საქართველოს ტექნიკური უნივერსიტეტი

რომან სამხარაძე, ლია გაჩეჩილაძე

SQL სერვერი

დამტკიცებულია სახელმძღვანელოდ
საქართველოს ტექნიკური უნივერსიტეტის
სარედაქციო-საგამომცემლო საბჭოს მიერ
01.01.2016, ოქმი № 1

თბილისი
2016

უაკ: 004.65

სახელმძღვანელო წარმოადგენს მეორე გადამუშავებულ გამოცემას. მასში გადმოცემულია Microsoft SQL Server გარემოში პროგრამების შემუშავების საკითხები. დაწვრილებითაა განხილული Transact SQL-ის საფუძვლები, მონაცემთა ბაზებისა და ცხრილების მართვის საკითხები, ფუნქციებთან, შენახულ პროცედურებთან, ინდექსებთან, წარმოდგენებთან, კურსორებთან და ტრიგერებთან მუშაობის პრინციპები. გადმოცემულია ადმინისტრირების საშუალებები. მეორე გამოცემაში დამატებულია ოპერაციები სიმრავლეებზე, საერთო ცხრილური გამოსახულებები, მონაცემების რეორგანიზების საკითხები. დაწვრილებითაა გაშუქებული ცხრილებს შორის კავშირები და ქვემოთხოვნები.

წიგნს უხვად ახლავს მაგალითები და სავარჯიშოები.

განკუთვნილია როგორც კომპიუტერული ინჟინერიის დეპარტამენტის ბაკალავრების, მაგისტრებისა და დოქტორანტებისათვის, ისე დაპროგრამების შესწავლის მსურველთათვის.

რეცენზენტები: ტექნიკის მეცნიერებათა დოქტორი, სრული პროფესორი გ. სურგულაძე
ტექნიკის მეცნიერებათა კანდიდატი, ასოცირებული პროფესორი გ. ჭიკაძე

ISBN 978-9941-20-661-0

http://www.gtu.ge/publishinghouse/

სარჩევი

წინასიტყვაობა 12

შესავალი 13

SQL სერვერთან მუშაობის დაწყება 13

სინტაქსის წაკითხვის წესები 16

Shekveta მონაცემთა ბაზის აღწერა 17

თავი 1. მონაცემთა ბაზების თეორიული საფუძვლები 18

საინფორმაციო სისტემები 18

მონაცემთა ბაზები 18

მონაცემების რელაციური მოდელი 19

მონაცემთა რელაციური ბაზები 20

ცხრილების ნორმალიზება 21
პირველი ნორმალური ფორმა 22
მეორე ნორმალური ფორმა 23
მესამე ნორმალური ფორმა 25

ცხრილებს შორის არსებული კავშირები 26
პირველადი და გარე გასაღებები 26
„ერთი-ერთთან" კავშირი 28
„ერთი-ბევრთან" კავშირი 30
„ბევრი-ბევრთან" კავშირი 31

თავი 2. მონაცემთა ბაზები 33

მონაცემთა ბაზების არქიტექტურა 33

მონაცემთა ბაზების ობიექტები 33
ცხრილები 33
სისტემური ცხრილები 34
დროებითი ცხრილები 34
მონაცემთა ტიპები 34
მომხმარებლების ტიპები 36
წარმოდგენები 36
შენახული პროცედურები 36
ტრიგერები 37
ინდექსები 37

მთლიანობის შეზღუდვები 37
ნაგულისხმევი მნიშვნელობები 39
ფუნქციები 39

სისტემური მონაცემთა ბაზები 39
master სისტემური მონაცემთა ბაზა 39
tempdb სისტემური მონაცემთა ბაზა 40
model სისტემური მონაცემთა ბაზა 40
msdb სისტემური მონაცემთა ბაზა 40

ფაილები და ფაილების ჯგუფები 40

მონაცემთა ბაზების შექმნა, წაშლა და სახელის შეცვლა 41
მონაცემთა ბაზის შექმნა 42
მონაცემთა ბაზის წაშლა 49
მონაცემთა ბაზის სახელის შეცვლა 50

მონაცემთა ბაზის მიერთება და გამორთვა 50
ინფორმაციის მიღება მონაცემთა ბაზების შესახებ 51

მომხმარებლის საკუთარი ტიპის შექმნა 51

თავი 3. ცხრილები 54

ცხრილების დაპროექტება 54
ცხრილის პირველადი გასაღები 54
ცხრილის გარე გასაღები 54
სვეტის უნიკალურობის განსაზღვრა 55
შემმოწმებელი შეზღუდვების განსაზღვრა 55
ნაგულისხმევი მნიშვნელობის განსაზღვრა 55
სვეტი-მთვლელის განსაზღვრა 56

ცხრილის შექმნა 56

ცხრილის წაშლა 64

ცხრილისთვის სახელის შეცვლა 64

ინფორმაციის მიღება ცხრილების შესახებ 64
ინფორმაციის მიღება ცხრილზე დამოკიდებული ობიექტების შესახებ 65
ინფორმაციის მიღება ცხრილებს შორის კავშირების შესახებ 66

დიაგრამები 67

Personali, Shemkveti და Xelshekruleba ცხრილები 69

თავი 4. მონაცემების მართვა 74

Transact SQL-ის საფუძვლები 74
გამოსახულებები 74
იდენტიფიკატორები 75
ცვლადები 77
ოპერატორები 79

ცხრილში მონაცემების ჩამატება 81
INSERT ბრძანება 82
SELECT ... INTO ბრძანება 86

ცხრილის მონაცემების ცვლილება 87
UPDATE ბრძანება 87

ცხრილიდან მონაცემების წაშლა 90
DELETE ბრძანება 90

ცხრილიდან მონაცემების ამორჩევა 91
SELECT ბრძანება 91
SELECT განყოფილება 91
განყოფილებების დამუშავების მიმდევრობა 96
FROM განყოფილება 97
WHERE განყოფილება 97
ლოგიკის ოპერატორები 101
BETWEEN ოპერატორი 101
IN ოპერატორი 101
LIKE ოპერატორი 102
აგრეგირების ფუნქციები 105
GROUP BY განყოფილება 108
HAVING განყოფილება 111
ORDER BY განყოფილება 112
COMPUTE განყოფილება 114
OVER ელემენტი 116

NULL მნიშვნელობასთან მუშაობა 119

ერთდროულად შესრულებადი ოპერაციები 122

მონაცემების მასობრივი გადაწერა 123

თავი 5. შეერთებები 126

FROM განყოფილება 126

ჯვარედინი შეერთებები 127
ჯვარედინი თვითშეერთებები 129

შიგა შეერთებები 130

შიგა შეერთების სახეები 131
შედგენილი შეერთებები 132
შეერთებები უტოლობის შემთხვევაში 132
მრავალცხრილიანი შეერთებები 133

გარე შეერთებები 136

თავი 6. მმართველი კონსტრუქციები 140

BEGIN...END 140

IF...ELSE 140

CASE...END 141

COALESCE 142

WHILE...BREAK & CONTINUE 144

თავი 7. ქვემოთხოვნები 146

დამოუკიდებელი ქვემოთხოვნები 146
დამოუკიდებელი სკალარული ქვემოთხოვნები 146
დამოუკიდებელი ქვემოთხოვნები მრავლობითი მნიშვნელობით 148

ბმული ქვემოთხოვნები 150

ლოგიკის ოპერატორები 152
ALL ოპერატორი 152
SOME და ANY ოპერატორები 153
EXISTS პრედიკატი 153

თავი 8. ოპერაციები სიმრავლეებზე 156

UNION ოპერაცია 157
UNION ALL ოპერაცია 157
UNION ოპერაცია 158

INTERSECT ოპერაცია 159

EXCEPT ოპერაცია 160

პრიორიტეტი 161

თავი 9. ფუნქციები 162

მომხმარებლის ფუნქციები 162
Scalar ტიპის ფუნქციები 162

Inline ტიპის ფუნქციები 167
Multi-statement ტიპის ფუნქციები 168

ფუნქციის წაშლა 170

ჩადგმული ფუნქციები 170
მათემატიკის ფუნქციები 170
სტრიქონებთან სამუშაო ფუნქციები 173
თარიღებთან სამუშაო ფუნქციები 176

თავი 10. შენახული პროცედურები 178

შესავალი 178

შენახული პროცედურების ტიპები 178

შენახული პროცედურის შექმნა 179

შენახული პროცედურების მართვა 187
შენახული პროცედურის შესახებ ინფორმაციის მიღება 187
შენახული პროცედურის სახელის შეცვლა 187
შენახული პროცედურის წაშლა 188
შენახული პროცედურის ავტომატურად შესრულების მართვა 188

შენახული პროცედურის გამოყენების მაგალითები 189

თავი 11. ინდექსები 190

შესავალი 190

ინდექსების გამოყენების დაგეგმვა 190
არაკლასტერული ინდექსი 191
კლასტერული ინდექსი 191
უნიკალური ინდექსი 192
შევსების ფაქტორი 192

ინდექსის შექმნა 193

ინდექსების მართვა 196
ინდექსისთვის სახელის შეცვლა 196
ინდექსის წაშლა 196
ინდექსების გადაწყობა 196
ინდექსის შესახებ ინფორმაციის მიღება 197

თავი 12. წარმოდგენები 199

შესავალი 199

წარმოდგენის შექმნა 200

წარმოდგენის მართვა 204
წარმოდგენის სახელის შეცვლა 204
წარმოდგენის წაშლა 204
წარმოდგენის შესახებ ინფორმაციის მიღება 204
წარმოდგენის დამოკიდებულებების ნახვა....205

თავი 13. კურსორები 206

შესავალი....206

კურსორების რეალიზება....206

კურსორის ტიპები....207
სტატიკური კურსორები....207
დინამიკური კურსორები 207
მიმდევრობითი კურსორები 207
საგასაღებო კურსორები....207

კურსორების მართვა....208
კურსორის შექმნა 208
კურსორის გახსნა....211
მონაცემების წაკითხვა....212
მონაცემების შეცვლა....214
მონაცემების წაშლა 215
კურსორის დახურვა 216
კურსორის გათავისუფლება....216

თავი 14. ტრანზაქციები და დაბლოკვები 217

შესავალი....217

ტრანზაქციების მართვა....217
აშკარა ტრანზაქციები 218
ავტომატური ტრანზაქციები....219
არააშკარა ტრანზაქციები 220
განაწილებული ტრანზაქციები 220
ჩადგმული ტრანზაქციები 221
ტრანზაქციებში აკრძალული Transact SQL-ის ბრძანებები....223

დაბლოკვის მართვა 223

თავი 15. ტრიგერები 226

შესავალი....226

ტრიგერის შექმნა 227

ტრიგერის წაშლა 233

ტრიგერების შექმნისას გასათვალისწინებელი რეკომენდაციები 234

ტრიგერების მართვა 234
ტრიგერის სახელის შეცვლა 234
ტრიგერის შესახებ ინფორმაციის მიღება 234

თავი 16. ცხრილური გამოსახულებები 237

წარმოებული ცხრილები 237
სვეტებისთვის ფსევდონიმების მინიჭება 238
არგუმენტების გამოყენება 239
ჩადგმულობა 240
მრავლობითი მიმართვები 241

საერთო ცხრილური გამოსახულებები 243
სვეტებისთვის ფსევდონიმების დანიშვნა 243
არგუმენტების გამოყენება 244
რამდენიმე საერთო ცხრილური გამოსახულების განსაზღვრა 245
მრავლობითი მიმართვები 246

APPLY ოპერაცია 246

ჩამნაცვლებელი ცხრილური ფუნქციები 249

თავი 17. მონაცემების რეორგანიზება და დაჯგუფების ნაკრებები 251

მონაცემების გაშლა 251
მონაცემების გაშლა სტანდარტული SQL ენის საშუალებით 253
მონაცემების გაშლა T-SQL ენის PIVOT ოპერაციის გამოყენებით 254

მონაცემების შეკვეცა 256
მონაცემების შეკვეცა სტანდარტული SQL ენის საშუალებით 256
შეკვეცა T-SQL-ის UNPIVOT ოპერაციის საშუალებით 258

დაჯგუფების ნაკრებები 259
GROUPING SETS ჩადგმული ელემენტი 262
CUBE ჩადგმული ელემენტი 263
ROLLUP ჩადგმული ელემენტი 263
GROUPING() და GROUPING_ID() ფუნქციები 264

თავი 18. მონაცემთა ბაზების სარეზერვო ასლები 267

სარეზერვო ასლის ტიპები 267

დამგროვებლის არჩევა 267
მოწყობილობის განსაზღვრა სარეზერვო ასლის მოსათავსებლად 268

მონაცემთა ბაზის სრული და დიფერენცირებული ასლები 269
მონაცემთა ბაზის სრული ასლი 269
მონაცემთა ბაზის დიფერენცირებული ასლი 270
სრული და დიფერენცირებული სარეზერვო ასლების შექმნა 270
სრული და დიფერენცირებული სარეზერვო ასლიდან აღდგენა 272

ტრანზაქციების ჟურნალის სარეზერვო ასლი 274
ტრანზაქციების ჟურნალის სარეზერვო ასლის შექმნა 276
ტრანზაქციების ჟურნალიდან აღდგენა 277

ფაილებისა და ფაილების ჯგუფის სარეზერვო ასლი 279
ფაილებისა და ფაილების ჯგუფების სარეზერვო ასლების შექმნა 280
ფაილებისა და ფაილების ჯგუფიდან აღდგენა 281

შეზღუდვები სარეზერვო ასლის შექმნის ოპერაციაზე 282

თავი 19. მონაცემების იმპორტი და ექსპორტი 283

მონაცემების იმპორტი და ექსპორტი 283

მონაცემების იმპორტი SQL სერვერზე 283
მონაცემების იმპორტი Access სისტემიდან 283
მონაცემების იმპორტი Excel სისტემიდან 284

მონაცემების ექსპორტი SQL სერვერიდან 284
მონაცემების ექსპორტი Access სისტემაში 284
მონაცემების ექსპორტი Excel სისტემაში 285

თავი 20. უსაფრთხოების სისტემა 295

უფლებამოსილების გამოჯვნის წესები 295

სერვერის უსაფრთხოების სისტემის არქიტექტურა 296
აუტენტიფიცირების რეჟიმები 296
Windows NT-ის აუტენტიფიცირების რეჟიმი 297
სერვერის აუტენტიფიცირების რეჟიმი 298

უსაფრთხოების სისტემის ელემენტები 299
სქემები 299
მომხმარებლები 301
სერვერის როლები 303
მონაცემთა ბაზების როლები 304
პროგრამა-დანართების როლები 306
როლის შეცვლა 307

უსაფრთხოების სისტემის ადმინისტრირება....308
სააღრიცხვო ჩანაწერების შექმნა და მართვა....308

მონაცემების დაცვა....309
მონაცემების დაშიფვრა....309
სერვერის ფაილებთან მიმართვის შეზღუდვა....309

მიმართვის უფლებები....310
მონაცემთა ბაზის ობიექტებთან მიმართვის უფლებები....310
Transact-SQL-ის ბრძანებების შესრულების უფლებები....313
არაცხადი უფლებები....314
მიმართვის აკრძალვა....314
მიმართვის არაცხადი უარყოფა....315
მიმართვის კონფლიქტები....316
უსაფრთხოების სისტემასთან სამუშაო პროცედურები და ბრძანებები....318

დანართი 1. სავარჯიშოები თავების მიხედვით....319

დანართი 2. სავარჯიშოების ამოხსნები....350

გამოყენებული ლიტერატურა....449

წინასიტყვაობა

როგორც ცნობილია, SQL სერვერი წარმოადგენს მონაცემთა ბაზების მართვის თანამედროვე და ფართოდ გავრცელებულ სისტემას, რომელიც წარმატებით გამოიყენება რთული საინფორმაციო სისტემების ასაგებად.

წიგნი ძირითადად განკუთვნილია დამწყები პროგრამისტებისათვის. მისი მიზანია მკითხველს შეასწავლოს Transact SQL ენაზე პროგრამების შედგენა და სერვერის ადმინისტრირება. ამიტომ, ძირითადი აქცენტი გადატანილია ამ საკითხების შესწავლაზე. პირველი გამოცემისგან განსხვავებით დამატებულია ოპერაციები სიმრავლეებზე, მონაცემების რეორგანიზების საკითხები და საერთო ცხრილური გამოსახულებები. დამატებულია სავარჯიშოები თავიანთი ამოხსნებით. წიგნში უხვადაა მაგალითები და სავარჯიშოები.

წარმოდგენილი სახელმძღვანელო არის SQL სერვერის მუშაობის პრინციპების ქართულ ენაზე გადმოცემის ერთ-ერთი პირველი მცდელობა, ამიტომ ის არ არის დაზღვეული ხარვეზებისაგან. ავტორი მადლიერებით მიიღებს წიგნში გადმოცემული მასალის დახვეწისა და სრულყოფის მიზნით გამოთქმულ შენიშვნებსა და მოსაზრებებს. ისინი შეგიძლიათ მომაწოდოთ მისამართებზე:
rsamkharadze@mail.ru, roman.samkharadze@yahoo.com, samkharadze.roman@gmail.com.

საქართველოს ტექნიკური უნივერსიტეტის
ინფორმატიკისა და მართვის სისტემების ფაკულტეტის
კომპიუტერული ინჟინერიის დეპარტამენტის
პროფესორი, ტექნიკის მეცნიერებათა დოქტორი

რომან სამხარაძე

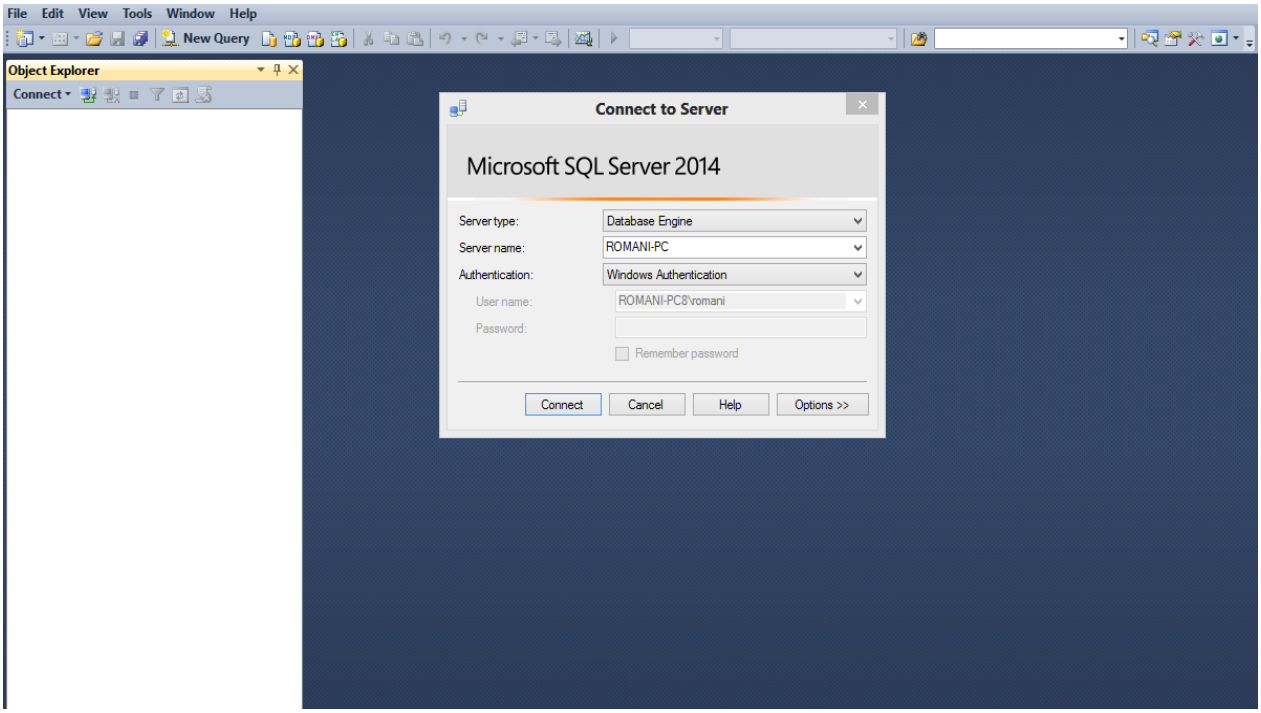
შესავალი

SQL სერვერთან მუშაობის დაწყება

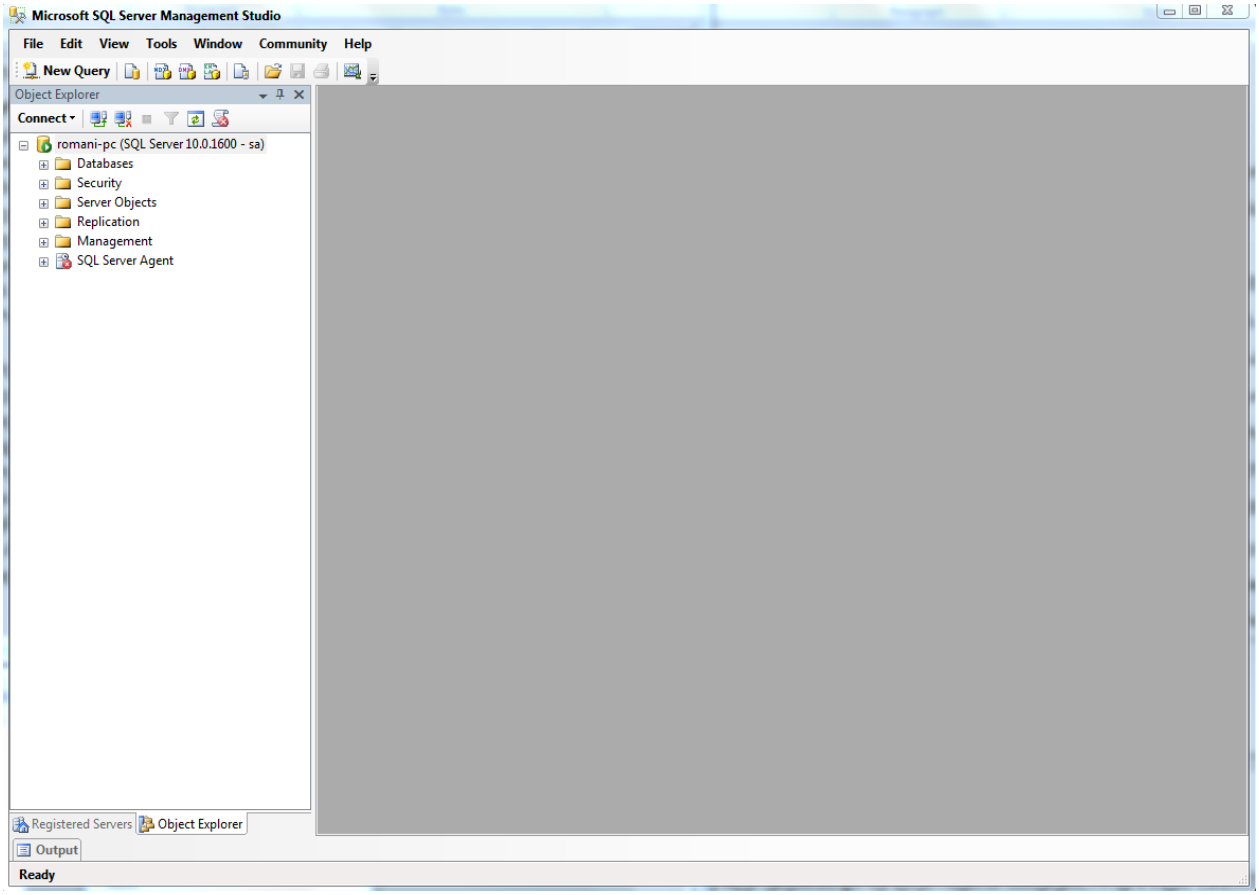
SQL სერვერი არის მონაცემთა ბაზების მართვის სისტემა. მასთან მუშაობის დასაწყებად ვასრულებთ Start→All Programs→Microsoft SQL Server 2014→SQL Server Business Intelligence Development Studio ბრძანებას. გაიხსნება Microsoft SQL Server Management Studio ფანჯარა (ნახ. 1). Server name ველში ჩანს სერვერის სახელი. საჭიროებისამებრ შეგვაქვს სხვა სერვერის სახელი. Authentication ველში ჩანს ვუინდოუსის აუტენტიფიცირების რეჟიმი - Windows Authentication. საჭიროებისამებრ ვირჩევთ SQL სერვერის აუტენტიფიცირების რეჟიმს - Microsoft SQL Server Authentication. Login ველში შეგვაქვს მომხმარებლის სახელი, Password ველში კი - პაროლი.

შემდეგ, ვაჭერთ Connect კლავიშს. გაიხსნება მომდევნო ფანჯარა (ნახ. 2). Object Explorer ფანჯარაში ჩანს ROMANI-PC სერვერის ობიექტები. თუ დავაჭერთ Databases კატალოგის მარცხნივ მოთავსებულ + სიმბოლოს, გაიხსნება მასში მოთავსებული მონაცემთა ბაზების სია (ნახ. 3). ასეთი გზით შეგვიძლია გამოვაჩინოთ ნებისმიერი მონაცემთა ბაზის ობიექტები, მაგალითად, Shekveta მონაცემთა ბაზის ობიექტები (ნახ. 4) და ამ ობიექტების ელემენტები.

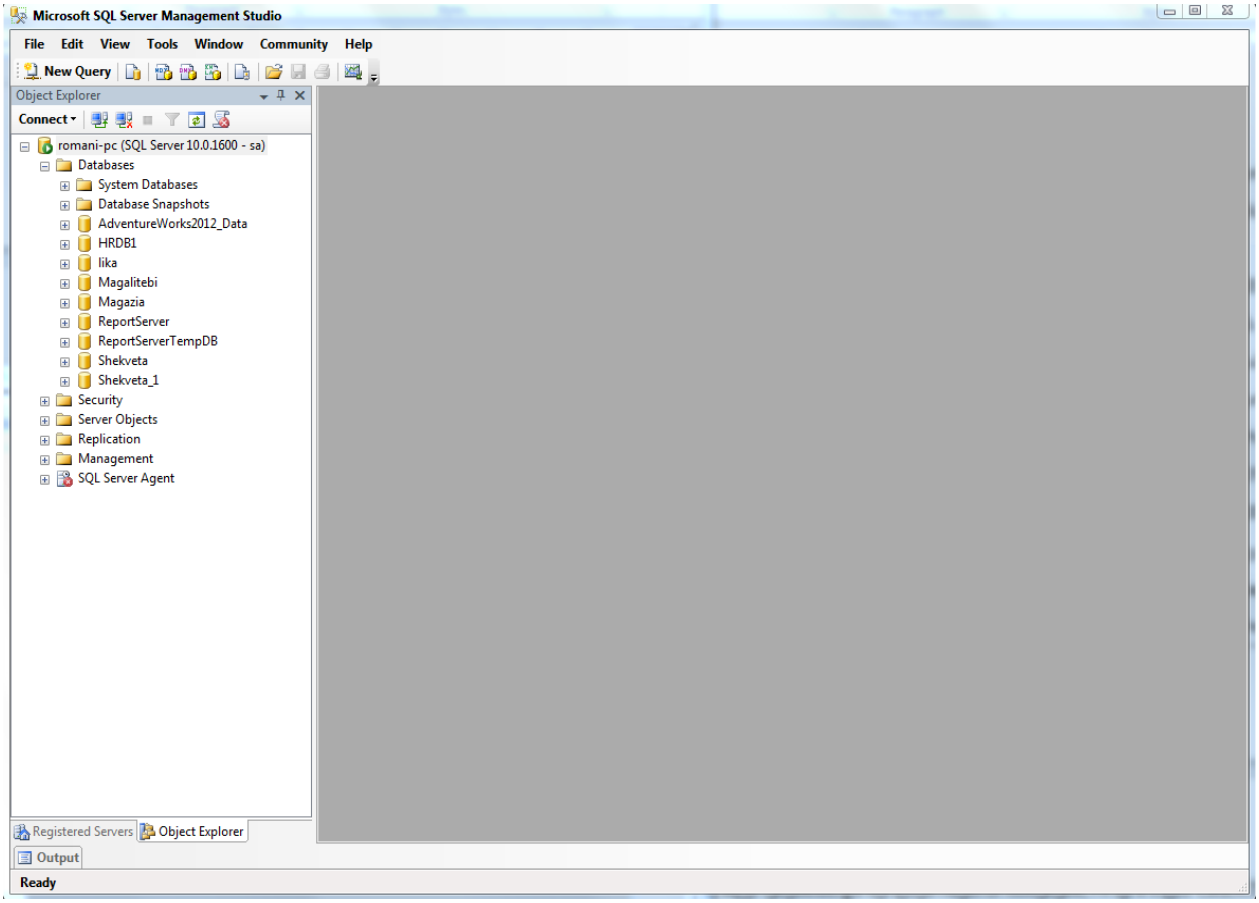
მოთხოვნის შესასრულებლად ინსტრუმენტების პანელზე უნდა დავაჭიროთ New Query კლავიშს. გაიხსნება ახალი განყოფილება - SQLQuery1.sql (ნახ. 5), რომელშიც უნდა შევიტანოთ მოთხოვნა. მოთხოვნის შესასრულებლად ვაჭერთ F5 კლავიშს.



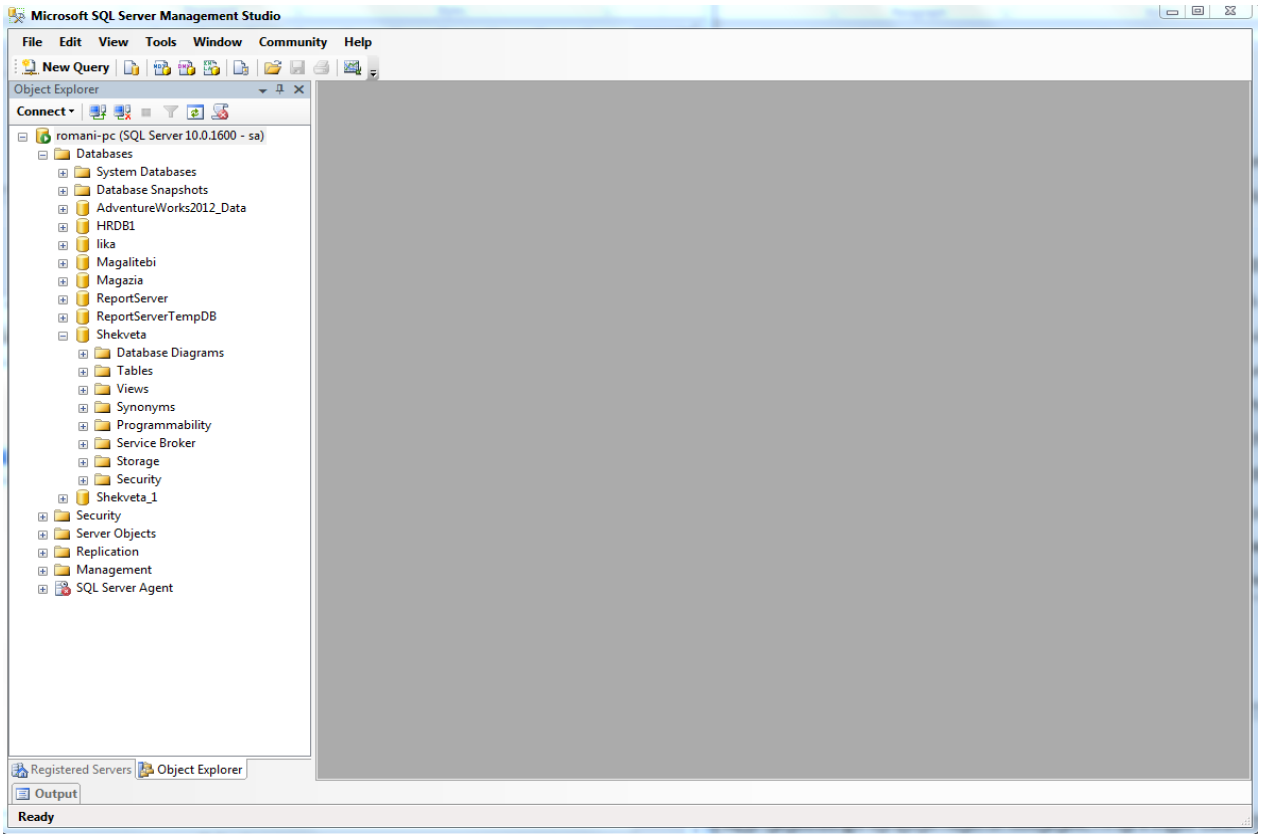
ნახ. 1.


Microsoft SQL Server Management Studio
File Edit View Tools Window Community Help
New Query
Object Explorer
Connect
romani-pc (SQL Server 10.0.1600 - sa)
Databases
Security
Server Objects
Replication
Management
SQL Server Agent
Registered Servers
Object Explorer
Output
Ready

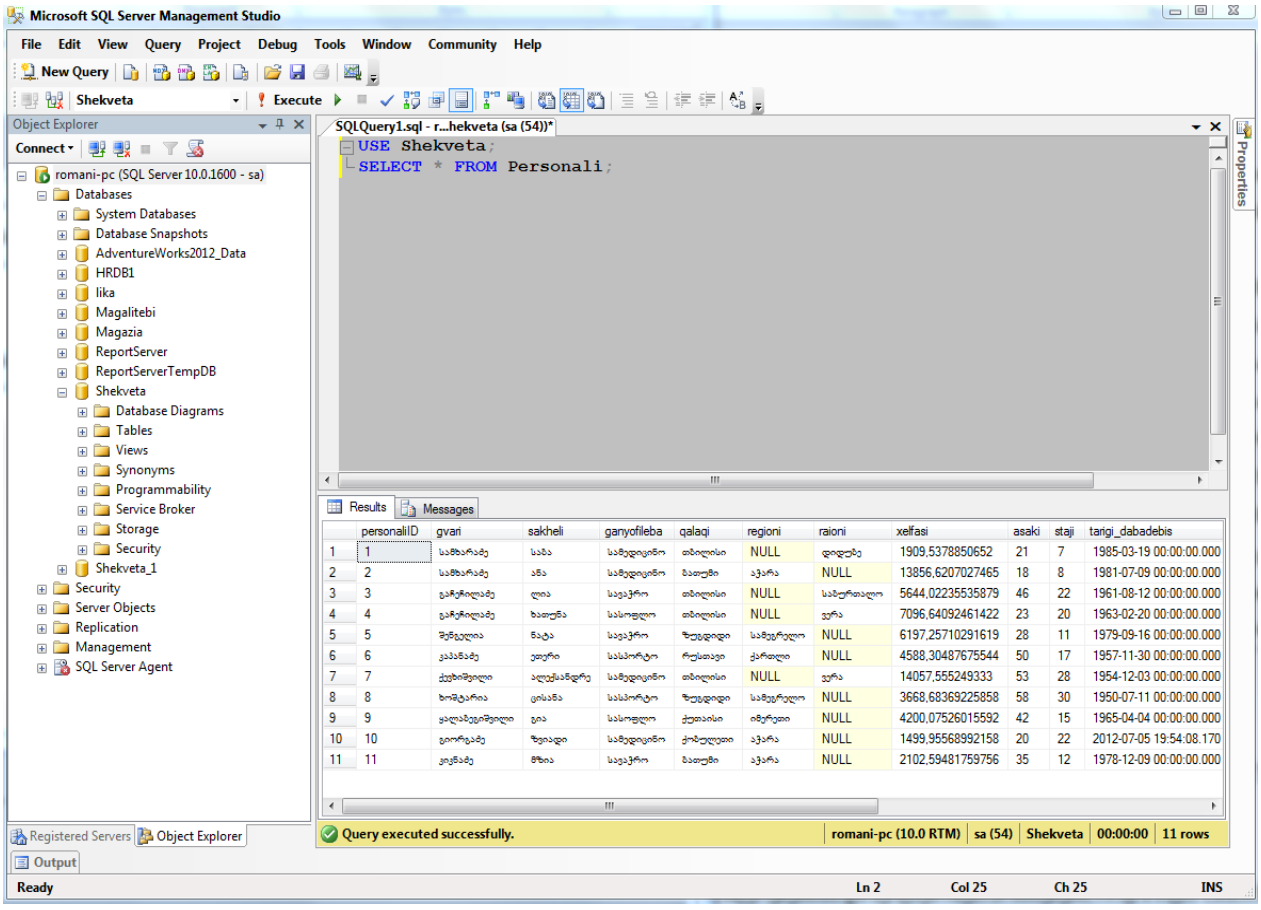
ნახ. 2.


Microsoft SQL Server Management Studio
File Edit View Tools Window Community Help
New Query
Object Explorer
Connect
romani-pc (SQL Server 10.0.1600 - sa)
Databases
System Databases
Database Snapshots
AdventureWorks2012_Data
HRDB1
lika
Magalitebi
Magazia
ReportServer
ReportServerTempDB
Shekveta
Shekveta_1
Security
Server Objects
Replication
Management
SQL Server Agent
Registered Servers
Object Explorer
Output
Ready

ნახ. 3.



ნახ. 4.



ნახ. 5.

სინტაქსის წაკითხვის წესები

T-SQL ენის თითოეულ ბრძანებას აქვს საკუთარი სინტაქსი ანუ სწორად ჩაწერის წესი. სინტაქსის ელემენტების შესწავლის მიზნით განვიხილოთ CREATE VIEW ბრძანება. ის წარმოდგენის შესაქმნელად გამოიყენება. 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 }

კვადრატულ ფრჩხილებში ("[" და "]") მოთავსებული არგუმენტი შეგვიძლია არ მივუთითოთ. მაგალითად, შეგვიძლია არ მივუთითოთ *სქემის_სახელი* ან WITH CHECK OPTION არგუმენტი და ა.შ.

კუთხურ ფრჩხილებში ("<" და ">") მოთავსებული არგუმენტი წარმოადგენს კონსტრუქციას, რომელსაც თავისი სინტაქსი აქვს. მოყვანილ მაგალითში ასეთია <*წარმოდგენის_ატრიბუტები*> კონსტრუქცია.

წარმოდგენის_სახელი აუცილებლად უნდა მივუთითოთ. თუ მივუთითებთ სვეტის სახელს, მაშინ ის მრგვალ ფრჩხილებში უნდა მოვათავსოთ.

[, . . . n] ნიშნავს, რომ შეგვიძლია მივუთითოთ არგუმენტის რამდენიმე მნიშვნელობა, რომლებიც ერთმანეთისაგან მძიმეებით იქნება გამოყოფილი. მაგალითად, შეგვიძლია მივუთითოთ რამდენიმე სვეტის სახელი - (gvari, asaki, staji).

ფიგურულ ფრჩხილებში ({ და }) მოთავსებული არგუმენტებიდან ერთ-ერთი უნდა ავირჩიოთ. | ნიშნავს "ან" ოპერატორს. მოყვანილი მაგალითიდან გამომდინარე უნდა ავირჩიოთ ENCRYPTION, SCHEMABINDING ან VIEW_METADATA არგუმენტი.

ზემოთ თქმულიდან გამომდინარე CREATE VIEW ბრძანება რამდენიმე გზით შეგვიძლია ჩავწეროთ:

1.
```
CREATE VIEW Warmodgena_1
WITH ENCRYPTION
AS
SELECT * FROM Personali
```
2.
```
CREATE VIEW Sqema_1.Warmodgena_2
WITH ENCRYPTION
AS
SELECT * FROM Personali
WITH CHECK OPTION
```
3.
```
CREATE VIEW Warmodgena_3
AS
SELECT * FROM Personali
```
4.
```
CREATE VIEW Sqema_1.Warmodgena_1
WITH ENCRYPTION, VIEW_METADATA
AS
SELECT * FROM Personali
```

განვიხილოთ კიდევ ერთი მაგალითი. მონაცემთა ბაზის მომხმარებელის ჩასართავად მიმდინარე მონაცემთა ბაზის როლში გამოიყენება sp_addrolemember შენახული პროცედურა. მისი სინტაქსია:

sp_addrolemember [@rolename =] 'როლის_სახელი',
[@membername =] 'უსაფრთხოების_ობიექტის_სახელი'

აქ @rolename არის ცვლადი, რომელიც შეიცავს როლის სახელს. სინტაქსიდან გამომდინარე sp_addrolemember ბრძანება რამდენიმე გზით შეგვიძლია ჩავწეროთ:

1. USE Baza_8;
   EXEC sp_addrolemember 'db_owner', 'U_81';
2. USE Baza_8;
   DECLARE @db_owner NVARCHAR(20), @U_81 NVARCHAR(20);
   SET @db_owner = N'db_owner';
   SET @U_81 = N'U_81';
   EXEC sp_addrolemember @db_owner, @U_81;

Shekveta მონაცემთა ბაზის აღწერა

მთელს წიგნში განხილული იქნება ვირტუალური საპროექტო ფირმის მაგალითი, რომელსაც საქართველოს სხვადასხვა ქალაქში ფილიალი აქვს გახსნილი. ფირმა იღებს სხვადასხვა ტიპის ობიექტის დაპროექტების შეკვეთებს, აფორმებს ხელშეკრულებას შემკვეთთან და ასრულებს შესაბამის საპროექტო სამუშაოებს. ფირმას შემდეგი განყოფილებები აქვს: სამედიცინო, სასპორტო, სასოფლო, სავაჭრო, ენერგეტიკის და ა.შ. თითოეულ განყოფილებაში რამდენიმე ადამიანი მუშაობს. ისინი იღებენ და ასრულებენ შეკვეთებს. მაგალითად, სასოფლო განყოფილების თანამშრომლები ახდენენ ისეთი ობიექტების დაპროექტებას, როგორიცაა მეფრინველეობის ფერმა, ხილის გადამამუშავებელი ფაბრიკა, ღვინის ქარხანა და ა.შ. სამედიცინო განყოფილების თანამშრომლები აპროექტებენ საავადმყოფოებს, პოლიკლინიკებს და ა.შ. სასპორტო განყოფილებაში აპროექტებენ სპორტულ მოედნებს, კომპლექსებს და ა.შ. სავაჭრო განყოფილებაში სრულდება სუპერმარკეტების, მინიმარკეტების, ბაზრებისა და ა.შ. დაპროექტება.

ჩვენი ვირტუალური საპროექტო ფირმის თანამშრომლებს შემდეგი პარამეტრები აქვთ: სახელი და გვარი; დაბადების წელი; ასაკი; მუშაობის სტაჟი; განყოფილება, რომელშიც ის მუშაობს; ქალაქი, რაიონი, ქუჩის დასახელება და ბინის ნომერი, სადაც ის ცხოვრობს; სახლის, სამსახურისა და მობილური ტელეფონის ნომრები; ელ. ფოსტა და სურათი. ჩვენ, ამ მონაცემებს Personali ცხრილში მოვათავსებთ.

შემკვეთს შემდეგი პარამეტრები აქვს: სახელი და გვარი; იურიდიული თუ ფიზიკური პირია; ფირმის დასახელება; ქალაქი, რაიონი, ქუჩის დასახელება და სახლის ნომერი, სადაც ის მდებარეობს ან ცხოვრობს (თუ ფიზიკური პირია); ბინის, სამსახურისა და მობილური ტელეფონის ნომრები; ფირმის წარმომადგენელი და მისი ტელეფონი; პასპორტის ნომერი ფიზიკური პირისთვის; ელ. ფოსტა და საბანკო ანგარიში. ჩვენ, ამ მონაცემებს Shemkveti ცხრილში მოვათავსებთ.

თითოეული თანამშრომელი შეკვეთას იღებს როგორც იურიდიული, ისე ფიზიკური პირებისაგან. მათ შორის ფორმდება ხელშეკრულება, რომელსაც შემდეგი პარამეტრები აქვს: ხელშეკრულების დაწყებისა და დამთავრების ვადები, გადასახდელი თანხა, მხარეთა რეკვიზიტები. ჩვენ, ამ მონაცემებს Xelshekruleba ცხრილში მოვათავსებთ. ამავე ცხრილში დამატებით მოვათავსებთ შემდეგ ინფორმაციას: ხელშეკრულება შესრულდა თუ არა; ვისი მიზეზით არ შესრულდა; ვალუტის კურსი; გადასახდელი თანხა ლარებსა და დოლარებში; რეალურად გადახდილი თანხა ლარებსა და დოლარებში; ვალი ლარებსა და დოლარებში.

აღნიშნული ცხრილები Shekveta მონაცემთა ბაზაშია მოთავსებული. ცხადია, რომ ამ ცხრილებში მოთავსებული მონაცემები ვირტუალურია.

თავი 1. მონაცემთა ბაზების თეორიული საფუძვლები

საინფორმაციო სისტემები

საინფორმაციო სისტემა არის სტრუქტურირებული (მოწესრიგებული) მონაცემებისა და აპარატურულ-პროგრამული საშუალებების ერთობლიობა, განკუთვნილი მონაცემების შენახვისა და დამუშავებისათვის. საინფორმაციო სისტემებია: საბანკო, ბილეთების დაჯავშნის, თელასის აბონენტების აღრიცხვის, უნივერსიტეტის სტუდენტების აღრიცხვის და ა.შ.

არსებობს საინფორმაციო სისტემების ორი კლასი: საინფორმაციო-საძებნი და მონაცემების დამუშავების. პირველი მათგანი ორიენტირებულია ჩვენთვის საჭირო მონაცემების ძებნასა და მიღებაზე, მეორე კი - მონაცემების დამუშავებასა და დამუშავებული ინფორმაციის მიღებაზე.

საინფორმაციო სისტემა შემდეგ ფუნქციებს ასრულებს:

- ინფორმაციის შეტანა, ცვლილება და შენახვა;
- ინფორმაციის ნახვა და ძებნა;
- ინფორმაციის ამორჩევა სათანადო კრიტერიუმის მიხედვით;
- ანგარიშების ფორმირება;
- ინფორმაციის სისწორის შემოწმება.

მონაცემთა ბაზები

ფართო აზრით, მონაცემთა ბაზა არის სათანადო საგნობრივი სფეროს ობიექტების შესახებ მონაცემების მოწესრიგებული ერთობლიობა. საგნობრივი სფერო შეიძლება იყოს უნივერსიტეტი, საავადმყოფო, ფირმა და ა.შ., ობიექტი კი შესაბამისად, სტუდენტი, ავადმყოფი, თანამშრომელი და ა.შ. მონაცემთა ბაზასთან მუშაობა შეიძლება მონაცემთა ბაზის მართვის სისტემის საშუალებით. ეს არის პროგრამული საშუალებების ერთობლიობა, რომელიც საჭიროა მონაცემთა ბაზის შესაქმნელად და მასში მოთავსებულ ობიექტებზე მანიპულირებისათვის.

არსებობს მონაცემთა განაწილებული და ცენტრალიზებული ბაზები. მონაცემთა განაწილებული ბაზა რამდენიმე ნაწილისაგან შედგება, რომლებიც მოთავსებულია კომპიუტერული ქსელის სხვადასხვა კომპიუტერზე (სერვერზე). მონაცემთა ცენტრალიზებული ბაზა მოთავსებულია ერთ კომპიუტერზე, რომელიც შეიძლება იყოს ლოკალურ ქსელში ჩართული და შესაბამისად, შესაძლებელი იქნება სხვა კომპიუტერების მხრიდან ამ ბაზასთან მიმართვა. მონაცემთა ცენტრალიზებული ბაზა არქიტექტურის მიხედვით ორგვარია: ფაილ-სერვერი და კლიენტ-სერვერი.

ფაილ-სერვერული არქიტექტურის შემთხვევაში მონაცემთა ბაზა განთავსებულია ერთ ან მეტ ფაილ-სერვერზე, რომელიც წარმოადგენს ქსელში ჩართულ მძლავრ კომპიუტერს. მონაცემების დამუშავება სრულდება ლოკალურ კომპიუტერებზე. ფაილ-სერვერული მონაცემთა ბაზების მართვის სისტემებია: Microsoft Visual FoxPro, Microsoft Access, Paradox for Windows, dBase for Windows და ა.შ.

კლიენტ-სერვერული არქიტექტურის შემთხვევაში მონაცემთა ბაზა განთავსებულია სერვერზე და აქვე ხდება მისი დამუშავება. კლიენტის მხრიდან მონაცემების დამუშავების მოთხოვნა სერვერს ეგზავნება. მასზე სრულდება მონაცემების დამუშავება და შედეგები კლიენტს ეგზავნება. სისტემა, რომელიც იყენებს კლიენტ-სერვერულ ტექნოლოგიას ორი ნაწილისაგან შედგება: კლიენტის ნაწილი (front-end) და სერვერის ნაწილი (back-end). კლიენტის ნაწილი უზრუნველყოფს გრაფიკულ ინტერფეისს და იმყოფება მომხმარებლის

კომპიუტერზე. სერვერის ნაწილი მოთავსებულია სერვერზე და უზრუნველყოფს მონაცემების მართვას, დანაწილებას, ადმინისტრირებასა და უსაფრთხოებას. კლიენტ-სერვერული მონაცემთა ბაზების მართვის სისტემებია: Microsoft SQL Server, Oracle, IBM DB2, Sybase და ა.შ. ამ არქიტექტურისათვის დამახასიათებელია მოთხოვნების სტრუქტურირებული ენის (SQL, Structured Query Language) გამოყენება.

მონაცემების რელაციური მოდელი

მონაცემთა ბაზების კლასიფიცირება შეიძლება მოვახდინოთ მონაცემების მოდელების მიხედვით. მონაცემების მოდელი არის მონაცემების სტრუქტურისა და მათი დამუშავების ოპერაციების ერთობლიობა. მონაცემების მოდელის მიხედვით შესაძლებელია ობიექტების სტრუქტურისა და მათ შორის არსებული კავშირების წარმოდგენა. არსებობს მონაცემების იერარქიული, ქსელური და რელაციური მოდელები. შესაბამისად, არსებობს იერარქიული, ქსელური და რელაციური მონაცემთა ბაზები. ჩვენ განვიხილავთ მონაცემების რელაციურ მოდელს, რადგან ის ხშირად გამოიყენება.

მონაცემების იერარქიული მოდელის თითოეული ელემენტი სხვა ელემენტებს წარმოქმნის. წარმოქმნილი ელემენტები, თავის მხრივ, სხვა ელემენტებს წარმოქმნის და ა.შ. ამასთანავე, ნებისმიერ წარმოქმნილ ელემენტს აქვს მხოლოდ ერთი წარმომქმნელი ელემენტი. იერარქიული სტრუქტურის მაგალითია უნივერსიტეტი, საავადმყოფო და ა.შ. მონაცემების ასეთი ორგანიზების ნაკლია ის, რომ შეუძლებელია შემუშავებული იერარქიის შეცვლა და მოგვიწევს მხოლოდ იმ იერარქიის გამოყენება, რომელიც შექმნილი იყო მონაცემთა ბაზის დაპროექტების დროს. მონაცემების იერარქიის ცვლილების აუცილებლობამ გამოიწვია უფრო ზოგადი - ქსელური მოდელის შექმნა.

მონაცემების ქსელური მოდელი წარმოადგენს მონაცემების იერარქიული მოდელის გაფართოებას. იერარქიული მოდელისაგან განსხვავებით ქსელური მოდელის წარმოქმნილ ელემენტს შეიძლება ჰქონდეს ერთზე მეტი წარმომქმნელი ელემენტი. ქსელური სტრუქტურის მაგალითია მაღაზია, სუპერმარკეტი და ა.შ. მაღაზიის გამყიდველი ემსახურება რამდენიმე მომხმარებელს. ამასთან, თითოეულ მომხმარებელს შეიძლება მოემსახუროს რამდენიმე გამყიდველი. ქსელურ მოდელს აქვს ნაკლი - იმისათვის, რომ მონაცემები დავამუშავოთ, უნდა ვიცოდეთ მონაცემთა ბაზის სტრუქტურა.

მონაცემთა რელაციური მოდელის შემთხვევაში მონაცემების წარმოდგენა ხდება ორგანზომილებიანი ცხრილის სახით. მარტივ შემთხვევაში, რელაციური მოდელი აღწერს ერთ ორგანზომილებიან ცხრილს, უფრო ხშირად კი - რამდენიმე ცხრილის სტრუქტურასა და მათ შორის კავშირებს. რელაციური მოდელი ეფუძნება მათემატიკის ორ განყოფილებას: სიმრავლეთა თეორიასა და ლოგიკას (პრედიკატების აღრიცხვას).

ცხრილი 1.1. მონაცემთა ბაზების ტერმინოლოგია.

მონაცემთა ბაზების თეორია	მონაცემთა რელაციური ბაზები	SQL სერვერი
მიმართება (Relation)	ცხრილი (Table)	ცხრილი (Table)
კორტეჟი (Tuple)	ჩანაწერი (Record)	სტრიქონი (Row)
ატრიბუტი (Attribute)	ველი (Field)	სვეტი (Column)

მონაცემთა რელაციური ბაზის აღწერისას ერთი და იმავე ცნებისათვის სხვადასხვა ტერმინები გამოიყენება. ეს დამოკიდებულია აღწერის დონესა (თეორია თუ პრაქტიკა) და სისტემაზე (Access, SQL Server, dBase). ცხრილში 1.1 მოყვანილია მონაცემთა ბაზების

ტერმინოლოგია.

მონაცემთა რელაციური ბაზების თეორია წინა საუკუნის სამოცდაათიან წლებში შეიმუშავა ამერიკელმა მათემატიკოსმა ე. კოდმა. მან მონაცემების დამუშავებისათვის შემოგვთავაზა სიმრავლეთა თეორიის აპარატი და დაამტკიცა, რომ მონაცემთა ნებისმიერი ნაკრები შეიძლება ორგანზომილებიანი ცხრილის სახით წარმოვადგინოთ. სიტყვა „რელაციური" წარმოდგება ინგლისური სიტყვისაგან „relation", რაც მიმართებას ნიშნავს. მიმართებების წარმოდგენა მოხერხებულია ორგანზომილებიანი ცხრილების სახით. ამრიგად, რელაციური არის მონაცემთა ისეთი ბაზა, რომელშიც მონაცემები წარმოდგენილია სწორკუთხა ორგანზომილებიანი ცხრილებით.

ცხრილს აქვს სახელი, რომელიც უნიკალურია მონაცემთა ბაზის შიგნით. ის შედგება *სვეტებისა* (ველებისა) და *სტრიქონებისაგან* (ჩანაწერებისაგან). ცხრილი შეიცავს ინფორმაციას ერთტიპური ობიექტების შესახებ, მისი *სტრიქონი* კი - ინფორმაციას კონკრეტული ობიექტის შესახებ. მაგალითად, Palata ცხრილი (ცხრილი 1.2) შეიცავს ინფორმაციას მასში მწოლიარე ავადმყოფების შესახებ, ხოლო მისი თითოეული სტრიქონი კი - ინფორმაციას კონკრეტული ავადმყოფის შესახებ ანუ კონკრეტული ავადმყოფის ატრიბუტების მნიშვნელობების ნაკრებს. ცხრილის თითოეული სვეტი წარმოადგენს ობიექტების კონკრეტული ატრიბუტის მნიშვნელობების ერთობლიობას. მაგალითად, gvari სვეტი არის პალატაში მწოლიარე ავადმყოფების გვარების ერთობლიობა, asaki სვეტი არის ავადმყოფების ასაკის ერთობლიობა.

ცხრილის თითოეული ელემენტი განისაზღვრება მისი მისამართით - A[i,j]. აქ A არის მონაცემის ელემენტი, i - ცხრილის სტრიქონის ნომერი, j - ცხრილის სვეტის ნომერი. ცხრილში სვეტების რაოდენობა განსაზღვრავს მის რიგს (ხარისხს). მაგალითად, Palata ცხრილის რიგი 5-ის ტოლია. ცხრილის სტრიქონი ქმნის კორტეჟს. სტრიქონების რაოდენობა განსაზღვრავს ცხრილის სიმძლავრეს. მაგალითად, Palata ცხრილის სიმძლავრეა 4. ყველა სტრიქონის ერთობლიობა ქმნის ცხრილს.

ცხრილი 1.2. ინფორმაცია ავადმყოფების შესახებ (Palata ცხრილი).

avadmyopiID	gvari	asaki	palatis_nomeri	ganyofileba
1	სამხარაძე	49	III-1	თერაპიული
2	ქევხიშვილი	50	IV-3	ქირურგიული
3	ყალაბეგიშვილი	52	V-4	კარდიოლოგიური
4	მამაიაშვილი	55	III-2	თერაპიული

თითოეულ სვეტს (ველს) აქვს სახელი და ტიპი. ტიპები მოყვანილია ცხრილში 1.2. ერთი ცხრილის სვეტებს სხვადასხვა სახელი უნდა ჰქონდეს. თუმცა, სხვადასხვა ცხრილს შეიძლება ერთნაირი სახელის მქონე სვეტები ჰქონდეთ. ნებისმიერ ცხრილს ერთი სვეტი მაინც უნდა ჰქონდეს. ცხრილში სვეტები განლაგებულია იმ მიმდევრობით, რა მიმდევრობითაც იყო მითითებული მათი სახელები ცხრილის შექმნის დროს. ცხრილის სტრიქონებს სახელები არ აქვთ, მათი რიგითობა განსაზღვრული და რაოდენობა შეზღუდული არ არის.

მონაცემთა რელაციური ბაზები

მონაცემთა რელაციური ბაზა არის ერთმანეთთან ლოგიკურად დაკავშირებული ცხრილების ერთობლიობა. მონაცემთა რელაციური ბაზის დაპროექტებისას უნდა დავიცვათ შემდეგი წესები:

– მონაცემთა ბაზაში თითოეულ ცხრილს უნდა ჰქონდეს უნიკალური სახელი და უნდა შედგებოდეს ერთტიპური სტრიქონებისაგან.

– თითოეული ცხრილი შედგება სვეტებისა და მნიშვნელობების ფიქსირებული რაოდენობისაგან. სტრიქონის ერთ სვეტში მოთავსებული უნდა იყოს მხოლოდ ერთი მნიშვნელობა. მაგალითად, ნებისმიერი სტრიქონის gvari სვეტი უნდა შეიცავდეს მხოლოდ ერთ გვარს.
– ცხრილში არ უნდა იყოს ზუსტად ერთნაირი ორი სტრიქონი. სტრიქონები უნდა განსხვავდებოდეს ერთი სვეტის მნიშვნელობით მაინც, რომ შესაძლებელი იყოს მათი ცალსახად იდენტიფიცირება.
– ცხრილის შიგნით თითოეულ სვეტს უნდა ჰქონდეს უნიკალური სახელი. სვეტისთვის უნდა განისაზღვროს მასში მოთავსებული მონაცემების ტიპი. მაგალითად, მთელი, წილადი, სტრიქონი, თარიღი და ა.შ.
– მონაცემების დამუშავებისას შესაძლებელი უნდა იყოს თავისუფალი მიმართვა ნებისმიერ სტრიქონთან ან სვეტთან.

ცხრილების ნორმალიზება

მონაცემთა ბაზის დაპროექტების მიზნებია:
– მონაცემთა ბაზაში საჭირო მონაცემების შენახვის უზრუნველყოფა.
– მონაცემების სიჭარბის გამორიცხვა.
– ცხრილების რაოდენობის მინიმუმამდე დაყვანა.

ცხრილი 1.3. არანორმალიზებული ცხრილი.

gamyidveli	myidveli	misamarti	tarigi	saqoneli	tanxa
სამხარაძე რ.	ხოშტარია ც.	თბილისი, ბათუმის 27 ა	12.01.07	მაცივარი	520
გაჩეჩილაძე ლ.	სამხარაძე ს.	ბათუმი, რუსთაველის 10	12.01.07	გაზქურა, მობილური ტელეფონი	860
შენგელია ნ.	კაპანაძე ე.	თბილისი, ზანდუკელის 11	10.02.07	გაზქურა, სარეცხი მანქანა, კომპიუტერი	2110
სამხარაძე რ.	მამაიაშვილი ჯ.	ქუთაისი, აღმაშენებლის 5	19.03.07	ტელევიზორი	290
ქევხიშვილი ა.	ყალაბეგიშვილი გ.	თბილისი, არბოს 2	05.10.07	კომპიუტერი, ციფრული კამერა	2600

ცხრილების ნორმალიზება არის მონაცემების წარმოდგენის პროცესი მარტივი ორგანზომილებიანი ცხრილებით, რაც საშუალებას გვაძლევს თავიდან ავიცილოთ მონაცემების დუბლირება და უზრუნველვყოთ ბაზაში შენახული მონაცემების არაწინააღმდეგობრიობა. ნორმალიზების მიზანია მონაცემთა ბაზის ისეთი პროექტის მიღება, რომელშიც ინფორმაციის ნებისმიერი ნაწილი შენახული იქნება მხოლოდ ერთ ადგილზე, ე.ი. გამორიცხული იქნება ინფორმაციის სიჭარბე. ეს კეთდება არა მარტო ადგილის ეკონომიის, არამედ, შენახული მონაცემების წინააღმდეგობრიობის გამორიცხვის მიზნით.

ცხრილი ითვლება *ნორმალიზებულად გარკვეულ დონეზე*, როცა ის აკმაყოფილებს

ნორმალიზების შესაბამისი ფორმის მოთხოვნებს. ნორმალიზების პროცესი წარმოადგენს ცხრილის სტრუქტურის მიმდევრობით ცვლილებას მანამ, სანამ ის არ დააკმაყოფილებს ნორმალიზების რომელიმე ფორმის მოთხოვნებს.

არსებობს ცხრილის ნორმალიზების ექვსი ფორმა. ყველაზე ხშირად, ნორმალიზების პირველი სამი ფორმა გამოიყენება, რადგან მათი რეალიზება საკმაოდ ადვილია და ნორმალიზების საკმარის დონეს უზრუნველყოფენ. ესენია:

1. პირველი ნორმალური ფორმა (First Normal Form, 1NF);
2. მეორე ნორმალური ფორმა (Second Normal Form, 2NF);
3. მესამე ნორმალური ფორმა (Third Normal Form, 3NF);

არანორმალიზებულ ცხრილს (ცხრილი 1.3) აქვს რიგი ნაკლოვანებებისა, რაც მასთან მუშაობის დროს გარკვეულ პრობლემებს ქმნის. დავუშვათ, არანორმალიზებული ცხრილი შეიცავს ინფორმაციას გამყიდველისა და მყიდველის შესახებ. მასში სტრიქონის დამატების დროს უნდა მივუთითოთ გამყიდველისა და მყიდველის გვარი, მყიდველის მისამართი, ყიდვის თარიღი, საქონლის დასახელება და თანხა. თუ იგივე მომხმარებელი მომავალში სხვა საქონელს იყიდის, მაშინ მეორეჯერ მოგვიწევს მისი მონაცემების შეტანა. შედეგად, ადგილი ექნება ინფორმაციის სიჭარბეს, რადგან ინფორმაცია ერთი მყიდველის შესახებ ორჯერ და მეტჯერ იქნება შეტანილი მონაცემთა ბაზაში. გარდა ამისა, როცა ერთი და იგივე მონაცემი ერთი მყიდველის შესახებ რამდენიმეჯერ შეგვაქვს, შეიძლება შეცდომა დავუშვათ. მაგალითად, თუ მყიდველის გვარის შეტანისას ერთი სიმბოლო შეგვეშალა, მაშინ ერთი მყიდველი ორ სხვადასხვა მყიდველად ჩაითვლება და როცა დავაპირებთ ინფორმაციის ამორჩევას მყიდველების შესახებ, არასწორ შედეგს მივიღებთ. გარდა ამისა, სტრიქონის წაშლის შემთხვევაში დაიკარგება ინფორმაცია გამყიდველისა და მყიდველის შესახებ. ამ პრობლემის გადასაჭრელად საჭიროა ცხრილების ნორმალიზება.

პირველი ნორმალური ფორმა

ცხრილი იმყოფება *პირველ ნორმალურ ფორმაში* მაშინ, როცა ის არ შეიცავს გამეორებად სტრიქონებს და სვეტების შედგენილ მნიშვნელობებს. ანუ თითოეული სვეტი უნდა შეიცავდეს ერთ მნიშვნელობას და არა მათ კომბინაციას. სტრიქონის უნიკალურობა მიიღწევა ცხრილში გასაღების უნიკალური მნიშვნელობების მოცემით. მონაცემთა რელაციური ბაზის დაპროექტებისადმი წაყენებული მოთხოვნებიდან გამომდინარე, ყველა ცხრილი მონაცემთა რელაციურ ბაზაში ავტომატურად იმყოფება პირველ ნორმალურ ფორმაში. ნებისმიერი სვეტი შედგება ისეთი მნიშვნელობების სიმრავლისაგან, რომლებიც ვეღარ დაიყოფა უფრო მცირე შემადგენელ ნაწილებად. ცხრილში 1.4 ნაჩვენებია ცხრილი 1.3 ნორმალიზების შემდეგ.

როგორც ვხედავთ, „misamarti” სვეტი დაიყო ორ სვეტად: „qalaqi” და „qucha”. ეს გაკეთდა იმიტომ, რომ შესაძლებლობა გვქონდეს შევასრულოთ ქალაქებსა და მათ რაიონებში გაყიდული საქონლის სტატისტიკური დამუშავება, სტრიქონების გაფილტვრა ქალაქებისა და მათი რაიონების მიხედვით. თუ ასეთი საჭიროება არ არის, მაშინ შეგვიძლია არ შევასრულოთ „misamarti” სვეტის დაყოფა ნაწილებად.

როგორც ვხედავთ, ცხრილის ზომა, პირველ ნორმალურ ფორმაზე დაყვანის შემდეგ, გაიზარდა. პირველ ნორმალურ ფორმაში მყოფ ცხრილს *უნივერსალურ ცხრილს* უწოდებენ. მასში შეიძლება ჩავრთოთ ყველა საჭირო სვეტი და გამოვიყენოთ მონაცემთა ბაზაში შესანახი მთელი ინფორმაციის მოსათავსებლად. პირველ ნორმალურ ფორმაში მყოფ ცხრილს შემდეგი ნაკლი აქვს:

ცხრილი 1.4. ცხრილი პირველ ნორმალურ ფორმაში.

gamyidveli	myidveli	qalaqi	qucha	tarigi	saqoneli	tanxa
სამხარაძე რ.	ხოშტარია ც.	თბილისი	ბათუმის 27	12.01.07	მაცივარი	520
გაჩეჩილაძე ლ.	სამხარაძე ს.	ბათუმი	რუსთაველის 10	12.01.07	გაზქურა	460
გაჩეჩილაძე ლ.	სამხარაძე ს.	ბათუმი	რუსთაველის 10	12.01.07	მობილური ტელეფონი	400
შენგელია ნ.	კაპანაძე ე.	თბილისი	ზანდუკელის 11	10.02.07	სარეცხი მანქანა	650
შენგელია ნ.	კაპანაძე ე.	თბილისი	ზანდუკელის 11	10.02.07	გაზქურა	460
შენგელია ნ.	კაპანაძე ე.	თბილისი	ზანდუკელის 11	10.02.07	კომპიუტერი	1000
სამხარაძე რ.	მამაიაშვილი ჯ.	ქუთაისი	აღმაშენებლის 5	19.03.07	ტელევიზორი	290
ქევხიშვილი ა.	ყალაბეგიშვილი გ.	თბილისი	არბოს 2	05.10.07	კომპიუტერი	1000
ქევხიშვილი ა.	ყალაბეგიშვილი გ.	თბილისი	არბოს 2	05.10.07	ციფრული კამერა	1600

– ახალი მონაცემების დამატება ითხოვს ყველა სვეტის მნიშვნელობის შეტანას, თუნდაც ეს მონაცემები უკვე არსებობდეს ცხრილში. შედეგად, გვექნება მონაცემების სიჭარბე, რაც ზრდის შეცდომის დაშვების ალბათობას ორი ერთნაირი მონაცემის შეტანისას. შედეგად, ერთის ნაცვლად გვექნება ორი სხვადასხვა მნიშვნელობა. გარდა ამისა, შეუძლებელია ინფორმაციის დამატება უფრო ადრე, ვიდრე ის მოთხოვნილი იქნება. მაგალითად, შეუძლებელია გამყიდველის გვარის შეტანა მანამ, სანამ ის საქონელს არ გაყიდის.

– თუ გამყიდველის გვარი შეიცვალა, მაშინ ცვლილებები უნდა შესრულდეს ყველა სტრიქონში, სადაც ეს გვარი გვხვდება.

– სტრიქონის წაშლისას დაიკარგება ინფორმაცია გამყიდველის, მყიდველისა და საქონლის შესახებ.

ამ პრობლემების თავიდან ასაცილებლად, ცხრილი უნდა დავიყვანოთ მეორე ან მესამე ნორმალურ ფორმაზე.

მეორე ნორმალური ფორმა

ცხრილი იმყოფება *მეორე ნორმალურ ფორმაში* მაშინ, როცა ის აკმაყოფილებს პირველი ნორმალური ფორმის მოთხოვნებს, შეიცავს ერთ ან მეტ სვეტს, რომლებიც ქმნის პირველად გასაღებს (უნიკალურ იდენტიფიკატორს) თითოეული სტრიქონისათვის და ყველა ის სვეტი, რომელიც არ შედის პირველად გასაღებში, პირველად გასაღებთან დაკავშირებულია სრული ფუნქციური დამოკიდებულებით, ე.ი. ნებისმიერი არასაგასაღებო სვეტი ცალსახად განისაზღვრება საგასაღებო სვეტების სრული ნაკრებით. თითოეული არასაგასაღებო სვეტი უნდა იყოს სრულად ფუნქციურად დამოკიდებული მთელ პოტენციურ

ცხრილი 1.5. ინფორმაცია შეკვეთების შესახებ (ცხრილი Shekveta).

shekvetaID	gamyidveliID	myidveliID	tarigi	tanxa
1	1	1	12.01.07	520
2	2	2	12.01.07	860
3	3	3	10.02.07	2110
4	4	5	19.03.07	290
5	1	4	05.10.07	2600

ცხრილი 1.6. ინფორმაცია გამყიდვლების შესახებ (ცხრილი Gamyidveli).

gamyidveliID	gvari
1	სამხარაძე რ.
2	გაჩეჩილაძე ლ.
3	შენგელია ნ.
4	ქევხიშვილი ა.

ცხრილი 1.7. ინფორმაცია მყიდველების შესახებ (ცხრილი Myidveli).

myidveliID	gvari	qalaqi	qucha
1	ხოშტარია ც.	თბილისი	ბათუმის 27
2	სამხარაძე ს.	ბათუმი	რუსთაველის 10
3	კაპანაძე ე.	თბილისი	ზანდუკელის 11
4	მამაიაშვილი ჯ.	ქუთაისი	აღმაშენებლის 5
5	ყალაბეგიშვილი გ.	თბილისი	არბოს 2

ცხრილი 1.8. ინფორმაცია საქონლის შესახებ (ცხრილი Saqoneli).

saqoneliID	saqoneli	pasi
1	მაცივარი	520
2	გაზქურა	460
3	მობილური ტელეფონი	400
4	სარეცხი მანქანა	650
5	კომპიუტერი	1000
6	ტელევიზორი	290
7	ციფრული კამერა	1600

ცხრილი 1.9. ცხრილი შეკვეთებს საქონელთან აკავშირებს (ცხრილი ShekvetaSaqoneli).

shekvetaSaqoneliID	shekvetaID	saqoneliID
1	1	1
2	2	2
3	2	3
4	3	2
5	3	4
6	3	5
7	4	7
8	4	5
9	5	6

გასაღებზე ანუ გასაღებში შემავალ ყველა სვეტზე. სხვა სიტყვებით, არასაგასაღებო სვეტი არ შეიძლება იყოს სრულად ფუნქციურად დამოკიდებული პოტენციური გასაღების ნაწილზე ანუ გასაღების ზოგ სვეტზე.

თუ ცხრილს აქვს მარტივი პირველადი გასაღები, რომელიც მხოლოდ ერთი სვეტისაგან შედგება, მაშინ ის ავტომატურად იმყოფება მეორე ნორმალურ ფორმაში. თუ პირველადი გასაღები შედგენილია (ანუ რამდენიმე სვეტისაგან შედგება), მაშინ არ არის აუცილებელი, რომ ცხრილი იმყოფებოდეს მეორე ნორმალურ ფორმაში. ამ შემთხვევაში, ცხრილი უნდა დაიყოს ორ ან მეტ ცხრილად ისე, რომ პირველადი გასაღები ცალსახად განსაზღვრავდეს მნიშვნელობას ნებისმიერ სვეტში. თუ ცხრილში არის თუნდაც ერთი სვეტი, რომელიც არ არის დამოკიდებული პირველად გასაღებზე, მაშინ პირველად გასაღებში უნდა ჩავრთოთ დამატებითი სვეტები. თუ ასეთი სვეტები არ არის, მაშინ აუცილებელია ახალი სვეტის დამატება.

ჩვენს ცხრილს არ აქვს ისეთი სვეტები, რომელთა მნიშვნელობები ცალსახად განსაზღვრავს სტრიქონს. პირველად გასაღებად შეგვიძლია გამოვიყენოთ თარიღის, გამყიდვლის გვარის, მყიდველის გვარისა და საქონლის დასახელების კომბინაცია. „qalaqi" და „qucha" სვეტების მნიშვნელობები დამოკიდებულია მყიდველის გვარზე, ხოლო „pasi" სვეტის მნიშვნელობები კი - „saqoneli" სვეტის მნიშვნელობებზე.

ჩვენი ცხრილის დასაყვანად მეორე ნორმალურ ფორმაზე ის უნდა დავყოთ ოთხ ცხრილად (ცხრილები 1.5-1.8). ამ ცხრილების გარდა უნდა შემოვიტანოთ კიდევ ერთი (ცხრილი 1.9) - შეკვეთების ცხრილის დასაკავშირებლად საქონლის ცხრილთან.

მეორე ნორმალურ ფორმაზე ერთი და იგივე ცხრილი შეიძლება სხვადასხვა გზით დავიყვანოთ. ცხრილის მარტივმა დაყოფამ რამდენიმე მცირე ზომის ცხრილად შეიძლება მონაცემების ლოგიკური მთლიანობის დაკარგვა გამოიწვიოს. თითოეულ ცხრილში პირველადი გასაღებისათვის უნდა გავითვალისწინოთ დამატებითი სვეტი ან სვეტები. პირველადი გასაღების გამოყენებით ადვილია სხვადასხვა ცხრილში მოთავსებული მონაცემების დაკავშირება.

მეორე ნორმალურ ფორმაზე ჩვენი ცხრილის დაყვანისას თითოეულ ცხრილს დაემატა სვეტი, რომელიც გამოყენებული იქნა როგორც პირველადი გასაღები. მნიშვნელობა ახალ სვეტში ცალსახად განსაზღვრავს სტრიქონში ნებისმიერ არასაგასაღებო მნიშვნელობას. მეორე ნორმალურ ფორმაზე ცხრილის დაყვანა აგვაცილებს ერთი და იმავე მონაცემის გამეორებას.

მესამე ნორმალური ფორმა

ცხრილი იმყოფება *მესამე ნორმალურ ფორმაში* მაშინ, როცა ის აკმაყოფილებს მეორე ნორმალური ფორმის მოთხოვნებს და მისი არც ერთი არასაგასაღებო სვეტი ფუნქციურად არ არის დამოკიდებული სხვა ნებისმიერ არასაგასაღებო სვეტზე.

მესამე ნორმალური ფორმის მოთხოვნა დაიყვანება იმაზე, რომ ყველა არასაგასაღებო სვეტი დამოკიდებული იყოს მხოლოდ პირველად გასაღებზე და არ იყოს დამოკიდებული ერთმანეთზე. სხვა სიტყვებით რომ ვთქვათ, ნებისმიერი არასაგასაღებო სვეტის მნიშვნელობის შეცვლისას არ უნდა შეიცვალოს სხვა სვეტის მნიშვნელობა.

ზემოთ მოყვანილი ცხრილებიდან მხოლოდ ცხრილი 1.7 არ აკმაყოფილებს მესამე ნორმალური ფორმის მოთხოვნებს. საქმე ის არის, რომ „qucha" სვეტი დამოკიდებულია „qalaqi" სვეტზე. ამ დამოკიდებულების აღმოსაფხვრელად ცხრილი ორ მცირე ზომის ცხრილად უნდა დავყოთ (ცხრილები 1.10 და 1.11). გარდა ამისა, ცხრილში 1.7 უნდა შევიტანოთ ცვლილებები, კერძოდ, „qalaqi" და „qucha" სვეტები უნდა შევცვალოთ „quchaID" სვეტით.

ცხრილი 1.10. ინფორმაცია მისამართების შესახებ (ცხრილი Qucha).

quchaID	qalaqiID	qucha
1	1	ბათუმის 27
2	1	ზანდუკელის 11
3	2	რუსთაველის 10
4	3	აღმაშენებლის 5
5	1	არბოს 2

ცხრილი 1.11. ინფორმაცია ქალაქების შესახებ (ცხრილი Qalaqi).

qalaqiID	qalaqi
1	თბილისი
2	ბათუმი
3	ქუთაისი

ცხრილებს შორის არსებული კავშირები

ცხრილის ნორმალიზების შემდეგ მიიღება რამდენიმე ცხრილი და დგება ორი ცხრილის სტრიქონების ერთმანეთთან დაკავშირების აუცილებლობა. არანორმალიზებული მონაცემები ერთ ცხრილში (ცხრილი 1.3) ინახებოდა, რომლის თითოეული სტრიქონი წარმოადგენდა მონაცემების ლოგიკურად დასრულებულ ობიექტს. ამ ცხრილის მონაცემები, მესამე ნორმალურ ფორმაზე დაყვანის შემდეგ, რამდენიმე ცხრილში აღმოჩნდა. მიღებული ცხრილებიდან საჭირო მონაცემების ამოსარჩევად საჭიროა ამ ცხრილების დაკავშირება, რისთვისაც პირველადი და გარე გასაღებები გამოიყენება.

პირველადი და გარე გასაღებები

ორ ცხრილს შორის კავშირის დამყარების დროს ერთი არის მთავარი (მშობელი), მეორე კი - დამოკიდებული (შვილობილი). კავშირის დასამყარებლად მთავარი ცხრილის პირველადი გასაღები უკავშირდება დამოკიდებული ცხრილის გარე გასაღებს. როცა მთავარ ცხრილში ავირჩევთ რომელიმე სტრიქონს, მაშინ დამოკიდებული ცხრილიდან ამოირჩევა შესაბამისი სტრიქონები.

ნებისმიერ ცხრილს აქვს ერთი ან მეტი სვეტი, რომელიც ცალსახად განსაზღვრავს თითოეულ სტრიქონს. ასეთ სვეტს (ან სვეტებს) *პირველადი გასაღები* (PRIMARY KEY) ეწოდება. ის უნდა აკმაყოფილებდეს შემდეგ მოთხოვნებს:

– *უნიკალურობა.* ეს იმას ნიშნავს, რომ ცხრილში არ უნდა იყოს ორი ისეთი სტრიქონი, რომლებსაც პირველადი გასაღების ერთნაირი მნიშვნელობები ექნებათ.

– *მინიმალურობა.* ეს იმას ნიშნავს, რომ პირველად გასაღებში შემავალი სვეტებიდან ნებისმიერის გამოკლება უნდა იწვევდეს გასაღების უნიკალურობის დარღვევას. ანუ, პირველადი გასაღები არ უნდა შეიცავდეს ისეთ სვეტს, რომლის მოცილების შემდეგ გასაღები ისევ უნიკალური დარჩება.

როგორც აღვნიშნეთ, ცხრილების დასაკავშირებლად პირველადი და გარე გასაღებები გამოიყენება. *პირველადი გასაღები* (PRIMARY KEY) ცალსახად განსაზღვრავს ცხრილში სტრიქონს. ის შეიძლება შეიცავდეს ცხრილის ერთ ან მეტ არსებულ სვეტს, ან შეგვიძლია შევქმნათ ახალი სვეტი. გასაღებში შემავალმა არც ერთმა სვეტმა არ უნდა მიიღოს განუსაზღვრელი მნიშვნელობა. რამდენიმე სვეტისაგან შედგენილი პირველადი გასაღების გამოყენების ნაცვლად, ხშირად უმჯობესია ცხრილს ერთი სვეტი დავუმატოთ, რომლის

მნიშვნელობები უნიკალური იქნება. გარდა ამისა, ახალი სვეტის შექმნა აადვილებს მონაცემების ცვლილებას ცხრილში. მაგალითად, თუ ცხრილის პირველადი გასაღებია პიროვნების გვარი, რომელიც გამოიყენება დამოკიდებულ ცხრილებთან კავშირისათვის (მისათვის), მაშინ გვარის შეცვლის შემთხვევაში, გვარის შეცვლა მოგვიწევს როგორც მთავარ, ისე დამოკიდებულ ცხრილებში. პირველადი გასაღებისათვის სპეციალური სვეტის შექმნა თავიდან აგვაცილებს ასეთ პრობლემას, რადგან ამ სვეტის მნიშვნელობა არასოდეს შეიცვლება. თუ რაიმე სპეციალური მოთხოვნები არ არსებობს, მაშინ სასურველია, რომ პირველადი გასაღები ყველა ცხრილს ჰქონდეს.

გარე გასაღები (FOREIGN KEY) იქმნება დამოკიდებულ ცხრილში, რომელიც მთავარი ცხრილის მონაცემებს მიმართავს. გარე გასაღები შეიძლება შედგებოდეს ერთი ან მეტი სვეტისაგან. გარე გასაღების მნიშვნელობა ყოველთვის უნდა არსებობდეს მთავარ ცხრილში, ან უნდა იყოს განუსაზღვრელი. ამავე დროს, მთავარი ცხრილის რომელიმე სტრიქონის პირველადი გასაღების მნიშვნელობა შეიძლება არ იყოს დაკავშირებული დამოკიდებული ცხრილის არც ერთ სტრიქონთან.

qalaqiID	qalaqi
1	თბილისი
2	ბათუმი
3	ქუთაისი

quchaID	qalaqiID	qucha
1	1	ბათუმის 27
2	1	ზანდუკელის 11
5	1	არბოს 2

ნახ. 1.1.

განვიხილოთ მაგალითი. დავუშვათ, Qalaqi ცხრილი არის მთავარი, ხოლო Qucha ცხრილი - დამოკიდებული. Qalaqi ცხრილის qalaqiID სვეტი (პირველადი გასაღები) დავუკავშიროთ Qucha ცხრილის qalaqiID სვეტს (გარე გასაღები). კავშირის დამყარების შემდეგ (რასაც მომდევნო თავებში ვისწავლით) Qalaqi ცხრილის ნებისმიერი სტრიქონის არჩევისას აირჩევა Qucha ცხრილის შესაბამისი სტრიქონები (ნახ. 1.1).

მთავარ ცხრილში პირველადი გასაღების მნიშვნელობის შეცვლისას არსებობს დამოკიდებული ცხრილის ქცევის სამი ვარიანტი:

– *კასკადირება (Cascading).* მთავარ ცხრილში პირველადი გასაღების მნიშვნელობების შეცვლისას სრულდება დამოკიდებული ცხრილის გარე გასაღების შესაბამისი მნიშვნელობების ცვლილება. ხდება ყველა არსებული კავშირების შენარჩუნება.

– *შეზღუდვა (Restrict).* დაუშვებელია პირველადი გასაღების იმ მნიშვნელობების შეცვლა, რომლებიც დაკავშირებულია დამოკიდებული ცხრილის სტრიქონებთან. დასაშვებია პირველადი გასაღების იმ მნიშვნელობების შეცვლა, რომლებიც არ არის დაკავშირებული დამოკიდებული ცხრილის სტრიქონებთან.

– *დაყენება (Relation).* პირველადი გასაღების მნიშვნელობის შეცვლის დროს გარე გასაღები იღებს განუსაზღვრელ მნიშვნელობას (NULL). ინფორმაცია დამოკიდებული ცხრილის სტრიქონების კავშირის შესახებ იკარგება (ანუ მთავარი ცხრილის რომელ სტრიქონებთან იყოს დაკავშირებული). შედეგად, შეუძლებელი ხდება იმის გარკვევა, თუ რომელი სტრიქონი რომელ პირველად გასაღებთან იყო დაკავშირებული.

მთავარი ცხრილის სტრიქონის წაშლისას არსებობს დამოკიდებული ცხრილის ქცევის შემდეგი ვარიანტები:

– *კასკადირება (Cascading).* მთავარ ცხრილში სტრიქონების წაშლისას სრულდება დამოკიდებული ცხრილის შესაბამისი სტრიქონების წაშლა.

