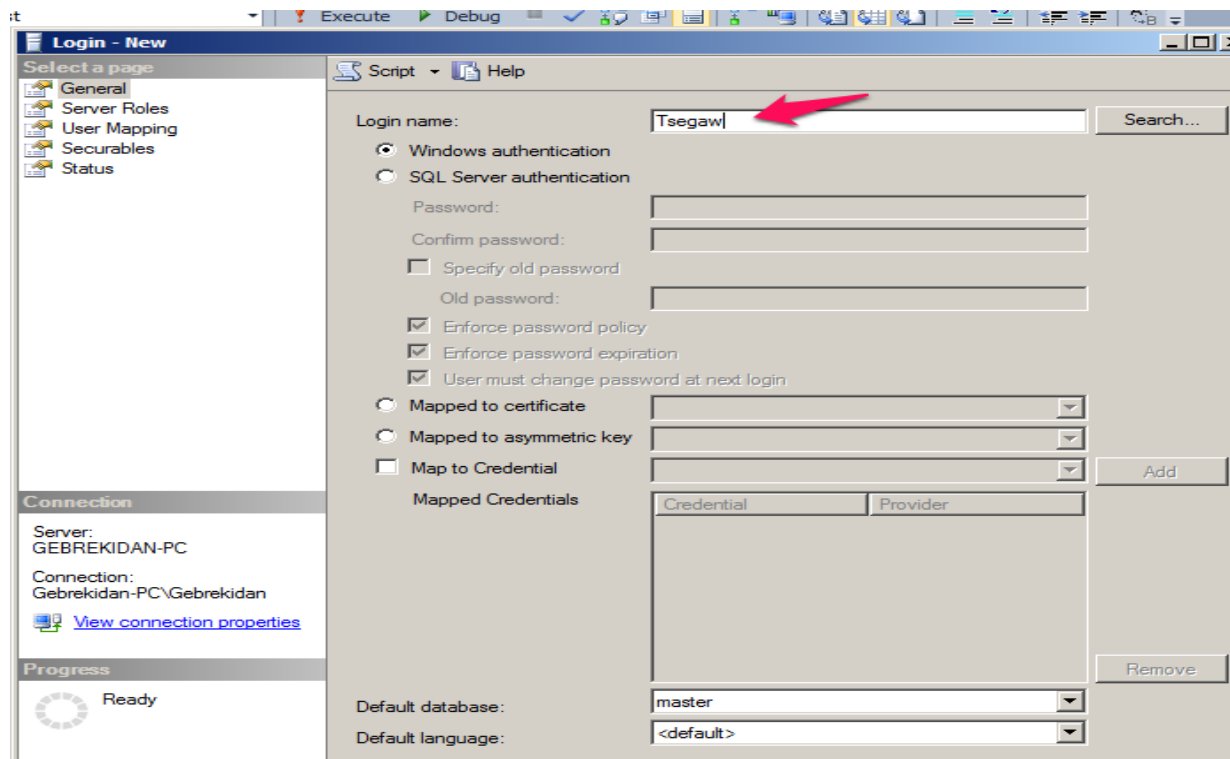


Figure 5.1 Creating New Login

- 3) Select SQL Server Authentication. Windows Authentication is the more secure option.
- 4) Enter a password for the login.
- 5) Select the password policy options that should be applied to the new login. In general, enforcing password policy is the more secure option.
- 6) Click OK.

To create a SQL Server login that uses Windows Authentication using Transact-SQL code

In the New Query Editor, enter the following Transact-SQL command:

CREATE LOGIN <Dormain\name of Windows User> FROM WINDOWS

Example: CREATE LOGIN [computer20\Zewdu] from windows

To create a SQL Server login that uses SQL Server Authentication using T-SQL code,

Enter the following Transact-SQL command in the Query editor:

Syntax:

CREATE LOGIN <login name> WITH PASSWORD = '<password>'

Example: CREATE LOGIN Abebe WITH PASSWORD='abc'

We can also drop login by using the DROP LOGIN login_name statement

5.2 Creating and Managing database Users

To create a database user using SQL Server Management Studio:

1. In SQL Server Management Studio, open Object Explorer and expand the Databases folder.
2. Expand the database in which to create the new database user.
3. Right-click the Security folder, point to New, and then click User.
4. On the General page, enter a name for the new user in the User name box.

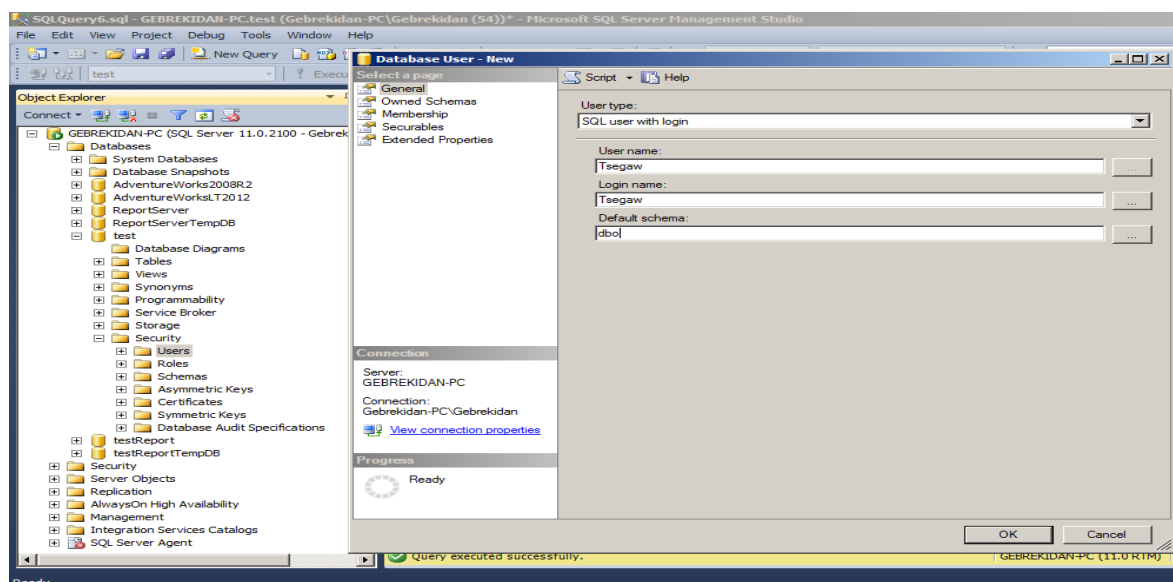


Figure 5.2: creating Database user

5. In the Login name box, enter the name of a SQL Server login to map to the database user.
6. Click OK.

To create a database user using T-SQL code:

CREATE USER <new user name> **FOR LOGIN** <login name>

Example: CREATE USER Hagos for login Hagos

We can also create a database user and assign a default schema

Example: Create user Mohamed for login Mohamed with default_schema =dbo

5.3 Creating Schemas

In SQL Server, schema is an object that conceptually holds definitions for other database objects such as tables, views, stored procedures etc. The main advantage of creating a schema is that you can grant permissions to database objects by a single CREATE SCHEMA statement.

Syntax:

CREATE SCHEMA schema_name_clause[<schema_element> [...n]]

<schema_name_clause> ::=

```
{
  schema_name
| AUTHORIZATION owner_name
| schema_name AUTHORIZATION owner_name
}
```

<schema_element> ::=

```
{
  table_definition | view_definition | grant_statement |
  revoke_statement | deny_statement
}
```

Example: we can create a student schema owned by Hagos as follows

Create schema student Authorization Hagos

Create table stud (fname varchar(20), Id int, sex char(6))

We can use the DROP SCHEMA schema_name statement to remove schema from the database. However, if the schema contains objects, you cannot DROP it.

5.4. Creating Roles

Roles are groups that have specific access rights and permissions. These are a random set of privileges that is granted to users.

There are three types of roles in SQL server: Fixed server roles Fixed database roles and

- User defined database roles

We cannot create or change server level roles. After you create a database level role, configure the database-level permissions of the role by using GRANT, DENY, and REVOKE.

Users can be added to a fixed server level role using sp_addsrvrolemember stored procedure

Example: To add a login account 'Hailu' to dbcreator server role, we will write the following statement:

```
sp_addsrvrolemember Hailu,dbcreator,
```

Users can also be added to a database role (either fixed or user defined database roles), using sp_addrolemember stored procedure.

Example: To add a database user Zewdu to db_owner fixed database role, we will write the following statement:

```
sp_addrolemember dbcreator, Zewdu
```

Creating user defined database roles:

Syntax: CREATE ROLE role_name[AUTHORIZATION owner_name]

role_name: Is the name of the role to be created.

owner_name: Is the database user or role that is to own the new role. If no user is specified, the role will be owned by the user that executes CREATE ROLE.

Example: CREATE ROLE student

-To add user zewdu to be the member of student role, EXECUTE sp_addrolemember 'student',zewdu'

-To add user zewdu to be the member of fixed database role, EXECUTE sp_addrolemember

Example: sp_addrolemember 'db_accessadmin','zewdu'

- We can drop roles using the Drop role role_name code

- we can remove membership from roles using sp_droprolemember stored procedure

Example; sp_droprolemember db_accessadmin,'zewdu'

Note: SQL server 2012 added the following new features with respect to security which is not available in previous versions

- It is possible to create user defined server roles.
- Inside contained databases, database users can be created with password without mapping to any login account

5.5 Granting Permissions

The GRANT statement is used to give privilege to users or roles.

Syntax:

```
GRANT { ALL [ PRIVILEGES ] }  
      | permission [ ( column [ ,...n ] ) ] [ ,...n ]  
[ ON [ class :: ] securable ] TO principal [ ,...n ]  
[ WITH GRANT OPTION ] [ AS principal ]
```

Note: if the permission is given via the [WITH GRANT OPTION],all users in the TO clause can themselves pass on the privilege to other users.

Examples: GRANT SELECT ON student to u1

```
GRANT SELECT, INSERT, UPDATE (slalry) ON employee to u1  
      GRANT SELECT ON student to u1 WITH GRANT OPTION  
      GRANT CREATE TABLE TO u1 WITH GRANT OPTION
```

5.6 Revoking Permissions

Revoke statement is used to withdraw privileges from a user without deleting that user.

Syntax:

```
REVOKE [ GRANT OPTION FOR ]  
      {  
        [ALL [ PRIVILEGES ] ]  
        |  
          permission [ ( column [ ,...n ] ) ] [ ,...n ]  
      }  
[ ON [ class :: ] securable ]  
{TO | FROM } principal [ ,...n ]  
[ CASCADE ] [ AS principal ]
```

[GRANT OPTION FOR]:-

Indicates that the ability to grant the specified permission will be revoked. This is required when you are using the CASCADE argument. If the principal has the specified permission without the GRANT option, the permission itself will be revoked.

class

Specifies the class of the securable on which the permission is being revoked. The scope qualifier :: is required.

securable

Specifies the securable on which the permission is being revoked.

TO | FROM *principal*

Is the name of a principal. The principals from which permissions on a securable can be revoked.

CASCADE

Indicates that the permission that is being revoked is also revoked from other principals to which it has been granted by this principal. When you are using the CASCADE argument, you must also include the GRANT OPTION FOR argument.

Examples: REVOKE DELETE ON employee from u1

REVOKE DELETE,INSERT ON employee from u1

REVOKE GRANT OPTION FOR DELETE ON EMPLOYEE FROM U1 CASCAD

5.7 Exercise

Suppose that you are assigned to work as a database administrator (DBA) for XYZ Company. The following sample tables and schemas are taken from the library database of this company.

Tables

- a. Student (fname,studId,sex,department),
- b. Instructor (fname, InstId,sex,Academic_Rank, salary),
- c. Book (ISBN,Title, Year)
- d. Book_Author (ISBN, AuthorID)
- e. Author(AuthorID,fname, lname,sex)
- f. Book_Borrowing(instID,ISBN,Due_DATE)

Schemas

- a. Library_Members
- b. Library_Resources
- c. Author

1. Create three login accounts named as Aster, Tolossa and Kedir. Under the **company** database, create three user accounts using the same name for login name and user mappings (Example: for user Aster, associate it with Login name=Aster).
2. Create the student and Instructor tables under the **Library_Members** schema and assign Aster as the owner of this schema
3. Under **Library_Resource** schema, create a view for all books of **Advanced Networking** (assume that the Title column in the book table has values like Advanced Networking) and assign Kedir as the owner of this schema
4. Create three roles namely student, Librarian and Instructor
5. Grant the following permissions to Librarian:
 - a. Select, insert and delete records on Book table under **Library_Resource** schema
 - b. Only update the Due_DATE values in the **Book_Borrowing** table
 - c. Full control on the Author schema
6. Grant the permission to kedir to see all records on the author table. Grant this permission to Kedir so that he can also grant this permission to other users.
7. Assign Aster to the member of Librarian
8. Assign Kedir to a fixed server role so that he can set server and database-level permissions and set password.
9. Add Kedir to a fixed database role so that he can make any modifications to any user's data
10. Suppose that we have thousands of users accessing the Library system. Grant a permission to read Title and Year of all Books to Every user in the database
11. Use the following security information to create a new login account and corresponding user using your name under Library Database

- a. Library database as default database
 - b. **Library_Resources** as a default schema
 - c. Grant a persimmon to alter any login
 - d. Grant a persimmon to create table in any database
12. Write SQL statement to view the detail information from DBMS catalog about
- a. All server level accounts/principals
 - b. All database level accounts/principals
13. Deny all permissions given to Kediri
14. Remove Aster from the role she has been assigned
15. Remove Tolossa from the role he has been assigned

6. Laboratory Session 6: Configuring oracle database Security

6.1. Creating user accounts

Syntax:

Create user <user_Name> identified by <password>

Example: Create user student identified by abc

6.2 Granting permission

Grant permissionName to username

Example: grant create session to student

6.3. Assigning membership to roles

Grant roleName to username

Example: Grant sysdba to student

6.4. Granting permission

Revoke permissionName to username

Example: Revoke select from student

6.5. Exercise

1. Practice the exercise given under Lab 5 using Oracle codes

7. Laboratory Session7: Managing Transactions

Learning Objective

At the end of this lesson, students will able to:

- Define the concepts of transactions
- Explain transaction modes and transaction isolation levels
- Apply transaction concepts to provide programming solutions to business problems

What a transaction is?

A transaction specifies a sequence of Transact-SQL statements that is used by database programmers to package together read and write operations, so that the database system can guarantee the consistency of data.

Transactions group a set of tasks into a single execution unit. Each transaction begins with a specific task and ends when all the tasks in the group successfully complete. If any of the tasks fail, the whole transaction fails. Therefore, a transaction has only two results: success or failure. Incomplete steps result in the failure of the transaction.

7.1. Transaction management modes

There are three transaction modes. These are auto commit mode, implicit mode and explicit mode.

Auto commit mode

It is the default transaction management mode of the SQL Server Database Engine. Every Transact-SQL statement is committed or rolled back when it completes. If a statement completes successfully, it is committed. If it encounters any error, it is rolled back. A connection to an instance of the Database Engine operates in auto commit mode whenever this default mode has not been overridden by either explicit or implicit transactions.

Implicit mode

Specifies any single INSERT, UPDATE, or DELETE etc... statement as a Transaction unit. When set ON, SET IMPLICIT_TRANSACTIONS sets the connection to implicit transaction mode. When OFF, it returns the connection to auto commit transaction mode.

Explicit mode

Generally, a group of Transact-SQL statements, where the beginning and the end of the group are marked using statements such as BEGIN, COMMIT and ROLLBACK TRANSACTION

Users can group two or more Transact-SQL statements into a single transaction using the following statements:

- Begin Transaction
- Rollback Transaction
- Commit Transaction

If anything goes wrong with any of the grouped statements, all changes need to be aborted.

