საქართველოს ტექნიკური უნივერსიტეტი

რომან სამხარაძე, ნინო ჯოჯუა, ლია გაჩეჩილაძე

# მეთოდური მითითებები ლაბორატორიული სამუშაოების შესასრულებლად საგანში:

# Transact-SQL ენა 2

## II ნაწილი

საქართველოს ტექნიკური უნივერსიტეტი

რომან სამხარაძე, ნინო ჯოჯუა, ლია გაჩეჩილაძე

# მეთოდური მითითებები ლაბორატორიული სამუშაოების შესასრულებლად საგანში:

# Transact-SQL ენა 2

## II ნაწილი

დამტკიცებულია
მეთოდურ მითითებად
სტუ-ის სარედაქციო
საგამომცემლო
საბჭოს მიერ

თბილისი - 2013

მეთოდიკური მითითებების მეორე ნაწილი განკუთვნილია იმს ფაკულტეტის ბაკალავრიატის სტუდენტებისთვის ლაბორატორიული სამუშაოების ჩასატარებლად. მათში განხილულია Transact-SQL ენის ისეთი ფუნდამენტალური საკითხები, როგორიც არის: ფუნქციები, შენახული პროცედურები, კურსორები, წარმოდგენები, ინდექსები და ა. შ. თითოეულ ლაბორატორიულ სამუშაოს გააჩნია საკითხის შესაბამისი მოკლე თეორიული ცნობები და მისი პრაქტიკული გადაწყვეტის მეთოდი.

რეცენზენტები: ასოც. პროფესორი ალექსანდრე ბენაშვილი
ასოც. პროფესორი მზია კიკნაძე

# სარჩევი

## ლაბორატორიული სამუშაო N1
## Scalar ტიპის ფუნქციები.

Scalar ტიპის ფუნქციები იმით ხასიათდებიან, რომ გასცემენ ნებისმიერი ტიპის სკალარულ მნიშვნელობას, გარდა timestamp (rowversion), text, ntext, image, table და cursor ტიპებისა. ფუნქცია შეიძლება შეიცავდეს ერთ ან მეტ ბრძანებას, რომელიც BEGIN...END ბლოკში უნდა იყოს მოთავსებული

ამ ტიპის ფუნქცია იქმნება CREATE FUNCTION ბრძანების გამოყენებით. მისი სინტაქსია:

CREATE FUNCTION [ სქემის_სახელი. ] ფუნქციის_სახელი
( [ { @პარამეტრის_სახელი მონაცემთა_ტიპი [ = DEFAULT ] } [,...n] ] )
RETURNS დასაბრუნებელი_მნიშვნელობის_ტიპი
[ WITH <ფუნქციის_რეჟიმი> [,...n] ]
[ AS ]
BEGIN
ფუნქციის_კოდი
RETURN სკალარული_გამოსახულება
END

განვიხილოთ არგუმენტების დანიშნულება.

1. სქემის_სახელი მიუთითებს იმ სქემის სახელს, რომელსაც ფუნქცია ეკუთვნის.
2. ფუნქციის_სახელი არის ფუნქციის სახელი.
3. @პარამეტრის_სახელი მონაცემთა_ტიპი [= DEFAULT] განსაზღვრავს ფუნქციის პარამეტრებს. თითოეული პარამეტრის სახელი უნდა იყოს უნიკალური და იწყებოდეს @ სიმბოლოთი.
4. WITH <ფუნქციის_რეჟიმი> განსაზღვრავს დამატებით შესაძლებლობებს.

5. BEGIN...END საკვანძო სიტყვებს შორის მოთავსებულია ფუნქციის ტანი (კოდი).

6. RETURN სკალარული_გამოსახულება ფუნქციის მუშაობას ამთავრებს და შედეგს გასცემს.

დავალება 1. გამოვთვალოთ მითითებული განყოფილების მაქსიმალური ხელფასი.

```
CREATE FUNCTION Maqsimaluri_Xelfasi( @ganyofileba
nvarchar(30) )
RETURNS float
AS
BEGIN
DECLARE @Max_xelfasi float
SET @Max_xelfasi =
(
SELECT MAX(xelfasi) FROM Personali WHERE ganyofileba =
@ganyofileba GROUP BY ganyofileba
)
RETURN @Max_xelfasi
END
```

ფუნქციის გამოსაძახებლად შეგვაქვს მოთხოვნა:

```
SELECT dbo.Maqsimaluri_Xelfasi(N'სამედიცინო') AS
[სამედიცინო განყოფილების მაქსიმალური ხელფასი]
```

## ლაბორატორიული სამუშაო N2
## Inline ტიპის ფუნქციები.

Inline ტიპის ფუნქციები შეიცავენ მხოლოდ ერთ SELECT ბრძანებას. მისი საშუალებით ფორმირდება მონაცემთა ნაკრები, რომელიც გაიცემა table ტიპის მნიშვნელობის სახით.

ამ ტიპის ფუნქცია იქმნება CREATE FUNCTION ბრძანებით, რომლის სინტაქსია:

CREATE FUNCTION [ სქემის_სახელი.] ფუნქციის_სახელი

( [ { @პარამეტრის_სახელი მონაცემთა_ტიპი [ = DEFAULT ] } [,...n] ] )

RETURNS TABLE

[ WITH <ფუნქციის_რეჟიმი> [,...n]

[ AS ]

RETURN [ ( ] select_ბრძანება [ ) ]

როგორც სინტაქსიდან ჩანს, RETURNS სიტყვის შემდეგ მითითებულია TABLE სიტყვა. ეს იმას ნიშნავს, რომ ფუნქცია აბრუნებს table ტიპის მნიშვნელობას, რომელსაც ექნება SELECT მოთხოვნის მიერ გაცემული შედეგის სტრუქტურა.

Inline ფუნქციის თავისებურებაა ის, რომ table მნიშვნელობის სტრუქტურა დინამიურად იქმნება მოთხოვნის შესრულების დროს. გაცემული table მნიშვნელობა შეგვიძლია უშუალოდ გამოვიყენოთ SELECT მოთხოვნის FROM განყოფილებაში ფუნქციის სახელის მითითების გზით.

მაგალითი.

დავალება 1. ფუნქცია აბრუნებს ინფორმაციას მითითებულ ქალაქში მდებარე იურიდიული პირების შესახებ.

```
USE Shekveta
GO
CREATE FUNCTION Iuridiuli_Pirebi1(@qalaqi nvarchar(30), @iuridiuli_fizikuri nvarchar(30))
RETURNS TABLE
AS
RETURN
SELECT * FROM Shemkveti WHERE iuridiuli_fizikuri = @iuridiuli_fizikuri AND qalaqi = @qalaqi
```

ფუნქციის გამოსაძახებლად შეგვაქვს მოთხოვნა:

```
SELECT iuridiuli_fizikuri, gvari, qalaqi
FROM Iuridiuli_Pirebi1(N'თბილისი', N'იურიდიული') ORDER BY 1
```

დავალება 2. გამოვთვალოთ საშუალო ხელფასი განყოფილებების მიხედვით.

```
CREATE FUNCTION Sashualo_Xelfasi()
RETURNS TABLE
AS
```

```
RETURN
SELECT ganyofileba AS [განყოფილება], AVG(xelfasi) AS [საშუალო ხელფასი] FROM Personali
GROUP BY ganyofileba
```

ფუნქციის გამოსაძახებლად შეგვაქვს მოთხოვნა:

```
SELECT * FROM Sashualo_Xelfasi() ORDER BY 2
```

## ლაბორატორიული სამუშაო N3
## Multi-statement ტიპის ფუნქციები.

Multi-statement ტიპის ფუნქციები გასცემენ table ტიპის მნიშვნელობას, რომელიც მონაცემებს შეიცავს. ამასთან, ფუნქციის ტანი შეიძლება რამდენიმე ბრძანებას შეიცავდეს.

ამ ტიპის ფუნქცია იქმნება CREATE FUNCTION ბრძანებით, რომლის სინტაქსია:

CREATE FUNCTION [ სქემის_სახელი.] ფუნქციის_სახელი

( [ { @პარამეტრის_სახელი მონაცემთა_ტიპი [ = DEFAULT ] } [,...n] ] )

RETURNS @დასაბრუნებელი_ცვლადი TABLE <ცხრილის_აღწერა>

[ WITH <ფუნქციის_რეჟიმი> [,...n]

[ AS ]

BEGIN

ფუნქციის_კოდი

RETURN

END

როგორც ვხედავთ, TABLE სიტყვის შემდეგ აშკარად განისაზღვრება დასაბრუნებელი მნიშვნელობის სტრუქტურა. <ცხრილის_აღწერა> სტრუქტურის სინტაქსია:

<ცხრილის_აღწერა> ::= ( { <სვეტის_განსაზღვრა> | <ცხრილის_შეზღუდვა> } [,...n] )

ეს სინტაქსი მთლიანად ემთხვევა ცხრილის შექმნისას გამოყენებულ სინტაქსს.

ფუნქციის ტანში დასაშვებია Transact_SQL-ის ნებისმიერი კონსტრუქციის გამოყენება.

მონაცემთა დასაბრუნებელი ნაკრები უნდა შეივსოს INSERT ბრძანებით. INSERT ბრძანებაში აშკარად უნდა მივუთითოთ ობიექტის სახელი, რომელშიც უნდა მოხდეს სტრიქონების ჩასმა. ამიტომ, table ტიპის მნიშვნელობას აუცილებლად უნდა მიენიჭოს სახელი. სწორედ ამ სახელს შეიცავს @დასაბრუნებელი_ცვლადი პარამეტრი.

Multi_statement ფუნქციაში RETURN სიტყვის შემდეგ არ არის აუცილებელი დასაბრუნებელი მნიშვნელობის მითითება. სერვერი ავტომატურად დააბრუნებს table ტიპის მონაცემთა ნაკრებს, რომლის სახელი და სტრუქტურა მითითებული იყო RETURNS სიტყვის შემდეგ.

დავალება 1. შევადგინოთ Multi-statement ტიპის ფუნქცია, რომელსაც 2 პარამეტრი აქვს: მთელი ტიპის და სტრიქონული. ფუნქცია გასცემს 3 სვეტისგან შემდგარ ცხრილს. პირველი სვეტი არის პირველადი გასაღები, აქვს მთელი ტიპი, არ იღებს ნულოვან მნიშვნელობებს, მისი მნიშვნელობები იწყება 1-დან და ნაზრდი ერთის ტოლია. მეორე სვეტს აქვს მთელი ტიპი, მესამე სვეტს კი სტრიქონული. დასაბრუნებელ ცხრილს დავუმატოთ 3 სტრიქონი. მეორე და მესამე სვეტებში უნდა ჩავწეროთ ფუნქციის პარამეტრები. ყოველი ჩაწერის წინ მთელი ტიპის პარამეტრის მნიშვნელობა 10-ით გავზარდოთ.

```
CREATE FUNCTION f1 (@ricxvi INT, @striqoni NVARCHAR(20))
RETURNS @table1 TABLE
(
```

```
id1 INT PRIMARY KEY IDENTITY (1,1) NOT NULL, num INT, val NVARCHAR(20)
)
AS
BEGIN
SET @ricxvi = @ricxvi + 10
INSERT INTO @table1 VALUES(@ricxvi, @striqoni)
set @ricxvi = @ricxvi + 10
INSERT INTO @table1 VALUES(@ricxvi, @striqoni)
set @ricxvi = @ricxvi + 10
INSERT INTO @table1 VALUES(@ricxvi, @striqoni)
RETURN
END
```

ფუნქციის გამოძახება:

```
USE Shekveta
SELECT * FROM f1(15, N'საბა')
```

## ლაბორატორიული სამუშაო N4
## მათემატიკის ფუნქციები.

მათემატიკის ფუნქციების უმრავლესობა გასცემს იმ ტიპის შედეგს, რა ტიპიც აქვს არგუმენტს.