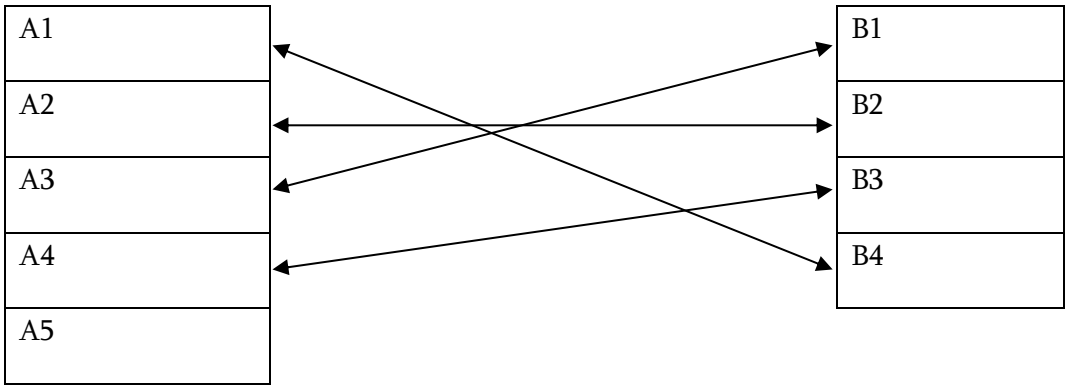
– *შეზღუდვა (Restrict).* დასაშვებია მთავარი ცხრილის მხოლოდ იმ სტრიქონების წაშლა, რომლებსაც არ აქვთ ბმული სტრიქონები დამოკიდებულ ცხრილში. საწინაღმდეგო შემთხვევაში, წაშლის ოპერაცია არ შესრულდება.
– *დაყენება (Relation).* მთავარი ცხრილიდან სტრიქონის წაშლის დროს დამოკიდებულ ცხრილში შესაბამისი გარე გასაღები იღებს განუსაზღვრელ მნიშვნელობას (NULL).

„ერთი-ერთთან" კავშირი

თუ პირველი (მთავარი) ცხრილის კონკრეტული სტრიქონი დროის ნებისმიერ მომენტში დაკავშირებულია მეორე (დამოკიდებული) ცხრილის ნულ (არც ერთ) ან ერთ სტრიქონთან, მაშინ მათ შორის დამყარებულია „ერთი-ერთთან" კავშირი (ნახ. 1.2). ეს კავშირი იქმნება იმ შემთხვევაში, როცა ორივე სვეტი არის პირველადი გასაღები ან ორივე სვეტს აქვს უნიკალურობის შეზღუდვა ანუ მათი მნიშვნელობები უნდა იყოს უნიკალური. „ერთი-ერთთან" კავშირი იქმნება იმ შემთხვევაში, როცა გვინდა ბევრი სვეტის მქონე ცხრილის დაყოფა, ცხრილის ნაწილის იზოლირება უსაფრთხოების მოსაზრებიდან გამომდინარე და ა.შ.

„ერთი-ერთთან" კავშირის პირველადი გასაღების მხარე აღინიშნება გასაღების სიმბოლოთი. გარე გასაღების მხარეც, აგრეთვე, აღინიშნება გასაღების სიმბოლოთი (ნახ. 1.4).

ასეთი კავშირის მაგალითია კავშირი ადამიანსა და მისი პასპორტის ნომერს შორის. თითოეულ ადამიანს ერთი პასპორტი აქვს და ერთი პასპორტი მხოლოდ ერთ ადამიანს ეკუთვნის. აქედან გამომდინარე, ლოგიკური იქნება, რომ ინფორმაცია ადამიანსა და მისი პასპორტის ნომრის შესახებ ერთ ცხრილში მოვათავსოთ.



ნახ. 1.2.

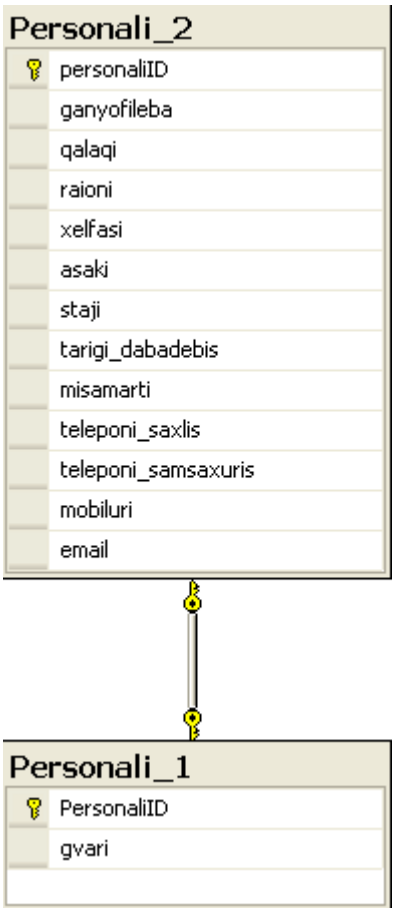
PersonaliID	gvari
1	ხოშტარია ც.
2	სამხარაძე ს.
3	კაპანაძე ე.
4	მამაიაშვილი ჯ.
5	ყალაბეგიშვილი გ.

PersonaliID	qalaqi	. . .	dabadebis_tarigi
1			
2	ბათუმი	19.03.2005	
3			
4			
5			

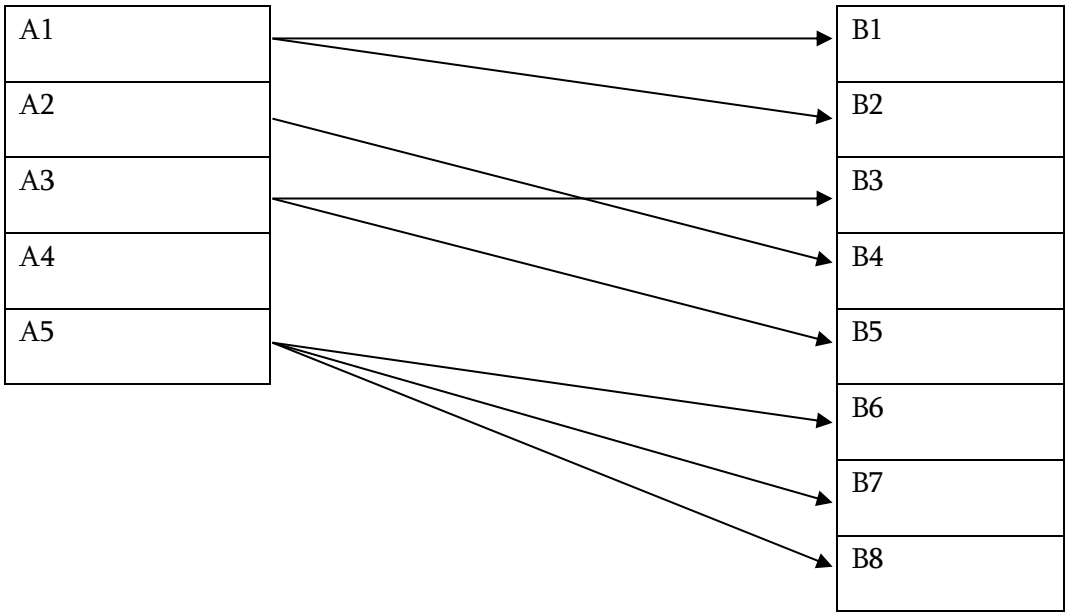
ნახ. 1.3.

განვიხილოთ მაგალითი. დავუშვათ გვაქვს Personali ცხრილი, რომელიც შეიცავს

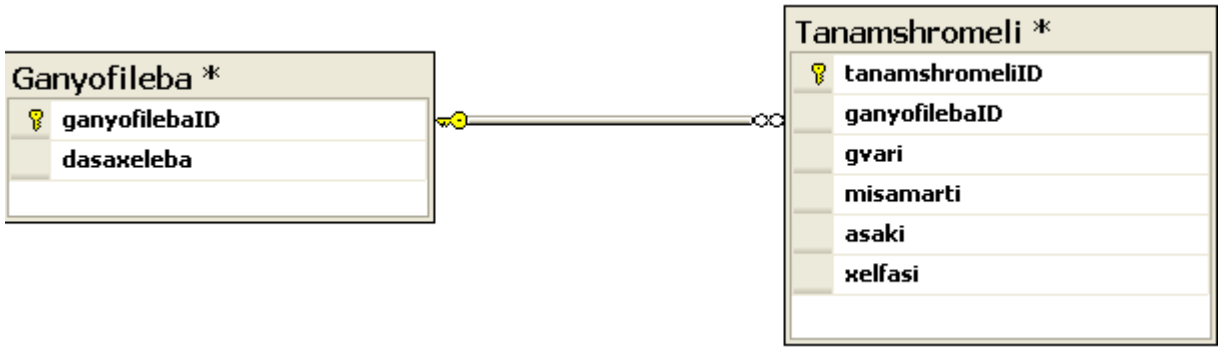
ინფორმაციას ფირმის თანამშრომლების შესახებ. ის შემდეგ სვეტებს შეიცავს: personaliID, gvari, qalaqi, qucha, telefoni_saxlis, telefoni_samsaxuris, mobiluri, pasportis nomeri, erovneba, sad_mushaobs, xelfasi, dabadebis_tarigi. თუ ამ ცხრილთან ხშირად გვიწევს მუშაობა, მაშინ უმჯობესია ის ორ ცხრილად დავყოთ: Personali1 და Personali2. Personali1 ცხრილი უნდა იყოს მცირე ზომის, ანუ შეიცავდეს სვეტების მცირე რაოდენობას, Personali2 ცხრილი კი - უფრო დიდი ზომის, ანუ შეიცავდეს სვეტების დიდ რაოდენობას. Personali1 ცხრილში მოვათავსოთ ორი სვეტი: personaliID და gvari. აქ PersonaliID სვეტი პირველადი გასაღებია. Personali2 ცხრილში მოვათავსოთ სვეტები: PersonaliID, qalaqi, qucha, telefoni_saxlis, telefoni_samsaxuris, mobiluri, pasportis_nomeri, erovneba, sad_mushaobs, xelfasi და dabadebis_tarigi. აქ personaliID სვეტი პირველადი გასაღებია. Personali ცხრილის personaliID სვეტი (პირველადი გასაღები) დავუკავშიროთ Personali2 ცხრილის personaliID სვეტს, რომელიც მუშაობს როგორც გარე გასაღები. შედეგად, Personali1 ცხრილში რომელიმე სტრიქონის არჩევისას, Personali2 ცხრილში აირჩევა შესაბამისი ერთი სტრიქონი (ნახ. 1.3).



ნახ. 1.4.



ნახ. 1.5.



ნახ. 1.6.

„**ერთი-ბევრთან**” კავშირი

თუ პირველი (მთავარი) ცხრილის კონკრეტული სტრიქონი დროის ნებისმიერ მომენტში დაკავშირებულია მეორე (დამოკიდებული) ცხრილის ნულ, ერთ ან მეტ სტრიქონთან და მეორე ცხრილის ნებისმიერი სტრიქონი დაკავშირებულია პირველის ცხრილის მხოლოდ ერთ სტრიქონთან, მაშინ მათ შორის დამყარებულია „ერთი-ბევრთან” კავშირი (ნახ. 1.5).

„ერთი-ბევრთან” კავშირი იქმნება იმ შემთხვევაში, როცა ბმული სვეტებიდან ერთ-ერთი არის პირველადი გასაღები ან აქვს უნიკალურობის შეზღუდვა. „ერთი-ბევრთან” კავშირის პირველადი გასაღების მხარე აღინიშნება სიმბოლოთი, გარე გასაღების მხარე კი - სიმბოლოთი (ნახ. 1.6).

ასეთი კავშირის მაგალითია კავშირი ფირმის თანამშრომლებსა და განყოფილებებს შორის. დროის ნებისმიერ მომენტში თითოეულ განყოფილებაში მუშაობს რამდენიმე თანამშრომელი და თითოეული თანამშრომელი მხოლოდ ერთ განყოფილებაში მუშაობს. ამიტომ, განყოფილებების შემცველი ცხრილი იქნება მთავარი, რომელსაც დაუკავშირდება თანამშრომლების ცხრილი. შედეგად, განყოფილებების ცხრილის თითოეულ სტრიქონს

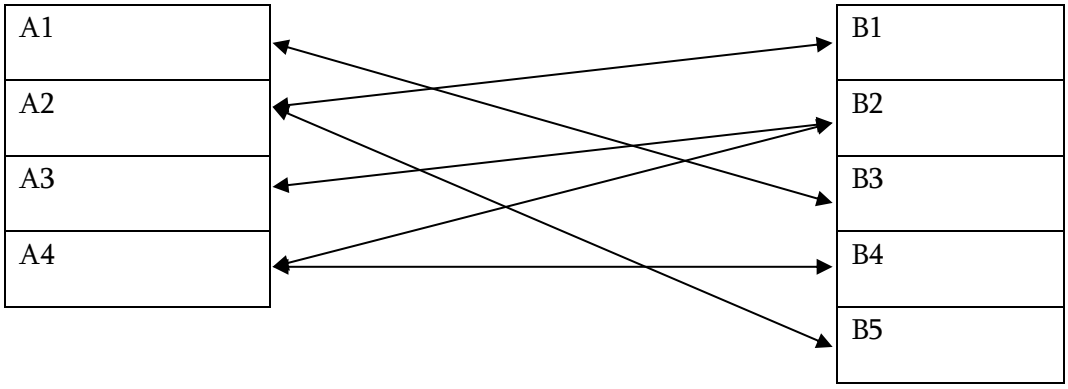
თანამშრომლების ცხრილის რამდენიმე სტრიქონი შეესაბამება და თანამშრომლების ცხრილის თითოეულ სტრიქონს კი - განყოფილებების ცხრილის მხოლოდ ერთი სტრიქონი.

„ბევრი-ბევრთან" კავშირი

თუ პირველი (მთავარი) ცხრილის კონკრეტული სტრიქონი დროის ნებისმიერ მომენტში დაკავშირებულია მეორე (დამოკიდებული) ცხრილის ნულ, ერთ ან მეტ სტრიქონთან და პირიქით, მაშინ მათ შორის დამყარებულია „ბევრი-ბევრთან" კავშირი (ნახ. 1.7). ასეთი კავშირის მაგალითებია:

ა) კავშირი ფირმის თანამშრომლებსა და პროექტს შორის. დროის ნებისმიერ მომენტში ერთ პროექტზე შეიძლება ფირმის რამდენიმე თანამშრომელი მუშაობდეს და პირიქით, ერთი თანამშრომელი შეიძლება რამდენიმე პროექტზე მუშაობდეს.

ბ) კავშირი წიგნის ავტორებსა და წიგნებს შორის. ერთ წიგნს შეიძლება რამდენიმე ავტორი ჰყავდეს და პირიქით, ერთ ავტორს შეიძლება რამდენიმე წიგნი ჰქონდეს.



ნახ. 1.7.

დავუშვათ, გვაქვს ორი ცხრილი Avtori და Wigni. Avtori ცხრილი შეიცავს მონაცემებს ავტორების შესახებ, Wigni ცხრილი კი - მონაცემებს წიგნების შესახებ. ერთ ავტორს შეიძლება გამოქვეყნებული ჰქონდეს რამდენიმე წიგნი და ერთ წიგნს შეიძლება რამდენიმე ავტორი ჰყავდეს. ამ ორ ცხრილს შორის დავამყაროთ „ბევრი-ბევრთან" კავშირი. ამისათვის, უნდა გამოვიყენოთ მესამე, მაკავშირებელი ცხრილი - Makavshirebeli (ნახ. 1.8).

Avtori ცხრილის პირველადი გასაღებია AvtoriID, Wigni ცხრილის კი - WigniID. Makavshirebeli ცხრილის პირველადი გასაღები ორი სვეტისაგან შედგება და წარმოადგენს AvtoriID და WigniID სვეტების მნიშვნელობების კომბინაციას (ნახ. 1.9).

Avtori და Wigni ცხრილებს შორის „ბევრი-ბევრი" კავშირის დასამყარებლად უნდა შევქმნათ „ერთი-ბევრთან" კავშირი Avtori და Makavshirebeli ცხრილებს შორის და „ერთი-ბევრთან" კავშირი Wigni და Makavshirebeli ცხრილებს შორის (ნახ. 1.9). აქ, Avtori არის მთავარი ცხრილი, Makavshirebeli კი - დამოკიდებული. ანალოგიურად, Wigni არის მთავარი ცხრილი, Makavshirebeli კი - დამოკიდებული. Avtori ცხრილის AvtoriID სვეტს ვუკავშირებთ Makavshirebeli ცხრილის AvtoriID სვეტს, ხოლო Wigni ცხრილის WigniID სვეტს კი - Makavshirebeli ცხრილის WigniID სვეტს.

როგორც ნახ. 1.8-დან ჩანს, სამხარაძე არის სამი წიგნის ავტორი: Visual C#.NET, SQL Server და MS Office. გაჩეჩილაძე არის ერთი წიგნის ავტორი - SQL Server. კაპანაძე არის ორი წიგნის ავტორი: Visual C#.NET და Visual C++.NET. მეორე მხრივ, Visual C#.NET წიგნს ორი ავტორი ჰყავს: სამხარაძე და კაპანაძე. Visual C++.NET წიგნს ერთი ავტორი ჰყავს - კაპანაძე. SQL

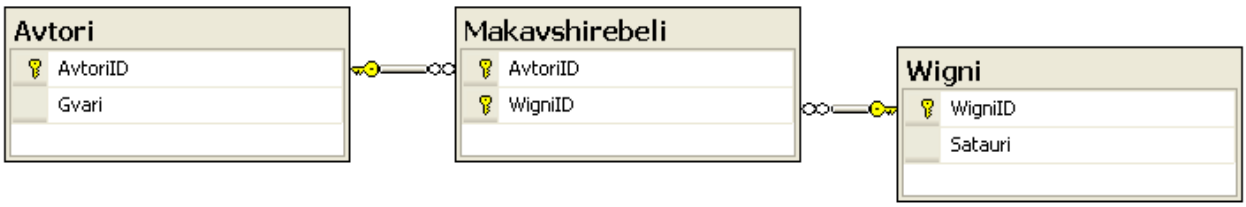
Server წიგნს ორი ავტორი ჰყავს: სამხარაძე და გაჩეჩილაძე. MS Office წიგნს ერთი ავტორი ჰყავს - სამხარაძე. „ბევრი-ბევრთან“ კავშირი სქემატურად ნაჩვენებია ნახ. 1.9-ზე.

Avtori	
AvtoriID	Gvari
1	სამხარაძე
2	გაჩეჩილაძე
3	კაპანაძე

Wigni	
WigniID	Satauri
1	Visual C#.NET
2	Visual C++.NET
3	SQL Server
4	MS Office

Makavshirebeli	
AvtoriID	WigniID
1	1
1	3
1	4
2	3
3	2
3	1

ნახ. 1.8.



ნახ. 1.9.

თავი 2. მონაცემთა ბაზები

მონაცემთა ბაზების არქიტექტურა

სერვერზე მონაცემები მონაცემთა ბაზებში ინახება. მონაცემთა ბაზის სტრუქტურა შეიძლება ორი კუთხით განვიხილოთ: ლოგიკური და ფიზიკური. მონაცემთა ბაზის *ლოგიკური სტრუქტურა* მოიცავს ცხრილების სტრუქტურას, მათ შორის კავშირებს, მომხმარებლებლების სიას, შენახულ პროცედურებს და მონაცემთა ბაზის სხვა ობიექტებს. მონაცემთა ბაზის *ფიზიკური სტრუქტურა* მოიცავს მონაცემთა ბაზის ფაილებისა და ტრანზაქციების ჟურნალის აღწერას, მათ საწყის ზომას, ნაზრდის ზომას, მაქსიმალურ ზომას, კონფიგურირების პარამეტრებს და ა.შ.

მონაცემთა ბაზების ობიექტები

მონაცემთა ბაზების ობიექტებია:

- *ცხრილები* (*tables*) - ორგანზომილებიანი მატრიცებია, რომლებშიც უშუალოდ მონაცემები ინახება;
- *შენახული პროცედურები* (*stored procedures*) - Transact_SQL-ის ბრძანებების ნაკრებია, შენახული გარკვეული სახელით. მათთან მიმართვა შესაძლებებლია სახელის საშუალებით;
- *ტრიგერები* (*triggers*) - სპეციალური შენახული პროცედურებია, რომლებიც ცხრილში მონაცემების შეცვლის დროს გამოიძახება;
- *წარმოდგენები* (*views*) - ვირტუალური ცხრილებია, რომლებიც საშუალებას გვაძლევს ამორჩევის შედეგთან ვიმუშაოთ როგორც ცხრილთან;
- *ინდექსები* (*indexes*) - სტრუქტურებია, რომლებიც ზრდის მონაცემებთან მუშაობის მწარმოებლურობას;
- *მომხმარებლების ტიპები* (*user-defined data types*) - მომხმარებლების მიერ შექმნილი მონაცემთა ტიპებია;
- *მომხმარებლების ფუნქციები* (*user-defined funqtions*) - მომხმარებლების მიერ შექმნილი ფუნქციებია;
- *გასაღებები* (*keys*) - მთლიანობის შეზღუდვის ერთ-ერთი სახეა, რომელიც უზრუნველყოფს მონაცემების მიმართვით მთლიანობას;
- *მთლიანობის შეზღუდვები (constraints)* - ობიექტებია, რომლებიც უზრუნველყოფენ მონაცემების ლოგიკურ მთლიანობას და არ არსებობს ცხრილებისაგან დამოუკიდებლად;
- *ავტომატური მნიშვნელობები* (defaults) - მონაცემთა ბაზის ობიექტებია, რომლებიც შეგვიძლია მივუთითოთ უშუალოდ ცხრილის სტრუქტურაში ან მომხმარებლების მიერ განსაზღვრულ ტიპებში.

ქვემოთ მოყვანილია თითოეული მათგანის მოკლე აღწერა.

ცხრილები

მონაცემთა ბაზის ობიექტებიდან მხოლოდ ცხრილი შეიცავს საკუთრივ მონაცემებს. ცხრილი შედგება სტრიქონებისა და სვეტებისაგან:

– *სტრიქონები* (*rows*). თითოეული სტრიქონი წარმოადგენს კონკრეტული ობიექტის მახასიათებლების ერთობლიობას. მაგალითად, სტუდენტისათვის ესაა: გვარი, სახელი, უნივერსიტეტის დასახელება, კურსი და ა.შ.

– *სვეტები* (*columns*). თითოეული სვეტი წარმოადგენს ობიექტის მახასიათებელს ან

მახასიათებლების ერთობლიობას. მაგალითად, სტუდენტების გვარი, სახელი და ა.შ. სტრიქონის სვეტი არის ცხრილის მინიმალური ელემენტი. ცხრილის თითოეულ სვეტს აქვს სახელი, ტიპი და ზომა.

ცხრილის ზოგიერთი სვეტი შეიძლება იყოს *გამოთვლადი (computed)*. ასეთ სვეტებში ეთითება ფორმულა, რომლის მიხედვით სვეტის მნიშვნელობა გამოითვლება.

სისტემური ცხრილები

სისტემური ცხრილები (*system tables*) შეიცავენ სერვერის მუშაობისათვის საჭირო ინფორმაციას. ამიტომ, უფლება არ გვაქვს შევცვალოთ ამ ცხრილებში მოთავსებული მონაცემები. აქედან გამომდინარე, ამ ცხრილების მიმართ შეგვიძლია მხოლოდ SELECT ბრძანების გამოყენება. სისტემურ ცხრილებში მონაცემების შესაცვლელად უნდა გამოვიყენოთ შენახული პროცედურები.

დროებითი ცხრილები

დროებითი ცხრილები (*temporary tables*) განკუთვნილია მონაცემების დროებით შესანახად და მოთავსებულია tempdb სისტემურ მონაცემთა ბაზაში. თუ დროებითი ცხრილის შექმნის დროს აშკარად მივუთითებთ მონაცემთა ბაზის სახელს, რომელშიც ის უნდა შეიქმნას, ცხრილი მაინც tempdb სისტემურ მონაცემთა ბაზაში შეიქმნება. დროებითი ცხრილები ავტომატურად იშლება შეერთების დახურვის დროს.

არსებობს ორი სახის დროებითი ცხრილი:

1. *ლოკალური დროებითი ცხრილი* (*local temporary tables*). ასეთი ცხრილის სახელი ერთი # სიმბოლოთი იწყება, მაგალითად, #LokaluriDroebitiCxrili. ლოკალური დროებითი ცხრილის შესაქმნელად საკმარისია ცხრილის სახელის წინ # სიმბოლოს მითითება. ლოკალური დროებითი ცხრილები ხილულია მხოლოდ იმ შეერთების შიგნით, რომელშიც ისინი შეიქმნა. შეერთების დახურვის დროს ლოკალური დროებითი ცხრილი ავტომატურად იშლება. თუ ლოკალური დროებითი ცხრილი შეიქმნა შენახული პროცედურის მუშაობისას, მაშინ ამ პროცედურიდან გამოსვლის შემდეგ ეს ცხრილი ავტომატურად წაიშლება.
2. *გლობალური დროებითი ცხრილი* (*global temporary table*). გლობალური დროებითი ცხრილის სახელი ## სიმბოლოებით იწყება, მაგალითად, ##GlobaluriDroebitiCxrili. გლობალური დროებითი ცხრილის შესაქმნელად საკმარისია ცხრილის სახელის წინ ## სიმბოლოების მითითება. ასეთ ცხრილთან მიმართვა შეგვიძლია ნებისმიერი შეერთებიდან. ასეთი ცხრილები განკუთვნილია მონაცემების გასაცვლელად სხვადასხვა პროგრამას შორის. გლობალური დროებითი ცხრილი არსებობს იმ შეერთების დამთავრებამდე, რომელშიც ეს ცხრილი შეიქმნა. მისი წაშლა და შეცვლა შეგვიძლია სხვადასხვა შეერთებიდან.

დროებითი ცხრილი ყოველთვის იქმნება მიმდინარე სერვერზე tempdb მონაცემთა ბაზაში. სერვერის გაჩერების შემთხვევაში ორივე სახის დროებითი ცხრილი ავტომატურად იშლება.

მონაცემთა ტიპები

ცხრილის თითოეულ სვეტს განსაზღვრული ტიპი აქვს. ის მიუთითებს თუ რა მნიშვნელობები იქნება მოთავსებული ცხრილის სვეტებში. ცხრილში 2.1 მოყვანილია სერვერის მონაცემების ტიპები (სისტემური ტიპები).

ცხრილი 2.1. სერვერის მონაცემთა ტიპები.

ტიპი	აღწერა
BINARY(N)	ფიქსირებული სიგრძის ორობითი მონაცემები 8000 ბაიტამდე
BIGINT	იკავებს 8 ბაიტს და ინახავს მთელ რიცხვს დიაპაზონში $-2^{63} \div 2^{63}-1$
BIT	ერთი ბიტი. იღებს მნიშვნელობას 0 ან 1
CHAR(N)	ფიქსირებული სიგრძის სტრიქონული ტიპი სიგრძით 8000 სიმბოლომდე
DATE	თარიღი 1 წლის 1 იანვრიდან 9999 წლის 31 დეკემბრამდე
DATETIME	თარიღი და დრო 1753 წლის 1 იანვრიდან 9999 წლის 31 დეკემბრამდე
DECIMAL	მოცემული სიზუსტის წილადი დიაპაზონში $2^{38}+1 \div 2^{38}-1$
FLOAT	წილადი დიაპაზონში -1,79E+308 ÷ -1,79E+308
IMAGE	ორობითი მონაცემები სიგრძით 2 გიგაბაიტამდე
INT	იკავებს 4 ბაიტს და ინახავს მთელ რიცხვს დიაპაზონში $-2^{31} \div 2^{31}-1$
MONEY	ფულის ერთეული დიაპაზონში $-2^{63} \div 2^{63}-1$. იკავებს 8 ბაიტს
NCHAR(N)	ფიქსირებული სიგრძის Unicode სიმბოლოების სტრიქონი 8000 სიმბოლომდე
NTEXT	1 გიგაბაიტამდე ზომის ცვლადი სიგრძის Unicode სიმბოლოების სტრიქონი
NVARCHAR(N)	ცვლადი სიგრძის Unicode სიმბოლოების სტრიქონი ზომით 8000 სიმბოლომდე
NUMERIC	იგივეა, რაც DECIMAL
REAL	მცურავწერტილიანი მნიშვნელობა დიაპაზონში -3,40E+38 ÷ -3,40E+38
SMALLDATETIME	თარიღი და დრო 1900 წლის 1 იანვრიდან 2079 წლის 6 ივნისამდე
SMALLINT	იკავებს 2 ბაიტს და ინახავს მთელ რიცხვს დიაპაზონში $-2^{15} \div 2^{15}-1$
SMALLMONEY	ფულის ერთეული დიაპაზონში $-2^{31} \div 2^{31}-1$. იკავებს 4 ბაიტს
SQL_VARIANT	ინახავს სხვადასხვა ტიპის მონაცემს. მაგალითად, ამ ტიპის სვეტში ერთდროულად შეგვიძლია შევინახოთ INT, DATETIME, FLOAT და NVARCHAR(50) ტიპის მონაცემები
SYSNAME	მომხმარებლის მიერ შექმნილი ტიპი
TABLE	ეს ტიპი გამოიყენება რთულ მონაცემთა ნაკრებების შესანახად. ეს ტიპი არ გამოიყენება სვეტების მიმართ. ის გამოიყენება Transact-SQL ცვლადების მიმართ აგრეთვე, ფუნქციებიდან დაბრუნებული მნიშვნელობების მიმართ
TEXT	ცვლადი სიგრძის სტრიქონი სიმბოლოების მაქსიმალური რაოდენობით $2^{31}-1$, რომელიც არ შეიცავს Unicode სიმბოლოებს
TIMESTAMP	დროითი შტამპია. ამ ტიპის სვეტის მნიშვნელობა ავტომატურად იცვლება სტრიქონის ყოველი ცვლილებისას.
TINYINT	იკავებს 1 ბაიტს და ინახავს მთელ რიცხვს დიაპაზონში 0 ÷ 255
VARBINARY(N)	ცვლადი სიგრძის ორობითი მონაცემები 8000 ბაიტამდე
VARCHAR(N)	ცვლადი სიგრძის სტრიქონი 8000 სიმბოლომდე, რომელიც არ შეიცავს Unicode სიმბოლოებს
UNIQUEIDENTIFIER	გამოიყენება გლობალური უნიკალური იდენტიფიკატორების (Global Unique Identifier, GUID) შესანახად

მომხმარებლების ტიპები

მონაცემთა მომხმარებლის ტიპები მომხმარებლების მიერ შექმნილი ტიპებია. ისინი იქმნება მონაცემების სისტემური ტიპების საფუძველზე. ჩვენი საკუთარი ტიპები შეგვიძლია შევქმნათ მაშინ, როცა რამდენიმე ცხრილში გვინდა ერთტიპური მნიშვნელობების შენახვა. მაგალითად, მომხმარებლის ტიპი შეგვიძლია გამოვიყენოთ სვეტში პასპორტის ნომრისა და სერიის, მობილური ტელეფონის ნომრის შესანახად და ა.შ.

მომხმარებლის ტიპის სახელი უნიკალური უნდა იყოს მფლობელის ფარგლებში. სხვადასხვა მფლობელს შეუძლია შექმნას ერთნაირი სახელის მქონე მომხმარებლის ტიპი.

სვეტის მნიშვნელობები უნდა შეესაბამებოდეს ამ სვეტისთვის დანიშნულ ტიპს, თუმცა სერვერს შეუძლია არაცხადად გარდაქმნას მონაცემები საწყისი ტიპიდან სვეტისთვის დანიშნულ ტიპამდე. მაგალითად, სერვერი უზრუნველყოფს INT ტიპის გარდაქმნას DECIMAL ტიპად.

წარმოდგენები

წარმოდგენა (*view*) არის ვირტუალური ცხრილი, რომლის შემცველობა განისაზღვრება SELECT მოთხოვნით. ცხრილის მსგავსად წარმოდგენა შედგება სვეტებისა და სტრიქონებისაგან. წარმოდგენა მონაცემებს არ შეიცავს, არამედ წარმოგვიდგენს ერთ ან მეტ ცხრილში მოთავსებულ მონაცემებს. ინფორმაცია, რომელსაც ეკრანზე ვხედავთ წარმოდგენის საშუალებით, არ ინახება მონაცემთა ბაზაში, როგორც დამოუკიდებელი ობიექტი.

ფიზიკურად წარმოდგენა რეალიზებულია SELECT მოთხოვნის სახით, რომლის საფუძველზე სრულდება მონაცემების ამორჩევა ერთი ან მეტი ცხრილიდან ან წარმოდგენიდან. უმარტივესი წარმოდგენა, შექმნილი ერთი ცხრილის ბაზაზე, არ შეიცავს ფილტრებს და შედგება მონაცემთა ისეთივე ნაკრებისაგან, რისგანაც საწყისი ცხრილი. უფრო რთული წარმოდგენა შეიცავს რამდენიმე ცხრილის გაფილტრულ ინფორმაციას.

წარმოდგენასთან მიმართვა შესაძლებელია სახელის მიხედვით. წარმოდგენა შეგვიძლია გამოვიყენოთ:

- ცხრილის გარკვეულ სტრიქონებთან მომხმარებლების მიმართვის შესაზღუდად.
- ცხრილის გარკვეულ სვეტებთან მომხმარებლების მიმართვის შესაზღუდად.
- სხვადასხვა ცხრილის სვეტების მონაცემების წარმოსადგენად ერთი ობიექტის სახით.
- ინფორმაციის სანახავად, რომელიც მიიღება მონაცემების გარდაქმნის შედეგად.

შენახული პროცედურები

შენახული პროცედურა (*stored procedure*) არის ერთ მოდულში გაერთიანებული ბრძანებების ჯგუფი, რომელსაც სახელი აქვს. ეს ბრძანებები კომპილირდება და სრულდება, როგორც ერთი მთლიანი.

სერვერზე ინახება სისტემური შენახული პროცედურების დიდი რაოდენობა. მათი სახელები იწყება sp_ პრეფიქსით. ისინი გამოიყენება სერვერის კონფიგურირებისა და მართვისათვის, სისტემურ მონაცემთა ბაზებსა და ცხრილებში მონაცემების შესაცვლელად და ა.შ. შესაძლებელია საკუთარი შენახული პროცედურების შექმნაც, რომლებიც მოთავსებული იქნება ჩვენს მონაცემთა ბაზაში. შენახული პროცედურის შესასრულებლად საკმარისია მისი სახელის მითითება, თუ საჭიროა პარამეტრებთან ერთად.

როცა შენახული პროცედურა პირველად სრულდება, სერვერი ქმნის მისი შესრულების გეგმას. შენახული პროცედურის განმეორებით შესრულებისას, სერვერი იყენებს მეხსიერებაში მოთავსებულ გეგმას, რაც ზრდის მის მწარმოებლურობას.

ტრიგერები

ტრიგერები (*triggers*) არის შენახული პროცედურის სპეციალური კლასი, რომლებიც ავტომატურად გაიშვება ცხრილებში მონაცემების დამატების, შეცვლის ან წაშლის დროს. ისინი სამ კატეგორიად იყოფა:

1. *შეცვლის ტრიგერები* (*UPDATE TRIGGER*);
2. *ჩამატების ტრიგერები* (*INSERT TRIGGER*);
3. *წაშლის ტრიგერები* (*DELETE TRIGGER*).

ერთი ცხრილისთვის შეიძლება განსაზღვრული იყოს თითოეული ტიპის რამდენიმე ტრიგერი. გარდა ამისა, ერთი ტრიგერიდან შეგვიძლია სხვა ტრიგერები გამოვიძახოთ. ასეთ შემთხვევაში გვაქვს *ჩადგმული ტრიგერები* (*nested triggers*).

ინდექსები

ინდექსი (*index*) არის ცხრილთან ან წარმოდგენასთან დაკავშირებული სტრუქტურა და განკუთვნილია შესაბამის ცხრილში ან წარმოდგენაში ინფორმაციის ძებნის დასაჩქარებლად. ინდექსი განისაზღვრება ერთი ან მეტი სვეტისთვის, რომლებსაც *ინდექსირებული სვეტები* ეწოდება. ინდექსი შეიცავს ინდექსირებული სვეტის ან სვეტების დახარისხებულ მნიშვნელობებს საწყისი ცხრილის ან წარმოდგენის შესაბამის სტრიქონზე მიმართვებთან ერთად.

მწარმოებლურობის ზრდა მიიღწევა სვეტის მონაცემების დახარისხების ხარჯზე. თუ სვეტი დაუხარისხებელია, მაშინ საჭირო მნიშვნელობის საპოვნელად მიმდევრობით უნდა გაისინჯოს ყველა მნიშვნელობა. როცა საჭიროა ინდექსირებულ სვეტში მნიშვნელობის პოვნა, მაშინ მისი ძებნა სრულდება ინდექსში. ინდექსების გამოყენება მნიშვნელოვნად ზრდის მონაცემების ძებნის მწარმოებლურობას, მაგრამ ითხოვს დამატებით სივრცეს მონაცემთა ბაზაში.

მთლიანობის შეზღუდვები

მთლიანობის შეზღუდვები (*constraints*) ეს მექანიზმია, რომელიც უზრუნველყოფს ჩვენ მიერ განსაზღვრულ პირობებთან (ან შეზღუდვებთან) მონაცემების შესაბამისობის ავტომატურ კონტროლს. მთლიანობის შეზღუდვები გამოიყენება მონაცემების მთლიანობის უზრუნველსაყოფად ლოგიკურ დონეზე.

დავუშვათ, გვაქვს ორი ცხრილი, რომლებიც ინახავს ლოგიკურად დაკავშირებულ მონაცემებს. მაშინ მათ შორის შეიძლება განვსაზღროთ კავშირი იმის გარანტირებისათვის, რომ მთავარი ცხრილიდან სტრიქონები არ წაიშლება, რომლებისთვისაც არსებობენ სტრიქონები დამოკიდებულ ცხრილში. სერვერი ავტომატურად ამოწმებს კავშირებს ცხრილებს შორის, არ გვაძლევს რა მონაცემების წაშლის უფლებას მთავარ ცხრილში მანამ, სანამ დამოკიდებული ცხრილიდან არ წაიშლება ყველა ბმული სტრიქონი. მთლიანობის შეზღუდვის მაგალითებია: თანამშრომლის საიდენტიფიკაციო ნომრის უნიკალურობის უზრუნველყოფა, ფირმის თანამშრომლების ასაკის კონტროლი, რომელიც არ უნდა იყოს მაგალითად, 25-ზე ნაკლები და ა.შ.

მთლიანობის შეზღუდვები შეიძლება გამოყენებული იყოს სვეტის ან ცხრილის დონეზე. მთლიანობის შეზღუდვები, რომლებიც სვეტის დონეზე გამოიყენება, მოქმედებს მხოლოდ ამ სვეტში შესატან მონაცემებზე. თუ მთლიანობის შეზღუდვა გამოიყენება

რამდენიმე სვეტის მიმართ, მაშინ შეზღუდვა მუშაობს ცხრილის დონეზე. მაგალითად, თუ PRIMARY KEY მთლიანობის შეზღუდვა შეიცავს ერთ სვეტს, მაშინ ის მუშაობს სვეტის დონეზე. თუ პირველადი გასაღები შედგება რამდენიმე სვეტისაგან, მაშინ ეს მთლიანობის შეზღუდვა მუშაობს ცხრილის დონეზე.

არსებობს მთლიანობის შეზღუდვის ხუთი ტიპი, რომლებიც განსხვავდებიან ფუნქციურობით და გამოყენებით:

1. *NULL მთლიანობის შეზღუდვა.* ის მოქმედებს სვეტისა და მომხმარებლის ტიპის დონეზე. მათთვის შეგვიძლია განვსაზღვროთ NULL ან NOT NULL მთლიანობის შეზღუდვა. შესაბამისად, სვეტში ნებადართული ან აკრძალული იქნება NULL მნიშვნელობის შენახვა. NULL არის სპეციალური მნიშვნელობა, რომელიც აღნიშნავს მნიშვნელობის არყოფნას. NULL არ არის იგივე, რაც ინტერვალის სიმბოლო, ნული ან ნულოვანი სიგრძის სტრიქონი. NULL მნიშვნელობა შეიძლება შეფასდეს როგორც UNKNOWN (უცნობი) და არ შეიძლება შეფასდეს როგორც TRUE ან FALSE.
2. *CHECK მთლიანობის შეზღუდვა* (*შემოწმება*). ის მოქმედებს სვეტის დონეზე და ზღუდავს სვეტში შესანახი მნიშვნელობების დიაპაზონს. მისი განსაზღვრისას შესატანი მონაცემებისათვის ეთითება ლოგიკური პირობა. მონაცემების შეტანის ან ჩამატების დროს შესატანი მნიშვნელობა მოთავსდება პირობაში. თუ შემოწმების შედეგია TRUE (ჭეშმარიტი), მაშინ მონაცემების შეცვლა ნებადართული იქნება. თუ შემოწმების შედეგია FALSE (მცდარი), მაშინ ცვლილებები აიკრძალება და გაიცემა შეტყობინება შეცდომის შესახებ. ერთი სვეტისთვის შეგვიძლია შევქმნათ რამდენიმე CHECK შეზღუდვა. მისი მითითება შეგვიძლია ცხრილის შექმნის დროს.
3. *UNIQUE მთლიანობის შეზღუდვა* (*უნიკალურობა*). ის მოქმედებს სვეტის დონეზე და იძლევა სვეტში მნიშვნელობების უნიკალურობის გარანტიას. ცხრილში არ იქნება ორი სტრიქონი, რომლებსაც მოცემულ სვეტში ერთნაირი მნიშვნელობები ექნება. პირველადი გასაღებისაგან განსხვავებით UNIQUE მთლიანობის შეზღუდვა უშვებს NULL მნიშვნელობის არსებობას.
4. *PRIMARY KEY* (*პირველადი გასაღები*). ის მოქმედებს სვეტის ან ცხრილის დონეზე. პირველადი გასაღები შეიძლება შედგებოდეს ერთი ან მეტი სვეტისაგან და არის სტრიქონის უნიკალური იდენტიფიკატორი ცხრილის ფარგლებში. თუ პირველადი გასაღები ერთი სვეტისაგან შედგება, მაშინ ამ სვეტისთვის უნდა იყოს დაყენებული UNIQUE შეზღუდვა, სვეტში მნიშვნელობების უნიკალურობის გარანტირებისათვის. თუ პირველადი გასაღები რამდენიმე სვეტისაგან შედგება, მაშინ თითოეულ სვეტში მნიშვნელობები შეიძლება გამეორდეს ანუ არ იყოს უნიკალური. მაგრამ, უნიკალური უნდა იყოს ამ სვეტების მნიშვნელობების კომბინაცია თითოეული სტრიქონისათვის. პირველად გასაღებში შემავალი არც ერთი სვეტისათვის არ უნდა იყოს დაყენებული NULL შეზღუდვა. ცხრილში შეგვიძლია შევქმნათ მხოლოდ ერთი პირველადი გასაღები.
5. *FOREIGN KEY* (*გარე გასაღები*). ის იქმნება ცხრილის დონეზე. დამოკიდებული ცხრილის გარე გასაღები უკავშირდება მთავარი ცხრილის პირველად გასაღებს. დამოკიდებულ ცხრილში არ შეიძლება სტრიქონის ჩასმა, თუ გარე გასაღებს არ აქვს შესაბამისი მნიშვნელობა მთავარ ცხრილში. მთავარი ცხრილიდან შეუძლებელია სტრიქონის წაშლა, თუ მასთან დაკავშირებულია თუნდაც ერთი სტრიქონი დამოკიდებულ ცხრილში. მთავარი ცხრილიდან სტრიქონის წაშლის წინ აუცილებელია წინასწარ წავშალოთ დამოკიდებული ცხრილის ყველა სტრიქონი.

ნაგულისხმევი მნიშვნელობები

ნაგულისხმევი მნიშვნელობა (*default, ავტომატური მნიშვნელობა, გაჩუმებითი მნიშვნელობა*) არის ის მნიშვნელობა, რომელიც ავტომატურად მიენიჭება ცხრილის სვეტს სტრიქონის ჩასმის დროს, თუ ჩასმის ბრძანებაში აშკარად არ არის მითითებული ამ სვეტის მნიშვნელობა. ნაგულისხმევი მნიშვნელობები როგორ სვეტისათვის, ისე მომხმარებლების ტიპისათვის განისაზღვრება უშუალოდ მათი შექმნისას.

ნაგულისხმევი მნიშვნელობა შეიძლება იყოს მუდმივა, ჩადგმული ფუნქციის ან მათემატიკური გამოსახულების მიერ გაცემული შედეგი.

ავტომატური მნიშვნელობები და მთლიანობის შეზღუდვები ერთმანეთთან კონფლიქტში არ უნდა იყოს. მაგალითად, თუ სვეტისთვის განსაზღვრულია შეზღუდვა, რომლის მიხედვით სვეტს უნდა მიენიჭოს 20-ზე მეტი მნიშვნელობა, მაშინ ავტომატური მნიშვნელობა არ შეიძლება იყოს 20-ზე ნაკლები.

ფუნქციები

ფუნქცია არის კონსტრუქცია, რომელიც შეიცავს ხშირად გამოყენებად კოდს და აქვს სახელი. ფუნქცია მონაცემებზე გარკვეულ მოქმედებებს ასრულებს და აბრუნებს კონკრეტულ მნიშვნელობას. მისი გამოძახება სახელის მიხედვით ხდება. არსებობს ფუნქციების ორი ტიპი:

1. *ჩადგმული ფუნქციები* (*built-in functions*). ისინი პროგრამირების გარემოს შემადგენელი ნაწილია და წინასწარ განსაზღვრულ მოქმედებებს ასრულებს. ასეთი ფუნქციებია: MAX, SUM და ა.შ.
2. *მომხმარებლის ფუნქციები* (*user-defined functions*). ასეთი ფუნქციის სახელსა და კოდს მომხმარებელი განსაზღვრავს.

სისტემური მონაცემთა ბაზები

სერვერს აქვს ოთხი სისტემური მონაცემთა ბაზა, რომლებიც სერვერის დაყენების დროს იქმნება. ესენია: master, msdb, model და tempdb. ისინი ინახავენ სისტემურ მონაცემებს და ემსახურება შიგა ამოცანების გადაწყვეტას.

როგორც სისტემური, ისე მომხმარებლების მიერ შექმნილი მონაცემთა ბაზები, შეიცავენ სისტემურ ცხრილებსა და წარმოდგენებს. ისინი შეიცავენ ინფორმაციას, რომელიც განსაზღვრავს შესაბამისი მონაცემთა ბაზის სტრუქტურას და ორგანიზებას.

სისტემურ ცხრილებთან მუშაობა UPDATE, INSERT და DELETE ბრძანებების საშუალებით აკრძალულია. დასაშვებია SELECT ბრძანების გამოყენება. ამ ცხრილებში მონაცემების შეცვლა შესაძლებელია მხოლოდ სისტემური შენახული პროცედურების გამოყენებით.

master სისტემური მონაცემთა ბაზა

master მონაცემთა ბაზა შეიცავს სერვერის მთელ სისტემურ ინფორმაციას, აგრეთვე, ინფორმაციას სხვა მონაცემთა ბაზებისა და მათი პირველადი ფაილების ადგილმდებარეობის შესახებ. master მონაცემთა ბაზა შემდეგი ფაილებისაგან შედგება: Master.mdf (მონაცემების ფაილი) და Mastlog.ldf (ტრანზაქციების ჟურნალის ფაილი). ორივე ფაილი ინახება \Data კატალოგში (C:\Program Files\Microsoft SQL Server\MSSQL.3\MSSQL\Data\).

დაუშვებელია master სისტემურ მონაცემთა ბაზაში მომხმარებლის ობიექტების შექმნა.

tempdb სისტემური მონაცემთა ბაზა

tempdb მონაცემთა ბაზა შეიცავს დროებით ობიექტებს, როგორიცაა ცხრილები, შენახული პროცედურები, ცვლადები, კურსორები და ა.შ. მასში, ინახება, როგორც სისტემური, ისე მიმხმარებლის მიერ შექმნილი ობიექტები. სერვერის ყოველი გაშვების დროს tempdb მონაცემთა ბაზა ხელახლა იქმნება და ყოველთვის იშლება სერვერის გაჩერების დროს. მონაცემთა ბაზის ობიექტები ინახება მხოლოდ ერთი სეანსის განმავლობაში.

tempdb მონაცემთა ბაზა არის გლობალური კურსორი, რომელიც ავტომატურად მისაწვდომია ყველა მომხმარებლისათვის (კურსორებს მე-10 თავში განვიხილავთ). შესაქმნელი დროებითი ობიექტები შეიძლება იყოს როგორც *ლოკალური*, ისე *გლობალური*. ლოკალური ობიექტი მისაწვდომია მხოლოდ მისი შემქმნელი მომხმარებლისთვის, გლობალური. კი - ყველა მომხმარებლისათვის. ლოკალური ობიექტები მოქმედებენ მხოლოდ სეანსის, შენახული პროცედურის, ტრიგერის ან ბრძანებების პაკეტის ფაგრლებში. დროებითი ობიექტის შემქმნელი სტრუქტურიდან გამოსვლისას ეს ობიექტი მაშინვე წაიშლება, როგორც კი მომხმარებელი დაამთავრებს სერვერთან მუშაობას. ლოკალური დროებითი ცხრილის სახელი # სიმბოლოთი იწყება, გლობალური დროებითი ცხრილის სახელი კი - ## სიმბოლოებით.

tempdb მონაცემთა ბაზა ორი ფაილისაგან შედგება: tempdb.mdf (მონაცემების ფაილი) და templogldf (ტრანზაქციების ჟურნალი).

model სისტემური მონაცემთა ბაზა

სერვერზე ახალი მონაცემთა ბაზის შექმნა ხდება მასში model სისტემური მონაცემთა ბაზის ობიექტების გადაწერის გზით. model მონაცემთა ბაზა მოთავსებულია \Data კატალოგში და ორი ფაილისგან შედგება: model.mdf (მონაცემების ფაილი) და model.ldf (ტრანზაქციების ჟურნალი).

msdb სისტემური მონაცემთა ბაზა

msdb სისტემური მონაცემთა ბაზა გამოიყენება SQL Server Agent სამსახურის მიერ მოვლენების (alerts), ამოცანებისა (jobs) და ოპერატორების (operators) რეგისტრირების დაგეგმვისათვის. msdb მონაცემთა ბაზა ინახავს მთელ ინფორმაციას, რომელიც ეხება ადმინისტრირების ავტომატიზებასა და სერვერის მართვას. msdb ბაზა ორი ფაილისგან შედგება: msdbdata.mdf (მონაცემების ფაილი) და msdblog.ldf (ტრანზაქციების ჟურნალი).

ფაილები და ფაილების ჯგუფები

მონაცემთა ბაზის შესანახად სამი ტიპის ფაილი გამოიყენება:

– Primary. ძირითადი ფაილია, რომელიც შეიცავს სისტემურ ინფორმაციას თვით მონაცემთა ბაზისა და მისი ობიექტების შესახებ. მასში მოთავსებულია სისტემური ცხრილები და მონაცემთა ბაზის ობიექტების აღწერა. ძირითად ფაილში შეიძლება, აგრეთვე ინახებოდეს მონაცემებიც. ნებისმიერ მონაცემთა ბაზას აქვს primary ტიპის ფაილი, ამასთან მონაცემთა ბაზაში მხოლოდ ერთი ფაილი შეიძლება იყოს ამ ტიპის. primary ტიპის ფაილებს აქვთ .mdf (Master Data File) გაფართოება.

– Secondary. მონაცემების არაძირითადი ფაილია, რომელიც არ შეიცავს სისტემურ

ინფორმაციას და განკუთვნილია მხოლოდ მონაცემების შესანახად. მონაცემები, რომლებიც არ დაეტია ძირითად ფაილში, მოთავსდება მონაცემების არაძირითადი ფაილებში. მონაცემთა ბაზას შეიძლება საერთოდ არ ჰქონდეს secondary ტიპის ფაილი ან ჰქონდეს რამდენიმე. secondary ტიპის ფაილებს აქვთ .ndf (Not Master Data File) გაფართოება.

– Transaction Log. ტრანზაქციების ჟურნალის ფაილია. ის გამოიყენება მონაცემთა ბაზაში შესრულებული ტრანზაქციების შესახებ ინფორმაციის შესანახად. ნებისმიერ მონაცემთა ბაზას აქვს ტრანზაქციების ჟურნალის მინიმუმ ერთი ფაილი. transaction log ტიპის ფაილებს აქვთ .ldf (Log Data File) გაფართოება.

მონაცემთა ბაზაში გამოყენებულ თითოეულ ფაილს ორი სახელი აქვს:

1. *ფაილის ლოგიკური სახელი* (*Logical File Name*). ის გამოიყენება Transact-SQL ბრძანებებში კონკრეტულ ფაილთან მიმართვისათვის.
2. *ფაილის სახელი ოპერაციულ სისტემაში* (*OS File Name*). ეს არის ფაილის ფიზიკური სახელი. ამ სახელით ინახება ფაილი დისკზე.

მონაცემთა ბაზის ყველა ფაილის ზომა შეიძლება ავტომატურად გაიზარდოს. ეს შესაძლებლობა ფაილის შექმნის დროს განისაზღვრება. ნაზრდის ზომა შეგვიძლია განვსაზღვროთ პროცენტებში (ფაილის საწყისი ზომიდან) ან მეგაბაიტებში. დამატებით, შეგვიძლია მივუთითოთ ფაილის მაქსიმალური ზომა. თუ მაქსიმალური ზომა არ არის მითითებული, მაშინ ფაილის ზომა გაიზრდება მანამ, სანამ არის თავისუფალი სივრცე დისკზე.

ადმინისტრირებისა და მონაცემთა ბაზის ობიექტების ფიზიკური განლაგების მართვის გაადვილების მიზნით უმჯობესია ფაილები საფაილო ჯგუფებში მოვათავსოთ. არსებობს შემდეგი ტიპის საფაილო ჯგუფები:

– *ფაილების ძირითადი ჯგუფი* (*Primary File Group*). ფაილების ეს ჯგუფი შეიცავს primary ტიპის ფაილსა და ყველა ფაილს, რომლებიც არ არის ჩართული სხვა ჯგუფში. მონაცემთა ბაზას შეიძლება ჰქონდეს ფაილების მხოლოდ ერთი ძირითადი ჯგუფი.

– *მომხმარებლების მიერ შექმნილი ფაილების ჯგუფი* (*User-defined File Group*). ამ ჯგუფებში ჩართულია ყველა ფაილი, რომლებიც მითითებულია FILEGROUP პარამეტრში მონაცემთა ბაზის შექმნის (CREATE DATABASE) ან შეცვლის (ALTER DATABASE) დროს. მონაცემთა ბაზაში შეიძლება შევქმნათ მომხმარებლების მიერ განსაზღვრული ფაილების რამდენიმე ჯგუფი ფაილების ნებისმიერი შემადგენლობით.

– *ფაილების ნაგულისხმევი ჯგუფი* (*Default File Group*). მონაცემთა ბაზაში ფაილების ერთ-ერთი ჯგუფი ხდება ნაგულისხმევი. თუ მონაცემთა ბაზის შექმნის დროს არ არის მითითებული ფაილების ასეთი ჯგუფი, მაშინ ფაილების ძირითადი ჯგუფი ხდება ნაგულისხმევი. თუ მონაცემთა ბაზის ობიექტის (ცხრილის ან სვეტის) შექმნის დროს აშკარად არ არის მითითებული, ფაილების რომელ ჯგუფს ეკუთვნის ის, მაშინ ეს ობიექტი შეიქმნება ფაილების ნაგულისხმევ ჯგუფში. მონაცემთა ბაზის მფლობელმა შეიძლება ფაილების ნებისმიერი ჯგუფი დანიშნოს ფაილების ნაგულისხმევ ჯგუფად. ფაილების მხოლოდ ერთი ჯგუფი შეიძლება იყოს ნაგულისხმევი.

ნებისმიერი ფაილი შეიძლება ჩართული იყოს ფაილების მხოლოდ ერთ ჯგუფში. არ შეიძლება ფაილების ჯგუფის განსაზღვრა ტრანზაქციების ჟურნალის ფაილებისთვის. თუ მონაცემთა ბაზაში არ იქმნება ფაილების ჯგუფები, მაშინ ყველა ობიექტი მოთავსდება ფაილების ძირითად ჯგუფში, რომელიც ავტომატურად იქმნება.

მონაცემთა ბაზების შექმნა, წაშლა და სახელის შეცვლა

სერვერთან შეერთებისას კავშირი მყარდება კონკრეტულ მონაცემთა ბაზასთან. მას

მიმდინარე მონაცემთა ბაზა ეწოდება. ჩვენ, ჩვეულებრივ, ვუკავშირდებით ადმინისტრატორის მიერ ჩვენთვის განსაზღვრულ მონაცემთა ბაზას. აუცილებლობისას, შეგვიძლია გადავერთოთ სხვა მონაცემთა ბაზაზე, ანუ მიმდინარე გავხადოთ სხვა მონაცემთა ბაზა. ამისათვის გამოიყენება USE ბრძანება. მისი სინტაქსია:

USE *მონაცემთა_ბაზის_სახელი*

მაგალითი. მიმდინარე გავხადოთ Baza1 მონაცემთა ბაზა. მოთხოვნას აქვს სახე:

```
USE Baza1;
```

Baza1 მონაცემთა ბაზა მიმდინარე იქნება მანამ, სანამ არ შევასრულებთ სხვა USE ბრძანებას.

მონაცემთა ბაზის შექმნა

მონაცემთა ბაზის შექმნა შესაძლებელია როგორც Transact-SQL-ის ბრძანებების, ისე გრაფიკული ინტერფეისის საშუალებით. მონაცემთა ბაზის შესაქმნელად უნდა მივუთითოთ მისი სახელი, მფლობელის სახელი, (ეს იქნება მონაცემთა ბაზის შემქმნელი მომხმარებელი), ზომა, ფაილები და ფაილების ჯგუფები, რომლებისგანაც იქნება შემდგარი შესაქმნელი მონაცემთა ბაზა.

მონაცემთა ბაზის შექმნის წინ უნდა გავითვალისწინოთ შემდეგი მოსაზრებები:

– მონაცემთა ბაზის შექმნა შეუძლიათ სერვერის sysadmin და dbcreator ფიქსირებული როლების წევრებს, თუმცა სხვა მომხმარებლებსაც შეიძლება მიეცეს მონაცემთა ბაზების შექმნის უფლება;

– მონაცემთა ბაზის შემქმნელი მომხმარებელი ავტომატურად ხდება მისი მფლობელი;

– მონაცემთა ბაზის სახელი უნდა აკმაყოფილებდეს ობიექტების სახელდების მოთხოვნებს (სახელდების წესები განხილულია მე-4 თავში).

მონაცემთა ბაზის შექმნის ეტაპს წინ უძღვის დაგეგმვის ფაზა, რომელიც მოიცავს ფაილებისა და ფაილების ჯგუფების შემადგენლობის დაპროექტებას, რომლისგანაც იქნება შემდგარი მონაცემთა ბაზა. ესაა ფიზიკური ნაწილის დაგეგმვა. მონაცემთა ბაზის შექმნის პროცესის ნაწილია, აგრეთვე ლოგიკური სტრუქტურის დაპროექტება, რომელშიც შედის ცხრილების დაპროექტება ნორმალიზების გათვალისწინებით. მონაცემთა ბაზის შექმნისას master მონაცემთა ბაზის sys.databases სისტემურ წარმოდგენაში შეიტანება შესაბამისი მონაცემები, რომელთა ნახვა შეგვიძლია შემდეგი მოთხოვნის გამოყენებით:

```
SELECT *
FROM sys.databases
ORDER BY name;
```

მონაცემთა ბაზის შესაქმნელად გამოიყენება CREATE DATABASE ბრძანება. მისი სინტაქსია:

```
CREATE DATABASE მონაცემთა_ბაზის_სახელი
[ ON [ PRIMARY ] [ <ფაილის_განსაზღვრა> [ ,...n] ] [ , <ფაილების_ჯგუფი> [ ,...n ] ] ]
[ LOG ON { <ფაილის_განსაზღვრა> [,...n] } ]
[ FOR ATTACH ]
```

განვიხილოთ არგუმენტების დანიშნულება.

– *მონაცემთა_ბაზის_სახელი*. შესაქმნელი მონაცემთა ბაზის სახელია. თუ ის შეიცავს ინტერვალებს ან სხვა დაუშვებელ სიმბოლოებს, მაშინ სახელი უნდა მოვათავსოთ შემზღუდავ სიმბოლოებში (კვადრატული ფრჩხილები). მონაცემთა ბაზის სახელი უნდა იყოს უნიკალური

სერვერის ფარგლებში. ის არ უნდა აღემატებოდეს 128 სიმბოლოს. თუ ტრანზაქციების ჟურნალის სახელი მითითებული არ არის, მაშინ სერვერი ტრანზაქციების ჟურნალის სახელად იღებს მონაცემთა ბაზის სახელს და მას უმატებს _Log სიმბოლოებს. ამ შემთხვევაში, მონაცემთა ბაზის სახელი არ უნდა აღემატებოდეს 123 სიმბოლოს.

– ON. მითითებს, რომ იწყება მონაცემთა ბაზის ფაილების განსაზღვრა.

– PRIMARY. მიუთითებს, რომ იწყება მონაცემთა ბაზის პირველადი ფაილის აღწერა. მონაცემთა ბაზაში მხოლოდ ერთი ფაილი შეიძლება იყოს განსაზღვრული როგორც პირველადი. თუ პირველადი ფაილი აშკარად არ არის განსაზღვრული, მაშინ პირველად ფაილად გამოყენებული იქნება <*ფაილის_განსაზღვრა*> კონსტრუქციაში მითითებული პირველი ფაილი. პირველადი ჯგუფი დაინიშნება ფაილების *ავტომატურ ჯგუფად* (*default file group*) და მასში ჩართული იქნება ყველა ფაილი, რომლებისთვისაც აშკარად არ არის მითითებული *მომხმარებლების ფაილების ჯგუფი* (*user file group*).

– LOG ON. განსაზღვრავს ტრანზაქციების ჟურნალის ფაილებს. თუ ეს არგუმენტი არ არის მითითებული, მაშინ სერვერი ავტომატურად ქმნის ტრანზაქციების ჟურნალის ერთ ფაილს, რომლის სახელი იქმნება მონაცემთა ბაზის სახელისათვის _Log სიმბოლოების დამატების გზით.

– FOR ATTACH. გამოიყენება მაშინ, როცა უნდა შესრულდეს სერვერთან მონაცემთა ბაზის მიერთება (attach). ამ შემთხვევაში უნდა არსებობდეს შესაბამისი ფაილები. მონაცემთა ბაზის მისაერთებლად საკმარისია მხოლოდ მონაცემთა ბაზის პირველადი ფაილის ადგილმდებარეობის მითითება. ინფორმაცია მონაცემთა ბაზის სხვა ფაილების (მეორადი და ტრანზაქციების ჟურნალის) ადგილმდებარეობის შესახებ ინახება მონაცემთა ბაზის პირველად ფაილში. მაგრამ, თუ მონაცემთა ბაზის ფაილების ადგილმდებარეობა მონაცემთა ბაზასთან შეერთების გაწყვეტის შემდეგ შეიცვალა, მაშინ საჭირო გახდება მონაცემთა ბაზის თითოეულ ფაილთან მიმართვისათვის სრული გზის მითითება. მონაცემთა ბაზის მიერთება შეიძლება შესრულდეს, აგრეთვე sp_attach_db სისტემური შენახული პროცედურის გამოყენებით, რომელიც პირდაპირ მუშაობს sys.databases სისტემურ წარმოდგენასთან და მასში საჭირო ცვლილებებს ასრულებს.

<*ფაილის_განსაზღვრა*> კონსტრუქციის სინტაქსია:

<*ფაილის_განსაზღვრა*> ::=
([NAME = *ფაილის_ლოგიკური_სახელი*,] FILENAME = '*ფაილის_ფიზიკური_სახელი*'
[, SIZE = ზომა] [, MAXSIZE = { *მაქსიმალური_ზომა* | UNLIMITED }]
[, FILEGROWTH = *ნაზრდი*]) [,...n]

განვიხილოთ არგუმენტების დანიშნულება.

– NAME = *ფაილის_ლოგიკური_სახელი*. ფაილის ლოგიკური სახელია. ის უნდა იყოს უნიკალური მონაცემთა ბაზის ფარგლებში.

– FILENAME = '*ფაილის_ფიზიკური_სახელი*'. შეიცავს ფაილის ფიზიკურ სახელსა და გზას, რომელიც შექმნილი იქნება დისკზე. ეს სახელი ფაილს ექნება ოპერაციული სისტემის დონეზე.

– SIZE = *ზომა*. მიუთითებს ფაილის საწყის ზომას მეგაბაიტებში (Mb). თუ ფაილის ზომა არ არის მითითებული, მაშინ ის აიღება 3 მეგაბაიტის ტოლი (ნაგულისხმევი მნიშვნელობა). ფაილის ზომა უნდა იყოს მთელი რიცხვი.

– FILEGROWTH = *ნაზრდი*. როგორც აღვნიშნეთ, შესაძლებელია მონაცემთა ბაზის ზომის ავტომატურად გაზრდა, რაც ხორციელდება მონაცემთა ბაზაში შემავალი ფაილების ზომების გაზრდით. სერვერი საშუალებას გვაძლევს ვაკონტროლოთ მონაცემთა ბაზის თითოეული ფაილის ზომის ზრდა სხვა ფაილებისაგან დამოუკიდებლად. არგუმენტი განსაზღვრავს ნაზრდის ზომას ფაილისათვის. ნაზრდის ზომა შეიძლება მივუთითოთ მეგაბაიტებში ან

პროცენტებში (ფაილის საწყისი ზომიდან). ავტომატურად ნაზრდის ზომა აიღება 1 მეგაბაიტის ტოლი (ნაგულისხმევი მნიშვნელობა). თუ ეს არგუმენტი მითითებული არ არის, მაშინ ფაილის ზომა გაიზრდება საწყისი ზომის 10%-ით. ფაილის საწყისი ზომა და ნაზრდის ზომა ჯამურად არ უნდა აღემატებოდეს ფაილის მაქსიმალურ ზომას.

– MAXSIZE = { *მაქსიმალური_ზომა* | UNLIMITED }. ფაილის მაქსიმალური ზომა ეთითება მეგაბაიტებში. ერთი ფაილის ზომის შეზღუდვა არ ზღუდავს მონაცემთა ბაზის ზომის ზრდას, რადგან მონაცემთა ბაზის ზომა შეიძლება გაიზარდოს სხვა ფაილების ზომის ზრდის ხარჯზე. თუ ფაილის ზომა არ უნდა შეიზღუდოს, მაშინ უნდა მივუთითოთ UNLIMITED მნიშვნელობა (ნაგულისხმევი მნიშვნელობა) ან საერთოდ გამოვტოვოთ MAXSIZE არგუმენტი.

ფაილები, რომლებიც არ ეკუთვნიან არც ერთ ჯგუფს, ჩართული იქნება ფაილების ავტომატურ ჯგუფში, რომელსაც თავდაპირველად წარმოადგენს ფაილების პირველადი ჯგუფი. თუ საჭიროა ფაილების დამატებითი ჯგუფების შექმნა, მაშინ უნდა გამოვიყენოთ <*ფაილების_ჯგუფი*> კონსტრუქცია. მისი სინტაქსია:

<*ფაილების_ჯგუფი*> ::=

FILEGROUP *ფაილების_ჯგუფის_სახელი* <*ფაილის_განსაზღვრა*> [,...n]

ფაილების_ჯგუფის_სახელი არგუმენტის შემდეგ ეთითება ჯგუფში მოსათავსებელი ფაილების სია.

მაგალითები.

ა. შევქმნათ Shekveta მონაცემთა ბაზა. თუ Shekveta ბაზა არსებობს, მაშინ ის წავშალოთ და შემდეგ ხელახლა შევქმნათ ნაგულისხმევი მნიშვნელობებით. მოთხოვნას აქვს სახე:

```
USE Master;
-- თუ Shekveta ბაზა არსებობს, მაშინ ის წაიშლება
IF DB_ID (N'Shekveta') IS NOT NULL
DROP DATABASE Shekveta;
-- Shekveta ბაზის შექმნა
CREATE DATABASE Shekveta;
```

მაგალითში გამოყენებულია DB_ID ფუნქცია, რომელიც მონაცემთა ბაზის საიდენტიფიკაციო ნომერს გასცემს. თუ ეს ნომერი არ არის ცარიელი, მაშინ შესაბამისი მონაცემთა ბაზა არსებობს და ის წაიშლება DROP ბრძანების გამოყენებით. DB_ID ფუნქციის სინტაქსია:

DB_ID('*მონაცემთა_ბაზის_სახელი*')

მოყვანილ მოთხოვნაში გაიცემა Shekveta მონაცემთა ბაზის იდენტიფიკატორი:

```
SELECT DB_ID(N'Shekveta') AS [მონაცემთა ბაზის იდენტიფიკატორი];
```

შედეგი:

	მონაცემთა ბაზის იდენტიფიკატორი
1	7

მონაცემთა ბაზისა და შესაბამისი საიდენტიფიკაციო ნომრების სია ინახება master სისტემური მონაცემთა ბაზის sysdatabases წარმოდგენაში.

ბ. შევქმნათ Baza2 მონაცემთა ბაზა. შექმნის დროს მივუთითოთ ფაილები. მოთხოვნას აქვს სახე:

```
USE Master;
-- თუ Shek Baza2 veta ბაზა არსებობს, მაშინ ის წაიშლება
IF DB_ID (N'Baza2') IS NOT NULL
DROP DATABASE Baza2;
-- Baza2 ბაზის შექმნა
```

```
CREATE DATABASE Baza2
ON
(
NAME = Baza2,
FILENAME =
'C:\Program Files\Microsoft SQL Server\MSSQL10.MSSQLSERVER\MSSQL\DATA\Baza2.mdf',
SIZE = 15MB,
MAXSIZE = 36MB,
FILEGROWTH = 3MB
)
LOG ON
( NAME = Baza2_log,
FILENAME =
'C:\Program Files\Microsoft SQL Server\MSSQL10.MSSQLSERVER\MSSQL\DATA\Baza2_Log.ldf',
SIZE = 3MB,
MAXSIZE = 25MB,
FILEGROWTH = 2MB );
```

გ. შევქმნათ Baza3 მონაცემთა ბაზა. CREATE DATABASE ბრძანების შესასრულებლად ვიყენებთ EXEC() ფუნქციას. მას პარამეტრად გადაეცემა CREATE DATABASE ბრძანების კოდი. მოთხოვნას აქვს სახე:

```
USE master;
-- თუ Baza3 ბაზა არსებობს, მაშინ ის წაიშლება
IF DB_ID (N'Baza3') IS NOT NULL
DROP DATABASE Baza3;
-- იმ კატალოგისკენ გზის განსაზღვრა, რომელშიც იქმნება Baza3 მონაცემთა ბაზა
DECLARE @data_path NVARCHAR(256);
SET @data_path =
(
SELECT SUBSTRING(physical_name, 1, CHARINDEX(N'master.mdf', LOWER(physical_name)) - 1)
FROM master.sys.master_files
WHERE database_id = 1 AND file_id = 1
);
-- Baza3 ბაზის შექმნა
EXEC ('CREATE DATABASE Baza3
ON
(
NAME = Baza3,
FILENAME = '''+ @data_path + 'Baza3.mdf'',
SIZE = 12MB,
MAXSIZE = 32MB,
FILEGROWTH = 4MB
)
LOG ON
(
NAME = Baza3_log,
FILENAME = '''+ @data_path + 'Baza3.ldf'',
```

```
SIZE = 2MB,
MAXSIZE = 17MB,
FILEGROWTH = 3MB
)'
);
```

იმისათვის, რომ ხელით არ შევიტანოთ ფაილის ფიზიკური სახელი, შეგვიძლია ეს მოთხოვნას გავაკეთებინოთ. ამ მიზნით მოთხოვნაში გამოყენებულია შემდეგი კოდი:

```
DECLARE @data_path NVARCHAR(256);
SET @data_path =
(
SELECT SUBSTRING(physical_name, 1, CHARINDEX(N'master.mdf', LOWER(physical_name)) - 1)
FROM master.sys.master_files
WHERE database_id = 1 AND file_id = 1
);
```

აქ DECLARE ოპერატორი ახდენს @data_path ცვლადის გამოცხადებას, რომელიც წარმოადგენს 256 უნიკოდ-სიმბოლოიან სტრიქონს. შემდეგ ამ ცვლადს SET ოპერატორი ანიჭებს SELECT ბრძანების მიერ გაცემულ სტრიქონს, რომელიც წარმოადგენს გზას master.mdf ფაილამდე. გზა არის კატალოგების სახელების მიმდევრობა დაწყებული ძირითადი კატალოგიდან (C:\) იმ კატალოგის ჩათვლით, რომელშიც master.mdf ფაილია მოთავსებული.

შემდეგ მოთხოვნაში გამოყენებულია EXEC() ფუნქცია, რომელიც ასრულებს CREATE DATABASE ბრძანებას. ამ ბრძანების კოდი მოთავსებულია აპოსტროფებში, რადგან EXEC() ფუნქციის პარამეტრი სტრიქონი უნდა იყოს. CREATE DATABASE ბრძანების კოდში FILENAME არგუმენტს ენიჭება გზისა და სახელის კონკატენაცია (გაერთიანება), რითაც ფორმირდება შესაქმნელი მონაცემთა ბაზის სრული სახელი.

დ. შევქმნათ Baza4 მონაცემთა ბაზა მონაცემებისa და ტრანზაქციების ჟურნალის რამდენიმე ფაილის მითითების გზით. Baza4 შედგება მონაცემების სამი ფაილისაგან და ტრანზაქციების ჟურნალის ორი ფაილისაგან. თითოეული ფაილის ზომაა 50 მბ. მოთხოვნას აქვს სახე:

```
USE master;
-- თუ Baza4 ბაზა არსებობს, მაშინ ის წაიშლება
IF DB_ID (N'Baza4') IS NOT NULL
DROP DATABASE Baza4;
-- იმ კატალოგისკენ გზის განსაზღვრა, რომელშიც იქმნება Baza4 მონაცემთა ბაზა
DECLARE @data_path NVARCHAR(256);
SET @data_path =
(
SELECT SUBSTRING(physical_name, 1, CHARINDEX(N'master.mdf', LOWER(physical_name)) - 1)
FROM master.sys.master_files
WHERE database_id = 1 AND file_id = 1
);
-- Baza4 ბაზის შექმნა
EXEC ('CREATE DATABASE Baza4
ON
PRIMARY
(
NAME = Baza4_1,
FILENAME = '''+ @data_path + 'Baza4_dat1.mdf'',
```

```
SIZE = 50MBMB,
MAXSIZE = 100MB,
FILEGROWTH = 10MB
),
(
NAME = Baza4_2,
FILENAME = '''+ @data_path + 'Baza4_dat2.ndf'',
SIZE = 50MB,
MAXSIZE = 100,
FILEGROWTH = 10
),
(
NAME = Baza4_3,
FILENAME = '''+ @data_path + 'Baza4_dat3.ndf'',
SIZE = 50MB,
MAXSIZE = 100MB,
FILEGROWTH = 10MB
)
LOG ON
(
NAME = Baza4_log1,
FILENAME = '''+ @data_path + 'Baza4_log1.ldf'',
SIZE = 50MB,
MAXSIZE = 100MB,
FILEGROWTH = 10MB
),
(
NAME = Baza4_log2,
FILENAME = '''+ @data_path + 'Baza4_log2.ldf'',
SIZE = 50MB,
MAXSIZE = 100MB,
FILEGROWTH = 10MB
)'
);
```

ე. შევქმნათ Baza5 მონაცემთა ბაზა ფაილების ჯგუფების მითითების გზით. Baza5 შედგება პირველადი ჯგუფისაგან, რომელიც შეიცავს File1_dat და File2_dat ფაილებს; Group1 ჯგუფისგან, რომელიც შეიცავს G1F3_dat და G1F4_dat ფაილებს; Group2 ჯგუფისგან, რომელიც შეიცავს G2F5 და G2F6 ფაილებს.

```
USE master;
-- თუ Baza5 ბაზა არსებობს, მაშინ ის წაიშლება
IF DB_ID (N'Baza5') IS NOT NULL
DROP DATABASE Baza5;
-- იმ კატალოგისკენ გზის განსაზღვრა, რომელშიც იქმნება Baza5 მონაცემთა ბაზა
DECLARE @data_path NVARCHAR(256);
SET @data_path =
(
```

```
SELECT SUBSTRING(physical_name, 1, CHARINDEX(N'master.mdf', LOWER(physical_name)) - 1)
FROM master.sys.master_files
WHERE database_id = 1 AND file_id = 1
);
-- Baza5 ბაზის შექმნა
EXEC ('CREATE DATABASE Baza5
ON PRIMARY
(
NAME = File1_dat,
FILENAME = '''+ @data_path + 'File1dat.mdf'',
SIZE = 20MB,
MAXSIZE = 50MB,
FILEGROWTH = 15%
),
(
NAME = File2_dat,
FILENAME = '''+ @data_path + 'File2dat.ndf'',
SIZE = 20MB,
MAXSIZE = 50MB,
FILEGROWTH = 15%
),
FILEGROUP Group1
(
NAME = G1F3_dat,
FILENAME = '''+ @data_path + 'G1F3dat.ndf'',
SIZE = 20MB,
MAXSIZE = 50MB,
FILEGROWTH = 5MB
),
(
NAME = G1F4_dat,
FILENAME = '''+ @data_path + 'G1F4dat.ndf'',
SIZE = 20MB,
MAXSIZE = 50MB,
FILEGROWTH = 5MB
),
FILEGROUP Group2
(
NAME = G2F5_dat,
FILENAME = '''+ @data_path + 'G2F5dat.ndf'',
SIZE = 20MB,
MAXSIZE = 50MB,
FILEGROWTH = 5MB
),
(
NAME = G2F6_dat,
```

```
FILENAME = '''+ @data_path + 'G2F6dat.ndf'',
SIZE = 20MB,
MAXSIZE = 50MB,
FILEGROWTH = 5MB
)
LOG ON
(
NAME = Baza5_log,
FILENAME = '''+ @data_path + 'Baza5_log.ldf'',
SIZE = 10MB,
MAXSIZE = 30MB,
FILEGROWTH = 5MB
)'
);
```

ვ. სერვერთან Baza4 მონაცემთა ბაზის მიერთება. ამ მაგალითში ხდება „დ“ მაგალითში შექმნილი მონაცემთა ბაზის გამორთვა სერვერიდან და შემდეგ მისი მიერთება სერვერთან FOR ATTACH არგუმენტის გამოყენებით.

```
USE master;
-- Baza4 მონაცემთა ბაზის გამორთვა სერვერიდან
EXEC sp_detach_db Baza4;
-- იმ კატალოგისკენ გზის განსაზღვრა, რომელშიც იქმნება Baza4 მონაცემთა ბაზა
DECLARE @data_path NVARCHAR(256);
SET @data_path =
(
SELECT SUBSTRING(physical_name, 1, CHARINDEX(N'master.mdf', LOWER(physical_name)) - 1)
FROM master.sys.master_files
WHERE database_id = 1 AND file_id = 1
);
-- Baza4 მონაცემთა ბაზის მიერთება სერვერთან
EXEC (
'CREATE DATABASE Baza4
ON
(
FILENAME = '''+ @data_path + 'Baza4_dat1.mdf''
)
FOR ATTACH'
);
```

მონაცემთა ბაზის წაშლა

მონაცემთა ბაზის წასაშლელად გამოიყენება DROP DATABASE ბრძანება. მისი სინტაქსია:

DROP DATABASE *მონაცემთა_ბაზის_სახელი* [,...n]

მაგალითები.

ა. წავშალოთ Baza1 მონაცემთა ბაზა:

```
DROP DATABASE Baza1;
```

ბ. წავშალოთ Baza1, Baza2 და Baza3 მონაცემთა ბაზები:

```
DROP DATABASE Baza1, Baza2, Baza3;
```

მონაცემთა ბაზის სახელის შეცვლა

მონაცემთა ბაზის სახელის შესაცვლელად გამოიყენება sp_renamedb შენახული პროცედურა. მისი სინტაქსია:

```
sp_renamedb [ @dbname = ] 'ძველი_სახელი', [ @newname = ] 'ახალი_სახელი'
```

ამ შენახული პროცედურის შესრულების უფლება აქვს სერვერის sysadmin ფიქსირებული როლის წევრებს.

მაგალითი. Baza1 მონაცემთა ბაზას დავარქვათ ახალი Baza2 სახელი. მოთხოვნას აქვს სახე:

```
EXEC sp_renamedb 'Baza1', 'Baza2';
```

მონაცემთა ბაზის მიერთება და გამორთვა

მონაცემთა ბაზის სერვერთან მიერთება და სერვერიდან გამორთვა შეგვიძლია გამოვიყენოთ მონაცემთა ბაზის გადასატანად ერთი სერვერიდან მეორეზე. ერთი სერვერიდან მეორეზე მონაცემთა ბაზის გადატანის მეორე გზაა - მისი სარეზერვო ასლის შექმნა და აღდგენა, მაგრამ, მონაცემთა ბაზის მიერთებისა და გამორთვის პროცედურა გაცილებით ნაკლებ დროს იკავებს, ვიდრე სარეზერვო ასლის შექმნა და აღდგენა.

სერვერიდან გამორთვის შემდეგ ველარ შევძლებთ მონაცემთა ბაზასთან მიმართვას და მასთან მუშაობას. მონაცემთა ბაზის გამოსართავად გამოიყენება sp_detach_db შენახული პროცედურა. მისი სინტაქსია:

```
sp_detach_db   [ @dbname = ] 'მონაცემთა_ბაზის_სახელი'
```

მაგალითი. Baza1 მონაცემთა ბაზის გამოსართავად უნდა შევასრულოთ ბრძანება:

```
EXEC sp_detach_db 'Baza1';
```

მონაცემთა ბაზის მიერთება არის პროცესი, როცა master მონაცემთა ბაზის sysdatabases წარმოდგენაში იქმნება ახალი სტრიქონი, რომელიც შეიცავს მომხმარებლის მონაცემთა ბაზის აღწერას თვით მონაცემთა ბაზის შექმნის გარეშე. მონაცემთა ბაზის პირველად ფაილში ინახება დანარჩენი ფაილების აღწერა ტრანზაქციების ჟურნალის ფაილების ჩათვლით.

სერვერთან მონაცემთა ბაზის მისაერთებლად გამოიყენება sp_attach_db სისტემური შენახული პროცედურა. მისი სინტაქსია:

```
sp_attach_db [ @dbname = ] 'მონაცემთა_ბაზის_სახელი'
        [ @filename1 = ] 'ფაილის_სახელი' [,...16]
```

განვიხილოთ არგუმენტების დანიშნულება.

– *'მონაცემთა_ბაზის_სახელი'* არგუმენტი მიუთითებს სახელს, რომელიც მონაცემთა ბაზას მიენიჭება მისი მიერთების შემდეგ.

– *'ფაილის_სახელი'* [,...16] მიუთითებს მონაცემთა ბაზის ფაილების სახელებს, რომელთა რაოდენობა 16-ს არ უნდა აღემატებოდეს. ეს არგუმენტი უნდა შეიცავდეს გზას მისაერთებელი მონაცემთა ბაზის პირველად ფაილამდე.

თუ მისაერთებელია 16-ზე მეტი ფაილი, მაშინ უნდა გამოვიყენოთ CREATE DATABASE FOR ATTACH ბრძანება.

მაგალითები.