Syntax:

```
BEGIN { TRAN | TRANSACTION }
      [ {transaction_name | @tran_name_variable }
        [ WITH MARK [ 'description' ] ]
      ]
[ ; ]
```

transaction_name

Is the name assigned to the transaction. Use transaction names only on the outermost pair of nested BEGIN...COMMIT or BEGIN...ROLLBACK statements.

@tran_name_variable

Is the name of a user-defined variable containing a valid transaction name. The variable must be declared with a **char**, **varchar**, **nchar**, or **nvarchar** data type.

WITH MARK ['description']

Specifies that the transaction is marked in the log. *Description* is a string that describes the mark. If WITH MARK is used, a transaction name must be specified. WITH MARK allows for restoring a transaction log to a named mark.

How a programmer marks the transaction boundary?

Transaction mode	Transaction Boundary	
	Begin	Commit/Rollback
Auto commit	X	X
Implicit	X	1.
Explicit	2.	3.

Auto commit: it is the default mode and there is no need to specify the begin and end point of SQL batch. Every single SQL statement is considered as one transaction

Implicit: the programmer has to clearly mark the end of the transaction by either commit or rollback

Explicit: Every SQL batch between begin and end is considered as one transaction. The programmer has to clearly mark the begin and end of the transaction by either commit or rollback.

7.2. Transaction Isolation models

SQL Server offers five different transaction isolation models. Before taking a detailed look at SQL Server's isolation models, we must first explore several of the database concurrency issues that they try to deal with:

- **Dirty Reads** occur when one transaction reads data written by another, uncommitted, transaction. The danger with dirty reads is that the other transaction might never commit, leaving the original transaction with "dirty" data.
- **Non-repeatable Reads** occur when one transaction attempts to access the same data twice and a second transaction modifies the data between the first transaction's read attempts. This may cause the first transaction to read two different values for the same data, causing the original read to be non-repeatable
- **Phantom Reads** occur when one transaction accesses a range of data more than once and a second transaction inserts or deletes rows that fall within that range between the first transaction's read attempts. This can cause "phantom" rows to appear or disappear from the first transaction's perspective.

SQL Server's isolation models each attempt to conquer a subset of these problems, providing database administrators with a way to balance transaction isolation and business requirements. The five SQL Server isolation models are:

- The **Read Committed Isolation Model** is SQL Server's default behavior. In this model, the database does not allow transactions to read data written to a table by an uncommitted transaction. This model protects against dirty reads, but provides no protection against phantom reads or non-repeatable reads.
- The **Read Uncommitted Isolation Model** offers essentially no isolation between transactions. Any transaction can read data written by an uncommitted transaction. This leaves the transactions vulnerable to dirty reads, phantom reads and non-repeatable reads.
- The **Repeatable Read Isolation Model** goes a step further than the Read Committed model by preventing transactions from writing data that was read by another transaction until the reading transaction completes. This isolation model protect against both dirty reads and non-repeatable reads.
- The **Serializable Isolation Model** uses range locks to prevent transactions from inserting or deleting rows in a range being read by another transaction. The Serializable model protects against all three concurrency problems.
- The **Snapshot Isolation Model** also protects against all three concurrency problems, but does so in a different manner. It provides each transaction with a "snapshot" of the data it requests. The transaction may then access that snapshot for all future references, eliminating the need to return to the source table for potentially dirty data.

If you need to change the isolation model in use by SQL Server, simply issue the following command:

SET TRANSACTION ISOLATION LEVEL <level>

Here <level> is replaced with any of the following keywords:

READ COMMITTED

- READ UNCOMMITTED
- REPEATABLE READ
- SERIALIZABLE
- SNAPSHOT

Example: Take the tables given below named as student and department to store student and department information respectively.

Student

Stid	Fname	Dno	
1	Abebe	4	

Department

Dnum	Dname
4	Mathematics

Suppose that Mathematics department has undergone a name change to “computer science” with a value of Dnum field =5 and student Abebe has changed his department to computer science and assigned a new **Stid**=2.

In order to accomplish all these changes without compromising database consistency, we need to group the following update actions in one single transaction.

The update statements to change the values for stid and Dno of the student

- The update statements to change the values for Dnum and Dname of the department

```
CREATE PROCEDURE changeval (@oldstidint, @newstidint, @oldDnumint,  
@newDnumint, @oldDnamevarchar(20), @newDnamevarchar(20))
```

as

```
BEGIN TRANSACTION T1
```

```
Update student set stid =@newstid where stid=@oldstid
```

```
Update department set dnum =@newDnum where dnum=@oldDnum
```

```
Update department set dname =@newDname where Dname=@oldDname
```

```
If (@@error<>0)
```

```
Rollback transaction T1
```

```
Commit
```

7.3 Exercise

Take the following table structure to answer the questions that follow

Item

ItemCode	ItemName	PurchaseUnitPrice	InitialQuantity	QunatityInStock
387	Copmputer	9580	50	15

Sales

ItemCode	Selling_UnitPrice	QunatitySold	SalesDate
387	1000	35	50

1. Apply the concept of transaction to properly handle the total quantity of items in stock whenever a given set of items are sold. (**Hint:** $QunatitySold + QuantiyInStock = InitialQuantity$)
2. Suppose that a bank customer has deposited money in his saving account. The information about deposit of each customer is stored in the customers table with Accno, Fname, Lname, and Balance columns. Write a stored procedure to control effective transfer of money from one customer account to another. Use the concept of transaction to provide solution for this exercise.

8. Laboratory Session 8: Data Import and Export

Learning Objective

At the end of this lesson, students will be able to:

- Load data from external sources to SQL Server
- Export data from SQL Server to other formats

Introduction

Import and export is the process of loading and unloading data from and to information repositories (such as SQL Server, Oracle).

How to import or export data by using the SQL Server Import and Export Wizard:

8.1. Importing from a text file

In this example, we create a simple text file that contains three columns of information. Each column is separated from its neighbor by a comma. We also have a destination table in the database that awaits its new information.

1. **To initiate the import process, begin by choosing the Import option from the database's**

Tasks menu.

This launches the Import and Export Wizard. Use the Next button to advance through this utility; the Back button takes you back one panel.

2. **Choose the source of your imported information, along with the server name.**

For the data source, a drop-down box allows you to select from one of many data access methods. For this example, use the Flat File Source. You also need to provide details on a number of other context-sensitive settings. In the case of a flat file, you must instruct SQL Server on the file's location and format, whether there's a header row, what delimiter to use, and so on.

3. **Select the destination for your information.**

If there's not a table already in place, don't worry; SQL Server will create a new one for you. Before launching a massive import job, you'd be wise to preview the information.

4. **Review the data type mappings for your upcoming import.**

At this point, SQL Server gives you the ability to determine what should happen in the case of an error.

5. **Schedule the job.**

You can elect to have the export task run immediately, or schedule it using the SQL Server Integration Services package. For the purposes of this example, run the job right away.

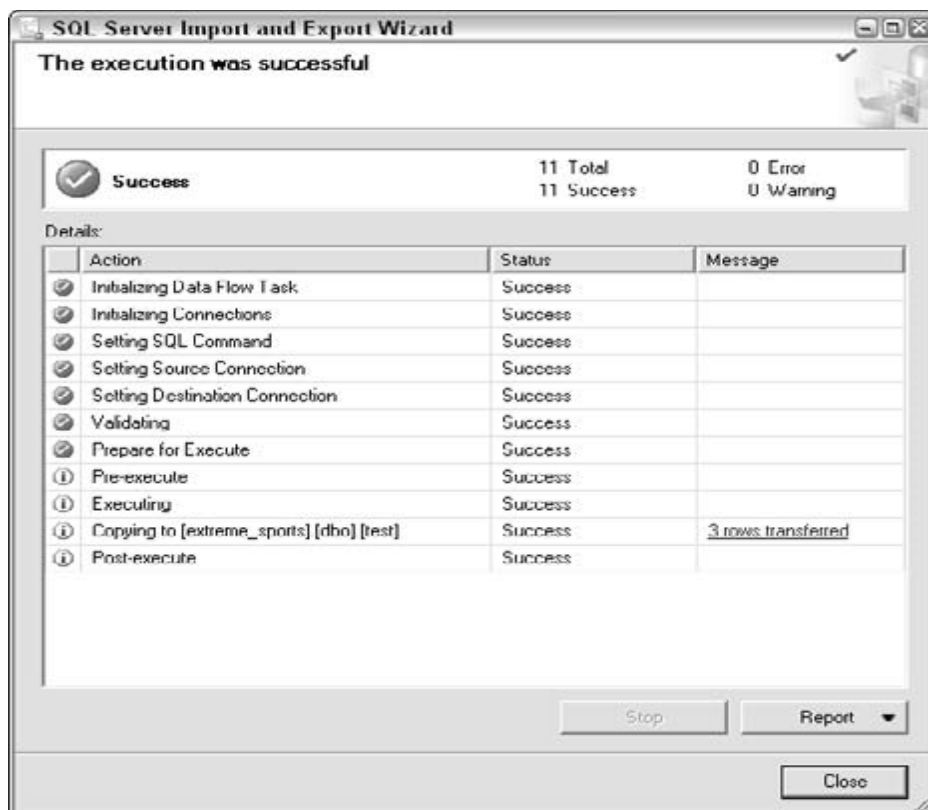
6. Review your choices from the wizard.

SQL Server gives you one more chance to check the details of the upcoming import job.

7. Run the job.

As with many administrative tasks, SQL Server displays a dialog box that summarizes the job's progress. Pay close attention to the details contained in this dialog box because errors are prominently displayed.

The following figure shows that the import process finished successfully.



8.2. Exporting to a spreadsheet

To extract information from your SQL Server database:

1. Initiate the export process by choosing the Export option from the database's Tasks menu.

This launches the Import and Export Wizard. Use the Next button to advance through this utility; the Back button takes you back one panel.

2. Choose the source of your exported information, along with the server name.

For the data source, a drop-down box allows you to select from one of many data access methods. For this example, use the SQL Server Native Client 10.0. You also need to decide the style of authentication to follow, along with the database name from which you wish to export information.

3. Select the destination for your information.

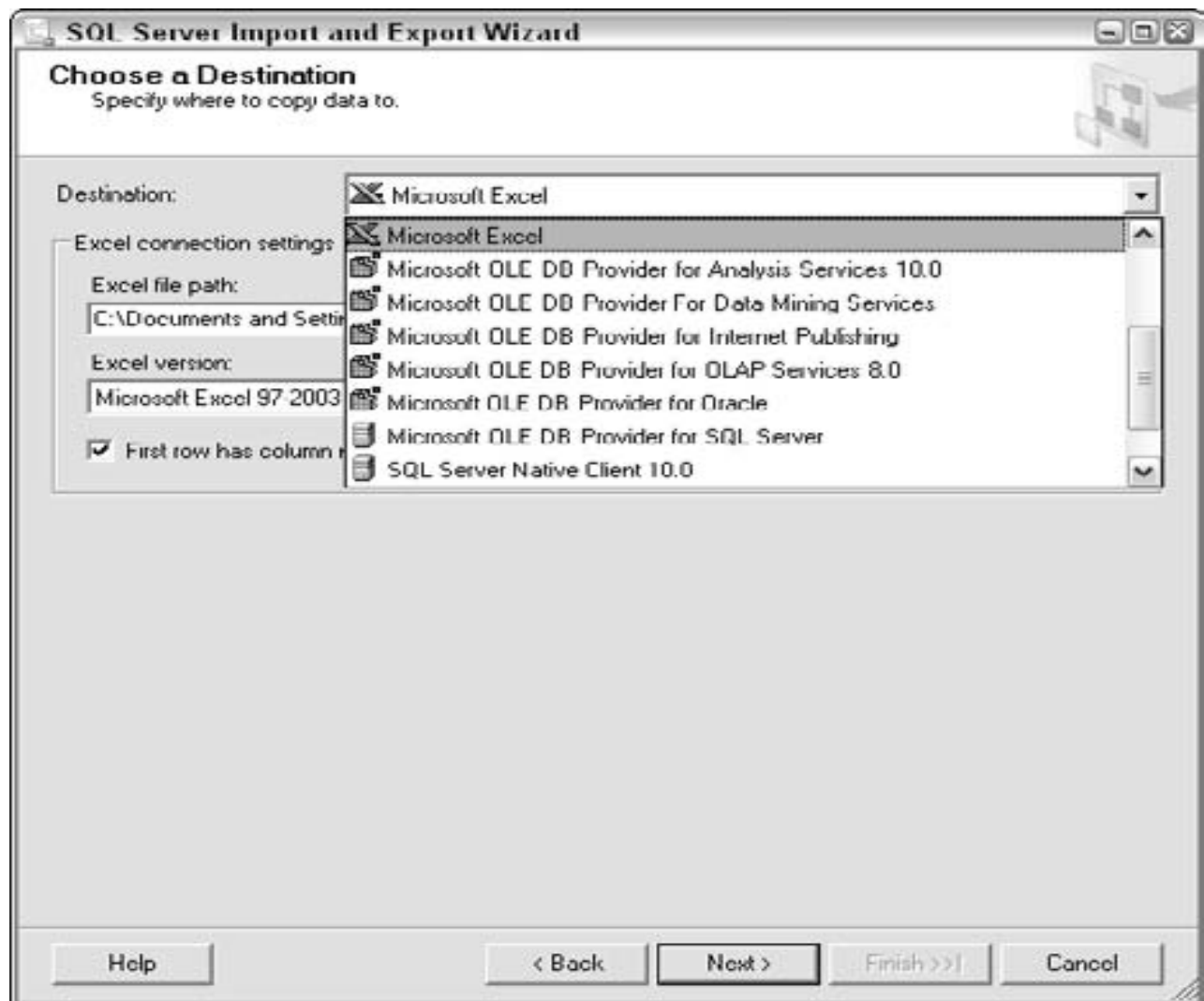
Figure 2-13 shows the wide variety of possible destinations. Microsoft Excel is the destination for the data. You also see that context-sensitive connection settings are dependent on the type of export that you're performing.