FLOOR (რიცხვითი_გამოსახულება) - მითითებული არგუმენტისთვის გასცემს უახლოეს მინიმალურ მთელ რიცხვს, ანუ ასრულებს დამრგვალებას ქვევით

CEILING (რიცხვითი_გამოსახულება) გასცემს უახლოეს მთელ რიცხვს, რომელიც არგუმენტზე მეტი ან ტოლია, ანუ ასრულებს დამრგვალებას ზევით

ROUND (რიცხვითი_გამოსახულება, სიგრძე [, ფუნქცია]) ასრულებს რიცხვითი_გამოსახულება დამრგვალებას მითითებული სიზუსტით. დამრგვალება შეიძლება შეს-რულდეს როგორც ათობითი წერტილის შემდეგ (სიგრძე > 0), ისე ათობით წერტილამდე (სიგრძე < 0)

POWER (რიცხვითი_გამოსახულება, ხარისხი) გასცემს რიცხვის ხარისხს

SQRT (წილადი_გამოსახულება) არგუმენტიდან იღებს კვადრატულ ფესვს

დავალება 1 . დაამრგვალეთ რიცხვები

```
SELECT FLOOR(3.7) AS [დამრგვალება ქვევით],
       CEILING(3.4) AS [დამრგვალება ზევით],
       ROUND(123.78956, 2) AS [დამრგვალება წერტილის
       შემდეგ],
       ROUND(9876.781, -2) AS [დამრგვალება წერტილამდე]
```

დავალება 2. გამოთვალეთ 5-ის მე-6 ხარისხი
SELECT POWER(5,6) AS [რიცხვის ხარისხი],

დავალება 3. გამოთვალეთ კვადრატული ფესვი 625 რიცხვიდან

SQRT(625) AS [კვადრატული ფესვი რიცხვიდან]

დავალება 4. მიიღეთ რაიმე 0 რიცხვის სინუსი და -10 რიცხვის აბსოლუტური მნიშვნელობა

SELECT SIN(0) AS [სინუსი], ABS(-10) AS [აბსოლუტური მნიშვნელობა]

## ლაბორატორიული სამუშაო N5
## შენახული პროცედურების შექმნა

შენახული პროცედურის შექმნამდე უნდა განისაზღვროს შესაქმნელი შენახული პროცედურის ტიპი, პარამეტრები, მიმართვის უფლებები და შემუშავდეს კოდი.

შენახული პროცედურის შესაქმნელად გამოიყენება CREATE PROCEDURE ბრძანება. მისი სინტაქსია:

CREATE PROC [ EDURE ] [სქემის_სახელი.] პროცედურის_სახელი

[ { @პარამეტრის_სახელი მონაცემთა_ტიპი } [ VARYING ] [ = DEFAULT ] [ OUTPUT ] ] [,...n]

[ WITH { RECOMPILE | ENCRYPTION | RECOMPILE, ENCRYPTION } ]

AS sql_ბრძანება [,...n]

სადაც,

1. პროცედურის_სახელი არის შესაქმნელი პროცედურის
2. @პარამეტრის_სახელი ცვლადი შეიცავს იმ პარამეტრის სახელს, რომელსაც შენახული პროცედურა გამოიყენებს შესასვლელი და გამოსასვლელი მონაცემების გადაცემის მიზნით.
3. მონაცემთა_ტიპი არის პარამეტრის ტიპი.
4. OUTPUT არგუმენტი მიუთითებს, რომ პარამეტრი არის გამოსასვლელი,
5. VARYING არგუმენტს აქვს cursor ტიპი და გამოიყენება OUTPUT არგუმენტთან ერთად.

6. RECOMPILE არგუმენტი მიუთითებს, რომ შენახული პროცედურის ყოველი გამოძახებისას უნდა შედგეს მისი შესრულების გეგმა და შესრულდეს შენახული პროცედურის კოდის ხელახალი კომპილირება.

7. ENCRYPTION არგუმენტი მიუთითებს, რომ უნდა შესრულდეს შენახული პროცედურის კოდის დაშიფვრა.

8. AS არგუმენტი იწყებს შენახული პროცედურის ტანს (კოდს). ტანში დასაშვებია Transact_SQL-ის პრაქტიკულად ყველა ბრძანების გამოყენება, ტრანზაქციების გამოცხადება და სხვა შენახული პროცედურების გამოძახება. შენახული პროცედურიდან გამოსასვლელად შეგვიძლია გამოვიყენოთ RETURN ბრძანება.

დავალება 1. შევქმნათ შენახული პროცედურა, რომელსაც გადავცემთ განყოფილების სახელს და მივიღებთ ინფორმაციას ამ განყოფილებაში მომუშავე თანამშრომლების შესახებ.

```
USE Shekveta
GO
-- თუ შენახული პროცედურა არსებობს, მაშინ ის წაიშლება

IF OBJECT_ID ('[dbo].[MYPROC_PAR]', 'P' ) IS NOT NULL
DROP PROCEDURE [dbo].[MYPROC_PAR];
GO
-- შენახული პროცედურის შექმნა
CREATE PROCEDURE [dbo].[MYPROC_PAR] @ganyofileba
nvarchar(30)
```

```
AS
SELECT * FROM Personali WHERE ganyofileba = @ganyofileba
```

გამოვიძახოთ ეს პროცედურა:

```
    EXEC [dbo].[MYPROC_PAR] N'სავაჭრო'
```

დავალება 2. შევადგინოთ შენახული პროცედურა, რომელიც გასცემს ინფორმაციას მითითებულ ქალაქში და მითითებულ განყოფილებაში მომუშავე თანამშრომლების შესახებ.

```
USE Shekveta
GO
-- თუ შენახული პროცედურა არსებობს, მაშინ ის წაიშლება

IF OBJECT_ID ( '[dbo].[qalaqi_ganyofileba]', 'P' ) IS NOT NULL
DROP PROCEDURE [dbo].[Sashualo];
GO
-- შენახული პროცედურის შექმნა
CREATE PROC qalaqi_ganyofileba @qalaqi nvarchar(30),
@ganyofileba nvarchar(30) = N'სამედიცინო'
AS
SELECT * FROM Personali WHERE @qalaqi = qalaqi AND
@ganyofileba = ganyofileba
```

გამოვიძახოთ ეს პროცედურა:

```
    EXEC qalaqi_ganyofileba N'თბილისი', N'სავაჭრო
```

## ლაბორატორიული სამუშაო N6
## ინდექსების შექმნა

ინდექსი მონაცემთა ბაზის დამოუკიდებელ ობიექტს წარმოადგენს. ის შეიძლება მოიცავდეს ცხრილის ერთ ან მეტ სვეტს.

არსებობს კლასტერული, არაკლასტერული და უნიკალური ინდექსები.

არაკლასტერული ინდექსები ახდენენ მიმართვების ორგანიზებას შესაბამის სტრიქონებზე.

კლასტერული ინდექსის გამოყენების დროს ცხრილში მონაცემების ფიზიკური განლაგება გადაეწყობა ინდექსის სტრუქტურის შესაბამისად. ამ შემთხვევაში ცხრილის ლოგიკური სტრუქტურა ლექსიკონს უფრო წააგავს.

უნიკალური ინდექსები იძლევიან ინდექსირებულ ცხრილში მნიშვნელობების უნიკალურობის გარანტიას. ის შეიძლება რეალიზებული იყოს როგორც კლასტერული, ისე არაკლასტერული ინდექსისათვის. ერთ ცხრილში შეიძლება არსებობდეს ერთი უნიკალური კლასტერული ინდექსი და რამდენიმე უნიკალური არაკლასტერული ინდექსი.

მონაცემების მთლიანობის უზრუნველსაყოფად უმჯობესია UNIQUE ან PRIMARY KEY შეზღუდვების და არა უნიკალური ინდექსის გამოყენება.

ინდექსის CREATE INDEX ბრძანებით შექმნის სინტაქსია
CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ]
INDEX ინდექსის_სახელი ON { <ობიექტი> } ( სვეტის_სახელი
[ ASC | DESC ] [,...n] )

[ WITH
[ PAD_INDEX ]
[ [ , ] FILLFACTOR = შევსების_ფაქტორი ]
[ [ , ] IGNORE_DUP_KEY ]
[ [ , ] DROP_EXISTING ]
[ [ , ] STATISTICS_NORECOMPUTE = { ON | OFF } ]

<ობიექტი> კონსტრუქციის სინტაქსია:
<ობიექტი> ::= [ მონაცემთა_ბაზის_სახელი. [ სქემის_სახელი ] . ] ცხრილის_ან_წარმოდგენის_სახელი

სადაც,

1. UNIQUE არგუმენტის მითითების შემთხვევაში შეიქმნება უნიკალური ინდექსი.

2. საინდექსებელ სვეტში სასურველია, აგრეთვე, NULL მნიშვნელობის შენახვის აკრძალვა, რათა ავიცილოთ მნიშვნელობების უნიკალურობასთან დაკავშირებული პრობლემები.

3. CLUSTERED მიუთითებს, რომ შესაქმნელი ინდექსი კლასტერული იქნება,

4. NONCLUSTERED მიუთითებს, რომ შესაქმნელი ინდექსი იქნება არაკლასტერული.

5. ინდექსის_სახელი უნიკალური უნდა იყოს ცხრილის ფარგლებში.

6. სვეტის_სახელი განსაზღვრავს იმ სვეტებს, რომლებიც ჩართული იქნება შესაქმნელ ინდექსში.

7. [ ASC | DESC ] არგუმენტი ინდექსში განსაზღვრავს საკვანძო ელემენტების დახარისხების მეთოდს. ASC შემთხვევაში ელემენტები დალაგდება ზრდადობის

მიხედვით, DESC შემთხვევაში კი - კლებადობის მიხედვით. ავტომატურად იგულისხმება ASC.

ინდექსი ყოველთვის იქმნება მხოლოდ მიმდინარე მონაცემთა ბაზის ცხრილის ან წარმოდგენისათვის.