ა. Baza1 მონაცემთა ბაზის მიერთებისას ეთითება მონაცემებისა და ტრანზაქციების ჟურნალის ფაილები:

EXEC sp_attach_db 'Baza1',
'C:\Program Files\Microsoft SQL Server\MSSQL10.MSSQLSERVER\MSSQL\DATA\Baza1.mdf',
'C:\Program Files\Microsoft SQL Server\MSSQL10.MSSQLSERVER\MSSQL\DATA\Baza1_log.ldf';
ბ. Baza2 მონაცემთა ბაზის მიერთებისას ეთითება მხოლოდ მონაცემების ფაილი:
EXEC sp_attach_db 'Baza2',
'C:\Program Files\Microsoft SQL Server\MSSQL10.MSSQLSERVER\MSSQL\DATA\Baza2.mdf';

ტრანზაქციების ჟურნალის მიერთება აუცილებელი არ არის, რადგან სერვერი ავტომატურად ქმნის ტრანზაქციების ახალ ჟურნალს. მიერთების დროს არ ხდება, აგრეთვე, მონაცემების მთლიანობის შემოწმება და არ მოწმდება მონაცემთა ბაზის მომხმარებლების კავშირების კორექტულობა სააღრიცხვო ჩანაწერებთან.

ინფორმაციის მიღება მონაცემთა ბაზების შესახებ

მონაცემთა ბაზების შესახებ ინფორმაციის მისაღებად შეგვიძლია გამოვიყენოთ sp_helpdb შენახული პროცედურა. მისი სინტაქსია:
sp_helpdb [[@dbname =] '*მონაცემთა_ბაზის_სახელი*']

ის გასცემს მონაცემთა ბაზის მიმდინარე ზომას, მონაცემთა ბაზის მფლობელის სააღრიცხვო ჩანაწერის სახელს, მონაცემთა ბაზის შექმნის თარიღს და მის მიმდინარე სტატუსს. ის გასცემს, აგრეთვე ინფორმაციას მონაცემთა ბაზის ფაილების შესახებ: მონაცემთა ბაზის ფაილის ფიზიკურ და ლოგიკურ სახელს, მის საიდენტიფიკაციო ნომერს, მიმდინარე და მაქსიმალურ ზომას, ნაზრდის ზომასა და ფაილების ამა თუ იმ ჯგუფთან კუთვნილობას.

მაგალითი. Shekveta მონაცემთა ბაზის შესახებ ინფორმაციის მისაღებად უნდა შევიტანოთ ბრძანება:
EXEC sp_helpdb 'Shekveta';

შედეგი:

	name	db_size	owner	dbid	created	status
1	Shekveta	6.25 MB	sa	7	Jul 17 2008	Status=ONLINE, Updateability=READ_WRITE, UserAcc...

	name	fileid	filename	filegroup	size	maxsize
1	Shekveta_Data	1	C:\Program Files\Microsoft SQL Server\MSSQL.1\MSS...	PRIMARY	2176 KB	Unlimited
2	Shekveta_Log	2	C:\Program Files\Microsoft SQL Server\MSSQL.1\MSS...	NULL	4224 KB	Unlimited

მომხმარებლის საკუთარი ტიპის შექმნა

მონაცემთა მომხმარებლის ტიპის შესაქმნელად გამოიყენება სისტემური შენახული პროცედურა sp_addtype. მისი სინტაქსია:
sp_addtype [@typename =] *ტიპის_სახელი*,
[@phystype =] *სისტემური_ტიპის_სახელი*
[, [@nulltype =] '*null_ტიპი*']

განვიხილოთ არგუმენტების დანიშნულება.

– *ტიპის_სახელი* შესაქმნელი ტიპის სახელია. ის უნდა იყოს უნიკალური მფლობელის ფარგლებში. სხვადასხვა მომხმარებლებს შეიძლება ჰქონდეთ ერთნაირი სახელის მქონე მომხმარებლის ტიპები.

– *სისტემური_ტიპის_სახელი* მონაცემების სისტემური ტიპია, რომლის საფუძველზეც იქმნება მომხმარებლის ტიპი. ეს სისტემური ტიპები მოყვანილია ცხრილში 2.2. აპოსტროფები საჭიროა მაშინ, როცა თვით ტიპის გარდა ეთითება არგუმენტებიც. n არგუმენტი არის მონაცემთა სისტემური ტიპის სიგრძე მომხმარებლის ტიპში, p არგუმენტი არის ათობითი ციფრების მაქსიმალური რაოდენობა (წერტილის წინ და შემდეგ) მონაცემების რიცხვითი ტიპებისათვის მომხმარებლის ტიპში, s არგუმენტი არის ათობითი ციფრების მაქსიმალური რაოდენობა მძიმის შემდეგ მომხმარებლის ტიპში. არ შეიძლება მომხმარებლის ტიპის შექმნა timestamp სისტემური ტიპის ბაზაზე.

– '*null_ტიპი*' არგუმენტის მნიშვნელობა განსაზღვრავს მომხმარებლის ტიპი შეინახავს თუ არა NULL მნიშვნელობას. ამ არგუმენტს შეუძლია მიიღოს ერთ-ერთი შემდეგი სამი მნიშვნელობიდან: NULL (ნებადართულია NULL მნიშვნელობის შენახვა), NOT NULL (აკრძალულია NULL მნიშვნელობის შენახვა) ან NONULL (გამოიყენება ავტომატურად განსაზღვრული მნიშვნელობა). თუ ეს არგუმენტი მითითებული არ არის, მაშინ მომხმარებლის ტიპის შექმნისას გამოიყენება ავტომატური მნიშვნელობა. თუ სვეტის შექმნის დროს აშკარადაა მითითებული NULL ან NOT NULL მნიშვნელობები, მაშინ ამ არგუმენტის მნიშვნელობა იგნორირდება.

მაგალითები.

ა. შევქმნათ მომხმარებლის ტიპი მობილური ტელეფონის ნომრების აღწერისათვის:

```
EXEC sp_addtype mobiluri, 'VARCHAR(10)', 'NOT NULL';
```

ბ. შევქმნათ მონაცემების მომხმარებლის ტიპი დაბადების დღის აღწერისათვის:

```
EXEC sp_addtype dabadebis_dge, DATETIME, 'NULL';
```

თუ გვინდა, რომ მომხმარებლის ტიპი მისაწვდომი გახდეს ყველა მონაცემთა ბაზაში, მაშინ ის უნდა დავუმატოთ model მონაცემთა ბაზას.

ცხრილი 2.2. მონაცემების სისტემური ტიპები

'BINARY(*n*)'	IMAGE	SMALLDATETIME
BIT	INT	SMALLINT
'CHAR(*n*)'	'NCHAR(*n*)'	TEXT
DATETIME	NTEXT	TINYINT
DECIMAL	NUMERIC	UNIQUEIDENTIFIER
'DECIMAL[(*p* [, *s*])]'	'NUMERIC[(*p* [, *s*])]'	'VARBINARY(*n*)'
FLOAT	'NVARCHAR(*n*)'	'VARCHAR(*n*)'
'FLOAT(*n*)'	REAL	

მომხმარებლის ტიპს შეგვიძლია სახელი შევუცვალოთ. ამისათვის, შეგვიძლია sp_rename შენახული პროცედურის გამოყენება. ის შეიძლება გამოვიყენოთ სხვადასხვა ობიექტისათვის სახელის შესაცვლელად. მისი სინტაქსია:

```
sp_rename [ @objname =] 'მომხმარებლის_ტიპის_ძველი_სახელი' ,
        [ @newname = ] 'მომხმარებლის_ტიპის_ახალი_სახელი' ,
        [ , [ @objtype = ] 'ობიექტის_ტიპი' ]
```

განვიხილოთ არგუმენტების დანიშნულება.

- '*მომხმარებლის_ტიპის_ძველი_სახელი*' არის მომხმარებლის ტიპის მიმდინარე სახელი, რომლის შეცვლაც გვინდა.
- '*მომხმარებლის_ტიპის_ახალი_სახელი*' არის მომხმარებლის ტიპის ახალი სახელი.
- '*ობიექტის_ტიპი*' არგუმენტი მიუთითებს იმ ობიექტის ტიპს, რომელსაც სახელს

ვუცვლით. მომხმარებლის ტიპისათვის სახელის შესაცვლელად ამ არგუმენტს უნდა მივანიჭოთ 'USERDATATYPE' მნიშვნელობა.

მაგალითი. 'mobiluri' მომხმარებლის ტიპს დავარქვათ ახალი 'Mobiluri_Telefoni' სახელი.

```
USE Shekveta;
EXEC sp_rename N'mobiluri', N'Mobiluri_Telefoni', N'USERDATATYPE';
```

თავი 3. ცხრილები

ცხრილების დაპროექტება

როგორც აღვნიშნეთ, ცხრილი არის მონაცემთა ბაზის ობიექტი, რომელიც მონაცემებს ინახავს. ცხრილების დაპროექტების დროს უნდა გადავწყვიტოთ შემდეგი საკითხები: რა მონაცემები უნდა მოვათავსოთ ცხრილებში? რომელი სვეტებისაგან უნდა შედგებოდეს ცხრილი? რომელი სვეტები შეიძლება შეიცავდეს NULL მნიშვნელობას? იქნება თუ არა სვეტებისათვის გამოყენებული მთლიანობაზე შეზღუდვები ან ავტომატური მნიშვნელობები? რომელი სვეტების ინდექსირება უნდა მოხდეს? რომელი სვეტები უნდა შევიდეს პირველად და გარე გასაღებში? და ა.შ.

ცხრილის პირველადი გასაღები

პირველადი გასაღების (*primary key*) დანიშნულებაა ცხრილის მთლიანობის უზრუნველყოფა. ის შეიძლება ერთი ან მეტი სვეტისაგან შედგებოდეს. თუ პირველადი გასაღები ერთი სვეტისაგან შედგება, მაშინ ამ სვეტის მნიშვნელობები უნიკალური უნდა იყოს. თუ პირველადი გასაღები რამდენიმე სვეტისაგან შედგება, მაშინ თითოეული სვეტის შიგნით მნიშვნელობები შეიძლება გამეორდეს. მაგრამ, პირველად გასაღებში შემავალი ყველა სვეტის მნიშვნელობების ნებისმიერი კომბინაცია უნიკალური უნდა იყოს (თავი 1, ნახ. 1.8. Makavshirebeli ცხრილი).

პირველადი გასაღების შექმნის დროს მასში შემავალი სვეტებისათვის სერვერი ავტომატურად ქმნის უნიკალურ ინდექსს. ეს ინდექსი აჩქარებს ამ სვეტების მონაცემებთან მიმართვას მოთხოვნებში პირველადი გასაღების გამოყენების დროს.

ცხრილს შეიძლება ჰქონდეს მხოლოდ ერთი PRIMARY KEY შეზღუდვა. ამასთან, პირველად გასაღებში შემავალი არც ერთი სვეტი არ უნდა შეიცავდეს NULL მნიშვნელობას. რადგან PRIMARY KEY შეზღუდვა იძლევა მონაცემების უნიკალურობის გარანტიას, ამიტომ ის ხშირად განისაზღვრება სვეტი-მთვლელებისათვის.

პირველადი გასაღების ერთ-ერთი დანიშნულებაა რამდენიმე ცხრილის მონაცემების მიმართვითი მთლიანობის უზრუნველყოფა. ამის რეალიზება შესაძლებელია სხვა ცხრილებში გარე გასაღებების განსაზღვრით.

ცხრილის გარე გასაღები

გარე გასაღები (*Foreign Key, FK*) შეიძლება შეიცავდეს ერთ ან მეტ სვეტს და გამოიყენება ორ ცხრილს შორის კავშირის დასამყარებლად.

დამოკიდებული ცხრილის გარე გასაღებში შემავალ სვეტებში მონაცემებმა მხოლოდ ის მნიშვნელობები უნდა მიიღონ, რომლებიც აქვთ მათთან დაკავშირებული მთავარი ცხრილის პირველადი გასაღების სვეტებს. ეს იძლევა იმის გარანტიას, რომ მთავარი ცხრილის პირველადი გასაღების მონაცემები არ იქნება შეცვლილი ან წაშლილი დამოკიდებული ცხრილის გარე გასაღების მონაცემებზე ასახვის გარეშე (*მონაცემების მიმართვითი მთლიანობა*). ამიტომ, მართალია, FOREIGN KEY შეზღუდვის მთავარი მიზანია დამოკიდებული ცხრილის გარე გასაღებში მოთავსებული მონაცემების მართვა, ის აგრეთვე საშუალებას გვაძლევს ვმართოთ მთავარი ცხრილის პირველად გასაღებში მონაცემების ცვლილება. მაგალითად, თუ Personali ცხრილიდან წავშლით რომელიმე სტრიქონს, მაშინ ფარდობითი მთლიანობა Personali და Xelshekruleba ცხრილებს შორის დაირღვევა, რადგან

წაშლილი თანამშრომლის შეკვეთები (ხელშეკრულებები) Xelshekruleba ცხრილში „ობლად” დარჩება. FOREIGN KEY შეზღუდვა თავიდან გვაცილებს ასეთ პრობლემებს - Personali ცხრილში ასეთი სტრიქონის წაშლას ვერ შევძლებთ.

თუ გარე გასაღების სვეტი NULL მნიშვნელობას შეიცავს, მაშინ შემოწმება FOREIGN KEY შეზღუდვაზე არ შესრულდება. შესაძლებელია გარე გასაღების ინდექსირება, რაც აჩქარებს საჭირო მონაცემების მოძებნას.

სვეტის უნიკალურობის განსაზღვრა

UNIQUE შეზღუდვა განკუთვნილია სვეტში მნიშვნელობების უნიკალურობის უზრუნველსაყოფად. როგორც აღვნიშნეთ, PRIMARY KEY შეზღუდვისათვის ავტომატურად ხდება მნიშვნელობების უნიკალურობის უზრუნველყოფა. თუ პირველადი გასაღები განსაზღვრულია და კიდევ ერთ ან მეტ სვეტშია საჭირო უნიკალურობის უზრუნველყოფა, მაშინ უნდა გამოვიყენოთ UNIQUE შეზღუდვა. FOREIGN KEY შეზღუდვისაგან განსხვავებით UNIQUE შეზღუდვა უშვებს სვეტში NULL მნიშვნელობის არსებობას. მაგრამ, შესაბამის სვეტში შეიძლება იყოს მხოლოდ ერთი NULL მნიშვნელობა.

შემმოწმებელი შეზღუდვების განსაზღვრა

CHECK შეზღუდვა განსაზღვრავს შესაძლო მნიშვნელობების დიაპაზონს ერთი ან მეტი სვეტისთვის. ერთი სვეტისთვის შეიძლება განისაზღვროს რამდენიმე შეზღუდვა. თუ რამდენიმე სვეტის მიმართ უნდა გამოვიყენოთ ერთი და იგივე შეზღუდვა, მაშინ ეს შეზღუდვა ცალ-ცალკე უნდა განისაზღვროს თითოეული სვეტისთვის.

CHECK შეზღუდვას საფუძვლად უდევს ლოგიკური გამოსახულება. თუ ის იღებს true მნიშვნელობას, მაშინ მთლიანობის შეზღუდვა სრულდება და მონაცემების შეცვლისა და ჩასმის ოპერაციები ნებადართული იქნება, საწინააღმდეგო შემთხვევაში ეს ოპერაციები არ შესრულდება.

თუ მითითებულია რამდენიმე CHECK შეზღუდვა, მაშინ ისინი გამოყენებული იქნება იმ მიმდევრობით, რა მიმდევრობითაც მოხდა მათი შექმნა. შეზღუდვებში დასაშვებია სხვა სვეტების სახელების მითითებაც.

ნაგულისხმევი მნიშვნელობის განსაზღვრა

თითოეული სტრიქონის სვეტი ყოველთვის შეიცავს კონკრეტულ ან NULL მნიშვნელობას. თუ სტრიქონის შევსებისას არ ვიცით რომელიმე სვეტის მნიშვნელობა ან არ გვინდა მისი შევსება გარკვეული მიზეზის გამო, მაშინ ამ სვეტს მიენიჭება NULL მნიშვნელობა, თუ ის ნებადართულია. თუ სვეტისათვის განსაზღვრულია *ნაგულისხმევი მნიშვნელობა* (*defaults, მნიშვნელობა გაჩუმებით, ავტომატური მნიშვნელობა*), მაშინ ის იქნება სვეტის მნიშვნელობა, თუ აშკარად არ მივუთითებთ ამ სვეტის მნიშვნელობას. ნაგულისხმევი მნიშვნელობა შეგვიძლია გამოვიყენოთ მაშინ, როცა სვეტს ხშირად ენიჭება ერთი და იგივე მნიშვნელობა. ნაგულისხმევი მნიშვნელობის განსაზღვრისას შეგვიძლია ჩადგმული ფუნქციების გამოყენებაც. მაგალითად, ნაგულისხმევი მნიშვნელობა შეგვიძლია გამოვიყენოთ იმ შემთხვევაში, როცა გვინდა მიმდინარე თარიღის შეტანა. ეს შესაძლებელია GETDATE() ფუნქციის გამოყენებით.

სვეტი-მთვლელის განსაზღვრა

ცხრილებთან მუშაობისას, ხშირად, მოხერხებულია სვეტი-მთვლელის განსაზღვრა, რომლისთვისაც შესრულდება უნიკალური მნიშვნელობების ავტომატურად გენერირება. ეს შესაძლებელია რამდენიმე გზით. ერთ-ერთია IDENTITY საკვანძო სიტყვის გამოყენება. ის უზრუნველყოფს მონაცემების უნიკალურობას ერთი სვეტის ფარგლებში. სვეტი-მთვლელის განსაზღვრისას უნდა მივუთითოთ მისი საწყისი მნიშვნელობა და ბიჯი. თუ ეს პარამეტრები მითითებული არ არის, მაშინ ორივეს მნიშვნელობა 1-ის ტოლი იქნება. შედეგად, ცხრილში სტრიქონის ჩამატებისას შესაბამისი სვეტის მნიშვნელობა 1-ით გაიზრდება.

ცხრილს შეიძლება ჰქონდეს მხოლოდ ერთი სვეტი-მთვლელი.

სვეტისთვის IDENTITY თვისების განსაზღვრისას შემდეგი მოთხოვნები უნდა დავიცვათ:

– სვეტის ტიპი უნდა იყოს: BIGINT, DECIMAL, INT, NUMERIC, SMALLINT ან TINYINT;
– სვეტისთვის აკრძალული უნდა იყოს NULL მნიშვნელობის შენახვა;
– სვეტისთვის არ უნდა იყოს განსაზღვრული ავტომატური მნიშვნელობა.

სვეტი-მთვლელი ხშირად გამოიყენება სტრიქონების საიდენტიფიკაციო ნომრების შესანახად. სვეტი-მთვლელი შეიძლება, აგრეთვე შევიდეს ცხრილის პირველადი გასაღების შემადგენლობაში.

მიმდინარე IDENTITY მნიშვნელობა (ანუ ცხრილში უკანასკნელად ჩასმული მნიშვნელობა) ინახება ამავე მონაცემთა ბაზის sys.identity_columns სისტემურ წარმოდგენაში. თუ ცხრილიდან ყველა სტრიქონს წავშლით, ეს არ გამოიწვევს სვეტი-მთვლელში ნუმერაციის თავიდან დაწყებას. უფრო მეტიც, თუ სტრიქონის ჩასმა არ მოხერხდა, სერვერი მაინც გაზრდის მთვლელის მნიშვნელობას. შედეგად, სვეტი-მთვლელში ადგილი ექნება „ხვრელებს". მაგალითად, თუ ცხრილის უკანასკნელ სტრიქონს წავშლით, რომელშიც სვეტი-მთვლელის მნიშვნელობა იყო 10 და შემდეგ ერთ სტრიქონს დავუმატებთ, მაშინ დამატებულ სტრიქონში სვეტი-მთვლელის მნიშვნელობა იქნება არა 10, არამედ 11.

ცხრილის შექმნა

ცხრილის შესაქმნელად გამოიყენება CREATE TABLE ბრძანება. მისი სინტაქსია:

CREATE TABLE [*მონაცემთა_ბაზის_სახელი.*[*სქემის_სახელი.*]] *ცხრილის_სახელი*
(
{ <*სვეტის_განსაზღვრა*>
| *გამოთვლადი_სვეტის_სახელი* AS *გამოსახულება*
| <*ცხრილის_შეზღუდვა*> } [,...n]
)

განვიხილოთ არგუმენტების დანიშნულება.

– *მონაცემთა_ბაზის_სახელი* არის იმ ბაზის სახელი, რომელშიც ცხრილი იქმნება. თუ ის არ არის მითითებული, მაშინ ცხრილი მიმდინარე მონაცემთა ბაზაში შეიქმნება. მომხმარებელს, რომელსაც ცხრილის შექმნა სურს, უნდა ჰქონდეს შესაბამისი უფლებები მონაცემთა ბაზაში.
– *სქემის_სახელი* არის იმ სქემის სახელი, რომელსაც ცხრილი ეკუთვნის.
– *ცხრილის_სახელი* არის შესაქმნელი ცხრილის სახელი. ცხრილის სახელისა და სქემის სახელის კომბინაცია უნიკალური უნდა იყოს მონაცემთა ბაზის ფარგლებში. თუ ცხრილი არ იქმნება მიმდინარე მონაცემთა ბაზაში, მაშინ მონაცემთა ბაზის სახელი აშკარად უნდა მივუთითოთ. თუ ცხრილის სახელის წინ მივუთითებთ # ან ## სიმბოლოებს, მაშინ შესაბამისად შეიქმნება ლოკალური ან გლობალური დროებითი ცხრილი.

– *გამოსახულება*. მისი გამოთვლის შედეგად მიიღება მნიშვნელობა გამოთვლადი სვეტისათვის. გამოთვლადი სვეტები ვირტუალური სვეტებია, ე.ი. ეს სვეტები ფიზიკურად ცხრილში არ ინახება. მათი მნიშვნელობა გამოითვლება ცხრილის გახსნის დროს ამავე ცხრილის სხვა სვეტების მნიშვნელობების გამოყენებით. გამოსახულებაში შეიძლება მონაწილეობდეს სვეტების სახელები, მუდმივები, ფუნქციები. ქვემოთხოვნების გამოყენება დაუშვებელია. გამოთვლადი სვეტები შეიძლება ჩართული იყოს SELECT მოთხოვნაში სვეტების სიის მითითების დროს.

გარდა ამისა, გამოთვლადი სვეტები შეიძლება გამოვიყენოთ ინდექსებში ან როგორც პირველადი გასაღების ნაწილი. ამ შემთხვევაში, გამოთვლადი სვეტის მნიშვნელობები უნდა განისაზღვრებოდეს დეტერმინირებული გამოსახულებით. მაგალითად, დავუშვათ ცხრილში გვაქვს ორი მთელრიცხვა სვეტი A და B, ხოლო გამოთვლადი სვეტის მნიშვნელობა გამოითვლება A + B გამოსახულებით. ასეთ შემთხვევაში, ამ სვეტის ინდექსირება შეიძლება. თუ გამოთვლადი სვეტი განისაზღვრება A + DATEPART(dd, GETDATE()) გამოსახულებით (A სვეტის მნიშვნელობას დაემატება მიმდინარე დღის მნიშვნელობა), მაშინ მისი ინდექსირება არ შეიძლება, რადგან მომდევნო მიმართვების დროს მისი მნიშვნელობები შეიძლება შეიცვალოს.

გამოთვლადი სვეტისათვის შეიძლება UNIQUE მნიშვნელობის განსაზღვრა. გამოთვლადი სვეტები არ უნდა შევიდეს გარე გასაღების შემადგენლობაში. მათთვის არ შეიძლება ავტომატური მნიშვნელობების განსაზღვრა. გამოთვლადი სვეტების გამოყენება არ შეიძლება INSERT და UPDATE ოპერაციებში.

<სვეტის_განსაზღვრა> კონსტრუქციის სინტაქსია:

<*სვეტის_განსაზღვრა*> ::=
სვეტის_სახელი ტიპი
[[DEFAULT *გამოსახულება*]
| [IDENTITY [(*საწყისი_მნიშვნელობა*, *ნაზრდი*)]]]
[<*სვეტის_შეზღუდვა*>] [,...n]

განვიხილოთ ამ კონსტრუქციის პარამეტრების დანიშნულება.

– *ტიპი* განსაზღვრავს სვეტში მოთავსებული მონაცემების ტიპს.

– DEFAULT განსაზღვრავს ნაგულისხმევ (ავტომატურ) მნიშვნელობას მოცემული სვეტისათვის. ეს მნიშვნელობა მიენიჭება სვეტს თუ სტრიქონის ჩასმისას აშკარად არ იქნება მითითებული მნიშვნელობა. ნაგულისხმევი მნიშვნელობის განსაზღვრა არ შეიძლება timestamp ტიპის სვეტებისათვის, აგრეთვე, IDENTITY თვისების მქონე სვეტებისათვის. ნაგულისხმევი მნიშვნელობა შეიძლება იყოს მუდმივა, სისტემური ფუნქციების მნიშვნელობები. თუ სვეტისათვის არ არის განსაზღვრული DEFAULT მნიშვნელობა და სვეტისათვის არ არის მითითებული მნიშვნელობა სტრიქონის დამატებისას, მაშინ სვეტს NULL მნიშვნელობა მიენიჭება, თუ ეს ნებადართულია.

– IDENTITY მიუთითებს, რომ შესაბამისი სვეტი იქნება სვეტი-მთვლელი. *საწყისი_მნიშვნელობა* არის მთვლელის საწყისი მნიშვნელობა, *ნაზრდი* კი - მთვლელის ნაზრდი.

<*სვეტის_შეზღუდვა*> კონსტრუქციაში ეთითება მთლიანობის შეზღუდვები, რომლებიც განსაზღვრული იქნება სვეტისათვის. მისი სინტაქსია:

<*სვეტის_შეზღუდვა*> ::=
[CONSTRAINT *შეზღუდვის_სახელი*]
{
[NULL | NOT NULL] | [{ PRIMARY KEY | UNIQUE }
| [[FOREIGN KEY] REFERENCES ref_*ცხრილი* [(ref_*სვეტი*)]
[ON DELETE { CASCADE | NO ACTION }]

[ON UPDATE { CASCADE | NO ACTION }]
| CHECK (*ლოგიკური_გამოსახულება*)
}

განვიხილოთ ამ კონსტრუქციის პარამეტრების დანიშნულება.

– CONSTRAINT პარამეტრის შემდეგ ეთითება სვეტის მნიშვნელობებზე შეზღუდვის სახელი, რომელიც მონაცემთა ბაზის ფარგლებში უნიკალური უნდა იყოს.

– NULL | NOT NULL. NULL პარამეტრი გვაძლევს სვეტში NULL მნიშვნელობის შენახვის უფლებას, NOT NULL პარამეტრი კი - კრძალავს სვეტში NULL მნიშვნელობის შენახვას.

– PRIMARY KEY პარამეტრი მიუთითებს, რომ სვეტის ბაზაზე უნდა შეიქმნას პირველადი გასაღები. თითოეული ცხრილისათვის შეიძლება შეიქმნას მხოლოდ ერთი ასეთი შეზღუდვა. პირველადი გასაღები ასეთი გზით განისაზღვრება სვეტის დონეზე. შედეგად, პირველადი გასაღები შემდგარი იქნება მხოლოდ ერთი სვეტისაგან. თუ საჭიროა პირველადი გასაღების ფორმირება ორი და მეტი სვეტის ბაზაზე, მაშინ საჭიროა მისი კონფიგურირება ცხრილის დონეზე. ამ საშუალებას შემდეგ განვიხილავთ.

– UNIQUE პარამეტრი მიუთითებს, რომ სვეტში მოთავსებული მონაცემები უნიკალური იქნება, ანუ სვეტში არ მოთავსდება გამეორებადი მნიშვნელობები. UNIQUE მთლიანობის შეზღუდვისათვის ავტომატურად იქმნება ინდექსი. ერთ ცხრილში სხვადასხვა სვეტისთვის შეგვიძლია შევქმნათ რამდენიმე UNIQUE შეზღუდვა.

– [FOREIGN KEY] REFERENCES *მთავარი_ცხრილი* [(*სვეტის_სახელი* [,...n])] პარამეტრი მიუთითებს, რომ სვეტი იქნება გარე გასაღები *მთავარი_ცხრილი* პარამეტრით განსაზღვრული ცხრილისათვის. გარე გასაღებში შემავალი სვეტები შეიძლება მიმართავდეს მხოლოდ *მთავარი_ცხრილი* პარამეტრით განსაზღვრული ცხრილის PRIMARY KEY ან UNIQUE შეზღუდვის მქონე სვეტებს. *სვეტის_სახელი* არის პირველად გასაღებში შემავალი სვეტი ან სვეტების სია.

– ON DELETE { CASCADE | NO ACTION }. თუ მივუთითებთ CASCADE საკვანძო სიტყვას, მაშინ მთავარი ცხრილიდან სტრიქონის წაშლის შემთხვევაში დამოკიდებულ ცხრილშიც წაიშლება შესაბამისი სტრიქონი (ან სტრიქონები). თუ მივუთითებთ NO ACTION საკვანძო სიტყვას, მაშინ სერვერი გასცემს შეტყობინებას შეცდომის შესახებ და სტრიქონები არ წაიშლება მთავარი ცხრილიდან. NO ACTION პარამეტრი ავტომატურად იგულისხმება.

– ON UPDATE { CASCADE | NO ACTION }. თუ მივუთითებთ CASCADE საკვანძო სიტყვას, მაშინ მთავარი ცხრილის პირველადი გასაღების მნიშვნელობის შეცვლისას დამოკიდებული ცხრილის შესაბამის სტრიქონებში შეიცვლება გარე გასაღების მნიშვნელობაც. თუ მივუთითებთ NO ACTION საკვანძო სიტყვას, მაშინ სერვერი გასცემს შეტყობინებას შეცდომის შესახებ და არ შეასრულებს ცვლილებებს მთავარ ცხრილში. NO ACTION პარამეტრი ავტომატურად იგულისხმება.

– CHECK (*ლოგიკური_გამოსახულება*) პარამეტრი ახორციელებს მთლიანობის შეზღუდვას იმ შესაძლო მნიშვნელობების შემოწმების გზით, რომლებიც შეგვაქვს სვეტში ან სვეტებში. თუ ლოგიკური გამოსახულება იღებს true მნიშვნელობას, მაშინ მონაცემების შეცვლის ან ჩასმის ოპერაციები ნებადართული იქნება, საწინააღმდეგო შემთხვევაში კი - არა.

<ცხრილის_შეზღუდვა> კონსტრუქცია განსაზღვრავს მთლიანობის შეზღუდვებს ცხრილის დონეზე. მისი სინტაქსია:

<ცხრილის_შეზღუდვა> ::=
[CONSTRAINT *შეზღუდვის_სახელი*]
{
[{ PRIMARY KEY | UNIQUE }
{ (*სვეტის_სახელი* [ASC | DESC] [,...n]) }]

| FOREIGN KEY
[(*სვეტის_სახელი* [,...n])]
REFERENCES *მთავარი_ცხრილი* [(*სვეტის_სახელი* [,...n])]
[ON DELETE { CASCADE | NO ACTION }]
[ON UPDATE { CASCADE | NO ACTION }]
| CHECK (*ლოგიკური_გამოსახულება*)
}

– *სვეტის_სახელი* პარამეტრი არის იმ სვეტის სახელი ან სვეტების სახელების სია, რომელსაც უნდა დაედოს შეზღუდვა.

– [ASC | DESC] პარამეტრი განსაზღვრავს ინდექსში მონაცემების დახარისხების მეთოდს. ინდექსი იქმნება PRIMARY KEY, UNIQUE, CLUSTERED და NONCLUSTERED საკვანძო სიტყვების მითითებისას. თუ მითითებულია ASC, მაშინ ინდექსში მონაცემები დახარისხდება ზრდადობის მიხედვით, თუ მითითებულია DESC, მაშინ - კლებადობის მიხედვით.

მაგალითები.

ა. შევქმნათ ცხრილი, რომლის cxriliID სვეტი იქნება პირველადი გასაღები და ის იქნება სვეტი-მთვლელი. მთვლელის საწყისი მნიშვნელობა და ნაზრდი 1-ის ტოლია. მოთხოვნას აქვს სახე:

```
USE Baza_1;
--      თუ ცხრილი არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID(N'Cxrili', N'U') IS NOT NULL
   DROP TABLE Cxrili;
--      ცხრილის შექმნა
CREATE TABLE Cxrili
(
cxriliID INT PRIMARY KEY IDENTITY (1,1),
sveti1   INT,
sveti2   NVARCHAR(20),
sveti3   FLOAT,
sveti4   DATETIME
);
```

ბ. შევქმნათ ცხრილი, რომლის cxriliID სვეტი იქნება პირველადი გასაღები და ის იქნება სვეტი-მთვლელი. მთვლელის საწყისი მნიშვნელობა და ნაზრდი 1-ის ტოლია. პირველადი გასაღების მნიშვნელობები დახარისხებული უნდა იყოს კლებადობით. ამ მიზნით გამოიყენება სვეტის შეზღუდვა (CONSTRAINT), რომლის სახელია FK_C1. მოთხოვნას აქვს სახე:

```
USE Baza_1;
--      თუ ცხრილი არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID(N'Cxrili', N'U') IS NOT NULL
   DROP TABLE Cxrili;
--      ცხრილის შექმნა
CREATE TABLE Cxrili
(
cxriliID INT     CONSTRAINT FK_C1 PRIMARY KEY (cxriliID DESC) IDENTITY (1,1),
sveti1     INT   NULL,
sveti2     INT,
sveti3     NVARCHAR(20),
sveti4     FLOAT,
sveti5    DATETIME
```

);
გ. შევქმნათ Cxrili_2 დამოკიდებული ცხრილი, რომლის cxriliID სვეტი იქნება გარე გასაღები და დაუკავშირდება მთავარი Cxrili ცხრილის cxriliID პირველად გასაღებს. მოთხოვნას აქვს სახე:
```
USE Baza_1;
--      თუ ცხრილი არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID(N'Cxrili_2', N'U') IS NOT NULL
   DROP TABLE Cxrili_2;
--      ცხრილის შექმნა
CREATE TABLE Cxrili_2
(
cxrili_2ID INT PRIMARY KEY IDENTITY (1,1),
cxriliID INT NULL REFERENCES Cxrili(cxriliID),
sveti2   NVARCHAR(20),
sveti3   FLOAT,
sveti4   DATETIME
);
```
გარე გასაღები შეგვიძლია განვსაზღვროთ, აგრეთვე, FOREIGN KEY საკვანძო სიტყვის მითითების გზით:
```
USE Baza_1;
--      თუ ცხრილი არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID(N'Cxrili_2', N'U') IS NOT NULL
   DROP TABLE Cxrili_2;
--      ცხრილის შექმნა
CREATE TABLE Cxrili_2
(
cxrili_2ID INT PRIMARY KEY IDENTITY (1,1),
cxriliID INT NULL FOREIGN KEY (cxriliID) REFERENCES Cxrili(cxriliID),
sveti2   NVARCHAR(20),
sveti3   FLOAT,
sveti4   DATETIME
);
```
დ. შევქმნათ Cxrili_3 ცხრილი, რომლის cxrili_3ID სვეტი იქნება გარე გასაღები და დაუკავშირდება მთავარი Cxrili_1 ცხრილის cxrili_1ID პირველად გასაღებს. განვსაზღვროთ რეჟიმი, როცა მთავარი ცხრილიდან სტრიქონის წაშლისას დამოკიდებულ ცხრილში შესაბამისი სტრიქონები წაიშლება. მოთხოვნას აქვს სახე:
```
USE Baza_1;
--      თუ ცხრილი არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID(N'Cxrili_3', N'U') IS NOT NULL
   DROP TABLE Cxrili_3;
--      ცხრილის შექმნა
CREATE TABLE Cxrili_3
(
cxrili_3ID INT PRIMARY KEY IDENTITY (1,1),
cxrili_1ID INT NULL REFERENCES Cxrili(cxriliID) ON DELETE CASCADE,
sveti2   NVARCHAR(20),
sveti3   FLOAT,
```

```
sveti4   DATETIME
);
```
ე. შევქმნათ Cxrili_3 ცხრილი, რომლის cxrili_3ID სვეტი იქნება გარე გასაღები და დაუკავშირდება მთავარი Cxrili_1 ცხრილის cxrili_1ID პირველად გასაღებს. განვსაზღვროთ რეჟიმი, როცა მთავარი ცხრილის პირველადი გასაღების მნიშვნელობის შეცვლისას დამოკიდებულ ცხრილში გარე გასაღების მნიშვნელობები შესაბამისად შეიცვლება. მოთხოვნას აქვს სახე:

```
USE Baza_1;
--        თუ ცხრილი არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID(N'Cxrili_3', N'U') IS NOT NULL
   DROP TABLE Cxrili_3;
--        ცხრილის შექმნა
CREATE TABLE Cxrili_3
(
cxrili_3ID INT PRIMARY KEY IDENTITY (1,1),
cxrili_1ID INT NULL REFERENCES Cxrili(cxriliID) ON UPDATE CASCADE,
sveti2   NVARCHAR(20),
sveti3   FLOAT,
sveti4   DATETIME
);
```
ვ. შევქმნათ Cxrili_3 ცხრილი, რომლის cxrili_3ID სვეტი იქნება გარე გასაღები და დაუკავშირდება მთავარი Cxrili_1 ცხრილის cxrili_1ID პირველად გასაღებს. განვსაზღვროთ რეჟიმი, როცა მთავარ ცხრილში პირველადი გასაღების მნიშვნელობის შეცვლისას დამოკიდებულ ცხრილში გარე გასაღების მნიშვნელობები შესაბამისად შეიცვლება, ხოლო მთავარი ცხრილიდან სტრიქონის წაშლისას კი - დამოკიდებულ ცხრილში შესაბამისი სტრიქონები წაიშლება. მოთხოვნას აქვს სახე:

```
USE Baza_1;
--        თუ ცხრილი არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID(N'Cxrili_3', N'U') IS NOT NULL
   DROP TABLE Cxrili_3;
--        ცხრილის შექმნა
CREATE TABLE Cxrili_3
(
cxrili_3ID INT PRIMARY KEY IDENTITY (1,1),
cxrili_1ID INT NULL REFERENCES Cxrili(cxriliID) ON UPDATE CASCADE
                                                ON DELETE CASCADE,
sveti2   NVARCHAR(20),
sveti3   FLOAT,
sveti4   DATETIME
);
```
ზ. შევქმნათ მთავარი Cxrili_1 ცხრილი, რომლის პირველადი გასაღები ორი სვეტისაგან შედგება: cxrili_1ID და sveti1. ორსვეტიანი პირველადი გასაღების შესაქმნელად გამოიყენება ცხრილის შეზღუდვა, რომლის სახელია PK_C1. შევქმნათ დამოკიდებული Cxrili_4 ცხრილი, რომლის გარე გასაღები ორი სვეტისაგან შედგება: cxrili_1ID და sveti1. ისინი მიმართავენ მთავარი Cxrili_1 ცხრილის პირველად გასაღებს, რომელიც ასევე, ორი სვეტისაგან შედგება: cxrili_1ID და sveti1. Cxrili_4 ცხრილისთვის შევქმნათ ცხრილის შეზღუდვა, რომლის სახელია

FK_C2. ის განსაზღვრავს რეჟიმს, როცა მთავარი ცხრილიდან სტრიქონის წაშლისას დამოკიდებულ ცხრილში შესაბამისი სტრიქონები წაიშლება. მოთხოვნას აქვს სახე:

```
USE Baza_1;
--      მთავარი ცხრილის შექმნა
--      თუ ცხრილი არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID(N'Cxrili_1', N'U') IS NOT NULL
   DROP TABLE Cxrili_1;
--      ცხრილის შექმნა
CREATE TABLE Cxrili_1
(
cxrili_1ID INT,
sveti1   INT,
sveti2   NVARCHAR(20),
sveti3   FLOAT,
sveti4   DATETIME,
CONSTRAINT PK_C1 PRIMARY KEY (cxrili_1ID, sveti1)
);

--      დამოკიდებული ცხრილის შექმნა
--      თუ ცხრილი არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID(N'Cxrili_4',N'U') IS NOT NULL
   DROP TABLE Cxrili_4
--      ცხრილის შექმნა
CREATE TABLE Cxrili_4
(
cxrili_4ID INT PRIMARY KEY IDENTITY (1,1),
cxrili_1ID INT NULL,
sveti1   INT,
sveti2   NVARCHAR(20),
sveti3   FLOAT,
sveti4   DATETIME,
CONSTRAINT FK_C2  FOREIGN KEY (cxrili_1ID, sveti1)
                  REFERENCES Cxrili_1(cxrili_1ID, sveti1)
                  ON DELETE CASCADE
);
```

თ. შევქმნათ ცხრილი, რომლის sveti1 და sveti2 სვეტებისთვის განსაზღვრული იქნება UNIQUE შეზღუდვა. მოთხოვნას აქვს სახე:

```
USE Baza_1;
--      თუ ცხრილი არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID(N'Cxrili_5',N'U') IS NOT NULL
   DROP TABLE Cxrili_5;
--      ცხრილის შექმნა
CREATE TABLE Cxrili_5
(
cxrili_5ID INT PRIMARY KEY IDENTITY (1,1),
sveti1   INT NOT NULL UNIQUE,
```

```
sveti2   NVARCHAR(20) NULL UNIQUE,
sveti3   FLOAT,
sveti4   DATETIME
);
```

ი. შევქმნათ ცხრილი, რომლის sveti1, sveti2 და sveti4 სვეტებისთვის განსაზღვრული იქნება ნაგულისხმევი მნიშვნელობები. მოთხოვნას აქვს სახე:

```
USE Baza_1;
--       თუ ცხრილი არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID(N'Cxrili_5',N'U') IS NOT NULL
   DROP TABLE Cxrili_5;
--       ცხრილის შექმნა
CREATE TABLE Cxrili_5
(
cxrili_5ID INT PRIMARY KEY IDENTITY (1,1),
sveti1   INT DEFAULT 25,
sveti2   NVARCHAR(20) DEFAULT N'რომან სამხარაძე',
sveti3   FLOAT,
sveti4   DATETIME DEFAULT GETDATE()
);
```

კ. შევქმნათ ცხრილი, რომლის sveti1, sveti2 და sveti4 სვეტებისთვის განსაზღვრული იქნება შეზღუდვები. მოთხოვნას აქვს სახე:

```
USE Baza_1;
--       თუ ცხრილი არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID(N'Cxrili_5',N'U') IS NOT NULL
   DROP TABLE Cxrili_5;
--       ცხრილის შექმნა
CREATE TABLE Cxrili_5
(
cxrili_5ID INT PRIMARY KEY IDENTITY (1,1),
sveti1   INT CHECK ( sveti1 > 0 AND sveti1 < 10 ),
sveti2   NVARCHAR(20) CHECK (sveti2 IN ('1389', '0736', '1756') OR sveti2 LIKE '99[0-9][0-9]'),
sveti3   FLOAT,
sveti4   DATETIME CHECK ( sveti4 >= '05.10.2000' AND sveti4 <= '19.03.2005' )
);
```

ლ. შევქმნათ ცხრილი, რომელსაც ექნება გამოთვლადი სვეტები. მოთხოვნას აქვს სახე:

```
USE Baza_1;
--       თუ ცხრილი არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID(N'Cxrili_5',N'U') IS NOT NULL
   DROP TABLE Cxrili_5;
--       ცხრილის შექმნა
CREATE TABLE Cxrili_5
(
cxrili_5ID INT PRIMARY KEY IDENTITY (1,1),
sveti1   INT,
sveti2   INT,
sveti3   AS sveti1 + sveti2 / 2,
```

```
sveti4   AS USER_NAME(),
sveti5   AS GETDATE()
);
```

ახლა თქვენ შეგიძლიათ Shekveta მონაცემთა ბაზაში დამოუკიდებლად შექმენათ შესავალში აღწერილი Personali, Shemkveti და Xelshekruleba ცხრილები.

ცხრილის წაშლა

ცხრილის წასაშლელად გამოიყენება DROP TABLE ბრძანება. მისი სინტაქსია:

DROP TABLE *ცხრილის_სახელი*

იმისათვის, რომ შევძლოთ ცხრილის წაშლა, უნდა ვიყოთ სერვერის sysadmin სტანდარტული როლის წევრი, ან მონაცემთა ბაზის db_owner ან db_dbadmin სტანდარტული როლის წევრი.

ჩვენ ვერ წავშლით ცხრილს, თუ არსებობენ ცხრილები, რომლებიც მიმართავს ამ ცხრილს გარე გასაღების საშუალებით; თუ ცხრილთან დაკავშირებულია ერთი ან მეტი წარმოდგენა და ა.შ. ამიტომ, სანამ ამ ცხრილს წავშლიდეთ, საჭიროა მონაცემთა ბაზის ყველა იმ ობიექტის წაშლა, რომლებიც წასაშლელ ცხრილს მიმართავს ან ამ ობიექტების ისე შეცვლა, რომ ისინი ამ ცხრილს აღარ მიმართავდეს.

მაგალითი.

```
DROP TABLE Cxrili_1;
```

ცხრილისთვის სახელის შეცვლა

ცხრილისათვის სახელის შესაცვლელად გამოიყენება sp_rename შენახული პროცედურა.

მაგალითი. Cxrili_4 ცხრილს დავარქვათ ახალი Cxrili_5 სახელი.

```
USE Basa_1;
EXEC sp_rename 'Cxrili_4', 'Cxrili_5';
```

ამავე პროცედურის გამოყენებით სახელი შეგვიძლია შევუცვალოთ ცხრილის სვეტს.

მაგალითი. Cxrili ცხრილის sveti3 სვეტს დავარქვათ ახალი sveti6 სახელი.

```
EXEC sp_rename 'Cxrili.sveti3', 'sveti6', 'COLUMN';
```

ინფორმაციის მიღება ცხრილების შესახებ

ინფორმაცია მონაცემთა ბაზაში შექმნილი ობიექტების, მათ შორის, ცხრილების შესახებ, ინახება ამავე მონაცემთა ბაზის sys.objects წარმოდგენაში. კონკრეტული ცხრილის შესახებ ინფორმაციის მისაღებად გამოიყენება OBJECTPROPERTY ფუნქცია. მაგალითად, Personali ცხრილის შესახებ ინფორმაციის მისაღებად უნდა შევასრულოთ მოთხოვნა:

```
USE Shekveta;
SELECT * FROM sys.objects
WHERE object_id =    object_id(N'[dbo].[Personali]') AND
                     OBJECTPROPERTY(object_id, N'IsUserTable') = 1;
```

შედეგი:

	name	object_id	principal_id	schema_id	parent_object_id	type	type_desc	create_date
1	Personali	2123154609	NULL	1	0	U	USER_TABLE	2014-05-27 09:43:04.610

ცხრილის თვისებების სანახავად შეგვიძლია გამოვიყენოთ სისტემური შენახული პროცედურა:

sp_help [[@objname =] *ცხრილის_სახელი*]

მაგალითი. მოთხოვნა გასცემს Personali ცხრილის თვისებებს:

```
EXEC sp_help Personali;
```

შედეგი:

	Name	Owner	Type	Created_datetime
1	Personali	dbo	user table	2014-05-27 09:43:04.610

	Column_name	Type	Computed	Length	Prec	Scale	Nullable	TrimTrailingBlanks
1	personaliID	int	no	4	10	0	no	(n/a)
2	gvari	nvarchar	no	60			yes	(n/a)
3	saxeli	nvarchar	no	30			yes	(n/a)
4	ganyofileba	nvarchar	no	60			yes	(n/a)
5	qalaqi	nvarchar	no	60			yes	(n/a)
6	regioni	nvarchar	no	60			yes	(n/a)
7	raioni	nvarchar	no	60			yes	(n/a)
8	xelfasi	float	no	8	53	NU...	yes	(n/a)

	Identity	Seed	Increment	Not For Replication
1	personaliID	1	1	0

	RowGuidCol
1	No rowguidcol column defined.

	Data_located_on_filegroup
1	PRIMARY

	index_name	index_description	index_keys
1	ind_ganyofileba	nonclustered located on PRIMARY	ganyofileba
2	PK_Personali	clustered, unique, primary key located on PRIMARY	personaliID

ამ ცხრილების სვეტების აღწერა შეგიძლიათ სერვერის დოკუმენტაციაში იხილოთ.

ინფორმაციის მიღება ცხრილზე დამოკიდებული ობიექტების შესახებ

ასეთი ინფორმაცია სასარგებლოა მაშინ, როცა ვაპირებთ ცხრილის წაშლას. ამისათვის, შეგვიძლია გამოვიყენოთ sp_depends სისტემური შენახული პროცედურა. მისი სინტაქსია:

sp_depends [@ობიექტის_სახელი =] '*ობიექტის_სახელი*'

მაგალითი. გვაინტერესებს Personali ცხრილზე დამოკიდებული ობიექტების სახელები და ტიპები. მოთხოვნას აქვს სახე:

```
USE Shekveta;
EXEC sp_depends 'Personali';
```

შედეგი:

	name	type
1	dbo.M3	inline function
2	dbo.Maqsimaluri_Xelfasi	scalar function
3	dbo.Sashualo_Xelfasi	inline function
4	dbo.View_1	view

ინფორმაციის მიღება ცხრილებს შორის კავშირების შესახებ

ინფორმაცია იმის შესახებ, თუ მოცემული ცხრილი რომელ ცხრილებს უკავშირდება PRIMARY KEY და FOREIGN KEY შეზღუდვების საშუალებით, შეგვიძლია მივიღოთ sp_fkeys შენახული პროცედურის საშუალებით. მისი სინტაქსია:

sp_fkeys [@pktable_name =] '*პგ_ცხრილის_სახელი*'
[, [@pktable_owner =] '*პგ_ცხრილის_მფლობელი*']
[, [@pktable_qualifier =] '*პგ_ბაზის_სახელი*']
[, [@fktable_name = '*გგ_ცხრილის_სახელი*']
[, [@fktable_owner = '*გგ_ცხრილის_მფლობელი*']
[, [@fktable_qualifier = '*გგ_ბაზის_სახელი*']

განვიხილოთ პარამეტრების დანიშნულება.

– '*პგ_ცხრილის_სახელი*'. მთავარი ცხრილის სახელია, რომლის შესახებაც გვინდა ინფორმაციის მიღება. ცხრილი უნდა შეიცავდეს პირველად გასაღებს;
– '*პგ_ცხრილის_მფლობელი*'. პირველადი გასაღების შემცველი მთავარი ცხრილის მფლობელის სახელია;
– '*პგ_ბაზის_სახელი*'. მონაცემთა ბაზის სახელია, რომელიც შეიცავს პირველადი გასაღების შემცველ მთავარ ცხრილს;
– '*გგ_ცხრილის_სახელი*'. დამოკიდებული ცხრილის სახელია, რომელიც გარე გასაღებს შეიცავს;
– '*გგ_ცხრილის_მფლობელი*'. გარე გასაღების შემცველი დამოკიდებული ცხრილის მფლობელის სახელია;
– '*გგ_ბაზის_სახელი*'. მონაცემთა ბაზის სახელია, რომელიც შეიცავს გარე გასაღების შემცველ დამოკიდებულ ცხრილს.

თუ შევიტანთ მხოლოდ პირველადი გასაღების შემცველი ცხრილის პარამეტრებს, მაშინ გაიცემა ინფორმაცია ამ ცხრილის პირველად გასაღებთან დაკავშირებული ყველა ცხრილის შესახებ. თუ მივითითებთ გარე გასაღების შემცველი ცხრილის პარამეტრებს, მაშინ გაიცემა ინფორმაცია იმ ცხრილების შესახებ, რომლებიც დაკავშირებულია ამ ცხრილის გარე გასაღებთან.

მაგალითები.

ა. მივიღოთ ინფორმაცია Personali ცხრილთან დაკავშირებული ცხრილების შესახებ. მოთხოვნას აქვს სახე:

```
USE Shekveta;
EXEC sp_fkeys @pktable_name = N'Personali';
```

შედეგი:

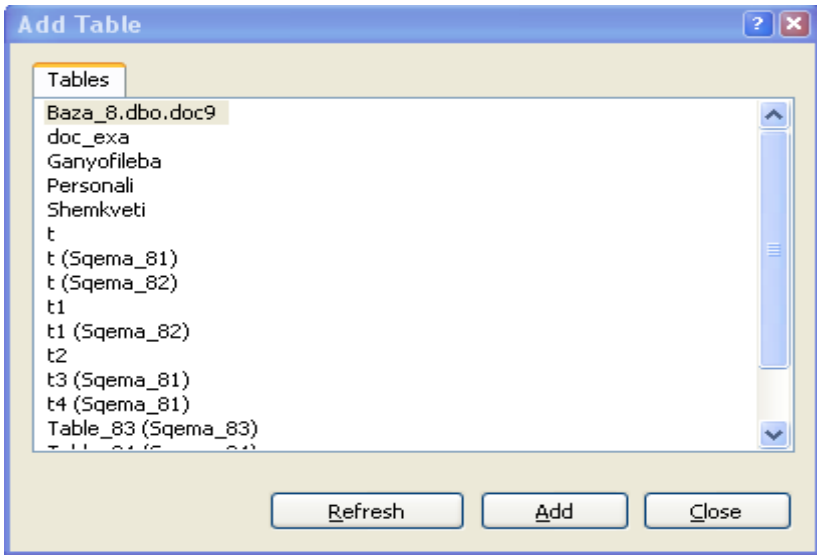
	PKTABLE_QUALIFIER	PKTABLE_OWNER	PKTABLE_NAME	PKCOLUMN_NAME	FKTA
1	Shekveta	dbo	Personali	personaliID	Shek
2	Shekveta	dbo	Personali	personaliID	Shek

ბ. მივიღოთ ინფორმაცია Xelshekruleba ცხრილთან დაკავშირებული ცხრილების შესახებ. მოთხოვნას აქვს სახე:

```
EXEC sp_fkeys @fktable_name = N'Xelshekruleba';
```

შედეგი:

	PKTABLE_QUALIFIER	PKTABLE_OWNER	PKTABLE_NAME	PKCOLUMN_NAME	FKTA
1	Shekveta	dbo	Personali	personaliID	Shek
2	Shekveta	dbo	Shemkveti	shemkvetiID	Shek

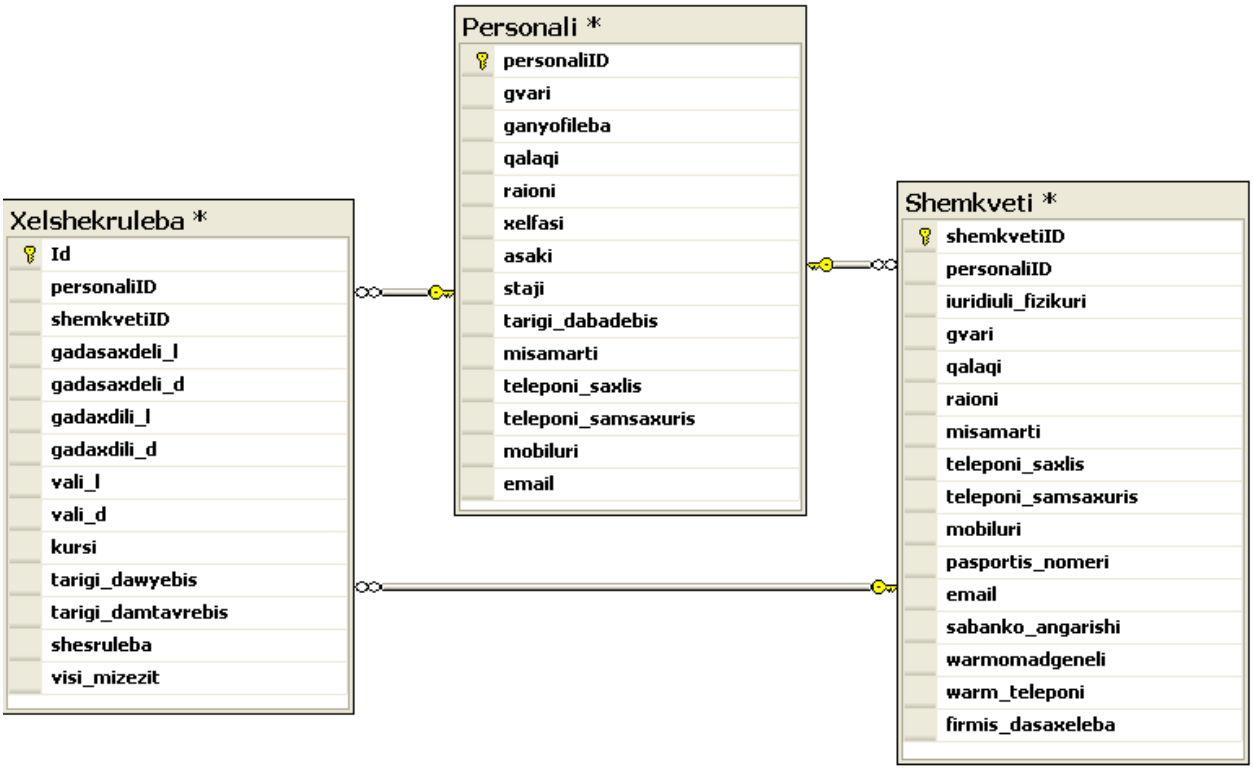


ნახ. 3.1.

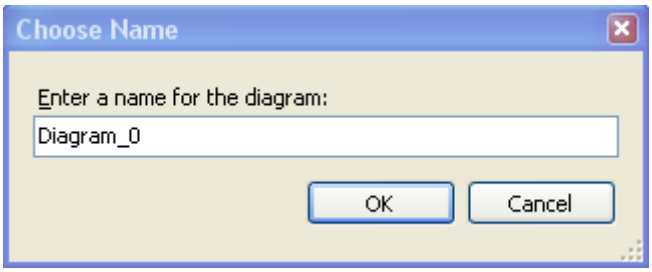
დიაგრამები

როცა მონაცემთა ბაზა ერთმანეთთან დაკავშირებულ ბევრ ცხრილს შეიცავს, მისი სტრუქტურის თვალსაჩინოდ წარმოსადგენად *დიაგრამები* უნდა გამოვიყენოთ. დიაგრამების დანიშნულებაა მონაცემთა ბაზის ადმინისტრირების გამარტივება. დიაგრამის შესაქმნელად ვხსნით ნახ. 4-ზე ნაჩვენებ ფანჯარას. Database Diagrams ელემენტზე ვხსნით კონტექსტურ მენიუს და ვასრულებთ New Database Diagram ბრძანებას. გაიხსნება ახალი ჩანართი და Add Table ფანჯარა (ნახ. 3.1). მოვნიშნოთ Personali, Shemkveti და Xelshekruleba ცხრილები და დავაჭიროთ Add კლავიშს. ფანჯრის დახურვის შემდეგ გამოჩნდება სამივე ცხრილის სტრუქტურა (ნახ. 3.2). დიაგრამის შესანახად ვაჭერთ ინსტრუმენტების პანელის Save კლავიშს. გახსნილ ფანჯარაში (ნახ. 3.3) შეგვაქვს დიაგრამის სახელი და ვაჭერთ OK კლავიშს. როგორც

დიაგრამიდან ჩანს, „ერთი-ბევრთან" კავშირი არსებობს Personali-Shemkveti და Personali-Xelshekruleba ცხრილებს შორის. ასეთივე კავშირი არსებობს Shemkveti-Xelshekruleba ცხრილებს შორის.



ნახ. 3.2.



ნახ. 3.3.

Personali, Shemkveti და Xelshekruleba ცხრილები

შესავალში აღწერილ Personali, Shemkveti და Xelshekruleba ცხრილებს შემდეგი სტრუქტურა ექნებათ:

ცხრილი 2.3. Personali ცხრილის სტრუქტურა

სვეტის სახელი	მონაცემის ტიპი
personaliID	INT
gvari	NVARCHAR(30)
ganyofileba	NVARCHAR(30)
qalaqi	NVARCHAR(30)
raioni	NVARCHAR(30)
xelfasi	FLOAT
asaki	INT
staji	INT
tarigi_dabadebis	DATETIME
misamarti	NVARCHAR(50)
teleponi_saxlis	NVARCHAR(8)
teleponi_samsaxuris	NVARCHAR(8)
mobiluri	NVARCHAR(12)
email	NVARCHAR(30)

ცხრილი 2.4. Shemkveti ცხრილის სტრუქტურა

სვეტის სახელი	მონაცემის ტიპი
shemkvetiID	INT
personaliID	INT
iuridiuli_fizikuri	NVARCHAR(10)
gvari	NVARCHAR(30)
qalaqi	NVARCHAR(30)
raioni	NVARCHAR(30)
misamarti	NVARCHAR(50)
teleponi_saxlis	NVARCHAR(8)
teleponi_samsaxuris	NVARCHAR(8)

ცხრილი 2.4. Shemkveti ცხრილის სტრუქტურა (გაგრძელება)

mobiluri	NVARCHAR(12)
pasportis_nomeri	NVARCHAR(15)
email	NVARCHAR(30)
sabanko_angarishi	NVARCHAR(15)
warmomadgeneli	NVARCHAR(30)
warm_teleponi	NVARCHAR(12)
firmis_dasaxeleba	NVARCHAR(30)

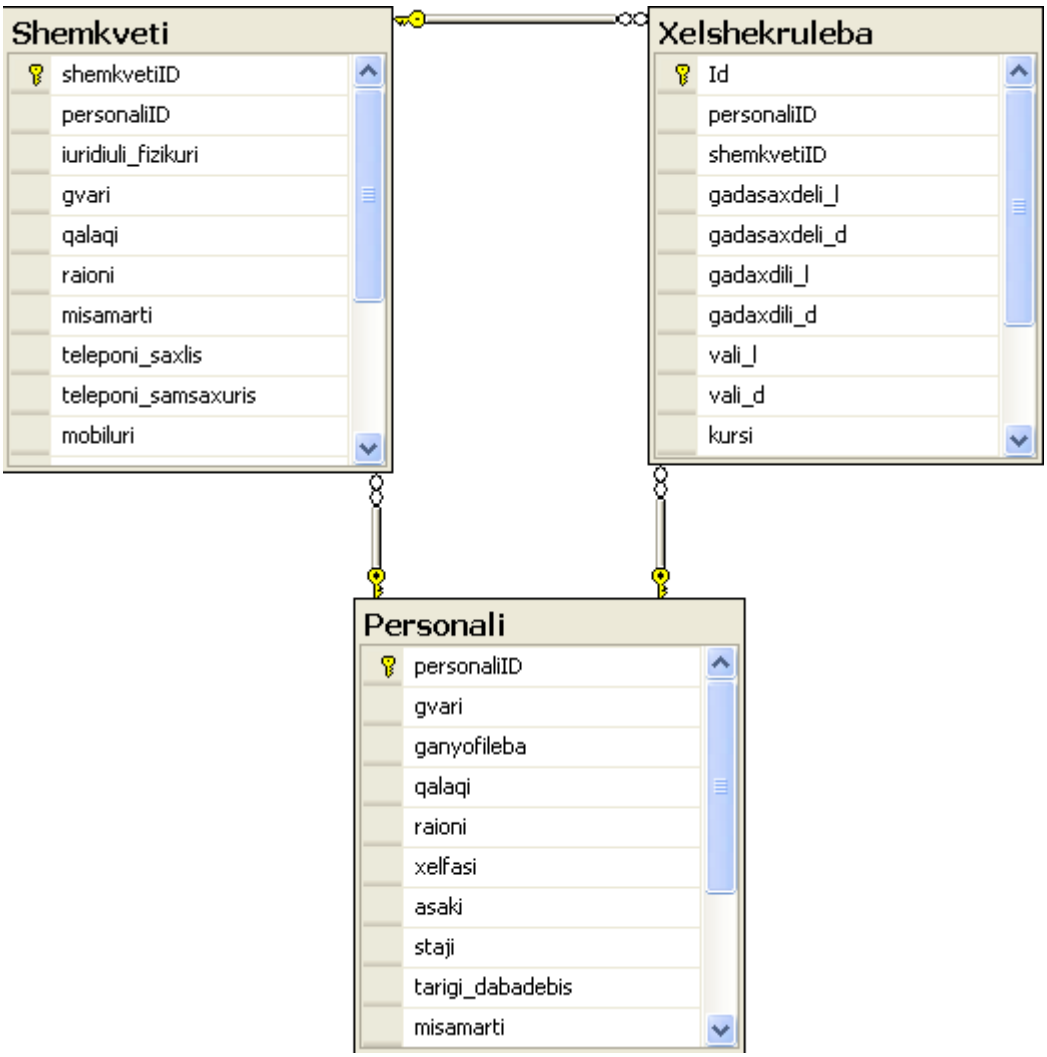
ცხრილი 2.5. Xelshekruleba ცხრილის სტრუქტურა

სვეტის სახელი	მონაცემის ტიპი
xelshekrulebaID	INT
personaliID	INT
shemkvetiID	INT
gadasaxdeli_l	FLOAT
gadasaxdeli_d	FLOAT
gadaxdili_l	FLOAT
gadaxdili_d	FLOAT
vali_l	FLOAT
vali_d	FLOAT

ცხრილი 2.5. Xelshekruleba ცხრილის სტრუქტურა (გაგრძელება)

kursi	FLOAT
tarigi_dawyebis	DATETIME
tarigi_damtavrebis	DATETIME
shesruleba	NVARCHAR(3)
visi_mizezit	NVARCHAR(15)

ცხრილებს შორის არსებულ კავშირს აქვს სახე:



ცხრილში მონაცემების მოსათავსებლად შეგვიძლია გრაფიკული ინტერფეისის გამოყენება. ამისათვის, საჭირო ცხრილის სახელზე ვხსნით კონტექსტურ მენიუს და ვასრულებთ Edit Top 200 Rows ბრძანებას.

ცხრილებში მოთავსებულია შემდეგი მონაცემები:

personaliID	qvari	saxeli	qanyofileba	qalaqi	reqioni	raioni	xelfasi	asaki	staji	tariqi_dabadebis	sqesi
1	სამხარაძე	საბა	სასპორტო	ზუგდიდი	სამეგრელო	NULL	1651,086382000...	21	2	1994-03-19 00:0...	კაცი
2	სამხარაძე	ანა	სამედიცინო	ბათუმი	აჭარა	NULL	1430,841951	26	7	1989-10-05 00:0...	ქალი
3	გაჩეჩილაძე	ლია	სამედიცინო	თბილისი	NULL	საბურთალო	1045,475332000...	54	10	1961-04-12 00:0...	ქალი
4	გაჩეჩილაძე	ხათუნა	სასოფლო	თბილისი	NULL	ვერა	979,7186630000...	46	18	1969-02-20 00:0...	ქალი
5	შენგელია	ნატა	სავაჭრო	ზუგდიდი	სამეგრელო	NULL	935,8452840000...	27	5	1988-09-16 00:0...	ქალი
6	კაპანაძე	ეთერი	სასპორტო	რუსთავი	ქართლი	NULL	1156,081531000...	49	9	1966-11-30 00:0...	ქალი
7	ქევხიშვილი	ალექსანდრე	სამედიცინო	თბილისი	NULL	ვერა	1254,353958000...	57	28	1958-12-03 00:0...	კაცი
8	ყალაბეგიშვილი	გია	სასოფლო	ქუთაისი	იმერეთი	NULL	880,1357950000...	61	25	1954-04-04 00:0...	კაცი
9	გიორგაძე	გივი	სამედიცინო	ქობულეთი	აჭარა	NULL	869,7468780000...	63	26	2052-07-05 19:5...	კაცი
10	კიკნაძე	მზია	სავაჭრო	ბათუმი	აჭარა	NULL	847,6035810000...	48	17	1967-12-09 00:0...	ქალი
11	სამხარაძე	გიორგი	სასოფლო	თბილისი	NULL	ვაკე	1078,967252000...	28	6	1987-09-07 00:0...	კაცი

shemkvetiID	iuridiuli_fizikuri	qvari	saxeli	qalaqi	reqioni	raioni	mobiluri	mobiluri_direq...	sqesi	firmis_dasaxel...
1	ფიზიკური	ნემსაძე	დოდო	თბილისი	NULL	ავლაბარი	551-743-434	NULL	ქალი	NULL
2	იურიდიული	სეხნიაშვილი	გია	თელავი	კახეთი	NULL	NULL	552-111-111	კაცი	Gia&Co.
3	იურიდიული	მამაიაშვილი	ჯემალი	თელავი	კახეთი	NULL	NULL	553-444-444	კაცი	Jemali&Co.
4	ფიზიკური	სამხარაძე	გიორგი	ქუთაისი	იმერეთი	NULL	554-222-222	NULL	კაცი	NULL
5	იურიდიული	ნონიაშვილი	ზურა	ბათუმი	აჭარა	NULL	NULL	555-555-555	კაცი	Zura&Co.
6	ფიზიკური	სამხარაძე	არჩილი	ბათუმი	აჭარა	NULL	556-565-556	NULL	კაცი	NULL
7	ფიზიკური	მიქელაძე	მერაბი	ზესტაფონი	იმერეთი	NULL	557-999-999	NULL	კაცი	NULL
8	იურიდიული	კამკამიძე	ლენა	ზუგდიდი	სამეგრელო	NULL	NULL	558-223-334	კაცი	Givi&Co.
9	იურიდიული	ჭიკაძე	გელა	გორი	ქართლი	NULL	NULL	559-333-333	კაცი	Gela&Co.
10	იურიდიული	ლომიძე	რუსუდანი	თბილისი	NULL	საბურთალო	NULL	571-334-344	ქალი	R&Co.
11	ფიზიკური	ხომტარია	სიმონი	თბილისი	NULL	მთაწმინდა	572-983-983	NULL	კაცი	NULL
12	ფიზიკური	ლასურაშვილი	ია	თბილისი	NULL	დიღომი	573-522-532	NULL	ქალი	NULL
13	ფიზიკური	სამხარაძე	ბექა	თელავი	კახეთი	NULL	574-661-621	NULL	კაცი	NULL
14	ფიზიკური	კაპანაძე	მერაბი	თბილისი	NULL	დიდუბე	574-777-564	NULL	კაცი	NULL

xelshekrulebaID	personaliID	shemkvetiID	qadasaxdeli_l	qadasaxdeli_d	qadaxdili_l	qadaxdili_d	vali_l	vali_d	kursi	tariqi_dawyebis	tariqi_she
17	5	*NULL*	5300	2849,4623655...	4800	2580,64516...	500	268,81...	1,86	2015-10-01 00:0...	*NULL*
18	6	12	4400	2365,5913978...	3900	2096,77419...	500	268,81...	1,86	2015-12-01 00:0...	*NULL*
19	7	13	3500	1881,7204301...	3500	1881,72043...	0	0	1,86	2010-07-10 00:0...	2015-05-1
20	9	14	8500	4569,8924731...	8400	4516,12903...	100	53,763...	1,86	2011-07-10 00:0...	2015-02-1
21	8	15	5600	3010,7526881...	5600	3010,75268...	0	0	1,86	2013-02-01 00:0...	*NULL*
1	2	1	5000	2631,5789473...	5000	2631,57894...	0	0	1,9	2011-01-01 00:0...	2012-02-0
2	2	2	3700	1989,2473118...	3000	1612,90322...	700	376,34...	1,86	2009-01-01 00:0...	*NULL*
3	2	2	8000	4301,0752688...	4300	2311,82795...	3700	1989,2...	1,86	2011-01-01 00:0...	2012-05-1
4	3	5	1900	1027,0270270...	1900	1027,02702...	0	0	1,85	2009-08-20 00:0...	2011-01-2
5	3	6	2500	1358,6956521...	2500	1358,69565...	0	0	1,84	2007-02-01 00:0...	2010-12-0
6	3	3	8200	4480,8743169...	8000	4371,58469...	200	109,28...	1,83	2012-02-25 00:0...	*NULL*
7	8	1	7700	4277,7777777...	7700	4277,77777...	0	0	1,8	2008-08-05 00:0...	2011-04-0
8	3	4	8000	4444,4444444...	1600	888,888888...	6400	3555,5...	1,8	2008-10-17 00:0...	2012-10-1
9	1	3	6000	3333,3333333...	5500	3055,55555...	500	277,77...	1,8	2012-10-10 00:0...	*NULL*
10	1	2	9800	5444,4444444...	9000	5000	800	444,44...	1,8	2009-05-07 00:0...	*NULL*
11	1	3	7100	4176,4705882...	7100	4176,47058...	0	0	1,7	2012-10-10 00:0...	2014-08-1
12	3	1	3000	1764,7058823...	3000	1764,70588...	0	0	1,7	2012-03-03 00:0...	2014-07-2
13	9	5	10000	5882,3529411...	10000	5882,35294...	0	0	1,7	2009-07-09 00:0...	2011-06-0
14	3	1	2000	1176,4705882...	1700	1000	300	176,47...	1,7	2012-03-03 00:0...	*NULL*
15	4	7	4400	2603,5502958...	4400	2603,55029...	0	0	1,69	2011-11-14 00:0...	2014-11-1
16	10	7	3900	2321,4285714...	3900	2321,42857...	0	0	1,68	2010-05-21 00:0...	2012-04-2

თავი 4. მონაცემების მართვა

Transact SQL-ის საფუძვლები

სტრუქტურირებული მოთხოვნების ენა (Structured Query Language, SQL) 1970 წელს შეიმუშავა IBM კორპორაციამ, როგორც მონაცემთა რელაციური ბაზების მართვის ენა. მონაცემთა ბაზების მართვის სისტემების ყველა შემმუშავებელი იყენებს SQL ენის ამა თუ იმ მოდიფიკაციას. შემუშავებული იქნა ამ ენის რამდენიმე ვერსია. ბოლოს, 1992 წელს სტანდარტების ამერიკის ეროვნულმა ინსტიტუტმა (American National Standard Institute, ANSI) შეიმუშავა სტანდარტი - SQL-92, რომელიც აღწერს სერვერის ქცევას და ახდენს მისი მუშაობის ძირითადი წესების განსაზღვრას. ამ სტანდარტის შემუშავების ერთ-ერთი მიზანი იყო SQL ენის სხვადასხვა ვარიანტის შეუთავსებლობის გადალახვა.

სერვერზე რეალიზებულია Transact-SQL-ის ის ვარიანტი (T-SQL), რომელიც უზრუნველყოფს ANSI SQL-92 სტანდარტის შესაძლებლობების დიდ ნაწილს. SQL ენას აქვს ინსტრუქციების რამდენიმე კატეგორია, რომლებიც მოიცავს: მონაცემების აღწერის ენას (Data Definition Language, DDL), მონაცემებით მანიპულირების ენასა (Data Manipulation Language, DML) და მონაცემების მართვის ენას (Data Control Language, DCL). DDL ენას საქმე აქვს განსაზღვრებებთან და შეიცავს ისეთ ბრძანებებს, როგორიცაა: CREATE, ALTER და DROP. DML ენა იძლევა მონაცემების მოთხოვნისა და შეცვლის შესაძლებლობას და შეიცავს ისეთ ბრძანებს, როგორიცაა: SELECT, INSERT, UPDATE, DELETE და MERGE. DCL ენა დაკავშირებულია მიმართვის უფლებებთან ან უფლებამოსილებებთან და შეიცავს ისეთ ბრძანებებს, როგორიცაა: GRANT და REVOKE.

ამ თავში განვიხილავთ Transact-SQL-ის მუშაობის საფუძვლებს, მონაცემთა ტიპებს, ცვლადებს, მუდმივებს, ფუნქციებს და ა.შ. ენას საფუძვლად შემდეგი ელემენტები უდევს:

– *იდენტიფიკატორები* (*identifiers*). ისინი გამოიყენება კონკრეტულ ობიექტთან მიმართვისათვის, როგორიცაა ცვლადი, ცხრილი, წარმოდგენა, სვეტი, ინდექსი, ტრიგერი და ა.შ. ამრიგად, იდენტიფიკატორი არის ობიექტის სახელი.

– *კომენტარები* (*comments*). კომენტარი არის ტექსტი, რომელიც მოთავსებულია ბრძანების კოდში და შეიცავს მის განმარტებას. Transact-SQL-ის მიერ კომენტარის დამუშავება არ ხდება. არსებობს ორი ტიპის კომენტარი - სტრიქონული და ბლოკური. სტრიქონული კომენტარი „--“ სიმბოლოებით იწყება და ერთ სტრიქონს იკავებს. ბლოკური კომენტარები „/*” სიმბოლოებით იწყება და „*/” სიმბოლოებით მთავრდება. ის რამდენიმე სტრიქონს იკავებს.

– *დარეზერვებული საკვანძო სიტყვები* (*reserved keywords*). ისინი გამოიყენება სერვერის მუშაობის მართვისთვის, არ არიან დამოკიდებული რეგისტრზე და არ შეიძლება წარმოადგენდნენ იდენტიფიკატორებს.

გამოსახულებები

გამოსახულება (*expression*) წარმოადგენს იდენტიფიკატორების, ფუნქციების, არითმეტიკისა და ლოგიკის ოპერაციების ნიშნების, მუდმივებისა და სხვა ობიექტების კომბინაციას. გამოსახულება შედგება *ოპერანდებისა* (მონაცემებისა) და *ოპერატორებისაგან* (ოპერაციის ნიშნებისაგან). არსებობს შემდეგი ტიპის ოპერანდები:

– *მუდმივები*. ეს მუდმივი სიდიდეებია, რომელთა მნიშვნელობები არ იცვლება. მუდმივა შეიძლება იყოს მთელი რიცხვი, წილადი, სტრიქონი და ა.შ.

– *ცვლადები*. ცვლადი არის გარკვეული ზომის მქონე მეხსიერების სახელდებული უბანი, რომელშიც მონაცემები ინახება.

– *სვეტების სახელები*. სვეტის სახელი შეიძლება გამოვიყენოთ როგორც ოპერანდი

გამოსახულებაში. ამ დროს სვეტის სახელის ნაცვლად სერვერი ავტომატურად ჩასვამს შესაბამის მნიშვნელობას. ცხრილის ერთი სტრიქონიდან მეორეზე გადასვლის დროს სვეტის სახელს შესაბამისი მნიშვნელობა ენიჭება.

– *ფუნქციები*. ესაა სახელდებული, მცირე ზომის პროგრამები, რომლებიც ახდენენ მონაცემების დამუშავებას და შედეგის დაბრუნებას. ფუნქციას შეიძლება ჰქონდეს ან არ ჰქონდეს პარამეტრები.

– *ქვემოთხოვნები*. გამოსახულება შეიძლება შეიცავდეს ქვემოთხოვნას, რომელიც ახდენს მონაცემების შესაბამისი ნაკრების მომზადებას.

იდენტიფიკატორები

არსებობს იდენტიფიკატორების ორი ტიპი:

- *სტანდარტული იდენტიფიკატორები* (*regular identifiers*). მათი განსაზღვრისას დაცულია იდენტიფიკატორების შექმნის წესები.

მაგალითი. მოყვანილ მოთხოვნაში Personali და personaliID არის სტანდარტული იდენტიფიკატორები:

```
SELECT * FROM Personali WHERE personaliID = 5;
```

- *შეზღუდული იდენტიფიკატორები* (*delimited identifiers*). მათი განსაზღვრისას არ არის დაცული იდენტიფიკატორების შექმნის წესები. ასეთი იდენტიფიკატორები უნდა მოვათავსოთ კვადრატულ ფრჩხილებში ([ ]) ან ბრჭყალებში (“ “).

მაგალითი. მოყვანილ მოთხოვნაში [MY TABLE] და [ORDER] არიან შეზღუდული იდენტიფიკატორები:

```
SELECT * FROM [MY TABLE]  WHERE [ORDER] = 5;
```

სტანდარტული იდენტიფიკატორის განსაზღვრის წესები. იდენტიფიკატორის პირველი სიმბოლო შემდეგ წესებს უნდა აკმაყოფილებდეს:

– უნდა შეესაბამებოდეს Unicode Standard 2.0 სტანდარტს. ამ სტანდარტის მიხედვით ეს სიმბოლო უნდა იყოს ნებისმიერი ლათინური სიმბოლო a-დან z-მდე, აგრეთვე, ეროვნული ანბანის სიმბოლო.

– შეიძლება იყოს ხაზგასმის სიმბოლო, @ ან # სიმბოლო. ზოგიერთ სიმბოლოს იდენტიფიკატორის დასაწყისში სპეციალური დანიშნულება აქვს. იდენტიფიკატორი, რომელიც # სიმბოლოთი იწყება, აღნიშნავს ლოკალურ დროებით ობიექტს, მაგალითად, ცხრილს ან პროცედურას. იდენტიფიკატორი, რომელიც ## სიმბოლოებით იწყება, აღნიშნავს გლობალურ დროებით ობიექტს. იდენტიფიკატორი, რომელიც @ სიმბოლოთი იწყება, აღნიშნავს ცვლადს.

იდენტიფიკატორის მომდევნო სიმბოლოები შეიძლება იყოს:

– Unicode Standard 2.0 სტანდარტით განსაზღვრული სიმბოლოები.

– ათობითი ციფრები.

– @, $, _, # სიმბოლოები.

იდენტიფიკატორის შიგნით დაუშვებელია სპეციალური სიმბოლოების გამოყენება: ~, !, %, ^, &, - , (,), {, }, “, ., \, ' და ინტერვალი. იდენტიფიკატორი არ უნდა იყოს დარეზერვებული სიტყვა და უნდა იყოს უნიკალური. სახელების განსხვავება რეგისტრის მიხედვით არ ხდება.

ზოგიერთი შეზღუდვის გვერდის ასავლელად შეგვიძლია ობიექტების სახელები ბრჭყალებში ან კვადრატულ ფრჩხილებში მოვათავსოთ ანუ შეზღუდული იდენტიფიკატორები გამოვიყენოთ. ასეთ შემთხვევაში, ობიექტის სახელში დასაშვებია ინტერვალები და სპეციალური სიმბოლოები, აგრეთვე დარეზერვებული სიტყვები.

ნებისმიერი ობიექტი გარკვეული მომხმარებლის მიერ იქმნება და ამა თუ იმ მონაცემთა

ბაზას ეკუთვნის. თავის მხრივ, მონაცემთა ბაზა მოთავსებულია კონკრეტულ სერვერზე. სერვერის, მონაცემთა ბაზის, სქემისა და ობიექტის სახელის საფუძველზე იქმნება ობიექტის *სრული სახელი* (*complete name*) ან *სრულად განსაზღვრული სახელი* (*full qualified name*), რომელსაც შემდეგი სინტაქსი აქვს:

[[[სერვერის_სახელი.] [მონაცემთა_ბაზის_სახელი].] [სქემის_სახელი].] ობიექტის_სახელი

როგორც სინტაქსიდან ჩანს, აუცილებელია მხოლოდ ობიექტის სახელის მითითება. დანარჩენი ნაწილების მითითება დამოკიდებულია კონკრეტულ სიტუაციაზე. მოვიყვანოთ სხვადასხვა ვარიანტი:

სერვერის_სახელი.მონაცემთა_ბაზის_სახელი.სქემის_სახელი.ობიექტის_სახელი
სერვერის_სახელი.მონაცემთა_ბაზის_სახელი..ობიექტის_სახელი
სერვერის_სახელი..სქემის_სახელი.ობიექტის_სახელი
სერვერის_სახელი...ობიექტის_სახელი
მონაცემთა_ბაზის_სახელი.სქემის_სახელი.ობიექტის_სახელი
მონაცემთა_ბაზის_სახელი..ობიექტის_სახელი
სქემის_სახელი.ობიექტის_სახელი
ობიექტის_სახელი

იმისათვის, რომ მივმართოთ ცხრილის ან წარმოდგენის რომელიმე სვეტს, აუცილებელია სრულ სახელში სვეტის სახელის დამატება:

[[[სერვერის_სახელი.] [მონაცემთა_ბაზის_სახელი].] [სქემის_სახელი].] ობიექტის_სახელი.
სვეტის_სახელი

სერვერის ნებისმიერ ობიექტს უნდა ჰქონდეს უნიკალური, სრულად განსაზღვრული სახელი. მაგალითად, ერთ ბაზაში შეიძლება იყოს ერთნაირი სახელის მქონე ორი ცხრილი, რომლებიც სხვადასხვა სქემაში იქნება მოთავსებული.

სვეტების სახელები უნდა იყოს უნიკალური ცხრილის ან წარმოდგენის შიგნით. დავუშვათ, ბაზაში გვაქვს Cxrili_1 და Cxrili_2 ცხრილები და თითოეულ მათგანს აქვს Sveti_1 სახელის მქონე სვეტი. თუ მოთხოვნაში მივმართავთ ორივე ცხრილის Sveti_1 სვეტს, მაშინ სვეტის სახელის წინ უნდა მივუთითოთ ცხრილის სახელი, მაგალითად, Cxrili_1.Sveti_1 და Cxrili_2.Sveti_1.