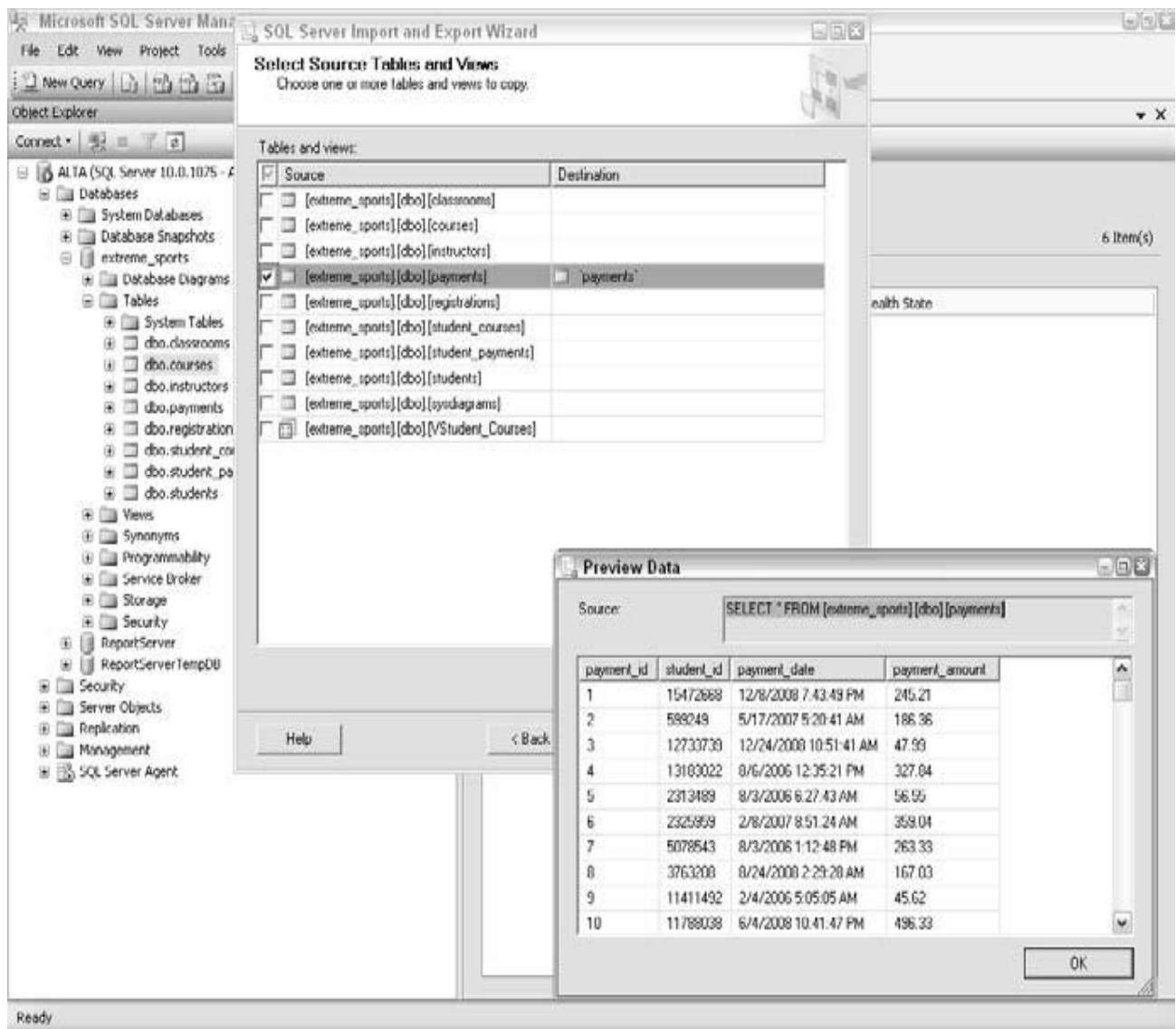
4. Decide whether you want to copy information, or construct a query to retrieve a subset of all available data.

In many cases, you'll simply want to copy information from your SQL Server database. However, you have the flexibility to create a specialized query if needed.

5. Select one or more tables and views from which to export data.

You can also edit your column mappings, as well as preview the upcoming export from this dialog box.





8.3. Exercise

1. Export an employee table from emp database to excel file called 'employeelist'
2. Import data that have been created using excel file called 'studentlist' to a database called registrar. Note: the excel file contains the following:

Id,	Name	Dept
123	Abebe	IS
124	Habtamu	IT

3. Repeat question 1 to using Microsoft Access and SQL Server as source and Destination Databases respectively.

9. Laboratory Session 9: Backup and Restore configuration

Learning Objective

At the end of this lesson, students will be able to:-

- Take backup
- Restore data in case of database failure or corrupted situation that need recovery

Introduction

Backup and restore process is an important database administration task that allows an entire database to be recovered to any given point in time.

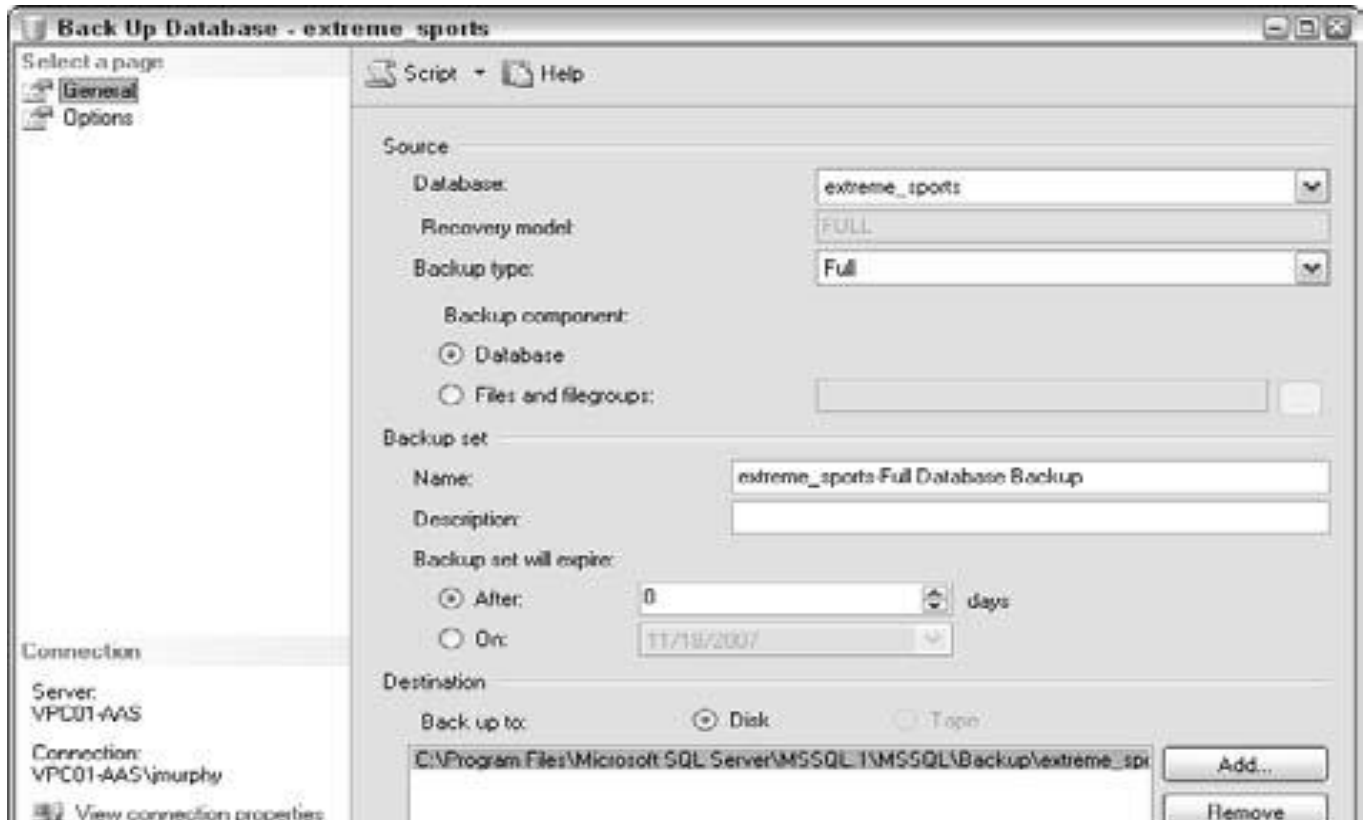
SQL Server supports three types of database backup: full, differential, and transaction log backups.

- **Full back up:-** A full database backup is a complete copy of the database. Full backups provide a known point from which to begin the database restore process.
- **Differential backup:-** A differential backup copies only the database pages modified after the last full database backup. Frequent differential backups minimize the number of transaction log backups that need to be applied to bring your database up to the last current transaction.
- **Transaction log backup:-** The transaction log backup copies the changes that have occurred since the last full or differential backup. You can make multiple transaction log backups per day depending on the activity level of your system. Transaction log backups may be applied after the last full or differential backup has been restored.

9.2. Backing up data

You can launch SQL Server's backup capabilities from the SQL Server Management Studio database's Tasks menu.

1. Right click on the target database and Select the Back Up menu option. You will be presented with a dialog box similar to the one shown below:-



2. Fill in values for all the relevant fields and click ok.

Here's a brief explanation of these fields:

- **Database:** Choose the database you want to back up from the drop down list.
- **Recovery model:** This setting (configured when creating the database and unchangeable here) describes your options when faced with a server outage as well as what steps you can take during backup and restore.
- **Backup Type:** Your choices here are full, differential, and transaction log. As you might expect, the full database backup archives all the information found in the database. The differential backup archives only data that has changed since the last full backup. Finally, the transaction log backup archives only information that has been written into the transaction log by SQL Server.
- **Backup Component:** You can elect to back up the entire database, or only a subset of the database's supporting files. Unless disk space is at a premium or there are other unique requirements (such as certain database files being offline), it's a good idea to back up the entire database rather than only a subset of your files.
- **Name:** This is a name that helps identify the backup set. SQL Server will provide one for you if you don't specify any.
- **Description:** This is an optional administrator-generated summary of the backup set.
- **Backup Set Will Expire:** This optional field allows you to determine when the information that you archive will expire. You can choose a date or a number of days after this backup.

- **Destination:** If you have a tape drive installed, SQL Server offers you the option of backing up your information to that drive. Comparatively, many administrators simply back up their data to a disk.

3. If necessary, fill in additional information on the options page.

Navigate to this page by clicking Options on the left of this dialog box.

On the Options page, shown in the following figure, you can set additional criteria



- **Overwrite Media:** You can instruct SQL Server whether it should append, overwrite, or create a new set of media in support of your backup.

- **Reliability:** These two settings help provide additional assurance that your backup has been done correctly.

Although it takes more time, it's a good idea to verify the backup and perform a checksum prior to writing to media.

- **Transaction Log:** If you're backing up the transaction log, you can choose whether to truncate (that is, delete) backed up transactions from the log, or simply back up the very end of the transaction log.

- **Tape Drive:** If you've elected to back up your information to a tape drive, these two settings help control the physical device.

- **Compression:** Because backups can be quite large, SQL Server allows you to specify whether you want these archives to be compressed.

4. When you've finished setting your backup options, click OK.

9.3. Restoring a backup

Resorting means bringing the database back to the state prior to failure. If a virus, hardware failure, or simple user error has damaged your organization's vital information, it is the time where your backup files are critically important.

To begin the restore process, follow these steps:

1. Launch the SQL Server Management Studio

2. **Connect to the appropriate SQL Server instance**
3. **Expand the connection's entry in the Object Explorer view.**

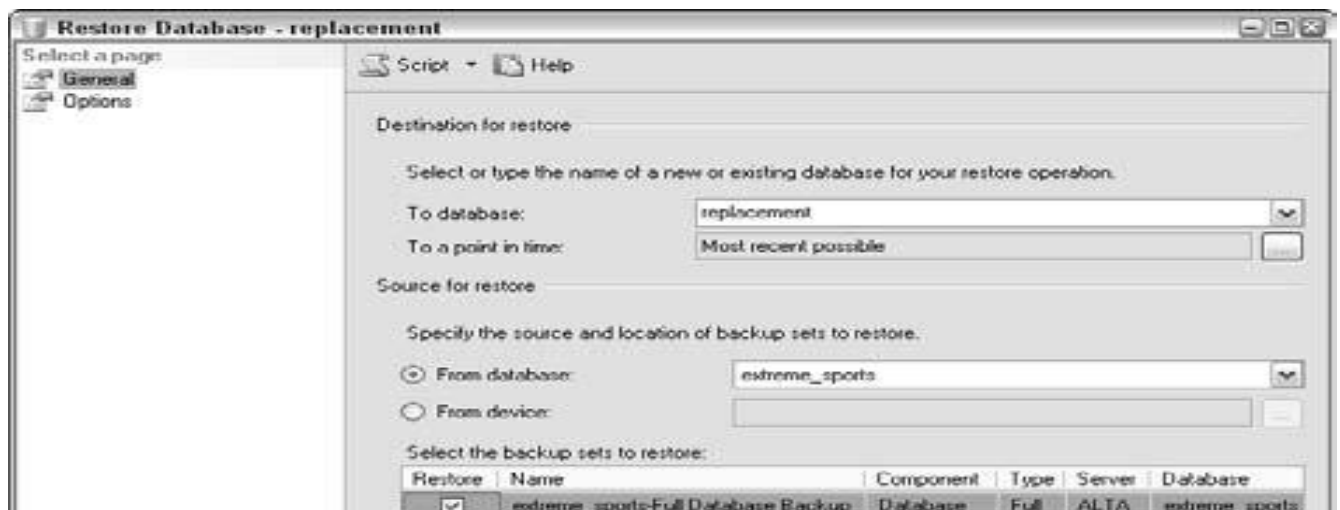
Assuming that you need to restore an entire database that is no longer present, right-click the Databases folder and choose either the Restore Database or Restore Files and File group's option. For the purposes of this example, imagine that you've lost the entire database and need to restore the full set of information

4. **Fill in values for all relevant fields.**

Two pages' worth of options are at your disposal. This section describes the General page; thereafter, the Options page is covered. The following fields require your attention:

- **Destination for Restore:** You can instruct SQL Server about which database should receive the restored information. If an original database has been damaged, a good strategy is to restore the archive into a brand-new database, rather than trying to overlay the already damaged database. You can also specify whether you want your restore to be as recent as possible, or to a specific date and time.
- **Source for Restore:** Your choices are to select an archive by database name, or from a backup device. In most cases, you should be able to use the From Database drop-down list to identify the database for recovery.

The following figure shows this page in more detail.



9.4. Exercise

1. Perform a database backup with options available in the backup utility. Then, demonstrate how it can be used to restore an already lost data.
2. Restore a database using backup under the following options and explain when to use them
 - a. Restore with No Recovery
 - b. Restore with Stand by
3. Perform a Transaction log backup

10. Laboratory Session10: Replication

Learning Objective

At the end of this lesson, students will be able to:

- Identify appropriate replication strategy.
- Implement replication.
- Troubleshoot replication problems.

Introduction

Replication are configured either to push data, pull data, and merge data across local area networks (LANs) and wide area networks (WANs).

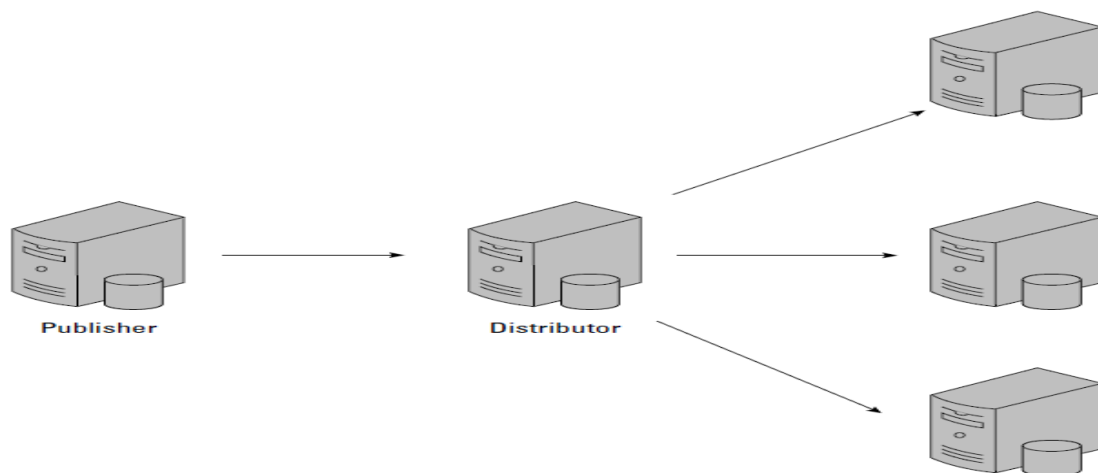
The goal of *replication* is to make copies of the data and ensure those copies are kept up to date. How often the data is updated depends on how often the data changes and how up to date the data needs to be. Some of the reasons to replicate data are to offload the reporting workload to another SQLServer, replicate the data to dedicated backup server so backups can occur without interfering with the online server, and keeping data centralized from several branch office databases.