დავალება 1. ganyofileba სვეტისთვის შევქმნათ უნიკალური კლასტერული Ind_ganyofileba ინდექსი.

```
USE Shekveta;
GO
თუ ინდექსი არსებობს, მაშინ ის წაიშლება
IF EXISTS ( SELECT name FROM sys.indexes WHERE name = N'Ind_ganyofileba' )
DROP INDEX Ind_ganyofileba ON Personali
GO
ინდექსის შექმნა
CREATE UNIQUE CLUSTERED INDEX Ind_ganyofileba
ON [dbo].[Personali] ([ganyofileba])
```

დავალება 2. შევქმნათ Ind_qalaqi_gvari უნიკალური, არაკლასტერული, შედგენილი ინდექსი qalaqi და gvari სვეტებისათვის. შევსების ფაქტორია 40%. qalaqi სვეტი დავალაგოთ კლებადობით, gvari სვეტიკი - ზრდადობით.

```
USE Shekveta;
GO
-- თუ ინდექსი არსებობს, მაშინ ის წაიშლება
```

```
IF EXISTS ( SELECT name FROM sys.indexes WHERE name =
N'Ind_qalaqi_gvari' )
DROP INDEX Ind_qalaqi_gvari ON Personali
GO
-- ინდექსის შექმნა
CREATE UNIQUE NONCLUSTERED INDEX Ind_qalaqi_gvari
ON Personali ([qalaqi] DESC, [gvari] ASC) WITH FILLFACTOR =
40
```

# ლაბორატორიული სამუშაო N7
# წარმოდგენის შექმნა

წარმოდგენა არის ვირტუალური ცხრილი და შედგება სვეტებისა და სტრიქონებისაგან, რომლებიც დინამიურად ამოირჩევა ერთი ან მეტი ცხრილიდან და/ან წარმოდგენიდან. ფიზიკურად წარმოდგენას SELECT მოთხოვნის სახე აქვს, რომლის საფუძველზეც სრულდება მონაცემების ამორჩევა.

წარმოდეგენის შექმნის წინ უნდა შევამოწმოთ გვაქვს თუ არა საწყის ცხრილებთან მიმართვის და მონაცემთა ბაზაში ობიექტების შექმნისათვის საჭირო უფლებები. წარმოდეგენის შესაქმნელად გამოიყენება CREATE VIEW ბრძანება. მისი სინტაქსია:

CREATE VIEW [ სქემის_სახელი . ] წარმოდგენის_სახელი [ (სვეტის_სახელი[ ,...n])][WITH <წარმოდგენის_ატრიბუტები> [ ,...n ] ] AS select_ბრძანება [ WITH CHECK OPTION ]

<წარმოდგენის_ატრიბუტები> კონსტრუქციის სინტაქსია:

<წარმოდგენის_ატრიბუტები> ::= { ENCRYPTION | SCHEMABINDING | VIEW_METADATA }

სადაც,

1. სვეტის_სახელი არის იმ სვეტის სახელი, რომელიც ჩართული იქნება წარმოდგენაში.
2. ENCRYPTION არგუმენტი მიუთითებს, რომ უნდა მოხდეს წარმოდგენის კოდის დაშიფვრა.
3. SCHEMABINDING არგუმენტი თუ მთითებულია წარმოდგენის შექმნის დროს, მაშინ სერვერი წარმოდგენის

სტრუქტურას დააკავშირებს იმ ობიექტების სტრუქტურასთან, რომლებსაც SELECT მოთხოვნა მიმართავს.

4. VIEW_METADATA არგუმენტის მითითების შემთხვევაში სერვერი გასცემს წარმოდგენის მეტამონაცემებს DB-LIBRARY და OLE DB ტექნოლოგიებისათვის.

5. WITH CHECK OPTION არგუმენტი მიუთითებს, რომ უნდა შესრულდეს იმ ცვლილებების შემოწმება, რომლებიც წარმოდგენის საშუალებით შესრულდება.

ა. შევქმნათ წარმოდგენა, რომელშიც გამოჩნდება Xelshekruleba ცხრილის ყველა სვეტი. წარმოდგენაში უნდა გამოჩნდეს ინფორმაცია მევალეების შესახებ.

```
USE Shekveta
GO
თუ წარმოდგენა არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID('View_3', 'VIEW') IS NOT NULL
DROP VIEW View_3
GO
წარმოდგენის შექმნა
CREATE VIEW View_3
AS
SELECT * FROM Xelshekruleba WHERE vali_l > 0
GO
წარმოდგენის გახსნა:
SELECT * FROM View_3
```

დავალება 2. შევქმნათ წარმოდგენა, რომელშიც გამოჩნდება Personali ცხრილის gvari სვეტი და Shemkveti ცხრილის qalaqi სვეტი. Personali ცხრილი არის მთავარი, Shemkveti ცხრილი - კი დამოკიდებული. გამოვიტანოთ ინფორმაცია იმ თანამშრომლების შესახებ, რომელთა ასაკი 45-ზე ნაკლებია. თანამშრომლების გვარები კლებადობით დავალაგოთ.

```
USE Shekveta
GO
-- თუ წარმოდგენა არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID('View_2', 'VIEW') IS NOT NULL
DROP VIEW View_2
GO
-- წარმოდგენის შექმნა
CREATE VIEW View_2
AS
SELECT TOP 100 PERCENT dbo.Personali.gvari, dbo.Shemkveti.qalaqi
FROM dbo.Personali INNER JOIN
dbo.Shemkveti ON dbo.Personali.PersonaliID = dbo.Shemkveti.PersonaliID
WHERE dbo.Personali.asaki < 45
ORDER BY dbo.Personali.gvari DESC
GO
```

წარმოდგენის გახსნა:

```
SELECT * FROM View_2
```

## ლაბორატორიული სამუშაო N8
## კურსორის შექმნა, გახსნა, მონაცემების წაკითხვა, შეცვლა და წაშლა, დახურვა და გათავისუფლება.

კურსორი კლიენტის პროგრამას საშუალებას აძლევს იმუშაოს არა ასობით ან ათასობით სტრიქონთან, არამედ ერთ სტრიქონთან ან სტრიქონების მცირე ბლოკთან.

კურსორის შესაქმნელად გამოიყენება DECLARE CURSOR ბრძანება.

SQL-92 სტანდარტით განსაზღვრული DECLARE CURSOR ბრძანების სინტაქსია:

DECLARE კურსორის_სახელი [ INSENSITIVE ] [ SCROLL ] CURSOR FOR select_ბრძანება [ FOR { READ ONLY | UPDATE [ OF სვეტის_სახელი [ ,...n ] ] } ]

სადაც,

1. INSENSITIVE. მიუთითებს, რომ უნდა შეიქმნას სტატიკური კურსორი. თუ ის მითითებული არ არის, მაშინ შეიქმნება დინამიური კურსორი.

2. SCROLL მიუთითებს, რომ შეიქმნება გადახვევადი კურსორი. თუ ის მითითებული არ არის, მაშინ შეიქმნება მიმდევრობითი კურსორი.

3. FOR select_ბრძანება. ეს არგუმენტი შეიცავს SELECT მოთხოვნას, რომელიც გასცემს კურსორის სტრიქონების შედეგობრივ ნაკრებს.

4. FOR READ ONLY თუ მითითებულია, მაშინ შეიქმნება „მხოლოდ წაკითხვადი‖ კურსორი.

5. FOR UPDATE [ OF სვეტის_სახელი [ ,...n ] ] თუ მითითებულია, მაშინ კურსორში შესაძლებელი იქნება სტრიქონების ცვლილება.

DECLARE CURSOR ბრძანების Transact-SQL-ით განსაზღვრული სინტაქსია:

DECLARE კურსორის_სახელი CURSOR [LOCAL | GLOBAL] [ FORWARD_ONLY | SCROLL ] [ STATIC | KEYSET | DYNAMIC | FAST_FORWARD ] [ READ_ONLY | SCROLL_LOCKS | OPTIMISTIC ] [ TYPE_WARNING ] FOR select_ბრძანება [ FOR UPDATE [ OF სვეტის_სახელი [ ,...n ] ] ]

სადაც,

1. LOCAL მიუთითებს, რომ შეიქმნება ლოკალური კურსორი, რომელიც ხილული იქნება მხოლოდ ამ კურსორის შემქმნელი პაკეტის, ტრიგერის, შენახული პროცედურის ან მომხმარებლის მიერ შექმნილი ფუნქციის შიგნით.
2. GLOBAL მიუთითებს, რომ შეიქმნება გლობალური კურსორი, რომელიც იარსებებს მიმდინარე შეერთების დახურვამდე.
3. FORWARD_ONLY მიუთითებს, რომ შეიქმნება მიმდევრობითი კურსორი.
4. SCROLL მიუთითებს, რომ შეიქმნება გადახვევადი კურსორი.
5. STATIC მიუთითებს, რომ შეიქმნება სტატიკური კურსორი.
6. KEYSET მიუთითებს, რომ შეიქმნება საგასაღებო კურსორი.

7. DYNAMIC მიუთითებს, რომ შეიქმნება დინამიური კურსორი.

8. FAST_FORWARD თუ მითითებულია READ_ONLY არგუმენტთან ერთად, მაშინ მოხდება შექმნილი კურსორის ოპტიმიზება მონაცემებთან სწრაფი მიმართვის მიზნით.

9. OPTIMISTIC მიუთითებს, რომ კურსორში იკრძალება იმ სტრიქონების შეცვლა ან წაშლა, რომლებიც ცხრილში შეიცვალა კურსორის გახსნის შემდეგ.

10. TYPE_WARNING თუ მითითებულია, მაშინ სერვერი მომხმარებელს შეატყობინებს კურსორის ტიპის შეცვლის შესახებ, თუ ის არათავსებადია SELECT მოთხოვნის ტიპთან.

დავალება 1. შექმენით სტატიკური კურსორი

```
USE Shekveta
DECLARE Xelshekruleba_Cursori INSENSITIVE CURSOR
FOR SELECT * FROM Xelshekruleba WHERE vali_l > 0
OPEN Xelshekruleba_Cursori
FETCH NEXT FROM Xelshekruleba_Cursori
CLOSE Xelshekruleba_Cursori
DEALLOCATE Xelshekruleba_Cursori
```

დავალება 2. შექმენით გადახვევადი კურსორი

```
USE Shekveta
DECLARE Xelshekruleba_Cursori SCROLL CURSOR
FOR SELECT * FROM Xelshekruleba WHERE vali_l > 0
```

```
OPEN Xelshekruleba_Cursori
FETCH NEXT FROM Xelshekruleba_Cursori
CLOSE Xelshekruleba_Cursori
DEALLOCATE Xelshekruleba_Cursori
```

დავალება 3. შექმენით მხოლოდ წაკითხვადი კურსორი

```
USE Shekveta
DECLARE Xelshekruleba_Cursori CURSOR
FOR SELECT * FROM Xelshekruleba WHERE vali_l > 0
FOR READ ONLY
OPEN Xelshekruleba_Cursori
FETCH NEXT FROM Xelshekruleba_Cursori
CLOSE Xelshekruleba_Cursori
DEALLOCATE Xelshekruleba_Cursori
```

შექმენით კურსორი, რომელშიც შესაძლებელი იქნება მონაცემების ცვლილება

```
 USE Shekveta
DECLARE Xelshekruleba_Cursori CURSOR
FOR SELECT * FROM Xelshekruleba WHERE vali_l > 0
FOR UPDATE
OPEN Xelshekruleba_Cursori
FETCH NEXT FROM Xelshekruleba_Cursori
CLOSE Xelshekruleba_Cursori
DEALLOCATE Xelshekruleba_Cursori
```

## ლაბორატორიული სამუშაო N9
## აშკარა ტრანზაქციები. BEGIN TRANSACTION, COMMIT და ROLLBACK ბრძანებები.

ტრანზაქცია არის პროცესი, როცა სრულდება მასში შემავალი ყველა ბრძანება ან არც ერთი. თუ ყველა ბრძანება წარმატებით შესრულდა, მაშინ მოხდება ტრანზაქციის ფიქსირება (Commit). თუ ტრანზაქციის რომელიმე ბრძანება ვერ შესრულდა, მაშინ მოხდება ტრანზაქციის უკუქცევა (Rollback), ანუ გაუქმდება ყველა ცვლილება.