ობიექტის შექმნისას შეგვიძლია არ მივუთითოთ სერვერის, მონაცემთა ბაზისა და სქემის სახელი. თუ ვასრულებთ მოთხოვნას, რომელიც ბევრ ობიექტთან მუშაობს, მაშინ უმჯობესია ობიექტების სრული სახელების გამოყენება პრობლემების თავიდან აცილების მიზნით. თუ ობიექტთან მიმართვისას არ არის მითითებული სერვერის სახელი, მაშინ აიღება მიმდინარე სერვერის სახელი. თუ მითითებული არ არის მონაცემთა ბაზის სახელი, მაშინ აიღება მიმდინარე მონაცემთა ბაზა. თუ მითითებული არ არის სქემის სახელი, მაშინ აიღება dbo სქემა. ის ავტომატურად იქმნება თითოეულ მონაცემთა ბაზაში, და გამოიყენება, როგორც ნაგულისხმევი სქემა იმ შემთხვევაში, როცა მომხმარებელი აშკარად არ არის დაკავშირებული სხვა სქემასთან. თუ მითითებული ობიექტი არ მოიძებნა, მაშინ გაიცემა შეტყობინება შეცდომის შესახებ. სქემები განხილულია მე-21 თავში.

შემზღუდველების გამოყენება. შემზღუდველების (“ “ და [ ] სიმბოლოები) გამოყენების წესები დამოკიდებულია QUOTED_IDENTIFIER პარამეტრის მნიშვნელობაზე. მას მნიშვნელობა შეგვიძლია SET ბრძანებით მივანიჭოთ. მისი სინტაქსია:

SET QUOTED_IDENTIFIER { ON | OFF }

თუ QUOTED_IDENTIFIER პარამეტრი იმყოფება ON მდგომარეობაში, მაშინ შემზღუდველების გამოყენება შემდეგნაირად ხდება:

– ბრჭყალები (“ “) და კვადრატული ფრჩხილები ([ ]) გამოიყენება იდენტიფიკატორების შეზღუდვისათვის. ბრჭყალები (“ “) არ გამოიყენება სიმბოლური სტრიქონებისათვის.

– სიმბოლური სტრიქონები უნდა მოთავსდეს აპოსტროფებში (' '). იდენტიფიკატორის შეზღუდვისათვის აპოსტროფების გამოყენება არ შეიძლება. თუ სიმბოლური სტრიქონი აპოსტროფს შეიცავს, მაშინ ამ აპოსტროფის გვერდით მეორე აპოსტროფი უნდა მოვათავსოთ.

მაგალითი.

```
SET QUOTED_IDENTIFIER ON;
SELECT * FROM "MY TABLE" WHERE "Last Name" = N'O''Brien';
ან
SET QUOTED_IDENTIFIER ON;
SELECT * FROM [MY TABLE] WHERE [Last Name] = N'O''Brien';
```

თუ QUOTED_IDENTIFIER პარამეტრი იმყოფება OFF მდგომარეობაში, მაშინ შემზღუდველების გამოყენება შემდეგნაირად ხდება:

– იდენტიფიკატორების შეზღუდვისათვის კვადრატული ფრჩხილები ([ ]) გამოიყენება.

– სიმბოლური სტრიქონები უნდა მოთავსდეს აპოსტროფებში (' ') ან ბრჭყალებში (" "). თუ სტრიქონი აპოსტროფს შეიცავს, მაშინ ეს სტრიქონი შეგვიძლია ბრჭყალებში მოვათავსოთ.

მაგალითი.

```
SET QUOTED_IDENTIFIER OFF;
SELECT * FROM [MY TABLE] WHERE [Last Name] = N'O''Brien';
ან
SET QUOTED_IDENTIFIER OFF;
SELECT * FROM [MY TABLE] WHERE [Last Name] = "O'Brien";
```

ამრიგად, არსებობს ბრჭყალების გამოყენების ორი გზა: როგორც ობიექტის სახელის შემზღუდველი (SET QUOTED_IDENTIFIER ON) და როგორც ტექსტური სტრიქონის შემზღუდველი (SET QUOTED_IDENTIFIER OFF). კვადრატული ფრჩხილები, როგორც შემზღუდველები, გამოიყენება ყოველთვის მიუხედავად QUOTED_IDENTIFIER პარამეტრის მნიშვნელობისა.

ცვლადები

ცვლადის გამოყენებამდე საჭიროა მისი გამოცხადება DECLARE ბრძანების საშუალებით. მისი სინტაქსია:

```
DECLARE @ლოკალური_ცვლადი მონაცემთა_ტიპი [,...n]
```

ცვლადის გამოცხადებისას ეთითება მისი სახელი, რომელიც აუცილებლად უნდა იწყებოდეს @ სიმბოლოთი. სახელის მომდევნო სიმბოლოები უნდა აკმაყოფილებდეს ობიექტების სახელდების წესებს. სახელის შემდეგ ეთითება ცვლადის ტიპი.

@ სიმბოლო ლოკალური ცვლადების სახელების გარდა გამოიყენება, აგრეთვე, ფუნქციებისა და შენახული პროცედურების არგუმენტების სახელების განსაზღვრისათვის.

სინტაქსის [,...n] ნაწილი მიუთითებს, რომ ერთი DECLARE ბრძანებით შეიძლება რამდენიმე ცვლადის გამოცხადება, რომელთა სახელები ერთმანეთისაგან მძიმეებით იქნება გამოყოფილი.

ცვადების გამოცხადების მაგალითი:

```
DECLARE @Asaki INT, @@ NVARCHAR(20), @1_@ FLOAT, @$ INT, @# INT;
```

ცვლადებს მნიშვნელობები შეგვიძლია მივანიჭოთ SET და SELECT ბრძანებების გამოყენებით:

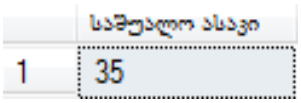
```
DECLARE @Asaki INT, @@ NVARCHAR(20), @1_@ FLOAT, @$ INT, @# INT;
SET @Asaki = 7;
SET @@ = N'ანა სამხარაძე';
```

```
SELECT @1_@ = 5.5;
SELECT @$ = 100;
SELECT @# = 150;
```

ცვლადს შეგვიძლია, აგრეთვე, მივანიჭოთ რომელიმე ფუნქციის მუშაობის შედეგი:

```
DECLARE @sashualo_asaki FLOAT;
SELECT @sashualo_asaki = AVG(asaki) FROM Personali;
SELECT @sashualo_asaki AS [საშუალო ასაკი];
```

შედეგი:

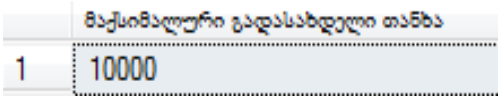


	საშუალო ასაკი
1	35

ცვლადს შეგვიძლია მივანიჭოთ მოთხოვნის მუშაობის შედეგი:

```
DECLARE @tanxa FLOAT;
SET @tanxa =
(
SELECT MAX(gadasaxdeli_l)
FROM Xelshekruleba
);
SELECT @tanxa AS [მაქსიმალური გადასახდელი თანხა];
```

შედეგი:



	მაქსიმალური გადასახდელი თანხა
1	10000

ცვლადების მნიშვნელობების გამოსატანად შეგვიძლია PRINT და SELECT ბრძანებების გამოყენება. SELECT ბრძანებებას მონაცემები გამოაქვს სტრიქონების სტანდარტულ ნაკრებში (recordset). PRINT ბრძანებებას მონაცემები გამოაქვს როგორც სამომსახურეო. ასეთი სტრიქონის მაგალითია შემდეგი სტრიქონი:

```
(1 row(s) affected)
```

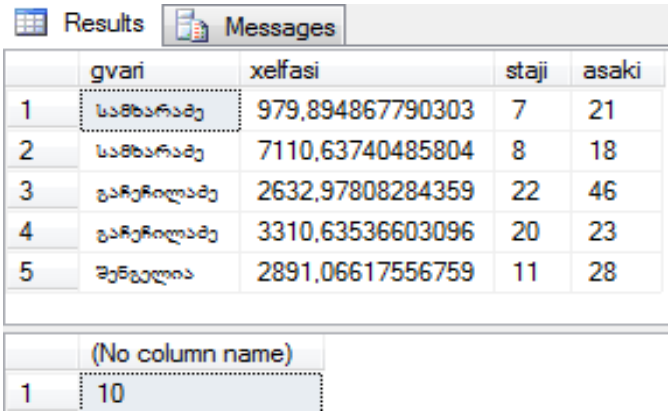
რომელიც გამოჩნდება, მაგალითად, ცხრილიდან სტრიქონების ამორჩევის შემდეგ.

ცვლადების მნიშვნელობების გამოსატანის მაგალითები:

```
SELECT TOP 5 gvari, xelfasi, staji, asaki
FROM Personali;
SELECT 10;
PRINT 230;
```

პირველი ბრძანების შესრულების შედეგად სტრიქონების სტანდარტულ ნაკრებში 5 სტრიქონი გამოჩნდება, მეორე ბრძანების შესრულების შედეგად, კი - რიცხვი 10 (Grids განყოფილება):

შედეგი:

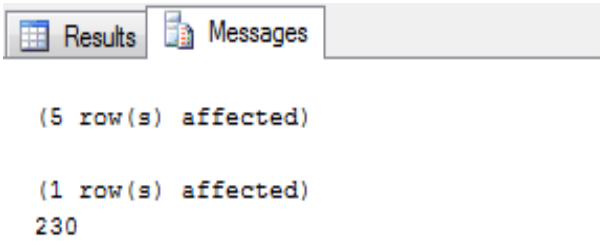


Results | Messages

	gvari	xelfasi	staji	asaki
1	სამხარაძე	979,894867790303	7	21
2	სამხარაძე	7110,63740485804	8	18
3	გაჩეჩილაძე	2632,97808284359	22	46
4	გაჩეჩილაძე	3310,63536603096	20	23
5	შენგელია	2891,06617556759	11	28

	(No column name)
1	10

სამივე ბრძანების შესრულების შედეგად გაიცემა შემდეგი სამომსახურეო ინფორმაცია (Messages განყოფილება):



Results | Messages

```
(5 row(s) affected)

(1 row(s) affected)
230
```

თუ ფანჯარაში არ ჩანს Grids და Messages ჩანართები, მაშინ უნდა შევასრულოთ Query→Results To→Results to Grid ბრძანება. Results განყოფილებაში გამოჩნდება SELECT ბრძანების მუშაობის შედეგი, ხოლო Messages განყოფილებაში - კი PRINT ბრძანების მუშაობის შედეგი.

ოპერატორები

ოპერატორები (*operators*) - იმ ოპერაციების ნიშნებია, რომლებიც უნდა შესრულდეს ერთ ან მეტ მარტივ გამოსახულებაზე უფრო რთული გამოსახულების შესაქმნელად. მაგალითად, tanxa * 1.2.

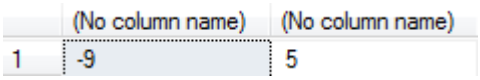
არსებობს ოპერატორების შემდეგი ტიპები:

– *უმარტივესი ოპერატორები*. ესაა უნარული ოპერატორები, რომლებიც მხოლოდ ერთ ოპერანდთან მუშაობენ. უმარტივესი ოპერატორები გამოიყენება როგორც მთელი, ისე წილადი რიცხვების მიმართ. არსებობს სამი ასეთი ოპერატორი: „+” - დადებითი რიცხვის აღნიშვნისათვის, „-“ - უარყოფითი რიცხვის აღნიშვნისათვის და „~” - რიცხვის დამატების მისაღებად.

მაგალითი. მოთხოვნას ეკრანზე გამოაქვს უარყოფითი და დადებითი რიცხვები:

```
SELECT -9, +5;
```

შედეგი:



	(No column name)	(No column name)
1	-9	5

– *მინიჭების ოპერატორი*. ის „=” სიმბოლოთი აღინიშნება და გამოიყენება ცვლადისათვის გარკვეული მნიშვნელობის მისანიჭებლად, აგრეთვე როგორც შედარების ოპერატორი.

მაგალითი.

```
DECLARE @cvladi INT;
SET @cvladi = 25;
SELECT @cvladi;
```

აქ DECLARE არის ცვლადების გამოცხადების, SET კი - მინიჭების ოპერატორი.

– *არითმეტიკის ოპერატორები.* ესაა ბინარული ოპერაციების ნიშნები, რომლებიც რიცხვებზე სრულდება. არითმეტიკის ოპერატორებია: + (შეკრება), - (გამოკლება), * (გამრავლება), / (გაყოფა) და % (ნაშთის მიღება). შეკრებისა და გამოკლების ოპერაციები შეიძლება შესრულდეს, აგრეთვე, DATETIME და SMALLDATETIME ტიპის ცვლადებზე.

მაგალითი. მოთხოვნაში ორი მათემატიკური გამოსახულება მძიმით არის გამოყოფილი:

```
SELECT 5 + 6 * 8, ( 5 - 3 ) / 2;
```

შედეგი:

	(No column name)	(No column name)
1	53	1

– *სტრიქონული ოპერატორი.* + არის კონკატენაციის (სტრიქონების გაერთიანების) ოპერატორი. მაგალითი:

```
DECLARE @gvari NVARCHAR (20), @saxeli NVARCHAR (15);
SET @saxeli = N'საბა';
SET @gvari   = N'სამხარაძე';
SELECT @saxeli + '   ' + @gvari AS [სახელი და გვარი];
```

შედეგი:

	სახელი და გვარი
1	საბა სამხარაძე

ცხრილი 4.1. შედარების ოპერატორები.

შედარების ოპერატორი	აღწერა
=	ტოლია
>	მეტია
<	ნაკლებია
>=	მეტია ან ტოლი
<=	ნაკლებია ან ტოლი
<> ან !=	არ არის ტოლი
!<	არ არის ნაკლები
!>	არ არის მეტი

– *შედარების ოპერატორები.* ისინი მოყვანილია ცხრილში 4.1.

შედარების ოპერატორის მუშაობის შედეგია true (ჭეშმარიტი) თუ პირობა სრულდება და false (მცდარი) წინააღმდეგ შემთხვევაში. თუ შედარება შეუძლებელია, მაგალითად, ტიპებია განსხვავებული, მაშინ გაიცემა NULL. შედარების შედეგების დასამუშავებლად IF ოპერატორი გამოიყენება.

– *ლოგიკის ოპერატორები.* ლოგიკის ოპერატორებია: ALL, AND, ANY, BETWEEN, EXISTS, IN, LIKE, NOT, OR და SOME. AND, OR და NOT ოპერატორები მოყვანილია ცხრილში 4.2. მათი მუშაობის შედეგად გაიცემა TRUE ან FALSE მნიშვნელობა.

ცხრილი 4.2. ლოგიკის ოპერატორების ჭეშმარიტების ცხრილი

B1	B2	B1 AND B2	B1 OR B2	B1 XOR B2	NOT B1
true	true	true	true	false	false
true	false	false	true	true	false
false	true	false	true	true	true
false	false	false	false	false	true

– *ბიტობრივი ოპერატორები.* ეს ოპერატორები ასრულებენ ბიტობრივ ოპერაციებს რიცხვებზე. ბიტობრივი ოპერატორებია: ბიტობრივი AND (&), ბიტობრივი OR (|) და ბიტობრივი XOR (გამომრიცხავი OR, ^).

მაგალითი. მოთხოვნა ასრულებს &, | და ^ ბიტობრივ ოპერაციებს:

```
SELECT 13 & 4 AS [AND ოპერაცია], 9 | 4 AS [OR ოპერაცია], 8 ^ 4 AS [XOR ოპერაცია];
```

შედეგი:

	AND ოპერაცია	OR ოპერაცია	XOR ოპერაცია
1	4	13	12

AND, OR და NOT ოპერატორებს უფრო დაწვრილებით WHERE განყოფილების შესწავლისას განვიხილავთ.

ცხრილში მონაცემების ჩამატება

ცხრილში შეგვიძლია ჩავუმატოთ, შევცვალოთ, წავშალოთ ან ამოვირჩიოთ მონაცემები. შესაბამისად მონაცემების დასამატებლად ძირითადად INSERT ბრძანება, მონაცემების შესაცვლელად UPDATE ბრძანება, წასაშლელად DELETE ბრძანება, ხოლო ამოსარჩევად კი SELECT ბრძანება გამოიყენება. ამ თავში დაწვრილებით განვიხილავთ თითოეულ მათგანს.

არსებობს ცხრილში მონაცემების ჩამატების რამდენიმე გზა:

– INSERT ბრძანების გამოყენება. მისი საშუალებით ცხრილში შეიძლება ერთი ან მეტი სტრიქონის ჩამატება.

– SELECT INTO ბრძანების გამოყენება. ამ დროს ცხრილში ჩაიწერება SELECT მოთხოვნის მიერ გაცემული სტრიქონები.

– bcp.exe და BULK INSERT ბრძანებების გამოყენება. ამ დროს საწყისი მონაცემები ცხრილს ტექსტური ფაილიდან ჩაემატება.

– მონაცემთა ბაზის პროგრამული ინტერფეისის გამოყენება (Database API). ის ითვალისწინებს ADO, OLE DB, ODBC და DB ბიბლიოთეკების გამოყენებას.

– Data Transformation Services ტექნოლოგია. ის საშუალებას გვაძლევს შევასრულოთ მონაცემების გადატანისა და გარდაქმნის ოპერაციები მონაცემების ისეთი წყაროებიდან, როგორიცაა Oracle, Excel, Paradox, Access და ა.შ.

INSERT ბრძანება

ის იძლევა ცხრილში ერთი ან მეტი სტრიქონის ჩამატების საშუალებას. მისი სინტაქსია:
INSERT [INTO] { *ცხრილის_სახელი* | *წარმოდგენის_სახელი* }
{ [(*სვეტების_სია*)]
{ VALUES ({ DEFAULT | NULL | *გამოსახულება* })
| *მიღებული_ცხრილი* } }
| DEFAULT VALUES

განვიხილოთ არგუმენტების დანიშნულება.

– [INTO] არააუცილებელი არგუმენტია, რომელსაც მოსდევს იმ ცხრილის სახელი, რომელშიც მონაცემების ჩამატება ხდება;

– *ცხრილის_სახელი* იმ ცხრილის სახელია, რომელშიც სტრიქონების ჩამატება ხდება;

– *წარმოდგენის_სახელი* იმ წარმოდგენის სახელია, რომელშიც ხდება სტრიქონების ჩამატება. უნდა გვახსოვდეს, რომ სტრიქონების ჩამატება შესაძლებელია წარმოდგენის მხოლოდ ერთ ცხრილში;

– (*სვეტების_სია*) იმ სვეტების სახელების სიაა, რომლებშიც უნდა მოხდეს მონაცემების ჩამატება. სვეტების მნიშვნელობები ეთითება VALUES არგუმენტში. თუ სვეტების სია მითითებული არ არის, მაშინ VALUES არგუმენტში მითითებული მნიშვნელობების რაოდენობა და მიმდევრობა უნდა შეესაბამებოდეს ცხრილის სვეტების რაოდენობას და მიმდევრობას. თუ გვინდა რამდენიმე სვეტის მნიშვნელობის შეტანა ჩვენთვის სასურველი მიმდევრობით, მაშინ მათი სახელები ასეთივე მიმდევრობით უნდა მივუთითოთ *სვეტების_სია* არგუმენტში. შესაბამისად, VALUE არგუმენტში ვუთითებთ მონაცემებს შესაბამისი მიმდევრობით. სვეტის მნიშვნელობა შეგვიძლია არ მივუთითოთ მხოლოდ იმ შემთხვევაში, თუ ამ სვეტისთვის განსაზღვრულია IDENTITY თვისება, NULL მნიშვნელობის შენახვის შესაძლებლობა ან timestap ტიპი.

– VALUES ({ DEFAULT | NULL | გამოსახულება }). VALUES საკვანძო სიტყვა განსაზღვრავს ცხრილში ჩასასმელ მონაცემებს. მისი არგუმენტების რაოდენობა უნდა ემთხვეოდეს ცხრილში ან *სვეტების_სია* არგუმენტში მითითებული სვეტების რაოდენობას. DEFAULT არგუმენტი მიუთითებს, რომ მოხდება ნაგულისხმევი (წინასწარ განსაზღვრული) მნიშვნელობების ჩასმა. თუ სვეტისთვის ნაგულისხმევი მნიშვნელობა განსაზღვრული არ არის და ნებადართულია NULL მნიშვნელობის შენახვა, მაშინ სვეტში NULL მნიშვნელობა მოთავსთავდება. DEFAULT არგუმენტის გამოყენება არ შეიძლება იმ სვეტებისათვის, რომლებისთვისაც განსაზღვრულია IDENTITY თვისება. NULL არგუმენტი მიუთითებს, რომ სვეტში მოთავსთდება NULL მნიშვნელობა თუ ამ სვეტისათვის ნებადართულია NULL მნიშვნელობის შენახვა. გამოსახულება არგუმენტი შეიძლება იყოს მუდმივა, ცვლადი ან გამოსახულება და განსაზღვრავს სვეტში ჩასასმელ მნიშვნელობას. მნიშვნელობას უნდა ჰქონდეს ისეთივე ტიპი, როგორიც აქვს შესაბამის სვეტს და უნდა აკმაყოფილებდეს შეზღუდვის პირობებს (თუ შეზღუდვა განსაზღვრულია).

– *მიღებული_ცხრილი* არგუმენტი შეიძლება შეიცავდეს SELECT ბრძანებას, რომლის საშუალებითაც მოცემულ ცხრილში ჩასასმელ სტრიქონებს სხვა ცხრილიდან მივიღებთ. ორივე ცხრილს ერთნაირი სვეტები უნდა ჰქონდეს.

– DEFAULT VALUES არგუმენტის მითითების შემთხვევაში თითოეულ სვეტში მოთავსდება ნაგულისხმევი მნიშვნელობა. თუ სვეტისათვის ასეთი მნიშვნელობა განსაზღვრული არ არის და ნებადართულია NULL მნიშვნელობის შენახვა, მაშინ სვეტში მოთავსდება NULL მნიშვნელობა. თუ ნაგულისხმევი მნიშვნელობა განსაზღვრული არ არის და არაა ნებადართული NULL მნიშვნელობის შენახვა, მაშინ გაიცემა შეტყობინება შეცდომის შესახებ.

მაგალითები.

ა. Shekveta ბაზის Personali ცხრილს ჩავუმატოთ ერთი სტრიქონი. ჩასამატებელი მნიშვნელობების მიმდევრობა უნდა შეესაბამებოდეს სვეტების მიმდევრობას ცხრილში. შედეგი ეკრანზე გამოვიტანოთ. მოთხოვნას აქვს სახე:

```
USE Shekveta;
SET DATEFORMAT DMY;
INSERT INTO Personali
VALUES (N'სამხარაძე', N'ბექა', N'სასოფლო', N'თბილისი', NULL, N'ვაკე', 300.55, 27, 5, '07.09.1980', N'კაცი', N'პეტრიაშვილის ქ.1', '221-11-00', '899-012345', N'beqa@geo.net.ge', 9);
SELECT * FROM Personali;
```

შედეგი:

personaliID	gvari	sakheli	ganyofileba	qalaqi	raioni	xelfasi	asaki	staji
14	სამხარაძე	ბექა	სასოფლო	თბილისი	ვაკე	300,55	27	5

INSERT მოთხოვნაში არ არის მითითებული personaliID სვეტი, რადგან მის IDENTITY თვისებას Yes მნიშვნელობა აქვს მინიჭებული და სერვერი თვითონ შეასრულებს ამ სვეტის მნიშვნელობის გენერირებას.

ჩვეულებრივ, სერვერზე თარიღი წარმოდგენილია ამერიკული ფორმატით: YMD (წელი-თვე-დღე). იმისათვის, რომ თარიღი ჩვენთვის ჩვეული ფორმატით (DMY, დღე-თვე-წელი) შევიტანოთ INSERT მოთხოვნაში, წინასწარ უნდა შევასრულოთ ბრძანება - SET DATEFORMAT DMY. როგორც ვხედავთ, INSERT მოთხოვნაში თარიღი შეტანილია ქართული ფორმატის მიხედვით - '07.09.1980', ანუ 1980 წლის 7 სექტემბერი, ხოლო SELECT მოთხოვნით გამოტანის დროს თარიღი მაინც ამერიკული ფორმატის მიხედვით გამოჩნდება - '1980-09-07'.

ბ. თუ გვინდა მონაცემების მინიჭება ჩვენთვის საჭირო მიმდევრობით, მაშინ INSERT ბრძანებაში უნდა მივუთითოთ სვეტების სახელებიც. მოთხოვნას აქვს სახე:

```
USE Shekveta;
SET DATEFORMAT DMY;
INSERT INTO Personali (saxeli, gvari, qalaqi, raioni, misamarti_saxlis, staji, tarigi_dabadebis, asaki,
                       mobiluri, sqesi, teleponi_saxlis, email, ganyofileba, ierarqia)
VALUES (N'გიორგი', N'სამხარაძე', N'თბილისი', N'ჩუღურეთი', N'ც. დადიანის 2', 3, '14.01.1982', 25, N'877-102332', N'კაცი', N'261-11-12', N'giorgi@geo.net.ge', N'სავაჭრო', 9);
SELECT * FROM Personali;
```

შედეგი:

personaliID	gvari	sakheli	ganyofileba	qalaqi	raioni	xelfasi	asaki	staji
15	სამხარაძე	გიორგი	სავაჭრო	თბილისი	ჩუღურეთი	NULL	25	3

გ. თუ სვეტის მნიშვნელობა შეიძლება იყოს NULL, მაშინ ცხრილში სტრიქონის ჩამატების დროს ასეთი სვეტის მნიშვნელობა შეგვიძლია გამოვტოვოთ ან მის ნაცვლად მივუთითოთ NULL. მოთხოვნას აქვს სახე:

```
USE Shekveta;
INSERT INTO Personali VALUES (N'ზვიადი', N'გიორგაძე', N'სასპორტო', N'ქობულეთი', N'აჭარა', NULL, NULL, NULL, 10, '02.03.1985', NULL, NULL, NULL, NULL, NULL, NULL);
SELECT * FROM Personali;
```

შედეგი:

personaliID	gvari	sakheli	ganyofileba	qalaqi	raioni	xelfasi	asaki	staji
10	გიორგაძე	ზვიადი	სამედიცინო	ქობულეთი	NULL	636,12763576	20	22

დ. INSERT ბრძანებით შესაძლებელია ერთი ცხრილიდან მეორეში სტრიქონების დამატება. მაგალითად, Personali ცხრილიდან Personali_1 ცხრილს დავუმატოთ მონაცემები თბილისელი თანამშრომლების შესახებ. Personali_1 ცხრილის PersonaliID სვეტი არ უნდა იყოს პირველადი გასაღები და მის IDENTITY თვისებას უნდა ჰქონდეს NO მნიშვნელობა, რადგან დამატების შემდეგ ამ სვეტში გამეორებადი მნიშვნელობები შეიძლება აღმოჩნდეს. მოთხოვნას აქვს სახე:

```
USE Shekveta;
INSERT INTO Personali_1
SELECT *
FROM Personali
WHERE qalaqi = N'თბილისი';
SELECT * FROM Personali_1;
```

შედეგი:

1	სამხარაძე	საბა	სამედიცინო	თბილისი	დიდუბე	890,813516173003	21	7
3	გაჩეჩილაძე	ლია	სავაჭრო	თბილისი	საბურთალო	2393,61643894872	46	22
4	გაჩეჩილაძე	ხათუნა	სასოფლო	თბილისი	ვერა	3009,6685145736	23	20
7	ქუხიშვილი	ალექსანდრე	სამედიცინო	თბილისი	ვერა	6557,9532715199	53	28

ე. Personali ცხრილიდან მონაცემები შეგვიძლია მოვათავსოთ, აგრეთვე, #DroebitiCxrili დროებით ცხრილში. მოთხოვნას აქვს სახე:

```
USE Shekveta;
--      თუ ცხრილი არსებობს, მაშინ ის წაიშლება
IF EXISTS
(
SELECT *
FROM tempdb..sysobjects
WHERE id = OBJECT_ID(N'tempdb..#DroebitiCxrili')
)
DROP TABLE #DroebitiCxrili;
--      ცხრილის შექმნა
CREATE TABLE #DroebitiCxrili
(
gvari           NVARCHAR (30),
ganyofileba     NVARCHAR (30),
qalaqi          NVARCHAR (30),
asaki           INT
);
--      სტრიქონების ჩამატება და ეკრანზე გამოტანა
INSERT INTO #DroebitiCxrili
SELECT gvari, ganyofileba, qalaqi, asaki
```

```
FROM Personali
WHERE qalaqi = N'თბილისი';
SELECT * FROM #DroebitiCxrili;
```

შედეგი:

	gvari	ganyofileba	qalaqi	asaki
1	სამხარაძე	სამედიცინო	თბილისი	21
2	გაჩეჩილაძე	სავაჭრო	თბილისი	46
3	გაჩეჩილაძე	სასოფლო	თბილისი	23
4	ქუხიშვილი	სამედიცინო	თბილისი	53
5	სამხარაძე	სასოფლო	თბილისი	27
6	სამხარაძე	სავაჭრო	თბილისი	25

თავდაპირველად მოწმდება დროებითი ცხრილის არსებობა. თუ ის არსებობს, მაშინ ის ჯერ წაიშლება, შემდეგ კი შეიქმნება. შექმნის შემდეგ, მას Personali ცხრილიდან ჩაემატება მონაცემები თბილისში მცხოვრები პერსონალის შესახებ. ბოლოს, ჩამატებული სტრიქონები ეკრანზე გამოგვაქვს.

ვ. Personali ცხრილს დავუმატოთ სტრიქონი, რომლის ყველა სვეტში ნაგულისხმევი მნიშვნელობა მოთავსდება. მოთხოვნას აქვს სახე:

```
USE Shekveta;
INSERT INTO Personali DEFAULT VALUES;
SELECT * FROM Personali;
```

შედეგი:

personaliID	gvari	sakheli	ganyofileba	qalaqi	raioni	xelfasi	asaki	staji
18	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

ზ. Personali ცხრილს დავუმატოთ სტრიქონი, რომლის ზოგიერთ სვეტში ნაგულისხმევი მნიშვნელობა მოთავსდება. მოთხოვნას აქვს სახე:

```
USE Shekveta;
INSERT INTO Personali
VALUES (N'სამხარაძე', N'ბექა', DEFAULT, N'თბილისი', DEFAULT, DEFAULT, DEFAULT, 27, 5,
'07.09.1980', N'კაცი', N'პეტრიაშვილის ქ.1', '221-11-00', '899-012345', DEFAULT, DEFAULT);
SELECT * FROM Personali;
```

შედეგი:

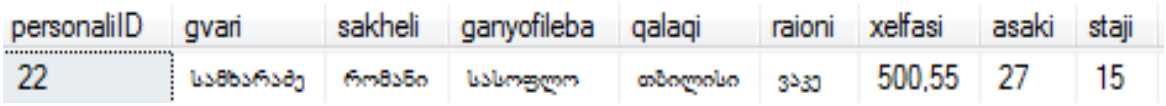
	personaliID	gvari	sakheli	ganyofileba	qalaqi	regioni	raioni	xelfasi	asaki	staji
1	14	სამხარაძე	ბექა	NULL	თბილისი	NULL	NULL	NULL	27	5

თ. Personali ცხრილს დავუმატოთ სტრიქონი, რომლის xelfasi და staji სვეტების მნიშვნელობები ფორმულის მიხედვით გამოითვლება. მოთხოვნას აქვს სახე:

```
USE Shekveta;
INSERT INTO Personali
VALUES (N'სამხარაძე', N'რომანი', N'სასოფლო', N'თბილისი', NULL, N'ვაკე', 100.11 * 5, 27, 5 * 3,
'07.09.1980', N'კაცი', N'პეტრიაშვილის ქ.1', '221-11-00', '899-012345', N'beqa@geo.net.ge', 9);
```

SELECT * FROM Personali;

შედეგი:



personaliID	gvari	sakheli	ganyofileba	qalaqi	raioni	xelfasi	asaki	staji
22	სამხარაძე	რომანი	სასოფლო	თბილისი	ვაკე	500,55	27	15

ი. ერთი INSERT ბრძანების გამოყენებით შეგვიძლია ბევრი სტრიქონის ჩასმა. ჩასამატებელი სტრიქონები ერთმანეთისაგან მძიმეებით უნდა გამოიყოს. მოთხოვნას აქვს სახე:

```
USE Shekveta;
INSERT INTO Personali
VALUES
(N'სამხარაძე', N'საბა', N'სასოფლო', N'თბილისი', NULL, N'ვაკე', 110.11 * 5, 27, 5 * 3, '07.09.1985', N'კაცი', N'პეტრიაშვილის ქ.1', '221-11-00', '599-012345', N'saba@geo.net.ge', 39),
(N'სამხარაძე', N'ანა', N'სავაჭრო', N'ქუთაისი', N'იმერეთი', NULL, 120.11 * 6, 28, 6 * 3, '07.09.1990', N'ქალი', N'პეტრიაშვილის ქ.1', '221-11-00', '574-012345', N'ana@geo.net.ge', 19),
(N'გაჩეჩილაძე', N'ლია', N'სამედიცინო', N'ლაგოდეხი', N'კახეთი', NULL, 130.11 * 7, 29, 7 * 3, '07.09.1995', N'ქალი', N'პეტრიაშვილის ქ.1', '221-11-00', '555-012345', N'lia@geo.net.ge', 29);
SELECT * FROM Personali;
```

SELECT ... INTO ბრძანება

ამ ბრძანების შესრულების შედეგად შეიქმნება ახალი ცხრილი, რომელშიც SELECT მოთხოვნის მიერ ამორჩეული სტრიქონები მოთავსდება. მისი სინტაქსია:

SELECT { *სვეტის_სახელი* [[AS] *სვეტის_ფსევდონიმი*], ... n }

INTO *ახალი_ცხრილის_სახელი* FROM { *საწყისი_ცხრილის_სახელი*, ... n }

განვიხილოთ არგუმენტების დანიშნულება.

– *სვეტის_სახელი* [[AS] *სვეტის_ფსევდონიმი*] განსაზღვრავს იმ სვეტის სახელს, რომელიც ჩართული იქნება შედეგში. თუ სხვადასხვა ცხრილის სვეტებს ერთნაირი სახელები აქვს, მაშინ მათი სახელების წინ უნდა მივუთითოთ *ფსევდონიმი* (ცხრილის სახელი). ფსევდონიმის გამოყენება სასარგებლოა იმ შემთხვევაშიც, როცა გვინდა, რომ შესაქმნელი ცხრილის სვეტებს საწყისი ცხრილისაგან განსხვავებული სახელები ჰქონდეს;

– INTO *ახალი_ცხრილს_სახელი* მიუთითებს შესაქმნელი ცხრილის სახელს. თუ დროებით ცხრილს ვქმნით, მაშინ ცხრილის სახელის წინ უნდა მივუთითოთ # ან ##;

– FROM { *საწყისი_ცხრილის_სახელი*, ... n } არგუმენტი შეიცავს საწყისი ცხრილების სახელებს.

მაგალითები.

ა. შევქმნათ Pers_1 ცხრილი და მასში Personali ცხრილიდან ჩავწეროთ gvari, ganyofileba და staji სვეტების მნიშვნელობები. მოთხოვნას აქვს სახე:

```
USE Shekveta;
SELECT gvari AS [გვარი], ganyofileba AS [განყოფილება], staji AS [სტაჟი]
INTO Pers_1
FROM Personali;
SELECT * FROM Pers_1;
```

შედეგი:

	გვარი	განყოფილება	სტაჟი
1	სამხარაძე	სამედიცინო	7
2	სამხარაძე	სამედიცინო	8
3	გაჩეჩილაძე	სავაჭრო	22
4	გაჩეჩილაძე	სასოფლო	20
5	შენგელია	სავაჭრო	11
6	კაპანაძე	სასპორტო	17
7	ქუხიშვილი	სამედიცინო	28
8	ხოშტარია	სასპორტო	30
9	ყალაბეგიშვილი	სასოფლო	15
10	გიორგაძე	სამედიცინო	22

ბ. შევქმნათ Pers_2 ცხრილი და მასში Personali ცხრილიდან ჩავწეროთ თბილისელი თანამშრომლების გვარები. მოთხოვნას აქვს სახე:

```
SELECT gvari AS [გვარი], saxeli AS [სახელი], qalaqi AS [ქალაქი]
INTO Pers_2
FROM Personali
WHERE qalaqi = N'თბილისი';
SELECT * FROM Pers_2;
```

შედეგი:

	გვარი	სახელი	ქალაქი
1	სამხარაძე	საბა	თბილისი
2	გაჩეჩილაძე	ლია	თბილისი
3	გაჩეჩილაძე	ხათუნა	თბილისი
4	ქუხიშვილი	ალექსანდრე	თბილისი
5	სამხარაძე	ბექა	თბილისი

ცხრილის მონაცემების ცვლილება

UPDATE ბრძანება

ცხრილებში მონაცემების შესაცვლელად UPDATE ბრძანება გამოიყენება. მისი სინტაქსია:

```
UPDATE { ცხრილის_სახელი | წარმოდგენის_სახელი }
SET { სვეტის_სახელი = { გამოსახულება | DEFAULT | NULL }
| @ცვლადის_სახელი = გამოსახულება
| @ცვლადის_სახელი = სვეტის_სახელი = გამოსახულება } [,...n]
{  [ FROM { <წყარო_ცხრილი> } [,...] ] [ WHERE <ძებნის_პირობა> ]  }
```

განვიხილოთ არგუმენტების დანიშნულება.

- *ცხრილის_სახელი* იმ ცხრილის სახელია, რომელშიც უნდა შესრულდეს მონაცემების ცვლილება.
- *წარმოდგენის_სახელი* იმ წარმოდგენის სახელია, რომელშიც უნდა შესრულდეს მონაცემების ცვლილება. UPDATE ბრძანება მონაცემებს წარმოდგენის მხოლოდ ერთ ცხრილში ცვლის. თუ წარმოდგენა ორი ცხრილისაგან შედგება, მაშინ UPDATE ბრძანება ორჯერ უნდა

შევასრულოთ თითოეულ ცხრილში მონაცემების შესაცვლელად. გარდა ამისა, ნებადართული უნდა იყოს მონაცემების ცვლილება წარმოდგენის საშუალებით.

- SET საკვანძო სიტყვა იწყებს ბლოკს, რომელშიც მითითებულია შესაცვლელი სვეტების ან ცვლადების სია.
- *სვეტის_სახელი* = { *გამოსახულება* | DEFAULT | NULL }. თითოეული სვეტისათვის უნდა მიეთითოს მისთვის მისანიჭებელი მნიშვნელობა. DEFAULT საკვანძო სიტყვა მიუთითებს, რომ სვეტს უნდა მიენიჭოს ნაგულისხმევი (წინასწარ განსაზღვრული) მნიშვნელობა. NULL საკვანძო სიტყვა მიუთითებს, რომ სვეტს უნდა მიენიჭოს NULL მნიშვნელობა. *გამოსახულება* შეიძლება იყოს მუდმივა, ცვლადი ან გამოსახულება. ცვლილება შეეხება იმ სტრიქონებს, რომლებიც აკმაყოფილებენ WHERE განყოფილებაში მითითებულ პირობას. თუ *გამოსახულება* არგუმენტში გამოიყენება ის სვეტები, რომლებსაც ცვლილება შეეხო, მაშინ გამოყენებული იქნება მათი ძველი მნიშვნელობები. ეს იმიტომ ხდება, რომ მხოლოდ ბრძანების შესრულების შემდეგ შეიცვლება მითითებული სვეტების მნიშვნელობები.
- @ცვლადის_სახელი = *გამოსახულება* არგუმენტი ცვლადს ანიჭებს გამოსახულების მნიშვნელობას.
- @ცვლადის_სახელი = სვეტის_სახელი = გამოსახულება კონსტრუქცია საშუალებას გვაძლევს შევათავსოთ ცვლადებისა და სვეტების სახელების გამოყენება.

ცვლილება ყოველთვის შეეხება ამორჩეულ სტრიქონებს. თუ ფილტრი არ არის მითითებული, მაშინ ცვლილება შეეხება ცხრილის ყველა სტრიქონის მითითებულ სვეტებს.

მაგალითები.

ა. დავთვალოთ იმ ხელშეკრულებების რაოდენობა, რომლებიც არ შესრულდა შემკვეთის მიზეზით. მიღებული მნიშვნელობა მივანიჭოთ @raodenoba ცვლადს. მოთხოვნას აქვს სახე:

```
USE Shekveta;
DECLARE @raodenoba INT;
SET @raodenoba = 0;
UPDATE Xelshekruleba SET @raodenoba = @raodenoba + 1
WHERE visi_mizezit = N'შემკვეთის';
SELECT @raodenoba AS [რაოდენობა];
```

შედეგი:

	რაოდენობა
1	3

ბ. თითოეული თანამშრომლის ხელფასი გავზარდოთ 10%-ით. მოთხოვნას აქვს სახე:

```
UPDATE Personali SET xelfasi = xelfasi * 1.1;
SELECT * FROM Personali;
```

შედეგი:

	personaliID	gvari	sakheli	ganyofileba	qalaqi	raioni	xelfasi	asaki
1	1	სამხარაძე	საბა	საშედიცინო	თბილისი	დიდუბე	979,894...	21
2	2	სამხარაძე	ანა	საშედიცინო	ბათუმი	NULL	7110,63...	18
3	3	გაჩეჩილაძე	ლია	სავაჭრო	თბილისი	საბურთალო	2632,97...	46
4	4	გაჩეჩილაძე	ხათუნა	სასოფლო	თბილისი	ვერა	3310,63...	23
5	5	შენგელია	ნატა	სავაჭრო	ზუგდიდი	NULL	2891,06...	28
6	6	კაპანაძე	ეთერი	სასპორტო	რუსთავი	NULL	2354,52...	50
7	7	ქუხიშვილი	ალექსანდრე	საშედიცინო	თბილისი	ვერა	7213,74...	53
8	8	ხოშტარია	ცისანა	სასპორტო	ზუგდიდი	NULL	1882,61...	58
9	9	ყალაბეგიშვილი	გია	სასოფლო	ქუთაისი	NULL	2370,83...	42
10	10	გიორგაძე	ზვიადი	საშედიცინო	ქობულეთი	NULL	699,740...	20
11	11	კიკნაძე	მზია	სავაჭრო	ბათუმი	NULL	1078,96...	35

გ. გამოვთვალოთ ვალის მნიშვნელობები ლარებსა და დოლარებში. მოთხოვნას აქვს სახე:
UPDATE Xelshekruleba SET vali_l = gadasaxdeli_l - gadaxdili_l, vali_d = gadasaxdeli_d - gadaxdili_d;
SELECT * FROM Xelshekruleba;

შედეგი:

	xelshekrulebaID	personaliID	shemkvetiID	gadasaxdeli_l	gadasaxdeli_d	gadaxdili_l	gadaxdili_d	vali_l	vali_d
1	1	1	13	5000	2631,58	5000	2631,58	0	0
2	2	1	13	3700	1989,25	3000	1612,9	700	376,35
3	3	2	6	8000	4301,08	4300	2311,83	3700	1989,25
4	5	3	2	1900	1027,03	1900	1027,03	0	0
5	6	3	2	2500	1358,7	2500	1358,7	0	0
6	7	4	9	8200	4480,87	8000	4371,58	200	109,29
7	8	5	11	7700	4277,78	7700	4277,78	0	0
8	9	6	10	8000	4444,44	1600	888,89	6400	3555,55
9	10	7	3	6000	3333,33	5500	3055,56	500	277,77
10	11	1	15	9800	5444,44	9000	5000	800	444,44
11	12	8	4	7100	4176,47	7100	4176,47	0	0
12	13	8	16	3000	1764,71	3000	1764,71	0	0
13	14	9	5	10000	5882,35	10000	5882,35	0	0
14	15	7	13	2000	1176,47	1700	1000	300	176,47
15	16	4	3	4400	2603,55	4400	2603,55	0	0
16	17	10	7	3900	2321,43	3900	2321,43	0	0

დ. მოყვანილ მაგალითში ცვლადს მიენიჭება #Shemkveti ცხრილის უკანასკნელი შეცვლილი სტრიქონის gvari სვეტის მნიშვნელობა:
SELECT * INTO #Shemkveti FROM Shemkveti;
DECLARE @var1 NVARCHAR(30);
UPDATE #Shemkveti SET @var1 = gvari = gvari + N' - გვარი'
WHERE iuridiuli_fizikuri = N'იურიდიული';
SELECT @var1 AS [გვარი];
SELECT * FROM #Shemkveti;

შედეგი:

	გვარი
1	ლომიძე - გვარი

	shemkvetiID	personaliID	iuridiuli_fizikuri	gvari	sakheli	qalaqi
1	1	3	ფიზიკური	ნემსაძე	დოდო	თბილისი
2	2	7	იურიდიული	სეხნიაშვილი - გვარი	გია	თელავი
3	3	8	იურიდიული	მამაიაშვილი - გვარი	ჯემალი	თელავი
4	4	9	ფიზიკური	შენგელია	მამუკა	ქუთაისი
5	5	2	იურიდიული	ნონიაშვილი - გვარი	ზურა	ბათუმი
6	6	3	ფიზიკური	სამხარაძე	არჩილი	ბათუმი
7	7	4	ფიზიკური	მიქელაძე	მერაბი	ქუთაისი
8	8	6	იურიდიული	გიორგაძე - გვარი	გივი	ზუგდიდი
9	9	2	იურიდიული	ჭიკაძე - გვარი	გელა	გორი
10	10	1	იურიდიული	ლომიძე - გვარი	რუსუ...	თბილისი
11	11	1	ფიზიკური	ხოშტარია	სიმონი	თბილისი
12	12	8	ფიზიკური	ლასურაშვილი	ია	თბილისი
13	16	NULL	ფიზიკური	აჭარაძე	გურამი	თელავი

ცხრილიდან მონაცემების წაშლა

DELETE ბრძანება

ცხრილიდან სტრიქონების წასაშლელად DELETE ბრძანება გამოიყენება. მისი სინტაქსია:

DELETE [FROM] { *ცხრილის_სახელი* | *წარმოდგენის_სახელი* }
[FROM { <*ცხრილის_სახელი*> } [,…]]
[WHERE <*ძებნის_პირობა*>]

თუ WHERE განყოფილება არ არის მითითებული, მაშინ ცხრილიდან ყველა სტრიქონი წაიშლება.

მაგალითი. #Shemkveti ცხრილიდან წავშალოთ იურიდიული პირები. ეს ცხრილი შეგვიძლია SELECT * INTO #Shemkveti FROM Shemkveti ბრძანების გამოყენებით შევქმნათ. მოთხოვნას აქვს სახე:

```
USE Shekveta;
SELECT * INTO #Shemkveti
FROM Shemkveti;
DELETE FROM #Shemkveti
WHERE iuridiuli_fizikuri = N'იურიდიული';
SELECT * FROM #Shemkveti;
```

შედეგი:

	shemkvetiID	personaliID	iuridiuli_fizikuri	gvari	sakheli	qalaqi
1	1	3	ფიზიკური	ნემსაძე	დოდო	თბილისი
2	4	9	ფიზიკური	შენგელია	მამუკა	ქუთაისი
3	6	3	ფიზიკური	სამხარაძე	არჩილი	ბათუმი
4	7	4	ფიზიკური	მიქელაძე	მერაბი	ქუთაისი
5	11	1	ფიზიკური	ხოშტარია	სიმონი	თბილისი
6	12	8	ფიზიკური	ლასურაშვილი	ია	თბილისი
7	16	NULL	ფიზიკური	აჭარაძე	გურამი	თელავი

ცხრილიდან მონაცემების ამორჩევა

SELECT ბრძანება

ცხრილებიდან მონაცემების ამოსარჩევად SELECT ბრძანება გამოიყენება. მისი სინტაქსია:

SELECT *სვეტების სია* [INTO *ახალი_ცხრილის_სახელი*]
FROM *საწყისი_ცხრილის_სახელი*
[WHERE *ძებნის_პირობა*]
[GROUP BY *დაჯგუფების_გამოსახულება*]
[HAVING *ძებნის_პირობა*]
[ORDER BY *დახარისხების_გამოსახულება* [ASC | DESC]]

განვიხილოთ SELECT ბრძანების განყოფილებები.

SELECT განყოფილება.

მასში ეთითება გამოსატანი სვეტების სია. მისი სინტაქსია:

SELECT [ALL | DISTINCT] [TOP n [PERCENT]] <*სია*>

განვიხილოთ არგუმენტების დანიშნულება.

- ALL არგუმენტი მიუთითებს, რომ მოთხოვნის შესრულების შედეგში დასაშვებია გამეორებადი (ერთნაირი) სტრიქონების გამოტანა. ეს არგუმენტი ავტომატურად იგულისხმება იმ შემთხვევაში, თუ ის მითითებული არ არის.
- DISTINCT არგუმენტი კრძალავს ერთნაირი სტრიქონების გამოტანას.
- TOP n [PERCENT] კონსტრუქციის გამოყენების შედეგად გაიცემა პირველი n სტრიქონი. თუ მითითებულია PERCENT, მაშინ n აღიქმება როგორც პროცენტი.
- <*სია*> კონსტრუქცია შეიცავს გამოსატანი სვეტების სახელებს, აგრეთვე, სხვადასხვა ობიექტებს. მას შემდეგი სტრუქტურა აქვს:

<*სია*> ::=
{ * | { *ცხრილის_სახელი* | *წარმოდგენის_სახელი* | *ცხრილის_ფსევდონიმი* } .*
| { *სვეტის_სახელი* | *გამოსახულება* | IDENTITYCOL }
[[AS] *სვეტის_ფსევდონიმი*] | *სვეტის_ფსევდონიმი* = *გამოსახულება* } [,...n]

განვიხილოთ არგუმენტების დანიშნულება.

- * მიუთითებს, რომ გამოტანილი უნდა იყოს ცხრილის ან წარმოდგენის ყველა სვეტის მნიშვნელობა.
- *ცხრილის_სახელი* იმ შემთხვევაში ეთითება, როცა მოთხოვნაში რამდენიმე ცხრილია მითითებული.
- *წარმოდგენის_სახელი* იმ შემთხვევაში ეთითება, როცა მოთხოვნაში რამდენიმე წარმოდგენაა მითითებული.

- *ცხრილის_ფსევდონიმი* შეგვიძლია გამოვიყენოთ ცხრილის სახელის ნაცვლად.
- *სვეტის_სახელი* [[AS] *სვეტის_ფსევდონიმი*] არგუმენტი განსაზღვრავს იმ სვეტის სახელს, რომლის მნიშვნელობაც უნდა გამოვიტანოთ. სვეტი უნდა ეკუთვნოდეს FROM განყოფილებაში მითითებული ცხრილებიდან ერთ-ერთს. გამოტანისას სვეტების სახელები შეგვიძლია *სვეტის_ფსევდონიმით* შევცვალოთ. ფსევდონიმების გამოყენება აუცილებელია იმ შემთხვევაში, როცა გამოსატანი ცხრილების სვეტებს ერთნაირი სახელი აქვთ. თუ ფსევდონიმი შეიცავს დაუშვებელ სიმბოლოებს, ინტერვალებს ან ეროვნულ სიმბოლოებს, მაშინ ის კვადრატულ ფრჩხილებში უნდა მოვათავსოთ.
- გამოსახულება [[AS] *სვეტის_ფსევდონიმი*] განსაზღვრავს გამოსახულებას, რომელიც გამოითვლება და მიღებული მნიშვნელობა მოთავსდება შედეგში. გამოსახულება შეიძლება შეიცავდეს მუდმივებს, ცვლადებს, სვეტების სახელებს, ფუნქციებს ან მათ კომბინაციას.
- IDENTITYCOL [[AS] *სვეტის_ფსევდონიმი*] არგუმენტის გამოიყენება იწვევს იმ სვეტის გამოტანას, რომელსაც IDENTITY თვისება დაყენებული აქვს YES მდგომარეობაში. თუ მოთხოვნაში რამდენიმე ცხრილია მითითებული, რომლებსაც ასეთი სვეტები აქვს, მაშინ აშკარად უნდა მივუთითოთ იმ ცხრილის სახელი, რომელსაც ეს სვეტი ეკუთვნის.

მაგალითები.

ა. Xelshekruleba ცხრილიდან გამოვიტანოთ personaliID სვეტის მნიშვნელობები. მოთხოვნას აქვს სახე:

```
Use Shekveta;
SELECT ALL personaliID
FROM Xelshekruleba;
```

შედეგი:

	personaliID
1	5
2	6
3	7
4	9
5	8
6	2
7	2
8	2
9	3
10	3
11	3
12	8
13	3
14	1
15	1
16	1
17	3
18	9
19	3
20	4
21	10

ბ. წინა მოთხოვნაში ALL სიტყვა შევცვალოთ DISTINCT სიტყვით. მოთხოვნას აქვს სახე:

```
SELECT DISTINCT personaliID
FROM Xelshekruleba;
```

შედეგი:

	personaliID
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10

გ. Personali ცხრილიდან გამოვიტანოთ ინფორმაცია პირველი 5 თანამშრომლის შესახებ. მოთხოვნას აქვს სახე:

```
SELECT TOP 5 *
FROM Personali;
```

შედეგი:

	personaliID	gvari	saxeli	ganyofileba	qalaqi	regioni	raioni	xelfasi	asaki	staji
1	1	სამხარაძე	საბა	სასპორტო	ზუგდიდი	სამეგრელო	NULL	1651,086382	21	2
2	2	სამხარაძე	ანა	სამედიცინო	ბათუმი	აჭარა	NULL	1430,841951	26	7
3	3	გაჩეჩილაძე	ლია	სამედიცინო	თბილისი	NULL	საბურთალო	1045,475332	54	10
4	4	გაჩეჩილაძე	ხათუნა	სასოფლო	თბილისი	NULL	ვერა	979,718663	46	18
5	5	შენგელია	ნატა	სავაჭრო	ზუგდიდი	სამეგრელო	NULL	935,845284	27	5

დ. Personali ცხრილიდან გამოვიტანოთ ინფორმაცია ფირმის თანამშრომლების 40%-ის შესახებ. მოთხოვნას აქვს სახე:

```
SELECT TOP 40 PERCENT *
FROM Personali;
```

შედეგი:

	personaliID	gvari	saxeli	ganyofileba	qalaqi	regioni	raioni	xelfasi	asaki	staji
1	1	სამხარაძე	საბა	სასპორტო	ზუგდიდი	სამეგრელო	NULL	1651,086382	21	2
2	2	სამხარაძე	ანა	სამედიცინო	ბათუმი	აჭარა	NULL	1430,841951	26	7
3	3	გაჩეჩილაძე	ლია	სამედიცინო	თბილისი	NULL	საბურთალო	1045,475332	54	10
4	4	გაჩეჩილაძე	ხათუნა	სასოფლო	თბილისი	NULL	ვერა	979,718663	46	18
5	5	შენგელია	ნატა	სავაჭრო	ზუგდიდი	სამეგრელო	NULL	935,845284	27	5

ბრძანების შესრულების შედეგად ამოირჩევა პირველი 5 სტრიქონი, Personali ცხრილი 11 სტრიქონს შეიცავს, ხოლო 11-ის 40% კი არის 5. მოთხოვნას აქვს სახე:

```
SELECT *
FROM Personali;
```

შედეგი:

	personaliID	gvari	saxeli	ganyofileba	qalaqi	regioni	raioni	xelfasi	asaki	staji
1	1	სამხარაძე	საბა	სასპორტო	ზუგდიდი	სამეგრელო	NULL	1651,086382	21	2
2	2	სამხარაძე	ანა	სამედიცინო	ბათუმი	აჭარა	NULL	1430,841951	26	7
3	3	გაჩეჩილაძე	ლია	სამედიცინო	თბილისი	NULL	საბურთალო	1045,475332	54	10
4	4	გაჩეჩილაძე	ხათუნა	სასოფლო	თბილისი	NULL	ვერა	979,718663	46	18
5	5	შენგელია	ნატა	სავაჭრო	ზუგდიდი	სამეგრელო	NULL	935,845284	27	5
6	6	კაპანაძე	ეთერი	სასპორტო	რუსთავი	ქართლი	NULL	1156,081531	49	9
7	7	ქუხიშვილი	ალექსანდრე	სამედიცინო	თბილისი	NULL	ვერა	1254,353958	57	28
8	8	ყალაბეგიშვილი	გია	სასოფლო	ქუთაისი	იმერეთი	NULL	880,135795	61	25
9	9	გიორგაძე	გივი	სამედიცინო	ქობულეთი	აჭარა	NULL	869,746878	63	26
10	10	კიკნაძე	მზია	სავაჭრო	ბათუმი	აჭარა	NULL	847,603581	48	17
11	11	სამხარაძე	გიორგი	სასოფლო	თბილისი	NULL	ვაკე	1078,967252	28	6

ე. Personali ცხრილიდან გამოვიტანოთ თანამშრომლების გვარები, დაბადების თარიღები და იმ ქალაქების დასახელება, რომელშიც ისინი ცხოვრობენ. მოთხოვნას აქვს სახე:

```
SELECT gvari AS [თანამშრომლის გვარი],
       saxeli AS [თანამშრომლის სახელი],
       qalaqi, asaki AS [თანამშრომლის ასაკი],
       tarigi_dabadebis AS [დაბადების თარიღი]
FROM Personali;
```

შედეგი:

	თანამშრომლის გვარი	თანამშრომლის სახელი	qalaqi	თანამშრომლის ასაკი	დაბადების თარიღი
1	სამხარაძე	საბა	ზუგდიდი	21	1994-03-19 00:00:00.000
2	სამხარაძე	ანა	ბათუმი	26	1989-10-05 00:00:00.000
3	გაჩეჩილაძე	ლია	თბილისი	54	1961-04-12 00:00:00.000
4	გაჩეჩილაძე	ხათუნა	თბილისი	46	1969-02-20 00:00:00.000
5	შენგელია	ნატა	ზუგდიდი	27	1988-09-16 00:00:00.000
6	კაპანაძე	ეთერი	რუსთავი	49	1966-11-30 00:00:00.000
7	ქუხიშვილი	ალექსანდრე	თბილისი	57	1958-12-03 00:00:00.000
8	ყალაბეგიშვილი	გია	ქუთაისი	61	1954-04-04 00:00:00.000
9	გიორგაძე	გივი	ქობულეთი	63	2052-07-05 19:54:08.170
10	კიკნაძე	მზია	ბათუმი	48	1967-12-09 00:00:00.000
11	სამხარაძე	გიორგი	თბილისი	28	1987-09-07 00:00:00.000

ზ. Personali და Shemkveti ცხრილებიდან გამოვიტანოთ თანამშრომლების გვარები, სახელები და მათთან გაფორმებული ხელშეკრულებების ნომრები. მოთხოვნას აქვს სახე:

```
USE Shekveta;
SELECT  Personali.gvari  AS [შემკვეთის გვარი],
        Personali.saxeli AS [შემკვეთის სახელი],
        Xelshekruleba.xelshekrulebaID AS [ხელშეკრულების ნომერი]
FROM    Personali, Xelshekruleba
WHERE  Personali.personaliID = Xelshekruleba.personaliID;
```

შედეგი:

	შემკვეთის გვარი	შემკვეთის სახელი	ხელშეკრულების ნომერი
1	შენგელია	ნატა	17
2	კაპანაძე	ეთერი	18
3	ქუხიშვილი	ალექსანდრე	19
4	გიორგაძე	გივი	20
5	ყალაბეგიშვილი	გია	21
6	სამხარაძე	ანა	1
7	სამხარაძე	ანა	2
8	სამხარაძე	ანა	3
9	გაჩეჩილაძე	ლია	4
10	გაჩეჩილაძე	ლია	5
11	გაჩეჩილაძე	ლია	6
12	ყალაბეგიშვილი	გია	7
13	გაჩეჩილაძე	ლია	8
14	სამხარაძე	საბა	9
15	სამხარაძე	საბა	10
16	სამხარაძე	საბა	11
17	გაჩეჩილაძე	ლია	12
18	გიორგაძე	გივი	13
19	გაჩეჩილაძე	ლია	14
20	გაჩეჩილაძე	ხათუნა	15
21	კიკნაძე	მზია	16

თ. გვაინტერესებს ფირმის თანამშრომლების დაბადების წლები. მოთხოვნას აქვს სახე:

```
SELECT gvari, saxeli, YEAR(tarigi_dabadebis) AS [დაბადების წელი]
FROM Personali;
```

შედეგი:

	gvari	saxeli	დაბადების წელი
1	სამხარაძე	საბა	1994
2	სამხარაძე	ანა	1989
3	გაჩეჩილაძე	ლია	1961
4	გაჩეჩილაძე	ხათუნა	1969
5	შენგელია	ნატა	1988
6	კაპანაძე	ეთერი	1966
7	ქუხიშვილი	ალექსანდრე	1958
8	ყალაბეგიშვილი	გია	1954
9	გიორგაძე	გივი	2052
10	კიკნაძე	მზია	1967
11	სამხარაძე	გიორგი	1987

ი. იგივე მოთხოვნა შეიძლება ასეც ჩავწეროთ:

```
SELECT gvari, saxeli, [დაბადების წელი] = YEAR(tarigi_dabadebis)
FROM Personali;
```

შედეგი:

	gvari	saxeli	დაბადების წელი
1	სამხარაძე	საბა	1994
2	სამხარაძე	ანა	1989
3	გაჩეჩილაძე	ლია	1961
4	გაჩეჩილაძე	ხათუნა	1969
5	შენგელია	ნატა	1988
6	კაპანაძე	ეთერი	1966
7	ქუხიშვილი	ალექსანდრე	1958
8	ყალაბეგიშვილი	გია	1954
9	გიორგაძე	გივი	2052
10	კოკნაძე	მზია	1967
11	სამხარაძე	გიორგი	1987

კ. Personali ცხრილიდან გამოვიტანოთ იმ სვეტის მნიშვნელობები, რომლისთვისაც IDENTITY თვისებას YES მნიშვნელობა აქვს. გამოვიტანოთ, აგრეთვე თანამშრომლების გვარები და იმ ქალაქების დასახელება, რომელშიც ისინი ცხოვრობენ. მოთხოვნას აქვს სახე:

```
SELECT TOP 50 PERCENT IDENTITYCOL AS [იდენტიფიკატორი], gvari, saxeli, qalaqi
FROM Personali;
```

შედეგი:

	იდენტიფიკატორი	gvari	saxeli	qalaqi
1	1	სამხარაძე	საბა	ზუგდიდი
2	2	სამხარაძე	ანა	ბათუმი
3	3	გაჩეჩილაძე	ლია	თბილისი
4	4	გაჩეჩილაძე	ხათუნა	თბილისი
5	5	შენგელია	ნატა	ზუგდიდი
6	6	კაპანაძე	ეთერი	რუსთავი

განყოფილებების დამუშავების მიმდევრობა

SELECT ბრძანების დამუშავება არ ხდება იმ მიმდევრობით, რა მიმდევრობითაც ისინი არის ჩაწერილი. განვიხილოთ პროგრამული კოდი:

```
USE Shekveta;
SELECT personaliID, YEAR(tarigi_dawyebis) AS xelshyear, COUNT(*) AS numxelsh
FROM Xelshekruleba
WHERE shemkvetiID = 6
GROUP BY personaliID, YEAR(tarigi_dawyebis)
HAVING COUNT(*) > 1
ORDER BY personaliID, xelshyear;
```

ეს პროგრამული კოდი იწყება USE ბრძანებით, რომელიც გვაკავშირებს Shekveta მონაცემთა ბაზასთან. მიუხედავად იმისა, რომ მოთხოვნაში SELECT განყოფილება (ელემენტი) პირველია, ლოგიკურად ის დამუშავდება თითქმის ბოლოს. SELECT ბრძანების განყოფილები, ლოგიკურად შემდეგი მიმდევრობით დამუშავდება:

1. FROM
2. WHERE
3. GROUP BY
4. HAVING

5. SELECT
6. ORDER BY.

ამრიგად, მიუხედავად იმისა, რომ სინტაქსურად მოთხოვნა იწყება SELECT განყოფილებით, SELECT ბრძანების ელემენტები მუშავდება შემდეგი მიმდევრობით:

1. FROM Xelshekruleba
2. WHERE shemkvetiID = 6
3. GROUP BY personaliID, YEAR(tarigi_dawyebis)
4. HAVING COUNT(*) > 1
5. SELECT personaliID, YEAR(tarigi_dawyebis) AS xelshyear, COUNT(*) AS numxelsh
6. ORDER BY personaliID, xelshyear;

ყველა ბრძანება შეიძლება არ დამთავრდეს წერტილ-მძიმით. მაგრამ, ზოგ შემთხვევაში, მისი გამოყენების გარეშე, პროგრამული კოდის აზრი შეიძლება არაცალსახა იყოს. ამიტომ, სასურველია რომ, ყველა ბრძანება წერტილ-მძიმით დამთავრდეს, რადგან ეს სტანდარტული სტილია და აადვილებს პროგრამული კოდის კითხვას.

FROM განყოფილება

FROM საკვანძო სიტყვის შემდეგ ეთითება იმ ცხრილების ან წარმოდგენების სახელები, საიდანაც ხდება სვეტების ამორჩევა. მისი სინტაქსია:

FROM { *ცხრილის_სახელი* [[AS] *ცხრილის_ფსევდონიმი*]
| *წარმოდგენის_სახელი* [[AS] *წარმოდგენის_ფსევდონიმი*] }

მაგალითი. მივიღოთ ინფორმაცია თანამშრომლების შესახებ. მოთხოვნას აქვს სახე:

SELECT * FROM Personali;

FROM განყოფილებას დაწვრილებით მე-5 თავში განვიხილავთ.

WHERE განყოფილება

WHERE განყოფილება გამოიყენება ცხრილის სტრიქონების გაფილტვრისათვის (ჰორიზონტალური ფილტრი). ის გვიბრუნებს მხოლოდ იმ სტრიქონებს, რომლებისთვისაც გამოსათვლელი ლოგიკური გამოსახულების მნიშვნელობა TRUE-ის ტოლია. უნდა გვახსოვდეს, რომ T-SQL ენაში გამოიყენება სამობითი პრედიკატების ლოგიკა, რომელშიც ლოგიკურმა გამოსახულებებმა შეიძლება მიიღოს TRUE, FALSE ან UNKNOWN მნიშვნელობა. სამობით ლოგიკაში მტკიცება „აბრუნებს TRUE-ს“ - არ არის იგივე, რაც მტკიცება „არ აბრუნებს FALSE-ს“. WHERE სტადიაზე ამოირჩევა სტრიქონები, რომლისთვისაც გამოსათვლელი ლოგიკური მნიშვნელობა TRUE-ის ტოლია, და არ ამოირჩევა სტრიქონები, რომლისთვისაც ეს გამოსახულება FALSE-ის ან UNKNOWN-ის ტოლია. WHERE ელემენტის სინტაქსია:

WHERE <*ძებნის_პირობა*>

- <*ძებნის_პირობა*> არის ნებისმიერი ლოგიკური გამოსახულება, რომელიც გასცემს TRUE ან FALSE მნიშვნელობას. თუ ლოგიკური გამოსახულება რამდენიმეა, მაშინ ისინი უნდა გავაერთიანოთ AND (ლოგიკური და), OR (ლოგიკური ან) და NOT (ლოგიკური არა) ოპერატორების გამოყენებით.

ორ ცხრილს შორის კავშირის (რელაციის) დასამყარებლად WHERE პირობის ნაცვლად უმჯობესია FROM განყოფილებაში { INNER | { { LEFT | RIGHT | FULL } [OUTER] } JOIN } საკვანძო სიტყვების გამოყენება, რადგან მოთხოვნა უფრო სწრაფად შესრულდება (იხილე მე-5 თავი).

მაგალითი.

ა. გვაინტერესებს ინფორმაცია 1990 წელს დაბადებული თანამშრომლების შესახებ. მოთხოვნას აქვს სახე:

```
USE Shekveta;
SET DATEFORMAT DMY;
SELECT *
FROM Personali
WHERE YEAR(tarigi_dabadebis) = 1966;
```

შედეგი:

	personaliID	gvari	saxeli	ganyofileba	qalaqi	regioni	raioni	xelfasi	asaki	staji
1	6	კაპანაძე	ეთერი	სასპორტო	რუსთავი	ქართლი	NULL	1156,081531	49	9

ბ. გვანტერესებს ინფორმაცია 10 წელზე მეტი სტაჟის მქონე თანამშრომლების შესახებ. მოთხოვნას აქვს სახე:

```
USE Shekveta;
SELECT *
FROM Personali
WHERE staji > 10;
```

შედეგი:

	personaliID	gvari	saxeli	ganyofileba	qalaqi	regioni	raioni	xelfasi	asaki	staji
1	4	გაჩეჩილაძე	ხათუნა	სასოფლო	თბილისი	NULL	ვერა	979,718663	46	18
2	7	ქუხიშვილი	ალექსანდრე	სამედიცინო	თბილისი	NULL	ვერა	1254,353958	57	28
3	8	ყალაბეგიშვილი	გია	სასოფლო	ქუთაისი	იმერეთი	NULL	880,135795	61	25
4	9	გიორგაძე	გივი	სამედიცინო	ქობულეთი	აჭარა	NULL	869,746878	63	26
5	10	კიკნაძე	მზია	სავაჭრო	ბათუმი	აჭარა	NULL	847,603581	48	17

გ. გვანტერესებს ინფორმაცია თბილისში მცხოვრები თანამშრომლების შესახებ. მოთხოვნას აქვს სახე:

```
USE Shekveta;
SELECT *
FROM Personali
WHERE qalaqi = N'თბილისი';
```

შედეგი:

	personaliID	gvari	saxeli	ganyofileba	qalaqi	regioni	raioni	xelfasi	asaki	staji
1	3	გაჩეჩილაძე	ლია	სამედიცინო	თბილისი	NULL	საბურთალო	1045,475332	54	10
2	4	გაჩეჩილაძე	ხათუნა	სასოფლო	თბილისი	NULL	ვერა	979,718663	46	18
3	7	ქუხიშვილი	ალექსანდრე	სამედიცინო	თბილისი	NULL	ვერა	1254,353958	57	28
4	11	სამხარაძე	გიორგი	სასოფლო	თბილისი	NULL	ვაკე	1078,967252	28	6

დ. გვანტერესებს ინფორმაცია თბილისელი და ბათუმელი თანამშრომლების შესახებ. მოთხოვნას აქვს სახე:

```
USE Shekveta;
SELECT *
FROM Personali
WHERE qalaqi = N'თბილისი' OR qalaqi = N'ბათუმი';
```

შედეგი:

	personaliID	gvari	saxeli	ganyofileba	qalaqi	regioni	raioni	xelfasi	asaki	staji
1	2	სამხარაძე	ანა	სამედიცინო	ბათუმი	აჭარა	NULL	1430,841951	26	7
2	3	გაჩეჩილაძე	ლია	სამედიცინო	თბილისი	NULL	საბურთალო	1045,475332	54	10
3	4	გაჩეჩილაძე	ხათუნა	სასოფლო	თბილისი	NULL	ვერა	979,718663	46	18
4	7	ქუხიშვილი	ალექსანდრე	სამედიცინო	თბილისი	NULL	ვერა	1254,353958	57	28
5	10	კიკნაძე	მზია	სავაჭრო	ბათუმი	აჭარა	NULL	847,603581	48	17
6	11	სამხარაძე	გიორგი	სასოფლო	თბილისი	NULL	ვაკე	1078,967252	28	6

ე. გვანტერესებს ინფორმაცია თბილისელი თანამშრომლების შესახებ, რომლებიც სამედიცინო განყოფილებაში მუშაობენ. მოთხოვნას აქვს სახე:

```
USE Shekveta;
SELECT *
FROM Personali
WHERE qalaqi = N'თბილისი' AND ganyofileba = N'სამედიცინო';
```

შედეგი:

	personaliID	gvari	saxeli	ganyofileba	qalaqi	regioni	raioni	xelfasi	asaki	staji
1	3	გაჩეჩილაძე	ლია	სამედიცინო	თბილისი	NULL	საბურთალო	1045,475332	54	10
2	7	ქუხიშვილი	ალექსანდრე	სამედიცინო	თბილისი	NULL	ვერა	1254,353958	57	28

ვ. გვაინტერესებს ინფორმაცია იმ თანამშრომლების შესახებ, რომლებიც თბილისში არ ცხოვრობენ. მოთხოვნას აქვს სახე:

```
USE Shekveta;
SELECT *
FROM Personali
WHERE NOT qalaqi = N'თბილისი';
```

შედეგი:

	personaliID	gvari	saxeli	ganyofileba	qalaqi	regioni	raioni	xelfasi	asaki	staji
1	1	სამხარაძე	საბა	სასპორტო	ზუგდიდი	სამეგრელო	NULL	1651,086382	21	2
2	2	სამხარაძე	ანა	სამედიცინო	ბათუმი	აჭარა	NULL	1430,841951	26	7
3	5	შენგელია	ნატა	სავაჭრო	ზუგდიდი	სამეგრელო	NULL	935,845284	27	5
4	6	კაპანაძე	ეთერი	სასპორტო	რუსთავი	ქართლი	NULL	1156,081531	49	9
5	8	ყალაბეგიშვილი	გია	სასოფლო	ქუთაისი	იმერეთი	NULL	880,135795	61	25
6	9	გიორგაძე	გივი	სამედიცინო	ქობულეთი	აჭარა	NULL	869,746878	63	26
7	10	კიკნაძე	მზია	სავაჭრო	ბათუმი	აჭარა	NULL	847,603581	48	17

ეს მოთხოვნა შეგვიძლია ასეც ჩავწეროთ:

```
USE Shekveta;
SELECT *
FROM Personali
WHERE qalaqi != N'თბილისი';
```

ან

```
SELECT *
FROM Personali
WHERE qalaqi <> N'თბილისი';
```

მათი შესრულების შედეგად იმავე შედეგს მივიღებთ.

ზ. გვაინტერესებს ინფორმაცია მევალეების შესახებ. მოთხოვნას აქვს სახე:

```
USE Shekveta;
SELECT Shemkveti.gvari, Shemkveti.saxeli, Shemkveti.iuridiuli_fizikuri, Xelshekruleba.vali_l,
        Xelshekruleba.gadasaxdeli_l, Xelshekruleba.gadaxdili_l
FROM Shemkveti, Xelshekruleba
WHERE Xelshekruleba.vali_l > 0 AND  Shemkveti.shemkvetiID = Xelshekruleba.shemkvetiID;
```

შედეგი:

	gvari	saxeli	iuridiuli_fizikuri	vali_l	gadasaxdeli_l	gadaxdili_l
1	სეხნიაშვილი	გია	იურიდიული	700	3700	3000
2	სეხნიაშვილი	გია	იურიდიული	3700	8000	4300
3	მამაიაშვილი	ჯემალი	იურიდიული	200	8200	8000
4	სამხარაძე	გიორგი	ფიზიკური	6400	8000	1600
5	მამაიაშვილი	ჯემალი	იურიდიული	500	6000	5500
6	სეხნიაშვილი	გია	იურიდიული	800	9800	9000
7	ნემსაძე	დოდო	ფიზიკური	300	2000	1700

თ. გვაინტერესებს თითოეული თანამშრომლის შემკვეთები და ვალის მნიშვნელობები. მოთხოვნას აქვს შემდეგი სახე:

```
SELECT  Personali.gvari, Personali.saxeli, Personali.qalaqi, Shemkveti.gvari, Xelshekruleba.vali_l
FROM    Personali, Shemkveti, Xelshekruleba
WHERE Personali.personaliID = Xelshekruleba.personaliID AND
        Xelshekruleba.shemkvetiID = Shemkveti.shemkvetiID
ORDER BY Personali.gvari, Personali.saxeli;
```

შედეგი:

	gvari	saxeli	qalaqi	gvari	vali_l
1	გაჩეჩილაძე	ლია	თბილისი	ნონიაშვილი	0
2	გაჩეჩილაძე	ლია	თბილისი	სამხარაძე	0
3	გაჩეჩილაძე	ლია	თბილისი	მამაიაშვილი	200
4	გაჩეჩილაძე	ლია	თბილისი	სამხარაძე	6400
5	გაჩეჩილაძე	ლია	თბილისი	ნემსაძე	0
6	გაჩეჩილაძე	ლია	თბილისი	ნემსაძე	300
7	გაჩეჩილაძე	ხათუნა	თბილისი	მიქელაძე	0
8	გიორგაძე	გივი	ქობულეთი	ნონიაშვილი	0
9	კიკნაძე	მზია	ბათუმი	მიქელაძე	0
10	სამხარაძე	ანა	ბათუმი	ნემსაძე	0
11	სამხარაძე	ანა	ბათუმი	სეხნიაშვილი	700
12	სამხარაძე	ანა	ბათუმი	სეხნიაშვილი	3700
13	სამხარაძე	ანა	ბათუმი	ნემსაძე	0
14	სამხარაძე	საბა	ზუგდიდი	მამაიაშვილი	500
15	სამხარაძე	საბა	ზუგდიდი	სეხნიაშვილი	800
16	სამხარაძე	საბა	ზუგდიდი	მამაიაშვილი	0
17	ქუხიშვილი	ალექ...	თბილისი	კამკამიძე	0

ამ მოთხოვნაში ORDER BY ელემენტი გამოტანილ სტრიქონებს ზრდადობით ახარისხებს gvari სვეტის მიხედვით.

ლოგიკის ოპერატორები

BETWEEN ოპერატორი

ის საშუალებას გვაძლევს მივუთითოთ მნიშვნელობების დიაპაზონი. მისი სინტაქსია:
გამოსახულება [NOT] BETWEEN *საწყისი_გამოსახულება* AND *საბოლოო_გამოსახულება*

გამოსახულება არის შესამოწმებელი გამოსახულება, რომელიც შეიძლება შეიცავდეს როგორც სვეტის, ისე ცვლადის სახელს. *საწყისი_გამოსახულება* განსაზღვრავს დიაპაზონის დასაწყისს, *საბოლოო_გამოსახულება* კი - დასასრულს.

მაგალითები.

ა. გვაინტერესებს 2005 წლის მარტის თვეში დადებული ხელშეკრულებები. მოთხოვნას აქვს სახე:

```
SET DATEFORMAT DMY;
SELECT *
FROM Xelshekruleba
WHERE tarigi_dawyebis BETWEEN '1.03.2005' AND '31.03.2005';
```

შედეგი:

	xelshekrulebaID	personaliID	shemkvetiID	gadasaxdeli_l	gadasaxdeli_d	gadaxdili_l	gadaxdili_d	vali_l	vali_d	kursi
1	12	3	1	3000	1764,71	3000	1764,71	0	0	1,7

ბ. BETWEEN სიტყვის წინ შეიძლება ცვლადის სახელის მითითებაც:

```
SET DATEFORMAT DMY;
DECLARE @tarigi DATETIME;
SET @tarigi = '19.03.2005';
SELECT @tarigi WHERE  @tarigi BETWEEN '1.03.2005' AND '31.03.2005';
```

შედეგი:

	(No column name)
1	2005-03-19 00:00:00.000

IN ოპერატორი

მისი საშუალებით შეგვიძლია შევამოწმოთ ემთხვევა თუ არა *გამოსახულების* მნიშვნელობა ქვემოთხოვნის მიერ დაბრუნებული მნიშვნელობებიდან ან ჩამოთვლილი სიდიდეებიდან ერთ-ერთს. მისი სინტაქსია:
გამოსახულება [NOT] IN (*ქვემოთხოვნა* | *გამოსახულება* [,...n])

მაგალითები.

ა. დავუშვათ, გვაინტერესებს ინფორმაცია თბილისელი ან ქუთაისელი თანამშრომლების შესახებ. მოთხოვნას აქვს სახე:

```
SELECT *
FROM Personali
WHERE qalaqi IN (N'თბილისი', N'ქუთაისი');
```

იმავე შედეგს მივიღებთ, თუ შევასრულებთ მოთხოვნას:

```
SELECT *
FROM Personali
WHERE qalaqi = N'თბილისი' OR qalaqi = N'ქუთაისი';
```

შედეგი:

	personaliID	gvari	saxeli	ganyofileba	qalaqi	regioni	raioni	xelfasi	asaki	staji
1	3	გაჩეჩილაძე	ლია	სავაჭრო	თბილისი	NULL	საბურთალო	950,43212	30	10
2	4	გაჩეჩილაძე	ხათუნა	სასოფლო	თბილისი	NULL	ვერა	890,65333	40	18
3	7	ქუხიშვილი	ალექსანდრე	სამედიცინო	თბილისი	NULL	ვერა	1140,32178	53	28
4	8	ყალაბეგიშვილი	გია	სასოფლო	ქუთაისი	იმერეთი	NULL	800,12345	55	25
5	11	სამხარაძე	ბექა	სასოფლო	თბილისი	NULL	ვაკე	980,87932	25	6
6	12	სამხარაძე	გიორგი	სავაჭრო	თბილისი	NULL	ჩუღურეთი	1020,43278	22	4
7	15	სამხარაძე	ბექა	სასოფლო	თბილისი	NULL	ვაკე	300,55	27	5

სერვერი პირველ მოთხოვნას უფრო სწრაფად შეასრულებს, ვიდრე მეორეს.

ბ. დავუშვათ, გვაინტერესებს ფირმის ის თანამშრომლები, რომლებიც ერთ ქალაქში იმყოფებიან იმ შემკვეთებთან ერთად, რომლებიც იურიდიულ პირებს წარმოადგენენ. მოთხოვნას აქვს სახე:

```
SELECT gvari, saxeli, qalaqi
FROM Personali
WHERE qalaqi IN
(
SELECT qalaqi
FROM Shemkveti
WHERE iuridiuli_fizikuri = N'იურიდიული'
);
```

შედეგი:

	gvari	saxeli	qalaqi
1	სამხარაძე	საბა	ზუგდიდი
2	სამხარაძე	ანა	ბათუმი
3	გაჩეჩილაძე	ლია	თბილისი
4	გაჩეჩილაძე	ხათუნა	თბილისი
5	შენგელია	ნატა	ზუგდიდი
6	ქუხიშვილი	ალექსანდრე	თბილისი
7	კიკნაძე	მზია	ბათუმი
8	სამხარაძე	ბექა	თბილისი
9	სამხარაძე	გიორგი	თბილისი
10	სამხარაძე	ბექა	თბილისი

როგორც ვხედავთ, IN ოპერატორის შემდეგ მრგვალ ფრჩხილებში მოთავსებულია ქვემოთხოვნა (SELECT ბრძანება). IN ოპერატორი შეამოწმებს SELECT ბრძანების მიერ გაცემულ მნიშვნელობებს.

LIKE ოპერატორი

მისი საშუალებით შესაძლებელია გამოსახულების შედარება მოცემულ შაბლონთან. LIKE ოპერატორის გამოყენება მოხერხებულია მაშინ, როცა ზუსტად არ ვიცით სტრიქონის მნიშვნელობა. მისი სინტაქსია:

გამოსახულება [NOT] LIKE *შაბლონი* [ESCAPE escape_*სიმბოლო*]

გამოსახულება არგუმენტი განსაზღვრავს შესამოწმებელ გამოსახულებას. შაბლონი

შეიცავს სიმბოლო_შემცვლელებს. ისინი მოყვანილია ცხრილში 4.3.

ცხრილი 4.3. სიმბოლო-შემცვლელები.

#		სიმბოლო	აღწერა
1		_	შეესაბამება ნებისმიერ ერთ სიმბოლოს
2		%	შეესაბამება სიმბოლოების ნებისმიერ მიმდევრობას
3		[ ]	შეესაბამება ფრჩხილებში ჩამოთვლილ ნებისმიერ ერთ სიმბოლოს
4		[^]	შეესაბამება ნებისმიერ ერთ სიმბოლოს, გარდა ფრჩხილებში ჩამოთვლილი სიმბოლოებისა
5		[-]	შეესაბამება სიმბოლოების დიაპაზონს
6		#	შეესაბამება ნებისმიერ ერთ ციფრს

როგორც ვხედავთ, შაბლონში დარეზერვებული სიმბოლოები (%, _, [,] და ^) გამოიყენება. escape_*სიმბოლო* არგუმენტით მოიცემა მმართველი სიმბოლო. ამ სიმბოლოს შემდეგ მოთავსებული სიმბოლო-შემცვლელი აღარ მოქმედებს და ჩაითვლება ჩვეულებრივ სიმბოლოდ. მაგალითად, LIKE '__45#%' ESCAPE '#' ბრძანება შეასრულებს ხუთსიმბოლოიანი სტრიქონის ძებნას, რომელიც „45%” სიმბოლოებით მთავრდება. ამ შემთხვევაში % განიხილება როგორც ჩვეულებრივი სიმბოლო.