Replication is based on a publishing metaphor. In the publishing world, there are publishers, distributors, and subscribers. Replication defines different roles based on the real-world example of publishing.

◆**Publishers:** Publishers maintain the source database that will be replicated.

◆**Distributors:** Distributors send the desired portions of the database to subscribers. Often, the publisher also fulfills the role as a distributor, but this isn't required. When a large number of subscribers exist, distributors take some of the load off the publisher.

◆**Subscribers:** Subscribers receive the replicated data in the form of a publication.



This works similarly in the real world. *Newsweek* magazine, which is published internationally, has one central publisher that creates all the content. After a magazine is published, it's handed over to distributors who send the copies to subscribers. Each magazine has articles within it based on the publication. The publication you subscribe to determines which articles you receive.

Subscriptions can either be push subscriptions or pull subscriptions:

◆ **Push Subscription:** A *push subscription* is initiated by the distributor.

Push subscriptions are sent either continuously as changes occur or on a preset schedule.

◆ **Pull Subscription:** A *pull subscription* is initiated by the subscriber.

Defining a Replication Publishing Model

With replication, four different models exist. These models define whether the entire database is replicated (or only the changes), how often the replication occurs, and who can make changes to the source database.

◆ **Snapshot replication:** Snapshot replication involves making a copy of the publication at a moment in time. The entire publication is then replicated to the receiving database. Snapshot replication is the easiest to implement, but takes the most bandwidth.

◆ **Transactional replication:** Transactional replication is used to constantly update and publish the articles. This is used when the subscribers need access to changes as they occur. All changes to the database are recorded in the transaction log. The log reader agent then reads the transaction log to replicate the changes to the receiving database. These changes are then applied to the receiving database.

◆ **Peer-to-peer transactional replication:** Peer-to-peer replication is used for applications that might read or modify data at any of multiple databases participating in replication. It's built on the foundation of transactional

Replication, but peers in transactional replication can both read and modify changes; subscribers in transactional replication can only receive the changes.

◆ **Merge replication:** Merge replication is used when multiple locations need to be able to update the data. For example, a retail chain might have multiple stores that all need to submit sales and inventory data to the headquarters location. If they don't have a product in stock, they check other locations for customers, so they need access to inventory data at other locations. Merge replication allows each of the retail stores to be publishers. In other words, instead of changes made at only one central location, changes can be made in databases at multiple locations and then merged.

10.1.Replication Architecture

SQL Server 2012 replication uses specific terminology to describe how components interact in the replication topology.

The important terms are as follows:

Publisher: An instance that makes data available through publication.

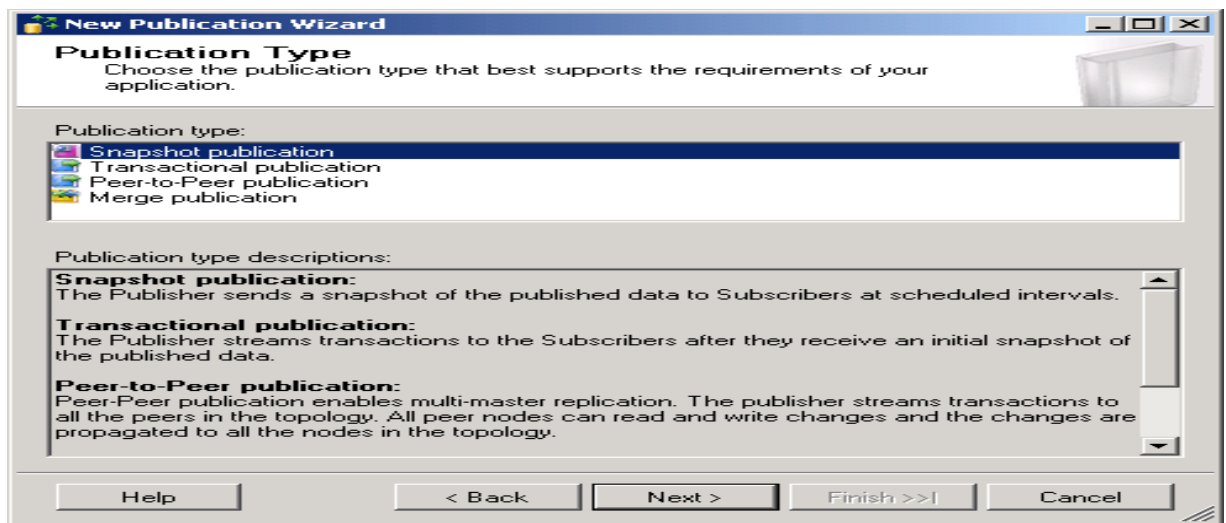
Article: A published object which can be a table, a stored procedure, a view, an indexed view, or a user-defined function.

Publication: A collection of articles.

Distributor: An instance that manages the transmission from publisher to subscriber. A distributor on the same LAN as the publisher is a local distributor. A distributor on network remote from the publisher is a remote distributor.

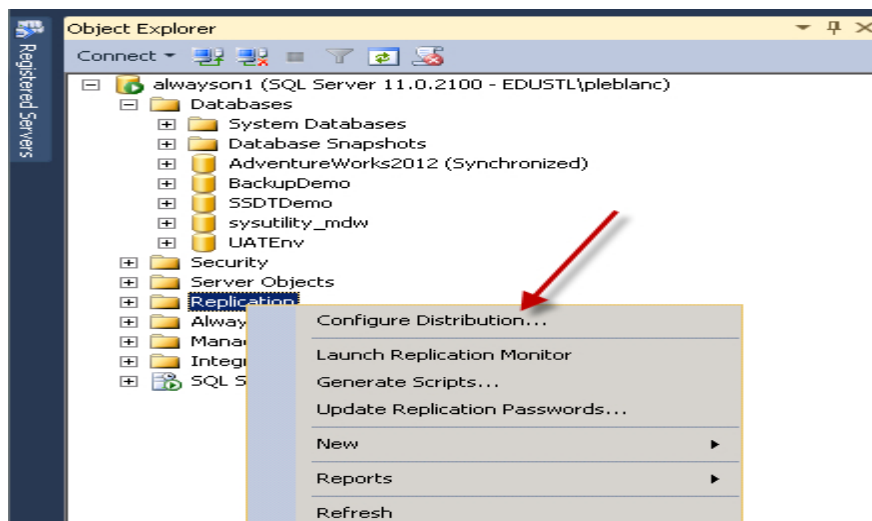
Subscriber: An instance that receives publications.

Agent: A service that enables the publisher, distributor, or subscriber to perform



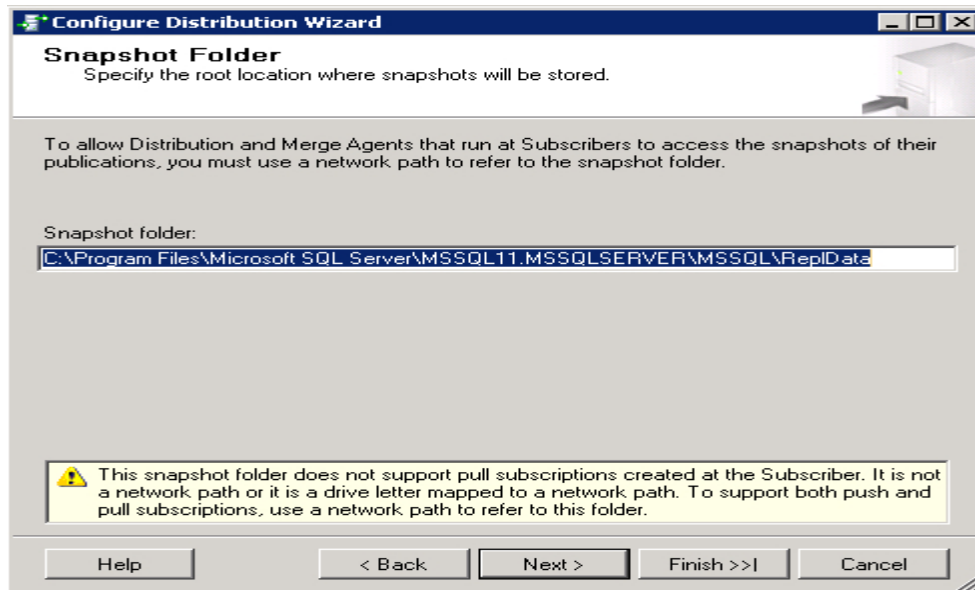
10.2. Configuring the Distributor

1. Open SQL Server Management Studio (SSMS) and connect to an instance of SQL Server.
2. In Object Explorer, expand the Databases folder.
3. Right-click the Replication folder.
4. Select Configure Distribution from the menu.



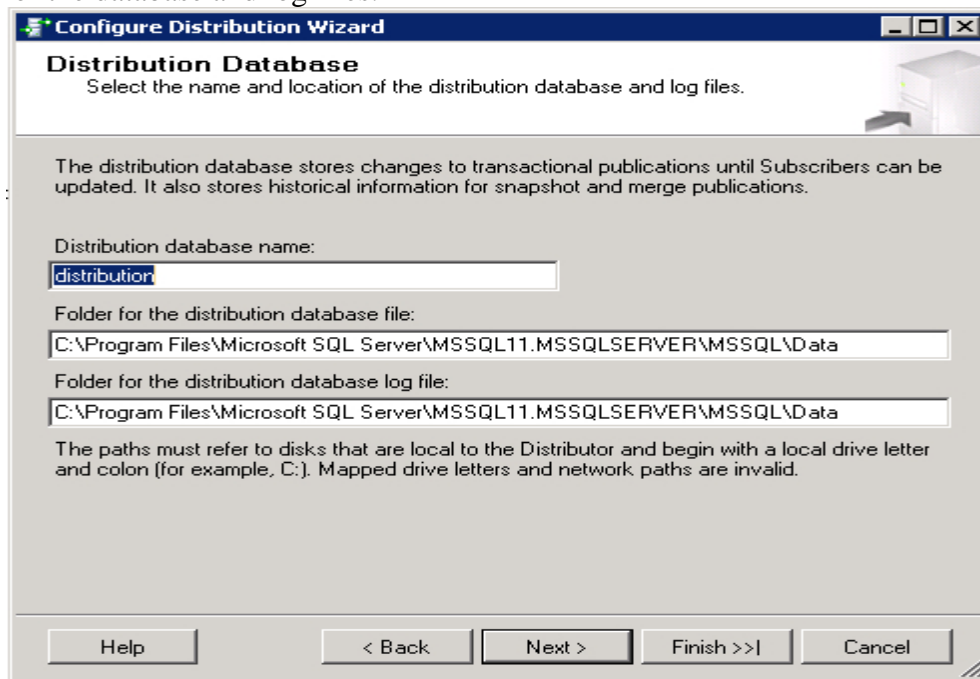
5. Click Next.
6. On the Distributor page, accept the defaults and click next.

On the next page, Snapshot Folder, you will see a warning in reference to the location of the snapshot folder. If you are configuring a remote Distributor, you must specify a UNC path: \\Computer Name\folder path.



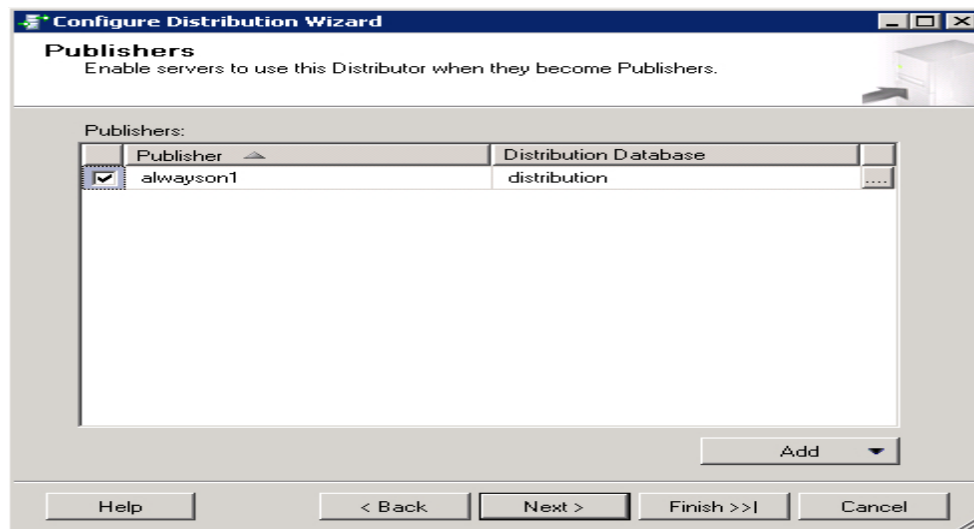
7. Click Next

8. On the Distribution Database page, change the name of the database and specify a new location for the database and log files.



9. Accept the defaults and click next.

10. On the Publishers page, you can enable other servers to use this server as a Distributor. By default, the current server is automatically included.

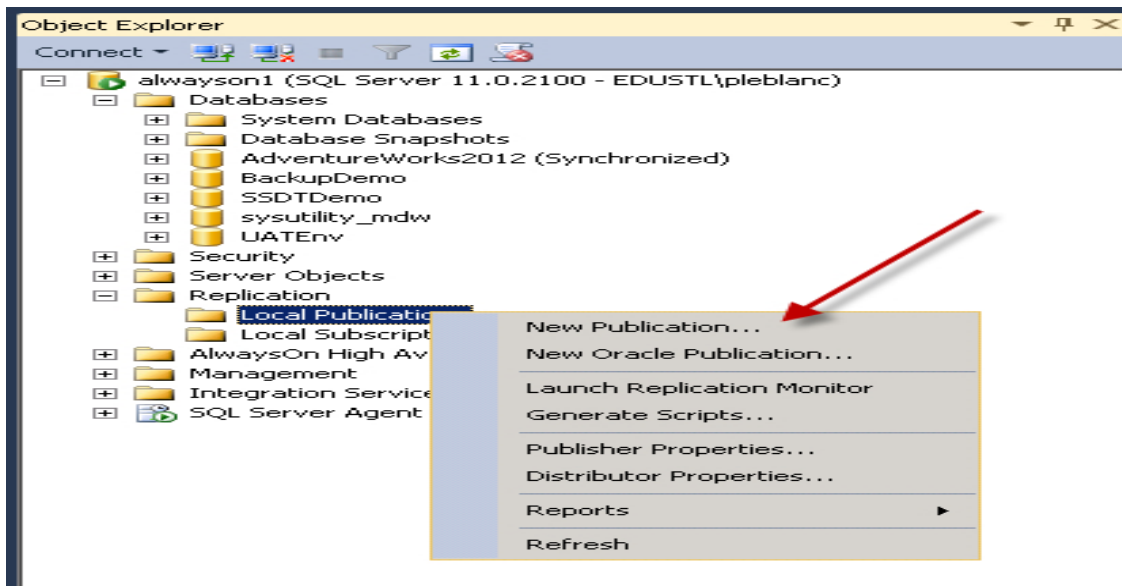


11. Accept the defaults and click next.
12. On the final page, Wizard Actions, in addition to configuring distribution, you can generate a script that contains all the steps from the wizard. For now, accept the defaults and click next.
13. On the Complete the Wizard page, review the summary and click Finish.
14. Configuration begins and the progress is shown on the Configuring page. Click Close.