აშკარა ტრანზაქციების გამოყენების შემთხვევაში აშკარად უნდა მიუთითოთ ტრანზაქციის დაწყება და დამთავრება. ამისათვის, გამოიყენება შემდეგი ბრძანებები: BEGIN TRANSACTION, COMMIT და ROLLBACK.

BEGIN TRANSACTION ბრძანება განსაზღვრავს ტრანზაქციის დასაწყისს. ამ დროს ტრანზაქციების ჟურნალში ფიქსირდება შესაცვლელი მონაცემების საწყისი მნიშვნელობები და ტრანზაქციის დაწყების მომენტი. მისი სინტაქსია:

BEGIN TRAN[SACTION]

[ ტრანზაქციის_სახელი | @tran_name_variable [ WITH MARK [ 'აღწერა' ] ] ]

სადაც,

1. ტრანზაქციის_სახელი ჩვეულებრივ, გამოიყენება მხოლოდ ჩადგმულ ტრანზაქციებთან სამუშაოდ ყველაზე დაბალი დონის ტრანზაქციის სახელდებისთვის.
2. @tran_name_variable ლოკალური ცვლადია, რომელიც ტრანზაქციის სახელს შეიცავს. მისი ტიპი უნდა იყოს char, varchar, nchar ან nvarchar.

3. WITH MARK ['აღწერა'] არგუმენტი საშუალებას გვაძლევს სპეციალური გზით მოვახდინოთ ტრანზაქციების მარკირება ტრანზაქციების ჟურნალში. ასეთი მარკირება საჭიროა სარეზერვო ასლიდან ტრანზაქციების ჟურნალის აღდგენისათვის, მონაცემთა ბაზის დასაბრუნებლად იმ მდგომარეობაში, რომელშიც ის იყო ტრანზაქციის დაწყებამდე ან მისი დამთავრების შემდეგ.

COMMIT TRANSACTION ან COMMIT WORK ბრძანებები განსაზღვრავენ ტრანზაქციის დასასრულს. თუ ტრანზაქციის შესრულების დროს ადგილი არ ჰქონდა შეცდომებს, მაშინ შესრულებული ცვლილებები დაფიქსირდება.

ROLLBACK TRANSACTION ან ROLLBACK WORK ბრძანებების შესრულება იწვევს ტრანზაქციის შეწყვეტას და უკუქცევას. უკუქცევის დროს, შესრულებული ცვლილებები უქმდება და აღდგება სისტემის პირვანდელი მდგომარეობა.

როცა მონაცემებს მივმართავთ OLE DB და ADO მექანიზმებით, მაშინ შეგვიძლია გამოვიყენოთ ტრანზაქციის აშკარა განსაზღვრა.

დავალება 1. ტრანზაქციის გამოყენებით Personali ცხრილში გააორმაგეთ ხელფასები.

```
USE Shekveta;
GO
DECLARE @TranSaxeli nvarchar(20);
SELECT @TranSaxeli = N'ChemiTranzaqcia';
BEGIN TRANSACTION @TranSaxeli;
```

```
UPDATE Personali_3 SET xelfasi = xelfasi * 2 --WHERE iuridiuli_fizikuri = N'ფიზიკური';
COMMIT TRANSACTION ChemiTranzaqcia;
GO
SELECT * FROM Personali_3
```

დავალება 2. ტრანზაქციის გამოყენებით Personali ცხრილში გააორმაგეთ ხელფასები. შეასრულეთ ტრანზაქციის მარკირება.

```
USE Shekveta;
GO
DECLARE @TranSaxeli nvarchar(20);
SELECT @TranSaxeli = N'ChemiTranzaqcia';
BEGIN TRANSACTION @TranSaxeli
WITH MARK N'სრულდება ხელფასების გაორმაგება';
UPDATE Personali_3 SET xelfasi = xelfasi * 2
COMMIT TRANSACTION ChemiTranzaqcia;
GO
SELECT * FROM Personali_3
```

## ლაბორატორიული სამუშაო N10

## ტრიგერის შექმნა

ტრიგერი არის შენახული პროცედურის სპეციალური ტიპი, რომელიც სერვერის მიერ ავტომატურად გაიშვება ამა თუ იმ ცხრილზე მოქმედებების შესრულების დროს.

ტრიგერის შესაქმნელად გამოიყენება CREATE TRIGGER ბრძანება. მისი სინტაქსია:

CREATE TRIGGER [ სქემის_სახელი . ] ტრიგერის_სახელი
ON { ცხრილის_სახელი | წარმოდგენის_სახელი }
[ WITH ENCRYPTION ]
{
{ { FOR | AFTER | INSTEAD OF }
{ [ DELETE ] [ , ] [ INSERT ] [ , ] [ UPDATE ] }
[ NOT FOR REPLICATION ] AS sql_ბრძანება [,...] } |
{ ( FOR | AFTER | INTEAD OF ) { [ INSERT ] [ , ] [ UPDATE ] }
AS { IF UPDATE ( სვეტის_სახელი )
[ { AND | OR } UPDATE ( სვეტის_სახელი ) ] [,...n] |
IF ( COLUMNS_UPDATED ( ) { ბიტობრივი_ოპერატორი } ბიტების_ნიღაბი ) { შედარების_ოპერატორი } სვეტის_ნიღაბი [,...n] } sql_ბრძანება [,...n] } }

სადაც,

1. ტრიგერის_სახელი უნდა იყოს უნიკალური მონაცემთა ბაზის ფარგლებში.
2. ცხრილის_სახელი | წარმოდგენის_სახელი იმ ცხრილის ან წარმოდგენის სახელია, რომელსაც ტრიგერი უკავშირდება.

3. თუ მითითებულია AFTER, მაშინ ტრიგერი მხოლოდ იმ შემთხვევაში გაიშვება, როცა მისი გამომძახებელი ბრძანებები წარმატებით შესრულდება.

4. INSTEAD OF თუ მითითებულია, მაშინ ტრიგერი შესრულდება მისი გამომძახებელი მოთხოვნის ნაცვლად.

5. [ DELETE ] [ , ] [ INSERT ] [ , ] [ UPDATE ]. ეს კონსტრუქცია განსაზღვრავს, თუ რომელ ბრძანებაზე მოახდენს ტრიგერი რეაგირებას.

6. AS sql_ბრძანება [,...n] არგუმენტი შეიცავს Transact_SQL-ის ბრძანებებს, რომლებიც შესრულდება ტრიგერის გაშვებისას.

7. FOR [ INSERT ] [ , ] [ UPDATE ]. განსაზღვრავს ბრძანებას, რომლის შესრულების დროსაც ტრიგერი გაიშვება.

8. IF UPDATE (სვეტის_სახელი) გამოყენება საშუალებას გვაძლევს ტრიგერი შევასრულოთ ცხრილის კონკრეტული სვეტის მოდიფიცირების დროს.

9. { AND | OR } UPDATE (სვეტის_სახელი) კონსტრუქცია გამოიყენება წინა არგუმენტთან ერთად, თუ საჭიროა ტრიგერის გაშვება რამდენიმე სვეტის მოდიფიცირების შემთხვევაში.

10. IF (COLUMNS_UPDATED()) გვატყობინებს თუ რომელი სვეტები შეიცვალა ან დაემატა.

11. ბიტობრივი_ოპერატორი არის ბიტობრივი დამუშავების ოპერატორი, რომლის გამოყენებით შეგვიძლია განვსაზღვროთ შეიცვალა თუ არა კონკრეტული სვეტი.

12. ბიტების_ნიღაბი განსაზღვრავს ბიტების ნიღაბს ერთ ან მეტ სვეტში ცვლილებების განსაზღვრისათვის.

13. შედარების_ოპერატორია, რომელიც განკუთვნილია შედარების ბიტობრივი ოპერაციის შესრულების შედეგად დაბრუნებული მნიშვნელობის შესადარებლად გარკვეულ კრიტერიუმებთან.

14. სვეტის_ნიღაბი ბიტების ნიღაბია, რომელიც განსაზღვრავს მართლა შეიცვალა თუ არა შესამოწმებელი სვეტები.

დავალება 1. შევქმნათ Personali_2 ცხრილი. მისთვის შევქმნათ UPDATE ტრიგერი, რომელიც გამოიტანს შეცვლილი სტრიქონების რაოდენობას და ინფორმაციას შეცვლის მცდელობის შესახებ.

```
USE Shekveta
GO
-- თუ ტრიგერი არსებობს, მაშინ ის წაიშლება
IF EXISTS ( SELECT name FROM sysobjects WHERE name = 'Personali3_Update' AND type = 'TR' )
DROP TRIGGER Personali3_Update
GO
-- ტრიგერის შექმნა
CREATE TRIGGER Personali3_Update ON Personali_3
FOR UPDATE AS
PRINT STR(@@ROWCOUNT) + N' სტრიქონის წაშლის მცდელობა Personali_2 ცხრილიდან'
PRINT N'მომხმარებელი - ' + CURRENT_USER
```

```
IF CURRENT_USER <> 'dbo'
BEGIN
PRINT N'შეცვლა აკრძალულია'
ROLLBACK TRANSACTION
END
ELSE
PRINT N'შეცვლა ნებადართულია'
```

ახლა Personali_2 ცხრილიდან ერთი სტრიქონი წავშალოთ:

```
UPDATE Personali_3 SET xelfasi = xelfasi + 300 WHERE ganyofileba = N'სამედიცინო'
```

დავალება 2. შევქმნათ ტრიგერი, რომელიც ახალ ხელშეკრულებას არ გაგვაფორმებინებს შემკვეთთან, რომლის სახელი და გვარია 'სამხარაძე ბექა'.

```
USE Shekveta
GO
-- თუ ტრიგერი არსებობს, მაშინ ის წაიშლება
IF EXISTS ( SELECT name FROM sysobjects WHERE name = 'Trigeri_3' AND type = 'TR' )
DROP TRIGGER Trigeri_3
GO
-- ტრიგერის შექმნა
CREATE TRIGGER Trigeri_3 ON Shemkveti FOR INSERT, UPDATE AS
```

```
IF EXISTS
( SELECT * FROM Shemkveti WHERE gvari = N'სამხარაძე რომანი' )
BEGIN
PRINT N'სამხარაძე რომანისთან ახალი ხელშეკრულების გაფორმება დაუშვებელია '
ROLLBACK TRANSACTION
END
```

ახლა ვცადოთ ხელშეკრულების გაფორმება შემკვეთთან - _სამხარაძე რომანი‘:

```
UPDATE Shemkveti SET gvari = N'სამხარაძე რომანი' WHERE gvari = N'ნონიაშვილი ზურა'
```

## ლაბორატორიული სამუშაო N11
## სარეზერვო ასლების შექმნა

სარეზერვო ასლი წარმოადგენს ერთ ან მეტ ფაილს, რომელშიც მონაცემთა ბაზაა მოთავსებული მთლიანად ან ნაწილობრივ. სარეზერვო ასლის შექმნა საჭიროა მონაცემების აღსადგენად კომპიუტერის, დისკის ან მონაცემების დაზიანების შემთხვევაში და ა.შ.