% სიმბოლო ცვლის სიმბოლოების ნებისმიერ მიმდევრობას. მაგალითად, 'ბა%ი' შაბლონს შეესაბამება 'ბავშვი', 'ბარტყი', ბარი' და ა.შ. _ (ხაზგასმის სიმბოლო) სიმბოლო ცვლის ნებისმიერ ერთ სიმბოლოს. მაგალითად, 'ბა_ი' შაბლონს შეესაბამება 'ბალი', 'ბარი' და არა 'ბავშვი'.

მაგალითები.

ა. გვაინტერესებს შემკვეთები, რომელთა სახელები 'ჯ' ასოთი იწყება. მოთხოვნას აქვს სახე:

```
SELECT firmis_dasaxeleba, gvari, qalaqi
FROM Shemkveti
WHERE firmis_dasaxeleba LIKE N'g%';
```

შედეგი:

	firmis_dasaxeleba	gvari	qalaqi
1	Gia&Co.	სეხნიაშვილი	თელავი
2	Givi&Co.	კამკამიძე	ზუგდიდი
3	Gela&Co.	ჭიკაძე	გორი

ბ. დავუშვათ, გვაინტერესებს ის თანამშრომლები, რომელთა გვარის მეორე სიმბოლოა 'ა'. მოთხოვნას აქვს სახე:

```
SELECT gvari, qalaqi
FROM Personali
WHERE gvari LIKE N'_ა%';
```

შედეგი:

	gvari	qalaqi
1	სამხარაძე	ზუგდიდი
2	სამხარაძე	ბათუმი
3	გაჩეჩილაძე	თბილისი
4	გაჩეჩილაძე	თბილისი
5	კაპანაძე	რუსთავი
6	ყალაბეგიშვილი	ქუთაისი
7	სამხარაძე	თბილისი
8	სამხარაძე	თბილისი
9	სამხარაძე	თბილისი

გ. გვაინტერესებს ის შემკვეთები, რომლებიც იმ ქალაქებში ცხოვრობენ, რომელთა სახელების პირველი ასოებია 'თ' ან 'ბ'. მოთხოვნას აქვს სახე:

```
SELECT *
FROM Shemkveti
WHERE qalaqi LIKE N'[თბ]%';
```

შედეგი:

	shemkvetiID	personaliID	iuridiuli_fizikuri	gvari	saxeli	qalaqi	regioni	raioni
1	1	3	ფიზიკური	ნემსაძე	დოდო	თბილისი	NULL	ავლაბარი
2	2	7	იურიდიული	სეხნიაშვილი	გია	თელავი	კახეთი	NULL
3	3	8	იურიდიული	მამაიაშვილი	ჯემალი	თელავი	კახეთი	NULL
4	5	2	იურიდიული	ნონიაშვილი	ზურა	ბათუმი	აჭარა	NULL
5	6	3	ფიზიკური	სამხარაძე	არჩილი	ბათუმი	აჭარა	NULL
6	10	1	იურიდიული	ლომიძე	რუსუდანი	თბილისი	NULL	საბურთალო
7	11	1	ფიზიკური	ხომტარია	სიმონი	თბილისი	NULL	მთაწმინდა
8	12	8	ფიზიკური	ლასურაშვილი	ია	თბილისი	NULL	დიდუბი
9	13	NULL	ფიზიკური	სამხარაძე	ბექა	თელავი	კახეთი	NULL
10	14	NULL	ფიზიკური	კაპანაძე	მერაბი	თბილისი	NULL	დიდუბე
11	15	7	იურიდიული	ჯდარკავა	ანა	თბილისი	NULL	დიდომი

დ. გვაინტერესებს ის ფირმები, რომლებიც იმ ქალაქებში მდებარეობენ, რომელთა დასახელება არ იწყება 'თ' და 'ბ' ასოებით. მოთხოვნას აქვს სახე:

```
SELECT qalaqi, firmis_dasaxeleba, gvari
FROM Shemkveti
WHERE qalaqi LIKE N'[^თბ]%';
```

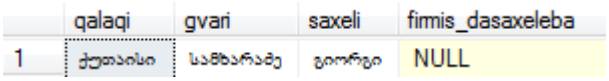
შედეგი:

	qalaqi	firmis_dasaxeleba	gvari
1	ქუთაისი	NULL	სამხარაძე
2	ზესტაფონი	NULL	მიქელაძე
3	ზუგდიდი	Givi&Co.	კამკამიძე
4	გორი	Gela&Co.	ჭიკაძე

ე. გვაინტერესებს ის შემკვეთები, რომლებიც ცხოვრობენ იმ ქალაქებში, რომელთა სახელების პირველი სიმბოლო მოთავსებულია 'მ' და 'ჯ' ასოებს შორის. მოთხოვნას აქვს სახე:

```
SELECT qalaqi, gvari, saxeli, firmis_dasaxeleba
FROM Shemkveti
WHERE qalaqi LIKE N'[მ-ჯ]%';
```

შედეგი:

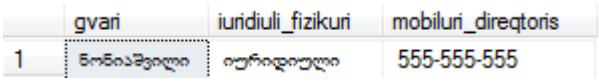


	qalaqi	gvari	saxeli	firmis_dasaxeleba
1	ქუთაისი	სამხარაძე	გიორგი	NULL

ვ. გვაინტერესებს ის შემკვეთები, რომელთა მობილური ტელეფონის ნომრები 555-ით იწყება. მოთხოვნას აქვს სახე:

```
SELECT gvari, iuridiuli_fizikuri, mobiluri_direqtoris
FROM Shemkveti
WHERE mobiluri_direqtoris LIKE '555%';
```

შედეგი:



	gvari	iuridiuli_fizikuri	mobiluri_direqtoris
1	ნონიაშვილი	იურიდიული	555-555-555

აგრეგირების ფუნქციები

აგრეგირების ფუნქციები მონაცემების სტატისტიკურ დამუშავებას ასრულებენ. განვიხილოთ თითოეული მათგანი.

AVG ფუნქცია. ის მითითებული სვეტისთვის გამოთვლის საშუალო არითმეტიკულს. თუ მონაცემები დაჯგუფებულია, მაშინ თითოეული ჯგუფისთვის გამოითვლება საშუალო არითმეტიკული. მისი სინტაქსია:

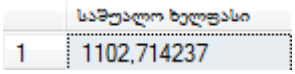
AVG ([ALL | DISTINCT] *გამოსახულება*)

აგრეგირების ფუნქციების უმრავლესობა არ ითვალისწინებს NULL მნიშვნელობის შემცველ სვეტებს. ALL საკვანძო სიტყვა მიუთითებს, რომ აგრეგირება უნდა შეეხოს გამეორებად (ერთნაირ) სტრიქონებს. ეს არგუმენტი ავტომატურად იგულისხმება იმ შემთხვევაში, თუ ის მითითებული არ არის. DISTINCT საკვანძო სიტყვა მიუთითებს, რომ აგრეგირება შეეხება მხოლოდ უნიკალურ სტრიქონებს.

მაგალითი. გვაინტერესებს თანამშრომლების საშუალო ხელფასი. მოთხოვნას აქვს სახე:

```
USE Shekveta;
SELECT AVG(xelfasi) AS [საშუალო ხელფასი]
FROM Personali;
```

შედეგი:



	საშუალო ხელფასი
1	1102,714237

COUNT ფუნქცია თვლის მნიშვნელობების რაოდენობას მოცემულ სვეტში ან სტრიქონების საერთო რაოდენობას ცხრილში. მისი სინტაქსია:

COUNT ({ [ALL | DISTINCT] *გამოსახულება*] | * })

* მიუთითებს, რომ გაიცემა ცხრილში მოთავსებული სტრიქონების რაოდენობა.

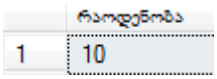
მაგალითები.

ა. გვაინტერესებს ფირმის რამდენ თანამშრომელს აქვს ხელშეკრულება გაფორმებული, ანუ Xelshekruleba ცხრილში თანამშრომლების იდენტიფიკატორების რაოდენობა. მოთხოვნას აქვს სახე:

```
USE Shekveta;
SELECT COUNT(DISTINCT personaliID) AS [რაოდენობა]
```

FROM Xelshekruleba;

შედეგი:



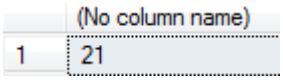
	რაოდენობა
1	10

ყურადღება მივაქციოთ იმას, რომ SELECT-მოთხოვნაში გამოყენებულია DISTINCT სიტყვა, შედეგად, personaliID სვეტში დაითვლება უნიკალური მნიშვნელობები.

ბ. თუ DISTINCT სიტყვას არ მივუთითებთ, მაშინ გაიცემა ხელშეკრულებების რაოდენობა. მოთხოვნას აქვს სახე:

SELECT COUNT(personaliID)
FROM Xelshekruleba;

შედეგი:

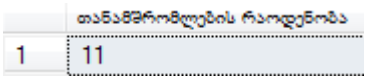


	(No column name)
1	21

გ. დავუშვათ, გვაინტერესებს ფირმაში თანამშრომლების რაოდენობა. მოთხოვნას აქვს სახე:

SELECT COUNT(*) AS [თანამშრომლების რაოდენობა]
FROM Personali;

შედეგი:

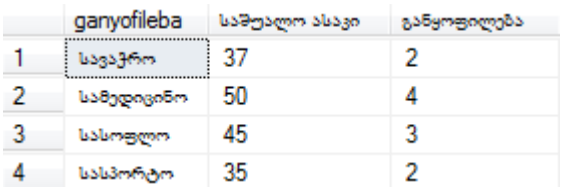


	თანამშრომლების რაოდენობა
1	11

დ. გვაინტერესებს თითოეულ განყოფილებაში მომუშავე თანამშრომლების საშუალო ასაკი და მათი რაოდენობა. მოთხოვნას აქვს სახე:

SELECT ganyofileba, AVG(asaki) AS [საშუალო ასაკი], COUNT(*) AS [განყოფილება]
FROM Personali
GROUP BY ALL ganyofileba;

შედეგი:



	ganyofileba	საშუალო ასაკი	განყოფილება
1	სავაჭრო	37	2
2	სამედიცინო	50	4
3	სასოფლო	45	3
4	სასპორტო	35	2

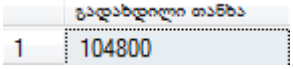
SUM ფუნქცია ასრულებს მითითებული სვეტის მნიშვნელობების შეკრებას. მისი სინტაქსია:

SUM ([ALL | DISTINCT] *გამოსახულება*)

მაგალითი. გვაინტერესებს ყველა შემკვეთის მიერ გადახდილი თანხა. მოთხოვნას აქვს სახე:

USE Shekveta;
SELECT SUM(gadaxdili_l) AS [გადახდილი თანხა]
FROM Xelshekruleba;

შედეგი:



	გადახდილი თანხა
1	104800

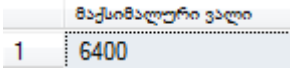
MAX ფუნქცია გასცემს მითითებული სვეტის მნიშვნელობებიდან მაქსიმალურს. მისი სინტაქსია:

MAX ([ALL | DISTINCT] *გამოსახულება*)

მაგალითი. გვაინტერესებს ვალის მაქსიმალური სიდიდე. მოთხოვნას აქვს სახე:

```
USE Shekveta;
SELECT MAX(vali_l) AS [მაქსიმალური ვალი]
FROM Xelshekruleba;
```

შედეგი:



	მაქსიმალური ვალი
1	6400

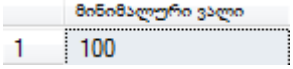
MIN ფუნქცია გასცემს მითითებული სვეტის მნიშვნელობებიდან მინიმალურს. მისი სინტაქსია:

MIN ([ALL | DISTINCT] *გამოსახულება*)

მაგალითი. გვაინტერესებს ვალის მინიმალური სიდიდე. მოთხოვნას აქვს სახე:

```
USE Shekveta;
SELECT MIN(vali_l) AS [მინიმალური ვალი]
FROM Xelshekruleba WHERE vali_l > 0;
```

შედეგი:



	მინიმალური ვალი
1	100

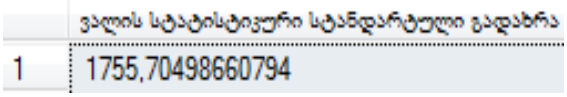
STDEV ფუნქცია გასცემს სტატისტიკურ სტანდარტულ გადახრას მითითებული სვეტის მნიშვნელობებისათვის. მისი სინტაქსია:

STDEV (*გამოსახულება*)

მაგალითი. გვაინტერესებს ვალის სტატისტიკური სტანდარტული გადახრა. მოთხოვნას აქვს სახე:

```
USE Shekveta;
SELECT STDEV(vali_l) AS [ვალის სტატისტიკური სტანდარტული გადახრა]
FROM Xelshekruleba;
```

შედეგი:



	ვალის სტატისტიკური სტანდარტული გადახრა
1	1755,70498660794

STDEVP ფუნქცია გასცემს სტანდარტული გადახრის შერეულ შეფასებას მითითებული სვეტის მნიშვნელობებისათვის. მისი სინტაქსია:

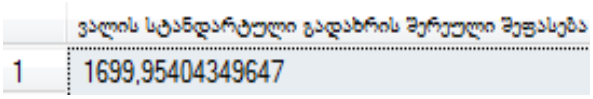
STDEVP (*გამოსახულება*)

მაგალითი. გვაინტერესებს ვალის სტანდარტული გადახრის შერეული შეფასება. მოთხოვნას აქვს სახე:

```
USE Shekveta;
SELECT STDEVP(vali_l) AS [ვალის სტანდარტული გადახრის შერეული შეფასება]
```

FROM Xelshekruleba;

შედეგი:



	ვალის სტანდარტული გადახრის შერეული შეფასება
1	1699,95404349647

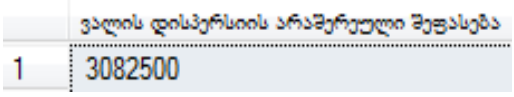
VAR ფუნქცია გასცემს მითითებული სვეტის მნიშვნელობების დისპერსიის არაშერეულ შეფასებას. მისი სინტაქსია:

VAR (*გამოსახულება*)

მაგალითი. გვაინტერესებს ვალის დისპერსიის არაშერეული შეფასება. მოთხოვნას აქვს სახე:

USE Shekveta;
SELECT VAR(vali_l) AS [ვალის დისპერსიის არაშერეული შეფასება]
FROM Xelshekruleba;

შედეგი:



	ვალის დისპერსიის არაშერეული შეფასება
1	3082500

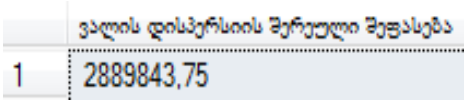
VARP ფუნქცია გასცემს მითითებული სვეტის მნიშვნელობების დისპერსიის შერეულ შეფასებას. მისი სინტაქსია:

VARP (*გამოსახულება*)

მაგალითი. გვაინტერესებს ვალის დისპერსიის შერეული შეფასება. მოთხოვნას აქვს სახე:

USE Shekveta;
SELECT VARP(vali_l) AS [ვალის დისპერსიის შერეული შეფასება]
FROM Xelshekruleba;

შედეგი:



	ვალის დისპერსიის შერეული შეფასება
1	2889843,75

GROUP BY განყოფილება

GROUP BY საკვანძო სიტყვა საშუალებას გვაძლევს ცხრილის სტრიქონები გარკვეული კრიტერიუმის მიხედვით დავაჯგუფოთ. თითოეული ჯგუფისათვის შეიძლება აგრეგირების ფუნქციების გამოყენება (ისინი ამ თავშია განხილული), რომლებიც შესრულდება ჯგუფის ყველა სტრიქონის მიმართ. ყველა სტადია, რომელიც მოსდევს GROUP BY სტადიას, HAVING, SELECT და ORDER BY ჩათვლით, უნდა ოპერირებდეს ჯგუფებზე, და არა ცალკეულ სტრიქონებზე. შედეგად, მოთხოვნის საბოლოო შედეგში თითოეული ჯგუფი წარმოდგენილია ერთი სტრიქონით. ეს იმას ნიშნავს რომ, გამოსახულებები, რომლებიც მუშავდება GROUP BY სტადიის მომდევნო სტადიებზე, თითოეული ჯგუფისთვის უნდა გასცემდეს სკალარულ სიდიდეს. GROUP BY საკვანძო სიტყვის სინტაქსია:

[GROUP BY [ALL] *სვეტების_სია* [,…]]

განვიხილოთ არგუმენტების დანიშნულება.

- *სვეტების_სია* არგუმენტში ეთითება იმ სვეტების სახელები, რომელთა მიხედვით უნდა შესრულდეს დაჯგუფება.
- ALL. თუ მოთხოვნაში განსაზღვრულია პირობა, რომელიც ზღუდავს ამოსარჩევი სტრიქონების დაჯგუფების დიაპაზონს, მაშინ GROUP BY განყოფილებაში ALL სიტყვის გამოყენება გამოიწვევს ყველა ჯგუფის გამოტანას. აგრეგირების ფუნქცია არ შესრულდება იმ ჯგუფებისათვის, რომლებიც არ აკმაყოფილებენ მოთხოვნაში მითითებულ პირობას.

მაგალითები.

ა. გვაინტერესებს შეკვეთის მაქსიმალური თანხა თითოეული თანამშრომლისათვის. მოთხოვნას აქვს სახე:

```
USE Shekveta;
SELECT personaliID, MAX(gadasaxdeli_l) AS [გადასახდელი თანხა]
FROM Xelshekruleba
GROUP BY personaliID;
```

შედეგი:

	personaliID	გადასახდელი თანხა
1	1	9800
2	2	8000
3	3	8200
4	4	4400
5	5	5300
6	6	4400
7	7	3500
8	8	7700
9	9	10000
10	10	3900

აქ MAX არის აგრეგირების ფუნქცია, რომელიც გასცემს მითითებული სვეტის მაქსიმალურ მნიშვნელობას.

ბ. გვაინტერესებს თითოეული თანამშრომლის შეკვეთების მაქსიმალური თანხები თითოეული თარიღისათვის. მოთხოვნას აქვს სახე:

```
SELECT personaliID, tarigi_dawyebis, MAX(gadasaxdeli_l) AS [გადასახდელი თანხა]
FROM Xelshekruleba
GROUP BY personaliID, tarigi_dawyebis
ORDER BY personaliID, tarigi_dawyebis;
```

შედეგი:

	personaliID	tarigi_dawyebis	გადასახდელი თანხა
1	1	2009-05-07 00:00:00.000	9800
2	1	2012-10-10 00:00:00.000	7100
3	2	2009-01-01 00:00:00.000	3700
4	2	2011-01-01 00:00:00.000	8000
5	3	2007-02-01 00:00:00.000	2500
6	3	2008-10-17 00:00:00.000	8000
7	3	2009-08-20 00:00:00.000	1900
8	3	2012-02-25 00:00:00.000	8200
9	3	2012-03-03 00:00:00.000	3000
10	4	2011-11-14 00:00:00.000	4400
11	5	2015-10-01 00:00:00.000	5300
12	6	2015-12-01 00:00:00.000	4400
13	7	2010-07-10 00:00:00.000	3500
14	8	2008-08-05 00:00:00.000	7700
15	8	2013-02-01 00:00:00.000	5600
16	9	2009-07-09 00:00:00.000	10000
17	9	2011-07-10 00:00:00.000	8500
18	10	2010-05-21 00:00:00.000	3900

როგორც ვხედავთ, დაჯგუფება ორივე სვეტის მიხედვით შესრულდა სამი თანამშრომლისთვის, რომელთა იდენტიფიკატორებია 1, 2 და 3. თანამშრომელს, რომლის იდენტიფიკატორია 1, გაფორმებული აქვს სამი ხელშეკრულება. ორი ხელშეკრულება გაფორმდა 2012 წლის 10 ოქტომბერს, ერთი კი - 2009 წლის 7 მაისს. დაჯგუფება შესრულდა 2012 წლის 10 ოქტომბერს გაფორმებული ხელშეკრულებებისთვის, რომელთა თანხებია 6000 და 7100. გაიცა მათ შორის მაქსიმალური - 7100. რადგან მესამე ხელშეკრულება სხვა დროს გაფორმდა, ამიტომ ის ვერ დაჯგუფდა. იგივე ეხება თანამშრომლებს, რომელთა იდენტიფიკატორია 2 და 3.

გ. გვაინტერესებს, თითოეული თანამშრომლის იმ შეკვეთების თანხების ჯამი, რომელთა თანხა 7000-ზე ნაკლებია. მოთხოვნას აქვს სახე:

```
SELECT personaliID, SUM(gadasaxdeli_l) AS [გადასახდელი თანხა]
FROM Xelshekruleba
WHERE gadasaxdeli_l < 7000
GROUP BY personaliID;
```

შედეგი:

	personaliID	გადასახდელი თანხა
1	1	6000
2	2	8700
3	3	9400
4	4	4400
5	5	5300
6	6	4400
7	7	3500
8	8	5600
9	10	3900

დ. წინა მოთხოვნას დავუმატოთ ALL საკვანძო სიტყვა:

```
SELECT personaliID, SUM(gadasaxdeli_l) AS [გადასახდელი თანხა]
FROM Xelshekruleba
WHERE gadasaxdeli_l < 7000
GROUP BY ALL personaliID;
```

შედეგი:

	personaliID	გადასახდელი თანხა
1	1	6000
2	2	8700
3	3	9400
4	4	4400
5	5	5300
6	6	4400
7	7	3500
8	8	5600
9	9	NULL
10	10	3900

როგორც შედეგიდან ჩანს, SUM() აგრეგირების ფუნქცია არ შესრულდა იმ ჯგუფებისათვის, რომლებმაც არ დააკმაყოფილეს მოთხოვნაში მითითებული პირობა (gadasaxdeli_l < 7000).

HAVING განყოფილება

ეს განყოფილება გამოიყენება ძებნის პირობის მისათითებლად ჯგუფისთვის. მისი სინტაქსია:

HAVING <*ძებნის_პირობა*>

თუ HAVING განყოფილება გამოიყენება GROUP BY განყოფილების გარეშე, მაშინ ის იმუშავებს WHERE განყოფილების მსგავსად. თუ HAVING განყოფილება გამოიყენება GROUP BY ALL კონსტრუქციასთან ერთად, მაშინ HAVING ფარავს ALL სიტყვის მოქმედებას.

მაგალითები.

ა. გვაინტერესებს თითოეული თანამშრომლის შეკვეთების მაქსიმალური თანხა, რომელიც აღემატება 7000-ს. მოთხოვნას აქვს სახე:

```
USE Shekveta;
SELECT personaliID, MAX(gadasaxdeli_l) AS [გადასახდელი თანხა]
FROM Xelshekruleba
GROUP BY personaliID
HAVING MAX(gadasaxdeli_l) > 7000;
```

შედეგი:

	personaliID	გადასახდელი თანხა
1	1	9800
2	2	8000
3	3	8200
4	8	7700
5	9	10000

ბ. გვაინტერესებს იმ თანამშრომლების რაოდენობა, რომელთა სტაჟი მეტია თბილისელი თანამშრომლების საშუალო სტაჟზე. მოთხოვნას აქვს სახე:

```
SELECT staji, COUNT(DISTINCT PersonaliID)
FROM Personali
GROUP BY staji
HAVING staji >
(
SELECT AVG(staji)
FROM Personali
WHERE qalaqi = N'თბილისი'
);
```

შედეგი:

	staji	(No column name)
1	17	1
2	18	1
3	25	1
4	26	1
5	28	1

ORDER BY განყოფილება

ORDER BY განყოფილება გამოიყენება ცხრილის დასახარისხებლად ამა თუ იმ სვეტის მნიშვნელობების ზრდის ან კლების მიხედვით. მისი სინტაქსია:

ORDER BY { *სვეტების_სია* [ASC | DESC] } [,…]

სვეტების_სია არგუმენტი შეიცავს ერთი ან მეტი სვეტის სახელს. ASC საკვანძო სიტყვა გამოიყენება მითითებული სვეტის მონაცემების დასალაგებლად ზრდადობის მიხედევით, DESC სიტყვა კი - კლებადობის მიხედვით. თუ ეს არგუმენტები მითითებული არ არის, მაშინ დახარისხება ზრდადობის მიხედვით შესრულდება.

მაგალითები.

ა. მოყვანილი მოთხოვნის მიერ გაცემულ შედეგში tarigi_dawyebis სვეტის მონაცემები დახარისხებული იქნება ზრდადობით:

```
USE Shekveta;
SELECT personaliID, tarigi_dawyebis, gadasaxdeli_l, vali_l
FROM Xelshekruleba
ORDER BY tarigi_dawyebis;
```

შედეგი:

	personaliID	tarigi_dawyebis	gadasaxdeli_l	vali_l
1	3	2007-02-01 00:00:00.000	2500	0
2	8	2008-08-05 00:00:00.000	7700	0
3	3	2008-10-17 00:00:00.000	8000	6400
4	2	2009-01-01 00:00:00.000	3700	700
5	1	2009-05-07 00:00:00.000	9800	800
6	9	2009-07-09 00:00:00.000	10000	0
7	3	2009-08-20 00:00:00.000	1900	0
8	10	2010-05-21 00:00:00.000	3900	0
9	7	2010-07-10 00:00:00.000	3500	0
10	2	2011-01-01 00:00:00.000	5000	0
11	2	2011-01-01 00:00:00.000	8000	3700
12	9	2011-07-10 00:00:00.000	8500	100
13	4	2011-11-14 00:00:00.000	4400	0
14	3	2012-02-25 00:00:00.000	8200	200
15	3	2012-03-03 00:00:00.000	2000	300
16	3	2012-03-03 00:00:00.000	3000	0
17	1	2012-10-10 00:00:00.000	7100	0
18	1	2012-10-10 00:00:00.000	6000	500
19	8	2013-02-01 00:00:00.000	5600	0
20	5	2015-10-01 00:00:00.000	5300	500
21	6	2015-12-01 00:00:00.000	4400	500

ბ. მოყვანილი მოთხოვნის მიერ გაცემულ შედეგში tarigi_dawyebis სვეტის მონაცემები დახარისხებული იქნება კლებადობით:

```
SELECT personaliID, tarigi_dawyebis, gadasaxdeli_l, vali_l
FROM Xelshekruleba
ORDER BY tarigi_dawyebis DESC;
```

შედეგი:

	personaliID	tarigi_dawyebis	gadasaxdeli_l	vali_l
1	6	2015-12-01 00:00:00.000	4400	500
2	5	2015-10-01 00:00:00.000	5300	500
3	8	2013-02-01 00:00:00.000	5600	0
4	1	2012-10-10 00:00:00.000	6000	500
5	1	2012-10-10 00:00:00.000	7100	0
6	3	2012-03-03 00:00:00.000	3000	0
7	3	2012-03-03 00:00:00.000	2000	300
8	3	2012-02-25 00:00:00.000	8200	200
9	4	2011-11-14 00:00:00.000	4400	0
10	9	2011-07-10 00:00:00.000	8500	100
11	2	2011-01-01 00:00:00.000	5000	0
12	2	2011-01-01 00:00:00.000	8000	3700
13	7	2010-07-10 00:00:00.000	3500	0
14	10	2010-05-21 00:00:00.000	3900	0
15	3	2009-08-20 00:00:00.000	1900	0
16	9	2009-07-09 00:00:00.000	10000	0
17	1	2009-05-07 00:00:00.000	9800	800
18	2	2009-01-01 00:00:00.000	3700	700
19	3	2008-10-17 00:00:00.000	8000	6400
20	8	2008-08-05 00:00:00.000	7700	0
21	3	2007-02-01 00:00:00.000	2500	0

გ. ORDER BY სიტყვის შემდეგ შეგვიძლია მივუთითოთ ორი ან მეტი სვეტის სახელი:

```
SELECT personaliID, tarigi_dawyebis, gadasaxdeli_l, vali_l
FROM Xelshekruleba
ORDER BY tarigi_dawyebis, vali_l DESC;
```

შედეგი:

	personaliID	tarigi_dawyebis	gadasaxdeli_l	vali_l
1	3	2007-02-01 00:00:00.000	2500	0
2	8	2008-08-05 00:00:00.000	7700	0
3	3	2008-10-17 00:00:00.000	8000	6400
4	2	2009-01-01 00:00:00.000	3700	700
5	1	2009-05-07 00:00:00.000	9800	800
6	9	2009-07-09 00:00:00.000	10000	0
7	3	2009-08-20 00:00:00.000	1900	0
8	10	2010-05-21 00:00:00.000	3900	0
9	7	2010-07-10 00:00:00.000	3500	0
10	2	2011-01-01 00:00:00.000	8000	3700
11	2	2011-01-01 00:00:00.000	5000	0
12	9	2011-07-10 00:00:00.000	8500	100
13	4	2011-11-14 00:00:00.000	4400	0
14	3	2012-02-25 00:00:00.000	8200	200
15	3	2012-03-03 00:00:00.000	2000	300
16	3	2012-03-03 00:00:00.000	3000	0
17	1	2012-10-10 00:00:00.000	6000	500
18	1	2012-10-10 00:00:00.000	7100	0
19	8	2013-02-01 00:00:00.000	5600	0
20	5	2015-10-01 00:00:00.000	5300	500
21	6	2015-12-01 00:00:00.000	4400	500

ამ შემთხვევაში Xelshekruleba ცხრილის სტრიქონები ჯერ დახარისხდება - tarigi_dawyebis სვეტის მნიშვნელობების მიხედვით ზრდადობით, შემდეგ კი vali_l სვეტის მნიშვნელობების მიხედვით - კლებადობით.

COMPUTE განყოფილება

ეს განყოფილება საშუალებას გვაძლევს არჩეული სვეტების მიმართ გამოვიყენოთ აგრეგირების ფუნქციები. აგრეგირების შედეგი გაიცემა ცალკე სტრიქონის სახით, მონაცემების შემდეგ. მისი სინტაქსია:

COMPUTE { { AVG | COUNT | MAX | MIN | STDEV | STDEVP | VWR | VARP | SUM }
(*გამოსახულება*) } [,...] [BY *გამოსახულება* [,...]]]

გამოსახულება არგუმენტი უნდა შეიცავდეს სააგრეგირებელი სვეტის სახელს, რომელიც ჩართული იქნება შედეგში. BY სიტყვა მიუთითებს, რომ გამოთვლის შედეგი უნდა დაჯგუფდეს. BY სიტყვის შემდეგ მოთავსებული *გამოსახულება* არგუმენტი შეიცავს იმ სვეტის სახელს, რომლის მიხედვით უნდა შესრულდეს დაჯგუფება. შედეგი წინასწარ უნდა იყოს დახარისხებული ამ სვეტის მიხედვით, ე.ი. სვეტის სახელი უნდა იყოს მითითებული SELECT მოთხოვნის ORDER BY განყოფილებაში. COMPUTE განყოფილებაში დაუშვებელია DISTINCT საკვანძო სიტყვის გამოყენება.

მაგალითები.

ა. გვაინტერესებს ყველა იმ ხელშეკრულებით გადასახდელი თანხების ჯამი და ხელშეკრულებების რაოდენობა, რომლებშიც გადასახდელი თანხა 2000-ს აღემატება. მოთხოვნას აქვს სახე:

```
USE Shekveta;
SELECT personaliID, gadasaxdeli_l,  vali_l, shesruleba
FROM Xelshekruleba
WHERE gadasaxdeli_l > 2000
ORDER BY personaliID
```

COMPUTE SUM(gadasaxdeli_l), COUNT(gadasaxdeli_l);

შედეგი:

	personaliID	gadasaxdeli_l	vali_l	shesruleba
1	1	6000	500	არა
2	1	9800	800	არა
3	1	7100	0	კი
4	2	5000	0	კი
5	2	3700	700	არა
6	2	8000	3700	კი
7	3	2500	0	კი
8	3	8200	200	არა
9	3	3000	0	კი
10	3	8000	6400	კი
11	4	4400	0	კი
12	5	5300	500	არა
13	6	4400	500	არა
14	7	3500	0	კი
15	8	7700	0	კი
16	8	5600	0	არა
17	9	10000	0	კი

	sum	cnt
1	114600	19

ბ. ახლა იგივე მოთხოვნა ჩავწეროთ BY სიტყვის გამოყენებით:

SELECT personaliID, gadasaxdeli_l, vali_l, shesruleba
FROM Xelshekruleba
WHERE gadasaxdeli_l > 9000
ORDER BY personaliID
COMPUTE SUM(gadasaxdeli_l), COUNT(gadasaxdeli_l) BY personaliID;

შედეგი:

	personaliID	gadasaxdeli_l	vali_l	shesruleba
1	1	9800	800	არა

	sum	cnt
1	9800	1

	personaliID	gadasaxdeli_l	vali_l	shesruleba
1	9	10000	0	კი

	sum	cnt
1	10000	1

OVER ელემენტი

OVER ელემენტი წარმოგვიდგენს სტრიქონების ფანჯარას ან სექციას სათანადო სახის გამოთვლებისთვის. ფანჯარა არის სტრიქონების სიმრავლე, რომლებზეც გამოთვლები სრულდება. გამოთვლების მაგალითად, რომელსაც OVER ელემენტი ასრულებს, გამოდგება აგრეგირების ან მარანჟირებელი ფუნქციები. რადგან, OVER ელემენტი ამ ფუნქციებს სტრიქონების ფანჯრის მიმართ ასრულებს, ამიტომ ამ ფუნქციებს ფანჯრული ეწოდება.

აგრეგირების ფანჯრული ფუნქცია ამუშავებს მნიშვნელობების სიმრავლეს სტრიქონების ფანჯრიდან. სტრიქონების ეს სიმრავლე აგრეგირების ფუნქციას მიეწოდება OVER ელემენტის, და არა GROUP BY განყოფილების საშუალებით. შედეგად, აღარ დაგვჭირდება მონაცემების დაჯგუფება, და შეგვეძლება საბაზო სტრიქონის სვეტებისა და აგრეგირების ფუნქციების მიერ გაცემული შედეგების ერთსა და იმავე სტრიქონში დაბრუნება.

OVER ელემენტის მუშაობის პრინციპის აღწერის მიზნით, განვიხილოთ Xelshekruleba ცხრილი. მისი თითოეული სტრიქონი შეიცავს შეკვეთის იდენტიფიკატორს (xelshekrulebaID), შემკვეთის იდენტიფიკატორს (shemkvetiID), თანამშრომლის იდენტიფიკატორს (personaliID), შეკვეთის დაწყების თარიღს (tarigi_dawyebis) და შეკვეთის ღირებულებას (gadasaxdeli_l).

თუ OVER ელემენტს მოსდევს ცარიელი ფრჩხილები, მაშინ ის გასცემს ყველა სტრიქონს გამოთვლისთვის. ფრაზა "ყველა სტრიქონი" არ ნიშნავს იმ ცხრილის ყველა სტრიქონს, რომელიც FROM განყოფილებაშია მითითებული. გაიცემა ყველა ის სტრიქონი, რომელიც მიიღება FROM, WHERE, GROUP BY და ORDER BY განყოფილებების დამუშავების შემდეგ.

OVER ელემენტის გამოყენება ნებადართულია მხოლოდ SELECT და ORDER BY განყოფილებებში. მისი გამოყენება განვიხილოთ SELECT სტადიაზე. მაგალითად, თუ გამოვიყენებთ SELECT xelshekrulebaID, shemkvetiID, gadasaxdeli_l, SUM(gadasaxdeli_l) OVER() გამოსახულებას, რომელიც Xelshekruleba ცხრილს მიმართავს, მაშინ ფუნქცია შედეგობრივ მნიშვნელობას გამოთვლის ყველა სტრიქონისთვის, რომელიც SELECT სტადიაზე მუშავდება. თუ მოთხოვნაში მონაცემები არ იფილტრება, ან SELECT სტადიის წინ არ გამოიყენება ლოგიკური დამუშავების სხვა სტადიები, მაშინ გამოსახულება დაგვიბრუნებს Xelshekruleba ცხრილის ყველა სტრიქონის შედეგობრივ მნიშვნელობას. მოყვანილ მოთხოვნაში გამოითვლება gadasaxdeli_l სვეტის მნიშვნელობების ჯამი და შედეგი მოთავსდება saertotanxa სვეტში. მოთხოვნას აქვს სახე:

```
USE Shekveta;
SELECT xelshekrulebaID, shemkvetiID, gadasaxdeli_l, SUM(gadasaxdeli_l) OVER() AS saertotanxa
FROM Xelshekruleba;
```

შედეგი:

	xelshekrulebaID	shemkvetiID	gadasaxdeli_l	saertotanxa
1	17	NULL	5300	118500
2	18	12	4400	118500
3	19	13	3500	118500
4	20	14	8500	118500
5	21	15	5600	118500
6	1	1	5000	118500
7	2	2	3700	118500
8	3	2	8000	118500
9	4	5	1900	118500
10	5	6	2500	118500
11	6	3	8200	118500
12	7	1	7700	118500
13	8	4	8000	118500
14	9	3	6000	118500
15	10	2	9800	118500
16	11	3	7100	118500
17	12	1	3000	118500
18	13	5	10000	118500
19	14	1	2000	118500
20	15	7	4400	118500
21	16	7	3900	118500

თუ გვინდა შევზღუდოთ ან დავყოთ სტრიქონები სექციებად შეგვიძლია გამოვიყენოთ PARTITION BY ელემენტი. მაგალითად, Xelshekruleba ცხრილის ყველა სტრიქონისათვის შედეგობრივი მნიშვნელობის მიღების ნაცვლად, თუ გვინდა მივიღოთ შედეგობრივი მნიშვნელობა თითოეული შემკვეთისთვის, უნდა გამოვიყენოთ SUM(gadasaxdeli_l) OVER (PARTITION BY shemkvetiID) გამოსახულება.

ქვემოთ მოყვანილია მოთხოვნა, რომელიც ახდენს როგორც სექციონირებული, ისე არასექციონირებული გამოსახულებების დემონსტრირებას და გასცემს Xelshekruleba ცხრილის ყველა სტრიქონს. მოთხოვნა გასცემს საერთო შედეგობრივ მნიშვნელობას და შედეგობრივ მნიშვნელობას თითოეული შემკვეთისთვის:

```
USE Shekveta;
SELECT xelshekrulebaID, shemkvetiID, gadasaxdeli_l,
SUM(gadasaxdeli_l) OVER() AS saertotanxa,
SUM(gadasaxdeli_l) OVER(PARTITION BY shemkvetiid) AS shemkvetisaertotanxa
FROM Xelshekruleba;
```

მოცემული მოთხოვნა გვიბრუნებს შედეგს:

	xelshekrulebaID	shemkvetiID	gadasaxdeli_l	saertotanxa	shemkvetisaertotanxa
1	17	NULL	5300	118500	5300
2	1	1	5000	118500	17700
3	7	1	7700	118500	17700
4	12	1	3000	118500	17700
5	14	1	2000	118500	17700
6	10	2	9800	118500	21500
7	2	2	3700	118500	21500
8	3	2	8000	118500	21500
9	11	3	7100	118500	21300
10	6	3	8200	118500	21300
11	9	3	6000	118500	21300
12	8	4	8000	118500	8000
13	13	5	10000	118500	11900
14	4	5	1900	118500	11900
15	5	6	2500	118500	2500
16	15	7	4400	118500	8300
17	16	7	3900	118500	8300
18	18	12	4400	118500	4400
19	19	13	3500	118500	3500
20	20	14	8500	118500	8500
21	21	15	5600	118500	5600

. . .

saertotanxa სვეტი შედეგობრივი ნაკრების თითოეულ სტრიქონში შეიცავს შეკვეთების საერთო ღირებულებას, რომელიც ყველა სტრიქონშია მოყვანილი. shemkvetisaertotanxa სვეტში გამოჩნდება შედეგობრივი მნიშვნელობა ყველა სტრიქონისთვის, რომლებსაც აქვთ shemkvetiID სვეტის ერთნაირი მნიშვნელობა.

OVER ელემენტის ერთ-ერთი უპირატესობა იმაში მდგომარეობს, რომ ის იძლევა ერთ სტრიქონში საბაზო სვეტებისა და მათი შედეგობრივი მნიშვნელობების გამოტანის საშუალებას. ის, ასევე საშუალებას გვაძლევს, განვსაზღვროთ შესაბამისი გამოსახულებები. მაგალითად, მოყვანილ მოთხოვნაში Xelshekruleba ცხრილის თითოეული სტრიქონისთვის გამოითვლება შეკვეთის პროცენტული წილი საერთო შედეგში (procentisaerto სვეტი) და შეკვეთის მიმდინარე ღირებულების პროცენტული წილი შეკვეთების შედეგობრივ ღირებულებაში მოცემული შემკვეთისთვის (procentishemkveti სვეტი). მოთხოვნას აქვს სახე:

```
USE Shekveta;
SELECT xelshekrulebaID, shemkvetiID, gadasaxdeli_l,
100. * gadasaxdeli_l / SUM(gadasaxdeli_l) OVER() AS procentisaerto,
100. * gadasaxdeli_l / SUM(gadasaxdeli_l) OVER(PARTITION BY shemkvetiID) AS procentishemkveti
FROM Xelshekruleba;
```

გამოსახულებაში მითითებულია წილადი - 100. მთელი 100 მნიშვნელობის ნაცვლად. საქმე ის არის, რომ როცა მთელი რიცხვი იყოფა მთელზე, შედეგი იქნება მთელი რიცხვი, რადგან წილადი ნაწილი გადაიგდება. იმისათვის, რომ წილადი შედეგი მივიღოთ, საჭიროა, რომ გასაყოფი ან გამყოფი წილადი იყოს.

მოცემული მოთხოვნა გვიბრუნებს შედეგს:

	xelshekrulebaID	shemkvetiID	gadasaxdeli_I	procentisaerto	procentishemkveti
1	17	NULL	5300	4,47257383966245	100
2	1	1	5000	4,21940928270042	28,2485875706215
3	7	1	7700	6,49789029535865	43,5028248587571
4	12	1	3000	2,53164556962025	16,9491525423729
5	14	1	2000	1,68776371308017	11,2994350282486
6	10	2	9800	8,27004219409283	45,5813953488372
7	2	2	3700	3,12236286919831	17,2093023255814
8	3	2	8000	6,75105485232068	37,2093023255814
9	11	3	7100	5,9915611814346	33,3333333333333
10	6	3	8200	6,91983122362869	38,4976525821596
11	9	3	6000	5,06329113924051	28,169014084507
12	8	4	8000	6,75105485232068	100
13	13	5	10000	8,43881856540084	84,0336134453782
14	4	5	1900	1,60337552742616	15,9663865546218
15	5	6	2500	2,10970464135021	100
16	15	7	4400	3,71308016877637	53,0120481927711
17	16	7	3900	3,29113924050633	46,9879518072289
18	18	12	4400	3,71308016877637	100
19	19	13	3500	2,9535864978903	100
20	20	14	8500	7,17299578059072	100
21	21	15	5600	4,72573839662447	100

NULL მნიშვნელობასთან მუშაობა

NULL მნიშვნელობა გამოიყენება გამოტოვებული მნიშვნელობებისა და მნიშვნელობების არარსებობის აღსანიშნავად. T-SQL ენა იყენებს სამობით ლოგიკას. ეს იმას ნიშნავს, რომ პრედიკატებმა შეიძლება სამიდან ერთ-ერთი მნიშვნელობა მიიღოს: TRUE (ჭეშმარიტი), FALSE (მცდარი) და UNKNOWN (უცნობი). ლოგიკური გამოსახულება, რომელიც კონკრეტული მნიშვნელობებისთვის მოწმდება, იღებს TRUE (ჭეშმარიტი), ან FALSE (მცდარი) მნიშვნელობას. მაგრამ, თუ ლოგიკური გამოსახულება მოწმდება გამოტოვებული მნიშვნელობისთვის (NULL, მნიშვნელობის არარსებობა), მაშინ ის იღებს UNKNOWN (უცნობი) მნიშვნელობას. განვიხილოთ პრედიკატი xelfasi > 0. თუ ხელფასი 500-ის ტოლია, ლოგიკური გამოსახულება იღებს TRUE მნიშნელობას, თუ ხელფასი -500-ის ტოლია, მაშინ ლოგიკური გამოსახულება იღებს FALSE მნიშვნელობას. თუ ხელფასი NULL-ის ტოლია, მაშინ ლოგიკური გამოსახულება იღებს UNKNOWN მნიშვნელობას.

თუ xelfasi > 0 პრედიკატი გამოიყენება მოთხოვნა-ფილტრში (WHERE და HAVING განყოფილებებში), მაშინ გვიბრუნდება სტრიქონები ან ჯგუფები, რომლებისთვისაც ლოგიკური გამოსახულება TRUE მნიშვნელობას იღებს, ხოლო ის სტრიქონები და ჯგუფები, რომლებისთვისაც ლოგიკური გამოსახულება FALSE მნიშვნელობას იღებს, უარიყოფა.

T-SQL ენის სხვადასხვა სინტაქსურ ელემენტში UNKNOWN მნიშვნელობა სხვადასხვანაირად აღიქმება. გამფილტრავი მოთხოვნებისთვის „TRUE-ს მიღება“ ნიშნავს FALSE და UNKNOWN მნიშვნელობის უარყოფას. მეორეს მხრივ, CHECK ტიპის შეზღუდვისთვის „FALSE-ის უარყოფა“ ნიშნავს TRUE და UNKNOWN მნიშვნელობების მიიღებას. პრედიკატების სამობით ლოგიკაში „TRUE-ს მიღება“ უარყოფს FALSE და UNKNOWN მნიშვნელობებს, ხოლო „FALSE-ის უარყოფა“ ნიშნავს TRUE და UNKNOWN მნიშვნელობებს.

xelfasi > 0 პრედიკატის გამოყენების შემთხვევაში თუ ხელფასი NULL მნიშვნელობის ტოლია, მაშინ გამოსახულების მნიშვნელობა UNKNOWN იქნება. თუ ეს პრედიკატი გვხვდება ცხრილის CHECK შეზღუდვაში, მაშინ სტრიქონი NULL-ის ტოლი ხელფასით იქნება მიღებული.

UNKNOWN მნიშვნელობის ერთ-ერთი თავისებურებაა ის, რომ მის მიმართ NOT (არა) ოპერაციის გამოყენება ისევ UNKNOWN მნიშვნელობას გვაძლევს. მაგალითად, თუ მოცემულია პრედიკატი NOT (xelfasi > 0) და ხელფასი NULL-ის ტოლია, მაშინ xelfasi > 0 გამოსახულება გასცემს UNKNOWN მნიშვნელობას და NOT UNKNOWN დარჩება UNKNOWN მნიშვნელობის ტოლი.

ორი NULL მნიშვნელობის შემდარებელი გამოსახულება (NULL = NULL) გასცემს UNKNOWN მნიშვნელობას. საქმე ის არის, რომ NULL აღნიშნავს გამოტოვებულ ან უცნობ მნიშვნელობას. ჩვენ კი არ შეგვიძლია დავადგინოთ, ერთი უცნობი მნიშვნელობა მეორის ტოლია თუ არა. ამიტომ, = NULL და <> NULL შედარებების ნაცვლად უნდა გამოვიყენოთ IS NULL და IS NOT NULL პრედიკატები.

Shemkveti ცხრილში არის regioni, raioni და qalaqi სვეტები, რომლებშიც ინახება ინფორმაცია შემკვეთის ადგილმდებარეობის შესახებ. მოყვანილმა მოთხოვნამ უნდა დაგვიბრუნოს ყველა შემკვეთი, რომლებისთვისაც რეგიონი 'იმერეთი'-ის ტოლია:

```
USE Shekveta;
SELECT shemkvetiID, regioni, qalaqi
FROM Shemkveti
WHERE regioni = N'იმერეთი';
```

შედეგი:

	shemkvetiID	raioni	qalaqi
1	4	იმერეთი	ქუთაისი
2	7	იმერეთი	ზესტაფონი

Shemkveti ცხრილის სტრიქონებიდან მოთხოვნა გვიბრუნებს ორ სტრიქონს, რომლებშიც regioni სვეტი 'იმერეთი'-ის ტოლია. მოთხოვნა არ გვიბრუნებს სტრიქონებს, რომლებშიც regioni სვეტის მნიშვნელობა არსებობს და განსხვავდება 'იმერეთი' მნიშვნელობისაგან (პრედიკატი იღებს FALSE მნიშვნელობას), და სტრიქონებს, რომლებშიც regioni სვეტი NULL მნიშვნელობის ტოლია (პრედიკატი იღებს UNKNOWN მნიშვნელობას).

მოყვანილი მოთხოვნა ცდილობს, ამოირჩიოს ყველა კლიენტი, რომლისთვისაც რეგიონი განსხვავდება 'იმერეთი' მნიშვნელობისაგან:

```
SELECT shemkvetiID, regioni, qalaqi
FROM Shemkveti
WHERE regioni <> N'იმერეთი';
```

შედეგი:

	shemkvetiID	regioni	qalaqi
1	2	კახეთი	თელავი
2	3	კახეთი	თელავი
3	5	აჭარა	ბათუმი
4	6	აჭარა	ბათუმი
5	8	სამეგრელო	ზუგდიდი
6	9	ქართლი	გორი
7	16	კახეთი	თელავი

როგორც ვხედავთ, 12 სტრიქონის (ცხრილის 14 სტრიქონს მინუს 2 სტრიქონი) ნაცვლად მივიღეთ 7 სტრიქონი. ეს იმიტომ მოხდა, რომ დაგვიბრუნდა ის სტრიქონები, რომლებშიც raioni (რეგიონი) სვეტს ჰქონდა 'იმერეთი'-სგან განსხვავებული მნიშვნელობები. არ დაგვიბრუნდა არც ერთი სტრიქონი, რომლებშიც რეგიონი 'იმერეთი'-ის ტოლი იყო და არც ერთი სტრიქონი, სადაც რეგიონი NULL-ის ტოლი იყო. ჩვენ იგივე შედეგს მივიღებთ, თუ გამოვიყენებთ NOT (regioni = N'იმერეთი') პრედიკატს, რადგან სტრიქონებში, სადაც რეგიონი NULL-ის ტოლია, regioni = 'იმერეთი' ლოგიკური გამოსახულება გასცემს UNKNOWN მნიშვნელობას და NOT (regioni = 'იმერეთი') ლოგიკური გამოსახულებაც გასცემს UNKNOWN მნიშვნელობას.

თუ გვინდა მივიღოთ ყველა სტრიქონი, რომლებშიც regioni სვეტის მნიშვნელობა არის NULL-ის ტოლი, არ უნდა გამოვიყენოთ regioni = NULL პრედიკატი, რადგან გამოსახულების მნიშვნელობა UNKNOWN-ის ტოლი იქნება ყველა სტრიქონში, და იმ სტრიქონებში, სადაც მნიშვნელობა არის, და იმ სტრიქონებშიც, სადაც მნიშვნელობა გამოტოვებულია (NULL-ის ტოლია). ქვემოთ მოყვანილი მოთხოვნა დაგვიბრუნებს ცარიელ სიმრავლეს:

```
SELECT shemkvetiID, regioni, qalaqi
FROM Shemkveti
WHERE regioni = NULL;
```

raioni = NULL ლოგიკური გამოსახულების ნაცვლად, უნდა გამოვიყენოთ raioni IS NULL პრედიკატი:

```
SELECT shemkvetiID, regioni, qalaqi
FROM   Shemkveti
WHERE regioni IS NULL;
```

შედეგი:

	shemkvetiid	regioni	qalaqi
1	1	NULL	თბილისი
2	10	NULL	თბილისი
3	11	NULL	თბილისი
4	12	NULL	თბილისი
5	14	NULL	თბილისი
6	15	NULL	თბილისი

თუ ჩვენ გვინდა ყველა სტრიქონის მიღება, რომლებშიც regioni სვეტი არ უდრის 'იმერეთი'-ს, მაშინ მოთხოვნა ასე უნდა ჩავწეროთ:

```
SELECT shemkvetiID, regioni, qalaqi
FROM Shemkveti
WHERE regioni <> N'იმერეთი' OR regioni IS NULL;
```

შედეგი:

	shemkvetiid	regioni	qalaqi
1	1	NULL	თბილისი
2	2	კახეთი	თელავი
3	3	კახეთი	თელავი
4	5	აჭარა	ბათუმი
5	6	აჭარა	ბათუმი
6	8	სამეგრელო	ზუგდიდი
7	9	ქართლი	გორი
8	10	NULL	თბილისი
9	11	NULL	თბილისი
10	12	NULL	თბილისი
11	13	კახეთი	თელავი
12	14	NULL	თბილისი
13	15	NULL	თბილისი

T-SQL ენა სხვადასხვანაირად ახდენს NULL მნიშვნელობის ინტერპრეტირებას მის სხვადასხვა სინტაქსურ ელემენტში, რომლებიც განკუთვნილია შედარებისა და დახარისხებისთვის. ზოგიერთი ელემენტი ორ NULL მნიშვნელობას ტოლად თვლის, ზოგი - განსხვავებულად.

მაგალითად, დაჯგუფებისა და დახარისხების დროს ორი NULL მნიშვნელობა ითვლება ტოლად. ეს იმას, ნიშნავს, რომ GROUP BY ელემენტი აგროვებს ყველა NULL მნიშვნელობას ერთ ჯგუფში, ხოლო ORDER BY ელემენტი ერთად ახარისხებს ყველა NULL მნიშვნელობას. T-SQL ენა დახარისხებულ ნაკრებში NULL მნიშვნელობებს ათავსებს სხვების წინ, რომლებსაც აქვთ მნიშვნელობები.

UNIQUE შეზღუდვის გამოყენების შემთხვევაში სვეტებში დასაშვებია მხოლოდ ერთი NULL მნიშვნელობა.

ერთდროულად შესრულებადი ოპერაციები

T-SQL ენა უზრუნველყოფს ერთდროულად შესრულებად ოპერაციებს. ეს იმას ნიშნავს, რომ ყველა გამოსახულება, რომლებიც გამოჩნდება ლოგიკური დამუშავების ერთ სტადიაზე, თითქოს გამოითვლება დროის ერთსა და იმავე მომენტში. ამით აიხსნება ის ფაქტი, თუ რატომ არ შეიძლება SELECT ელემენტში მივმართოთ ცხრილების ფსევდონიმებს, რომლებიც მინიჭებულია იმავე SELECT ელემენტში. განვიხილოთ შემდეგი მოთხოვნა:

```
SELECT xelshekrulebaID, YEAR(tarigi_dawyebis) AS xelshyear, xelshyear + 1 AS nextyear
FROM Xelshekruleba;
```

SELECT განყოფილების სიის მესამე გამოსახულებაში მიმართვა xelshyear (შეკვეთის წელი) სვეტის ფსევდონიმზე დაუშვებელია, მიუხედავად იმისა, რომ გამოსახულება რომელიც ამ მიმართვას შეიცავს, გამოჩნდება იმ გამოსახულების „შემდეგ“, რომელშიც ხდება ფსევდონიმის მინიჭება. მიზეზი ის არის, რომ არ არსებობს SELECT სიაში გამოთვლების ლოგიკური მიმდევრობა. ის წარმოადგენს გამოსახულებების სიმრავლეს. ლოგიკურ დონეზე SELECT განყოფილების სიის ყველა გამოსახულება გამოითვლება ერთდროულად. ამიტომ, მოთხოვნის შესრულების შედეგად გაიცემა შეტყობინება შეცდომის შესახებ:

```
Msg 207, Level 16, State 1, Line 4
Invalid column name 'xelshyear'.
```

განვიხილოთ კიდევ ერთი მაგალითი. დავუშვათ, გვინდა მივიღოთ ყველა სტრიქონი, რომლებშიც sveti2 / sveti1 არის 5-ზე მეტი. რადგან, ცხრილში შეიძლება იყოს სტრიქონები,

რომლებისთვისაც sveti1 მნიშვნელობა 0-ის ტოლია, ჩვენ უნდა დავრწმუნდეთ იმაში, რომ ამ სტრიქონებში მსგავსი გაყოფა არ შესრულდება - საწინააღმდეგო შემთხვევაში, მოთხოვნა ავარიულად დამთავრდება ნულზე გაყოფის გამო. მოთხოვნას შეიძლება ასეთი სახე ჰქონდეს:

```
SELECT sveti1, sveti2
FROM dbo.Cxrili1
WHERE sveti1 <> 0 AND sveti2 / sveti1 > 5;
```

SQL სერვერი გამოსახულებას მარცხნიდან მარჯვნივ გამოთვლის. ამიტომ, თუ sveti1 <> 0 გამოსახულების მნიშვნელობა FALSE-ის ტოლია, მაშინ SQL სერვერი არ გამოთვლის sveti2 / sveti1 > 5 გამოსახულებას, რადგან ამ მომენტისთვის უკვე ცნობილია, რომ მთელი გამოსახულება FALSE-ის ტოლია.

მონაცემების მასობრივი გადაწერა

მონაცემების იმპორტისა და ექსპორტისათვის BCP (Bulk Copy Program) უტილიტი გამოიყენება. ის ასრულებს მონაცემების გაცვლას სერვერსა და ტექსტურ ფაილებს შორის. კერძოდ, ცხრილის სტრიქონებს წერს ტექსტურ ფაილში და პირიქით. BCP უტილიტის მუშაობის შედეგები არ აისახება ტრანზაქციის ჟურნალში. ეს იმას ნიშნავს, რომ თუ გადაწერის ოპერაცია წარუმატებლად დამთავრდა, მაშინ საწყისი მონაცემების აღდგენას ვერ შევძლებთ. ამ უტილიტის გამოყენების ერთი სფეროა ძველი პროგრამა-დანართების გამოსასვლელი ტექსტური მონაცემების გადატანა სერვერზე. BCP უტილიტი მუშაობს ავტომატურ და ინტერაქტიურ რეჟიმში. თუ დაფორმატების გასაღებები მითითებულია საბრძანებო სტრიქონში ან დაფორმატების ფაილში, მაშინ BCP უტილიტი ავტომატურ რეჟიმში იმუშავებს. თუ დაფორმატების პარამეტრები მითითებული არ არის, მაშინ გაიცემა შესაბამისი შეკითხვები. BCP ბრძანების შესასრულებლად ის უნდა შევიტანოთ Command Prompt ფანჯრის (Start→All Programs→Accessories→Command Prompt) საბრძანებო სტრიქონში და დავაჭიროთ Enter კლავიშს.

BCP ბრძანების სინტაქსია:

```
bcp { [ [ მონაცემთა_ბაზის_სახელი.] [ მფლობელი ] . ]
{ ცხრილის_სახელი | წარმოდგენის_სახელი } | 'მოთხოვნა' }
{ in | out | queryout | format } მონაცემების_ფაილი
[ -F პირველი_სტრიქონი ] [ -L უკანასკნელი_სტრიქონი ]
[ -w ] [ -N ]
[ -q ] [ -t სვეტების_გამყოფი ] [ -r სტრიქონების_გამყოფი]
[ -S სერვერის_სახელი ] [ -U სააღრიცხვო_ჩანაწერის_სახელი ] [ -P პაროლი ]
[ -T ] [ -R ] [ -k ]
```

განვიხილოთ არგუმენტების დანიშნულება.

– *მონაცემთა_ბაზის_სახელი*. იმ მონაცემთა ბაზის სახელია, რომელშიც ის ცხრილი ან წარმოდგენა ინახება, საიდანაც უნდა შესრულედეს ასლის აღება. თუ სახელი მითითებული არ არის, მაშინ გამოყენებულია ის მონაცემთა ბაზა, რომელიც ეკუთვნის იმ მომხმარებელს, რომლის სააღრიცხვო ჩანაწერით მუშაობს ეს უტილიტი.

– *მფლობელი*. იმ ცხრილის ან წარმოდგენის მფლობელის სახელია, რომელთანაც სრულდება მუშაობა. თუ ის მითითებული არ არის, მაშინ ცხრილი ან წარმოდგენა ეკუთვნის იმ ფლობელს, რომლის სააღრიცხვო ჩანაწერით მუშაობს ეს უტილიტი.

– *ცხრილის_სახელი* | *წარმოდგენის_სახელი* } | '*მოთხოვნა*'. ცხრილის ან წარმოდგენის სახელია, რომელთანაც შესრულდება მუშაობა. *მოთხოვნა* არგუმენტი შეიცავს SELECT-მოთხოვნას.

– in | out | queryout | format. ინფორმაციის გადაცემის მიმართულებაა. თუ მითითებულია in, მაშინ მონაცემების გადაწერა სრულდება ტექსტური ფაილიდან მონაცემთა ბაზის ცხრილში. თუ მითითებულია out, მაშინ მონაცემების გადაწერა სრულდება მონაცემთა ბაზის ცხრილიდან ტექსტურ ფაილში. თუ მონაცემების წყაროდ მითითებულია *მოთხოვნა*, მაშინ უნდა მივუთითოთ queryout არგუმენტი. თუ მითითებულია format გასაღები, მაშინ შეიქმნება -n, -c, -w, -6 ან -N პარამეტრების მნიშვნელობების მიხედვით დაფორმატებული ფაილი.
– მონაცემების_ფაილი. სრული გზაა ფაილისკენ. თუ მონაცემების გადაწერა ხდება სერვერზე, მაშინ ამ ფაილში მოთავსებული იქნება საწყისი მონაცემები. თუ მონაცემების გადაწერა ხდება სერვერიდან, მაშინ ისინი ამ ფაილში ჩაიწერება. სრული გზა არ უნდა შეიცავდეს 255 სიმბოლოზე მეტს.
– -F პირველი_სტრიქონი. პირველი სტრიქონის ნომერია, საიდანაც დაიწყება გადაწერა. თუ ის მითითებული არ არის, მაშინ გადაწერა პირველი სტრიქონიდან დაიწყება.
– -L უკანასკნელი_სტრიქონი. უკანასკნელი სტრიქონის ნომერია, რომელიც გადაწერილი იქნება. ავტომატურად აიღება 0, რაც იმას ნიშნავს, რომ გადაიწერება ყველა სტრიქონი დამთავრებული უკანასკნელით.
– -w. მიუთითებს, რომ მონაცემები უნდა დამუშავდეს Unicode ფორმატში nchar ტიპის გამოყენებით. მიღებული ფაილი შეგვიძლია ვნახოთ ტექსტური რედაქტორის გამოყენებით.
– -N. მიუთითებს, რომ არასიმბოლური მონაცემებისთვის უნდა გამოვიყენოთ “ძველი” ფორმატი, სიმბოლური მონაცემებისთვის კი - Unicode ფორმატი. -N გასაღები -w გასაღებთან შედარებით ზრდის გადაწერის ოპერაციის მწარმოებლურობას. ნაკლია ის, რომ ტექსტური რედაქტორით ვერ ვნახავთ ფაილში ჩაწერილ მონაცემებს. ეს გასაღები ხშირად გამოიყენება სერვერებს შორის მონაცემების გადატანისას.
– -q. მიუთითებს, რომ ცხრილების ან წარმოდგენების სახელები დაუშვებელ სიმბოლოებს შეიცავს. ასეთი სახელები ბრჭყალებში უნდა მოვათავსოთ.
– -t სვეტების_გამყოფი. მიუთითებს სვეტების გამყოფს ტექსტურ ფაილში. ავტომატურად იგულისხმება \t სიმბოლო.
– -r სტრიქონების_გამყოფი. მიუთითებს სტრიქონების გამყოფს ტექსტურ ფაილში. ავტომატურად იგულისხმება \n სიმბოლო.
– -S სერვერის_სახელი. მიუთითებს სერვერის სახელს, რომელსაც მიუერთდება bcp უტილიტი. ავტომატურად კავშირი მყარდება ლოკალურ სერვერთან. მისი მითითება საჭიროა მაშინ, როცა ვუკავშირდებით დაშორებულ სერვერს.
– -U სააღრიცხვო_ჩანაწერის_სახელი. მომხმარებლის სახელია, რომლის სააღრიცხვო ჩანაწერით იმუშავებს bcp უტილიტი.
– -P პაროლი. სააღრიცხვო ჩანაწერის პაროლია, რომელიც გამოიყენება სერვერთან კავშირის დასამყარებლად. ცარიელი პაროლის შემთხვევაში უნდა მივუთითოთ მხოლოდ -P.
– -T. მიუთითებს, რომ შეერთების დასამყარებლად გამოყენებული უნდა იყოს Windows NT სააღრიცხვო ჩანაწერი. ამასთან, -U და -P იგნორირდება.
– -k. მიუთითებს, რომ ცხრილში ცარიელი სტრიქონების ჩასმის დროს გამოყენებული უნდა იყოს NULL მნიშვნელობები და არა ავტომატური.

მაგალითები.

ა. Shekveta მონაცემთა ბაზის Personali ცხრილიდან ტექსტურ ფაილში გადავწეროთ ყველა სტრიქონი. მონაცემები უნდა დამუშავდეს Unicode ფორმატში. გამოვიყენოთ Windows NT-ის სააღრიცხვო ჩანაწერი.

```
bcp "Shekveta.dbo.Personali"
out "C:\Users\Romani\Documents\SQL Server Management Studio\Projects\BCP\Personali1.txt"
-w -T
```

ბ. Shekveta მონაცემთა ბაზის Personali ცხრილიდან ტექსტურ ფაილში გადავწეროთ ყველა სტრიქონი. გამოვიყენოთ Windows NT-ის სააღრიცხვო ჩანაწერი. სერვერის სახელია Romani-PC.

```
bcp "Shekveta.dbo.Personali"
out "C:\Users\Romani\Documents\SQL Server Management Studio\Projects\BCP\Personali2.txt"
-w -S"Romani-PC" -T
```

გ. Shekveta მონაცემთა ბაზის Personali ცხრილიდან ტექსტურ ფაილში გადავწეროთ ყველა სტრიქონი. მივუთითოთ მომხმარებლის სახელი და პაროლი. სერვერის სახელია Romani-PC.

```
bcp "Shekveta.dbo.Personali"
out "C:\Users\Romani\Documents\SQL Server Management Studio\Projects\BCP\Personali3.txt"
-w -S"Romani-PC" –U"sa" –P"paroli"
```

დ. Shekveta მონაცემთა ბაზის Personali ცხრილიდან ტექსტურ ფაილში გადავწეროთ ყველა სტრიქონი. გამოვიყენოთ SELECT მოთხოვნა და Windows NT-ის სააღრიცხვო ჩანაწერი.

```
bcp "SELECT * FROM Shekveta.dbo.Personali"
queryout "C:\Users\Romani\Documents\SQL Server Management Studio\Projects\BCP\Personali4.txt"
-w -T
```

ე. Shekveta მონაცემთა ბაზის Personali ცხრილიდან ტექსტურ ფაილში გადავწეროთ ყველა ის სტრიქონი, სადაც asaki სვეტის მნიშვნელობა 40-ს აღემატება. სტრიქონები ზრდადობით დავალაგოთ asaki სვეტის მიხედვით. გამოვიყენოთ SELECT მოთხოვნა. მივუთითოთ სერვერის სახელი და პაროლი. სერვერის სახელია Romani-PC.

```
bcp "SELECT * FROM Shekveta.dbo.Personali WHERE asaki > 40 ORDER BY asaki"
queryout "C:\Users\Romani\Documents\SQL Server Management Studio\Projects\BCP\Personali5.txt"
-w -S"Romani-PC" –U"sa" –P"paroli"
```

ვ. შევასრულოთ Personali1.txt ტექსტური ფაილიდან სტრიქონების ჩასმა Shekveta.dbo.Pers_1 ცხრილში. ამისათვის, ჯერ უნდა შევქმნათ Pers_1 ცხრილი, შემდეგ კი შევასრულოთ სტრიქონების ჩასმა. გამოვიყენოთ Windows NT-ის სააღრიცხვო ჩანაწერი.

```
bcp "Shekveta.dbo.Pers_1"
in "C:\Users\Romani\Documents\SQL Server Management Studio\Projects\BCP\Personali1.txt"
-w -T
```

ზ. შევასრულოთ Personali2.txt ტექსტური ფაილიდან სტრიქონების ჩასმა Shekveta.dbo.Pers_1 ცხრილში. ამისათვის, ჯერ უნდა შევქმნათ Pers_1 ცხრილი, შემდეგ კი შევასრულოთ სტრიქონების ჩასმა. მივუთითოთ სერვერის სახელი და პაროლი. სერვერის სახელია Romani-PC.

```
bcp "Shekveta.dbo.Pers_1"
in "C:\Users\Romani\Documents\SQL Server Management Studio\Projects\BCP\Personali2.txt"
-w –S"Romani-PC" –U"sa" –P"paroli"
```

თავი 5. შეერთებები

FROM განყოფილება

როგორც აღვნიშნეთ, SELECT მოთხოვნის FROM განყოფილება (ელემენტი) ლოგიკურად პირველი მუშავდება. ამ განყოფილებაში შესასვლელი (საწყისი) ცხრილების მიმართ JOIN, APPLY, PIVOT და UNPIVOT ცხრილური ოპერაციები გამოიყენება. JOIN ცხრილური ოპერაცია სტანდარტულია, ხოლო APPLY, PIVOT და UNPIVOT ცხრილური ოპერაციები კი - T-SQL ენის სტანდარტის გაფართოება.

არსებობს შეერთების სამი ძირითადი ტიპი: ჯვარედინი, შიგა და გარე. შეერთების სამივე ტიპი ერთმანეთისაგან მოთხოვნის ლოგიკური დამუშავების სტადიებით განსხვავდება და სტადიების სხვადასხვა შემადგენლობა აქვს. ჯვარედინი შეერთება დამუშავების მხოლოდ ერთ სტადიას მოიცავს - დეკარტულ ნამრავლს (წარმოებულს). შიგა შეერთება დამუშავების ორ სტადიას მოიცავს - დეკარტულ ნამრავლს და გაფილტვრას. გარე შეერთება დამუშავების სამ სტადია მოიცავს - დეკარტულ ნამრავლს, გაფილტვრასა და გარე სტრიქონების დამატებას.

FROM განყოფილების სინტაქსია:

FROM { *ცხრილის_სახელი* [[AS] *ცხრილის_ფსევდონიმი*]
| *წარმოდგენის_სახელი* [[AS] *წარმოდგენის_ფსევდონიმი*] |
<*მრული_ცხრილი*> }

აქ <*მრული_ცხრილი*> კონსტრუქცია გამოიყენება რამდენიმე ცხრილს შორის კავშირის უზრუნველსაყოფად. მისი სინტაქსია:

<*მრული_ცხრილი*> ::=
<*მარცხენა_ცხრილი*> <*მის_ტიპი*> <*მარჯვენა_ცხრილი*> ON <*მის_პირობა*>
| <*მარცხენა_ცხრილი*> CROSS JOIN <*მარჯვენა_ცხრილი*>
| <*მრული_ცხრილი*>

აქ <*მარცხენა_ცხრილი*> კონსტრუქცია შეიცავს მთავარი ცხრილის სახელს, რომელსაც დაუკავშირდება სხვა ცხრილები. <*მის_ტიპი*> კონსტრუქცია განსაზღვრავს ორ ცხრილს შორის კავშირის ტიპს. მთავარი ცხრილის სახელი ეთითება <*მის_ტიპი*> კონსტრუქციის მარცხნივ (მას მარცხენა ცხრილი ეწოდება - left table), მარჯვნივ კი ეთითება დამოკიდებული ცხრილის სახელი (მას მარჯვენა ცხრილი ეწოდება - right table). <მის_ტიპი> კონსტრუქციის სინტაქსია:

<*მის_ტიპი*> ::= [INNER | { { LEFT | RIGHT | FULL } [OUTER] }] JOIN

განვიხილოთ არგუმენტების დანიშნულება.

- INNER კავშირის ტიპი ავტომატურად გამოიყენება. შედეგში (სტრიქონების შედეგობრივ სიმრავლეში) ჩართული იქნება მარცხენა ცხრილის მხოლოდ ის სტრიქონები, რომლებისთვისაც არსებობს სტრიქონები ბმულ (მარჯვენა) ცხრილში. შედეგში ჩართული იქნება, აგრეთვე, მარჯვენა ცხრილის მხოლოდ ის სტრიქონები, რომლებისთვისაც არსებობს შესაბამისი სტრიქონები მარცხენა ცხრილში.
- LEFT [OUTER] არგუმენტის გამოყენება უზრუნველყოფს შედეგში მარცხენა ცხრილის ყველა სტრიქონის ჩართვას. თუ მარცხენა ცხრილის რომელიმე სტრიქონისთვის არ არსებობს შესაბამისი სტრიქონები მარჯვენა ცხრილში, მაშინ მარჯვენა ცხრილის შესაბამის სვეტებს NULL მნიშვნელობა ექნება.
- RIGHT [OUTER] არგუმენტის გამოყენება უზრუნველყოფს შედეგში მარჯვენა ცხრილის ყველა სტრიქონის ჩართვას. თუ მარჯვენა ცხრილის რომელიმე სტრიქონისთვის არ არსებობს შესაბამისი სტრიქონები მარცხენა ცხრილში, მაშინ მარცხენა ცხრილის შესაბამის სვეტებს NULL მნიშვნელობა ექნება.

- FULL [OUTER] არგუმენტის გამოყენება უზრუნველყოფს შედეგში მარცხენა და მარჯვენა ცხრილების ყველა სტრიქონის ჩართვას.
- JOIN არგუმენტის შემდეგ ეთითება მარჯვენა ცხრილი.
- ON <*გმის_პირობა*> არის ორივე ცხრილის დაკავშირების ლოგიკური პირობა.
- CROSS JOIN საკვანძო სიტყვის გამოყენებისას სრულდება მარცხენა ცხრილის თითოეული სტრიქონის დაკავშირება მარჯვენა ცხრილის თითოეულ სტრიქონთან. თუ ცხრილებს შორის კავშირი არ არის მითითებული, მაშინ ავტომატურად სრულდება მარცხენა ცხრილის თითოეული სტრიქონის დაკავშირება მარჯვენა ცხრილის თითოეულ სტრიქონთან.

ჯვარედინი შეერთებები

ჯვარედინი შეერთება აფორმირებს ორი ცხრილის სტრიქონების დეკარტულ ნამრავლს. ერთი ცხრილის თითოეულ სტრიქონს შეეთანადება მეორე ცხრილის ყველა სტრიქონი. თუ ერთ ცხრილში გვაქვს m სტრიქონი და მეორეში - n სტრიქონი, მაშინ სტრიქონების შედეგობრივ ნაკრებში m x n სტრიქონს მივიღებთ.

ქვემოთ მოყვანილ მოთხოვნაში გამოიყენება Personali და Xelshekruleba ცხრილების ჯვარედინი შეერთება. შედეგობრივი ნაკრები შედგება personaliID (თანამშრომლის იდენტიფიკატორი) და xelshekrulebaID (ხელშეკრულების იდენტიფიკატორი) სვეტებისაგან. მოთხოვნას აქვს სახე:

```
USE Shekveta;
SELECT P.personaliID, X.xelshekrulebaID
FROM   Personali P CROSS JOIN Xelshekruleba X
ORDER BY P.personaliID, X.xelshekrulebaID;
```

შედეგი.

	personaliID	xelshekrulebaID
1	1	1
2	1	2
3	1	3
4	1	4
5	1	5
6	1	6
7	1	7
8	1	8
9	1	9
10	1	10
11	1	11
12	1	12
13	1	13
14	1	14
15	1	15
16	1	16
17	1	17
18	1	18
19	1	19
20	1	20
21	1	21
22	2	1
23	2	2
24	2	3

. . .

FROM განყოფილებაში P და X ფსევდონიმები Personali და Xelshekruleba ცხრილებს მიენიჭა. ჯვარედინი შეერთების მიერ ფორმირებული შედეგობრივი ნაკრები არის ვირტუალური ცხრილი, რომელიც შეერთებაში მონაწილე ორივე საწყისი ცხრილის სვეტებისაგან შედგება. რადგან, საწყის ცხრილებს ფსევდონიმები დაენიშნა, ამიტომ ვირტუალურ ცხრილში სვეტების სახელებს აქვს პრეფიქსები ფსევდონიმების სახით (P.personaliID, X.xelshekrulebaID). თუ FROM განყოფილებაში ცხრილებს ფსევდონიმებს არ ვანიჭებთ, მაშინ სვეტების სახელების პრეფიქსები მიღებულ (შედეგობრივ) ცხრილში იქნება საწყისი ცხრილების სრული სახელები (Personali.personaliID, Xelshekruleba.xelshekrulebaID).

პრეფიქსი იძლევა სვეტის ცალსახად იდენტიფიცირების საშუალებას მაშინ, როცა ორივე ცხრილში გვხვდება სვეტების ერთნაირი სახელები. ცხრილებს ფსევდონიმები სახელის სიმოკლისთვის ენიჭება. დაბოლოს, შევნიშნოთ რომ, სვეტების პრეფიქსების მითითება აუცილებელია მაშინ, როცა სვეტების სახელები არაცალსახაა, ე.ი. სვეტის ერთი და იგივე სახელი ორივე ცხრილში გვხვდება. თუ სვეტების სახელები ცალსახადაა განსაზღვრული, მაშინ პრეფიქსები შეგვიძლია არ გამოვიყენოთ.

მაგალითები.

ა. CROSS JOIN კავშირი WHERE განყოფილების გარეშე წარმოქმნის ცხრილს, რომლის ზომა (სტრიქონების რაოდენობა) პირველი და მეორე ცხრილების სტრიქონების რაოდენობების ნამრავლით განისაზღვრება. Personali და Xelshekruleba ცხრილებს შორის დავამყაროთ CROSS JOIN კავშირი. შესაბამის მოთხოვნას აქვს სახე:

```
USE Shekveta;
SELECT P.personaliID, X.gadasaxdeli_l AS [გადასახდელი ლარებში]
FROM   Personali P CROSS JOIN Xelshekruleba X
ORDER BY P.personaliID, X.gadasaxdeli_l;
```

შედეგი.

	personaliID	გადასახდელი ლარებში
1	1	1900
2	1	2000
3	1	2500
4	1	3000
5	1	3500
6	1	3700
7	1	3900
8	1	4400
9	1	4400
10	1	5000
11	1	5300
12	1	5600
13	1	6000
14	1	7100
15	1	7700
16	1	8000
17	1	8000
18	1	8200
19	1	8500
20	1	9800
21	1	10000
22	2	1900
23	2	2000
24	2	2500

შედეგში გამოჩნდება 231 სტრიქონი, რადგან Personali ცხრილში 11 სტრიქონია, Xelshekruleba ცხრილში კი - 21.

ბ. თუ წინა მოთხოვნას WHERE განყოფილებას დავუმატებთ, მაშინ CROSS JOIN კავშირი იმუშავებს ზუსტად ისე, როგორც INNER JOIN კავშირი. მოთხოვნას აქვს სახე:

```
USE Shekveta;
SELECT  P.personaliID, X.gadasaxdeli_l AS [გადასახდელი ლარებში]
FROM    Personali P CROSS JOIN Xelshekruleba X
WHERE P.personaliID = X.personaliID
ORDER BY P.personaliID, X.gadasaxdeli_l;
```

იმავე შედეგს მივიღებთ, თუ შევასრულებთ შემდეგ მოთხოვნას:

```
USE Shekveta;
SELECT P.personaliID, X.gadasaxdeli_l AS [გადასახდელი ლარებში]
FROM   Personali P INNER JOIN Xelshekruleba X
        ON P.personaliID = X.personaliID
ORDER BY P.personaliID, X.gadasaxdeli_l;
```

შედეგი.

	personaliID	გადასახდელი ლარებში
1	1	6000
2	1	7100
3	1	9800
4	2	3700
5	2	5000
6	2	8000
7	3	1900
8	3	2000
9	3	2500
10	3	3000
11	3	8000
12	3	8200
13	4	4400
14	5	5300
15	6	4400
16	7	3500
17	8	5600
18	8	7700
19	9	8500
20	9	10000
21	10	3900

ჯვარედინი თვითშეერთებები

ჩვენ შეგვიძლია შევაერთოთ ერთი და იმავე ცხრილის რამდენიმე ეგზემპლარი. ამ შესაძლებლობას *თვითშეერთება* ეწოდება. თვითშეერთების დროს შეგვიძლია გამოვიყენოთ ყველა ძირითადი ტიპის შეერთება. მოყვანილი მოთხოვნა ასრულებს Personali ცხრილის ორი ეგზემპლარის ჯვარედინ თვითშეერთებას:

```
SELECT P1.personaliID, P1.gvari, P1.saxeli, P2.personaliID, P2.gvari, P2.saxeli
```

FROM Personali AS P1 CROSS JOIN Personali AS P2;

ეს მოთხოვნა აფორმირებს თანამშრომლების წყვილის ყველა შესაძლო კომბინაციას. რადგან Personali ცხრილში 11 სტრიქონია, ამიტომ მოთხოვნა გასცემს 121 სტრიქონს:

	personaliID	gvari	saxeli	personaliID	gvari	saxeli
1	1	სამხარაძე	საბა	1	სამხარაძე	საბა
2	2	სამხარაძე	ანა	1	სამხარაძე	საბა
3	3	გაჩეჩილაძე	ლია	1	სამხარაძე	საბა
4	4	გაჩეჩილაძე	ხათუნა	1	სამხარაძე	საბა
5	5	შენგელია	ნატა	1	სამხარაძე	საბა
6	6	კაპანაძე	ეთერი	1	სამხარაძე	საბა
7	7	ქუხიშვილი	ალექსანდრე	1	სამხარაძე	საბა
8	8	ყალაბეგიშვილი	გია	1	სამხარაძე	საბა
9	9	გიორგაძე	გივი	1	სამხარაძე	საბა
10	10	კიკნაძე	მზია	1	სამხარაძე	საბა
11	11	სამხარაძე	გიორგი	1	სამხარაძე	საბა
12	1	სამხარაძე	საბა	2	სამხარაძე	ანა
13	2	სამხარაძე	ანა	2	სამხარაძე	ანა
14	3	გაჩეჩილაძე	ლია	2	სამხარაძე	ანა
15	4	გაჩეჩილაძე	ხათუნა	2	სამხარაძე	ანა
16	5	შენგელია	ნატა	2	სამხარაძე	ანა
17	6	კაპანაძე	ეთერი	2	სამხარაძე	ანა
18	7	ქუხიშვილი	ალექსანდრე	2	სამხარაძე	ანა
19	8	ყალაბეგიშვილი	გია	2	სამხარაძე	ანა
20	9	გიორგაძე	გივი	2	სამხარაძე	ანა
21	10	კიკნაძე	მზია	2	სამხარაძე	ანა

. . .

თვითშეერთებაში ცხრილებისთვის ფსევდონიმების მინიჭება აუცილებელია.

შიგა შეერთებები

შიგა შეერთების დროს სრულდება მოთხოვნის ლოგიკური დამუშავების ორი სტადია. ჯერ, განისაზღვრება ორი შესასვლელი ცხრილის დეკარტული ნამრავლი, შემდეგ კი ამოირჩევა სტრიქონები ჩვენ მიერ განსაზღვრული პრედიკატის (გამოსახულების) საფუძველზე.

ANSI SQL-92 სინტაქსურ ჩანაწერში ცხრილების სახელებს შორის ჩვენ ვიყენებთ INNER JOIN საკვანძო სიტყვას. INNER სიტყვა შეგვიძლია არ მივუთითოთ და გამოვიყენოთ მხოლოდ JOIN საკვანძო სიტყვა. პრედიკატი, რომელიც გამოიყენება სტრიქონების გასაფილტრად, მოიცემა ON ელემენტში. ამ პრედიკატს, აგრეთვე *შეერთების პირობა* ეწოდება.