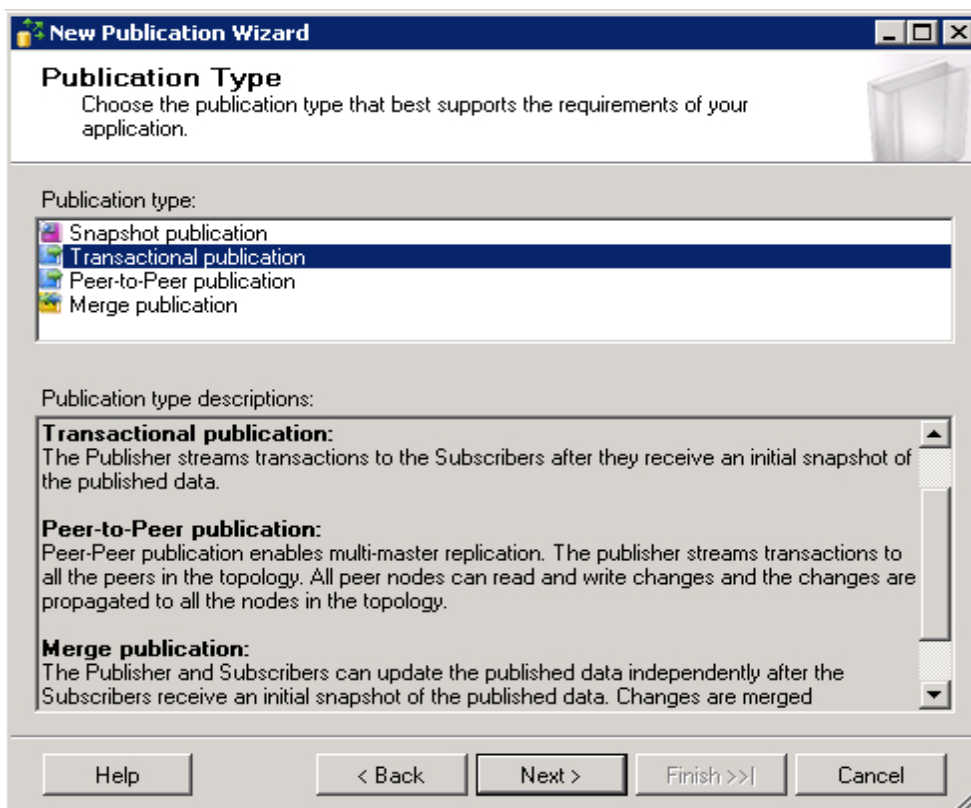
10.3. Configuring a transactional Publisher

Open SSMS and connect to an instance of SQL Server.

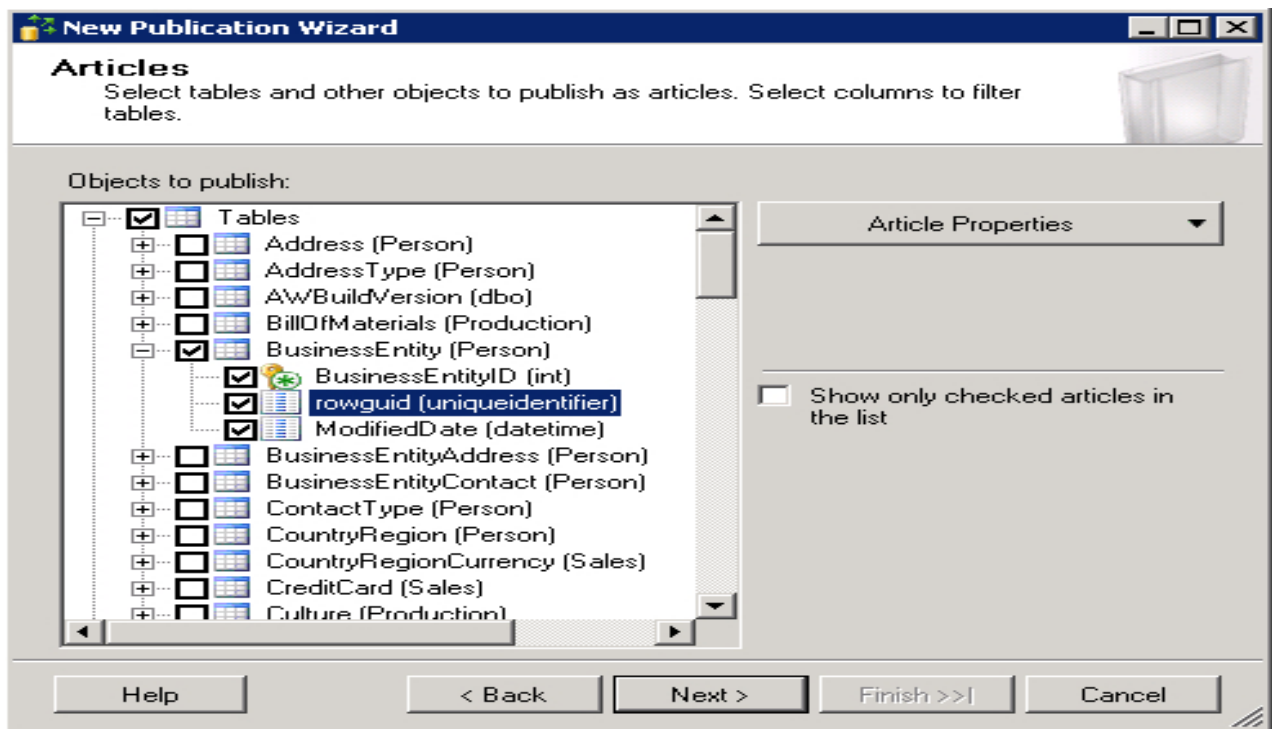
1. Expand the Replication folder.
2. Right-click the Local Publications folder and select New Publication from the menu.
3. Right-click the Local Publications folder and select New Publication from the menu.



4. Click next on the New Publication Wizard page.
5. Select the database on the Publication Database page and click next.
6. On the Publication Type page, you are presented with four types. The type you select depends on your requirements. Select Transactional Publication from the list and click next.

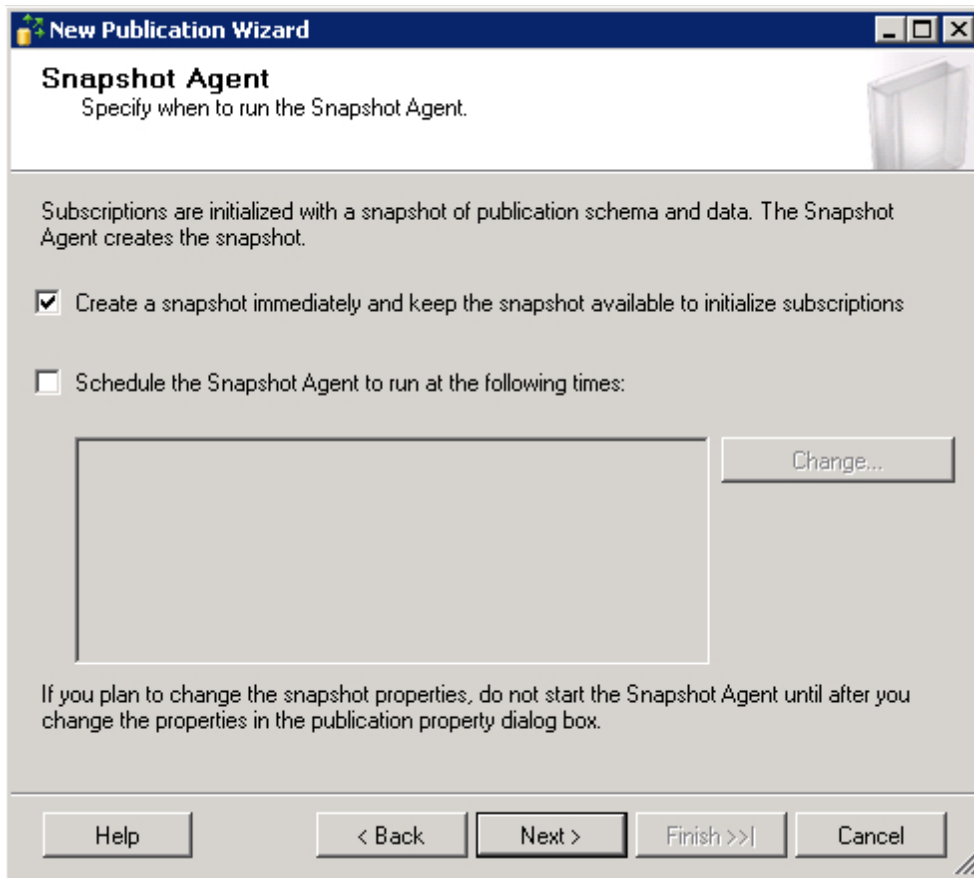


7. The Articles page is where you specify the database objects that you would like to replicate. As shown on the page, you can replicate several objects. You will focus primarily on tables. Each replicated object is referred to as an article. Expand Tables, and then expand the Business Entity table. You can holistically replicate the entire table or select only the columns that you want to replicate.
8. Select the box next to the table.



9. To the right of the list of articles is an Article Properties button. Click the drop-down arrow located on the button and select Set Properties of Highlighted Table Article. Using the Article Properties dialog box, you specify whether or not you want to copy constraints, triggers, clustered indexes, and so on. Scroll through the list and familiarize yourself with what is available.
10. Accept all the defaults for now and click OK. On the Articles page, click next.
In some cases, you may want to replicate only certain rows from a table. By using the filter option, you can limit the rows with a WHERE clause.
11. Suppose that you are going to replicate all the rows in the specified table, so just click next.

As discussed earlier, prior to configuring all types of replication, a snapshot must be taken. On the Snapshot Agent page, you can specify when to create a snapshot and how often. Select the box next to the option Create a Snapshot Immediately and Keep the Snapshot Available to Initialize Subscriptions.



12. Click Next.
13. On the Agent Security page, you must specify under which account the Snapshot Agent and the Log Reader Agent will run. The Windows account that is used for the agents must be a member of the db_owner database role in the distribution database and in the publication database. Click the Security Settings button next to the Snapshot Agent text box.
14. In the Snapshot Agent Security dialog box, select the Run under the SQL Server Agent Service Account option. Note that the dialog box states that selecting this option is not recommended as a best practice, but for testing purposes, it will suffice. However, when going to production you should create a new Active Directory account for sole use by the Snapshot Agent.
15. In the lower section of the same dialog box, select the By Impersonating the Process Account option.

Snapshot Agent Security

Specify the domain or machine account under which the Snapshot Agent process will run.

☐ Run under the following \Windows account:

Process account:

Example: domain\account

Password:

Confirm Password:

☒ Run under the SQL Server Agent service account (This is not a recommended security best practice.)

Connect to the Publisher

☒ By impersonating the process account

☐ Using the following SQL Server login:

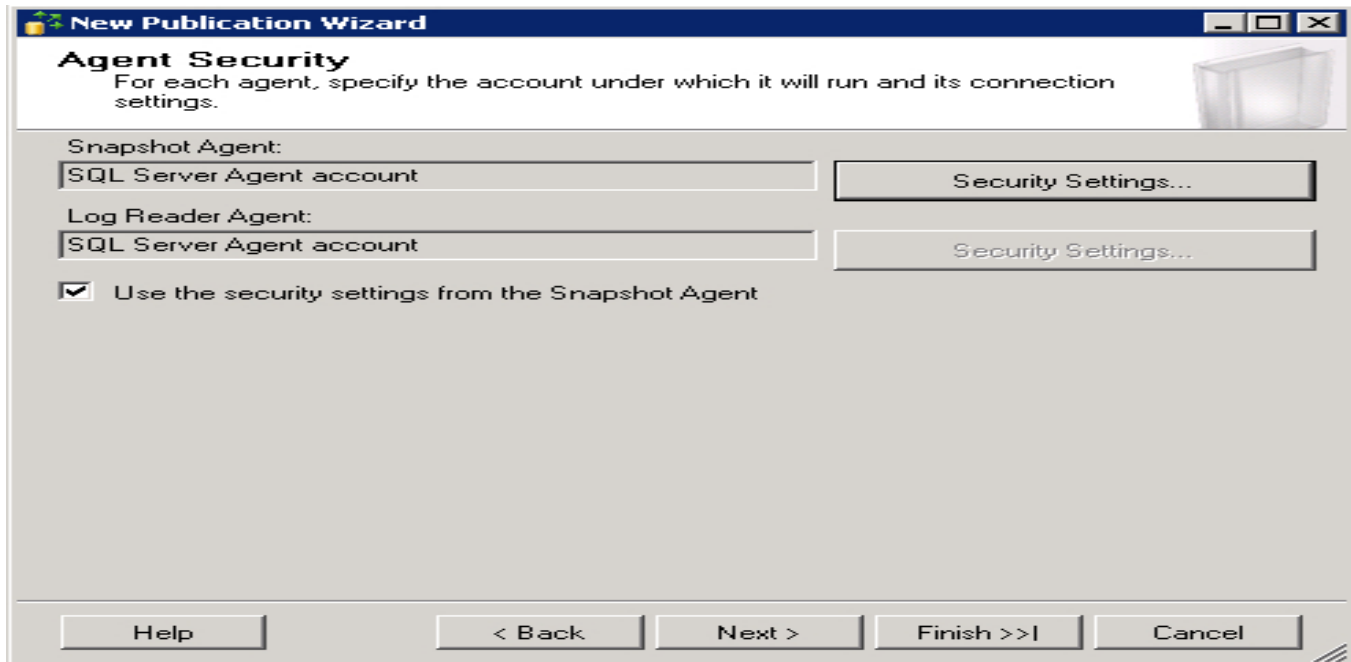
Login:

Password:

Confirm Password:

OK Cancel Help

16. Click OK. Remember, these settings are not a best practice since the service account running the agent may have permissions beyond what is required by replication.
17. Back on the Agent Security page, you will see that both agents have been configured. This is because by default the wizard uses the security settings from the Snapshot Agent for the Log Reader Agent. You can override this default by clearing the Use the Security Settings from the Snapshot Agent check box and manually configuring the settings.



18. For now, accept the defaults and click Next
19. On the Wizards Actions page, ensure that the Create the Publication check box is selected. If you want to generate a script, you can select the Generate a Script file with steps to create the publication also. For now, leave it cleared.
20. On the Complete the Wizard page, enter Business Entity in the Publication Name text box, review the summary, and click Finish.

Configuration begins and the progress is shown on the Creating Publication page.

21. Click Close.

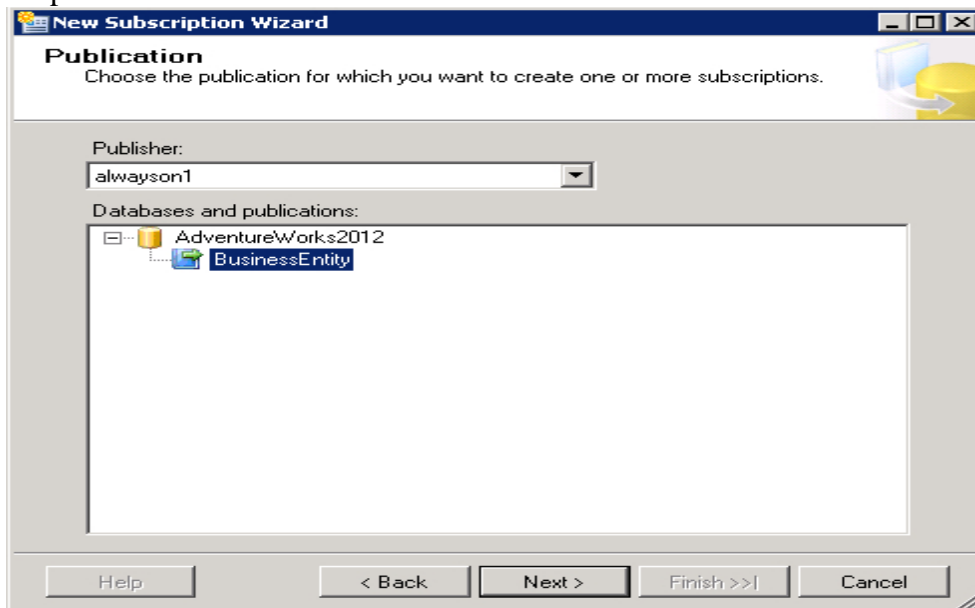
You will see the newly created publication in Object Explorer under Replication | Local Publications. In addition, a Snapshot Agent job and a Log Reader Agent job have been created on your SQL Server instance; you can view these in Object Explorer under SQL Server Agent jobs.