სრული და დიფერენცირებული სარეზერვო ასლების შესაქმნელად გამოიყენება BACKUP DATABASE ბრძანება, რომლის სინტაქსია:

BACKUP DATABASE { მონაცემთა_ბაზის_სახელი | @database_name_var } TO <ინფორმაციის_მატარებელი> [ ,...n ] [ WITH [ BLOCKSIZE = { ბლოკის_ზომა | @blocksize_variable } ] [ [ , ] DESCRIPTION = { 'ტექსტი' | @text_variable } ] [ [ , ] DIFFERENTIAL ] [ [ , ] EXPIREDATE = { თარიღი | @date_var } |

RETAINDAYS = { დღეების_რაოდენობა | @days_var } ] [ [ , ] FORMAT | NOFORMAT ] [ [ , ] { INIT | NOINIT } ] [ [ , ] MEDIADESCRIPTION = { 'ტექსტი' | @text_variable } ] [ [ , ] MEDIANAME =

{ ინფორმაციის_მატარებლის_სახელი | @media_name_variable } ] [ [ , ] MEDIAPASSWORD =

{ ინფორმაციის_მატარებლის_პაროლი | @mediapassword_variable } ] [ [ , ] NAME = { არქივის_სახელი | @backup_set_name_var } ] [ [ , ] { NOSKIP | SKIP } ] [ [ , ] { NOREWIND | REWIND } ] [ [ , ] { NOUNLOAD | UNLOAD } ]

[ [ , ] PASSWORD = { პაროლი | @password_variable } ] [ [ , ] RESTART ] ] ]

სადაც,

- მონაცემთა_ბაზის_სახელი მიუთითებს მონაცემთა ბაზის სახელს, რომლის არქივირებაც უნდა შესრულდეს.
- <ინფორმაციის_მატარებელი> მიუთითებს ინფორმაციის მატარებელს, რომელზეც უნდა მოთავსთდეს სარეზერვო ასლი.
- DISK | TAPE | PIPE არგუმენტები განსაზღვრავენ შესაქმნელი მოწყობილობის ტიპს.
- BLOCKSIZE მიუთითებს ბლოკის ზომას ბაიტებში.
- DESCRIPTION = 'ტექსტი' უნდა მივუთითოთ იმ შემთხვევაში, როცა არქივთან ერთად უნდა ჩაიწეროს მისი ტექსტური აღწერა.
- DIFFERENTIAL მიუთითებს, რომ უნდა შეიქმნას მონაცემთა ბაზის დიფერენცირებული ასლი.
- INIT მითითების შემთხვევაში სარეზერვო ასლი ინფორმაციის მატარებელზე მოთავსთდება როგორც პირველი ფაილი.
- NOINIT მითითების შემთხვევაში სარეზერვო ასლი დაემატება ინფორმაციის მატარებელზე არსებულ ფაილებს.
- RESTART თუ მითითებულია, მაშინ სერვერი გააგრძელებს სარეზერვო ასლის შექმნის შეწყვეტილ ოპერაციას.
- RETAINDAYS = დღეების_რაოდენობა განსაზღვრავს არქივის აქტუალურობას დღეების რაოდენობით, რომელიც უნდა გავიდეს არქივის შექმნის მომენტიდან.
- SKIP უნდა მივუთითოთ მაშინ, თუ გადავწყვიტეთ, რომ არქივები, რომლებისთვისაც დაყენებული იყო თარიღის გასვლის ან დროის პერიოდის ამოწურვის

პარამეტრები, აღარ გვჭირდება. ასეთ შემთხვევაში, სარეზერვო ასლი ზემოდან გადაეწერება არსებულ ასლებს.

დავალება 1. შევქმნათ Shekveta მონაცემთა ბაზის სრული ასლი.

```
USE master
EXEC sp_addumpdevice 'DISK', 'Shekveta_Asli',
'C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Backup\Shekveta_Asli.bak'
BACKUP DATABASE Shekveta TO Shekveta_Asli WITH NAME = 'Shekvata_Asli'
GO
```

დავალება 2. შევქმნათ Shekveta მონაცემთა ბაზის დიფერენცირებული ასლი:

```
USE master
GO
BACKUP DATABASE Shekveta TO Shekveta_Asli
WITH DIFFERENTIAL, NAME = 'Shekvata_Asli_Dif'
GO
```

## ლაბორატორიული სამუშაო N12
## მონაცემთა ბაზების როლები.

მონაცემთა ბაზების როლები საშუალებას გვაძლევს მომხმარებლები გავაერთიანოთ ერთ ადმინისტრაციულ ერთეულში და მასთან ვიმუშაოთ როგორც ჩვეულებრივ მომხმარებელთან.

თუ მომხმარებელს უნდა მიენიჭოს იგივე უფლებები, როგორიც მინიჭებული აქვს როლს, მაშინ ეს მომხმარებელი ამ როლში უნდა ჩავრთოთ.

მონაცემთა ბაზის როლში შეგვიძლია ჩავრთოთ:

- სერვერის მომხმარებელი;
- სერვერის როლი;
- Windows NT-ის მომხმარებლები;
- Windows NT-ის ჯგუფები, რომლებსაც წინასწარ მიცემული აქვთ საჭირო მონაცემთა ბაზასთან მიმართვის უფლება.

მიმდინარე მონაცემთა ბაზაში როლის დასამატებლად sp_addrole შენახული პროცედურა გამოიყენება. მისი სინტაქსია:

sp_addrole [ @rolename = ] 'როლის_სახელი' [ , [ @ownername = ] 'მფლობელის_სახელი' ]

სადაც, მფლობელის_სახელი ახალი როლის მფლობელის სახელი. მფლობელი უნდა იყოს მიმდინარე მონაცემთა ბაზის მომხმარებელი ან მიმდინარე მონაცემთა ბაზის როლი.

დავა;ება 1. Shekveta მონაცემთა ბაზას დავამატოთ Roli_1, რომლის მფლობელი იქნება dbo.

```
USE Shekveta
GO
sp_addrole 'Roli_1'
```

დავალება 2. Shekveta მონაცემთა ბაზაში შევქმნათ Roli_2 როლი, რომლის მფლობელი იქნება მიმდინარე მონაცემთა ბაზის User_3 მომხმარებელი.

```
USE Shekveta
GO
sp_addrole 'Roli_3', 'User_3'
```

დავალება 3. Shekveta მონაცემთა ბაზაში შევქმნათ Roli_3 როლი, რომლის მფლობელი იქნება მიმდინარე მონაცემთა ბაზის ფიქსირებული db_owner როლი.

```
USE Shekveta
GO
sp_addrole 'Roli_3', 'db_owner'
```

## ლაბორატორიული სამუშაო N13
## სააღრიცხვო ჩანაწერის შექმნა

სააღრიცხვო ჩანაწერის შესაქმნელად გამოიყენება sp_addlogin შენახული პროცედურა. მისი სინტაქსია:

sp_addlogin [@loginame=] 'სააღრიცხვო_ჩანაწერის_სახელი'
[ , [ @passwd = ] 'პაროლი' ]
[ , [ @defdb = ] 'მონაცემთა_ბაზის_სახელი' ]
[ , [ @deflanguage = ] 'ენა' ]
[ , [ @sid = ] იდენტიფიკატორი ]
[ , [ @encryptopt = ] 'დაშიფვრის_რეჟიმი' ]

სადაც,

- 'სააღრიცხვო_ჩანაწერის_სახელი. შესაქმნელი სააღრიცხვო ჩანაწერის სახელია, რომელიც გამოყენებული იქნება მომხმარებლის აუტენტიფიცირებისათვის.
- 'მონაცემთა_ბაზის_სახელი' ავტომატურად ნაგულისხმევი მონაცემთა ბაზაა, რომელსაც მომხმარებელი მიუერთდება სერვერთან კავშირის დამყარებისთანავე.
- 'ენა' არის ნაგულისხმევი ენა, რომელიც ძალაში იქნება მომხმარებლისთვის სერვერთან დაკავშირებისთანავე.
- იდენტიფიკატორი არის 16-ბაიტიანი ორობითი რიცხვი და წარმოადგენს შესაქმნელი სააღრიცხვო ჩანაწერის უსაფრთხოების იდენტიფიკატორს. ის სასარგებლოა მაშინ, როცა სერვერის სააღრიცხვო ჩანაწერები ერთი სერვერიდან მეორეზე გადაგვაქვს და

გვინდა, რომ სააღრიცხვო ჩანაწერებს ერთნაირი SID (Security ID) ჰქონდეს.

- დაშიფვრის_რეჟიმი მიუთითებს, უნდა მოხდეს თუ არა სააღრიცხვო ჩანაწერის პაროლის დაშიფვრა. ის იღებს სამ მნიშვნელობას: NULL, skip_encryption, skip_encryption_old. თუ მითითებულია NULL მნიშვნელობა, მაშინ შესრულდება პაროლის დაშიფვრა. ეს მნიშვნელობა ავტომატურად იგულისხმება. skip_encryption მნიშვნელობა მიუთითებს, რომ პაროლი დაშიფრულია და მისი განმეორებით დაშიფვრა არ შესრულდება. skip_encryption_old მნიშვნელობა მიუთითებს, რომ პაროლი დაშიფრული იყო წინა ვერსიაში და მისი განმეორებით დაშიფვრა არ შესრულდება.

დავალება 1. შექმენით სააღრიცხვო ჩანაწერი, რომლისთვისაც მითითებული იქნება პაროლი.

sp_addlogin ' Login_2', 'paroli'

დავალება 2. შექმენით სააღრიცხვო ჩანაწერი, რომლისთვისაც მითითებული იქნება პაროლი და მონაცემთა ბაზა.

sp_addlogin ' Login_3', 'paroli', 'Shekveta'

დავალება 3. შექმენით სააღრიცხვო ჩანაწერი, რომლისთვისაც მითითებული იქნება პაროლი, მონაცემთა ბაზა და ენა.

sp_addlogin ' Login_4', 'paroli', 'Shekveta', 'french'

## ლაბორატორიული სამუშაო N14
## მიმართვის უფლებები

მონაცემებთან მუშაობა და შენახული პროცედურების შესრულება მოითხოვს მიმართვის კლასის არსებობას, რომელსაც მონაცემთა ბაზის ობიექტებთან მიმართვის უფლებები ეწოდება.

მონაცემთა ბაზის მომხმარებლისთვის მონაცემთა ბაზის ყველა ობიექტთან მიმართვის უფლების მიცემა ან წართმევა შეიძლება მონაცემთა ბაზის ჩადგმული როლების საშუალებით.

მონაცემთა ბაზის ობიექტებთან მომხმარებლის მიმართვის უფლებების მართვისთვის გამოიყენება GRANT ბრძანება. მისი სინტაქსია:

GRANT { ALL | ნებართვა [ ,...n ]}{[( სვეტის_სახელი[ ,...n] ) ]
ON { ცხრილის_სახელი | წარმოდგენის_სახელი } | ON { ცხრილის_სახელი| წარმოდგენის_სახელი }[( სვეტის_სახელი [ ,...n ] ) ] | ON { შენახული_პროცედურის_სახელი } | ON
{მომხმარებლის_ფუნქციის_სახელი}}TO
უსაფრთხოების_სისტემის_ობიექტის_სახელი [ ,...n ] [ WITH GRANT OPTION ] [ AS { ჯგუფის_სახელი | როლის_სახელი } ]

სადაც,

- ALL გამოყენება მხოლოდ sysadmin როლის წევრებს შეუძლიათ.
- ნებართვა მისაწვდომი უფლებების სიაა, რომელიც მომხმარებელს ეძლევა (SELECT, INSERT, UPDATE,DELETE, EXECUTE).