მაგალითი. გვაინტერესებს იმ თანამშრომლების სია (Personali ცხრილი), რომლებსაც შეკვეთები (Xelshekruleba ცხრილი) აქვთ გაფორმებული. ცხრილებს შორის დავამყაროთ INNER JOIN კავშირი. მოთხოვნას აქვს სახე:

```
USE Shekveta;
SELECT Personali.gvari, Personali.saxeli, Personali.qalaqi,
       Xelshekruleba.gadasaxdeli_l,   Xelshekruleba.vali_l
FROM   Personali INNER JOIN Xelshekruleba
       ON Personali.personaliID = Xelshekruleba.personaliID
ORDER BY Personali. gvari, Personali.saxeli;
```

შედეგი.

	gvari	saxeli	qalaqi	gadasaxdeli_l	vali_l
1	გაჩეჩილამე	ლია	თბილისი	1900	0
2	გაჩეჩილამე	ლია	თბილისი	2500	0
3	გაჩეჩილამე	ლია	თბილისი	8200	200
4	გაჩეჩილამე	ლია	თბილისი	8000	6400
5	გაჩეჩილამე	ლია	თბილისი	3000	0
6	გაჩეჩილამე	ლია	თბილისი	2000	300
7	გაჩეჩილამე	ხათუნა	თბილისი	4400	0
8	გიორგამე	გივი	ქობულეთი	8500	100
9	გიორგამე	გივი	ქობულეთი	10000	0
10	კაპანამე	ეთერი	რუსთავი	4400	500
11	კიკნამე	მზია	ბათუმი	3900	0
12	სამხარამე	ანა	ბათუმი	5000	0
13	სამხარამე	ანა	ბათუმი	3700	700
14	სამხარამე	ანა	ბათუმი	8000	3700
15	სამხარამე	საბა	ზუგდიდი	6000	500
16	სამხარამე	საბა	ზუგდიდი	9800	800
17	სამხარამე	საბა	ზუგდიდი	7100	0
18	ქვხიშვილი	ალექ...	თბილისი	3500	0
19	ყალაბეგი...	გია	ქუთაისი	5600	0
20	ყალაბეგი...	გია	ქუთაისი	7700	0
21	შენგელია	ნატა	ზუგდიდი	5300	500

Personali ცხრილის თითოეული სტრიქონი უკავშირდება Xelshekruleba ცხრილის მხოლოდ იმ სტრიქონებს, სადაც Personali ცხრილის personaliID (პირველადი გასაღები) სვეტის მნიშვნელობა Xelshekruleba ცხრილის personaliID (გარე გასაღები) სვეტის მნიშვნელობის ტოლია. შედეგში არ მოთავსდება Personali ცხრილის ის სტრიქონები, რომლებსაც არ აქვთ personaliID სვეტის შესაბამისი მნიშვნელობების მქონე სტრიქონები Xelshekruleba ცხრილში. შედეგში არ მოთავსდება, აგრეთვე, Xelshekruleba ცხრილის ის სტრიქონები, რომლებისთვისაც Personali ცხრილში არ არსებობს შესაბამისი მნიშვნელობები.

ეს მოთხოვნა შეგვიძლია ასეც ჩავწეროთ:

```
SELECT Personali.gvari, Personali.saxeli, Personali.qalaqi,
       Xelshekruleba.gadasaxdeli_l, Xelshekruleba.vali_l
FROM   Personali, Xelshekruleba
WHERE Personali.personaliID = Xelshekruleba.personaliID;
```

თუმცა, პირველ მოთხოვნას SQL სერვერი უფრო სწრაფად შეასრულებს, ამიტომ უმჯობესია მისი გამოყენება.

შიგა შეერთების სახეები

ამ განყოფილებაში განვიხილავთ შიგა შეერთების რამდენიმე სახეს: შედგენილ შეერთებებს (composite joins), შეერთებებს უტოლობის შემთხვევაში და მრავალცხრილურ შეერთებებს.

შედგენილი შეერთებები

შედგენილი შეერთება ეფუძნება პრედიკატს, რომელიც მოიცავს რამდენიმე სვეტს თითოეული ცხრილიდან. შედგენილ შეერთებას ადგილი აქვს მაშინ, როცა სრულდება ორი ცხრილის შეერთება „პირველადი გასაღები - გარე გასაღები" კავშირის საფუძველზე და კავშირი შედგენილია, ე.ი. პირველადი გასაღები შედგება რამდენიმე სვეტისაგან და გარე გასაღებიც შედგება ამდენივე სვეტისაგან. დავუშვათ, Table2 ცხრილში განსაზღვრულია გარე გასაღები, რომელიც col1 და col2 სვეტებისგან შედგება. ეს სვეტები მიმართავენ Table1 ცხრილის პირველად გასაღებს, რომელიც col1 და col2 სვეტებისგან შედგება. მოთხოვნას, რომელიც ასრულებს შეერთებას „პირველადი გასაღები - გარე გასაღები" კავშირის საფუძველზე, ასეთი სახე ექნება:

```
SELECT T1.col1, T1.col2, T2.col1, T2.col2
FROM Table1 AS T1 JOIN Table2 AS T2
        ON T1.col1 = T2.col1 AND T1.col2 = T2.col2;
```

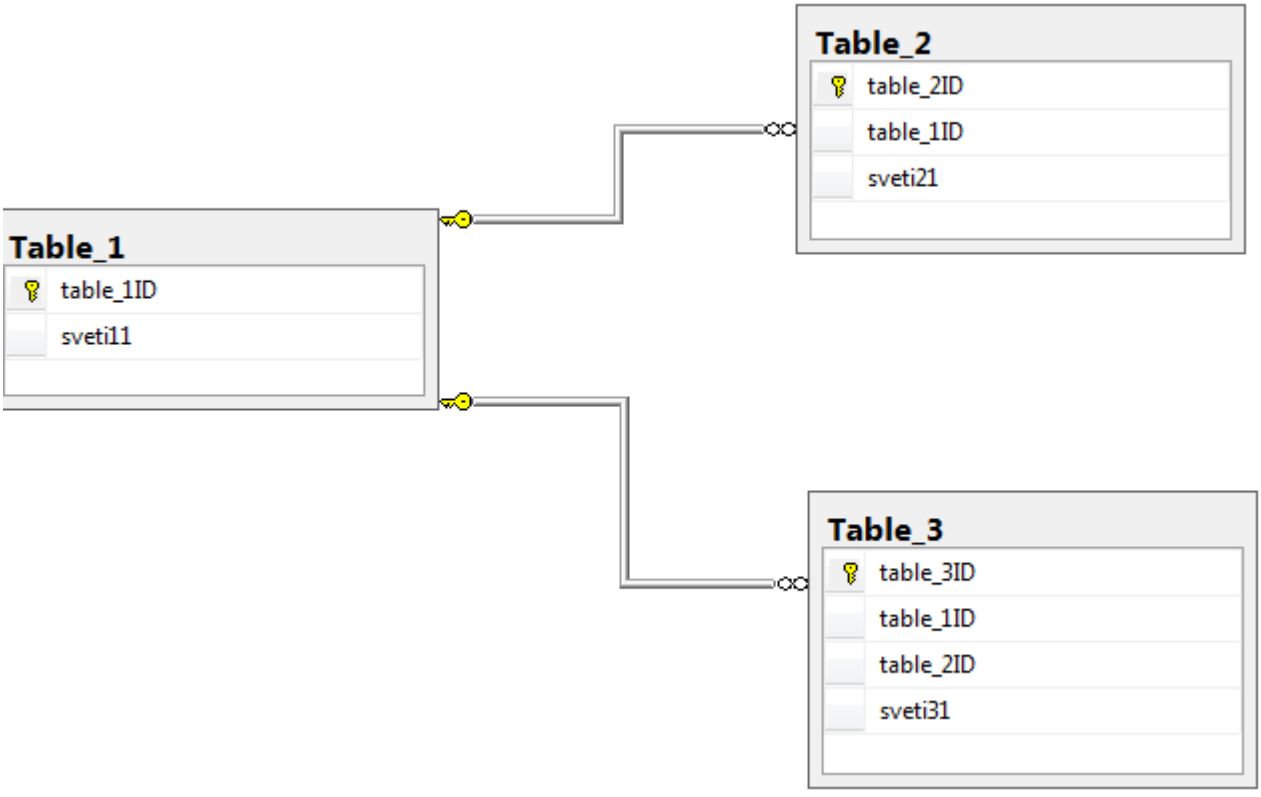
შეერთებები უტოლობის შემთხვევაში

თუ შეერთების პირობა მხოლოდ ტოლობის ოპერაციას შეიცავს, მაშინ ეს არის *შეერთება ტოლობის პირობით* ან *ეკვივალენტური შეერთება*. თუ შეერთების პირობაში გამოიყენება შედარების ნებისმიერი სხვა ოპერაცია, მაშინ ასეთ შეერთებას ეწოდება *შეერთება უტოლობის პირობით*. მოყვანილი მოთხოვნა ერთმანეთთან აკავშირებს Personali ცხრილის ორ ეგზემპლარს თანამშრომლების უნიკალური წყვილების ფორმირების მიზნით, რისთვისაც იყენებს უტოლობის პირობიან შეერთებას:

```
SELECT P1.personaliID, P1.saxeli, P1.gvari, P2.personaliID, P2.saxeli, P2.gvari
FROM   Personali AS P1 JOIN Personali AS P2
         ON P1.personaliID < P2.personaliID;
```

აქ, ჯვარედინი შეერთება (დაკავშირება) რომ გამოგვეყენებინა, მაშინ მივიღებდით ერთნაირი ნომრების მქონე თანამშრომლების წყვილებს (მაგალითად, 1 და 1) და სარკისებურად ასახულ წყვილებს (მაგალითად, 1 და 2 და 2 და 1). ამ ორ მიუღებელ ვარიანტს გამორიცხავს შიგა შეერთებაში გამოყენებული უტოლობის პირობა, რომლის თანახმად შედარების ოპერაციის მარცხნივ მყოფი გასაღების მნიშვნელობა უნდა იყოს ნაკლები მარჯვნივ მყოფი გასაღების მნიშვნელობაზე. ჩვენს მოთხოვნაში თანამშრომლების 132 შესაძლო წყვილიდან, რომელსაც ჯვარედინი შეერთება დაგვიბრუნებდა, ამოირჩევა მხოლოდ 55 უნიკალური წყვილი:

	personaliID	sakheli	gvari	personaliID	sakheli	gvari
1	1	საბა	სამხარაძე	2	ანა	სამხარაძე
2	1	საბა	სამხარაძე	3	ლია	გაჩეჩილაძე
3	2	ანა	სამხარაძე	3	ლია	გაჩეჩილაძე
4	1	საბა	სამხარაძე	4	ხათუნა	გაჩეჩილაძე
5	2	ანა	სამხარაძე	4	ხათუნა	გაჩეჩილაძე
6	3	ლია	გაჩეჩილაძე	4	ხათუნა	გაჩეჩილაძე
7	1	საბა	სამხარაძე	5	ნატა	შენგელია
8	2	ანა	სამხარაძე	5	ნატა	შენგელია
9	3	ლია	გაჩეჩილაძე	5	ნატა	შენგელია
10	4	ხათუნა	გაჩეჩილაძე	5	ნატა	შენგელია
11	1	საბა	სამხარაძე	6	ეთერი	კაპანაძე
12	2	ანა	სამხარაძე	6	ეთერი	კაპანაძე
13	3	ლია	გაჩეჩილაძე	6	ეთერი	კაპანაძე
14	4	ხათუნა	გაჩეჩილაძე	6	ეთერი	კაპანაძე
15	5	ნატა	შენგელია	6	ეთერი	კაპანაძე
16	1	საბა	სამხარაძე	7	ალექსანდრე	ქუხიშვილი



ნახ. 5.1. Table_1 ცხრილისთვის დამოკიდებულია Table_2 და Table_3 ცხრილები.

მრავალცხრილიანი შეერთებები

შეერთების ოპერაცია სრულდება მხოლოდ ორი ცხრილისთვის, მაგრამ, ერთ

მოთხოვნაში შეიძლება იყოს ბევრი შეერთება. როგორც წესი, FROM განყოფილებაში გვხვდება რამდენიმე ცხრილური ოპერაცია. ისინი ლოგიკურად მუშავდება მარცხნიდან მარჯვნივ. ეს იმას ნიშნავს, რომ შედეგობრივი ცხრილი, მიღებული პირველ ცხრილურ ოპერაციაში, გამოიყენება მარცხენა ცხრილად მეორე ცხრილური ოპერაციისათვის; მეორე ცხრილური ოპერაციის შედეგი გამოიყენება მარცხენა ცხრილად მესამე ცხრილური ოპერაციისთვის და ა.შ. ამრიგად, თუ FROM განყოფილებაში არის რამდენიმე შეერთება, პირველი შეერთება ასრულებს ორი საბაზო ცხრილის ლოგიკურ დამუშავებას, ხოლო ყველა დანარჩენი შეერთება იღებს წინა შეერთების შედეგს, როგორც მარცხენა ცხრილს.

რელაციის დემონსტრირებისთვის, ჯერ შევქმნათ Table_1, Table_2 და Table_3 ცხრილები, შემდეგ კი შევასრულოთ მათ შორის რელაცია:

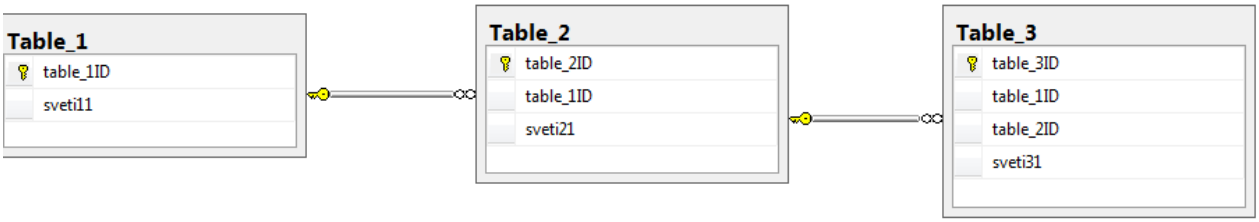
```
USE Shekveta;
-- ცხრილების შექმნა
CREATE TABLE Table_1
(
table_1ID INT PRIMARY KEY IDENTITY(1,1),
sveti11 INT
);
CREATE TABLE Table_2
(
table_2ID INT PRIMARY KEY IDENTITY(1,1),
table_1ID INT,            -- REFERENCES Table_1(table_1ID),
sveti21 INT
);
CREATE TABLE Table_3
(
table_3ID INT PRIMARY KEY IDENTITY(1,1),
table_1ID INT,
table_2ID INT,
sveti31 INT
);
-- ცხრილების შევსება
INSERT INTO Table_1 VALUES(11), (12), (13), (14), (15);
INSERT INTO Table_2 VALUES(1, 21), (1, 22), (3, 23), (2, 24), (3, 25);
INSERT INTO Table_3 VALUES(1, 1, 31), (2, 3, 32), (3, 4, 33), (3, 2, 34), (1, 2, 35);
```

მოყვანილ მოთხოვნაში მთავარია Table_1 ცხრილი, რომელსაც ორი დამოკიდებული ცხრილი აქვს: Table_2 და Table_3 (ნახ. 5.1). მოთხოვნას აქვს სახე:

```
USE Shekveta;
SELECT  T1.table_1ID, T1.sveti11,
          T2.table_2ID, T2.table_1ID, T2.sveti21,
          T3.table_3ID, T3.table_1ID,  T3.table_2ID, T3.sveti31
FROM  Table_1 T1 INNER JOIN Table_2 T2
                  ON   T1.table_1ID = T2.table_1ID
                       INNER JOIN Table_3 T3
                  ON T1.table_1ID = T3.table_1ID
ORDER BY T1.table_1ID;
```

შედეგი.

	table_1ID	sveti11	table_2ID	table_1ID	sveti21	table_3ID	table_1ID	table_2ID	sveti31
1	1	11	1	1	21	1	1	1	31
2	1	11	2	1	22	1	1	1	31
3	1	11	1	1	21	5	1	2	35
4	1	11	2	1	22	5	1	2	35
5	2	12	4	2	24	2	2	3	32
6	3	13	3	3	23	3	3	4	33
7	3	13	5	3	25	3	3	4	33
8	3	13	3	3	23	4	3	2	34
9	3	13	5	3	25	4	3	2	34



ნახ. 5.2. Table_1 ცხრილისთვის დამოკიდებულია Table_2 ცხრილი, ხოლო Table_2 ცხრილისთვის დამოკიდებულია Table_3 ცხრილი.

მოყვანილ მოთხოვნაში Table_1 ცხრილისთვის დამოკიდებულია Table_2 ცხრილი, ხოლო Table_2 ცხრილისთვის დამოკიდებულია Table_3 ცხრილი (ნახ. 5.2). მოთხოვნას აქვს სახე:

```
USE Shekveta;
SELECT  T1.table_1ID, T1.sveti11,
          T2.table_2ID, T2.table_1ID, T2.sveti21,
          T3.table_3ID, T3.table_1ID,  T3.table_2ID, T3.sveti31
FROM  Table_1 T1
          INNER JOIN Table_2 T2
                    ON   T1.table_1ID = T2.table_1ID
          INNER JOIN Table_3 T3
                    ON T2.table_2ID = T3.table_2ID
ORDER BY T1.table_1ID;
```

შედეგი.

	table_1ID	sveti11	table_2ID	table_1ID	sveti21	table_3ID	table_1ID	table_2ID	sveti31
1	1	11	1	1	21	1	1	1	31
2	1	11	2	1	22	4	3	2	34
3	1	11	2	1	22	5	1	2	35
4	2	12	4	2	24	3	3	4	33
5	3	13	3	3	23	2	2	3	32

გარე შეერთებები

როგორც ვიცით, *გარე შეერთებები* შიგა შეერთებების დროს გამოყენებული ლოგიკური დამუშავების ორივე სტადიის გარდა, მოიცავს მესამე, მხოლოდ მისთვის დამახასიათებელ სტადიას, რომელსაც *გარე სტრიქონების დამატება* ეწოდება.

გარე შეერთებაში ერთი ცხრილი მოინიშნება როგორც „შესანახი" (მთავარი) LEFT OUTER JOIN, RIGHT OUTER JOIN ან FULL OUTER JOIN საკვანძო სიტყვების საშუალებით, მეორე კი - როგორც „არაშესანახი“ (დამხმარე). OUTER საკვანძო სიტყვის მითითება არ არის აუცილებელი. LEFT საკვანძო სიტყვა ნიშნავს, რომ შეინახება მარცხენა ცხრილის სტრიქონები (შედეგში გამოჩნდება მარცხენა ცხრილის ყველა სტრიქონი), ხოლო RIGHT სიტყვა მიუთითებს, რომ შეინახება მარჯვენა ცხრილის სტრიქონები (შედეგში გამოჩნდება მარჯვენა ცხრილის ყველა სტრიქონი). FULL საკვანძო სიტყვა მიუთითებს, რომ შეინახება ორივე ცხრილის სტრიქონები (შედეგში გამოჩნდება ორივე ცხრილის ყველა სტრიქონი). გარე შეერთებაში მოთხოვნის ლოგიკური დამუშავების მესამე სტადია განსაზღვრავს სტრიქონებს შესანახი ცხრილიდან, რომლებისთვისაც არ მოინახა შესაბამისი სტრიქონები არაშესანახ ცხრილში ON ელემენტის პრედიკატის საფუძველზე. ამ სტადიაზე (მესამე სტადიაზე) შედეგობრივ ცხრილს, რომლის ფორმირებაც მოხდა პირველ ორ სტადიაზე, დაემატება აღნიშნული სტრიქონები, და მათში სვეტების მნიშვნელობების შემვსებებად გამოიყენება NULL მნიშვნელობები არაშესანახი (დამხმარე) ცხრილიდან.

მაგალითები.

ა. გვაინტერესებს თითოეული შემკვეთის (Shemkveti ცხრილი) ხელშეკრულებები (Xelshekruleba ცხრილი). ცხრილებს შორის დავამყაროთ LEFT OUTER JOIN კავშირი:

```
USE Shekveta;
SELECT S.shemkvetiID, S.firmis_dasaxeleba, X.xelshekrulebaID
FROM   Shemkveti AS S LEFT OUTER JOIN Xelshekruleba AS X
        ON S.shemkvetiID = X.shemkvetiID;
```

შედეგი.

	shemkvetiID	firmis_dasaxeleba	xelshekrulebaID
1	1	NULL	1
2	1	NULL	7
3	1	NULL	12
4	1	NULL	14
5	2	Gia&Co.	2
6	2	Gia&Co.	3
7	2	Gia&Co.	10
8	3	Jemali&Co.	6
9	3	Jemali&Co.	9
10	3	Jemali&Co.	11
11	4	NULL	8
12	5	Zura&Co.	4
13	5	Zura&Co.	13
14	6	NULL	5
15	7	NULL	15
16	7	NULL	16
17	8	Givi&Co.	NULL
18	9	Gela&Co.	NULL
19	10	R&Co.	NULL
20	11	NULL	NULL
21	12	NULL	18
22	13	NULL	19
23	14	NULL	20

შედეგში გამოჩნდება Shemkveti ცხრილის ყველა სტრიქონი, მათ შორის ის სტრიქონებიც, რომლებსაც არ აქვთ შესაბამისი სტრიქონები Xelshekruleba ცხრილში. მოყვანილი მოთხოვნა აერთებს (აკავშირებს) Shemkveti და Xelshekruleba ცხრილებს. გაერთიანება ხორციელდება Shemkveti ცხრილის shemkvetiID სვეტისა (პირველადი გასაღები) და Xelshekruleba ცხრილის shemkvetiID სვეტის (გარე გასაღები) მნიშვნელობების დამთხვევის საფუძველზე. შეერთების ტიპია მარცხენა გარე შეერთება. შედეგად, მოთხოვნა გასცემს იმ შემკვეთებსაც, რომლებმაც არც ერთი შეკვეთა არ გააფორმეს. ასეთი შემკვეთების იდენტიფიკატორებია: 8, 9, 10 და 11.

თუ გვინდა ფილტრის მითითება, რომელიც გამოიყენება მას შემდეგ, რაც გარე სტრიქონები დაემატება შედეგობრივ ნაკრებს, და გვინდა, რომ ჩვენი ამორჩევა საბოლოო იყოს, მაშინ უნდა გამოვიყენოთ WHERE განყოფილება. ის მუშავდება FROM განყოფილების შემდეგ. კერძოდ, მას შემდეგ, რაც შესრულებულია ყველა ცხრილური ოპერაცია. ამას გარდა, WHERE პირობა ON ელემენტისაგან განსხვავებით გამოსარიცხი სტრიქონებისთვის არის საბოლოო.

დავუშვათ, რომ გვინდა იმ შემკვეთების პოვნა, რომლებსაც არც ერთი შეკვეთა არ გააფორმეს. ეს იმას ნიშნავს, რომ უნდა გავცეთ მხოლოდ გარე სტრიქონები. ამ მიზნით, წინა მოთხოვნას დავუმატოთ WHERE განყოფილება, რომელიც მხოლოდ გარე სტრიქონებს ამოირჩევს. მოთხოვნას აქვს სახე:

```
SELECT S.shemkvetiID, S.firmis_dasaxeleba
FROM   Shemkveti AS S LEFT OUTER JOIN Xelshekruleba AS X
         ON S.shemkvetiID = X.shemkvetiID
WHERE X.shemkvetiID IS NULL;
```

შედეგი.

	shemkvetiID	firmis_dasaxeleba
1	8	Givi&Co.
2	9	Gela&Co.
3	10	R&Co.
4	11	NULL

მხედველობაში უნდა მივიღოთ ორი მნიშვნელოვანი შენიშვნა, რომლებიც ამ მოთხოვნას ეხება. პირველი, NULL მნიშვნელობის ძებნისას უნდა გამოვიყენოთ IS NULL ოპერატორი, და არა ტოლობის ოპერაცია. მეორე, ფილტრში უნდა ამოვირჩიოთ სვეტი არაშესანახი ცხრილიდან, რომელიც შეერთებაში მონაწილეობს. უნდა ავირჩიოთ სვეტი, რომელიც იღებს NULL მნიშვნელობას მხოლოდ გარე სტრიქონში და არა სხვა შემთხვევაში (მაგალითად, NULL მნიშვნელობა, გადმოსული საწყისი ცხრილიდან). აქ ადგილი აქვს სამ შემთხვევას: პირველადი გასაღების სვეტი, შემაერთებელი სვეტი, და სვეტი, რომელიც განსაზღვრულია როგორც NOT NULL (NULL მნიშვნელობები არ დაიშვება). პირველადი გასაღების სვეტს არ შეიძლება ჰქონდეს NULL მნიშვნელობები, შედეგად, NULL ასეთ სვეტში შეიძლება ნიშნავდეს მხოლოდ გარე სტრიქონს. თუ სტრიქონის მაკავშირებელ სვეტში არის NULL მნიშვნელობა, მაშინ ასეთი სტრიქონი უკუიგდება შეერთების მეორე სტადიის მიერ. ამიტომ NULL ასეთ სვეტში შეიძლება ნიშნავდეს მხოლოდ იმას, რომ ეს გარე სტრიქონია. დაბოლოს, NULL მნიშვნელობა სვეტში, რომელიც განსაზღვრულია როგორც NOT NULL, შეიძლება ნიშნავდეს მხოლოდ იმას, რომ მოცემული სტრიქონი გარეა.

ბ. გვაინტერესებს თითოეული შეკვეთის (Xelshekruleba ცხრილი) შემკვეთები (Shemkveti ცხრილი). ცხრილებს შორის დავამყაროთ RIGHT OUTER JOIN კავშირი. მოთხოვნას აქვს სახე:

```
USE Shekveta;
SELECT S.shemkvetiID, S.firmis_dasaxeleba, X.xelshekrulebaID
FROM   Shemkveti AS S RIGHT OUTER JOIN Xelshekruleba AS X
          ON S.shemkvetiID = X.shemkvetiID;
```

შედეგი.

	shemkvetiID	firmis_dasaxeleba	xelshekrulebaID
1	NULL	NULL	17
2	12	NULL	18
3	13	NULL	19
4	14	NULL	20
5	NULL	NULL	21
6	1	NULL	1
7	2	Gia&Co.	2
8	2	Gia&Co.	3
9	5	Zura&Co.	4
10	6	NULL	5
11	3	Jemali&Co.	6
12	1	NULL	7
13	4	NULL	8
14	3	Jemali&Co.	9
15	2	Gia&Co.	10
16	3	Jemali&Co.	11
17	1	NULL	12

გ. გვაინტერესებს თითოეული შემკვეთის (Shemkveti ცხრილი) შეკვეთები (Xelshekruleba ცხრილი). ცხრილებს შორის დავამყაროთ FULL OUTER JOIN კავშირი. მოთხოვნას აქვს სახე:

```
USE Shekveta;
SELECT S.shemkvetiID, S.firmis_dasaxeleba, X.xelshekrulebaID
FROM   Shemkveti AS S FULL OUTER JOIN Xelshekruleba AS X
          ON S.shemkvetiID = X.shemkvetiID;
```

შედეგი.

	shemkvetiID	firmis_dasaxeleba	xelshekrulebaID
1	1	NULL	1
2	1	NULL	7
3	1	NULL	12
4	1	NULL	14
5	2	Gia&Co.	2
6	2	Gia&Co.	3
7	2	Gia&Co.	10
8	3	Jemali&Co.	6
9	3	Jemali&Co.	9
10	3	Jemali&Co.	11
11	4	NULL	8
12	5	Zura&Co.	4
13	5	Zura&Co.	13
14	6	NULL	5
15	7	NULL	15
16	7	NULL	16
17	8	Givi&Co.	NULL
18	9	Gela&Co.	NULL
19	10	R&Co.	NULL
20	11	NULL	NULL
21	12	NULL	18
22	13	NULL	19
23	14	NULL	20
24	NULL	NULL	17
25	NULL	NULL	21

თავი 6. მმართველი კონსტრუქციები

BEGIN...END

ეს კონსტრუქცია იძლევა ერთ ბლოკში ორი და მეტი ბრძანების დაჯგუფების საშუალებას. BEGIN საკვანძო სიტყვა იწყებს ბლოკს, END კი - ამთავრებს. ასეთი დაჯგუფება შეგვიძლია გამოვიყენოთ განშტოებებში, პირობისა და ციკლის კონსტრუქციებში.

მაგალითი. პროგრამულ კოდში ხდება შეკვეთის მაქსიმალური და მინიმალური თანხების განსაზღვრა:

```
BEGIN
DECLARE @max FLOAT, @min FLOAT;
SET @max =
(
SELECT MAX(gadasaxdeli_l)
FROM   Xelshekruleba
);
SET @min =
(
SELECT MIN(gadasaxdeli_l)
FROM   Xelshekruleba
);
SELECT @max AS [მაქსიმალური გადასახდელი თანხა],
        @min AS [მინიმალური გადასახდელი თანხა];
END ;
```

შედეგი:

	მაქსიმალური გადასახდელი თანხა	მინიმალური გადასახდელი თანხა
1	10000	1900

BEGIN...END ბლოკი შეიძლება იყოს ჩადგმული. ჩადგმულობის დონე შეზღუდული არ არის.

IF...ELSE

კონსტრუქცია საშუალებას გვაძლევს ბრძანება ან ბლოკი შევასრულოთ მხოლოდ მითითებული ლოგიკური პირობის შესრულების შემდეგ. მისი სინტაქსია:

```
IF  ლოგიკური_გამოსახულება
        { sql_ბრძანება | ბრძანებების_ბლოკი }
[ ELSE
        { sql_ბრძანება | ბრძანებების_ბლოკი } ]
```

ლოგიკური_გამოსახულება არის ლოგიკური პირობა, რომელიც იღებს TRUE (ჭეშმარიტი) ან FALSE (მცდარი) მნიშვნელობას. თუ ის იღებს TRUE მნიშვნელობას, მაშინ შესრულდება პირველი sql_*ბრძანება* ან *ბრძანებების_ბლოკი*. თუ პირობა იღებს FALSE მნიშვნელობას, მაშინ შესრულდება ELSE სიტყვის შემდეგ მოთავსებული sql_*ბრძანება* ან *ბრძანებების_ბლოკი*.

მაგალითები.

ა. განვსაზღვროთ რიცხვი კენტია თუ ლუწი. პროგრამულ კოდს აქვს სახე:

```
DECLARE @kenti_luwi INT;
SET @kenti_luwi = 10;
IF    @kenti_luwi % 2 = 0
        SELECT N'რიცხვი ლუწია';
ELSE
        SELECT N'რიცხვი კენტია';
```

შედეგი:

	(No column name)
1	რიცხვი ლუწია

ბ. განვსაზღვროთ ამჟამად თვის პირველი ნახევარია, თუ მეორე. პროგრამულ კოდს აქვს სახე:

```
DECLARE @striqoni NVARCHAR (20);
IF DAY(GETDATE()) < 15
        SET @striqoni = N'პირველი';
ELSE
        SET @striqoni = N'მეორე';
SELECT N'ახლა არის თვის ' + @striqoni + N' ნახევარი';
```

შედეგი:

	(No column name)
1	ახლა არის თვის მეორე ნახევარი

CASE...END

ეს არის მრავალგანშტოებიანი კონსტრუქცია. მისი სინტაქსია:

```
CASE case_გამოსახულება
WHEN { when_გამოსახულება | ლოგიკური_გამოსახულება }  [,...n]
THEN then_გამოსახულება [,...n]
[ ELSE else_გამოსახულება ]
END
```

case_*გამოსახულება* არგუმენტი შეიძლება შეიცავდეს ცვლადის სახელს ან ფუნქციას. when_*გამოსახულება* იღებს case_*გამოსახულება* არგუმენტის მნიშვნელობებიდან ერთ-ერთს. თუ ამ ორი არგუმენტის მნიშვნელობა ერთმანეთს დაემთხვევა, მაშინ გაიცემა then_*გამოსახულება* არგუმენტის მნიშვნელობა. შეიძლება მიეთითოს რამდენიმე WHEN...THEN სტრიქონი, რომლებიც შეიცავს case_*გამოსახულება* არგუმენტის შესაძლო მნიშვნელობას. თუ case_*გამოსახულება* არგუმენტის მნიშვნელობა არ ემთხვევა when_*გამოსახულება* არგუმენტის არც ერთ მნიშვნელობას, მაშინ გაიცემა else_*გამოსახულება* არგუმენტის მნიშვნელობა.

მაგალითი.

ა. შევადგინოთ პროგრამა, რომელიც შეასრულებს არითმეტიკულ ოპერაციას ოპერაციის ნიშანზე დამოკიდებულებით. პროგრამულ კოდს აქვს სახე:

```
DECLARE @ricxvi_1 INT,  @ricxvi_2 INT, @Shedegi INT, @operacia CHAR(1);
SET @ricxvi_1 = 10;
```

```
SET @ricxvi_2 = 20;
SET @operacia = '*';
SET @Shedegi  =
CASE @operacia
   WHEN '+' THEN @ricxvi_1 + @ricxvi_2
   WHEN '-' THEN @ricxvi_1 - @ricxvi_2
   WHEN '*' THEN @ricxvi_1 * @ricxvi_2
   WHEN '/' THEN @ricxvi_1 / @ricxvi_2
ELSE 0
END
SELECT @Shedegi AS [შედეგი];
```

შედეგი:

	შედეგი
1	200

ბ. შევადგინოთ პროგრამა, რომელიც გასცემს იმ თანამშრომლების გვარებს, რომელთა იდენტიფიკატორია 1÷3. პროგრამულ კოდს აქვს სახე:

```
USE Shekveta;
SELECT PersonaliID,
CASE PersonaliID
   WHEN 1 THEN N'სამხარაძე საბა'
   WHEN 2 THEN N'სამხარაძე ანა'
   WHEN 3 THEN N'გაჩეჩილაძე ლია'
   ELSE N'უცნობი'
END
FROM Personali;
```

შედეგი:

	PersonaliID	(No column name)
1	3	გაჩეჩილაძე ლია
2	5	უცნობი
3	11	უცნობი
4	1	სამხარაძე საბა
5	2	სამხარაძე ანა
6	7	უცნობი
7	10	უცნობი
8	4	უცნობი
9	9	უცნობი
10	6	უცნობი
11	8	უცნობი

COALESCE

კონსტრუქცია გასცემს პირველ არანულოვან მნიშვნელობას. მისი სინტაქსია:

COALESCE (*გამოსახულება* [,...])

მაგალითები.

ა. მოყვანილი პროგრამული კოდი გასცემს მეორე არგუმენტის მნიშვნელობას, რადგან პირველ არგუმენტს მნიშვნელობა არ აქვს:

```
DECLARE @ricxvi_1 INT,  @ricxvi_2 FLOAT, @ricxvi_3 INT;
SET @ricxvi_1 = 33;
SET @ricxvi_2 = 20.55;
PRINT COALESCE(@ricxvi_3, @ricxvi_1, @ricxvi_2);
```

შედეგი:

```
33
```

ბ. შევადგინოთ პროგრამა, რომელიც გამოიტანს ხელშეკრულების რეალურად შესრულების თარიღს. თუ სვეტში შესაბამისი მნიშვნელობა არ არის (სვეტი შეიცავს NULL მნიშვნელობას), მაშინ უნდა გამოვიტანოთ ხელშეკრულების დამთავრების თარიღი. კოდს აქვს სახე:

```
USE Shekveta;
SELECT COALESCE(tarigi_shesrulebis, tarigi_damtavrebis)
        AS 'ხელშეკრულების დასრულების თარიღი'
FROM Xelshekruleba;
```

შედეგი:

	ხელშეკრულების დასრულების თარიღი
1	2002-02-01 00:00:00.000
2	2000-11-10 00:00:00.000
3	2001-05-15 00:00:00.000
4	2003-01-20 00:00:00.000
5	2002-12-01 00:00:00.000
6	2004-02-25 00:00:00.000
7	2003-04-05 00:00:00.000
8	2003-10-17 00:00:00.000
9	2004-03-23 00:00:00.000
10	2006-09-07 00:00:00.000
11	2005-08-12 00:00:00.000
12	2007-07-27 00:00:00.000
13	2006-06-09 00:00:00.000
14	2006-09-03 00:00:00.000
15	2007-12-14 00:00:00.000
16	2008-03-21 00:00:00.000

როგორც ვხედავთ, tarigi_shesrulebis სვეტიდან გამოტანილ იქნა არსებული მნიშვნელობები, NULL მნიშვნელობის ნაცვლად კი - tarigi_damtavrebis სვეტის მნიშვნელობები.

გ. შევადგინოთ მოთხოვნა, რომელიც ერთ სვეტში გამოიტანს ფიზიკური და იურიდიული პირების მობილური ტელეფონების ნომერებს. მოთხოვნას აქვს სახე:

```
SELECT COALESCE(mobiluri, mobiluri_direqtoris)
FROM   Shemkveti;
```

შედეგი:

	(No column name)
1	551-743-434
2	552-111-111
3	553-444-444
4	554-222-222
5	555-555-555
6	556-565-556
7	557-999-999
8	558-223-334
9	559-333-333
10	571-334-344
11	572-983-983
12	573-522-532
13	574-661-621
14	574-777-564
15	598-456-899

WHILE...BREAK & CONTINUE

კონსტრუქცია გამოიყენება ციკლების ორგანიზებისთვის. მისი სინტაქსია:

WHILE *ლოგიკური_გამოსახულება*

{ sql_*ბრძანება* | *ბრძანებების_ბლოკი* }

[BREAK]

{ sql_*ბრძანება* | *ბრძანებების_ბლოკი* }

[CONTINUE]

{ sql_*ბრძანება* | *ბრძანებების_ბლოკი* }

თუ *ლოგიკური_გამოსახულება* იღებს TRUE მნიშვნელობას, მაშინ შესრულდება ციკლის ტანი (კოდი). ციკლის კოდი არის ციკლში შესასრულებელი ბრძანებების ერთობლიობა. ციკლის შესრულება შეწყდება, როგორც კი *ლოგიკური_გამოსახულება* FALSE მნიშვნელობას მიიღებს.

BREAK ბრძანება იწვევს ციკლის შესრულების იძულებით შეწყვეტას და ციკლიდან გამოსვლას. CONTINUE ბრძანების შესრულება იწვევს ლოგიკური პირობის შემოწმებაზე გადასვლას. BREAK და CONTINUE ბრძანების შემდეგ მოთავსებული ბრძანებები არ შესრულდება.

მაგალითები.

ა. გამოვთვალოთ 1-დან 10-მდე რიცხვების კვადრატი. პროგრამულ კოდს აქვს სახე:

```
DECLARE @ricxvi_1 INT;
SET      @ricxvi_1 = 1;
WHILE @ricxvi_1 <= 10
BEGIN
PRINT STR(@ricxvi_1) + N'-ის კვადრატი =' + STR(SQUARE(@ricxvi_1));
SET @ricxvi_1 = @ricxvi_1 + 1;
END
```

შედეგი:

```
    1-ის კვადრატი =           1
    2-ის კვადრატი =           4
    3-ის კვადრატი =           9
    4-ის კვადრატი =          16
    5-ის კვადრატი =          25
    6-ის კვადრატი =          36
    7-ის კვადრატი =          49
    8-ის კვადრატი =          64
    9-ის კვადრატი =          81
   10-ის კვადრატი =         100
```

ბ. იგივე პროგრამა გადავწეროთ BREAK და CONTINUE ოპერატორების გამოყენებით. პროგრამულ კოდს აქვს სახე:

```
DECLARE @ricxvi_1 INT;
SET @ricxvi_1 = 1;
WHILE 10 = 10
BEGIN
PRINT STR(@ricxvi_1) + N'-ის კვადრატი =' + STR(SQUARE(@ricxvi_1));
SET @ricxvi_1 = @ricxvi_1 + 1;
IF    @ricxvi_1 <= 10 CONTINUE;
BREAK;
END
```

შედეგი:

```
 1-ის კვადრატი =           1
 2-ის კვადრატი =           4
 3-ის კვადრატი =           9
 4-ის კვადრატი =          16
 5-ის კვადრატი =          25
 6-ის კვადრატი =          36
 7-ის კვადრატი =          49
 8-ის კვადრატი =          64
 9-ის კვადრატი =          81
10-ის კვადრატი =         100
```

გ. თუ თანამშრომლების საშუალო ხელფასი 600 ლარზე ნაკლებია, მაშინ გავაორმაგოთ მათი ხელფასი და შემდეგ ამოვირჩიოთ მათ შორის მაქსიმალური. თუ მაქსიმალური ხელფასი ნაკლებია ან ტოლი 1000 ლარის, მაშინ ისევ შევასრულოთ ციკლის კოდი (CONTINUE), საწინააღმდეგო შემთხვევაში ციკლის შესრულება შევწყვიტოთ (BREAK).

```
USE Shekveta;
WHILE ( SELECT AVG(xelfasi) FROM Personali_1 ) < 600
BEGIN
UPDATE Personali_1 SET xelfasi = xelfasi * 2;
IF ( SELECT MAX(xelfasi) FROM Personali_1 ) > 1000
        BREAK
ELSE
        CONTINUE;
END
```

შედეგი:

	(No column name)
1	1401

თავი 7. ქვემოთხოვნები

T-SQL ენა საშუალებას გვაძლევს ერთი მოთხოვნა მოვათავსოთ მეორე მოთხოვნის შიგნით და შევქმნათ *ჩადგმული მოთხოვნები*. მოთხოვნას, რომელიც მეორე მოთხოვნას მოიცავს, *გარე* ეწოდება. *შიგა მოთხოვნა (ქვემოთხოვნა)* - ესაა მოთხოვნა, რომლის მიერ გაცემული სტრიქონები (შედეგობრივი ნაკრები) გამოიყენება გარე მოთხოვნის მიერ. ქვემოთხოვნის მიერ გაცემული შედეგი შეიძლება იცვლებოდეს მასში მითითებულ ცხრილებში შეტანილი ცვლილებების გამო. ქვემოთხოვნების გამოყენება საშუალებას გვაძლევს, თავი ავარიდოთ ისეთ მოქმედებას, რომელიც ითხოვს ცვლადებში მოთხოვნის შუალედური შედეგების შენახვას.

ქვემოთხოვნა შეიძლება იყოს დამოუკიდებელი (მარტივი) ან ბმული (კორელირებული). დამოუკიდებელი ქვემოთხოვნა არ არის დამოკიდებული გარე მოთხოვნაზე, რომელსაც ის ეკუთვნის. ბმული ქვემოთხოვნა დამოკიდებულია გარე მოთხოვნაზე, რომელსაც ის ეკუთვნის. როგორც დამოუკიდებელ, ისე ბმულ ქვემოთხოვნას შეუძლია გასცეს ერთი (სკალარული) მნიშვნელობა, ბევრი (მრავლობითი) მნიშვნელობა ან ცხრილი.

დამოუკიდებელი ქვემოთხოვნები

თითოეულ ქვემოთხოვნას აქვს გარე მოთხოვნა, რომელსაც ის ეკუთვნის. *დამოუკიდებელია ქვემოთხოვნა*, რომელიც არ არის დამოკიდებული იმ გარე მოთხოვნაზე, რომელსაც ის ეკუთვნის. ჯერ სრულდება დამოუკიდებელი შიგა მოთხოვნა, შემდეგ კი გარე. დამოუკიდებელი ქვემოთხოვნის გამოყენება მოხერხებულია პროგრამული კოდის გამართვის დროს, რადგან ყოველთვის შეგვიძლია მისი დამოუკიდებლად შესრულება და იმაში დარწმუნება, რომ ის სწორ შედეგებს გასცემს.

დამოუკიდებელი სკალარული ქვემოთხოვნები

სკალარულია ქვემოთხოვნა, რომელიც გასცემს მხოლოდ ერთ მნიშვნელობას, მიუხედავად იმისა, ის ბმულია თუ არა. სკალარული ქვემოთხოვნა შეიძლება ჩავრთოთ გარე მოთხოვნის იმ ელემენტებში (WHERE, SELECT და ა.შ.), რომლებიც შეიძლება სკალარულ გამოსახულებებს შეიცავდეს.

დავუშვათ, გვინდა მივმართოთ Xelshekruleba ცხრილს და მივიღოთ ინფორმაცია მაქსიმალური ვალის მქონე შეკვეთის შესახებ. ეს ამოცანა შეგვიძლია გადავწყვიტოთ ცვლადის გამოყენებით. პროგრამა ჯერ ირჩევს ხელშეკრულების მაქსიმალურ ვალს Xelshekruleba ცხრილიდან და მას @maxvali_l ცვლადში ათავსებს. შემდეგ, პროგრამული კოდი მიმართავს Xelshekruleba ცხრილს და იმ ხელშეკრულებას ირჩევს, სადაც ვალის მნიშვნელობა @maxvali_l ცვლადში შენახული მნიშვნელობის ტოლია. პროგრამულ კოდს აქვს სახე:

```
USE Shekveta;
DECLARE @maxvali_l AS INT =
(
SELECT MAX(vali_l) FROM Xelshekruleba
);
SELECT  xelshekrulebaID,personaliID, shemkvetiID, tarigi_dawyebis, vali_l
FROM    Xelshekruleba
```

WHERE vali_l = @maxvali_l;

გაიცემა შედეგი:

	xelshekrulebaID	personaliID	shemkvetiID	tarigi_dawyebis	vali_l
1	9	3	4	2002-12-17 00:00:00.000	6400

ცვლადის გამოყენების ნაცვლად შეგვიძლია ჩადგმული ქვემოთხოვნის გამოყენება. ამისათვის, უნდა მივუთითოთ დამოუკიდებელი სკალარული ქვემოთხოვნა, რომელიც დაგვიბრუნებს ხელშეკრულების მაქსიმალურ ვალს. მოთხოვნას ექნება სახე:

```
SELECT xelshekrulebaID,personaliID, shemkvetiID, tarigi_dawyebis, vali_l
FROM Xelshekruleba
WHERE vali_l =
(
SELECT MAX(X.vali_l)
FROM   Xelshekruleba AS X
) ;
```

დავუშვათ, ვიცით ფირმის თანამშრომლის გვარი და სახელი - 'სამხარაძე საბა', მაგრამ არ ვიცით მისი კოდი - personaliID და გვაინტერესებს მისი ყველა შეკვეთა. მოთხოვნას ექნება სახე:

```
SELECT * FROM Xelshekruleba
WHERE personaliID =
(
SELECT personaliID
FROM Personali
WHERE gvari = N'სამხარაძე' AND saxeli = N'საბა'
) ;
```

	Id	personaliID	shemkvetiID	gadasaxdeli_l	gadasaxdeli_d	gadaxdili_l
1	1	1	1	5000.0	2631.57999...	5000.0
2	2	1	13	3700.0	1989.25	3000.0
3	11	1	15	9800.0	5444.43999...	9000.0

ჯერ შესრულდა შიგა მოთხოვნა, შემდეგ კი გარე. შიგა მოთხოვნის შესრულების დროს Personali ცხრილიდან ამოირჩევა სტრიქონი, რომელშიც gvari სვეტის მნიშვნელობაა 'სამხარაძე', saxeli სვეტის მნიშვნელობა კი - 'საბა'. შემდეგ ამოირჩევა personaliID სვეტის მნიშვნელობა. ასეთია ერთი სტრიქონი, რომელშიც personaliID = 1. ეს შედეგი მოთავსთდება გარე მოთხოვნის პირობაში შიგა მოთხოვნის ნაცვლად. შედეგად, გარე მოთხოვნის პირობა მიიღებს სახეს:

```
WHERE personaliID = 1
```

ამის შემდეგ შესრულდება გარე მოთხოვნა.

იმისათვის, რომ სკალარული ქვემოთხოვნა კორექტული იყოს, მან ყოველთვის ერთი მნიშვნელობა უნდა გასცეს. თუ სკალარულ ქვემოთხოვნას შეუძლია რამდენიმე მნიშვნელობა გასცეს, მაშინ შესრულების დროს ის შეიძლება ავარიულად დამთავრდეს. ქვემოთ მოყვანილია ქვემოთხოვნა, რომელიც იმ შეკვეთების იდენტიფიკატორებს გასცემს, რომელთა შემსრულებლების გვარები 'გ' ასოთი იწყება. მოთხოვნას აქვს სახე:

```
SELECT  xelshekrulebaID
FROM    Xelshekruleba
```

```
WHERE personaliID =
(
SELECT  P.personaliID
FROM    Personali AS P
WHERE P.gvari LIKE N'გ%'
) ;
```

რადგან ტოლობის ოპერაცია გულისხმობს გამოსახულებას, რომელიც ერთ მნიშვნელობას გასცემს, ამიტომ ქვემოთხოვნა განიხილება, როგორც სკალარული. მაგრამ, რადგან ქვემოთხოვნას შეუძლია რამდენიმე შედეგის გაცემა, ტოლობის ოპერაციისა და სკალარული ქვემოთხოვნის გამოყენება მოცემულ შემთხვევაში არაკორექტულია. თუ ქვემოთხოვნა გასცემს რამდენიმე შედეგს, მაშინ მისი შესრულება ავარიულად დამთავრდება.

როცა სკალარული ქვემოთხოვნა არც ერთ მნიშვნელობას არ გასცემს, მისი შედეგი NULL მნიშვნელობად გარდაიქმნება. როგორც ვიცით, NULL მნიშვნელობასთან შედარება შედეგად UNKNOWN მნიშვნელობას იძლევა და მოცემული მოთხოვნის ფილტრი სტრიქონს არ გასცემს. მაგალითად, Personali ცხრილი არ შეიცავს თანამშრომლებს, რომელთა გვარები 'ა' სიმბოლოთი იწყება. ამიტომ, მოყვანილი მოთხოვნა ცარიელ შედეგობრივ ნაკრებს დაგვიბრუნებს:

```
SELECT xelshekrulebaID
FROM Xelshekruleba
WHERE personaliID =
(
SELECT P.personaliID
FROM Personali AS P
WHERE P.gvari LIKE N'ა%'
);
```

დამოუკიდებელი ქვემოთხოვნები მრავლობითი მნიშვნელობით

ქვემოთხოვნა მრავლობითი მნიშვნელობებით არის ქვემოთხოვნა, რომელიც რამდენიმე (მრავლობით) მნიშვნელობას გასცემს ერთი სვეტის სახით იმისგან დამოუკიდებლად, ბმულია თუ არა ეს ქვემოთხოვნა. ზოგიერთ პრედიკატში, მაგალითად, IN შეგვიძლია გამოვიყენოთ ქვემოთხოვნები მრავლობითი მნიშვნელობებით. IN პრედიკატი გამოიყენება შემდეგი სახით:

<*სკალარული_გამოსახულება*> IN (<*ქვემოთხოვნა_მრავლობითი_მნიშვნელობებით*>)

თუ *სკალარული_გამოსახულება* ერთ-ერთი მნიშვნელობის ტოლია, რომელიც ქვემოთხოვნამ გასცა, მაშინ პრედიკატი TRUE მნიშვნელობას იღებს. რადგან, რამდენიმე თანამშრომლის გვარი შეიძლება ერთი და იმავე ასოთი იწყებოდეს, ამიტომ მოთხოვნაში უნდა გამოვიყენოთ IN პრედიკატი. მაგალითად, მოყვანილი მოთხოვნა დაგვიბრუნებს ხელშეკრულებების იდენტიფიკატორებს (ნომრებს) იმ თანამშრომლებისთვის, რომელთა გვარები 'გ' ასოთი იწყება.

```
SELECT xelshekrulebaID
FROM Xelshekruleba
WHERE personaliID IN
(
SELECT P.personaliID
FROM Personali AS P
WHERE P.gvari LIKE N'გ%'
```

);

რადგან გამოვიყენეთ IN პრედიკატი, ამიტომ ეს მოთხოვნა კორექტულად შესრულდება ქვემოთხოვნის მიერ 0, 1 ან რამდენიმე მნიშვნელობის დაბრუნებისას. მოთხოვნის შესრულების შედეგი:

	xelshekrulebaID
1	5
2	6
3	7
4	9
5	13
6	15
7	16
8	17

ეს ამოცანა შეგვიძლია აგრეთვე გადავწყვიტოთ შეერთების გამოყენებით:

```
SELECT X.xelshekrulebaID
FROM   Personali AS P JOIN Xelshekruleba AS X
        ON P.personaliID = X.personaliID
WHERE P.gvari LIKE N'გ%';
```

მოთხოვნა იგივე შედეგს გასცემს.

ძალიან ბევრი ამოცანა შეიძლება გადავწყვიტოთ როგორც ქვემოთხოვნების, ისე შეერთებების საშუალებით. ერთ შემთხვევაში შეერთების გამოყენებაა უპირატესი, მეორე შემთხვევაში კი - ქვემოთხოვნების.

დავუშვათ, გვინდა დავწეროთ მოთხოვნა, რომელიც დაგვიბრუნებს ხელშეკრულებებს, რომლებიც გააფორმეს თბილისელმა შემკვეთებმა. მოთხოვნას ექნება სახე:

```
SELECT  shemkvetiID, xelshekrulebaID, tarigi_damtavrebis, personaliID
FROM    Xelshekruleba
WHERE shemkvetiID IN
(
SELECT  S.shemkvetiID
FROM    Shemkveti AS S
WHERE S.qalaqi = N'თბილისი'
);
```

ეს მოთხოვნა დაგვიბრუნებს შედეგს:

	xelshekrulebaID	personaliID	shemkvetiID	tarigi_damtavrebis
1	1	2	1	2002-02-01 00:00:00.000
2	8	2	1	2003-04-05 00:00:00.000
3	13	3	1	2007-07-27 00:00:00.000
4	15	3	1	2006-09-03 00:00:00.000

იმავე, როგორც ნებისმიერ სხვა პრედიკატში, ჩვენ IN პრედიკატში შეგვიძლია გამოვიყენოთ უარყოფა - NOT (არა) ლოგიკური ოპერაცია. მაგალითად, მოყვანილი მოთხოვნა დაგვიბრუნებს შემკვეთების იდენტიფიკატორებს, რომლებმაც არც ერთი შეკვეთა არ გააფორმეს:

```
SELECT  ShemkvetiID, firmis_dasaxeleba
```

```
FROM    Shemkveti
WHERE shemkvetiID NOT IN
(
SELECT X.shemkvetiID
FROM   Xelshekruleba AS X
);
```

მოთხოვნა გასცემს ასეთ შედეგს:

	ShemkvetiID	firmis_dasaxeleba
1	8	Givi&Co.
2	9	Gela&Co.
3	10	R&Co.
4	11	NULL
5	12	NULL
6	16	NULL

დამოუკიდებელი ქვემოთხოვნა მრავლობითი მონაცემებით გვიბრუნებს შემკვეთების ყველა იდენტიფიკატორს, რომლებიც Xelshekruleba ცხრილში შეგვხვდა. ბუნებრივია, ამ ცხრილში მოთავსებულია მხოლოდ იმ შემკვეთების იდენტიფიკატორები, რომლებმაც ხელშეკრულება გააფორმეს. გარე მოთხოვნა გასცემს შემკვეთებს Shemkveti ცხრილიდან, რომელთა იდენტიფიკატორები არ არის ჩართული ქვემოთხოვნის მიერ გაცემული მნიშვნელობების სიმრავლეში ანუ გასცემს იმ შემკვეთებს, რომლებმაც არც ერთი ხელშეკრულება არ გააფორმეს.

ბმული ქვემოთხოვნები

ბმული ქვემოთხოვნა არის ისეთი ქვემოთხოვნა, რომელიც გარე მოთხოვნაში გამოყენებული ცხრილის სვეტებს მიმართავს. ეს იმას ნიშნავს, რომ ქვემოთხოვნა დამოკიდებულია გარე მოთხოვნაზე და არ შეუძლია შესრულდეს დამოუკიდებლად. ლოგიკურად, ეს ქვემოთხოვნის ცალკე შესრულების ტოლფასია გარე მოთხოვნის თითოეული სტრიქონისთვის. ბმული ქვემოთხოვნების გამოყენების დროს შიგა მოთხოვნა გარე მოთხოვნაში მითითებული ცხრილის თითოეული სტრიქონისათვის სრულდება. ამ დროს, ჯერ ამოირჩევა გარე მოთხოვნის სტრიქონი, შემდეგ კი მისთვის შესრულდება შიგა მოთხოვნა.

ბმული მოთხოვნების შესრულების ალგორითმია:

1. გარე მოთხოვნის სტრიქონის ამორჩევა.
2. შიგა მოთხოვნის შესრულება.
3. პირობის შეფასება გარე მოთხოვნაში შიგა მოთხოვნის შედეგების საფუძველზე. განისაზღვრება, აირჩევა თუ არა გარე მოთხოვნის სტრიქონი გამოტანისათვის.
4. 1÷3 ბიჯები მეორდება გარე მოთხოვნის ყველა სტრიქონისთვის.

მაგალითები.

ა. გვაინტერესებს ის შემკვეთები, რომლებმაც ხელშეკრულება 2012 წლის 3 მარტს გააფორმეს. პროგრამულ კოდს ექნება სახე:

```
USE Shekveta;
SET DATEFORMAT DMY;
SELECT  *
FROM    Shemkveti S
WHERE '03-03-2012' IN
```

```
(
SELECT  tarigi_dawyebis
FROM    Xelshekruleba X
WHERE S.shemkvetiID = X.shemkvetiID
);
```

	shemkvetiID	iuridiuli_fizikuri	gvari	saxeli	qalaqi	regioni	raioni	sqesi	misamarti
1	1	ფიზიკური	ნემსაძე	დოდო	თბილისი	NULL	ავლაბარი	ქალი	ქეთევან წამებულის 50

მოთხოვნებში გამოყენებული S და X სახელები არის ცხრილების ფსევდონიმები. კერძოდ, S არის Shemkveti ცხრილის ფსევდონიმი, X კი Xelshekruleba ცხრილის ფსევდონიმი. ეს იმას ნიშნავს, რომ Shemkveti და Xelshekruleba სახელების ნაცვლად შეგვიძლია S და X სახელები გამოვიყენოთ. ამ მაგალითში შიგა მოთხოვნა შესრულდება გარე მოთხოვნის თითოეული სტრიქონისათვის, ანუ Shemkveti ცხრილის თითოეული სტრიქონისათვის. ჯერ, გარე მოთხოვნაში ამოირჩევა Shemkveti ცხრილის პირველი სტრიქონი და შემდეგ შესრულდება შიგა მოთხოვნა. შედეგად, Xelshekruleba ცხრილიდან ამოირჩევა ის სტრიქონები, რომელშიც ერთმანეთს ემთხვევა S.shemkvetiID და X.shemkvetiID სვეტების მნიშვნელობები. ეს მნიშვნელობაა 1. შესაბამისად, ამოირჩევა სამი სტრიქონი, რომლებშიც X.shemkvetiID = 1. ამ სამი სტრიქონიდან ორი მათგანის tarigi_dawyebis სვეტის მნიშვნელობა ემთხვევა გარე მოთხოვნის პირობაში მითითებულ მონაცემს, კერძოდ - '03-03-2012'-ს. ამიტომ, გაიცემა გარე მოთხოვნის მიერ ამორჩეული სტრიქონი როგორც საბოლოო შედეგი. შემდეგ, Shemkveti ცხრილიდან ამოირჩევა მეორე სტრიქონი და მეორდება აღნიშნული პროცესი.

ბ. გვაინტერესებს იმ თანამშრომლების გვარები და ნომრები, რომლებსაც ერთზე მეტი შემკვეთი ჰყავთ. მოთხოვნას ექნება სახე:

```
SELECT   P.personaliID, P.gvari, P.saxeli
FROM     Personali P
WHERE  1 <
(
SELECT   COUNT(*)
FROM     Xelshekruleba X
WHERE  P.personaliID = X.personaliID
)
ORDER BY P.personaliID;
```

შედეგი:

	personaliID	gvari	saxeli
1	1	სამხარაძე	საბა
2	2	სამხარაძე	ანა
3	3	გაჩეჩილაძე	ლია
4	8	ყალაბეგიშვილი	გია
5	9	გიორგაძე	გივი

ბმული ქვემოთხოვნების დამოკიდებულება გარე მოთხოვნაზე მათ გამართვას ართულებს დამოუკიდებელ ქვემოთხოვნებთან შედარებით. ბმული მოთხოვნების გამართვისათვის ჩვენ მოგვიწევს კავშირის შეცვლა მუდმივათი და პროგრამული კოდის სისწორის შემოწმების შემდეგ მუდმივას შეცვლა ნამდვილი მიმართვით. უკანასკნელ მაგალითში Personali და Xelshekruleba ცხრილებს შორის კავშირი შევცვალოთ მუდმივათი.

კერძოდ, X.personaliID სვეტს მივანიჭოთ მნიშვნელობა 2. მოთხოვნა მიიღებს სახეს:

```
SELECT   P.personaliID, P.gvari, P.saxeli
FROM     Personali P
WHERE  1 <
(
SELECT   COUNT(*)
FROM     Xelshekruleba X
WHERE  X.personaliID = 2
)
ORDER BY P.personaliID;
```

მოთხოვნამ შედეგი უნდა გასცეს X.personaliID სვეტის შემდეგი მნიშვნელობებისთვის: 1, 2, 3, 8, 9. დანარჩენი მნიშვნელობებისთვის უნდა გაიცეს სტრიქონების ცარიელი სიმრავლე. ამის შემდეგ, შეგვიძლია, აღვადგინოთ რელაცია Personali და Xelshekruleba ცხრილებს შორის.

ლოგიკის ოპერატორები

ALL ოპერატორი

ALL ოპერატორი სკალარულ სიდიდეს ადარებს ქვემოთხოვნის მიერ დაბრუნებულ თითოეულ მნიშვნელობას. თუ ლოგიკური პირობა სრულდება ყველა დაბრუნებული მნიშვნელობისათვის, მაშინ პირობა ჩაითვლება შესრულებულად და გაიცემა TRUE მნიშვნელობა, საწინააღმდეგო შემთხვევაში - FALSE. მისი სინტაქსია:

გამოსახულება { = | <> | != | > | >= | !> | < | <= | !< } ALL (*ქვემოთხოვნა*)

განვიხილოთ არგუმენტების დანიშნულება.

- *გამოსახულება*. ნებისმიერი დასაშვები გამოსახულებაა.
- *ქვემოთხოვნა*. ქვემოთხოვნაა, რომელიც გასცემს ერთი სვეტის მნიშვნელობებს. მნიშვნელობების ტიპი უნდა ემთხვეოდეს *გამოსახულების* ტიპს.

მაგალითი. გვაინტერესებს ყველა ხელშეკრულება თუ შესრულდა. თუ არა, მაშინ რომელი ხელშეკრულება არ შესრულდა. პროგრამულ კოდს აქვს სახე:

```
USE Shekveta;
IF N'კი' = ALL
(
SELECT shesruleba
FROM Xelshekruleba
)
   SELECT N'ყველა ხელშეკრულება შესრულდა';
ELSE
  BEGIN
   SET NOCOUNT ON;
   SELECT X.xelshekrulebaID AS [არ შესრულდა შემდეგი ხელშეკრულებები],
              P.gvari AS [შემსრულებელი], S.gvari AS [შემკვეთი],
              S.firmis_dasaxeleba AS [ფირმის დასახელება]
   FROM   Personali P, Shemkveti S, Xelshekruleba X
   WHERE  X.shesruleba = N'არა'
     AND P.personaliID = X.personaliID
     AND P.personaliID = S.shemkvetiID
```

```
  ORDER BY X.xelshekrulebaID;
END ;
```

შედეგი:

	არ შესრულდა შემდეგი ხელშეკრულებები	შემსრულებელი	შემკვეთი	ფირმის დასახელება
1	2	სამხარაძე	სეხნიაშვილი	Gia&Co.
2	6	გაჩეჩილაძე	მამაიაშვილი	Jemali&Co.
3	9	სამხარაძე	ნემსაძე	NULL
4	10	სამხარაძე	ნემსაძე	NULL
5	14	გაჩეჩილაძე	მამაიაშვილი	Jemali&Co.
6	17	შენგელია	ნონიაშვილი	Zura&Co.
7	18	კაპანაძე	სამხარაძე	NULL
8	21	ყალაბეგიშვილი	კამკამიძე	Givi&Co.

SOME და ANY ოპერატორები

ეს ოპერატორები სკალარულ სიდიდეს ადარებენ ქვემოთხოვნის მიერ გაცემულ თითოეულ მნიშვნელობას. თუ ლოგიკური პირობა სრულდება ერთ-ერთი მათგანისათვის, მაშინ პირობა ჩაითვლება შესრულებულად და გაიცემა TRUE მნიშვნელობა, საწინააღმდეგო შემთხვევაში - FALSE. მათი სინტაქსია:

სკალარული_გამოსახულება { = | <> | != | > | >= | !> | < | <= | !< } ANY (*ქვემოთხოვნა*)

სკალარული_გამოსახულება { = | <> | != | > | >= | !> | < | <= | !< } SOME (*ქვემოთხოვნა*)

ეს ოპერატორები ხშირად გამოიყენება იმის გასარკვევად, არის თუ არა სვეტში საჭირო მნიშვნელობა.

მაგალითი. გვაინტერესებს, რომელიმე ხელშეკრულება შესრულდა თუ არა.

```
IF N'არა' = ANY
(
SELECT shesruleba
FROM Xelshekruleba
)
        SELECT N'ზოგიერთი ხელშეკრულება არ შესრულდა';
ELSE
        SELECT N'ყველა ხელშეკრულება შესრულდა';
```

შედეგი:

	(No column name)
1	ზოგიერთი ხელშეკრულება არ შესრულდა

EXISTS პრედიკატი

EXISTS პრედიკატი შესასვლელზე იღებს ქვემოთხოვნას და გვიბრუნებს TRUE მნიშვნელობას, თუ ქვემოთხოვნა გასცემს თუნდაც ერთ სტრიქონს, და FALSE მნიშვნელობას საწინააღმდეგო შემთხვევაში.

მაგალითები.

ა. შემდეგი მოთხოვნა გასცემს თბილისელ შემკვეთებს, რომლებმაც შეკვეთა გააფორმეს:

```
USE Shekveta;
SELECT shemkvetiID, firmis_dasaxeleba
FROM   Shemkveti AS S
WHERE  qalaqi = N'თბილისი' AND
EXISTS
(
SELECT   *
FROM     Xelshekruleba AS X
WHERE  S.shemkvetiID = X.shemkvetiID
) ;
```

გარე მოთხოვნა ირჩევს თბილისელ შემკვეთებს, რომლებისთვისაც EXISTS პრედიკატი TRUE მნიშვნელობას გასცემს. EXISTS პრედიკატი TRUE მნიშვნელობას გასცემს, თუ მიმდინარე შემკვეთს აქვს მასთან ბმული ხელშეკრულებები Xelshekruleba ცხრილში.

მოცემული მოთხოვნა დაგვიბრუნებს შედეგს:

	shemkvetiID	firmis_dasaxeleba
1	1	NULL
2	12	NULL
3	14	NULL
4	15	სთედი ლოჯიკ

ისევე, როგორც სხვა პრედიკატებში, EXISTS პრედიკატში შეგვიძლია გამოვიყენოთ NOT (არა) ლოგიკური ოპერაცია.

ბ. შემდეგი მოთხოვნა დაგვიბრუნებს თბილისელ კლიენტებს, რომლებმაც არც ერთი შეკვეთა არ გააფორმეს:

```
USE Shekveta;
SELECT   shemkvetiID, firmis_dasaxeleba
FROM     Shemkveti AS S
WHERE  qalaqi = N'თბილისი' AND
NOT EXISTS
(
SELECT   *
FROM     Xelshekruleba AS X
WHERE  S.shemkvetiID = X.shemkvetiID
) ;
```

შედეგი:

	shemkvetiID	firmis_dasaxeleba
1	10	R&Co.
2	11	NULL

გ. გვაინტერესებს ის თანამშრომლები, რომლებსაც რამდენიმე შემკვეთი ჰყავთ. მოთხოვნას ექნება სახე:

```
SELECT DISTINCT P.personaliID, P.gvari, P.saxeli, P.qalaqi
FROM   Personali P JOIN Xelshekruleba X1
          ON P.personaliID = X1.personaliID
WHERE EXISTS
```

```
(
SELECT   *
FROM     Xelshekruleba X2
WHERE  X1.personaliID    =  X2.personaliID
  AND     X1.shemkvetiID <> X2.shemkvetiID
  AND     P.personaliID      =  X1.personaliID
);
```

	personaliID	gvari	saxeli	qalaqi
1	1	სამწარაძე	საბა	ზუგდიდი
2	2	სამწარაძე	ანა	ბათუმი
3	3	გაჩეჩილაძე	ლია	თბილისი
4	8	ყალაბეგიშვილი	გია	ქუთაისი
5	9	გიორგაძე	გივი	ქობულეთი

გარე მოთხოვნის თითოეული სტრიქონისთვის (ესაა თანამშრომელი, რომელსაც მოცემულ მომენტში ვამოწმებთ), შიგა მოთხოვნა პოულობს იმ სტრიქონებს, რომლებიც ემთხვევა personaliID სვეტის მნიშვნელობას (რომელიც ჰქონდა თანამშრომელს), და არ ემთხვევა shemkvetiID სვეტის მნიშვნელობას (რომელიც სხვა შემკვეთებს შეესაბამება). თუ შიგა მოთხოვნის მიერ მოიძებნა ასეთი სტრიქონები, ეს იმას ნიშნავს, რომ არის ორი სხვადასხვა შემკვეთი, რომლებსაც ერთი თანამშრომელი ემსახურება. ამიტომ, EXISTS ოპერატორი ნამდვილია მიმდინარე სტრიქონისათვის და თანამშრომლის ნომერი (personaliID) იქნება გამოტანილი. DISTINCT სიტყვა რომ არ იყოს მითითებული, მაშინ თითოეული ეს თანამშრომელი ამორჩეული იქნება იმდენჯერ, რამდენი შემკვეთიც ჰყავს.

დ. გვაინტერესებს თანამშრომელები, რომლებსაც ერთი შემკვეთი ჰყავთ. მოთხოვნას ექნება სახე:

```
SELECT DISTINCT P.personaliID, P.gvari, P.saxeli, P.qalaqi
FROM   Personali P JOIN Xelshekruleba X1
          ON P.personaliID = X1.personaliID
WHERE NOT EXISTS
(
SELECT   *
FROM     Xelshekruleba X2
WHERE  X1.personaliID    =  X2.personaliID
  AND     X1.shemkvetiID <> X2.shemkvetiID
  AND     P.personaliID      =  X1.personaliID );
```

	personaliID	gvari	saxeli	qalaqi
1	4	გაჩეჩილაძე	ნათუნა	თბილისი
2	5	შენგელია	ნატა	ზუგდიდი
3	6	კაპანაძე	ეთერი	რუსთავი
4	7	ქევხიშვილი	ალექსანდრე	თბილისი
5	10	კიკნაძე	მზია	ბათუმი

თავი 8. ოპერაციები სიმრავლეებზე

სტრიქონების სიმრავლეებზე ოპერაციები სტრიქონების ორ შესასვლელ (საწყის) სიმრავლეზე ან სტრიქონების ორ მულტისიმრავლეზე სრულდება. შედეგად, ასევე მულტისიმრავლე (სტრიქონების გამოსასვლელი სიმრავლე, სტრიქონების შედეგობრივი ნაკრები) მიიღება. შესასვლელი მულტისიმრავლეები სიმრავლეებზე ოპერაციებში მონაწილე მოთხოვნების შესრულების შედეგად გაიცემა. სტრიქონების მულტისიმრავლე შეიძლება ერთნაირ სტრიქონებს შეიცავდეს.

T-SQL ენაში სტრიქონების სიმრავლეებზე განსაზღვრულია სამი ოპერაცია: UNION, INTERSECT და EXCEPT. მათი სინტაქსია:

მოთხოვნა 1
<სიმრავლეებზე_შესასრულებელი_ოპერაცია>
მოთხოვნა 2
[ORDER BY . . .]

შევნიშნოთ, რომ ორივე მოთხოვნაში დაუშვებელია ORDER BY საკვანძო სიტყვის გამოყენება. მისი გამოყენება დასაშვებია მხოლოდ საბოლოო შედეგის მიმართ.

ორივე მოთხოვნის მიერ გაცემულ თითოეულ შედეგობრივ ნაკრებში სვეტების რაოდენობა და მიმდევრობა ერთნაირი უნდა იყოს. შესაბამის სვეტებს მონაცემთა თავსებადი ტიპები უნდა ჰქონდეს. მონაცემთა თავსებად ტიპებში ქვეშ იგულისხმება ტიპების არაცხადი გარდაქმნა. ტიპების თავსებადობა განისაზღვრება შემდეგი ცხრილის მიხედვით:

ცხრილი 8.1. სვეტების ტიპების თავსებადობა

სვეტის მონაცემთა ტიპი	შედეგის ტიპი
ორივე სვეტი CHAR ტიპისაა L1 და L2 ფიქსირებული სიგრძეებით	NVARCHAR ტიპი, რომლის სიგრძე L1-სა და L2-ს შორის უდიდესის ტოლია
ორივე სვეტი BINARY ტიპისაა L1 და L2 ფიქსირებული სიგრძეებით	NVARCHAR ტიპი, რომლის სიგრძე L1-სა და L2-ს შორის უდიდესის ტოლია
ერთი ან ორივე სვეტი VARCHAR ტიპისაა	NVARCHAR ტიპი, რომლის სიგრძე L1-სა და L2-ს შორის უდიდესის ტოლია
ერთი ან ორივე სვეტი VARBINARY ტიპისაა	NVARCHAR ტიპი, რომლის სიგრძე L1-სა და L2-ს შორის უდიდესის ტოლია
ორივე სვეტი რიცხვითი ტიპისაა (SMALLINT, MONEY, FLOAT)	უდიდესი სიზუსტის მქონე მონაცემის ტიპი

სიმრავლეებზე ოპერაციის მიერ გაცემულ შედეგობრივ ნაკრებში სვეტების სახელები პირველი მოთხოვნით განისაზღვრება. ამიტომ, თუ შედეგობრივ სვეტებს უნდა მივანიჭოთ ფსევდონიმები, ეს პირველ მოთხოვნაში უნდა გავაკეთოთ.

უნდა გავითვალისწინოთ, აგრეთვე სიმრავლეებზე ოპერაციების ის თავისებურება, რომ სტრიქონების შედარებისას სიმრავლეებზე ოპერაცია ორ NULL მნიშვნელობას ერთნაირად თვლის.

ANSI SQL სტანდარტი უზრუნველყოფს სტრიქონების სიმრავლეებზე თითოეული

ოპერაციის ორ სახესხვაობას: DISTINCT (ნაგულისხმევი) და ALL. DISTINCT ელემენტი ორი შესასვლელი მულტისიმრავლიდან ერთნაირ სტრიქონებს შლის და შედეგად სიმრავლე გაიცემა, ALL ელემენტი კი - ერთნაირ სტრიქონებს არ შლის და შედეგად მულტისიმრავლე გაიცემა. DISTINCT ელემენტი უზრუნველყოფილია UNION, INTERSECT და EXCEPT ოპერაციებისათვის, ALL ელემენტი კი - მხოლოდ UNION ოპერაციისთვის. DISTINCT ელემენტს სინტაქსურად ვერ მივუთითებთ, რადგან ის იგულისხმება მაშინ, როცა არ არის მითითებული ALL ელემენტი.

UNION ოპერაცია

სიმრავლეთა თეორიაში ორი სიმრავლის გაერთიანება არის სიმრავლე, რომელიც ორივე სიმრავლის ყველა ელემენტს შეიცავს. სხვა სიტყვებით, თუ ელემენტი ეკუთვნის ერთ-ერთ შესასვლელ სიმრავლეს, მაშინ ის ეკუთვნის შედეგობრივ სიმრავლესაც. T-SQL ენაში UNION ოპერაცია აერთიანებს ორი მოთხოვნის მიერ გაცემული სტრიქონების შედეგობრივ ნაკრებებს. თუ სტრიქონი ეკუთვნის ერთ-ერთ შედეგობრივ ნაკრებს, მაშინ ის გამოჩნდება UNION ოპერაციის მიერ გაცემულ შედეგშიც.

UNION ALL ოპერაცია

UNION ALL ოპერაცია აერთიანებს რამდენიმე მოთხოვნის შესრულების შედეგს გასცემს ყველა სტრიქონს, რომელიც გვხვდება შედეგობრივ მულტისიმრავლეში ერთნაირი სტრიქონების გამორიცხვის გარეშე. თუ *მოთხოვნა 1* გასცემს m სტრიქონს, ხოლო *მოთხოვნა 2* კი – n სტრიქონს, მაშინ

მოთხოვნა 1
UNION ALL
მოთხოვნა 2
გასცემს m + n სტრიქონს.

მაგალითი. ქვემოთ მოყვანილ პროგრამის კოდში UNION ALL ოპერაცია ორი მოთხოვნის მიერ გაცემული სტრიქონების მულტისიმრავლეზე სრულდება. პირველი მოთხოვნა Personali ცხრილიდან qalaqi და raioni სვეტებს ირჩევს, მეორე კი - Shemkveti ცხრილიდან qalaqi და raioni სვეტებს. მოთხოვნას აქვს სახე:

```
USE Shekveta;
SELECT qalaqi, raioni FROM Personali
UNION ALL
SELECT qalaqi, raioni FROM Shemkveti;
```

შედეგი შეიცავს 26 სტრიქონს, 11 სტრიქონს Perssonali ცხრილიდან, 15 სტრიქონს კი - Shemkveti ცხრილიდან. ჯერ გამოჩნდება Personali ცხრილის სტრიქონები, შემდეგ კი - Shemkveti ცხრილის სტრიქონები.

შედეგი:

	qalaqi	raioni
1	ზუგდიდი	NULL
2	ბათუმი	NULL
3	თბილისი	საბურთალო
4	თბილისი	ვერა
5	ზუგდიდი	NULL
6	რუსთავი	NULL
7	თბილისი	ვერა
8	ქუთაისი	NULL
9	ქობულეთი	NULL
10	ბათუმი	NULL
11	თბილისი	ვაკე
12	თბილისი	ავლაბარი
13	თელავი	NULL
14	თელავი	NULL
15	ქუთაისი	NULL
16	ბათუმი	NULL
17	ბათუმი	NULL
18	ზესტაფონი	NULL
19	ზუგდიდი	NULL
20	გორი	NULL
21	თბილისი	საბურთალო
22	თბილისი	მთაწმინდა
23	თბილისი	დიღომი
24	თელავი	NULL
25	თბილისი	დიდუბე
26	თბილისი	დიღომი

რადგან, UNION ALL ოპერაცია ერთნაირ სტრიქონებს ტოვებს, ამიტომ შედეგი მულტისიმრავლეს წარმოადგენს. შედეგში ერთი და იგივე სტრიქონი შეიძლება ბევრჯერ შეგვხვდეს. მაგალითად, სტრიქონი - (*თბილისი, ვერა*) - ორჯერ გვხვდება.

UNION ოპერაცია

UNION ოპერაცია აერთიანებს რამდენიმე მოთხოვნის შესრულების შედეგს, შედეგიდან შლის ერთნაირ სტრიქონებს და გასცემს სიმრავლეს, რომელიც ორივე შესასვლელ სიმრავლეში შემავალი სტრიქონებისაგან შედგება.

მაგალითი. მოყვანილი პროგრამული კოდი გასცემს თანამშრომლებისა და შემკვეთების ერთმანეთისაგან განსხვავებულ ადგილმდებარეობებს (ქალაქი, რაიონი):

```
USE Shekveta;
SELECT qalaqi, raioni FROM Personali
UNION
SELECT qalaqi, raioni FROM Shemkveti;
```

შედეგიდან წაშლილია ერთნაირი სტრიქონები. ის შეიცავს 15 ერთმანეთისაგან განსხვავებულ სტრიქონს:

	qalaqi	raioni
1	ბათუმი	NULL
2	გორი	NULL
3	ზესტაფონი	NULL
4	ზუგდიდი	NULL
5	თბილისი	ავლაბარი
6	თბილისი	დიდუბე
7	თბილისი	დიღომი
8	თბილისი	ვაკე
9	თბილისი	ვერა
10	თბილისი	მთაწმინდა
11	თბილისი	საბურთალო
12	თელავი	NULL
13	რუსთავი	NULL
14	ქობულეთი	NULL
15	ქუთაისი	NULL

INTERSECT ოპერაცია

სიმრავლეთა თეორიაში ორი სიმრავლის გადაკვეთა არის ყველა იმ ელემენტების ერთობლიობა, რომლებიც ორივე სიმრავლეს ეკუთვნის. INTERSECT ოპერაცია იღებს ორი მოთხოვნის მიერ გაცემული სტრიქონების შედეგობრივ ნაკრებს და გასცემს მხოლოდ იმ სტრიქონებს, რომლებიც გვხვდება ორივე შესასვლელ ნაკრებში.

INTERSECT ოპერაცია ჯერ ერთნაირ სტრიქონებს წაშლის ორივე შესასვლელი მულტისიმრავლიდან, შემდეგ კი გასცემს მხოლოდ იმ სტრიქონებს, რომლებიც გვხვდება ორივე სიმრავლეში. სხვა სიტყვებით, რომ ვთქვათ, სტრიქონი გაიცემა იმ შემთხვევაში, თუ ის ერთხელ მაინც გვხვდება ორივე მულტისიმრავლეში.

მაგალითი. მოყვანილი პროგრამული კოდი გასცემს თანამშრომლებისა და შემკვეთების ერთმანეთისაგან განსხვავებულ საერთო ადგილმდებარეობას (ქალაქი, რაიონი):

```
USE Shekveta;
SELECT qalaqi, raioni FROM Personali
INTERSECT
SELECT qalaqi, raioni FROM Shemkveti;
```

შედეგი:

	qalaqi	raioni
1	ბათუმი	NULL
2	ზუგდიდი	NULL
3	თბილისი	საბურთალო
4	ქუთაისი	NULL

შედეგიდან ჩანს, რომ არის ოთხი საერთო ქალაქი, სადაც ცხოვრობენ თანამშრომლებიც და შემკვეთებიც. უნდა გვახსოვდეს, რომ სტრიქონების შედარებისას, სიმრავლეებზე ოპერაცია ორ NULL მნიშვნელობას განიხილავს როგორც ტოლს და გასცემს შესაბამის სტრიქონს.

EXCEPT ოპერაცია

A და B სიმრავლეების სხვაობა არის იმ ელემენტების სიმრავლე, რომლებიც ეკუთვნის A-ს და არ ეკუთვნის B-ს. სიმრავლეების სხვაობა შეგვიძლია განვიხილოთ როგორც A სიმრავლე B სიმრავლის ელემენტების გარეშე. EXCEPT ოპერაცია ზემოქმედებს ორი მოთხოვნის მიერ გაცემული სტრიქონების შედეგობრივ ნაკრებზე და გასცემს იმ სტრიქონებს, რომლებიც გვხვდება პირველ ნაკრებში, მაგრამ არ გვხვდება მეორეში.

EXCEPT ოპერაცია ჯერ გამორიცხავს ერთნაირ სტრიქონებს ორივე შესასვლელი მულტისიმრავლიდან, შემდეგ კი - გასცემს მხოლოდ იმ სტრიქონებს, რომლებიც გვხვდება პირველ სიმრავლეში, მაგრამ არ გვხვდება მეორეში. UNION და INTERSECT ოპერაციებისგან განსხვავებით EXCEPT ოპერაცია ასიმეტრიულია, ე.ი. UNION და INTERSECT ოპერაციებისთვის მნიშვნელობა არ აქვს, თუ რომელი მოთხოვნაა პირველი და რომელი მეორე, EXCEPT ოპერაციის შემთხვევაში კი - ამას მნიშვნელობა აქვს.

მაგალითი. მოყვანილი პროგრამის კოდი გასცემს თანამშრომლების სხვადასხვა ადგილმდებარეობას, რომელიც არ ემთხვევა შემკვეთების ადგილმდებარეობას (ქალაქი, რაიონი):

```
USE Shekveta;
SELECT qalaqi, raioni FROM Personali
EXCEPT
SELECT qalaqi, raioni FROM Shemkveti;
```

შედეგი:

Results | Messages

	qalaqi	raioni
1	თბილისი	დიდუბე
2	თბილისი	ვერა
3	რუსთავი	NULL
4	ქობულეთი	NULL

მოყვანილი პროგრამის კოდი გასცემს შემკვეთების სხვადასხვა ადგილმდებარეობას, რომელიც არ ემთხვევა თანამშრომლების ადგილმდებარეობას (ქალაქი, რაიონი):

```
USE Shekveta
SELECT qalaqi, raioni FROM Shemkveti
EXCEPT
SELECT qalaqi, raioni FROM Personali;
```

შედეგი:

	qalaqi	raioni
1	გორი	NULL
2	ზესტაფონი	NULL
3	თბილისი	ავლაბარი
4	თბილისი	დიდუბე
5	თბილისი	დიღომი
6	თბილისი	მთაწმინდა
7	თელავი	NULL

პრიორიტეტი

სიმრავლეებზე ოპერაციებისთვის განსაზღვრულია პრიორიტეტები. INTERSECT ოპერაციას უფრო მაღალი პრიორიტეტი აქვს, ვიდრე UNION და EXCEPT ოპერაციებს, ხოლო ამ ორ ოპერაციას კი - ერთნაირი. მოთხოვნაში, რომელიც შეიცავს სტრიქონების სიმრავლეებზე რამდენიმე ოპერაციას, პირველად შესრულდება INTERSECT ოპერაცია, შემდეგ კი ერთნაირი პრიორიტეტის ოპერაციები მათი მიმდევრობის მიხედვით.