Configure a transactional Subscriber

1. Open SSMS and connect to an instance of SQL Server.
2. Open a new query window and enter the following T-SQL code:

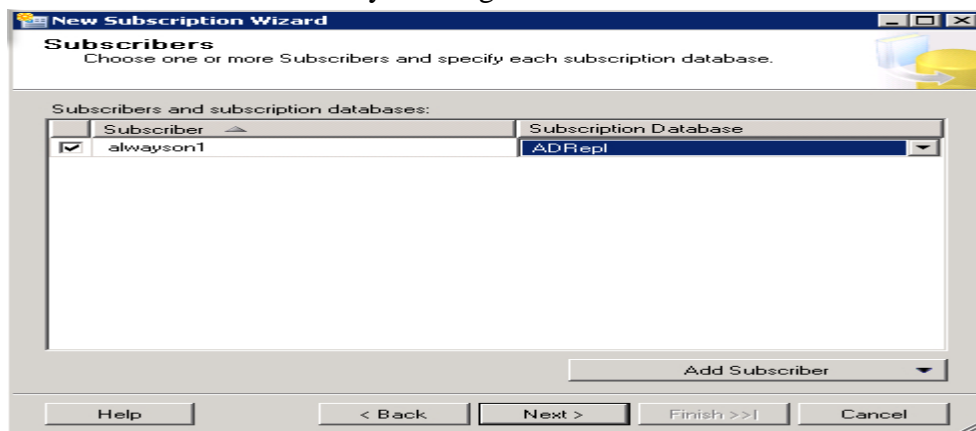
```
USE master; CREATE DATABASE ADRepl;
```
3. Execute the query.
4. Expand the Replication folder in Object Explorer.
5. Right-click the Local Subscriptions folder and select New Subscriber from the menu.

6. Click Next.
7. On the Publication page of the New Subscription Wizard, ensure that the BusinessEntity publication is selected and click next.



8. Now you are presented with the choice of running the Distribution Agent on the Distributor for push replication or on the Subscriber for pull replication. While running the Distribution Agent on the Subscriber does offload some of the work from the Publisher, if it is on the subscriber, the subscriber will bear the load. When making this choice, consider how each will be affected, and ensure that the selected choice has sufficient resources to support your decision. In general, place the Distributor on a more powerful machine or the machine with lowest network connection latency. For now, accept the defaults and click next.

9. On the Subscribers page, select the box to the available choice, which will be the local machine, and select ADRepl from the Subscription Database column. You can optionally add a new subscriber by clicking the Add Subscriber button.



10. Click Next.

11. In the Distribution Agent Security dialog box, click the ellipsis button. Similar to the Snapshot Agent and Log Reader Agent, select to run the agent under the SQL Server Agent services account and connect to the Distributor and Subscriber using the process account. Again, as a best practice, you should use a new Active Directory account that has been created specifically for this purpose.

Distribution Agent Security

Specify the domain or machine account under which the Distribution Agent process will run when synchronizing this subscription.

☒ Run under the following Windows account:

Process account: Example: domain\account

Password:

Confirm Password:

☐ Run under the SQL Server Agent service account (This is not a recommended security best practice.)

Connect to the Distributor

☒ By impersonating the process account

☐ Using a SQL Server login

The connection to the server on which the agent runs must impersonate the process account. The process account must be a member of the Publication Access List.

Connect to the Subscriber

☒ By impersonating the process account

☐ Using the following SQL Server login:

Login:

Password:

Confirm password:

The login used to connect to the Subscriber must be a database owner of the subscription database.

OK Cancel Help

12. Click OK and click next.
 13. On the next page, Synchronization Schedule, select Run Continuously in the column labeled Agent Schedule. You can also synchronize on demand or define a schedule. Click next.
 14. On the Initialize Subscriptions page, you have two choices: Initialize Immediately or At First Synchronization. Accept the defaults and click next.
 15. On the final page, Wizard Actions, in addition to configuring distribution, you can generate a script that contains all the steps from the wizard. For now, accept the default and click next.
 16. Review the summary and click Finish.
- Configuration begins and the progress is shown on the Creating Subscriptions page.
17. Click Close.
 18. Right-click the Replication folder in Object Explorer and select Refresh.
 19. Expand the Local Publications folder.
 20. Expand the Business Entity publication, and you will see the new subscription.

21. Open a new query window and enter the following T-SQL code:

```
USE ADRepl;SELECT * FROM Person.BusinessEntity
```
22. Execute the query.
23. You will now see that the table is created and the data has been copied.

10.4. Exercise

Take a hypothetical assumption that there are fifty branches of ABC Company where each branches are connected with a network.

Suppose that the company want to replicate data from Addis Ababa server to Bahir Dar server using transactional replication. Use the following replication architecture into consideration and configure replication.

Publication: Employee records containing Bahir Dar employees

- a. Publisher, distribution agent, log reader agent, snapshot agent, and all publications are placed at the Addis Ababa server
- b. Push subscription option is used to distribute data from publisher to subscriber
- c. Any change in the schema of the employee table should not be replicated to Bahir Dar server

Demonstrate how the replication can be configured with this scenario.

11. Laboratory Session 11: Database Mirroring

Learning Objective:

At the end of this lesson, students will be able to:

- Prepare a database for mirroring
- Mirror a database
- Perform failover from principal to mirrored instance

Introduction or What a database mirroring is?

Database mirroring is the process of creating and maintaining an always up-to-date copy of a database on another SQL Server instance. Transactions applied to the database on the principal instance are also applied to the database on the mirrored instance. It is a strategy for increasing the availability of a database. Database mirrors are paired copies of a single database that are hosted on separate instances of SQL Server. With database mirroring, all the changes in a protected database called the principal database are sent to a backup database called the mirrored database. If the principal database fails, the mirrored database located on a different SQL Server instance will be almost-instantly available.

Once mirroring is configured, every insert, update, and delete operation performed against the principal database is redone on the mirror database through a stream of active transaction logs that the Database Engine instance applies in sequence immediately after it receives them.

Depending on how you configure mirroring, the mirror instance functions in either high safety or high-performance mode. You can mirror some or all the user databases on an instance. However, you can mirror only databases that are configured to use the full recovery model.

11.1 Mirroring Prerequisites

Before you create a new mirroring session, ensure that your environment will support mirroring. The primary, mirror, and witness instances must run the same version of SQL Server. The primary and the mirror instances must run the same edition, whereas the witness instance must run only an edition of SQL Server that supports witnessing. Mirroring support on a per edition basis in SQL Server 2012 is as follows:

Enterprise: Supports high-performance, high-safety, and witness modes

Business Intelligence: Supports high-safety and witness modes

Standard: Supports high-safety and witness modes

Web: Supports witness mode only

Express: (all versions) Supports witness mode only

Database mirroring on SQL Server 2012 has the following restrictions:

- You can mirror only user databases. It is not possible to mirror the master, msdb, tempdb, or model databases.
- You cannot rename mirrored databases.
- Cross-database and distributed transactions are not supported for mirrored databases.

Full Recovery Model

You must configure a database to use the full recovery model before you can mirror it. You can view the recovery model of a database on the Options page of the Database Properties dialog box, as shown in the next figure.

You can also configure the recovery model by using the ALTER DATABASE statement with the SET RECOVERY option. For example, to configure the EXEMPLAR database to use the FULL recovery model, execute the following Transact-SQL statement:

```
USE MASTER;
```

```
ALTER DATABASE Exemplar SET RECOVERY FULL;
```

Prior to commencing a mirroring session, you must restore a database by using the NORECOVERY option on the mirror instance. To do this, you need a full backup of the database that you mirror.

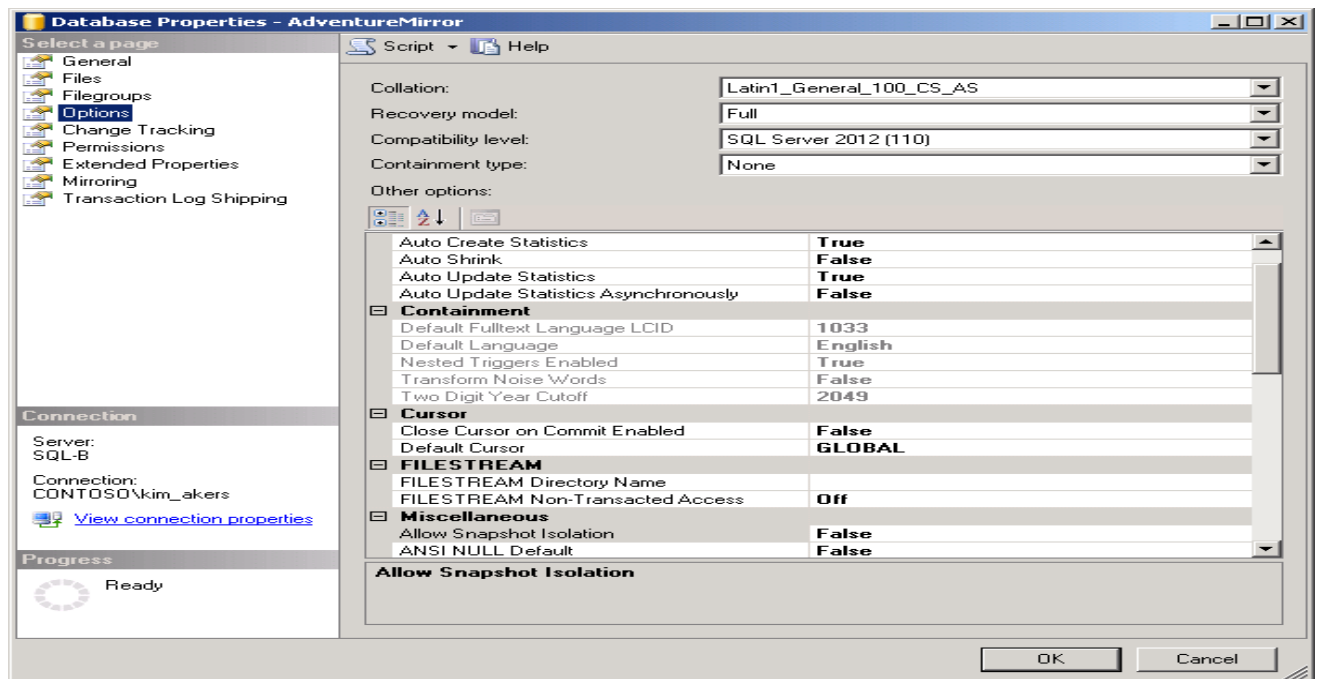


Figure 11.1: Database properties showing recovery model

You must then restore the database backup on the mirror instance by using the NORECOVERY option. You can configure this option on the Options page of the Restore Database dialog box. You can also do this by using the RESTORE DATABASE Transact-SQL statement with the WITH NORECOVERY option.

After you have performed these steps, go back to the principal database and perform a transaction log backup. You can do this from SQL Server Management Studio or using BACKUP LOG statement. For example, to perform a transaction log backup of the RegistrarDBMirror database to the c:\backup\ RegistrarDBMirror.trn location, execute this statement:

```
BACKUP LOG RegistrarDBMirror
TO DISK = 'C:\backup\ RegistrarDBMirror.trn'
GO
```

You then must apply this transaction log backup to the database that you restored by using the WITH NORECOVERY option on the mirroring instance.

11.2. Endpoint Firewall Rules

When configuring a mirroring session, you must configure firewall rules to allow traffic through on the endpoint. For example, to support the mirroring configuration shown in 11.1, it is necessary to allow traffic on TCP port 7024.

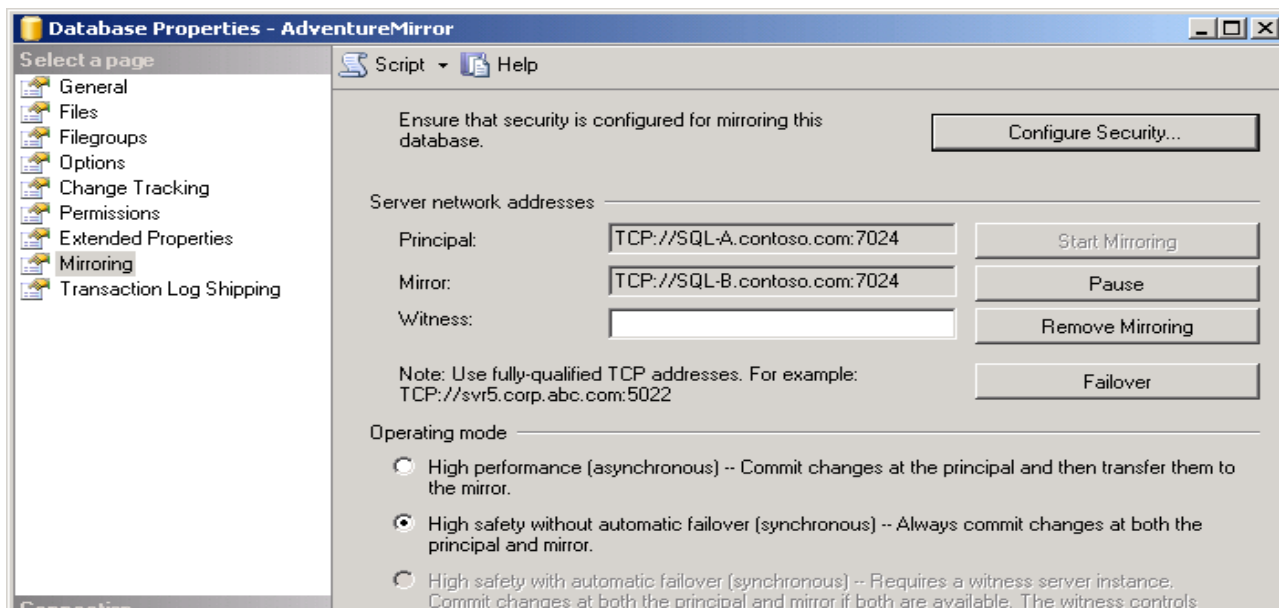


FIGURE 11.2 TCP Port 7024 used for endpoint

How to Configure Mirroring with Windows Authentication

You can configure mirroring with Windows-based authentication if the SQL Server service accounts on the principal, mirror, and witness are members of the same Active Directory domain or trusted domains. You can configure mirroring with Windows-based authentication by using SQL Server Management Studio or Transact-SQL.

To configure mirroring using high-safety mode with a witness server, using Windows authentication, perform the following steps:

1. Ensure that you have performed the prerequisite steps, including restoring a backup of the database and transaction log on the mirror instance by using the **WITH NORECOVERY** option.
2. Ensure that necessary logins have been created if different domain accounts are used for the SQL Server service on each instance that will participate in the mirroring. You must also perform this step on the witness server.
3. In SQL Server Management Studio, connect to the primary instance, right-click the database that you want to mirror, choose **Tasks**, and then choose **Mirror**.
4. On the **Mirroring** page of **Database Properties**, click **Configure Security** to open the **Configure Database Mirroring Security Wizard**. Click **Next**.
5. On the **Include Witness Server** page, choose **Yes** and click **Next**.
6. On the **Choose Servers To Configure** page, choose whether to save security configuration information on the Witness server instance. Security configuration information is saved by default on the Principal and Mirror server instances.
7. On the **Principal Server Instance** page, shown in Figure 11.3, either accept the default listener port or configure an alternate port. You can also accept the default endpoint name or configure an alternate endpoint name.