- უსაფრთხოების_სისტემის_ობიექტის_სახელი

უსაფრთხოების სისტემის ობიექტის სახელია, რომელიც როლში უნდა იყოს ჩართული.

- ცხრილის_სახელი, წარმოდგენის_სახელი, სვეტის_სახელი, შენახული_პროცედურის_-სახელი, მომხმარებლის_ფუნქციის_სახელი არგუმენტები შეიცავენ მიმდინარე მონაცემთა ბაზის ობიექტების სახელებს, რომლებთანაც მიმართვა სრულდება.
- WITH GRANT OPTION არგუმენტი მომხმარებელს უფლებას აძლევს ობიექტთან მიმართვის უფლებები სხვა მომხმარებლებს დაუნიშნოს.
- AS { ჯგუფის_სახელი | როლის_სახელი } არგუმენტი მიუთითებს მომხმარებლის წევრობაზე იმ როლში, რომელსაც უფლება აქვს უფლებები სხვა მომხმარებლებს მისცეს.

დავალება 1. დავუშვათ, გვინდა რომ SELECT და INSERT ბრძანებების გამოყენების უფლება მივცეთ Roli_1 როლს Pers_1 ცხრილის მიმართ. ამასთან, საჭიროა, რომ მომავალში ამ ჯგუფის წევრებს ანალოგიური უფლებების სხვა მომხმარებლებისათვის გადაცემა შეეძლოთ. ამისათვის, უნდა შევასრულოთ შემდეგი ბრძანებები:

```
USE Shekveta
GO
GRANT SELECT, INSERT ON Pers_1 TO Roli_1 WITH GRANT OPTION
```

GO

შემდგომში, User_1 მომხმარებელს, რომელიც Roli_1 ჯგუფის წევრია, შეუძლია ანალოგიური უფლება მისცეს User_2 მომხმარებელს:

```
GRANT SELECT, INSERT ON Pers_1 TO User2 AS Roli_1
```

მოცემულ შემთხვევაში უნდა დავადასტუროთ თუ რის საფუძველზე გადასცა User_1 მომხმარებელმა უფლებები, ამიტომ გამოიყენება AS არგუმენტი.

დავალება 2. მაგალითი გვიჩვენებს ნებართვების მინიჭების მიმდევრობას. ჯერ public როლს მიენიჭება SELECT უფლება, შემდეგ კი INSERT, UPDATE, DELETE უფლებები მიენიჭა saba, User2, Login_1 და guest მომხმარებლებს. შედეგად, ამ მომხმარებლებს ექნებათ Pers_1 ცხრილთან მუშაობის ყველა უფლება.

```
USE Shekveta
GO
GRANT SELECT ON Pers_1 TO public
GO
GRANT INSERT, UPDATE, DELETE ON Pers_1 TO saba, User2, Login_1, guest
GO
```

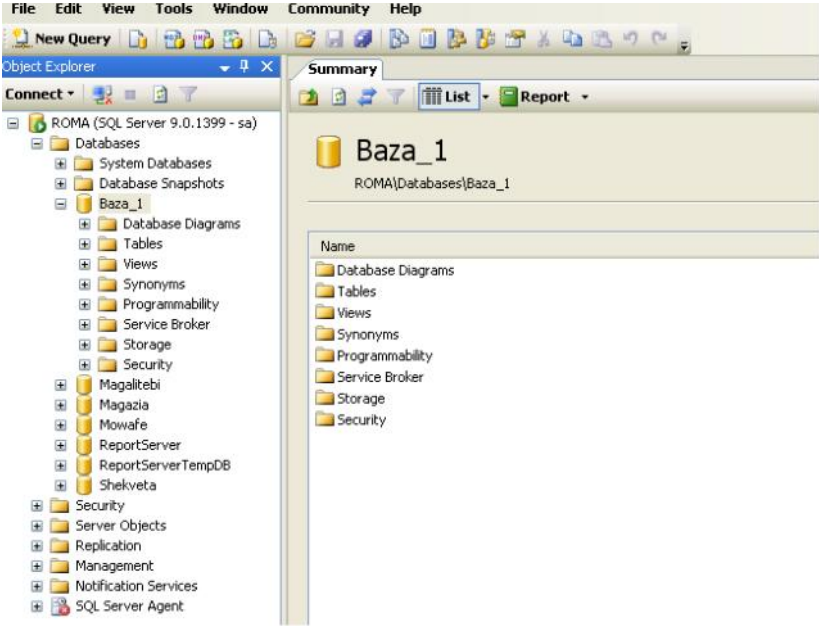
ყურადღებით უნდა ვიყოთ WITH GRANT OPTION არგუმენტის გამოყენებისას, რადგან ამ დროს ჩვენ ვკარგავთ

კონტროლს სხვა მომხმარებლებისთვის მიმართვის უფლების მიცემის პროცესზე. უმჯობესია შევზღუდოთ პირთა წრე, რომლებსაც აქვთ უფლებების გაცემის მართვის უფლება.

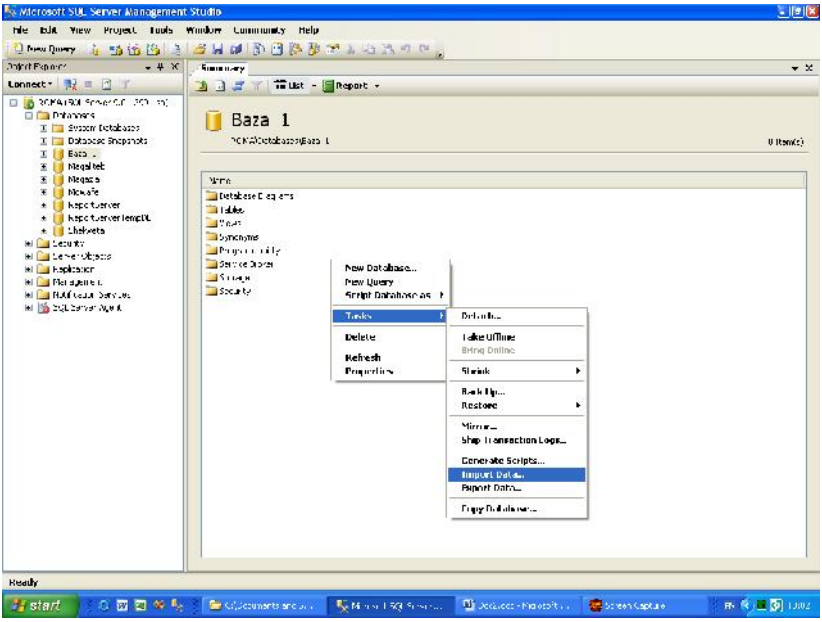
## ლაბორატორიული სამუშაო N15
## მონაცემების იმპორტი Access სისტემიდან

SQL სერვერის რომელიმე მონაცემთა ბაზაში, მაგალითად, Baza_1 მონაცემთა ბაზაში Access სისტემიდან მონაცემების იმპორტისათვის ჯერ უნდა გავააქტიუროთ ეს ბაზა. ამისათვის, საკმარისია მისი მონიშვნა (ნახ. 14.1). შემდეგ მარჯვენა ფანჯარაში ვხსნით კონტექსტურ მენიუს და ვასრულებთ Tasks ქვემენიუს Import Data ბრძანებას (ნახ. 14.2). გაიხსნება SQL Server Import and Export Wizard ფანჯარა (ნახ. 14.3). ვაჭერთ Next კლავიშს. გახსნილი ფანჯრის (ნახ. 14.4) Data source სიიდან ვირჩევთ Microsoft Access ელემენტს (ნახ. 14.5). Browse კლავიშის საშუალებით ვირჩევთ საჭირო ფაილს, მაგალითად db5.mdb და ვაჭერთ Next კლავიშს. მომდევნო ფანჯრაში ჩანს სერვერისა და მონაცემთა ბაზის სახელი, რომელშიც უნდა შესრულდეს მონაცემების იმპორტი (ნახ. 14.6). სურვილის შემთხვევაში New კლავიშის საშუალებით შეგვიძლია შევქმნათ ახალი ბაზა. ვირჩევთ აუტენტიფიცირების რეჟიმს და ვაჭერთ Next კლავიშს. გახსნილ ფანჯარაში (ნახ. 14.7) ვირჩევთ Copy data from one or more tables or views გადამრთველს და ვაჭერთ Next კლავიშს. გახსნილ ფანჯარაში (ნახ. 14.8) გამოჩნდება db5.mdb მონაცემთა ბაზის ობიექტები, კერძოდ Cxrili_1 ცხრილი. მოვნიშნოთ ეს ცხრილი და დავაჭიროთ Next კლავიშს. მომდევნო ფანჯარაში (ნახ. 14.9) ვაჭერთ Next კლავიშს. გახსნილ ფანჯარაში (ნახ. 14.10) ვაჭერთFinish კლავიშს. დაიწყება მონაცემების იმპორტი (ნახ. 14.11). ოპერაციის დამთავრების შემდეგ ვხურავთ ფანჯარას. იმპორტირებული

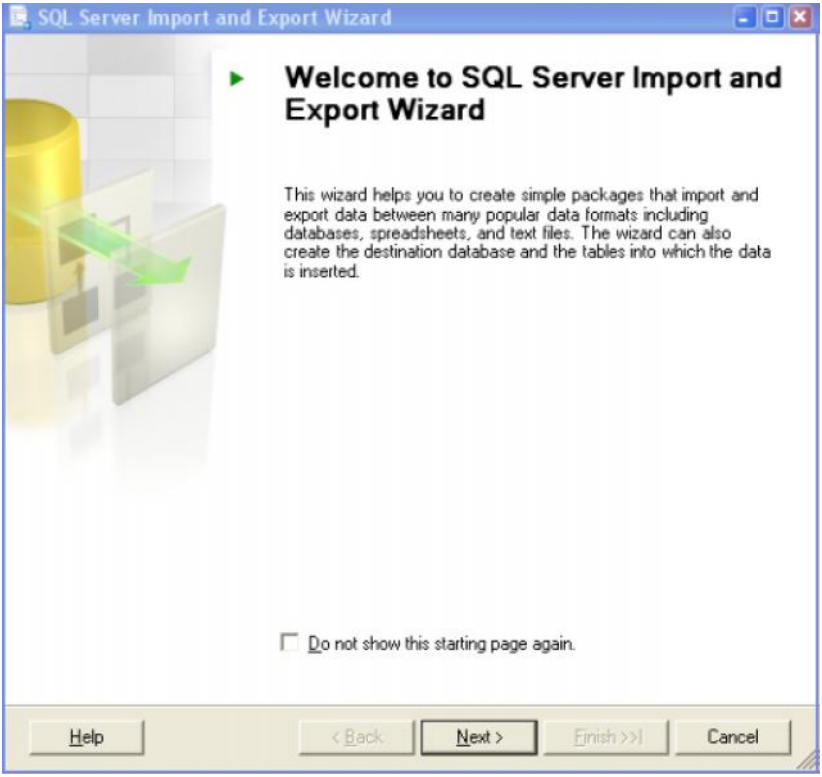
ცხრილის სანახავად უნდა მოვნიშნოთ Baza_1 მონაცემთა ბაზის Tables ელემენტი (ნახ. 14.1). მარჯვენა ფანჯარაში გამოჩნდება ცხრილების სახელები. მონაცემების სანახავად Cxrili_1 სახელზე ვხსნით კონტექსტურ მენიუს და ვასრულებთ Open Table ბრძანებას.