მაგალითი. გვაინტერესებს იმ შემკვეთების ადგილმდებარეობა, რომლებიც ფიზიკურ პირებს წარმოადგენენ და რომელთა ადგილმდებარეობა, არ ემთხვევა თანამშრომლებისა და იმ შემკვეთების საერთო ადგილმდებარეობას, რომლებიც იურიდიულ პირებს წარმოადგენენ. პროგრამულ კოდს აქვს სახე:

```
USE Shekveta;
SELECT qalaqi, raioni FROM Shemkveti WHERE iuridiuli_fizikuri = N'ფიზიკური'
EXCEPT
SELECT qalaqi, raioni FROM Personali
INTERSECT
SELECT qalaqi, raioni FROM Shemkveti WHERE iuridiuli_fizikuri = N'იურიდიული';
```

შედეგი:

	qalaqi	raioni
1	ზესტაფონი	NULL
2	თბილისი	ავლაბარი
3	თბილისი	დიდუბე
4	თბილისი	დიღომი
5	თბილისი	მთაწმინდა
6	თელავი	NULL
7	ქუთაისი	NULL

ამ მოთხოვნაში ჯერ შესრულდება INTERSECT, შემდეგ კი - EXCEPT ოპერაცია.

სიმრავლეებზე ოპერაციების შესრულების მართვისთვის, მაგალითად, პრიორიტეტის შესაცვლელად, გამოიყენება მრგვალი ფრჩხილები, რომლებსაც უმაღლესი პრიორიტეტი აქვს.

მაგალითი. გვაინტერესებს იმ შემკვეთების ადგილმდებარეობა, რომლებიც ფიზიკურ პირებს წარმოადგენენ და რომელთა ადგილმდებარეობა არ ემთხვევა იმ შემკვეთების ადგილმდებარეობას, რომლებიც იურიდიულ პირებს წარმოადგენენ, მაგრამ ემთხვევა თანამშრომლების ადგილმდებარეობას. პროგრამულ კოდს აქვს სახე:

```
USE Shekveta;
(
SELECT qalaqi, raioni FROM Shemkveti WHERE iuridiuli_fizikuri = N'ფიზიკური'
EXCEPT
SELECT qalaqi, raioni FROM Shemkveti WHERE iuridiuli_fizikuri = N'იურიდიული'
)
INTERSECT
SELECT qalaqi, raioni FROM Personali;
```

შედეგი:

	qalaqi	raioni
1	ქუთაისი	NULL

თავი 9. ფუნქციები

მომხმარებლის ფუნქციები

მომხმარებლის მიერ შემუშავებული ფუნქციები წარმოადგენს მონაცემთა ბაზის დამოუკიდებელ ობიექტებს და მოთავსებულა შესაბამის მონაცემთა ბაზაში. არსებობს მომხმარებლების მიერ შექმნილი ფუნქციების სამი კლასი:

– Scalar ტიპის ფუნქციები იმით ხასიათდება, რომ გასცემს 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.

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

– Multi-statement ტიპის ფუნქციები გასცემენ table ტიპის მნიშვნელობას, რომელიც მონაცემებს შეიცავს. Inline ფუნქციისგან განსხვავებით Multi-statement ფუნქციის ტანი შეიძლება რამდენიმე ბრძანებას შეიცავდეს.

ამ ფუნქციებს შეიძლება ჰქონდეთ ერთი ან მეტი პარამეტრი, ან არც ერთი. პარამეტრის ტიპი არ უნდა იყოს timestamp (rowversion), cursor და table. ფუნქციის ტანში დასაშვებია ლოკალური ცვლადების გამოცხადება, სხვა ფუნქციებისა და შენახული პროცედურების გამოძახება და კურსორების შექმნა.

ფუნქციის კოდში INSERT, UPDATE და DELETE ბრძანებების გამოყენებისას არსებობს შეზღუდვა - დასაშვებია მუშაობა მხოლოდ იმ მონაცემებთან, რომელიც ინახება ფუნქციის ტანში შექმნილი table ტიპის ცვლადში. ფუნქციის ტანში დაუშვებელია PRINT და SELECT ბრძანებების გამოყენება მონაცემების უშუალოდ დაბრუნების მიზნით.

Scalar ტიპის ფუნქციები

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

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

განვიხილოთ არგუმენტების დანიშნულება.

– *სქემის_სახელი* არგუმენტი მიუთითებს იმ სქემის სახელს, რომელსაც ფუნქცია ეკუთვნის (სქემებს 21 თავში განვიხილავთ).

– *ფუნქციის_სახელი* არის შესაქმნელი ფუნქციის სახელი, რომელიც უნდა აკმაყოფილებდეს ობიექტის სახელდების სტანდარტულ წესებს.

– *@პარამეტრის_სახელი პარამეტრის_ტიპი* [= DEFAULT] არგუმენტი ფუნქციის

პარამეტრებს განსაზღვრავს. თითოეული პარამეტრის სახელი უნდა იყოს უნიკალური და იწყებოდეს @ სიმბოლოთი. სახელის შემდეგ ეთითება პარამეტრის ტიპი. დასაშვებია ყველა ტიპი, გარდა timestamp (rowversion), cursor და table ტიპებისა. DEFAULT არგუმენტი მიუთითებს ნაგულისხმევ მნიშვნელობას, რომელიც ავტომატურად მიენიჭება პარამეტრს, თუ მისი მნიშვნელობა აშკარად არ იქნება მითითებული. პარამეტრები ერთმანეთისაგან მძიმეებით გამოიყოფა.

– WITH { ENCRYPTION | SCHEMABINDING } არგუმენტი დამატებით შესაძლებლობებს განსაზღვრავს.

ENCRYPTION საკვანძო სიტყვის გამოყენება იწვევს CREATE FUNCTION ბრძანების კოდის დაშიფვრას. ეს კოდი ინახება მიმდინარე მონაცემთა ბაზის syscomments სისტემური წარმოდგენის text სვეტში. თუ ფუნქციის კოდი დაშიფრული არ არის, მაშინ მისი ნახვა SELECT ბრძანებით შეიძლება: SELECT text FROM syscomments.

SCHEMABINDING არგუმენტი კრძალავს იმ ობიექტების შეცვლას ან წაშლას, რომელსაც ფუნქცია მიმართავს. ფუნქცია შეიძლება მიმართავდეს მონაცემთა ბაზის სხვადასხვა ობიექტს. მათი შეცვლა ან წაშლა გამოიწვევს ფუნქციის შესრულების შეფერხებას.

– AS საკვანძო სიტყვას მოსდევს ფუნქციის ტანი.

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

– RETURN *სკალარული_გამოსახულება* არგუმენტი ფუნქციის მუშაობას ამთავრებს და შედეგს გასცემს.

მაგალითები.

ა. შევადგინოთ ფუნქცია, რომელიც გამოთვლის მითითებული განყოფილების მაქსიმალურ ხელფასს. ფუნქციას პარამეტრად განყოფილების სახელი გადაეცემა. პროგრამის კოდს აქვს სახე:

```
USE Shekveta;
GO
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
);
RETURN @Max_xelfasi;
END
```

ფუნქციის გამოსაძახებლად შეგვაქვს მოთხოვნა:

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

შედეგი:

	განყოფილების მაქსიმალური ხელფასი
1	1020,43278

ბ. შევადგინოთ ფუნქცია, რომელიც მითითებულ ქალაქში მცხოვრები და მითითებულ

განყოფილებაში მომუშავე თანამშრომლების საშუალო ხელფასს გამოთვლის. ფუნქციას ორი პარამეტრი აქვს. პროგრამის კოდს აქვს სახე:

```
USE Shekveta;
GO
CREATE FUNCTION Sahualo_Xelfasi( @qalaqi NVARCHAR(30), @ganyofileba NVARCHAR(30) )
RETURNS FLOAT
AS
BEGIN
DECLARE @Sahualo_xelfasi FLOAT;
SET @Sahualo_xelfasi =
(
SELECT AVG(xelfasi)
FROM   Personali
WHERE qalaqi = @qalaqi AND ganyofileba = @ganyofileba
);
RETURN @Sahualo_xelfasi;
END
```

ფუნქციის გამოსაძახებლად შეგვაქვს მოთხოვნა:

```
SELECT dbo.Sahualo_Xelfasi(N'თბილისი', N'სასოფლო') AS [განყოფილების საშუალო ხელფასი];
```

შედეგი:

	განყოფილების საშუალო ხელფასი
1	935,766325

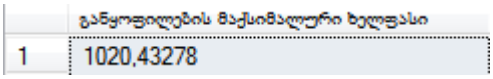
გ. შევადგინოთ ფუნქცია, რომელიც გამოთვლის მითითებული განყოფილების მაქსიმალურ ხელფასს. ნაგულისხმევ მნიშვნელობად მითითებულია 'სავაჭრო'. პროგრამის კოდს აქვს სახე:

```
USE Shekveta;
GO
CREATE FUNCTION Maqsimaluri_Xelfasi_1( @ganyofileba NVARCHAR(30) = N'სავაჭრო' )
RETURNS FLOAT
AS
BEGIN
DECLARE @Max_xelfasi FLOAT ;
SET @Max_xelfasi =
(
SELECT MAX(xelfasi)
FROM   Personali
WHERE ganyofileba = @ganyofileba
);
RETURN @Max_xelfasi;
END
```

ფუნქციის გამოსაძახებლად შეგვაქვს მოთხოვნა:

```
SELECT dbo.Maqsimaluri_Xelfasi_1(DEFAULT)  AS [განყოფილების მაქსიმალური ხელფასი];
```

შედეგი:

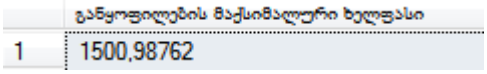


	განყოფილების მაქსიმალური ხელფასი
1	1020,43278

ამ ფუნქციას გამოძახების დროს შეგვიძლია პარამეტრიც გადავცეთ:

```
SELECT dbo.Maqsimaluri_Xelfasi_1(N'სასპორტო')  AS [განყოფილების მაქსიმალური ხელფასი];
```

შედეგი:



	განყოფილების მაქსიმალური ხელფასი
1	1500,98762

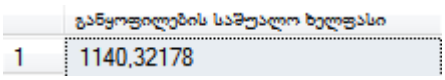
დ. შევადგინოთ ფუნქცია, რომელიც მითითებულ ქალაქში მცხოვრები და მითითებულ განყოფილებაში მომუშავე თანამშრომლების საშუალო ხელფასს გამოთვლის. ფუნქციას ორი პარამეტრი აქვს. მეორე პარამეტრი არის ნაგულისხმევი. პროგრამის კოდს აქვს სახე:

```
USE Shekveta;
GO
CREATE FUNCTION Sahualo_Xelfasi_1( @qalaqi NVARCHAR(30),
                                   @ganyofileba NVARCHAR(30)  = N'სამედიცინო')
RETURNS FLOAT
AS
BEGIN
DECLARE @Sahualo_xelfasi FLOAT;
SET @Sahualo_xelfasi =
(
SELECT AVG(xelfasi)
FROM   Personali
WHERE qalaqi = @qalaqi AND ganyofileba = @ganyofileba
);
RETURN @Sahualo_xelfasi;
END
```

ფუნქციის გამოსაძახებლად შეგვაქვს მოთხოვნა:

```
SELECT dbo.Sahualo_Xelfasi_1(N'თბილისი', DEFAULT) AS [განყოფილების საშუალო ხელფასი];
```

შედეგი:

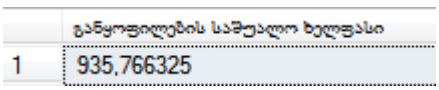


	განყოფილების საშუალო ხელფასი
1	1140,32178

ამავე ფუნქციის გამოძახებისას, მეორე პარამეტრი შეგვიძლია აშკარად მივუთითოთ:

```
SELECT dbo.Sahualo_Xelfasi_1(N'თბილისი', N'სასოფლო')
  AS [განყოფილების საშუალო ხელფასი];
```

შედეგი:



	განყოფილების საშუალო ხელფასი
1	935,766325

ე. შევადგინოთ ფუნქცია, რომელიც გამოთვლის მითითებული განყოფილების მაქსიმალურ ხელფასს. ფუნქციას პარამეტრად განყოფილების სახელი გადაეცემა. ფუნქციის შექმნისას გამოიყენება SCHEMABINDING საკვანძო სიტყვა. პროგრამის კოდს აქვს სახე:

```
USE Shekveta;
GO
CREATE FUNCTION Sahualo_Xelfasi_3( @qalaqi NVARCHAR(30),
@ganyofileba NVARCHAR(30)  = N'სამედიცინო')
RETURNS FLOAT
WITH  SCHEMABINDING
AS
BEGIN
DECLARE @Sahualo_xelfasi FLOAT;
SET @Sahualo_xelfasi =
(
SELECT AVG(xelfasi)
FROM   dbo.Personali
WHERE qalaqi = @qalaqi AND ganyofileba = @ganyofileba
);
RETURN @Sahualo_xelfasi;
END
```

ფუნქციის გამოსაძახებლად შეგვაქვს მოთხოვნა:

```
SELECT dbo.Sahualo_Xelfasi_3(N'სამედიცინო', DEFAULT)
```

AS [განყოფილების საშუალო ხელფასი];

შედეგი:

	განყოფილების საშუალო ხელფასი
1	935,766325

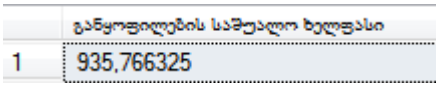
3. შევადგინოთ ფუნქცია, რომელიც გამოთვლის მითითებული განყოფილების მაქსიმალურ ხელფასს. ფუნქციას პარამეტრად განყოფილების სახელი გადაეცემა. ფუნქციის კოდის დაშიფვრის მიზნით გამოიყენება ENCRYPTION საკვანძო სიტყვა. პროგრამის კოდს აქვს სახე:

```
USE Shekveta;
GO
CREATE FUNCTION Sahualo_Xelfasi_4( @qalaqi NVARCHAR(30),
@ganyofileba NVARCHAR(30)  = N'სამედიცინო')
RETURNS FLOAT
WITH ENCRYPTION
AS
BEGIN
DECLARE @Sahualo_xelfasi FLOAT;
SET @Sahualo_xelfasi =
(
SELECT AVG(xelfasi)
FROM   dbo.Personali
WHERE qalaqi = @qalaqi AND ganyofileba = @ganyofileba
);
RETURN @Sahualo_xelfasi ;
END
```

ფუნქციის გამოსაძახებლად შეგვაქვს მოთხოვნა:

```
SELECT dbo.Sahualo_Xelfasi_4(N'სამედიცინო', DEFAULT)
    AS [განყოფილების საშუალო ხელფასი];
```

შედეგი:



	განყოფილების საშუალო ხელფასი
1	935,766325

Inline ტიპის ფუნქციები

ამ ტიპის ფუნქცია იქმნება CREATE FUNCTION ბრძანებით, რომლის სინტაქსია:

```
CREATE FUNCTION [ სქემის_სახელი.] ფუნქციის_სახელი
( [ { @პარამეტრის_სახელი პარამეტრის_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 დინამიკურად იქმნება მოთხოვნის შესრულების დროს. გაცემული table მნიშვნელობა შეგვიძლია უშუალოდ გამოვიყენოთ SELECT მოთხოვნის FROM განყოფილებაში ფუნქციის სახელის მითითების გზით.

მაგალითი.

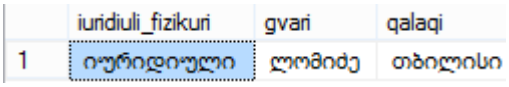
ა. შევადგინოთ ფუნქცია, რომელიც დაგვიბრუნებს ინფორმაციას მითითებულ ქალაქში მყოფი შემკვეთების შესახებ. პროგრამის კოდს შემდეგი სახე აქვს:

```
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 2;
```

შედეგი:



	iuridiuli_fizikuri	gvari	qalaqi
1	იურიდიული	ლომიძე	თბილისი

ბ. გამოვთვალოთ საშუალო ხელფასი განყოფილებების მიხედვით. პროგრამის კოდს შემდეგი სახე აქვს:

```
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;
```

შედეგი:

	განყოფილება	საშუალო ხელფასი
1	სასოფლო	1994,0061182176
2	სასპორტო	2118,57035816384
3	სავაჭრო	2201,0026194704
4	სამედიცინო	4001,00531766406

Multi-statement ტიპის ფუნქციები

ამ ტიპის ფუნქცია იქმნება CREATE FUNCTION ბრძანებით, რომლის სინტაქსია:

```
CREATE FUNCTION [ სქემის_სახელი.] ფუნქციის_სახელი
( [ { @პარამეტრის_სახელი პარამეტრის_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.

როგორც ვიცით, ფუნქცია მუშაობას ამთავრებს მაშინ, როცა გვხვდება RETURN საკვანძო სიტყვა. 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.

მაგალითი.

ა) 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. პროგრამის კოდს შემდეგი სახე აქვს:

```
CREATE FUNCTION Funqcia_1(@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 Funqcia_1(15, N'საბა');
```

შედეგი:

	id1	num	val
1	1	25	საბა
2	2	35	საბა
3	3	45	საბა

ბ. შევასრულოთ წინა მაგალითში მოყვანილი მოთხოვნა. შემდეგ მესამე სტრიქონში მეორე სვეტის მნიშვნელობა 5-ით გავზარდოთ. წავშალოთ მეორე სტრიქონი. პროგრამის კოდს შემდეგი სახე აქვს:

```
CREATE FUNCTION Funqcia_2(@ricxvi INT, @striqoni NVARCHAR(20))
RETURNS @table1 TABLE
(
id        INT PRIMARY KEY IDENTITY (1,1) NOT NULL,
ricxvi    INT,
striqoni NVARCHAR(20)
```

```
)
AS
BEGIN
SET @ricxvi = @ricxvi + 10;
INSERT INTO @table1 VALUES(@ricxvi, @striqoni);
SET @ricxvi = @ricxvi + 10;
INSERT INTO @table1 VALUES(@ricxvi, @striqoni);
SETricxvi = @ricxvi + 10;
INSERT INTO @table1 VALUES(@ricxvi, @striqoni);
UPDATE @table1 SET ricxvi = ricxvi + 5 WHERE id = 3;
DELETE FROM @table1 WHERE id = 2;
RETURN
END
```

ფუნქციის გამოძახება:

```
USE Shekveta;
SELECT * FROM Funqcia_2(15, N'საბა');
```

შედეგი:

	id	ricxvi	striqoni
1	1	25	საბა
2	3	50	საბა

ფუნქციის წაშლა

ფუნქციის წასაშლელად გამოიყენება DROP FUNCTION ბრძანება. მისი სინტაქსია:

DROP FUNCTION { [*სქემის_სახელი.*] *ფუნქციის_სახელი* } [,...n]

ბრძანებით შესაძლებელია სამივე ტიპის ფუნქციის წაშლა. თუ ვშლით ფუნქციას, რომელიც სხვა მომხმარებელს ან მონაცემთა ბაზის სხვა მფლობელს ეკუთვნის, მაშინ უნდა მივუთითოთ ფუნქციის მფლობელის სახელიც. ერთი ბრძანებით შეიძლება რამდენიმე ფუნქციის წაშლა.

მაგალითი. წავშალოთ Funqcia_1, Funqcia_2 და Funqcia_3 ფუნქციები:

```
DROP FUNCTION Funqcia_1, Funqcia_2, Funqcia_3;
```

ჩადგმული ფუნქციები

სერვერს დიდი რაოდენობით ჩადგმული (builtin) ფუნქციები აქვს, რომლებიც ახდენენ ხშირად გამოყენებადი ალგორითმების რეალიზებას. ისინი შეგვიძლია გამოვიყენოთ როგორც მოთხოვნებში, ისე გამოსახულებებში. ჩვენ განვიხილავთ ჩადგმული ფუნქციების სამ ჯგუფს:

- მათემატიკის ფუნქციები;
- სტრიქონებთან სამუშაო ფუნქციები;
- თარიღთან და დროსთან სამუშაო ფუნქციები;

მათემატიკის ფუნქციები

მათემატიკის ფუნქციების უმრავლესობა გასცემს იმ ტიპის შედეგს, რა ტიპიც აქვს არგუმენტს. ზოგიერთი მათგანი მოყვანილია ცხრილში 9.1.

ცხრილი 9.1. მათემატიკის ფუნქციები

ფუნქცია	დანიშნულება
ABS(*რიცხვითი_გამოსახულება*)	გასცემს რიცხვის აბსოლუტურ მნიშვნელობას
ISNUMERIC(*გამოსახულება*)	გასცემს 1-ს, თუ არგუმენტს რიცხვითი ტიპი აქვს, საწინააღმდეგო შემთხვევაში - 0-ს.
SIGN(*რიცხვითი_გამოსახულება*)	გასცემს 1-ს, თუ რიცხვი დადებითია, 0-ს თუ რიცხვი 0-ის ტოლია და -1-ს თუ რიცხვი უარყოფითია
RAND([*რიცხვი*])	გასცემს შემთხვევით წილად რიცხვს 0÷1 დიაპაზონში. არგუმენტს უნდა ჰქონდეს TINYINT, INT ან SMALLINT ტიპი. თუ არგუმენტი არ არის მითითებული, მაშინ შემთხვევითი რიცხვის გენერირება მოხდება სისტემური დროის საფუძველზე
FLOOR(*რიცხვითი_გამოსახულება*)	მითითებული არგუმენტისთვის გასცემს უახლოეს მინიმალურ მთელ რიცხვს, ანუ დამრგვალებას ქვევით ასრულებს
CEILING(*რიცხვითი_გამოსახულება*)	გასცემს უახლოეს მთელ რიცხვს, რომელიც არგუმენტზე მეტი ან ტოლია, ანუ დამრგვალებას ზევით ასრულებს
ROUND(*რიცხვითი_გამოსახულება, სიგრძე* [, *ფუნქცია*])	ასრულებს რიცხვითი გამოსახულების დამრგვალებას მითითებული სიზუსტით. დამრგვალება შეიძლება შესრულდეს როგორც ათობითი წერტილის შემდეგ (*სიგრძე* > 0), ისე ათობით წერტილამდე (*სიგრძე* < 0)
POWER(*რიცხვითი_გამოსახულება, ხარისხი*)	გასცემს რიცხვის ხარისხს
SQUARE(*წილადი_გამოსახულება*)	გასცემს არგუმენტის კვადრატს
SQRT(*წილადი_გამოსახულება*)	არგუმენტიდან იღებს კვადრატულ ფესვს
PI()	გასცემს π მნიშვნელობას (3.14)
DEGREES(*რიცხვითი_გამოსახულება*)	ასრულებს კუთხის გარდაქმნას რადიანიდან გრადუსში
RADIANS(*რიცხვითი_გამოსახულება*)	ასრულებს კუთხის გარდაქმნას გრადუსიდან რადიანში
SIN(*წილადი_გამოსახულება*)	რადიანებში გასცემს მითითებული კუთხის სინუსს
COS(*წილადი_გამოსახულება*)	რადიანებში გასცემს მითითებული კუთხის კოსინუსს
TAN(*წილადი_გამოსახულება*)	რადიანებში გასცემს მითითებული კუთხის ტანგენსს
COT(*წილადი_გამოსახულება*)	რადიანებში გასცემს მითითებული კუთხის კოტანგენსს
EXP(*წილადი_გამოსახულება*)	გასცემს არგუმენტის ექსპონენტას
LOG(*წილადი_გამოსახულება*)	გასცემს არგუმენტის ნატურალურ ლოგარითმს
LOG10(*წილადი_გამოსახულება*)	გასცემს არგუმენტის ათობით ლოგარითმს

მაგალითები.

ა. 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];

შედეგი:

	დამრგვალება ქვევით	დამრგვალება ზევით	დამრგვალება წერტილის შემდეგ	დამრგვალება წერტილამდე
1	3	4	123.79000	9900.000

ბ. SELECT POWER(5,2) AS [რიცხვის ხარისხი],
SQRT(25) AS [კვადრატული ფესვი რიცხვიდან];
შედეგი:

	რიცხვის ხარისხი	კვადრატული ფესვი რიცხვიდან
1	25	5

გ. SELECT SIN(0) AS [სინუსი],
COS(0) AS [კოსინუსი],
ABS(-10) AS [აბსოლუტური მნიშვნელობა];
შედეგი:

	სინუსი	კოსინუსი	აბსოლუტური მნიშვნელობა
1	0	1	10

დ. SELECT RAND(3) AS [შემთხვევითი რიცხვი],
RAND(5) * 10 AS [შემთხვევითი რიცხვი],
RAND() AS [შემთხვევითი რიცხვი];
შედეგი:

	შემთხვევითი რიცხვი	შემთხვევითი რიცხვი	შემთხვევითი რიცხვი
1	0,71362925915544	7,13666525097956	0,454560299686459

ამ მოთხოვნის მრავალჯერ შესრულების შედეგად, ყოველთვის მიიღება ერთი და იგივე რიცხვები. თუ გვინდა სხვადასხვა შემთხვევითი რიცხვების გენერირება, მაშინ
SELECT RAND() AS [შემთხვევითი რიცხვი];
მოთხოვნა უნდა შევასრულოთ ცალკე, დამოუკიდებელი მოთხოვნის სახით. ის, აგრეთვე შეგვიძლია ციკლშიც მოვათავსოთ:

```
DECLARE @i INT = 1;
WHILE @i <= 10
BEGIN
PRINT RAND();
SET @i = @i + 1;
END
```

ამ კოდის ყოველი შესრულებისას მიიღება სხვადასხვა შემთხვევითი რიცხვები:

Messages

```
0.252101
0.489124
0.448349
0.753851
0.493519
0.960948
0.849049
0.149037
0.0987421
0.242907
```

სტრიქონებთან სამუშაო ფუნქციები

ზოგიერთი მათგანი მოყვანილია ცხრილში 9.2.

მაგალითები.

ა. SELECT NCHAR(97),
NCHAR(65),
NCHAR(49);

შედეგი:

	(No column name)	(No column name)	(No column name)
1	a	A	1

ბ. SELECT N'საბა' + SPACE(5) + N'სამხარაძე';

შედეგი:

	(No column name)
1	საბა სამხარაძე

გ. SELECT STR(1234.98765, 7, 2);

შედეგი:

	(No column name)
1	1234.99

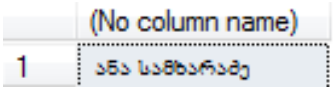
დ. SELECT QUOTENAME('ana'),
QUOTENAME('saba', '"'),
QUOTENAME('saba', '{'),
QUOTENAME('saba', '('),
QUOTENAME('saba', '''');

შედეგი:

	(No column name)	(No column name)	(No column name)	(No column name)	(No column name)
1	[ana]	"saba"	{saba}	(saba)	'saba'

ე. SELECT STUFF(N'საბა სამხარაძე', 1, 4, N'ანა');

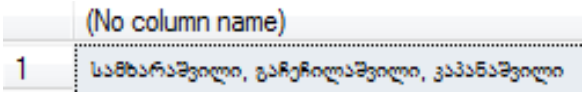
შედეგი:



	(No column name)
1	ანა სამხარაძე

ვ. SELECT REPLACE(N'სამხარაძე, გაჩეჩილაძე, კაპანაძე', N'ძე', N'შვილი');

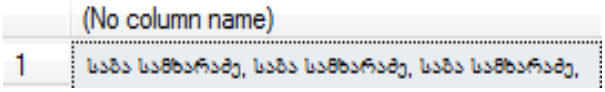
შედეგი:



	(No column name)
1	სამხარაშვილი, გაჩეჩილაშვილი, კაპანაშვილი

ზ. SELECT REPLICATE(N'საბა სამხარაძე, ', 3);

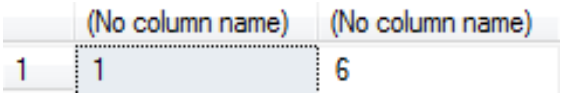
შედეგი:



	(No column name)
1	საბა სამხარაძე, საბა სამხარაძე, საბა სამხარაძე,

თ. SELECT CHARINDEX(N'სა', N'საბა სამხარაძე'),
CHARINDEX(N'სა', N'საბა სამხარაძე', 2);

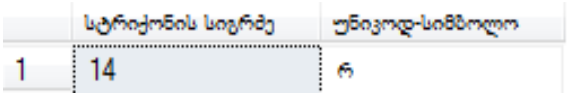
შედეგი:



	(No column name)	(No column name)
1	1	6

ი. SELECT LEN(N'საბა სამხარაძე') AS [სტრიქონის სიგრძე],
NCHAR(4320) AS [უნიკოდ-სიმბოლო];

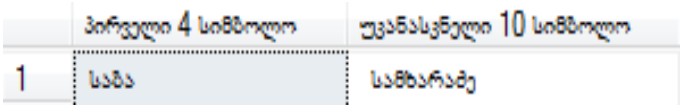
შედეგი:



	სტრიქონის სიგრძე	უნიკოდ-სიმბოლო
1	14	რ

კ. SELECT LEFT(N'საბა სამხარაძე', 4) AS [პირველი 4 სიმბოლო],
RIGHT(N'საბა სამხარაძე', 10) AS [უკანასკნელი 10 სიმბოლო];

შედეგი:



	პირველი 4 სიმბოლო	უკანასკნელი 10 სიმბოლო
1	საბა	სამხარაძე

ცხრილი 9.2. სტრიქონებთან სამუშაო ფუნქციები.

ფუნქცია	დანიშნულება
ASCII(*სტრიქონი*)	გასცემს სტრიქონის მარცხენა სიმბოლოს ASCII კოდს
UNICODE('უნიკოდ_*სტრიქონი*')	გასცემს სტრიქონის მარცხენა სიმბოლოს UNICODE კოდს
CHAR(*მთელრიცხვა_გამოსახულება*)	გასცემს არგუმენტის შესაბამის სიმბოლოს
NCHAR(*მთელრიცხვა_გამოსახულება*)	გასცემს არგუმენტის შესაბამის Unicode სიმბოლოს
LEN(*სტრიქონი*)	გასცემს არგუმენტის ზომას ბაიტებში
LTRIM(*სტრიქონი*)	შლის ინტერვალებს სტრიქონის დასაწყისიდან
RTRIM(*სტრიქონი*)	შლის ინტერვალებს სტრიქონის ბოლოდან
LEFT(*სტრიქონი*, *მთელრიცხვა_გამოსახულება*)	გასცემს მითითებული რაოდენობის სიმბოლოს სტრიქონის დასაწყისიდან
RIGHT(*სტრიქონი*, *მთელრიცხვა_გამოსახულება*)	გასცემს მითითებული რაოდენობის სიმბოლოს სტრიქონის ბოლოდან
LOWER(*სტრიქონი*)	სტრიქონის სიმბოლოები ქვედა რეგისტრში გადაყავს
UPPER(*სტრიქონი*)	სტრიქონის სიმბოლოები ზედა რეგისტრში გადაყავს
STR(*წილადი_გამოსახულება*, */, თანრიგების_საერთო_რაოდენობა /, თანრიგების_რაოდენობა_წერტილის_შემდეგ*]])	რიცხვით მნიშვნელობას სიმბოლურ მნიშვნელობად გარდაქმნის. პირველი არგუმენტია გარდასაქმნელი რიცხვი, მეორეა - თანრიგების საერთო რაოდენობა წერტილის ჩათვლით, მესამე - წერტილის შემდეგ თანრიგების რაოდენობა
SUBSTRING(*სტრიქონი*, *საწყისი_პოზიცია*, *სიგრძე*)	მითითებული სტრიქონისათვის გასცემს მითითებული სიგრძის ქვესტრიქონს დაწყებული მითითებული პოზიციიდან
CHARINDEX(*სტრიქონი1*, *სტრიქონი2* [, *საწყისი_პოზიცია*])	*სტრიქონი2* სტრიქონში ეძებს *სტრიქონი1* ქვესტრიქონს. პოვნის შემთხვევისას გაიცემა იმ პოზიციის ნომერი სადაც პირველად მოიძებნა ქვესტრიქონი. თუ *სტრიქონი1* ქვესტრიქონი რამდენიმეჯერ მეორდება *სტრიქონი2* სტრიქონში, მაშინ რომელიმე მათგანის მოსაძებნად უნდა მივუთითოთ ძებნის დაწყების პოზიცია
PATINDEX("%შაბლონი%", *სტრიქონი*)	*სტრიქონი* არგუმენტში ეძებს შაბლონის საშუალებით მითითებულ ქვესტრიქონს
SPACE(*მთელრიცხვა_გამოსახულება*)	გასცემს მითითებული რაოდენობის ინტერვალს
REPLICATE(*სტრიქონი*, *მთელრიცხვა_გამოსახულება*)	ახდენს სტრიქონის ტირაჟირებას იმდენჯერ, რა რიცხვიც მითითებულია მეორე პარამეტრით
STUFF(*სტრიქონი1*, *საწყისი_პოზიცია*, *რაოდენობა*, *სტრიქონი2*)	*სტრიქონი1* სტრიქონიდან შლის მითითებული რაოდენობის სიმბოლოს დაწყებული მითითებული პოზიციიდან და მათ ნაცვლად *სტრიქონი1* სტრიქონში ათავსებს *სტრიქონი2* სტრიქონს

ცხრილი 9.2. სტრიქონებთან სამუშაო ფუნქციები (გაგრძელება)

REPLACE('*სტრიქონი*1', '*სტრიქონი*2', '*სტრიქონი*3')	პირველი პარამეტრით მითითებულ სტრიქონში მეორე პარამეტრით მითითებულ ქვესტრიქონს შეცვლის მესამე პარამეტრით მითითებული ქვესტრიქონით
REVERSE(*სტრიქონი*)	ფუნქცია ახდენს მითითებული სტრიქონის შებრუნებას
QUOTENAME('*სტრიქონი*' [, '*შემზღუდავი_სიმბოლო*'])	მითითებულ სტრიქონს ათავსებს კვადრატულ ფრჩხილებში, თუ შემზღუდავი სიმბოლო მითითებული არ არის. მეორე პარამეტრი განსაზღვრავს შემზღუდავ სიმბოლოს((,{,",'')

თარიღებთან სამუშაო ფუნქციები

ზოგიერთი მათგანი მოყვანილია ცხრილში 9.3.

ცხრილი 9.3. თარიღებთან სამუშაო ფუნქციები.

ფუნქცია	დანიშნულება
GETDATE()	გასცემს მიმდინარე სისტემურ დროს
ISDATE(*გამოსახულება*)	გასცემს 1-ს, თუ მითითებული გამოსახულება შეიძლება გარდაიქმნას DATETIME ან SMALLDATETIME ტიპად, საწინააღმდეგო შემთხვევაში გაიცემს 0-ს
DAY(*თარიღი*)	გასცემს დღეს მითითებული თარიღიდან
MONTH(*თარიღი*)	გასცემს თვეს მითითებული თარიღიდან
YEAR(*თარიღი*)	გასცემს წელს მითითებული თარიღიდან
DATEADD(*თარიღის_ნაწილი*, *რიცხვი*, *თარიღი*)	მითითებულ თარიღს უმატებს დღეებს, თვეებს, წლებს, საათებს, წუთებს და ა.შ. პირველი პარამეტრი განსაზღვრავს დასამატებელი მნიშვნელობის ტიპს: yy - წელი qq - კვარტალი mm - თვე dy - დღე dd - დღე wk - კვირა hh - საათი mi - წუთი ss - წამი ms - მილიწამი
DATEDIFF(*თარიღის_ნაწილი*, *საწყისი_თარიღი*, *საბოლოო_თარიღი*)	გამოთვლის სხვაობას ორი თარიღის მითითებულ ნაწილებს შორის
DATENAME(*თარიღის_ნაწილი*, *თარიღი*)	თარიღიდან გამოყოფს მითითებულ ნაწილს და გასცემს მას სიმბოლურ ფორმატში
DATEPART(*თარიღის_ნაწილი*, *თარიღი*)	თარიღიდან გამოყოფს მითითებულ ნაწილს და გასცემს მას რიცხვით ფორმატში

მაგალითები.

ა. SET DATEFORMAT DMY;

```
SELECT  DATEADD(dd, 5, '19.03.2005'),
        DATEADD(yy, -3, '05 Oct 2000');
SELECT  DATEADD(hh, 10, '19.03.2005 4:30:45');
```

შედეგი:

	(No column name)	(No column name)
1	2005-03-24 00:00:00.000	1997-10-05 00:00:00.000

	(No column name)
1	2005-03-19 14:30:45.000

ბ. SET DATEFORMAT DMY;
SELECT DATEDIFF(dd, '19.03.2005', '10.03.2005'),
DATEDIFF(yy, '05 Oct 2000', '05 Oct 2005'),
DATEDIFF(hh, '19.03.2005 4:30:45', '19.03.2005 14:40:55');
შედეგი:

	(No column name)	(No column name)	(No column name)
1	-9	5	10

გ. SET DATEFORMAT DMY;
SELECT MONTH('19/03/2005'), MONTH('05 Oct 2000');
შედეგი:

	(No column name)	(No column name)
1	3	10

დ. SET DATEFIRST 5;
SELECT @@DATEFIRST AS 'პირველი დღე', DATEPART(dw, GETDATE()) AS 'დღევანდელი დღე';
შედეგი:

	პირველი დღე	დღევანდელი დღე
1	5	3

თავი 10. შენახული პროცედურები

შესავალი

შენახული პროცედურა (*stored procedure*) არის Transact-SQL-ის ბრძანებების სახელდებული ნაკრები, რომელიც უშუალოდ სერვერზე ინახება და მონაცემთა ბაზის დამოუკიდებელ ობიექტს წარმოადგენს. თუ სერვერზე ხშირად გვიწევს ერთი და იმავე ოპერაციების შესრულება, მაშინ უმჯობესია მათი გაფორმება შენახული პროცედურის სახით.

შენახული პროცედურების გამოძახება შესაძლებელია კლიენტის პროგრამის, სხვა შენახული პროცედურის ან ტრიგერის მიერ. შენახული პროცედურის კოდის შეცვლა შეუძლია მხოლოდ მის მფლობელს ან მონაცემთა ბაზის db_owner ფიქსირებული როლის წევრს. საჭიროებისამებრ შენახული პროცედურის ფლობის უფლება შეგვიძლია ერთი მომხმარებლიდან მეორეს გადავცეთ.

შენახული პროცედურის გამოყენებას შემდეგი უპირატესობები აქვს:

– შენახული პროცედურის შესრულების წინ სერვერი მისთვის ადგენს *შესრულების გეგმას (execution plan)*, ასრულებს მის ოპტიმიზებას და კომპილირებას. ამრიგად, შენახული პროცედურის პირველი გამოძახებისას სერვერი დროს და რესურსებს ხარჯავს არა მარტო შენახული პროცედურის ბრძანებების შესრულებაზე, არამედ შესასრულებლად მათ მომზადებაზე, რაც საკმაო რესურსს მოითხოვს. მწარმოებლურობის ასამაღლებლად სერვერი ასრულებს შენახული პროცედურის შესრულების გეგმისა და კომპილირებული კოდის ქეშირებას. იმავე შენახული პროცედურის განმეორებითი გამოძახებისას სერვერი მაშინვე დაიწყებს ბრძანებების შესრულებას. ამრიგად, შენახული პროცედურები იძლევა ოპერაციების შესრულების სიჩქარის გაზრდის შესაძლებლობას, რადგან მათი განმეორებით გამოყენებისას, ისინი უკვე ჩატვირთულია ოპერატიულ მეხსიერებაში, სადაც მათი მოძებნა გაცილებით სწრაფად ხდება.

– შენახული პროცედურის შესასრულებლად საკმარისია მისი სახელის მითითება. ეს ამცირებს მოთხოვნის ზომას, რომელიც ქსელში იგზავნება კლიენტიდან სერვერისკენ. შედეგად, შენახული პროცედურების გამოყენება ამცირებს ქსელზე დატვირთვებს.

– შენახული პროცედურების გამოყენება აადვილებს მოდულური პროექტირების პრინციპის რეალიზებას, რადგან პროცედურები იძლევა დიდი ამოცანების უფრო პატარა, დამოუკიდებელ და ადვილად სამართავ ამოცანებად დაყოფის საშუალებას.

შენახული პროცედურების ტიპები

შენახული პროცედურის რამდენიმე ტიპი არსებობს:

– *სისტემური შენახული პროცედურები* (*System stored procedures*) შედის სერვერის შემადგენლობაში და განკუთვნილია ისეთი ადმინისტრაციული მოქმედებების შესასრულებლად, როგორიცაა, სააღრიცხვო ჩანაწერების შექმნა, მონაცემთა ბაზების ობიექტების შესახებ ინფორმაციის მიღება, სერვერისა და მონაცემთა ბაზის თვისებების მართვა, ავტომატიზებისა და რეპლიკაციის ქვესისტემის მართვა და ა.შ. პრაქტიკულად, სერვერის ადმინისტრირების ყველა მოქმედება სრულდება სისტემური შენახული პროცედურების საშუალებით. სისტემურ შენახულ პროცედურებს აქვთ sp_ პრეფიქსი (*system procedure*). ისინი ინახება master სისტემურ მონაცემთა ბაზაში და შეგვიძლია ნებისმიერი მონაცემთა ბაზიდან გამოვიძახოთ.

– *მომხმარებლის მიერ შექმნილი შენახული პროცედურები* (*User_defined stored procedures*). ისინი ინახება შესაბამის მონაცემთა ბაზაში და წარმოადგენს ამ მონაცემთა ბაზის

ობიექტს.

– *დროებითი შენახული პროცედურები* (*Temporary stored procedures*). ასეთი შენახული პროცედურები არსებობს მხოლოდ გარკვეული დროის განმავლობაში, რის შემდეგ ავტომატურად იშლება სერვერის მიერ. არსებობს ლოკალური და გლობალური დროებითი პროცედურები:

- *ლოკალური დროებითი შენახული პროცედურები* (*Local temporary stored procedures*) შეიძლება გამოძახებული იყოს მხოლოდ იმ შეერთებიდან, რომლებშიც ისინი შეიქმნა. ასეთი პროცედურების სახელები # სიმბოლოთი უნდა იწყებოდეს. დროებითი შენახული პროცედურები, სხვა დროებითი ობიექტების მსგავსად, ინახება tempdb მონაცემთა ბაზაში და ავტომატურად იშლება მომხმარებლის გამორთვისას, სერვერის გაჩერების ან ხელახალი გაშვების დროს.
- *გლობალური დროებითი შენახული პროცედურები* (*Global temporary stored procedures*) შეიძლება გამოძახებული იყოს ნებისმიერი შეერთებიდან. ასეთი პროცედურის განსაზღვრისას მისი სახელის წინ უნდა მოვათავსოთ ## სიმბოლოები. ეს პროცედურები ინახება tempdb მონაცემთა ბაზაში და იშლება სერვერის გაჩერების ან ხელახალი გაშვების დროს, აგრეთვე, იმ შეერთების დახურვისას, რომლებშიც ისინი შეიქმნა.

შენახული პროცედურის შექმნა

შენახული პროცედურის შექმნამდე უნდა გადავწყვიტოთ შემდეგი საკითხები:

– შესაქმნელი შენახული პროცედურის ტიპის განსაზღვრა. უნდა გადავწყვიტოთ შენახული პროცედურა დროებითი იქნება თუ მომხმარებლის. შეგვიძლია, აგრეთვე, შევქმნათ სისტემური შენახული პროცედურა, მისი სახელისათვის sp_ პრეფიქსის დამატების და მისი master სისტემურ მონაცემთა ბაზაში მოთავსების გზით.

– მიმართვის უფლებების დაგეგმვა. შენახული პროცედურის შექმნისას უნდა გავითვალისწინოთ ის, რომ მას ექნება მონაცემთა ბაზის ობიექტებთან მიმართვის ისეთივე უფლებები, როგორიც აქვს მის შემქმნელ მომხმარებელს. უნდა გვახსოვდეს, რომ შენახულ პროცედურას ექნება არა მისი შემსრულებელი მომხმარებლის, არამედ მისი შემქმნელი მომხმარებლის უფლებები. გარდა ამისა, შენახული პროცედურა მემკვიდრეობით იღებს ზოგიერთ პარამეტრს, რომელიც განსაზღვრული იყო მისი შექმნის დროს. კერძოდ, თუ პროცედურის შექმნისას ნებადართული იყო სისტემურ ცხრილებთან მიმართვა, მაშინ პროცედურა ყოველთვის შეძლებს სისტემურ ცხრილებთან მიმართვას.

– შენახული პროცედურის პარამეტრების განსაზღვრა. შენახულ პროცედურას შეიძლება ჰქონდეს შესასვლელი და გამოსასვლელი პარამეტრები. პროცედურის პარამეტრები შეგვიძლია გამოვიყენოთ როგორც ჩვეულებრივი ცვლადები.

– შენახული პროცედურის კოდის შემუშავება. პროცედურის კოდი შეიძლება შეიცავდეს ნებისმიერი ბრძანებების მიმდევრობას, სხვა შენახული პროცედურების გამოძახებების ჩათვლით.

ამ საკითხების გადაწყვეტის შემდეგ შეგვიძლია შენახული პროცედურის შექმნა.

შენახული პროცედურის შესაქმნელად გამოიყენება CREATE PROCEDURE ბრძანება. მისი სინტაქსია:

CREATE PROC [EDURE] [*სქემის_სახელი.*] *პროცედურის_სახელი*
[{ *@პარამეტრის_სახელი პარამეტრის_ტიპი* } [VARYING] [= DEFAULT] [OUTPUT]] [,...n]
[WITH { RECOMPILE | ENCRYPTION | RECOMPILE, ENCRYPTION }]
AS sql_*ბრძანება* [,...n]

განვიხილოთ არგუმენტების დანიშნულება.

– *პროცედურის_სახელი* არის შესაქმნელი პროცედურის სახელი. თუ გამოვიყენებთ sp_, # და ## პრეფიქსებს, მაშინ შესაბამისად შეგვიძლია შევქმნათ სისტემური, ლოკალური და გლობალური დროებითი პროცედურები. სინტაქსიდან გამომდინარე დაუშვებელია მფლობელის სახელისა და მონაცემთა ბაზის სახელის მითითება. შენახული პროცედურა ყოველთვის იქმნება მიმდინარე მონაცემთა ბაზაში.

– *@პარამეტრის_სახელი* ცვლადი შეიცავს იმ პარამეტრის სახელს, რომელსაც შენახული პროცედურა გამოიყენებს შესასვლელი და გამოსასვლელი მონაცემების გადაცემის მიზნით. შენახული პროცედურის პარამეტრების სახელები @ სიმბოლოთი უნდა იწყებოდეს. პარამეტრები ერთმანეთისაგან მძიმეებით გამოიყოფა. შენახული პროცედურის პარამეტრები წარმოადგენს ლოკალურ პარამეტრებს, ამიტომ სხვადასხვა შენახულ პროცედურას შეიძლება ერთნაირი სახელის პარამეტრები ჰქონდეს. შენახული პროცედურის კოდში პარამეტრის სახელი ცვლადის სახელს არ უნდა ემთხვეოდეს.

– *პარამეტრის_ტიპი* განსაზღვრავს პარამეტრის ტიპს. პარამეტრს შეიძლება ნებისმიერი ტიპი ჰქონდეს, მათ შორის მომხმარებლის მიერ განსაზღვრული ტიპები. რაც შეეხება cursor ტიპს, ის შეგვიძლია გამოვიყენოთ მხოლოდ გამოსასვლელი პარამეტრებისათვის.

– OUTPUT არგუმენტი მიუთითებს, რომ პარამეტრი არის გამოსასვლელი ანუ შეგვიძლია გამოვიყენოთ მონაცემების დასაბრუნებლად შენახული პროცედურიდან. ასეთი პარამეტრი შეგვიძლია გამოვიყენოთ როგორც შესასვლელიც. შენახული პროცედურიდან გამოსვლისას გამოსასვლელი პარამეტრის მიმდინარე მნიშვნელობა ენიჭება იმ ლოკალურ ცვლადს, რომელიც მითითებული იყო შენახული პროცედურის გამოძახების დროს.

– VARYING არგუმენტს აქვს cursor ტიპი და გამოიყენება OUTPUT არგუმენტთან ერთად. ის მიუთითებს, რომ გამოსასვლელი პარამეტრი იქნება სიმრავლე.

– DEFAULT არგუმენტი მიუთითებს ნაგულისხმევ მნიშვნელობას, რომელიც პარამეტრს მიენიჭება, თუ მისი მნიშვნელობა მითითებული არ იქნება.

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

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

– 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.

ერთი შენახული პროცედურის ტანიდან სხვა შენახული პროცედურების გამოძახებისას იქმნება *ჩადგმული პროცედურები* (*nested procedure*). ასეთ პროცედურებთან მუშაობისას უნდა გავითვალისწინოთ შეზღუდვა ჩადგმულობის რაოდენობაზე. ჩადგმულობის დონე არ უნდა აღემატებოდეს 32-ს. ჩადგმულობის მიმდინარე დონის მისაღებად შეგვიძლია @@NESTLEVEL ფუნქციის გამოძახება (SELECT @@NESTLEVEL AS [ჩადგმულობის დონე]).

შექმნილი შენახული პროცედურის სახელი შეიტანება ამავე მონაცემთა ბაზის sys.objects სისტემურ წარმოდგენაში. მის სანახავად შეგვიძლია შევასრულოთ მოთხოვნა:

```
SELECT name FROM sys.objects ORDER BY name
```

მაგალითი. შევქმნათ შენახული პროცედურა, რომელსაც გადავცემთ განყოფილების სახელს და მივიღებთ ინფორმაციას ამ განყოფილებაში მომუშავე თანამშრომლების შესახებ. პროგრამის კოდს აქვს სახე:

```
USE Shekveta;
```

```
--      თუ შენახული პროცედურა არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID ('MyProc', 'P' ) IS NOT NULL
   DROP PROCEDURE [dbo].[MyProc];
GO
--      შენახული პროცედურის შექმნა
CREATE PROCEDURE MyProc @ganyofileba NVARCHAR(30)
AS
SELECT  *
FROM    Personali
WHERE ganyofileba = @ganyofileba
```

გამოვიძახოთ ეს პროცედურა:

```
EXEC MyProc N'სავაჭრო';
```

შედეგი:

	personaliID	gvari	sakheli	ganyofileba	qalaqi	raioni	xelfasi
1	3	გაჩეჩილაძე	ლია	სავაჭრო	თბილისი	საბურთალო	2632,97808284359
2	5	შენგელია	ნატა	სავაჭრო	ზუგდიდი	NULL	2891,06617556759
3	11	კოკნაძე	მზია	სავაჭრო	ბათუმი	NULL	1078,9636
4	24	სამხარაძე	გიორგი	სავაჭრო	თბილისი	ჩუღურეთი	NULL

შენახული პროცედურის შექმნის მომენტში სრულდება ბრძანებების მხოლოდ სინტაქსური შემოწმება და არ მოწმდება მასში მითითებული ობიექტების არსებობა. მიმართვების სისწორის შემოწმება ხდება შენახული პროცედურის კომპილირების დროს, რაც უშუალოდ შენახული პროცედურის შესრულების წინ ხდება.

არსებობს შენახული პროცედურის შესრულების ორი გზა:

– მხოლოდ მისი სახელის მითითება;

– EXECUTE ბრძანების გამოყენება.

პირველ შემთხვევაში შენახული პროცედურის გამოძახება უნდა იყოს ერთადერთი ბრძანება შესასრულებლად გადასაცემ პაკეტში. როცა შენახული პროცედურის გამოძახება არ არის ერთადერთი ბრძანება პაკეტში, მაშინ უნდა გამოვიყენოთ EXECUTE ბრძანება. ეს ბრძანება უნდა გამოვიყენოთ აგრეთვე, შენახული პროცედურის გამოძახებისას სხვა შენახული პროცედურიდან ან ტრიგერიდან. EXECUTE ბრძანების სინტაქსია:

[EXEC [UTE]] *პროცედურის_სახელი*

[[*@პარამეტრის_სახელი* =] { *სიდიდე* | *@ცვლადის_სახელი* } [OUTPUT] | [DEFAULT]] [,...n]

OUTPUT არგუმენტის გამოყენება ნებადართულია იმ შემთხვევაში, როცა შესაბამისი პარამეტრი გამოცხადებული იყო OUTPUT სიტყვის გამოყენებით შენახული პროცედურის შექმნის დროს. ანალოგიურად, DEFAULT არგუმენის გამოყენება ნებადართულია იმ შემთხვევაში, როცა შესაბამისი პარამეტრისათვის განსაზღვრული იყო ნაგულისხმევი მნიშვნელობა შენახული პროცედურის შექმნის დროს. ამ შემთხვევაში, პარამეტრს ნაგულისხმევი მნიშვნელობა მიენიჭება.

შენახული პროცედურის გამოძახებისას პარამეტრების სახელები შეგვიძლია არ მივუთითოთ. ასეთ შემთხვევაში, პარამეტრების მნიშვნელობები ისეთივე მიმდევრობით უნდა მივუთითოთ, როგორც იყო ისინი ჩამოთვლილი შენახული პროცედურის შექმნის დროს. პარამეტრისათვის ნაგულისხმევი მნიშვნელობის მინიჭება, ჩამონათვალში მისი სახელის გამოტოვებით, არ შეიძლება. მიუხედავად ამისა, ბოლო პარამეტრები შეგვიძლია გამოვტოვოთ,

თუ მათთვის განსაზღვრულია ნაგულისხმევი მნიშვნელობა. თუ გვინდა გამოვტოვოთ ის პარამეტრები, რომლებისთვისაც განსაზღვრულია ნაგულისხმევი მნიშვნელობა, მაშინ მათი სახელები აშკარად უნდა მივუთითოთ შენახული პროცედურის გამოძახების დროს. უფრო მეტიც, ასეთი გზით შეგვიძლია ჩამოვთვალოთ პარამეტრები და მათი მნიშვნელობები ნებისმიერი მიმდევრობით. შენახული პროცედურის გამოძახებისას შეგვიძლია მივუთითოთ ან პარამეტრის სახელი მნიშვნელობასთან ერთად, ან მხოლოდ მნიშვნელობა სახელის გარეშე. მათი კომბინაცია დაუშვებელია.

მაგალითები.

ა. შევადგინოთ შენახული პროცედურა, რომელიც გასცემს თანამშრომლების სიას. პროცედურას პარამეტრები არ აქვს. პროგრამის კოდს აქვს სახე:

```
USE Shekveta;
--      თუ შენახული პროცედურა არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID ( 'Chemi_Procedura', 'P' ) IS NOT NULL
   DROP PROCEDURE Chemi_Procedura;
GO
--      შენახული პროცედურის შექმნა
CREATE PROCEDURE Chemi_Procedura
AS
SELECT * FROM Personali;
```

ეს პროცედურა შეგვიძლია გამოვიძახოთ ორი გზით:

1. თუ პროცედურის გამოძახება არ არის პირველი ოპერატორი პაკეტის შიგნით, მაშინ გამოძახებას ექნება სახე:

```
EXEC Chemi_Procedura;
```

2. თუ პროცედურის გამოძახება არის პირველი ოპერატორი პაკეტის შიგნით, მაშინ გამოძახებას ექნება სახე:

```
Chemi_Procedura;
```

შედეგი:

	personaliID	gvari	sakheli	ganyofileba	qalaqi	raioni	xelfasi
1	1	სამხარაძე	საბა	სამედიცინო	თბილისი	დიდუბე	979,894867790303
2	2	სამხარაძე	ანა	სამედიცინო	ბათუმი	NULL	7110,63740485804
3	3	გაჩეჩილაძე	ლია	სავაჭრო	თბილისი	საბურთალო	2632,97808284359
4	4	გაჩეჩილაძე	ხათუნა	სასოფლო	თბილისი	ვერა	3310,63536603096
5	5	შენგელია	ნატა	სავაჭრო	ზუგდიდი	NULL	2891,06617556759
6	6	კაპანაძე	ეთერი	სასპორტო	რუსთავი	NULL	2354,52589642459
7	7	ქუხიშვილი	ალექსანდრე	სამედიცინო	თბილისი	ვერა	7213,74859867189
8	8	ხოშტარია	ცისანა	სასპორტო	ზუგდიდი	NULL	1882,61481990309
9	9	ყალაბეგიშვილი	გია	სასოფლო	ქუთაისი	NULL	2370,83298862185
10	10	გიორგაძე	ზვიადი	სამედიცინო	ქობულეთი	NULL	699,740399336
11	11	კიკნაძე	მზია	სავაჭრო	ბათუმი	NULL	1078,9636
12	23	სამხარაძე	ბექა	სასოფლო	თბილისი	ვაკე	300,55

ბ. შევადგინოთ შენახული პროცედურა, რომელიც გასცემს მითითებული განყოფილების თანამშრომლების სიას. ნაგულისხმევ მნიშვნელობად მითითებულია 'სავაჭრო'. პროგრამის კოდს აქვს სახე:

```
USE Shekveta;
--      თუ შენახული პროცედურა არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID ( 'MYPROC_PAR2', 'P' ) IS NOT NULL
   DROP PROCEDURE MYPROC_PAR2;
GO
--      შენახული პროცედურის შექმნა
CREATE PROCEDURE MYPROC_PAR2 @ganyofileba NVARCHAR(30) = N'სავაჭრო'
AS
SELECT  *
FROM    Personali
WHERE ganyofileba = @ganyofileba
```

გამოვიძახოთ ეს პროცედურა:

```
EXEC MYPROC_PAR2 DEFAULT;
```

შედეგი:

	personaliID	gvari	sakheli	ganyofileba	qalaqi	raioni	xelfasi	asaki
1	3	გაჩეჩილაძე	ლია	სავაჭრო	თბილისი	საბურთალო	2632,97808284359	46
2	5	შენგელია	ნატა	სავაჭრო	ზუგდიდი	NULL	2891,06617556759	28
3	11	კიკნაძე	მზია	სავაჭრო	ბათუმი	NULL	1078,9636	35
4	24	სამხარაძე	გიორგი	სავაჭრო	თბილისი	ჩუღურეთი	NULL	25

გ. შევადგინოთ შენახული პროცედურა, რომელიც გამოთვლის და დაგვიბრუნებს მითითებული განყოფილების საშუალო ხელფასს. პროგრამის კოდს აქვს სახე:

```
USE Shekveta;
--      თუ შენახული პროცედურა არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID ( 'Sashualo', 'P' ) IS NOT NULL
   DROP PROCEDURE Sashualo;
GO
--      შენახული პროცედურის შექმნა
CREATE PROC Sashualo @ganyofileba NVARCHAR(30), @sashualo_xelfasi FLOAT OUTPUT
AS
SET @sashualo_xelfasi =
(
SELECT  AVG(xelfasi)
FROM    Personali
WHERE @ganyofileba = ganyofileba
)
```

გამოვიძახოთ ეს პროცედურა:

```
DECLARE @ganyofileba NVARCHAR(30), @sashualo_xelfasi FLOAT;
SET       @ganyofileba = N'სავაჭრო';
EXEC     Sashualo @ganyofileba, @sashualo_xelfasi OUTPUT;
SELECT @ganyofileba AS [განყოფილება], ROUND(@sashualo_xelfasi,2) AS [საშუალო ხელფასი];
```

შედეგი:

	განყოფილება	საშუალო ხელფასი
1	სავაჭრო	2201

დ. შევადგინოთ შენახული პროცედურა, რომელიც გასცემს ინფორმაციას მითითებულ ქალაქში და მითითებულ განყოფილებაში მომუშავე თანამშრომლების შესახებ. პროგრამის კოდს აქვს სახე:

```
USE Shekveta;
--      თუ შენახული პროცედურა არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID ( 'qalaqi_ganyofileba', 'P' ) IS NOT NULL
   DROP PROCEDURE qalaqi_ganyofileba;
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';
```

შედეგი:

	personaliID	gvari	sakheli	ganyofileba	qalaqi	raioni	xelfasi
1	3	გაჩეჩილაძე	ლია	სავაჭრო	თბილისი	საბურთალო	2632,97808284359
2	24	სამხარაძე	გიორგი	სავაჭრო	თბილისი	ჩუღურეთი	NULL

იმავე პროცედურის გამოძახების დროს შეგვიძლია არ მივუთითოთ DEFAULT პარამეტრი. ასეთ შემთხვევაში, მისი მნიშვნელობა იქნება პროცედურის შექმნის დროს განსაზღვრული მნიშვნელობა:

```
EXEC qalaqi_ganyofileba N'თბილისი';
```

შედეგი:

	personaliID	gvari	sakheli	ganyofileba	qalaqi	raioni	xelfasi
1	1	სამხარაძე	საბა	სამედიცინო	თბილისი	დიდუბე	979,894867790303
2	7	ქევხიშვილი	ალექსანდრე	სამედიცინო	თბილისი	ვერა	7213,74859867189

იმავე შედეგს მივიღებთ, თუ შევასრულებთ შემდეგ ბრძანებას:

```
EXEC qalaqi_ganyofileba N'თბილისი', DEFAULT;
```

ე. მოყვანილ მაგალითში, პროცედურის გამოძახების დროს, პარამეტრები მითითებულია ჩვენთვის საჭირო მიმდევრობით:

```
USE Shekveta;
--      თუ შენახული პროცედურა არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID ( '[dbo].[qalaqi_asaki]', 'P' ) IS NOT NULL
   DROP PROCEDURE qalaqi_asaki;
GO
```

```
--      შენახული პროცედურის შექმნა
CREATE PROC qalaqi_asaki @qalaqi NVARCHAR(30), @ganyofileba NVARCHAR(30), @asaki INT
AS
SELECT  *
FROM    Personali
WHERE  @qalaqi = qalaqi AND @ganyofileba = ganyofileba AND @asaki < asaki;
```

გამოვიძახოთ ეს პროცედურა:

```
EXEC qalaqi_asaki @asaki = 30, @qalaqi = N'თბილისი', @ganyofileba = N'სამედიცინო';
```

შედეგი:

	personaliID	gvari	sakheli	ganyofileba	qalaqi	raioni	xelfasi
1	7	ქუცნიშვილი	ალექსანდრე	სამედიცინო	თბილისი	ვერა	7213,74859867189

3. შენახული პროცედურის პარამეტრი შეიცავს შაბლონს, რომლის მიხედვით მოხდება gvari სვეტში მნიშვნელობების ძებნა:

```
USE Shekveta;
--      თუ შენახული პროცედურა არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID ( '[dbo].[Chemi_Procedura1]', 'P' ) IS NOT NULL
   DROP PROCEDURE Chemi_Procedura1;
GO
--      შენახული პროცედურის შექმნა
CREATE PROCEDURE Chemi_Procedura1 @gvari NVARCHAR(30) = N'გ%'
AS
SELECT  *
FROM    Shemkveti
WHERE gvari LIKE @gvari;
```

ეს პროცედურა რამდენიმე გზით შეგვიძლია გამოვიძახოთ:

1. EXEC Chemi_Procedura1;

შედეგი:

	shemkvetiID	personaliID	iuridiuli_fizikuri	gvari	sakheli	qalaqi	raioni	misamarti
1	8	6	იურიდიული	გიორგაძე	გივი	ზუგდიდი	NULL	კოსტავას 55

ამ პროცედურას შეგვიძლია გადავცეთ პარამეტრიც:

2. EXEC Chemi_Procedura1 N'ს%';

შედეგი:

	shemkvetiID	personaliID	iuridiuli_fizikuri	gvari	sakheli	qalaqi	raioni	misamarti
1	2	7	იურიდიული	სეხნიაშვილი	გია	თელავი	NULL	აღმაშენებლის 34
2	6	3	ფიზიკური	სამხარაძე	არჩილი	ბათუმი	NULL	ერას ქ. 1

3. EXEC Chemi_Procedura1 N'ნონიაშვილი';

შედეგი:

	shemkvetiID	personaliID	iuridiuli_fizikuri	gvari	sakheli	qalaqi	raioni	misamarti
1	5	2	იურიდიული	ნონიაშვილი	ზურა	ბათუმი	NULL	რუსთაველის 40

4. EXEC Chemi_Procedura1 N'[ნლ]%';

შედეგი:

	shemkvetiID	personaliID	iuridiuli_fizikuri	gvari	sakheli	qalaqi	raioni
1	1	3	ფიზიკური	ნემსაძე	დოდო	თბილისი	ავლაბარი
2	5	2	იურიდიული	ნონიაშვილი	ზურა	ბათუმი	NULL
3	10	1	იურიდიული	ლომიძე	რუსუდანი	თბილისი	საბურთალო
4	12	8	ფიზიკური	ლასურაშვილი	ია	თბილისი	დიდომი

ზ. მაგალითში ხდება RECOMPILE არგუმენტის გამოყენების დემონსტრირება. პროგრამის კოდს აქვს სახე:

```
USE Shekveta;
--      თუ შენახული პროცედურა არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID ( 'Chemi_Procedura', 'P' ) IS NOT NULL
    DROP PROCEDURE Chemi_Procedura;
GO
--      შენახული პროცედურის შექმნა
CREATE PROCEDURE Chemi_Procedura
WITH RECOMPILE
AS
SELECT *
FROM   Personali;
```

გამოვიძახოთ ეს პროცედურა:

```
EXEC Chemi_Procedura;
```

შედეგი:

	personaliID	gvari	sakheli	ganyofileba	qalaqi	raioni	xelfasi
1	1	სამხარაძე	საბა	სამედიცინო	თბილისი	დიდუბე	979,894867790303
2	2	სამხარაძე	ანა	სამედიცინო	ბათუმი	NULL	7110,63740485804
3	3	გაჩეჩილაძე	ლია	სავაჭრო	თბილისი	საბურთალო	2632,97808284359
4	4	გაჩეჩილაძე	ხათუნა	სასოფლო	თბილისი	ვერა	3310,63536603096
5	5	შენგელია	ნატა	სავაჭრო	ზუგდიდი	NULL	2891,06617556759
6	6	კაპანაძე	ეთერი	სასპორტო	რუსთავი	NULL	2354,52589642459
7	7	ქუხიშვილი	ალექსანდრე	სამედიცინო	თბილისი	ვერა	7213,74859867189
8	8	ხოშტარია	ცისანა	სასპორტო	ზუგდიდი	NULL	1882,61481990309
9	9	ყალაბეგიშვილი	გია	სასოფლო	ქუთაისი	NULL	2370,83298862185
10	10	გიორგაძე	ზვიადი	სამედიცინო	ქობულეთი	NULL	699,740399336

თ. მაგალითში ხდება ENCRYPTION არგუმენტის გამოყენების დემონსტრირება. პროგრამის კოდს აქვს სახე:

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

```
IF OBJECT_ID ( 'Chemi_Procedura', 'P' ) IS NOT NULL
    DROP PROCEDURE Chemi_Procedura;
GO
--        შენახული პროცედურის შექმნა
CREATE PROCEDURE Chemi_Procedura
WITH ENCRYPTION
AS
SELECT *
FROM   Personali ;
```

გამოვიძახოთ ეს პროცედურა:

```
EXEC Chemi_Procedura;
```

მივიღებთ იმავე შედეგს, რაც წინა მაგალითში გვქონდა.

ამ პროცედურის კოდის ნახვა შეუძლებელი იქნება. თუ შევასრულებთ

```
EXEC sp_helptext 'Chemi_Procedura';
```

სისტემურ პროცედურას, მაშინ გაიცემა შეტყობინება იმის შესახებ, რომ ამ პროცედურის კოდი დაშიფრულია:

```
The text for object 'Chemi_Procedura' is encrypted.
```

თუ შევეცდებით პირდაპირ მივმართოთ ამ პროცედურის კოდს

```
SELECT definition FROM sys.sql_modules
WHERE object_id = OBJECT_ID('Chemi_Procedura');
```

მოთხოვნის საშუალებით, მაშინ გაიცემა NULL.

შენახული პროცედურების მართვა

შენახული პროცედურის შესახებ ინფორმაციის მიღება

ნებისმიერი ობიექტის შესახებ, მათ შორის შენახული პროცედურის შესახებ ინფორმაციის მისაღებად, უნდა გამოვიყენოთ sp_help პროცედურა. მისი სინტაქსია:

```
sp_help 'პროცედურის_სახელი'
```

მაგალითი. მივიღოთ ინფორმაცია MYPROC_PAR პროცედურის შესახებ:

```
EXEC sp_help MyProc;
```

შედეგი:

	Name	Owner	Type	Created_datetime
1	MYPROC_PAR	dbo	stored procedure	2007-07-02 12:25:06.967

	Parameter_name	Type	Leng...	Prec	Scale	Param_order	Collation
1	@ganyofileba	nvarchar	60	30	NULL	1	Latin1_General_CI_AS

შენახული პროცედურის სახელის შეცვლა

შენახული პროცედურის სახელის შესაცვლელად გამოიყენება sp_rename ბრძანება. შენახული პროცედურისათვის სახელის შეცვლის დროს, იცვლება მხოლოდ სახელი, რომელიც sysobjects სისტემურ წარმოდგენაში ინახება. ამ პროცედურაზე მიმართვები უცვლელი რჩება.

ამიტომ ჩვენ თვითონ მოგვიწევს ამ მიმართვების შეცვლა.

მაგალითი. 'Chemi_Procedura3' შენახულ პროცედურას 'Chemi_Procedura' ახალი სახელი დავარქვათ:

```
EXEC sp_rename 'Chemi_Procedura3', 'Chemi_Procedura';
SELECT *
FROM   sysobjects
ORDER BY name;
```

შედეგი:

	name	id	xtype	uid	info	status
1	Chemi_Procedura	1595152728	P	1	0	0
2	Chemi_Procedura1	1563152614	P	1	0	0
3	Chemi_Procedura2	1579152671	P	1	0	0
4	cursor_proc	1659152956	P	1	0	0

შენახული პროცედურის წაშლა

შენახული პროცედურის წასაშლელად გამოიყენება DROP PROCEDURE ბრძანება. მისი სინტაქსია:

DROP PROC [EDURE] [*სქემის_სახელი.*] *პროცედურის_სახელი* [,...n]

შენახული პროცედურის წაშლის შედეგად მისი სახელი წაიშლება ამავე მონაცემთა ბაზის sysobjects სისტემური წარმოდგენიდან.

იმისათვის, რომ წავშალოთ და ისევ შევქმნათ ერთი და იმავე სახელის მქონე შენახული პროცედურა ერთსა და იმავე პაკეტში, ბრძანებებს შორის უნდა მივუთითოთ GO ბრძანება.

მაგალითი.

```
DROP PROC A1;
GO
CREATE PROC A1
AS ...
```

შენახული პროცედურის ავტომატურად შესრულების მართვა

შენახული პროცედურის ავტომატურად შესრულებისთვის საჭიროა მისი კონფიგურირება. პროცედურის კონფიგურირება სრულდება მისი თვისებების ცვლილების გზით. ამისათვის გამოიყენება sp_procoption პროცედურა. მისი სინტაქსია:

```
sp_procoption  [ @ProcName = ] 'პროცედურის_სახელი',
               [ @OptionName = ] 'რეჟიმი',
               [ @OptionValue = ] 'მნიშვნელობა'
```

განვიხილოთ არგუმენტების დანიშნულება.

– *რეჟიმი* არის შენახული პროცედურის იმ თვისების სახელი, რომელსაც ვცვლით. ამ შემთხვევაში, ეს არის startup თვისება, რომელიც განსაზღვრავს შენახული პროცედურის ავტომატურად გაშვების შესაძლებლობას.

– *მნიშვნელობა* არგუმენტი იღებს true (ON) ან false (OFF) მნიშვნელობას.

sp_procoption სისტემური შენახული პროცედურის შესრულების უფლება აქვს მხოლოდ სერვერის ფიქსირებული როლის წევრებს.

უნდა გავითვალისწინოთ რამდენიმე შეზღუდვა. ნებადართულია მხოლოდ master მონაცემთა ბაზის შენახული პროცედურების ავტომატურად გაშვება. ავტომატურად გასაშვები შენახული პროცედურა უნდა ეკუთვნოდეს მონაცემთა ბაზის მფლობელს - dbo მომხმარებელს. ამ შეზღუდვის შედეგად, ავტომატურად გაშვებულ შენახულ პროცედურას master მონაცემთა ბაზის ფარგლებში ექნება სრული უფლებები.

შენახული პროცედურის გამოყენების მაგალითები

შენახული პროცედურა შეგვიძლია გამოვიყენოთ მონაცემთა ბაზისა და ცხრილის შესაქმნელად.

მაგალითები.

ა. შევადგინოთ შენახული პროცედურა, რომელიც შემნის Basa_1 მონაცემთა ბაზას. პროგრამის კოდს აქვს სახე:

```
USE Baza;
--      თუ შენახული პროცედურა არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID ( 'Basis_Sheqmna', 'P' ) IS NOT NULL
    DROP PROCEDURE Basis_Sheqmna;
GO
--      შენახული პროცედურის შექმნა
CREATE PROCEDURE Basis_Sheqmna
AS
CREATE DATABASE Basa_1;
```

პროცედურის გამოძახება:

```
EXEC Basis_Sheqmna;
```

ბ. შევადგინოთ შენახული პროცედურა, რომელიც შემნის Table_1 ცხრილს. პროგრამის კოდს აქვს სახე:

```
USE Baza_1;
--      თუ შენახული პროცედურა არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID ( 'Cxrilis_Sheqmna', 'P' ) IS NOT NULL
   DROP PROCEDURE Cxrilis_Sheqmna;
GO
--      შენახული პროცედურის შექმნა
CREATE PROCEDURE Cxrilis_Sheqmna
AS
CREATE TABLE Table_1
(
table_1ID INT PRIMARY KEY IDENTITY(1,1),
sveti_1    INT,
sveti_2    FLOAT,
sveti_3    NVARCHAR(10),
sveti_4    DATETIME
);
```

პროცედურის გამოძახება:

```
EXEC Cxrilis_Sheqmna;
```

თავი 11. ინდექსები

შესავალი

არაინდექსირებულ ცხრილში, რომელშიც ათასობით სტრიქონია, მონაცემების ძებნა დიდ დროს იკავებს. ძებნის დაჩქარების მიზნით ინდექსები გამოიყენება. მონაცემების ძებნის დაჩქარება მათი ფიზიკური ან ლოგიკური მოწესრიგების ხარჯზე მიიღწევა. ინდექსი წარმოადგენს მიმართვების ნაკრებს, რომლებიც მოწესრიგებულია გარკვეული სვეტის მნიშვნელობების მიხედვით. ასეთ სვეტს *ინდექსირებული სვეტი* ეწოდება. მაგალითად, თუ Personali ცხრილში მოვახდენთ gvari სვეტის ინდექსირებას, მაშინ ინდექსში თანამშრომლების გვარები ანბანის მიხედვით დალაგდება.

ინდექსი მონაცემთა ბაზის დამოუკიდებელ ობიექტს წარმოადგენს. ის შეიძლება შედგებოდეს ცხრილის ერთ ან მეტ სვეტისაგან.

ფიზიკურად ინდექსი წარმოადგენს ინდექსირებული სვეტის მონაცემების მოწესრიგებულ ნაკრებს ცხრილის სტრიქონების ფიზიკური განლაგების ადგილმდებარეობაზე მიმთითებლებით. როცა სრულდება მოთხოვნა, რომელიც ინდექსირებულ სვეტს მიმართავს, სერვერი ინდექსს საჭირო მნიშვნელობის მოძებნის მიზნით ავტომატურად აანალიზებს.

ამჟამად, შემუშავებულია ეფექტური მათემატიკური ალგორითმები მოწესრიგებულ მიმდევრობაში მონაცემების მოსაძებნად. ერთ-ერთი მათგანი, რომელიც წარმატებით გამოიყენება, არის „შუაზე გაყოფის" მეთოდი.

ინდექსების გამოყენების დაგეგმვა

თუ ცხრილიდან მონაცემების ამორჩევა დიდ დროს იკავებს, ეს იმას ნიშნავს, რომ საჭიროა ინდექსის შექმნა. ბუნებრივია, ვიფიქროთ, რომ კარგი იქნება ინდექსის შექმნა თითოეული სვეტისთვის, მაგრამ, საქმე ის არის რომ, როცა სრულდება სტრიქონების ცვლილება, მაშინ საკუთრივ მონაცემების გაახლების გარდა, ხდება ყველა ინდექსის გაახლება. ეს კი, თავის მხრივ, გარკვეულ დროს იკავებს. ინდექსების გამოყენების მთავარი უპირატესობაა მონაცემების ამორჩევის მნიშვნელოვანი დაჩქარება, ხოლო მთავარი ნაკლია - მონაცემების გაახლების (დამატების, შეცვლის და წაშლის) პროცესის შენელება.

ინდექსისთვის სვეტის ამორჩევისას უნდა გავაანალიზოთ, თუ რა ტიპის მოთხოვნები შესრულდება ყველაზე ხშირად და რომელი სვეტები არის საკვანძო. საკვანძოა ის სვეტები, რომლებიც განსაზღვრავენ მონაცემების ამორჩევის კრიტერიუმებს.

არ უნდა მოვახდინოთ იმ სვეტების ინდექსირება, რომლებიც არ თამაშობს არანაირ როლს მოთხოვნის შესრულების დროს. არ უნდა მოვახდინოთ ძალიან გრძელი სვეტების ინდექსირება, რომელთა სიგრძეა რამდენიმე ათეული სიმბოლო. საქმე ის არის, რომ ინდექსში შედის არა მარტო სტრიქონზე მიმართვა, არამედ თვით სვეტის შემცველობაც. უკიდურეს შემთხვევაში შეგვიძლია შევქმნათ გრძელი სვეტის შემოკლებული ვარიანტი, რომელშიც მოვათავსებთ პირველ ათ სიმბოლოს და შევასრულებთ მის ინდექსირებას. არ შეიძლება ntext, text, varchar(max), nvarchar(max), varbinary(max), xml, ან image ტიპის სვეტების ინდექსირება.

არსებობს ინდექსების სამი ტიპი: კლასტერული, არაკლასტერული და უნიკალური.

არაკლასტერული ინდექსი

არაკლასტერული ინდექსი (*nonclustered index)* შეიცავს ერთი ან მეტი სვეტის მნიშვნელობებს დალაგებულს ზრდადობით ან კლებადობით, აგრეთვე სტრიქონებზე მიმართვებს. მნიშვნელობების ძებნა სრულდება ზრდადობით ან კლებადობით დალაგებულ უბანში. ნაპოვნი მნიშვნელობის გასწვრივ მოთავსებულია შესაბამის სტრიქონზე მიმართვა, რომელიც შეიცავს სტრიქონის რეალურ მისამართს ცხრილში. არაკლასტერული ინდექსის გამოყენებისას, სერვერი მიმართავს ინდექსს და ასრულებს საჭირო სტრიქონის ძებნას. სტრიქონის პოვნისას, მიმართვის მიხედვით შესრულდება გადასვლა ცხრილში, მოძებნილ სტრიქონზე.

არაკლასტერული ინდექსის შექმნა უმჯობესია ხშირადცვალებადი სვეტებისთვის.

ერთ ცხრილში შეიძლება განისაზღვროს 999-მდე არაკლასტერული ინდექსი. ერთი არაკლასტერული ინდექსი შეიძლება შეიცავდეს 16-მდე სვეტს, რომლებიც ერთ ცხრილს უნდა ეკუთვნოდეს. ასეთ ინდექსს შედგენილი ინდექსი ეწოდება. მასში შემავალი სვეტების მაქსიმალურად დასაშვები საერთო ზომაა 900 ბაიტი.

კლასტერული ინდექსი

ცხრილში მხოლოდ ერთი ინდექსი შეიძლება იყოს *კლასტერული* (*clustered index*). კლასტერული ინდექსის გამოყენების დროს ცხრილში ხდება მონაცემების ფიზიკური განლაგების გადაწყობა ინდექსის სტრუქტურის შესაბამისად.

კლასტერული ინდექსი მნიშვნელოვნად ზრდის მონაცემების ძებნის მწარმოებლურობას. მისი გამოყენებისას, მონაცემების მომდევნო პორცია განლაგდება უშუალოდ მოძებნილი მონაცემების შემდეგ. შედეგად აღარ შესრულდება ინდექსთან მიმართვის და სტრიქონის ძებნის ზედმეტი ოპერაცია.

კლასტერულ ინდექსად უნდა ავირჩიოთ ყველაზე ხშირად გამოყენებადი სვეტები. ის შეიძლება რამდენიმე სვეტისგან შედგებოდეს, მაგრამ ასეთი სვეტების რაოდენობა შეძლებისდაგვარად მცირე უნდა იყოს.

კლასტერული ინდექსი არ უნდა გამოვიყენოთ ხშირად ცვალებადი სვეტებისათვის, რადგან სერვერმა უნდა შეასრულოს მონაცემის ფიზიკური გადაადგლება ცხრილში, რომ ისინი მოწესრიგებულ მდგომარეობაში იმყოფებოდეს, როგორც ამას კლასტერული ინდექსი ითხოვს.

ცხრილში პირველადი გასაღების შექმნისას სერვერი მისთვის კლასტერულ ინდექსს ავტომატურად ქმნის, თუ ის ადრე არ იყო შექმნილი ან გასაღების განსაზღვრისას არ იყო აშკარად მითითებული ინდექსის სხვა ტიპი.

თუ ცხრილში განსაზღვრულია კლასტერული და არაკლასტერული ინდექსები, მაშინ არაკლასტერული ინდექსის მიმთითებელი მიმართავს სტრიქონის არა ფიზიკურ მდებარეობას, არამედ კლასტერული ინდექსის შესაბამის ელემენტს, რომელიც ამ სტრიქონს აღწერს. ეს საშუალებას გვაძლევს, არ გადავაწყოთ არაკლასტერული ინდექსების სტრუქტურა ყოველთვის, როცა კლასტერული ინდექსი ცვლის სტრიქონების ფიზიკურ განლაგებას ცხრილში. იცვლება მხოლოდ კლასტერული ინდექსი, ხოლო არაკლასტერული ინდექსები აახლებენ მხოლოდ ინდექსირებულ მნიშვნელობებს და არა მიმთითებელს.

თუ არაკლასტერული ინდექსის შექმნისას კლასტერული ინდექსი არ არის უნიკალური, მაშინ სერვერი ავტომატურად უმატებს მას დამატებით მნიშვნელობებს, რომლებიც მას უნიკალურს ხდის.

რადგან, არაკლასტერული ინდექსები შეიცავენ კლასტერულ ინდექსზე მიმართვას, ამიტომ მინიმუმამდე უნდა დავიყვანოთ კლასტერული ინდექსში შემავალი სვეტების რაოდენობა და ზომა. საწინააღმდეგო შემთხვევაში, სერვერის მწარმოებლურობა მკვეთრად

შემცირდება. ცხრილიდან კლასტერული ინდექსის წაშლისას, სერვერი გადააწყობს ყველა არაკლასტერულ ინდექსს, განსაზღვრავს რა მათთვის მიმართვებს სტრიქონების ფიზიკურ განლაგებაზე.

უნიკალური ინდექსი

უნიკალური ინდექსი (*unique indexes*) იძლევა ინდექსირებულ სვეტში 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.

შევსების ფაქტორი

შევსების ფაქტორი (*fill factor*) განსაზღვრავს განსაზღვრავს გვერდზე მონაცემების ჩაწერის სიჭიდროვეს ანუ საინდექსო გვერდების თავისუფალი სივრცის რამდენი პროცენტი იქნება შევსებული მონაცემებით. დარჩენილი თავისუფალი სივრცე თანდათანობით შეივსება ცხრილში მონაცემების დამატებასთან ერთად.

შევსების ფაქტორი საშუალებას გვაძლევს უფრო მოქნილად ვმართოთ სერვერის მუშაობა ცხრილების ინდექსირების დროს. რაც მეტია შევსების ფაქტორი, მით ნაკლები იქნება გვერდზე თავისუფალი სივრცე და მით უფრო კომპაქტურად იქნება განთავსებული ინფორმაცია ინდექსების შესახებ. შევსების ფაქტორის არჩევისას უნდა შევაფასოთ რამდენად ინტენსიურად შესრულდება ცხრილში მონაცემების ცვლილება (დამატება, წაშლა და შეცვლა). თუ ცხრილი ძირითადად მხოლოდ წაკითხვისათვის გამოიყენება, მაშინ უმჯობესი იქნება შევსების ფაქტორი 100%-თან ახლოს დავაყენოთ. ეს საშუალებას მოგვცემს მონაცემთა ბაზის სივრცე უფრო ეკონომიურად გამოვიყენოთ.