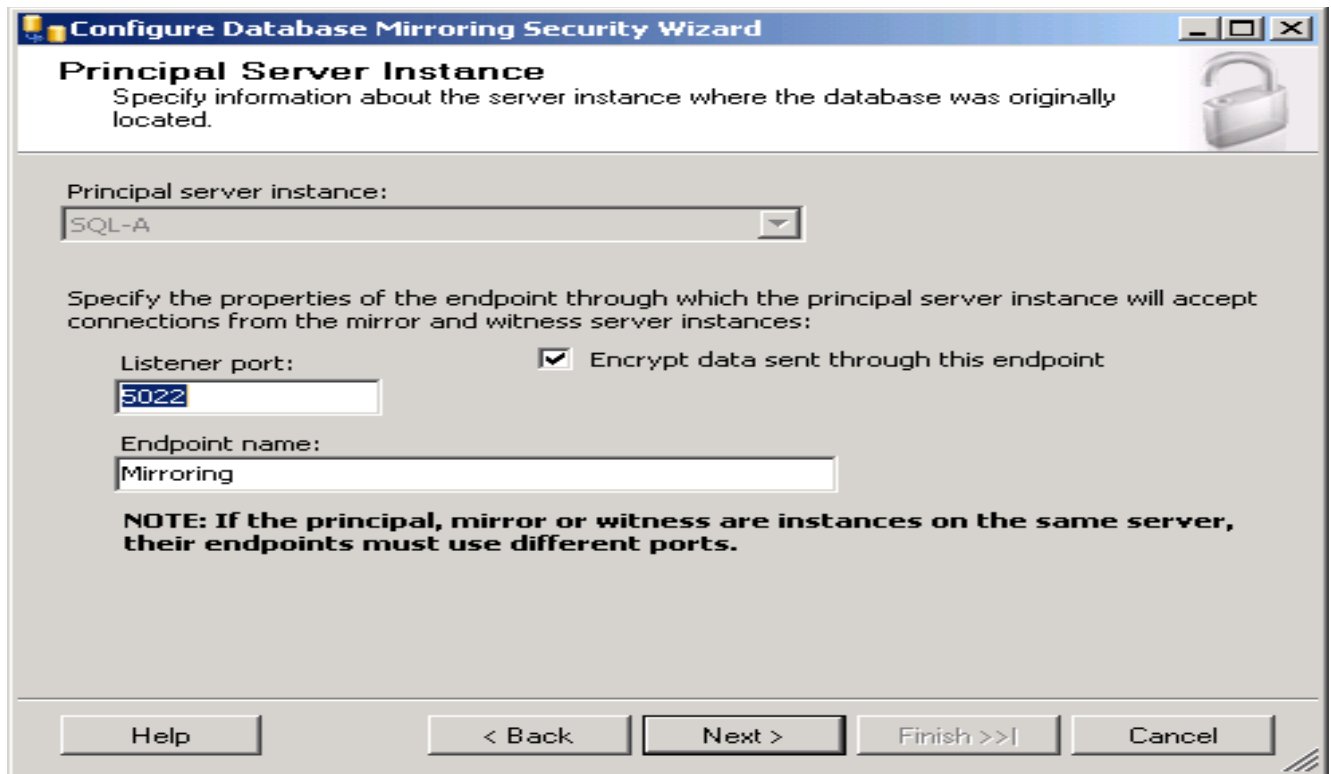


Figure 11.3 Principal server instance

8. On the Mirror Server Instance page, click Connect to open the Connect To Server dialog box. You must make this connection to the mirror instance by using a login or account that has the CREATE ENDPOINT permission or membership in the sysadmin fixed server role on the mirror instance. Note and, if necessary, alter the assigned listener port and endpoint name and then click next.
9. On the Witness Server Instance page, shown in Figure 11.4, choose a third instance to function as the witness server. Click Connect to open the Connect To Server dialog box. As with connecting to the mirror server, you must connect with an account that has the CREATE ENDPOINT permission or is a member of the sysadmin fixed server role on the witness instance.

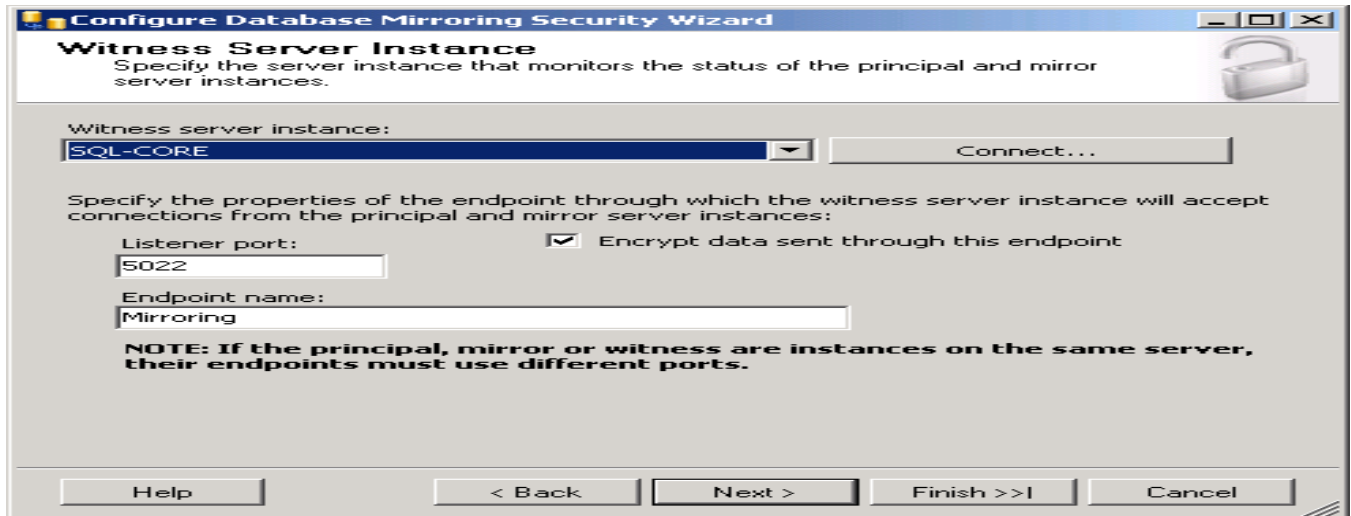


Figure 11.4 Witness Setup

On the Service Accounts page, shown in Figure 11.5, enter the domain service accounts that are used for the SQL Server service on each instance. Click Next and then click Finish to complete the wizard.

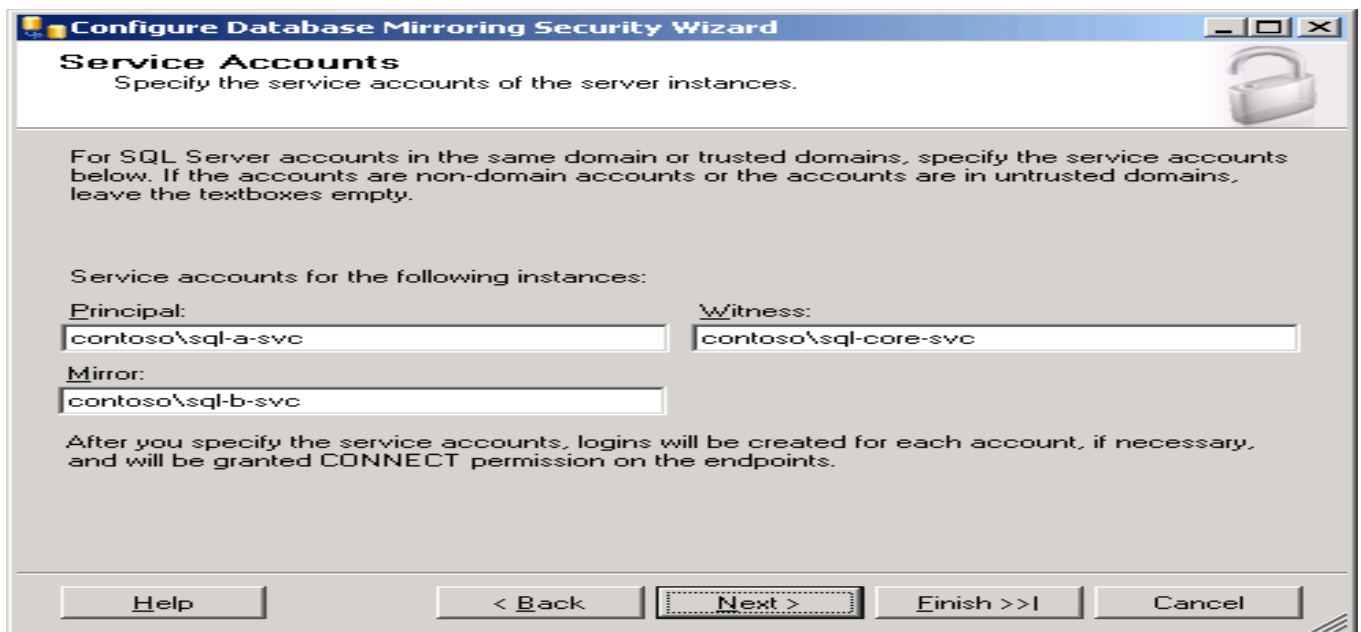


Figure 11.5 mirroring service accounts

11.3. Database Mirroring Monitor

The Database Mirroring Monitor, shown in Figure 11.6, enables you to monitor how data is being transmitted between the primary and mirrored instances in a mirroring session. You must register a database with the Database Mirroring Monitor before you can monitor that database.

The Database Mirroring Monitor provides among other things, the status of the connection to the witness if one is present, the witness address, and the operating mode.

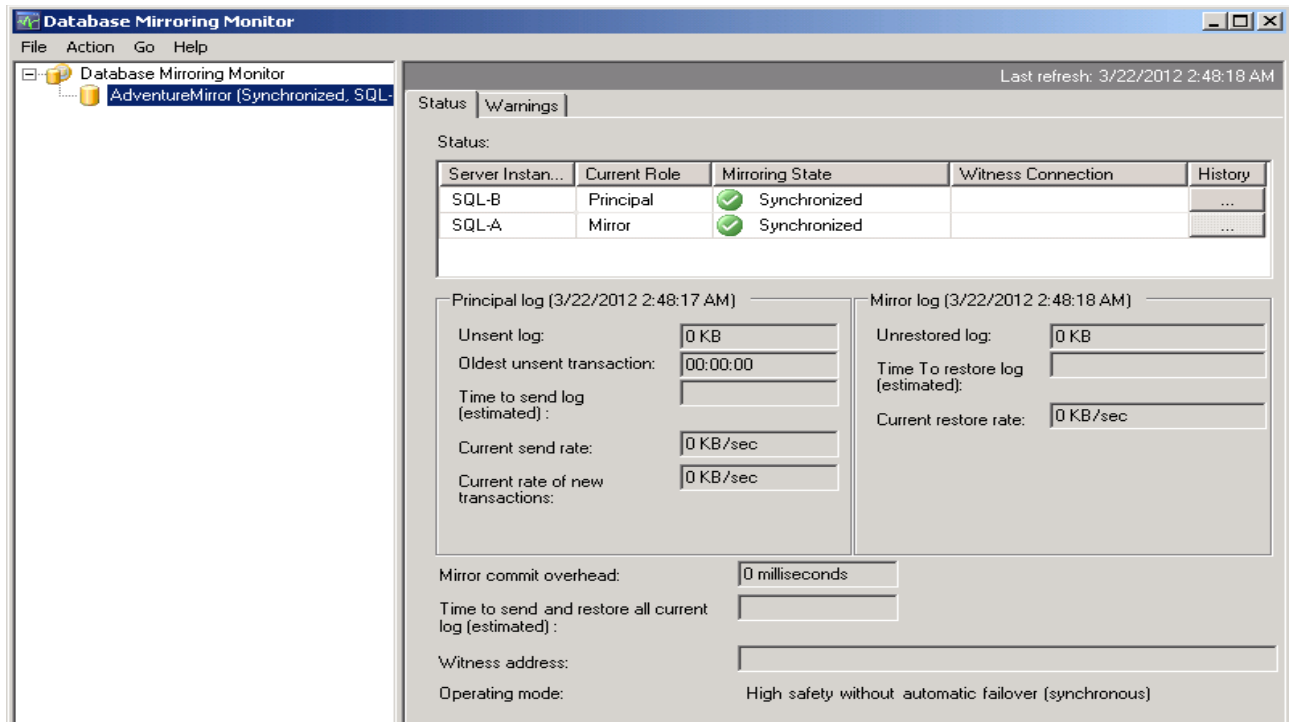


Figure 11.6 Database Mirroring Monitor

Exercise

1. Take Three SQL Server instances hosted on different computers named Serv_A, Serv_B and ServC
 - Serv_A primary server
 - Serv_B Mirror server
 - Serv_C Witness
2. Prepare a Database for mirroring
 - with appropriate backup and restore from primary to mirror databases
 - Configure Firewall Rules, for mirroring
 - Configure Logins, Users, Endpoint Permissions, and configure Database Mirroring
3. Configure Database Mirroring and test failover under the following options
 - High safety mode with automatic failover
 - High safety mode with manual failover
 - High performance mode

12. Laboratory Session 12: Log shipping

Learning Objective:

At the end of this lesson, students will be able to:

- Configure log shipping
- Provides a disaster-recovery solution

Log shipping is a technology that provides high availability at the database level, which is ideal for very low-latency networks. The transaction log for a specific database is sent to a secondary server from the primary server and restored. Just as with database mirroring, you can configure log shipping in a way that allows the secondary database to be read.

Log shipping is a high-availability and disaster-recovery solution that can be used to protect against unplanned downtime. Like database mirroring, log shipping works by first restoring a full backup of the primary database to a backup server. Then transaction log backups are periodically sent from the primary server's database and applied to the database on the backup server.