File Edit View Tools Window Community Help
New Query
Object Explorer
Connect
ROMA (SQL Server 9.0.1399 - sa)
Databases
System Databases
Database Snapshots
Baza_1
Database Diagrams
Tables
Views
Synonyms
Programmability
Service Broker
Storage
Security
Magalitebi
Magazia
Mowafe
ReportServer
ReportServerTempDB
Shekveta
Security
Server Objects
Replication
Management
Notification Services
SQL Server Agent
Summary
List
Report
Baza_1
ROMA\Databases\Baza_1
Name
Database Diagrams
Tables
Views
Synonyms
Programmability
Service Broker
Storage
Security

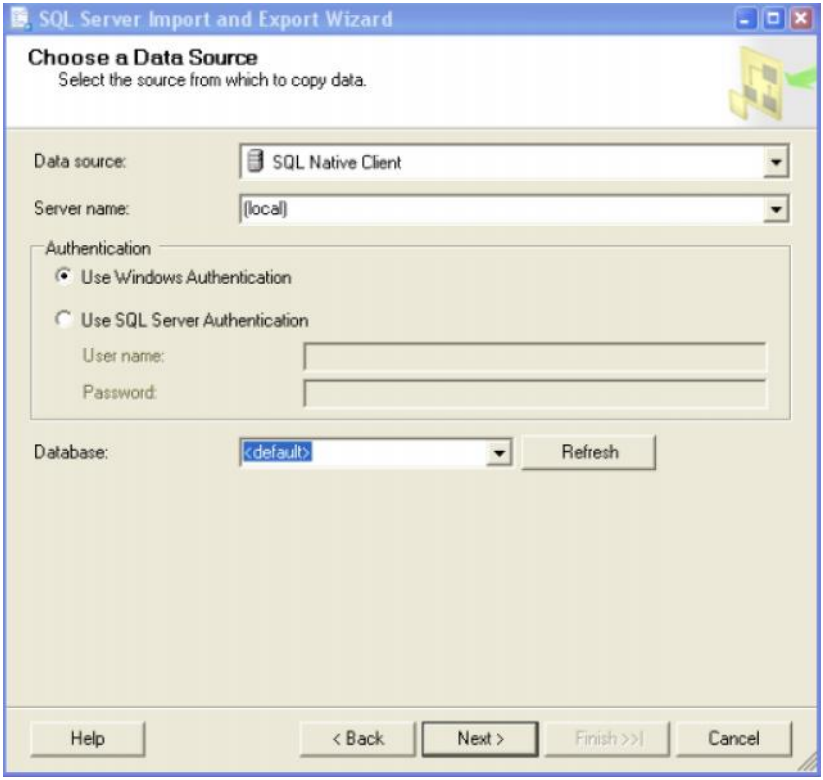
სურ. 1


Microsoft SQL Server Management Studio
File Edit View Project Tools Window Community Help
New Query
Object Explorer
Connect
Summary
List
Report
Baza 1
Name
New Database...
New Query
Script Database as
Tasks
Delete
Refresh
Properties
Take Offline
Bring Online
Shrink
Back Up...
Restore
Mirror...
Ship Transaction Logs...
Generate Scripts...
Import Data...
Export Data...
Ready
start

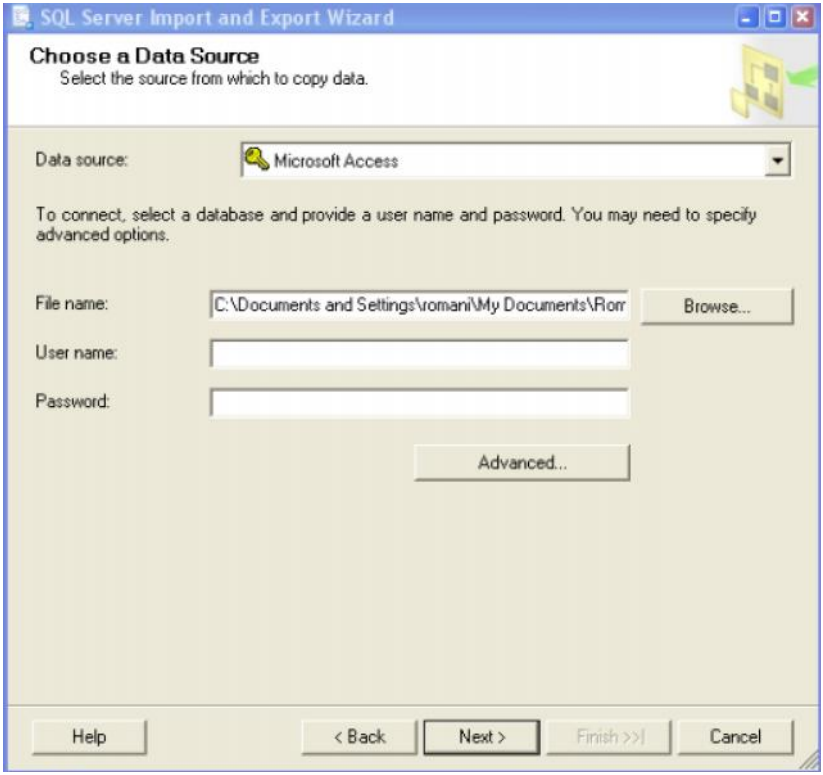
სურ. 2


SQL Server Import and Export Wizard
Welcome to SQL Server Import and Export Wizard
This wizard helps you to create simple packages that import and export data between many popular data formats including databases, spreadsheets, and text files. The wizard can also create the destination database and the tables into which the data is inserted.
Do not show this starting page again.
Help
< Back
Next >
Finish >>|
Cancel

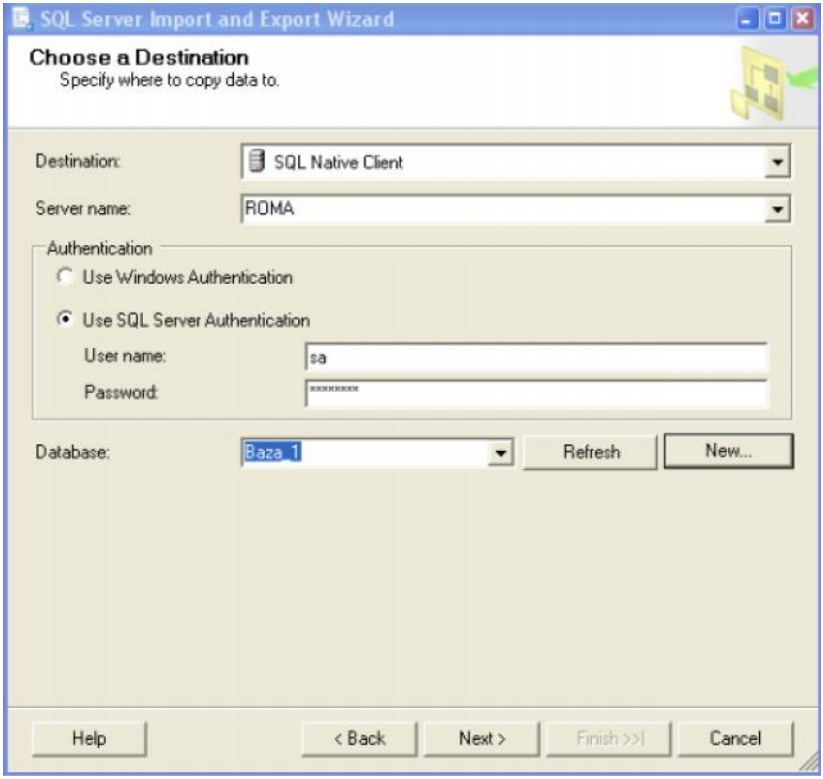
სურ. 3


SQL Server Import and Export Wizard
Choose a Data Source
Select the source from which to copy data.
Data source:
SQL Native Client
Server name:
(local)
Authentication
Use Windows Authentication
Use SQL Server Authentication
User name:
Password:
Database:
<default>
Refresh
Help
< Back
Next >
Finish >>|
Cancel

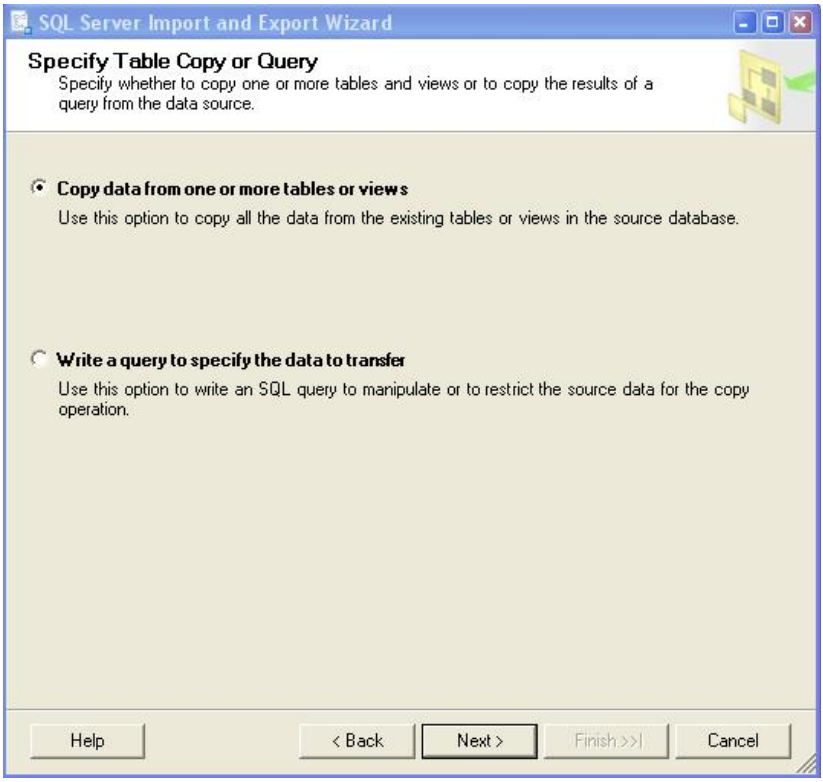
სურ. 4


SQL Server Import and Export Wizard
Choose a Data Source
Select the source from which to copy data.
Data source:
Microsoft Access
To connect, select a database and provide a user name and password. You may need to specify advanced options.
File name:
C:\Documents and Settings\romani\My Documents\Ron
Browse...
User name:
Password:
Advanced...
Help
< Back
Next >
Finish >>|
Cancel