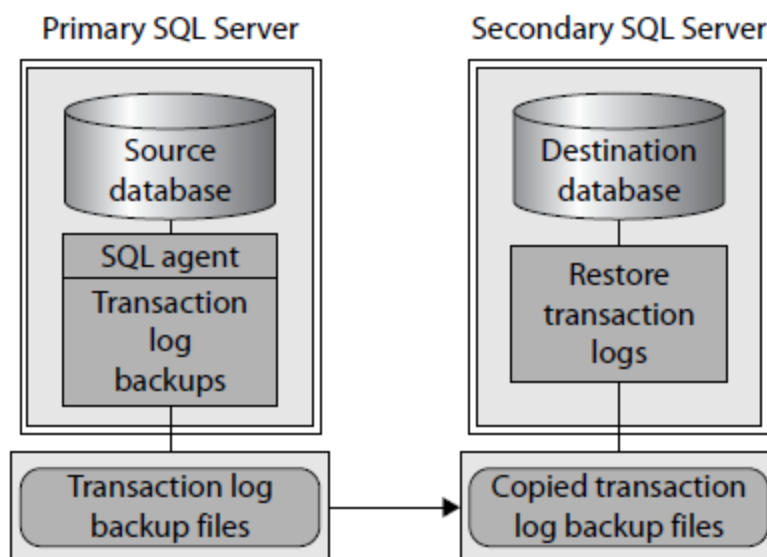


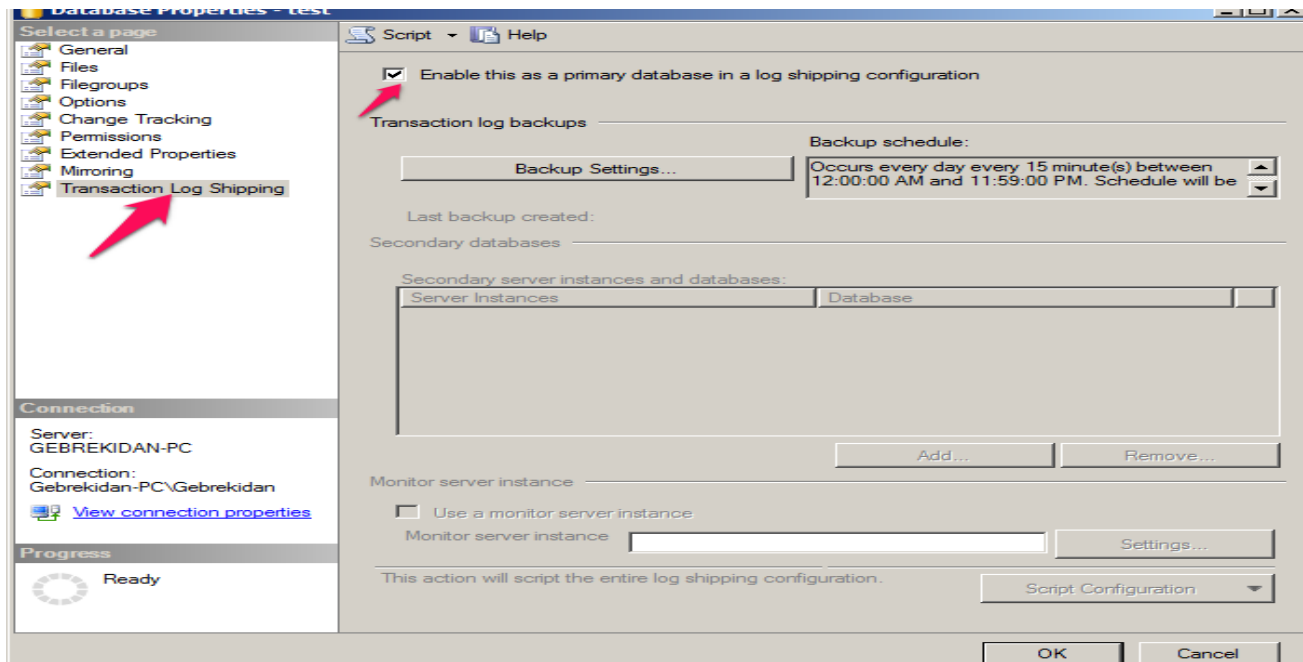
Figure 12.1 Log shipping

While database mirroring forwards log data to a mirror server as it occurs, log shipping sends transaction logs back, allowing you to choose the time when the transaction log will be captured and sent.

12.1. Steps to configure log shipping

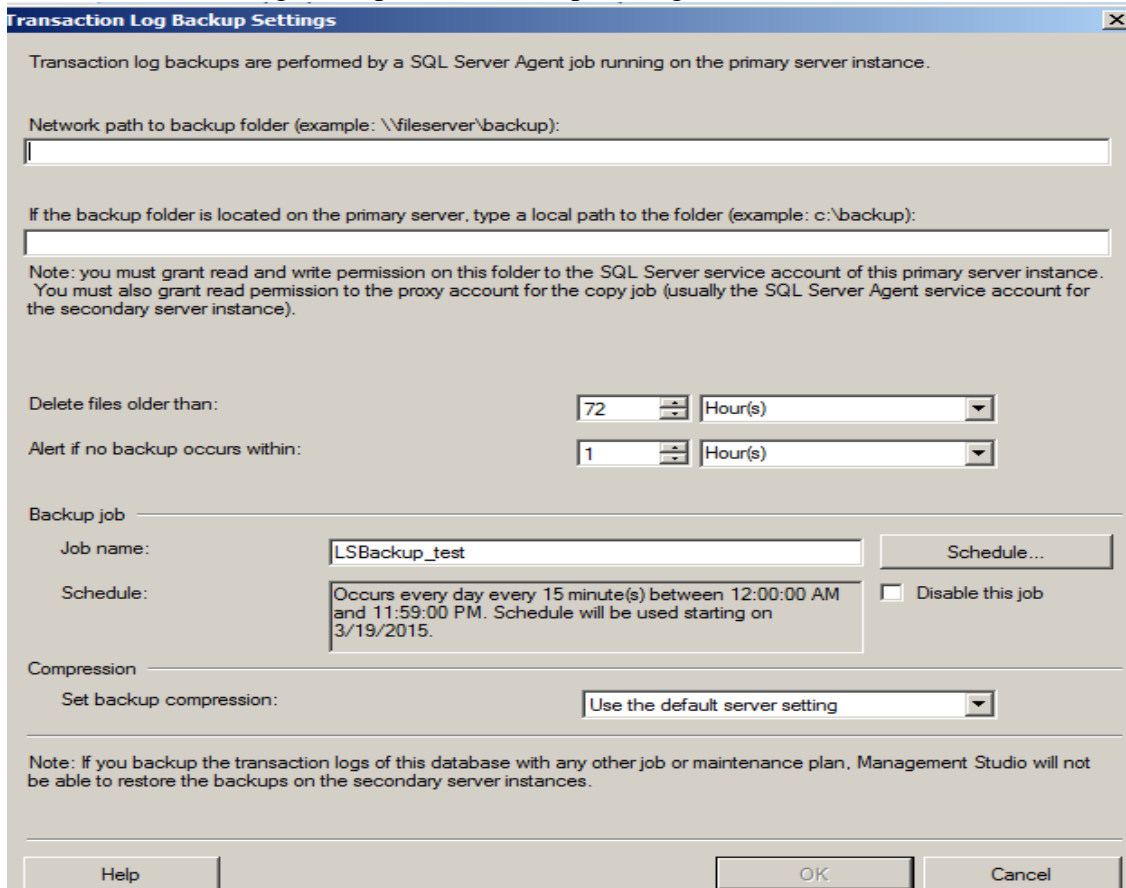
A. Using SQL Server Management Studio

1. Right click the database you want to use as your primary database in the log shipping configuration, and then click Properties.
2. Under Select a page, click Transaction Log Shipping.



3. Select the Enable this as a primary database in a log shipping configuration check box.

- Under Transaction log backups, click Backup Settings.



The image shows the 'Transaction Log Backup Settings' dialog box. It has a title bar with the text 'Transaction Log Backup Settings' and a close button. The main area contains several sections: 1. A text box for 'Network path to backup folder (example: \\fileserver\backup):'. 2. A text box for 'If the backup folder is located on the primary server, type a local path to the folder (example: c:\backup):'. 3. A note: 'Note: you must grant read and write permission on this folder to the SQL Server service account of this primary server instance. You must also grant read permission to the proxy account for the copy job (usually the SQL Server Agent service account for the secondary server instance)'. 4. Two rows of controls: 'Delete files older than:' with a spinner set to 72 and a dropdown for 'Hour(s)'; and 'Alert if no backup occurs within:' with a spinner set to 1 and a dropdown for 'Hour(s)'. 5. A 'Backup job' section with a 'Job name:' text box containing 'LSBackup_test', a 'Schedule:' text box containing 'Occurs every day every 15 minute(s) between 12:00:00 AM and 11:59:00 PM. Schedule will be used starting on 3/19/2015.', a 'Schedule...' button, and a checkbox for 'Disable this job'. 6. A 'Compression' section with a 'Set backup compression:' text box and a dropdown menu set to 'Use the default server setting'. 7. A note at the bottom: 'Note: If you backup the transaction logs of this database with any other job or maintenance plan, Management Studio will not be able to restore the backups on the secondary server instances.' 8. At the bottom are three buttons: 'Help', 'OK', and 'Cancel'.

- In the Network path to the backup folder box, type the network path to the share you created for the transaction log backup folder.
- If the backup folder is located on the primary server, type a local path to the folder box. (If the backup folder is not on the primary server, you can leave this box empty.)

Note

If the SQL Server service account on your primary server runs under the local system account, you must create your backup folder on the primary server and specify a local path to that folder.

Note the backup schedule listed in the Schedule box under Backup job. If you want to customize the schedule for your installation, then click Schedule and adjust the SQL Server Agent schedule as needed.

- Use the default server setting, Compress backup, or do not compress backup. Click OK.
- Under Secondary server instances and databases, click Add.

9. Click Connect and connect to the instance of SQL Server that you want to use as your secondary server.
10. In the Secondary Database box, choose a database from the list or type the name of the database you want to create.
11. On the Initialize Secondary database tab, choose the option that you want to use to initialize the secondary database.

Note

If you choose to have Management Studio initialize the secondary database from a database backup, the data and log files of the secondary database are placed in the same location as the data and log files of the **master** database. This location is likely to be different than the location of the data and log files of the primary database.

12. On the Copy Files tab, in the Destination folder for copied files box, type the path of the folder into which the transaction logs backups should be copied. This folder is often located on the secondary server.
13. Note the copy schedule listed in the Schedule box under Copy job. If you want to customize the schedule for your installation, click Schedule and then adjust the SQL Server Agent schedule as needed. This schedule should approximate the backup schedule.
14. On the Restore tab, under Database state when restoring backups, choose the No recovery mode or Standby mode option.
15. If you chose the Standby mode option, choose if you want to disconnect users from the secondary database while the restore operation is underway.
16. If you want to delay the restore process on the secondary server, choose a delay time under Delay restoring backups at least.
17. Choose an alert threshold under Alert if no restore occurs within.
18. Note the restore schedule listed in the Schedule box under Restore job. If you want to customize the schedule for your installation, click Schedule and then adjust the SQL Server Agent schedule as needed. This schedule should approximate the backup schedule.
19. Click OK.

20. Under Monitor server instance, select the Use a monitor server instance check box, and then click Settings.

Note:

To monitor this log shipping configuration, you must add the monitor server now. To add the monitor server later, you would need to remove this log shipping configuration and then replace it with a new configuration that includes a monitor server.

21. Click Connect and connect to the instance of SQL Server that you want to use as your monitor server.
22. Under Monitor connections, choose the connection method to be used by the backup, copy, and restore jobs to connect to the monitor server.
23. Under History retention, choose the length of time you want to retain a record of your log shipping history.
24. Click OK.
25. On the Database Properties dialog box, click OK to begin the configuration process.

B. Using Transact SQL

1. Initialize the secondary database by restoring a full backup of the primary database on the secondary server.
2. On the primary server, execute **sp_add_log_shipping_primary_database** to add a primary database. The stored procedure returns the backup job ID and primary ID.
3. On the primary server, execute **sp_add_jobschedule** to add a schedule for the backup job.
4. On the monitor server, execute **sp_add_log_shipping_alert_job** to add the alert job.
5. On the primary server, enable the backup job.
6. On the secondary server, execute **sp_add_log_shipping_secondary_primary** supplying the details of the primary server and database. This stored procedure returns the secondary ID and the copy and restore job IDs.
7. On the secondary server, execute **sp_add_jobschedule** to set the schedule for the copy and restore jobs.
8. On the secondary server, execute **sp_add_log_shipping_secondary_database** to add a secondary database.
9. On the primary server, execute **sp_add_log_shipping_primary_secondary** to add the required information about the new secondary database to the primary server.
10. On the secondary server, enable the copy and restore jobs.

Exercise

1. Take Three SQL Server instances hosted on different computers named Serv_A and Serv_B . Serv_A is a primary server and Serv_B is a secondary server. Configure Log shipping to transfer transaction log files from server A to Server B.

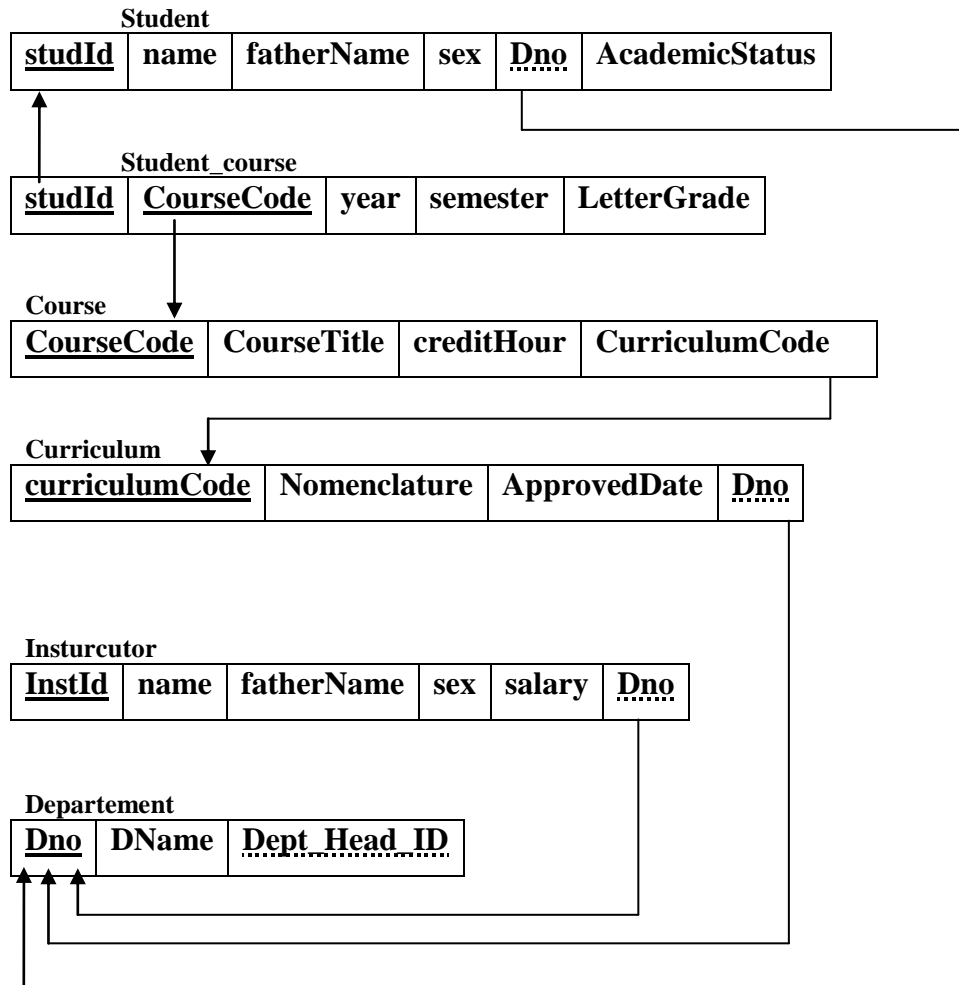
Bibliography

- 1) Microsoft SQL Server 2012 step by step. O'Reilly Media, Inc. 1005 Gravenstein Highway North Sebastopol, California 95472
- 2) Microsoft SQL Server 2008, All –in –one Desk Reference for Dummies. 2008. Robert D.Schneider. Wiley publishing inc.
- 3) Microsoft SQL Server 2008 a beginner's guide. 2008. DusanPetkovic. McGraw Hill.
- 4) Oracle Database 11g, the complete Reference. 2009. Kevin Loney, McGraw Hill.

Annexes

Annex 1

Logical data model of XYZ Registrar database



Annex 2

Logical data model of ABC publishing database

- The attribute “City” in the publisher table can have values like AA, Jimma, New York, London, etc...