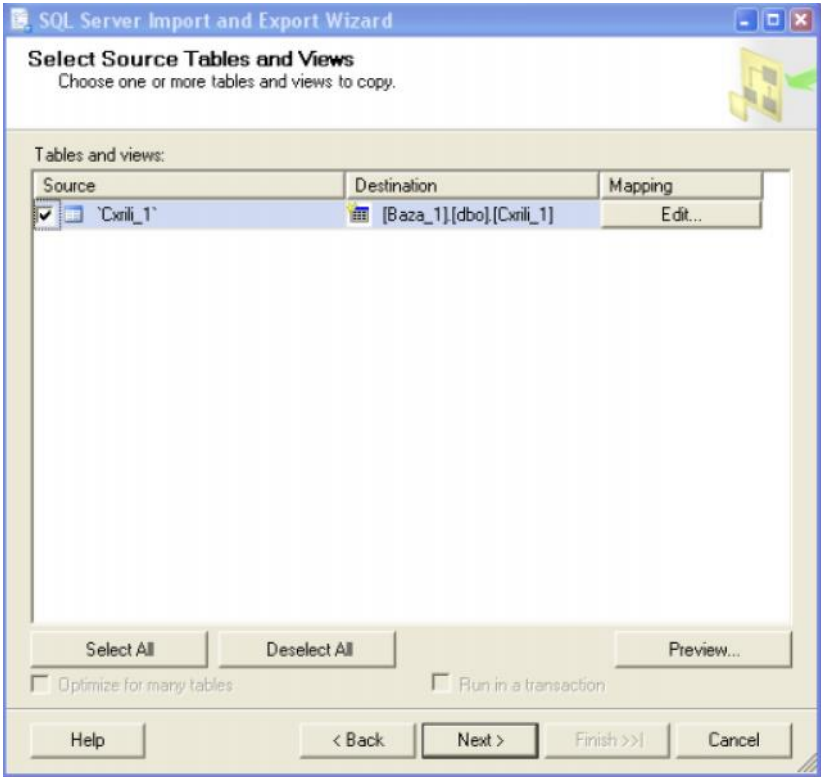
სურ. 5


SQL Server Import and Export Wizard
Choose a Destination
Specify where to copy data to.
Destination:
SQL Native Client
Server name:
ROMA
Authentication
Use Windows Authentication
Use SQL Server Authentication
User name:
sa
Password:
Database:
Baza_1
Refresh
New...
Help
< Back
Next >
Finish >>|
Cancel

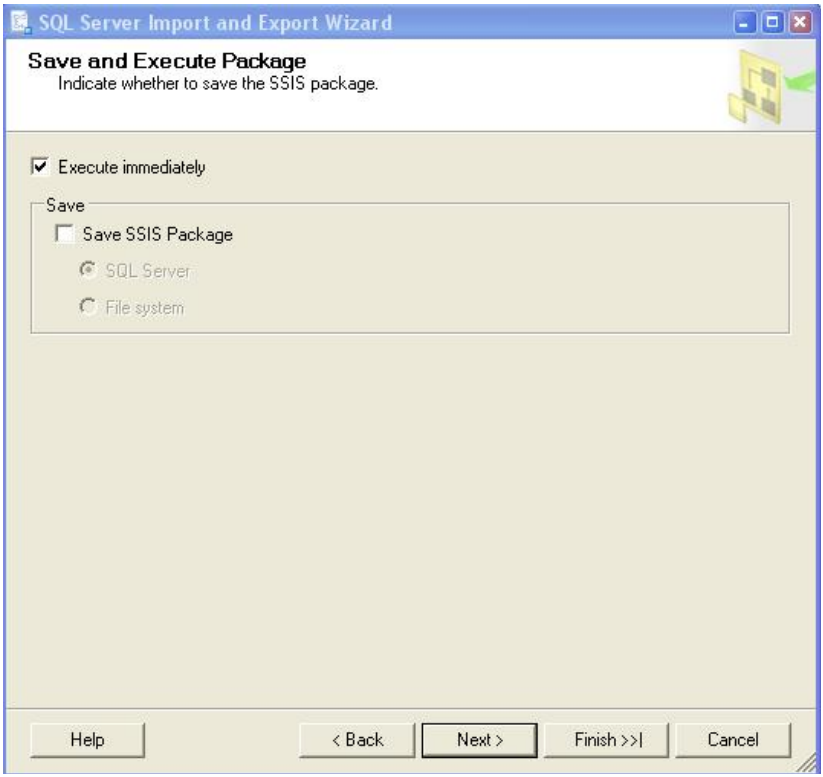
სურ. 6


SQL Server Import and Export Wizard
Specify Table Copy or Query
Specify whether to copy one or more tables and views or to copy the results of a query from the data source.
Copy data from one or more tables or views
Use this option to copy all the data from the existing tables or views in the source database.
Write a query to specify the data to transfer
Use this option to write an SQL query to manipulate or to restrict the source data for the copy operation.
Help
< Back
Next >
Finish >>|
Cancel

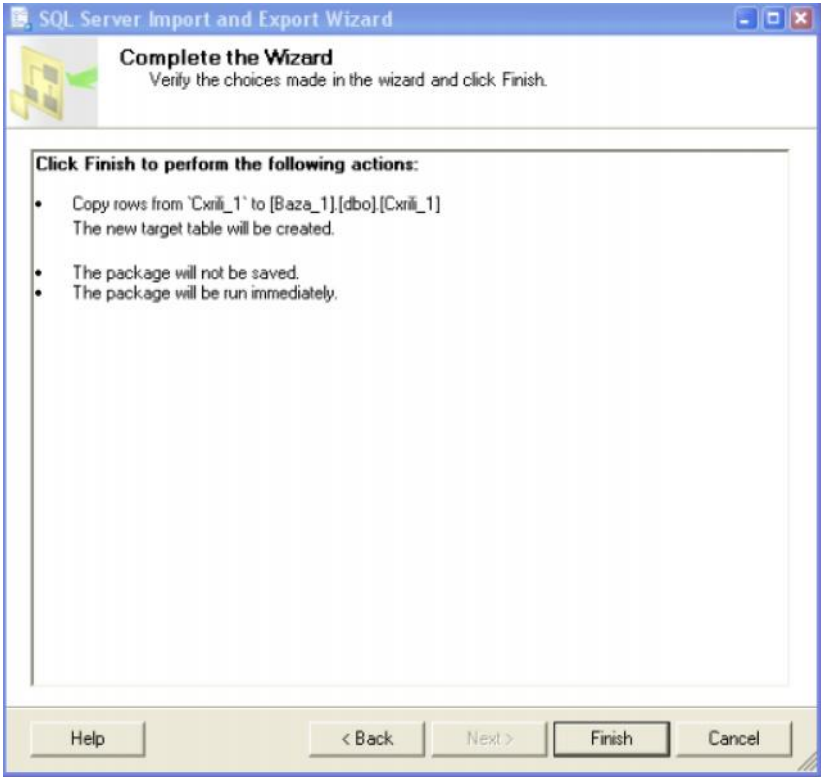
სურ. 7


SQL Server Import and Export Wizard
Select Source Tables and Views
Choose one or more tables and views to copy.
Tables and views:
Source
Destination
Mapping
`Cxrili_1`
[Baza_1].[dbo].[Cxrili_1]
Edit...
Select All
Deselect All
Preview...
Optimize for many tables
Run in a transaction
Help
< Back
Next >
Finish >>|
Cancel

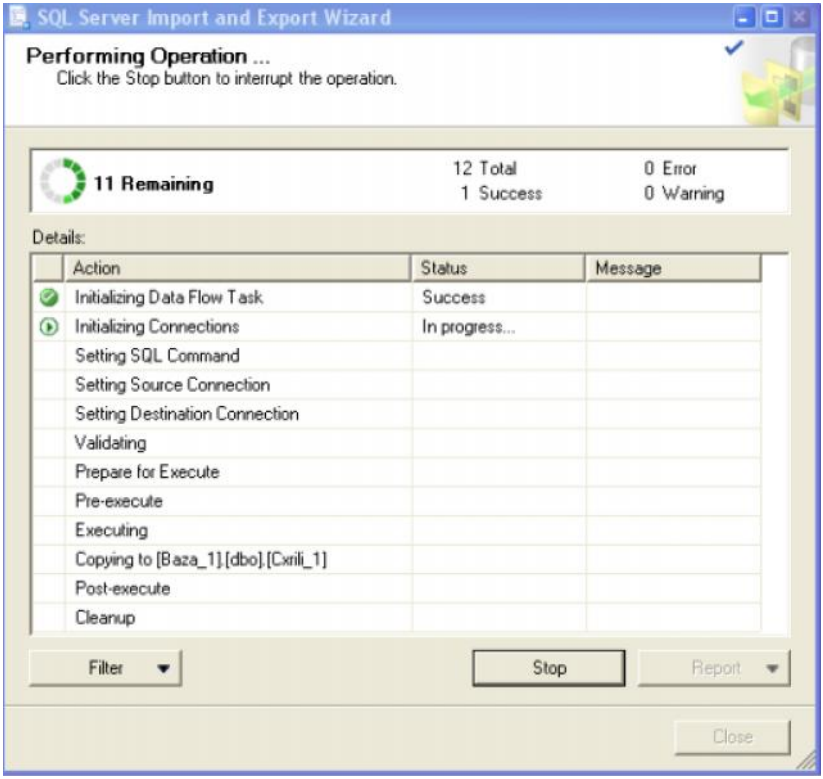
სურ. 8


SQL Server Import and Export Wizard
Save and Execute Package
Indicate whether to save the SSIS package.
Execute immediately
Save
Save SSIS Package
SQL Server
File system
Help
< Back
Next >
Finish >>|
Cancel

სურ. 9


SQL Server Import and Export Wizard
Complete the Wizard
Verify the choices made in the wizard and click Finish.
Click Finish to perform the following actions:
Copy rows from `Cxrili_1` to [Baza_1].[dbo].[Cxrili_1]
The new target table will be created.
The package will not be saved.
The package will be run immediately.
Help
< Back
Next >
Finish
Cancel

სურ. 10


SQL Server Import and Export Wizard
Performing Operation ...
Click the Stop button to interrupt the operation.
11 Remaining
12 Total
1 Success
0 Error
0 Warning
Details:
Action
Status
Message
Initializing Data Flow Task
Success
Initializing Connections
In progress...
Setting SQL Command
Setting Source Connection
Setting Destination Connection
Validating
Prepare for Execute
Pre-execute
Executing
Copying to [Baza_1].[dbo].[Cxrili_1]
Post-execute
Cleanup
Filter
Stop
Report
Close

სურ. 11

# ლიტერატურა


1. SQL სერვერი. რომან სამხარაძე. საგამომცემლო სახლი „ტექნიკური უნივერსიტეტი‖ , თბილისი.2009. ISBN 978-9941-14-190-4 . http://www.gtu.ge/publishinghouse

2. V i s u a l C#.N E T. რომან სამხარაძე. საგამომცემლო სახლი „ტექნიკური უნივერსიტეტი“, თბილისი.2009. ISBN978-9941-14-593. http://www.gtu.ge/publishinghouse

რედაქტორი
ტექნიკური რედაქტორი
კორექტორი
კომპიუტერული უზრუნველყოფა ნ. ჯოჯუასი

წარმოებას გადაეცა . ხელმოწერილია დასაბეჭდად .
ქაღალდის ზომა პირობითი ნაბეჭდი თაბახი . სააღრიცხვო-საგამომცემლო თაბახი . ტირაჟი 20 ეგზ.

გამომცემლობა “ტექნიკური უნივერსიტეტი”, თბილისი კოსტავას 77