თუ ცხრილში მონაცემები ხშირად იცვლება, მაშინ შევსების ფაქტორი არ უნდა იყოს დიდი, როგორც მონაცემებისათვის, ისე ინდექსებისთვის. მონაცემთა ბაზის სივრცე ამ შემთხვევაში არ იქნება გამოყენებული ეკონომიურად, სამაგიეროდ მონაცემების ცვლილების ოპერაციების შესრულების მწარმოებლურობა მაქსიმალური იქნება. თუ ინტენსიურად გამოყენებად ცხრილში დავაყენებთ შეფასების ფაქტორის დიდ მნიშვნელობას, მაშინ სერვერი იძულებული იქნება ხშირად შეასრულოს გვერდების დახლეჩა (*page split*) ახალი მონაცემების ჩასმის მიზნით. გვერდების დახლეჩა დიდ დროს იკავებს, ამიტომ შეძლებისდაგვარად უნდა მოვერიდოთ მის გამოყენებას.

საინდექსო გვერდების შევსების ფაქტორის მნიშვნელობა ინდექსის შექმნის დროს განისაზღვრება. ცხრილში მონაცემების ცვლილებასთან ერთად იცვლება გვერდების შევსების ხარისხი. ამიტომ საჭიროა ინდექსის ზომის შემცირება, რისთვისაც პერიოდულად უნდა შევასრულოთ ინდექსის გადაწყობა.

ინდექსების გადაწყობა შრომატევადი და ხანგრძლივი პროცესია, ამიტომ შეძლებისდაგვარად ის უნდა შევასრულოთ მომხმარებლების მცირე აქტიურობის დროს, მაგალითად, ღამით. ინდექსების გადაწყობის დროს სერვერი ახდენს მონაცემების დაბლოკვას და მომხმარებლები ვერ შეძლებენ მონაცემებთან მიმართვას გადაწყობის დამთავრებამდე.

ინდექსის შექმნა

ინდექსის შექმნის უფლება აქვს მხოლოდ ცხრილის მფლობელს და ამ უფლების გადაცემა სხვა მომხმარებლისთვის არ შეიძლება.

არსებობს ინდექსის შექმნის რამდენიმე საშუალება:

– ინდექსის ავტომატურად შექმნა პირველადი გასაღების განსაზღვრისას;
– ინდექსის ავტომატურად შექმნა UNIQUE შეზღუდვის განსაზღვრისას;
– ინდექსის განსაზღვრა ცხრილის შექმნისას;
– ინდექსის შექმნა CREATE INDEX ბრძანებით.

ჩვენ განვიხილავთ ინდექსის შექმნას CREATE INDEX ბრძანების საშუალებით. მისი სინტაქსია:

CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED]
INDEX *ინდექსის_სახელი* ON { <*ობიექტი*> } (*სვეტის_სახელი* [ASC | DESC] [,...n])
[WITH
[[,] FILLFACTOR = *შევსების_ფაქტორი*]
[[,] IGNORE_DUP_KEY]
[[,] DROP_EXISTING]

<*ობიექტი*> კონსტრუქციის სინტაქსია:

<*ობიექტი*> ::=
[*მონაცემთა_ბაზის_სახელი*. [*სქემის_სახელი*] .] *ცხრილის_ან_წარმოდგენის_სახელი*

განვიხილოთ არგუმენტების დანიშნულება.

– UNIQUE არგუმენტის მითითების შემთხვევაში იქმნება უნიკალური ინდექსი. მისი შექმნისას სერვერი ასრულებს სვეტის წინასწარ შემოწმებას მნიშვნელობების უნიკალურობაზე. თუ სვეტში არის ორი ერთნაირი მნიშვნელობა, მაშინ ინდექსი არ შეიქმნება და გაიცემა შეტყობინება შეცდომის შესახებ. ამ შემთხვევაში სერვერის ქცევა არ არის დამოკიდებული IGNORE_DUP_KEY საკვანძო სიტყვის არსებობაზე. სერვერი არ წაშლის დუბლირებულ მნიშვნელობებს. თუ ინდექსი შედგენილია ანუ შეიცავს ორ ან მეტ სვეტს, მაშინ უნიკალური უნდა იყოს ყველა საინდექსირებელი სვეტის მნიშვნელობების ერთობლიობა ცხრილის ან წარმოდგენის თითოეულ სტრიქონში.

საინდექსებელ სვეტში სასურველია, აგრეთვე, NULL მნიშვნელობის შენახვის აკრძალვა, რათა ავიცილოთ მნიშვნელობების უნიკალურობასთან დაკავშირებული პრობლემები. თუ სვეტში გვხვდება ორი NULL მნიშვნელობა, მაშინ სერვერი მას ორ ერთნაირ მნიშვნელობად ჩათვლის.

უნიკალური ინდექსის განსაზღვრის შემდეგ სერვერი არ შეგვასრულებინებს ისეთ INSERT ან UPDATE ბრძანებას, რომელიც ორი ერთნაირი მნიშვნელობის არსებობას გამოიწვევს.

– CLUSTERED არგუმენტი მიუთითებს, რომ შესაქმნელი ინდექსი კლასტერული იქნება, ე.ი. ფიზიკურად მონაცემები განლაგებული იქნება ინდექსში განსაზღვრული რიგითობით.
– NONCLUSTERED არგუმენტი მიუთითებს, რომ შესაქმნელი ინდექსი არაკლასტერული იქნება. ეს არგუმენტი ავტომატურად იგულისხმება, თუ არ არის მითითებული არც CLUSTERED და არც NONCLUSTERED არგუმენტი.
– *ინდექსის_სახელი* უნიკალური უნდა იყოს ცხრილის ფარგლებში.
– *სვეტის_სახელი* განსაზღვრავს იმ სვეტებს, რომლებიც ჩართული იქნება შესაქმნელ ინდექსში. თუ მითითებულია ერთზე მეტი სვეტი, მაშინ ინდექსი იქნება შედგენილი (composite index). ერთი შედგენილი ინდექსი შეიძლება შეიცავდეს 16-მდე სვეტს. ინდექსის აგება დაუშვებელია text, ntext ან image ტიპის მქონე სვეტების ბაზაზე. შედგენილ ინდექსში ჩართული სვეტების საერთო ზომა არ უნდა აღემატებოდეს 900 ბაიტს.

სვეტების სახელების მითითების მიმდევრობა გავლენას ახდენს მოთხოვნების

შესრულების მწარმოებლურობაზე. ინდექსის საკვანძო ელემენტი შეიცავს სვეტების მნიშვნელობებს იმ მიმდევრობით, რა მიმდევრობითაც ისინი იყო ჩამოთვლილი ინდექსის შექმნის დროს. მაქსიმალური მწარმოებლურობის უზრუნველსაყოფად რეკომენდებულია თავდაპირველად მიეთითოს მინიმალური სიგრძის, შემდეგ კი - მაქსიმალური სიგრძის მქონე სახელები.

– [ASC | DESC] არგუმენტი განსაზღვრავს სვეტის მნიშვნელობების დახარისხების მეთოდს. ASC შემთხვევაში სვეტის მნიშვნელობები დალაგდება ზრდადობის მიხედვით, DESC შემთხვევაში კი - კლებადობის მიხედვით. ავტომატურად იგულისხმება ASC.

– FILLFACTOR = *შევსების_ფაქტორი* აგრუმენტი განსაზღვრავს საინდექსო გვერდების შევსების ხარისხს. სერვერი ავტომატურად არ უზრუნველყოფს შევსების ფაქტორის გარკვეულ მნიშვნელობას.

– IGNORE_DUP_KEY არგუმენტი გამოიყენება მხოლოდ უნიკალური ინდექსის შექმნის დროს. ის განსაზღვრავს სერვერის ქცევას სტრიქონების დამატების ან შეცვლის ოპერაციის შესრულების მცდელობისას, რომელიც იწვევს გამეორებადი მნიშვნელობების გაჩენას. თუ უნიკალური ინდექსის შექმნისას (როგორც კლასტერული, ისე არაკლასტერული) მითითებული იყო IGNORE_DUP_KEY არგუმენტი, მაშინ იმ ოპერაციის შესრულებისას, რომელიც იწვევს დუბლირებული მნიშვნელობების გაჩენას, სერვერი გასცემს შეტყობინებას შეცდომის შესახებ და გააუქმებს მხოლოდ იმ სტრიქონების შეცვლას, რომლებმაც დუბლირება გამოიწვიეს. თუ IGNORE_DUP_KEY არგუმენტი არ იყო მითითებული, მაშინ სერვერი, ასევე გასცემს შეტყობინებას შეცდომის შესახებ, მაგრამ აუქმებს ყველა ცვლილებას.

– DROP_EXISTING. თუ ცხრილში განსაზღვრული იყო ინდექსი იმავე სახელით, როგორიც შესაქმნელ ინდექსს ექნება, მაშინ ამ არგუმენტის გამოყენება გამოიწვევს არსებული ინდექსის ახლით შეცვლას და შეასრულებს მის გადაწყობას. უმჯობესია ეს არგუმენტი გამოვიყენოთ, ვიდრე ინდექსი ჯერ წავშლოთ და შემდეგ შევქმნათ. ეს არგუმენტი განსაკუთრებით ეფექტურია კლასტერული ინდექსების გადაწყობის დროს მაშინ, როცა ცხრილში არაკლასტერული ინდექსების დიდი რაოდენობაა. თუ, ჯერ წავშლით კლასტერულ ინდექსს, სერვერს მოუწევს ყველა არაკლასტერული ინდექსის სტრუქტურის გადაწყობა, რათა შეცვალოს ცხრილის სტრიქონებზე მიმართვები. შემდეგში, ახალი კლასტერული ინდექსის შექმნა ისევ გამოიწვევს არაკლასტერული ინდექსების გადაწყობას. DROP_EXISTING არგუმენტის მითითების შემთხვევაში სერვერი არ შეასრულებს არაკლასტერული ინდექსების შუალედურ გადაწყობას, რაც მკვეთრად ზრდის ინდექსის შექმნის სიჩქარეს. ეს არგუმენტი უნდა გამოვიყენოთ მაშინ, როცა მონაცემთა ბაზაში ცხრილისთვის ან წარმოდგენისთვის უკვე არსებობს ამავე სახელის მქონე ინდექსი. თუ ინდექსი არ არსებობს, მაშინ გაიცემა შეტყობინება შეცდომის შესახებ.

ინდექსი ყოველთვის იქმნება მხოლოდ მიმდინარე მონაცემთა ბაზის ცხრილის ან წარმოდგენისთვის.

მაგალითები.

ა. Personali ცხრილის mobiluri სვეტისთვის შევქმნათ უნიკალური Ind_mobiluri ინდექსი. პროგრამის კოდს აქვს სახე:

```
USE Shekveta;
--      თუ ინდექსი არსებობს, მაშინ ის წაიშლება
IF EXISTS
(
SELECT   name
FROM     sys.indexes
WHERE  name = N'Ind_mobiluri'
```

```
)
DROP INDEX Ind_mobiluri ON Personali
--        ინდექსის შექმნა
CREATE UNIQUE INDEX Ind_mobiluri
ON Personali (mobiluri);
```

ბ. Personali ცხრილის qalaqi და gvari სვეტებისთვის შევქმნათ Ind_qalaqi_gvari არაკლასტერული, შედგენილი ინდექსი. შევსების ფაქტორია 40%. qalaqi სვეტი დავალაგოთ კლებადობით, gvari სვეტი კი - ზრდადობით. პროგრამის კოდს აქვს სახე:

```
USE Shekveta;
--        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
--        ინდექსის შექმნა
CREATE NONCLUSTERED INDEX Ind_qalaqi_gvari
ON Personali ([qalaqi] DESC, [gvari] ASC)
     WITH FILLFACTOR = 40;
```

გ. #Cxrili ცხრილის Sveti2 სვეტისთვის შევქმნათ Ind_Sveti2 უნიკალური კლასტერული ინდექსი. გამოვიყენოთ IGNORE_DUP_KEY არგუმენტი. პროგრამის კოდს აქვს სახე:

```
USE Shekveta;
CREATE TABLE #Cxrili
(
sveti1 NVARCHAR(10),
sveti2 NVARCHAR(50),
sveti3 DATETIME
);
--        თუ ინდექსი არსებობს, მაშინ ის წაიშლება
IF EXISTS
(
SELECT name
FROM    sys.indexes
WHERE name = N'Ind_Sveti2'
)
DROP INDEX Ind_Sveti2 ON #Cxrili
--        ინდექსის შექმნა
CREATE UNIQUE INDEX Ind_Sveti2 ON #Cxrili (Sveti2)
            WITH (IGNORE_DUP_KEY = ON);

INSERT INTO #Cxrili VALUES (N'ანა', N'სამხარაძე', GETDATE());
INSERT INTO #Cxrili VALUES (N'ანა', N'სამხარაძე', GETDATE());

SELECT COUNT( * ) AS [სტრიქონების რაოდენობა]
```

```
FROM #Cxrili;
```
დ. მაგალითში ხდება DROP_EXISTING არგუმენტის გამოყენების დემონსტრირება. პროგრამის კოდს აქვს სახე:
```
USE Shekveta;
-- ინდექსის შექმნა
CREATE NONCLUSTERED INDEX Ind_qalaqi
ON Personali(qalaqi)
WITH ( DROP_EXISTING = ON);
```
Ind_qalaqi ინდექსი უნდა არსებობდეს, საწინააღმდეგო შემთხვევაში გაიცემა შეტყობინება შეცდომის შესახებ.

ინდექსების მართვა

ინდექსისთვის სახელის შეცვლა

ინდექსისთვის სახელის შესაცვლელად sp_rename შენახული პროცედურა გამოიყენება.

მაგალითი. Personali ცხრილში Ind_ganyofileba ინდექსს დავარქვათ ახალი IndGanyofileba სახელი. მოთხოვნას აქვს სახე:
```
USE Shekveta;
EXEC sp_rename 'Personali.Ind_ganyofileba', 'IndGanyofileba', 'INDEX';
```

ინდექსის წაშლა

ინდექსის წასაშლელად DROP INDEX ბრძანება გამოიყენება. მისი სინტაქსია:

DROP INDEX *ინდექსის_სახელი*, [,...n]

ერთი DROP INDEX ბრძანებით შეიძლება ნებისმიერ ცხრილში რამდენიმე ინდექსის წაშლა.

მაგალითი. Personali ცხრილიდან წავშალოთ Ind_qalaqi_gvari ინდექსი. მოთხოვნას აქვს სახე:
```
DROP INDEX Personali.Ind_qalaqi_gvari;
```
ან
```
DROP INDEX Ind_qalaqi_gvari ON Personali
```

ინდექსების გადაწყობა

დროთა განმავლობაში საინდექსო ცხრილების შევსების ხარისხი მნიშვნელოვნად იცვლება, რასაც მივყავართ ცხრილში მონაცემების ცვლილების ოპერაციების მწარმოებლურობის დაქვეითებასთან. ასეთ შემთხვევებში უნდა შევასრულოთ ინდექსის გადაწყობა საინდექსო გვერდებზე თავისუფალი ადგილის მოწესრიგების მიზნით. გარდა ამისა, ინდექსის გადაწყობა შეიძლება საჭირო გახდეს ინდექსში შემავალი სვეტების შემადგენლობის შეცვლისას.

ინდექსის გადაწყობის რამდენიმე გზა არსებობს. ერთი გზაა ინდექსის წაშლა და მისი ხელახლა შექმნა. თუ ასეთი გზით შევეცდებით კლასტერული ინდექსის გადაწყობას, მაშინ სერვერს მოუწევს ყველა არაკლასტერული ინდექსის გადაწყობა, ეს კი დიდ დროს იკავებს. ამიტომ ეს არ არის ინდექსის გადაწყობის ეფექტური გზა.

ინდექსის გადაწყობის მეორე, უფრო ეფექტური გზაა CREATE INDEX ბრძანებაში DROP_EXISTING არგუმენტის მითითება. ამ შემთხვევაში არ მოხდება ყველა კლასტერული ინდექსის გადაწყობა.

ინდექსის გადაწყობის მესამე გზაა DBCC DBREINDEX ბრძანების გამოყენება. მისი სინტაქსია:

DBCC DBREINDEX
('*მონაცემთა_ბაზის_სახელი.სქემის_სახელი.ცხრილის_სახელი*'
[, *ინდექსის_სახელი* [, *შევსების_ფაქტორი*]])

განვიხილოთ არგუმენტების დანიშნულება.

– '*მონაცემთა_ბაზის_სახელი.სქემის_სახელი.ცხრილის_სახელი*' არის იმ ცხრილის სახელი, რომელშიც უნდა მოხდეს ერთი ან მეტი ინდექსის გადაწყობა.

– *ინდექსის_სახელი* არის იმ ინდექსის სახელი, რომლის გადაწყობაც უნდა მოხდეს. თუ ინდექსის სახელი არ არის მითითებული, მაშინ მოხდება ცხრილის ყველა ინდექსის გადაწყობა.

– *შევსების_ფაქტორი* განსაზღვრავს საინდექსო გვერდების შევსების ხარისხს. თუ ეს არგუმენტი არ არის მითითებული, მაშინ გამოიყენება შევსების ფაქტორის ის მნიშვნელობა, რომელიც გამოყენებული იყო ინდექსის შექმნისას.

მაგალითი.

ა. შევასრულოთ Personali ცხრილის PK_Personali ინდექსის გადაწყობა. შევსების ფაქტორია 80:

```
USE Shekveta;
DBCC DBREINDEX ('Personali', PK_Personali, 80);
```

ბ. შევასრულოთ Personali ცხრილის ყველა ინდექსის გადაწყობა. შევსების ფაქტორია 70:

```
USE Shekveta;
DBCC DBREINDEX ('Personali', '', 70);
```

გ. შევასრულოთ Personali ცხრილის ყველა ინდექსის გადაწყობა. მივუთითოთ სქემა, რომელსაც ცხრილი ეკუთვნის:

```
USE Shekveta;
DBCC DBREINDEX ('Sqema_1.Personali', '');
```

გ. შევასრულოთ Personali ცხრილის ყველა ინდექსის გადაწყობა. მივუთითოთ მონაცემთა ბაზა, და სქემა, რომელშიც ცხრილია მოთავსებული:

```
DBCC DBREINDEX ('Shekveta.Sqema_1.Personali', '');
```

გ. შევასრულოთ Personali ცხრილის ყველა ინდექსის გადაწყობა. მივუთითოთ მონაცემთა ბაზა, რომელშიც ცხრილია მოთავსებული:

```
DBCC DBREINDEX ('Shekveta..Personali', '');
```

ინდექსის შესახებ ინფორმაციის მიღება

ინდექსის შექმნის წინ უმჯობესია მივიღოთ ინფორმაცია არსებული ინდექსების შესახებ. ინდექსების შესახებ ინფორმაციის მისაღებად გამოიყენება sp_helpindex შენახული პროცედურა. მისი სინტაქსია:

sp_helpindex [@objname =] '*ცხრილის_სახელი*'

აქ '*ცხრილის_სახელი*' არის მიმდინარე მონაცემთა ბაზის ცხრილის სახელი, რომლის ინდექსების შესახებ გვინდა ინფორმაციის მიღება.

მაგალითი. მივიღოთ ინფორმაცია Personali ცხრილის ინდექსების შესახებ. მოთხოვნას აქვს სახე:

```
EXEC sp_helpindex 'Personali';
```

შედეგი:

	index_name	index_description	index_keys
1	Ind_qalaqi_gvari	nonclustered located on PRIMARY	personaliID
2	IX_Indeqsi_PersonaliID	nonclustered located on PRIMARY	personaliID
3	PK_Personali	clustered, unique, primary key located on PRIMARY	personaliID

ინფორმაციის მისაღებად ინდექსების მიერ დაკავებული სივრცის შესახებ გამოიყენება sp_spaceused შენახული პროცედურა.

მაგალითი. მივიღოთ ინფორმაცია Shekveta მონაცემთა ბაზაში არსებული ინდექსების მიერ დაკავებული სივრცის შესახებ. მოთხოვნას აქვს სახე:

```
USE Shekveta
EXEC  sp_spaceused;
```

შედეგი:

	database_name	database_size	unallocated space
1	Shekveta	261.75 MB	0.40 MB

	reserved	data	index_size	unused
1	2280 KB	1280 KB	928 KB	72 KB

ინდექსის თვისებების სანახავად გამოიყენება INDEXPROPERTY ბრძანება, რომლის სინტაქსია:

INDEXPROPERTY(*ცხრილის_იდენტიფიკატორი, ინდექსის_სახელი, თვისება*)

აქ *ცხრილის_იდენტიფიკატორი* არის მონაცემთა ბაზაში ცხრილის საიდენტიფიკაციო ნომერი. ამ ნომრის მისაღებად შეგვიძლია გამოვიყენოთ OBJECT_ID ფუნქცია. ინდექსის თვისება შეიძლება იყოს დაყენებული (გაიცემა 1) ან არ იყოს დაყენებული (გაიცემა 0). თუ თვისებისათვის გაიცემა NULL, მაშინ თვისების მდგომარეობის განსაზღვრა შეუძლებელია. მოვიყვანოთ *თვისება* არგუმენტის ზოგიერთი მნიშვნელობა:

– Is CLustered. თუ ეს თვისება 1 მდგომარეობაშია, მაშინ ინდექსი კლასტერულია, თუ 0 მდგომარეობაშია, მაშინ - არაკლასტერული;

– IsUnique. თუ ეს თვისება 1 მდგომარეობაშია, მაშინ ინდექსი უნიკალურია, საწინააღმდეგო შემთხვევაში - არა არის უნიკალური.

მაგალითი. შევამოწმოთ Personali ცხრილის Ind_qalaqi_gvari ინდექსი არის თუ არა კლასტერული. მოთხოვნას აქვს სახე:

```
SELECT INDEXPROPERTY(OBJECT_ID('Personali'), 'Ind_qalaqi_gvari', 'IsCLUSTERED');
```

შედეგი:

	(No column name)
1	0

თავი 12. წარმოდგენები

შესავალი

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

წარმოდგენა ხშირად გამოიყენება ისეთი სვეტების დასამალად, რომელიც კონფიდენციალურ ინფორმაციას შეიცავს, მაგალითად, ხელფასი, ასაკი და ა.შ. წარმოდგენაში შეგვიძლია ისეთი სვეტების გამოჩენა, რომელთა ნახვაც ნებადართულია მომხმარებლებისთვის. თუ წარმოდგენაში ჩართული არ არის ცხრილის რომელიმე სვეტი, მაშინ ცხრილზე დადებულია *ვერტიკალური ფილტრი*. თუ მოთხოვნა შეიცავს სტრიქონების ამორჩევის პირობებს, მაშინ ცხრილზე დადებულია *ჰორიზონტალური ფილტრი*.

წარმოდგენა შეიძლება შეიცავდეს ბმული ცხრილების სვეტებს. მაგალითად, წარმოდგენა შეიძლება შეიცავდეს Personali ცხრილიდან თანამშრომლების გვარებს და Xelshekruleba ცხრილიდან მათთან გაფორმებული ხელშეკრულებების თანხებს, ან Personali ცხრილიდან თანამშრომლების გვარებს და Shemkveti ცხრილიდან მათთან ხელშეკრულების დამდები კლიენტების გვარებს და ა.შ.

წარმოდგენასთან მიმართვის დროს სერვერი ამოწმებს იმ ობიექტების არსებობას, რომლებიც საჭიროა წარმოდგენის განმსაზღვრელი SELECT მოთხოვნის შესასრულებლად. თუ მოთხოვნაში მითითებული რომელიმე ცხრილი წაშლილია, მაშინ წარმოდგენა ვერ იმუშავებს და გაიცემა შეტყობინება შეცდომის შესახებ. თუ წაშლილი ცხრილის ნაცვლად შევქმნით იმავე სახელისა და სტრუქტურის მქონე ცხრილს, მაშინ წარმოდგენა იმუშავებს. თუ ახალ ცხრილს განსხვავებული სტრუქტურა აქვს, მაშინ წარმოდგენა უნდა წავშალოთ და ხელახლა შევქმნათ.

წარმოდგენას შეუძლია მონაცემების მიღება სხვა წარმოდგენებიდან. წარმოდგენა ყოველთვის იქმნება მიმდინარე მონაცემთა ბაზაში. განაწილებული მოთხოვნების გამოყენებით შეგვიძლია მივმართოთ მიმდინარე სერვერის სხვა მონაცემთა ბაზებში შექმნილ ცხრილებსა და წარმოდგენებს.

ერთი მომხმარებლის წარმოდგენებს სხვადასხვა სახელები უნდა ჰქონდეს. გარდა ამისა, წარმოდგენისა და ცხრილის სახელი ერთმანეთს არ უნდა ემთხვეოდეს.

წარმოდგენის სვეტს სახელი ორ შემთხვევაში უნდა მივანიჭოთ:

– თუ წარმოდგენაში სვეტი მიღებულია არითმეტიკული გამოსახულების გამოთვლის გზით;

– თუ წარმოდგენაში ორ ან მეტ სვეტს შეიძლება ერთნაირი სახელი ჰქონოდა, იმის გამო, რომ წარმოდგენაში მონაწილე სხვადასხვა ცხრილს ერთნაირი სახელის მქონე სვეტები აქვს.

წარმოდგენასთან მუშაობის დროს უნდა დავიცვათ გარკვეული მოთხოვნები. წარმოდგენა არ შეიძლება მიმართავდეს დროებით ცხრილებს. არ შეიძლება დროებითი წარმოდგენის შექმნა. რადგან წარმოდგენა მონაცემებს სხვადასხვა ცხრილიდან და წარმოდგენებიდან ირჩევს, ამიტომ მისთვის არ შეიძლება განისაზღვროს მთლიანობაზე შეზღუდვები, ტრიგერები და ავტომატური მნიშვნელობები. წარმოდგენისთვის არ შეიძლება ჩვეულებრივი ან სრულტექსტოვანი ინდექსის შექმნა.

წარმოდგენის სვეტებისათვის არ შეიძლება განვსაზღვროთ ნაგულისხმევი მნიშვნელობები. აუცილებლობის შემთხვევაში, ისინი უნდა განისაზღვროს საწყისი ცხრილისთვის. წარმოდგენისთვის SELECT მოთხოვნის განსაზღვრისას არ შეიძლება OPTION, ORDER BY, COMPUTE BY, COMPUTE და INTO განყოფილებების გამოყენება. ORDER BY

განყოფილების გამოყენება შეიძლება იმ შემთხვევაში, თუ მითითებულია TOP სიტყვა.

წარმოდგენის საშუალებით შეიძლება მონაცემების ცვლილება იმ ცხრილებში, რომლებიც წარმოდგენაში მონაწილეობენ. ასეთი წარმოდგენის შესაქმნელად აუცილებელია შემდეგი მოთხოვნების დაცვა:

– წარმოდგენა უნდა შეიცავდეს მინიმუმ ერთ ცხრილს, რომელიც მითითებულია SELECT ბრძანების FROM განყოფილებაში;

– დაუშვებელია აგრეგირების ფუნქციების გამოყენება (AVG, COUNT SUM, MIN, MAX, GROUPING, STDEV, VAR, VARP), აგრეთვე, GROUP BY, UNION, DISTINCT და TOP ელემენტების გამოყენება. აგრეგირების ფუნქციების გამოყენება დასაშვებია მხოლოდ იმ შემთხვევაში, თუ მათ მიერ გენერირებული მნიშვნელობები არ შეიცვლება;

– წარმოდგენას არ უნდა ჰქონდეს გამოთვლადი სვეტები.

წარმოდგენის საშუალებით არ შეიძლება რამდენიმე ცხრილში მონაცემების შეცვლა. ერთმა UPDATE ან INSERT ბრძანებამ მონაცემები მხოლოდ ერთ ცხრილში უნდა შეცვალოს. გარდა ამისა, მონაცემების შეცვლისას უნდა გავითვალისწინოთ WITH CHECK OPTION არგუმენტის არსებობა (მას ქვემოთ განვიხილავთ).

წარმოდგენის შექმნა

წარმოდგენის შესაქმნელად CREATE VIEW ბრძანება გამოიყენება. მისი სინტაქსია:

CREATE VIEW [*სქემის_სახელი* .] *წარმოდგენის_სახელი* [(*სვეტის_სახელი* [,...n])]
[WITH { ENCRYPTION | SCHEMABINDING } [,...n]]
AS
select_*ბრძანება*
[WITH CHECK OPTION]

განვიხილოთ არგუმენტების დანიშნულება.

– *სვეტის_სახელი* იმ სვეტის სახელია, რომელიც ჩართული იქნება წარმოდგენაში. ავტომატურად, წარმოდგენაში სვეტების სახელები ემთხვევა საწყისი ცხრილების სვეტების სახელებს. წარმოდგენაში სვეტის სახელი აშკარად უნდა მივუთითოთ შემდეგ შემთხვევებში: როცა სვეტის მნიშვნელობა ფორმირდება არითმეტიკული გამოსახულების, ფუნქციის ან მუდმივას საფუძველზე; როცა ორ ან მეტ სვეტს აქვს ერთნაირი სახელი; როცა წარმოდგენის სვეტს უნდა ჰქონდეს იმ სვეტისაგან განსხვავებული სახელი, რომლისგანაც ის მიიღება.

– ENCRYPTION არგუმენტი მიუთითებს, რომ უნდა მოხდეს წარმოდგენის კოდის დაშიფვრა.

– SCHEMABINDING არგუმენტი თუ მთითებულია წარმოდგენის შექმნისას, მაშინ სერვერი წარმოდგენის სტრუქტურას დააკავშირებს იმ ობიექტების სტრუქტურასთან, რომლებსაც SELECT მოთხოვნა მიმართავს. ასეთი დაკავშირება იძლევა იმის გარანტიას, რომ მომხმარებლები საწყისი ობიექტების შეცვლას ვერ შეძლებენ, თუ ისინი წარმოდგენის მუშაობას დაარღვევენ. თუ ეს არგუმენტი მითითებულია, მაშინ SELECT მოთხოვნაში ობიექტის სახელის გარდა უნდა მივუთითოთ მისი მფლობელის სახელიც, თუნდაც ის იყოს dbo. წარმოდგენის სტრუქტურისა და საწყისი ობიექტების დაკავშირება არის აუცილებელი მოთხოვნა წარმოდგენის ინდექსირებისთვის.

– WITH CHECK OPTION არგუმენტი მიუთითებს, რომ უნდა შესრულდეს იმ ცვლილებების შემოწმება, რომლებიც წარმოდგენის საშუალებით შესრულდება. ეს იმას ნიშნავს, რომ დაუშვებელია ისეთი ცვლილებების შესრულება, რომლებიც გამოიწვევს წარმოდგენიდან სტრიქონის წაშლას. ეს მოხდება მაშინ, როცა წარმოდგენისთვის დაყენებულია ჰორიზონტალური ფილტრი და მონაცემების ცვლილება იწვევს იმას, რომ

სტრიქონი აღარ შეესაბამება დაყენებულ ფილტრს.

წარმოდგენის შექმნისას მისი სახელი შეინახება მიმდინარე მონაცემთა ბაზის sys.objects სისტემურ წარმოდგენაში (SELECT name FROM sys.objects ORDER BY name), ხოლო კოდი კი - syscomments სისტემურ წარმოდგენაში (SELECT text FROM syscomments). წარმოდგენაში განსაზღვრული სვეტების სია ინახება sys.columns სისტემურ წარმოდგენაში (SELECT name FROM sys.columns ORDER BY name), ხოლო ინფორმაცია წარმოდგენის საწყის ცხრილებზე დამოკიდებულების შესახებ კი - sysdepends სისტემურ წარმოდგენაში (SELECT * FROM sysdepends).

მაგალითი.

ა. შევქმნათ წარმოდგენა, რომელშიც გამოჩნდება Xelshekruleba ცხრილის ყველა სვეტი. წარმოდგენაში უნდა გამოჩნდეს ინფორმაცია მევალეების შესახებ. პროგრამის კოდს აქვს სახე:

```
USE Shekveta;
--	თუ წარმოდგენა არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID('View_1', 'VIEW') IS NOT NULL
DROP VIEW View_1;
GO
--	წარმოდგენის შექმნა
CREATE VIEW View_1
AS
SELECT  *
FROM    Xelshekruleba
WHERE vali_l > 0;
```

წარმოდგენის გახსნა:

```
SELECT * FROM View_1;
```

შედეგი:

	xelshekrulebaID	personaliID	shemkvetiID	gadasaxdeli_l	gadasaxdeli_d	gadaxdili_l
1	17	5	NULL	5300	2849,4623655914	4800
2	18	6	12	4400	2365,59139784946	3900
3	20	9	14	8500	4569,89247311828	8400
4	2	2	2	3700	1989,24731182796	3000
5	3	2	2	8000	4301,0752688172	4300
6	6	3	3	8200	4480,87431693989	8000
7	8	3	4	8000	4444,44444444444	1600
8	9	1	3	6000	3333,33333333333	5500
9	10	1	2	9800	5444,44444444444	9000
10	14	3	1	2000	1176,47058823529	1700

ბ. შევქმნათ წარმოდგენა, რომელშიც გამოჩნდება Personali ცხრილის gvari და qalaqi სვეტები, და Xelshekruleba ცხრილის gadasaxdeli_l სვეტი. Personali ცხრილი არის მთავარი, Xelshekruleba ცხრილი - კი დამოკიდებული. გამოვიტანოთ ინფორმაცია იმ თანამშრომლების შესახებ, რომელთა ასაკი 45-ზე ნაკლებია. თანამშრომლების გვარები კლებადობით დავალაგოთ. პროგრამის კოდს აქვს სახე:

```
USE Shekveta;
--	თუ წარმოდგენა არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID('View_1', 'VIEW') IS NOT NULL
DROP VIEW View_1;
```

```
GO
--        წარმოდგენის შექმნა
CREATE VIEW View_1
AS
SELECT TOP 100  P.gvari, P.qalaqi, X.gadasaxdeli_l
FROM   Personali P INNER JOIN Xelshekruleba X
          ON P.personaliID = X.personaliID
WHERE P.asaki < 45
ORDER BY P.gvari DESC;
```

წარმოდგენის გახსნა:

```
SELECT * FROM View_1;
```

შედეგი:

	gvari	qalaqi	gadasaxdeli_l
1	შენგელია	ზუგდიდი	5300
2	სამხარაძე	ზუგდიდი	6000
3	სამხარაძე	ზუგდიდი	9800
4	სამხარაძე	ზუგდიდი	7100
5	სამხარაძე	ბათუმი	5000
6	სამხარაძე	ბათუმი	3700
7	სამხარაძე	ბათუმი	8000

გ. მაგალითში ხდება ENCRYPTION არგუმენტის გამოყენების დემონსტრირება. შედეგად View_1 წარმოდგენის კოდი დაიშიფრება. პროგრამის კოდს აქვს სახე:

```
USE Shekveta;
--        თუ წარმოდგენა არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID('View_1', 'VIEW') IS NOT NULL
DROP VIEW View_1;
GO
--        წარმოდგენის შექმნა
CREATE VIEW View_1 WITH ENCRYPTION
AS
SELECT  *
FROM    Xelshekruleba
WHERE vali_l > 0;
```

წარმოდგენის გახსნა:

```
SELECT * FROM View_1;
```

დ. მაგალითში ხდება WITH CHECK OPTION არგუმენტის გამოყენების დემონსტრირება. პროგრამის კოდს აქვს სახე:

```
USE Shekveta;
--        თუ წარმოდგენა არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID('View_5', 'VIEW') IS NOT NULL
DROP VIEW View_5;
GO
--        წარმოდგენის შექმნა
CREATE VIEW View_5
AS
```

```
SELECT TOP 100  P.gvari, P.qalaqi, X.gadasaxdeli_l
FROM   Personali P INNER JOIN Xelshekruleba X
          ON P.personaliID = X.personaliID
WHERE P.asaki < 45
ORDER BY P.gvari DESC
WITH CHECK OPTION;
```

წარმოდგენის გახსნა:

```
SELECT * FROM View_5;
```

შედეგი:

	gvari	qalaqi	gadasaxdeli_l
1	შენგელია	ზუგდიდი	5300
2	სამხარაძე	ზუგდიდი	6000
3	სამხარაძე	ზუგდიდი	9800
4	სამხარაძე	ზუგდიდი	7100
5	სამხარაძე	ბათუმი	5000
6	სამხარაძე	ბათუმი	3700
7	სამხარაძე	ბათუმი	8000

ე. მაგალითში ხდება ფუნქციის გამოყენების დემონსტრირება წარმოდგენის შიგნით. პროგრამის კოდს აქვს სახე:

```
USE Shekveta;
--       თუ წარმოდგენა არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID('View_6', 'VIEW') IS NOT NULL
DROP VIEW View_6;
GO
--       წარმოდგენის შექმნა
CREATE VIEW View_6
AS
SELECT PersonaliID, MAX(vali_l) AS [მაქსიმალური ვალი]
FROM   Xelshekruleba
GROUP BY PersonaliID;
```

წარმოდგენის გახსნა:

```
SELECT * FROM View_6;
```

შედეგი:

	PersonaliID	მაქსიმალური ვალი
1	1	800
2	2	3700
3	3	6400
4	4	0
5	5	500
6	6	500
7	7	0
8	8	0
9	9	100
10	10	0

წარმოდგენის მართვა

წარმოდგენის სახელის შეცვლა

წარმოდგენის სახელის შესაცვლელად გამოიყენება sp_rename შენახული პროცედურა.

მაგალითი. View_1 წარმოდგენას დავარქვათ View_2 სახელი:

```
EXEC sp_rename  View_1, View_2, 'OBJECT';
```

წარმოდგენის წაშლა

წარმოდგენის წასაშლელად გამოიყენება DROP VIEW ბრძანება. მისი სინტაქსია:

DROP VIEW [*სქემის_სახელი* .] *წარმოდგენის_სახელი* [,...n]

წარმოდგენის წაშლის უფლება აქვს მხოლოდ მის მფლობელს და ეს უფლება არ შეიძლება სხვას გადაეცეს. თუ საჭიროა, რომ სხვა მომხმარებელმა წაშალოს წარმოდგენა, მაშინ მას უნდა გადავცეთ წარმოდგენის ფლობის უფლება. db_owner და sysadmin როლის წევრებს შეუძლიათ წარმოდგენების წაშლა. ამ საკითხებს 21-ე თავში განვიხილავთ.

მაგალითი. წავშალოთ View_1 წარმოდგენა. მოთხოვნას აქვს სახე:

```
DROP VIEW View_1;
```

წარმოდგენის შესახებ ინფორმაციის მიღება

წარმოდგენის შესახებ ინფორმაციის მისაღებად გამოიყენება sp_help შენახული პროცედურა.

მაგალითი. მივიღოთ ინფორმაცია View_1 წარმოდგენის შესახებ. მოთხოვნას აქვს სახე:

```
EXEC sp_help 'View_1';
```

ამ ბრძანების შესრულების შედეგად გაიცემა ცხრილი, რომელიც შეიცავს წარმოდგენის სვეტებს მათი თვისებების აღწერით.

კოდის მისაღებად, რომლის საშუალებითაც შეიქმნა წარმოდგენა, შეგვიძლია გამოვიყენოთ sp_helptext შენახული პროცედურა. მისი სინტაქსია:

sp_helptext [@objname =] '*წარმოდგენის_სახელი*'

მაგალითი. მივიღოთ პროგრამული კოდი, რომლის გამოყენებით შეიქმნა View_1 წარმოდგენა. მოთხოვნას აქვს სახე:

```
EXEC sp_helptext 'View_1';
```

შედეგი:

	Text
1	-- წარმოდგენის შექმნა
2	CREATE VIEW View_1
3	AS
4	SELECT TOP 100 P.gvari, P.qalaqi, X.gadasaxdeli_I
5	FROM Personali P INNER JOIN Xelshekruleba X
6	ON P.personaliID = X.personaliID
7	WHERE P.asaki < 45
8	ORDER BY P.gvari DESC;

წარმოდგენის დამოკიდებულებების ნახვა

იმ ობიექტების სანახავად, რომლებზეც წარმოდგენაა დამოკიდებული, შეგვიძლია sp_depends შენახული პროცედურის გამოყენება. ამ შენახულ პროცედურას არ გამოაქვს ინფორმაცია იმ ობიექტების შესახებ, რომლებიც სხვა მონაცემთა ბაზაშია მოთავსებული.

მაგალითი. მივიღოთ იმ ობიექტების სია, რომლებზეც View_1 წარმოდგენაა დამოკიდებული. მოთხოვნას აქვს სახე:

```
EXEC sp_depends 'View_1';
```

შედეგი:

	name	type	updated	selected	column
1	dbo.Xelshekruleba	user table	no	yes	personaliID
2	dbo.Xelshekruleba	user table	no	yes	gadasaxdeli_I
3	dbo.Personali	user table	no	yes	personaliID
4	dbo.Personali	user table	no	yes	gvari
5	dbo.Personali	user table	no	yes	qalaqi
6	dbo.Personali	user table	no	yes	asaki

თავი 13. კურსორები

შესავალი

მოთხოვნის შესრულების შედეგად სერვერს შეუძლია კლიენტის პროგრამას ასობით ათასი სტრიქონი დაუბრუნოს. კლიენტის პროგრამები ყოველთვის ვერ ახერხებს ასეთი მოცულობის მონაცემებთან მუშაობას, რადგან მათ შესანახად დიდი ზომის მეხსიერებაა საჭირო. ასეთი პრობლემების გადასაწყვეტად კურსორები გამოიყენება. კურსორი კლიენტის პროგრამას საშუალებას აძლევს იმუშაოს არა ასობით ან ათასობით, არამედ ერთ სტრიქონთან ან სტრიქონების მცირე ბლოკთან.

ცხრილში მოთავსებული სტრიქონების ნაკრებს *მონაცემთა სრული ნაკრები* (*complete set of rows*) ეწოდება, SELECT ბრძანების მიერ გაცემული სტრიქონების ნაკრებს კი - *შედეგობრივი ნაკრები* (*rezult set*). სტრიქონების შედეგობრივი ნაკრები არის მონაცემთა სრული ნაკრების ნაწილი, გაფილტრული ჰორიზონტალურად WHERE განყოფილების გამოყენებით. დასაშვებია ვერტიკალური ფილტრის გამოყენებაც, თუ მოთხოვნაში არ ჩავრთავთ სვეტების ნაწილს. კურსორები მუშაობს მონაცემების შედეგობრივ ნაკრებთან და ჩვენ საშუალებას გვაძლევს ვიმუშაოთ ამ ნაკრების სხვადასხვა ნაწილთან.

კურსორები მხოლოდ აუცილებლობის შემთხვევაში უნდა გამოვიყენოთ. ჯერ ერთი, ისინი საშუალებას არ გვაძლევენ მონაცემების დამუშავების ოპერაციები შევასრულოთ მონაცემთა მთელ ნაკრებზე, და მეორე, კურსორების საშუალებით მონაცემების დამუშავების სიჩქარე გაცილებით დაბალია სერვერის სტანდარტულ საშუალებებთან შედარებით.

კურსორების რეალიზება

არსებობს სამი სახის კურსორი:

– *Transact-SQL-ის კურსორები* (*Transact-SQL cursors*). ამ სახის კურსორები, ძირითადად, ტრიგერების, შენახული პროცედურებისა და Transact-SQL-ის კოდის შიგნით გამოიყენება.

– *სერვერის კურსორები* (*API Server Cursors*). ამ სახის კურსორები ახდენს პროგრამა-დანართების პროგრამული ინტერფეისის რეალიზებას ODBC, OLE DB და DB-Library სისტემებისთვის. კლიენტი API ინტერფეისის საშუალებით სერვერს უგზავნის კურსორის ერთ-ერთი დასაშვები ოპერაციის შესრულების მოთხოვნას. მოთხოვნა მუშავდება სერვერის API ინტერფეისის მიერ და შემდეგ სრულდება.

– *კლიენტის კურსორები* (*Client cursors*). ამ სახის კურსორების რეალიზებას ახდენს კლიენტი. კლიენტის კურსორი იღებს სერვერის მიერ გაცემულ შედეგობრივ სტრიქონებს და მათ ლოკალურად ინახავს. ეს აჩქარებს მონაცემების დამუშავებას, რადგან მცირდება დროის დანახარჯები ქსელური ოპერაციების შესრულებაზე.

ერთი კურსორი შეიძლება მუშაობდეს სხვადასხვა ბაზაში მოთავსებულ რამდენიმე ცხრილთან. კურსორში მოთავსებული სტრიქონების წაკითხვის ოპერაციას *ამორჩევა* (*fetch*) ეწოდება. თუ ერთი ოპერაციის შესრულების შედეგად კურსორი საშუალებას გვაძლევს ამოვარჩიოთ რამდენიმე სტრიქონი, მაშინ ასეთ კურსორს *ბლოკური* ეწოდება.

კურსორები შეიძლება ორ კატეგორიად დაიყოს: *მიმდევრობითი* (*forward-only*) და *გადახვევადი* (*scrollable*). მიმდევრობითი კურსორები საშუალებას გვაძლევენ მონაცემები მიმდევრობით ამოვარჩიოთ მხოლოდ ერთი მიმართულებით - დასაწყისიდან ბოლოსკენ. გადახვევადი კურსორები საშუალებას გვაძლევს მონაცემები ამოვარჩიოთ ორივე მიმართულებით და მივმართოთ ნებისმიერ სტრიქონს.

კურსორის ტიპები

არსებობს კურსორის ოთხი ტიპი, რომლებიც ერთმანეთისაგან შესაძლებლობებით განსხვავდება. კურსორის ტიპი განისაზღვრება მისი შექმნის დროს და შემდეგ აღარ შეიცვლება.

სტატიკური კურსორები

სტატიკური კურსორის (*static cursor*) გახსნისას სერვერი სტრიქონების მთელი შედეგობრივი ნაკრების ასლს tempdb სისტემურ მონაცემთა ბაზაში ინახავს. სერვერი ბლოკავს ამ სტრიქონებს კურსორის გახსნის მომენტში. სტატიკური კურსორი არ იცვლება შექმნის შემდეგ და ყოველთვის ასახავს მონაცემების იმ ნაკრებს, რომელიც არსებობდა კურსორის გახსნის მომენტში.

სტატიკური კურსორი ყოველთვის იხსნება „მხოლოდ წაკითხვის" რეჟიმში. შედეგად, თუ სხვა მომხმარებლები საწყის ცხრილში ამატებენ, ცვლიან ან შლიან კურსორში შესულ სტრიქონებს, მაშინ ეს ცვლილებები სტატიკურ კურსორში არ აისახება.

დინამიკური კურსორები

დინამიკური კურსორის (*dynamic cursor*) გამოყენების დროს სრულდება სტრიქონების დინამიკურად ამორჩევა საწყისი ცხრილიდან მხოლოდ მაშინ, როცა ხდება მომხმარებლის მიერ ამ სტრიქონებთან მიმართვა. ამორჩევის მომენტში სერვერი სტრიქონებს ბლოკავს. კურსორის შედეგობრივ ნაკრებში მომხმარებლების მიერ შეტანილი ცვლილებები კურსორში გამოჩნდება ამორჩევის დროს. თუ სხვა მომხმარებელმა სტრიქონი შეცვალა კურსორში მისი ამორჩევის შემდეგ, მაშინ ეს ცვლილებები კურსორში არ აისახება.

კურსორში შეტანილი ცვლილებები უხილავია სხვა მომხმარებლებისთვის მანამ, სანამ კურსორი არ იქნება *დადასტურებული* (*committed*). მაგრამ, თუ კურსორის იზოლირების დონე უშვებს „ჭუჭყიან" კითხვას, მაშინ სხვა მომხმარებლები შეძლებენ კურსორიდან მონაცემების წაკითხვას შუალედურ მდგომარეობებში.

მიმდევრობითი კურსორები

მიმდევრობითი კურსორები (*forward-only cursors*) საშუალებას გვაძლევენ სტრიქონები ამოვირჩიოთ მხოლოდ დასაწყისიდან ბოლოსკენ. მიმდევრობითი კურსორი არ ინახავს სტრიქონების მთელ ნაკრებს. ცხრილიდან მაშინ სრულდება სტრიქონების წაკითხვა, როცა ისინი კურსორში ამოირჩევა. ეს იძლევა მომხმარებლის მიერ ცხრილში INSERT, UPDATE და DELETE ბრძანებებით შეტანილი ყველა ცვლილების დინამიკური ასახვის შესაძლებლობას. კურსორში გამოჩნდება მონაცემების უკანასკნელი მდგომარეობა.

საგასაღებო კურსორები

საგასაღებო კურსორები ან *გასაღებების ნაკრებზე დამოკიდებული კურსორები* (*keyset-driven cursors*) აგებულია უნიკალური იდენტიფიკატორების (გასაღებების) ბაზაზე. ცხრილის სტრიქონების უნიკალური იდენტიფიკატორების სიმრავლეს *გასაღებების სიმრავლე* (*keyset*) ეწოდება.

საგასაღებო კურსორი წარმოადგენს გასაღებების ნაკრებს, რომელიც ახდენს კურსორის მთელი შედეგობრივი ნაკრების სტრიქონების იდენტიფიცირებას. გასაღებების ნაკრები ინახება tempdb სისტემურ მონაცემთა ბაზაში.

რადგან ინფორმაცია ინახება მხოლოდ კურსორის მთელ შედეგობრივ ნაკრებში ჩართული სტრიქონების საგასაღებო სვეტების შესახებ, ამიტომ საგასაღებო კურსორები ასახავს სხვა მომხმარებლების მიერ შეტანილ ყველა ცვლილებას. კურსორის გახსნის შემდეგ დამატებული სტრიქონები არ გამოჩნდება კურსორში. კურსორში ჩართული სტრიქონები, მაგრამ წაშლილი სხვა მომხმარებლების მიერ, გამოჩნდება როგორც დაზიანებული (row missing).

გასაღებების ცხრილის შედგენის მომენტში სერვერი საწყისი ცხრილის სტრიქონებს ბლოკავს.

კურსორების მართვა

კურსორთან მუშაობის დროს შეგვიძლია ხუთი ძირითადი ოპერაცია შევასრულოთ:

– კურსორის შექმნა. კურსორი უნდა შევქმნათ მის გამოყენებამდე.

– კურსორის გახსნა. შექმნილი კურსორი მონაცემებს არ შეიცავს. გახსნის ოპერაცია კურსორს მონაცემებით ავსებს.

– სტრიქონების ამორჩევა კურსორიდან და მათი შეცვლა კურსორის საშუალებით. სტრიქონებით კურსორის შევსების შემდეგ შეგვიძლია მათთან მუშაობა. კურსორის ტიპზე დამოკიდებულებით შეგვეძლება სტრიქონების ან მხოლოდ წაკითხვა, ან წაკითხვა და შეცვლა.

– კურსორის დახურვა. კურსორთან მუშაობის დამთავრების შემდეგ ის უნდა დავხუროთ. ამ დროს სერვერი ათავისუფლებს სივრცეს tempdb მონაცემთა ბაზაში, რომელიც კურსორს გამოეყო შექმნის დროს.

– კურსორის გათავისუფლება. ამ დროს კურსორი იშლება.

კურსორის შექმნა

კურსორის შესაქმნელად გამოიყენება DECLARE CURSOR ბრძანება. არსებობს ამ ბრძანების ორი ფორმატი: განსაზღვრული SQL-92 სტანდარტით და Transact-SQL-ით. 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]] }]

განვიხილოთ არგუმენტების დანიშნულება.

– INSENSITIVE არგუმენტი მიუთითებს, რომ უნდა შეიქმნას სტატიკური კურსორი. თუ ის მითითებული არ არის, მაშინ შეიქმნება დინამიკური კურსორი.

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

– FOR select_*ბრძანება* არგუმენტი შეიცავს SELECT მოთხოვნას, რომელიც გასცემს კურსორის სტრიქონების შედეგობრივ ნაკრებს. SELECT მოთხოვნა არ უნდა შეიცავდეს INTO, FOR BROWSE და COMPUTE BY განყოფილებებს. თუ მოთხოვნა კონფლიქტშია კურსორის ტიპთან, მაშინ სერვერი შეასრულებს კურსორის არაცხად გარდაქმნას თავსებად ტიპში.

– FOR READ ONLY. თუ ეს არგუმენტი მითითებულია, მაშინ შეიქმნება „მხოლოდ წაკითხვადი” კურსორი. „მხოლოდ წაკითხვადი” კურსორი განსხვავდება სტატიკური კურსორისაგან. „მხოლოდ წაკითხვადი” შეიძლება იყოს დინამიკური კურსორიც. ამ

შემთხვევაში შესაძლებელი იქნება სხვა მომხმარებლების მიერ შესრულებული ცვლილებების ასახვა.

– FOR UPDATE [OF *სვეტის_სახელი* [,...n]]. თუ ის მითითებულია, მაშინ კურსორში შესაძლებელი იქნება სტრიქონების ცვლილება. თუ მითითებული არ არის OF *სვეტის_სახელი* არგუმენტი, მაშინ შესაძლებელია კურსორის ყველა სვეტის შეცვლა. საწინააღმდეგო შემთხვევაში, დასაშვები იქნება მხოლოდ *სვეტის_სახელი* სიაში მითითებული სვეტების შეცვლა.

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]]]

განვიხილოთ არგუმენტების დანიშნულება.

– LOCAL მიუთითებს, რომ შეიქმნება ლოკალური კურსორი, რომელიც ხილული იქნება მხოლოდ ამ კურსორის შემქმნელი პაკეტის, ტრიგერის, შენახული პროცედურის ან მომხმარებლის მიერ შექმნილი ფუნქციის შიგნით. პაკეტის, ტრიგერის ან შენახული პროცედურის დამთავრებისთანავე კურსორი წაიშლება. კურსორის შემცველობის გადასაცემად მისი შემქმნელი კონსტრუქციის გარეთ, საჭიროა ის მივანიჭოთ OUTPUT პარამეტრს.

– GLOBAL მიუთითებს, რომ შეიქმნება გლობალური კურსორი, რომელიც იარსებებს მიმდინარე შეერთების დახურვამდე.

– FORWARD_ONLY მიუთითებს, რომ შეიქმნება მიმდევრობითი კურსორი.

– SCROLL მიუთითებს, რომ შეიქმნება გადახვევადი კურსორი.

– STATIC მიუთითებს, რომ შეიქმნება სტატიკური კურსორი.

– KEYSET მიუთითებს, რომ შეიქმნება საგასაღებო კურსორი.

– DYNAMIC მიუთითებს, რომ შეიქმნება დინამიკური კურსორი.

– FAST_FORWARD. თუ ის მითითებულია READ_ONLY არგუმენტთან ერთად, მაშინ მოხდება შექმნილი კურსორის ოპტიმიზება მონაცემებთან სწრაფი მიმართვის მიზნით. ეს არგუმენტი არ შეიძლება გამოყენებული იყოს FORWARD_ONLY და OPTIMISTIC არგუმენტებთან ერთად.

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

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

მაგალითები.

ა. შევქმნათ კურსორი. მასში მოვათავსოთ მევალეების შემცველი სტრიქონები. კურსორიდან წავიკითხოთ ერთი სტრიქონი. შემდეგ კურსორი ჯერ დავხუროთ, შემდეგ კი - წავშალოთ. პროგრამის კოდს აქვს სახე:

```
USE Shekveta;
DECLARE Xelshekruleba_Cursori CURSOR
FOR
   SELECT *
```

```
    FROM Xelshekruleba
    WHERE vali_l > 0;
OPEN   Xelshekruleba_Cursori;
FETCH NEXT FROM Xelshekruleba_Cursori;
CLOSE Xelshekruleba_Cursori;
DEALLOCATE Xelshekruleba_Cursori;
```

შედეგი:

	Id	personaliID	shemkvetiID	gadasaxdeli_l	gadasaxdeli...	gadaxdili_l	gadaxdili_d	vali_l	vali_
1	2	1	13	3700	1989,25	3000	1612,9	700	378,3

ბ. შევადგინოთ კურსორი, რომელიც ეკრანზე გამოიტანს შემსრულებლებსა და მათი ხელშეკრულებების ნომრებსა და თანხებს. საჭირო იქნება ჩადგმული კურსორების გამოყენება გამოსასვლელი ფორმის ფორმირებისათვის. თითოეული შემსრულებლისათვის განსაზღვრულია შიგა კურსორი. პროგრამის კოდს აქვს სახე:

```
USE  Shekveta;
SET NOCOUNT ON;
DECLARE @personaliID INT, @gadasaxdeli_l NVARCHAR(30), @xelshekrulebaID INT,
            @gvari_P NVARCHAR(30), @saxeli_P NVARCHAR(30), @message NVARCHAR(80);
PRINT N'======== შემსრულებლები და ხელშეკრულებები ========';
DECLARE Kursori_1 CURSOR
FOR SELECT personaliID, gvari, saxeli FROM Personali ORDER BY gvari, saxeli;
OPEN Kursori_1;
FETCH NEXT FROM Kursori_1 INTO @personaliID, @gvari_P, @saxeli_P;
WHILE @@FETCH_STATUS = 0
BEGIN
  PRINT ' ';
  SELECT @message = N'***** შემსრულებელი:   ' + @gvari_P + '  ' + @saxeli_P;
  PRINT @message;
  -- შიგა კურსორის გამოცხადება
DECLARE Kursori_2 CURSOR
FOR
    SELECT DISTINCT xelshekrulebaID, X.gadasaxdeli_l
    FROM Personali P, Xelshekruleba X
    WHERE @personaliID = X.personaliID
    ORDER BY xelshekrulebaID, X.gadasaxdeli_l
  OPEN Kursori_2;
  FETCH NEXT FROM Kursori_2 INTO @xelshekrulebaID, @gadasaxdeli_l;
  IF @@FETCH_STATUS <> 0
    PRINT N'        ხელშეკრულება არ არის   ';
  WHILE @@FETCH_STATUS = 0
  BEGIN
    SELECT @message = N'ხელშეკრულების ნომერი ' +
    STR(@xelshekrulebaID) + '  ' + N'----- თანხა:    ' + STR(@gadasaxdeli_l);
    PRINT @message;
    FETCH NEXT FROM Kursori_2 INTO @xelshekrulebaID, @gadasaxdeli_l;
```

```
  END
  CLOSE Kursori_2;
  DEALLOCATE Kursori_2;
  -- მომდევნო შემსრულებელზე გადასვლა
  FETCH NEXT FROM Kursori_1 INTO @personaliID, @gvari_P, @saxeli_P;
END
CLOSE Kursori_1;
DEALLOCATE Kursori_1;
```

შედეგი:

```
======== შემსრულებლები და ხელშეკრულებები ========

***** შემსრულებელი:    გაჩეჩილაძე  ლია
ხელშეკრულების ნომერი          4  ----- თანხა:          1900
ხელშეკრულების ნომერი          5  ----- თანხა:          2500
ხელშეკრულების ნომერი          6  ----- თანხა:          8200
ხელშეკრულების ნომერი          8  ----- თანხა:          8000
ხელშეკრულების ნომერი         12  ----- თანხა:          3000
ხელშეკრულების ნომერი         14  ----- თანხა:          2000

***** შემსრულებელი:    გაჩეჩილაძე  ხათუნა
ხელშეკრულების ნომერი         15  ----- თანხა:          4400

***** შემსრულებელი:    გიორგაძე  გივი
ხელშეკრულების ნომერი         13  ----- თანხა:         10000
ხელშეკრულების ნომერი         20  ----- თანხა:          8500

***** შემსრულებელი:    კაპანაძე  ეთერი
ხელშეკრულების ნომერი         18  ----- თანხა:          4400

***** შემსრულებელი:    კიკნაძე  მზია
ხელშეკრულების ნომერი         16  ----- თანხა:          3900

***** შემსრულებელი:    სამხარაძე  ანა
ხელშეკრულების ნომერი          1  ----- თანხა:          5000
ხელშეკრულების ნომერი          2  ----- თანხა:          3700
ხელშეკრულების ნომერი          3  ----- თანხა:          8000

***** შემსრულებელი:    სამხარაძე  გიორგი
          ხელშეკრულება არ არის
```

კურსორის გახსნა

კურსორის გასახსნელად და მონაცემებით შესავსებად გამოიყენება OPEN ბრძანება. მისი სინტაქსია:

OPEN { { [GLOBAL] *კურსორის_სახელი* } | @cursor_variable_name }

@cursor_variable_name ცვლადი უნდა შეიცავდეს კურსორის სახელს. თუ საჭიროა გლობალური კურსორის გახსნა, მაშინ უნდა მივუთითოთ GLOBAL არგუმენტი.

მაგალითი. გავხსნათ curs1 კურსორი:

```
OPEN curs1;
```

მონაცემების წაკითხვა

კურსორიდან მონაცემების წასაკითხად გამოიყენება FETCH ბრძანება. მისი სინტაქსია:

FETCH [
[NEXT | PRIOR | FIRST | LAST | ABSOLUTE { n | @nvar } | RELATIVE { n | @nvar }]
FROM]
{ { [GLOBAL] *კურსორის_სახელი* } | @cursor_variable_name }
[INTO @variable_name [,...n]]

განვიხილოთ არგუმენტების დანიშნულება.

– FIRST. მისი მითითების შემთხვევაში გაიცემა კურსორის შედეგობრივი ნაკრების პირველი სტრიქონი, რომელიც მიმდინარე გახდება.
– LAST. მისი მითითების შემთხვევაში გაიცემა კურსორის შედეგობრივი ნაკრების უკანასკნელი სტრიქონი, რომელიც მიმდინარე გახდება.
– NEXT. მისი მითითების შემთხვევაში გაიცემა კურსორის შედეგობრივი ნაკრების მიმდინარე სტრიქონის შემდეგ მოთავსებული სტრიქონი, რომელიც მიმდინარე გახდება.
– PRIOR. მისი მითითების შემთხვევაში გაიცემა კურსორის შედეგობრივი ნაკრების მიმდინარე სტრიქონის წინ მოთავსებული სტრიქონი, რომელიც მიმდინარე გახდება.
– ABSOLUTE { n | @nvar }. გასცემს სტრიქონს კურსორის მთლიან შედეგობრივ ნაკრებში მისი აბსოლუტური რიგითი ნომრის მიხედვით. სტრიქონის ნომერი შეიძლება მივუთითოთ მუდმივას ან ცვლადის სახით. ცვლადს მთელი ტიპი უნდა ჰქონდეს. შეიძლება როგორც დადებითი, ისე უარყოფითი მნიშვნელობების მითითება. თუ მითითებულია დადებითი მნიშვნელობა, მაშინ სტრიქონი გადაითვლება ნაკრების დასაწყისიდან (პირველი სტრიქონიდან), თუ მითითებულია უარყოფითი მნიშვნელობა, მაშინ სტრიქონი გადაითვლება ნაკრების ბოლოდან (უკანასკნელი სტრიქონიდან). არჩეული სტრიქონი ხდება მიმდინარე. თუ მითითებულია ნულოვანი მნიშვნელობა, მაშინ სტრიქონი არ გაიცემა.
– RELATIVE { n | @nvar }. გასცემს სტრიქონს, რომელიც მდებარეობს მიმდინარე სტრიქონიდან n სტრიქონით წინ, თუ n დადებითია და n სტრიქონით უკან, თუ n უარყოფითია. გაცემული სტრიქონი ხდება მიმდინარე. თუ მითითებულია ნულოვანი მნიშვნელობა, მაშინ მიმდინარე სტრიქონი გაიცემა.
– INTO @variable_name [,...n]. ეს არგუმენტი მიუთითებს იმ ცვლადების სიას, რომლებიც შეიცავენ დასაბრუნებელი სტრიქონის სვეტების შესაბამის მნიშვნელობებს. ცვლადების მითითების მიმდევრობა უნდა შეესაბამებოდეს კურსორში სვეტების მიმდევრობას. ამასთან, ცვლადის ტიპი უნდა ემთხვეოდეს სვეტის ტიპს. თუ ეს არგუმენტი მითითებული არ არის, მაშინ მონაცემები ეკრანზე გამოიცემა.

მაგალითები.

ა. შევადგინოთ კურსორი, რომელიც სამედიცინო განყოფილების შემოსავალს დათვლის. პროგრამის კოდს აქვს სახე:

```
USE Shekveta;
--      კურსორის შექმნა
DECLARE curs4 CURSOR
GLOBAL SCROLL KEYSET TYPE_WARNING
FOR
   SELECT  gadasaxdeli_l  FROM Personali P, Xelshekruleba X
   WHERE P.ganyofileba = N'სამედიცინო'  AND P.personaliID = X.personaliID;
OPEN curs4;
DECLARE  @@Striqonebis_raodenoba INT, @@Jami FLOAT, @@gadasaxdeli_l FLOAT;
SET @@Striqonebis_raodenoba = 1;
```

```
SET @@Jami   = 0;
WHILE @@Striqonebis_raodenoba <= @@CURSOR_ROWS
 BEGIN
  IF @@Striqonebis_raodenoba = 1
     FETCH ABSOLUTE 1 FROM curs4 INTO @@gadasaxdeli_l;
  ELSE
     FETCH curs4 INTO @@gadasaxdeli_l;
  SET @@Striqonebis_raodenoba = @@Striqonebis_raodenoba + 1;
  SET @@Jami   =  @@Jami + @@gadasaxdeli_l;
 END
SELECT  N'სულ შემოსავალია:  ', @@Jami;
--        კურსორის დახურვა და გათავისუფლება
CLOSE curs4;
DEALLOCATE curs4;
```

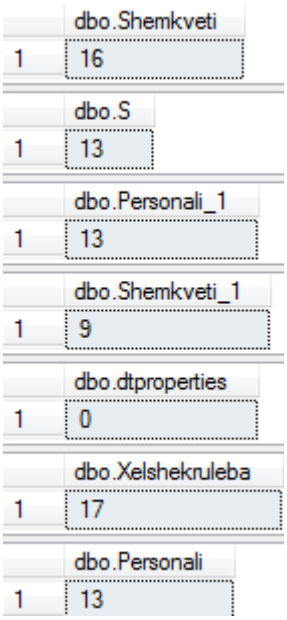
შედეგი:

	(No column name)	(No column name)
1	სულ შემოსავალია:	36000

ბ. შევადგინოთ კურსორი, რომელიც მიმდინარე მონაცემთა ბაზის თითოეული ცხრილის სტრიქონების რაოდენობას დათვლის და ეკრანზე გამოიტანს. პროგრამის კოდს აქვს სახე:

```
USE Shekveta;
--        კურსორის გამოცხადება
DECLARE Raodenoba CURSOR
FOR
  SELECT s_s.name + '.' + s_t.name FROM sys.tables AS s_t
  JOIN sys.schemas AS s_s ON s_s.schema_id = s_t.schema_id;
OPEN Raodenoba;
DECLARE @Cxrilis_saxeli sysname;
FETCH NEXT FROM Raodenoba INTO @Cxrilis_saxeli;
WHILE (@@FETCH_STATUS <> -1)
BEGIN
  IF (@@FETCH_STATUS <> -2)
  BEGIN
    SELECT @Cxrilis_saxeli = RTRIM(@Cxrilis_saxeli);
    EXEC ('SELECT ''' + @Cxrilis_saxeli + ''' = COUNT(*) FROM ' + @Cxrilis_saxeli );
  END;
  FETCH NEXT FROM Raodenoba INTO @Cxrilis_saxeli;
END;
CLOSE Raodenoba;
DEALLOCATE Raodenoba;
```

შედეგი:



მონაცემების შეცვლა

კურსორის საშუალებით მონაცემების შესაცვლელად UPDATE ბრძანება გამოიყენება. მისი სინტაქსია:

UPDATE *ცხრილის_სახელი* SET { *სვეტის_სახელი* = { DEFAULT | NULL | *გამოსახულება* } } [,...n]
WHERE CURRENT OF *კურსორის_სახელი*

განვიხილოთ არგუმეტების დანიშნულება.

– *სვეტის_სახელი* იმ სვეტის სახელია, რომელიც უნდა შეიცვალოს. ერთი ოპერაციით შეიძლება რამდენიმე სვეტის შეცვლა, მაგრამ ყველა შესაცვლელი სვეტი ერთ ცხრილს უნდა ეკუთვნოდეს. სვეტს მიენიჭება ნაგულისხმევი მნიშვნელობა, თუ მითითებულია DEFAULT საკვანძო სიტყვა. სვეტს მიენიჭება NULL მნიშვნელობა, თუ მითითებულია NULL საკვანძო სიტყვა. *გამოსახულება* შეიცავს სვეტისთვის მისანიჭებელ მნიშვნელობას.

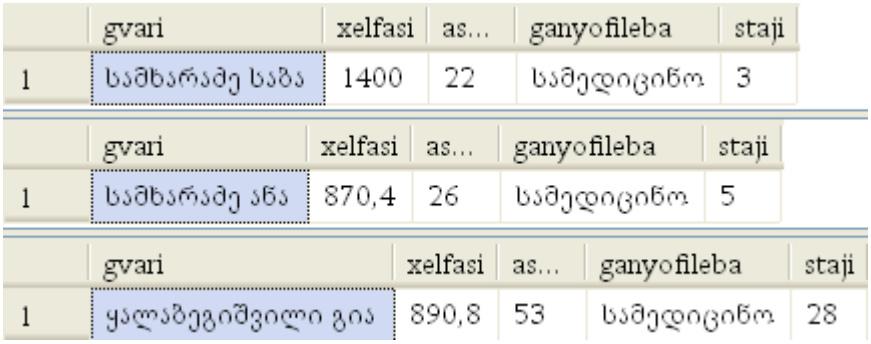
– *კურსორის_სახელი* იმ კურსორის სახელია, რომელშიც უნდა შესრულდეს ცვლილებები.

მაგალითი. შევადგინოთ კურსორი, რომელიც სამედიცინო განყოფილების თანამშრომლების ხელფასს 100 ლარით გაზრდის. პროგრამის კოდს აქვს სახე:

```
USE Shekveta_1;
DECLARE @n INT, @raod INT;
SET @n = 1;
DECLARE curs4 CURSOR
GLOBAL SCROLL DYNAMIC TYPE_WARNING
FOR
   SELECT  gvari, xelfasi, asaki, ganyofileba, staji
   FROM Personali  WHERE ganyofileba = N'სამედიცინო'
FOR UPDATE
OPEN curs4
--       მონაცემების შეცვლა
SELECT @raod = COUNT(*)
FROM Personali
WHERE ganyofileba = N'სამედიცინო'
```

```
WHILE @n <= @raod
BEGIN
   FETCH NEXT FROM curs4
   UPDATE Personali SET xelfasi = xelfasi + 100  WHERE CURRENT OF curs4
   SET @n = @n + 1
END
--       კურსორის დახურვა და გათავისუფლება
CLOSE curs4;
DEALLOCATE curs4;
```

შედეგი:



	gvari	xelfasi	as...	ganyofileba	staji
1	სამხარაძე საბა	1400	22	სამედიცინო	3

	gvari	xelfasi	as...	ganyofileba	staji
1	სამხარაძე ანა	870,4	26	სამედიცინო	5

	gvari	xelfasi	as...	ganyofileba	staji
1	ყალაბეგიშვილი გია	890,8	53	სამედიცინო	28

მონაცემების წაშლა

კურსორიდან მონაცემების წასაშლელად DELETE ბრძანება გამოიყენება. მისი სინტაქსია:

DELETE *ცხრილის_სახელი* WHERE
CURRENT OF *კურსორის_სახელი*
{ { [GLOBAL] *კურსორის_სახელი* } | @cursor_variable_name }

მაგალითი. შევადგინოთ კურსორი და მასში მოვათავსოთ 'სამედიცინო' განყოფილების შესაბამისი სტრიქონები. წავშალოთ კურსორის მეორე სტრიქონი. პროგრამის კოდს აქვს სახე:

```
USE Shekveta;
DECLARE curs4 CURSOR
GLOBAL SCROLL KEYSET TYPE_WARNING
FOR
   SELECT  *
   FROM Personali
   WHERE ganyofileba = N'სამედიცინო'
FOR UPDATE
OPEN curs4
--       სტრიქონის წაშლა
FETCH ABSOLUTE 2 FROM curs4
DELETE Personali WHERE CURRENT OF curs4
--       კურსორის დახურვა და გათავისუფლება
CLOSE curs4;
DEALLOCATE curs4;
```

კურსორის დახურვა

კურსორის დახურვა ათავისუფლებს მისთვის გამოყოფილ რესურსებს და შლის კურსორში მოთავსებულ სტრიქონებს. დახურვის დროს მოიხსნება კურსორის მუშაობის დროს დაყენებული ბლოკირებები. კურსორი, რომელიც დაიხურა, მაგრამ არ გათავისუფლდა, შეიძლება განმეორებით გაიხსნას. კურსორის დასახურად გამოიყენება CLOSE ბრძანება, რომლის სინტაქსია:

CLOSE { { [GLOBAL] *კურსორის_სახელი* } | @cursor_variable_name }

მაგალითი. დავხუროთ curs1 კურსორი:

```
CLOSE curs1
```

კურსორის გათავისუფლება

კურსორის გათავისუფლება იწვევს მის წაშლას მონაცემთა ბაზიდან. კურსორის გასათავისუფლებლად გამოიყენება DEALLOCATE ბრძანება. მისი სინტაქსია:

DEALLOCATE { { [GLOBAL] *კურსორის_სახელი* } | @cursor_variable_name }

მაგალითი. გავათავისუფლოთ curs1 კურსორი:

```
DEALLOCATE curs1
```

თავი 14. ტრანზაქციები და დაბლოკვები

შესავალი

ერთ მონაცემთა ბაზასთან დროის ნებისმიერ მომენტში ერთდროულად შეიძლება რამდენიმე მომხმარებელი მუშაობდეს. ამ დროს გვაქვს სხვადასხვა სიტუაციები. ყველაზე მარტივი სიტუაცია გვაქვს მაშინ, როცა რამდენიმე მომხმარებელი ერთდროულად ცდილობს ერთი და იმავე მონაცემების წაკითხვას, მაგალითად, ცხრილიდან ერთი და იმავე სტრიქონების წაკითხვას. სიტუაცია რთულდება მაშინ, როცა მომხმარებლების ერთი ნაწილი ცდილობს მონაცემების შეცვლას, მეორე კი - ამ მონაცემების წაკითხვას. ასეთ შემთხვევაში, მომხმარებლების მეორე ნაწილმა შეიძლება არასწორი მნიშვნელობები მიიღოს. ამ პრობლემების გადასაწყვეტად გამოიყენება ტრანზაქციები და დაბლოკვები.

ტრანზაქცია არის პროცესი, როცა სრულდება მასში შემავალი ყველა ბრძანება ან არც ერთი. ტრანზაქცია შეიძლება როგორც ერთი, ისე ასობით ბრძანებისგან შედგებოდეს. თუ ყველა ბრძანება წარმატებით შესრულდა, მაშინ მოხდება ტრანზაქციის ფიქსირება (Commit), ანუ ცვლილებების დაფიქსირება. თუ ტრანზაქციის რომელიმე ბრძანება ვერ შესრულდა, მაშინ დაიწყება ტრანზაქციის უკუქცევა (Rollback), ანუ გაუქმდება ყველა ცვლილება. სისტემა აღდგება იმ მდგომარეობაში, რომელიც მას ჰქონდა ტრანზაქციის დაწყებამდე. ინფორმაცია სისტემის საწყისი მდგომარეობის შესახებ ინახება ტრანზაქციების ჟურნალში.

ჩვეულებრივ, სერვერი ავტომატურად ბლოკავს იმ მონაცემებს, რომლებთანაც ტრანზაქცია მუშაობს. ამიტომ მცირე ზომის ტრანზაქციების გამოყენებისას სერვერის მუშაობის ეფექტურობა იზრდება დიდი ზომის ტრანზაქციების გამოყენებასთან შედარებით. დიდი ზომის ტრანზაქციების გამოყენებისას ხანგრძლივი დროით ხდება მონაცემების დაბლოკვა და, შესაბამისად, ადგილი აქვს ხანგრძლივ მოცდენებს. მაგალითად, თუ მომხმარებლები იყენებენ შენახულ პროცედურას, რომელიც მონაცემების შეცვლას ასრულებს როგორც ერთ დიდი ტრანზაქცია, მაშინ ადგილი ექნება ხანგრძლივ მოცდენებს. ასეთი შენახული პროცედურა ისე უნდა დაიწეროს, რომ მან მონაცემების შეცვლა შეასრულოს როგორც რამდენიმე, მცირე ზომის, სწრაფად შესრულებადმა და მცირე მონაცემების დამბლოკავმა ტრანზაქციამ.

დაბლოკვა (*lock*) არის მონაცემებზე დროებით დადებული შეზღუდვა მონაცემების დამუშავების ზოგიერთ ოპერაციის შესრულების დროს. შეიძლება დაიბლოკოს როგორც ცხრილის ერთი სტრიქონი, ისე მთელი მონაცემთა ბაზა. არსებობს სხვადასხვა სახის დაბლოკვა, რომელსაც *დაბლოკვების მენეჯერი* (*lock manager*) მართავს. ტრანზაქციები და დაბლოკვები ერთმანეთთან მჭიდროდაა დაკავშირებული. სერვერს აქვს დაბლოკვის მართვის ეფექტური მექანიზმები, მაგრამ საჭიროებისამებრ ჩვენ შეგვიძლია დაბლოკვის საკუთარი ალგორითმის რეალიზება.

ტრანზაქციების მართვა

ტრანზაქცია განისაზღვრება შეერთების დონეზე. შეერთების დახურვის დროს ავტომატურად იხურება ტრანზაქციაც. არსებობს სამი სახის ტრანზაქცია: აშკარა, ავტომატური და არააშკარა (ნაგულისხმევი).

ტრანზაქცია არის *ლოკალური*, თუ ის სრულდება ერთი მონაცემთა ბაზის შიგნით. *განაწილებული* არის ტრანზაქცია, რომელიც მიმართავს რამდენიმე მონაცემთა ბაზას, რომლებიც შეიძლება სხვადასხვა სერვერზე იყოს განთავსებული.

აშკარა ტრანზაქციები

აშკარა ტრანზაქციების (*explicit transaction*) გამოყენების შემთხვევაში აშკარად უნდა მიუთითოთ ტრანზაქციის დაწყება და დამთავრება. ამისთვის გამოიყენება შემდეგი ბრძანებები: BEGIN TRANSACTION, COMMIT და ROLLBACK.

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

BEGIN TRAN[SACTION]

[*ტრანზაქციის_სახელი* | @tran_name_variable

განვიხილოთ არგუმენტების დანიშნულება.

– *ტრანზაქციის_სახელი* ჩვეულებრივ, გამოიყენება მხოლოდ ჩადგმულ ტრანზაქციებთან სამუშაოდ ყველაზე დაბალი დონის ტრანზაქციის სახელდებისთვის. სახელის სიგრძე არ უნდა აღემატებოდეს 32 სიმბოლოს და უნდა აკმაყოფილებდეს ობიექტის სახელდების წესებს.

– @tran_name_variable ლოკალური ცვლადია, რომელიც ტრანზაქციის სახელს შეიცავს. მისი ტიპი უნდა იყოს CHAR, VARCHAR, NCHAR ან NVARCHAR. ჩადგმული ტრანზაქციების სახელდებისას უნდა გავითვალისწინოთ ის, რომ სერვერზე რეგისტრირდება მხოლოდ პირველი (ზედა დონის) ტრანზაქციის სახელი. დანარჩენი ტრანზაქციების სახელები იგნორირდება. ჩვეულებრივ შემთხვევაში ტრანზაქციის სახელდება საჭირო არ არის.

COMMIT TRANSACTION ან COMMIT WORK ბრძანებები განსაზღვრავენ ტრანზაქციის დასასრულს. თუ ტრანზაქციის შესრულების დროს ადგილი არ ჰქონდა შეცდომებს, მაშინ შესრულებული ცვლილებები დაფიქსირდება (roll forward). ამის შემდეგ, ტრანზაქციების ჟურნალში მოინიშნება, რომ ცვლილებები დაფიქსირებულია და ტრანზაქცია დამთავრებული. COMMIT ბრძანებების სინტაქსია:

COMMIT [WORK] |

COMMIT [TRAN [SACTION] [*ტრანზაქციის_სახელი* | @tran_name_variable]]

ROLLBACK TRANSACTION ან ROLLBACK WORK ბრძანებების შესრულება იწვევს ტრანზაქციის შეწყვეტას და უკუქცევას (roll back). უკუქცევის დროს, შესრულებული ცვლილებები უქმდება და აღდგება სისტემის პირვანდელი მდგომარეობა. ტრანზაქციების ჟურნალში მოინიშნება, რომ ტრანზაქცია იყო გაუქმებული. ROLLBACK ბრძანებების სინტაქსია:

ROLLBACK [WORK] |

ROLLBACK [TRAN [SACTION]

[*ტრანზაქციის_სახელი* | @tran_name_variable

როცა მონაცემებს მივმართავთ OLE DB და ADO მექანიზმებით, მაშინ შეგვიძლია გამოვიყენოთ ტრანზაქციის აშკარა განსაზღვრა. თუ მონაცემებს მივმართავთ ODBC მექანიზმის საშუალებით, მაშინ უნდა გამოვიყენოთ ტრანზაქციის ავტომატური და არააშკარა განსაზღვრა.

მაგალითები.

ა. მაგალითში ხდება ტრანზაქციის სახელდების დემონსტრირება:

```
USE Shekveta;
DECLARE @TranSaxeli NVARCHAR(20);
SELECT @TranSaxeli = N'ChemiTranzaqcia';
BEGIN TRANSACTION @TranSaxeli;
   DELETE FROM Shemkveti_2 WHERE iuridiuli_fizikuri = N'ფიზიკური';
IF @@ERROR = 0
  COMMIT TRANSACTION @TranSaxeli;
ELSE
```

```
   ROLLBACK TRANSACTION @TranSaxeli;
SELECT * FROM Shemkveti_2;
```

აქ @@ERROR ფუნქცია გასცემს შეცდომის კოდს. თუ შეცდომის კოდი ნულის ტოლია, ეს იმას ნიშნავს, რომ ტრანზაქციის შესრულების დროს შეცდომას ადგილი არ ჰქონდა. ამიტომ ტრანზაქცია დაფიქსირდება. თუ @@ERROR ფუნქცია ნულისაგან განსხვავებულ რიცხვს გასცემს, ეს იმას ნიშნავს, რომ ტრანზაქციის შესრულების დროს გვქონდა შეცდომ. ამიტომ დაიწყება ტრანზაქციის უკუქცევა ანუ შესრულებული ცვლილებების გაუქმება.

შედეგი:

	shemkvetiID	personaliID	iuridiuli_fizikuri	gvari	qalaqi	raioni	misa
1	1	1	იურიდიული	ქევხიშვილი ალექსანდრე	თბილისი	საბურთალო	ვაჟა
2	3	7	იურიდიული	სეხნიაშვილი გია	რუსთავი	NULL	აღმ
3	4	8	იურიდიული	მამაიაშვილი ჯემალი	თელავი	NULL	ჭავჭ
4	6	2	იურიდიული	წოწიაშვილი ზურა	ბათუმი	NULL	რუს
5	8	4	იურიდიული	შეწგელია ლუკა	ზუგდიდი	NULL	გამს
6	10	6	იურიდიული	გიორგაძე გივი	ზუგდიდი	NULL	კოს
7	11	5	იურიდიული	ჭიკაძე გელა	გორი	NULL	სტა
8	13	1	იურიდიული	ლომიძე რუსუდანი	თბილისი	საბურთალო	დო
9	14	2	იურიდიული	არაყიშვილი ავთო	ქუთაისი	NULL	ა. ყა

ბ. მოყვანილ მაგალითში ერთ ტრანზაქციაში ორი INSERT ბრძანება სრულდება:

```
USE Shekveta;
CREATE TABLE #Droebiti_Cxrili_1
(
Sveti_1 INT PRIMARY KEY IDENTITY(1,1),
Sveti_2 NVARCHAR(10) NOT NULL
)
--
BEGIN TRANSACTION Chemi_Trans_1;
   INSERT INTO #Droebiti_Cxrili_1 VALUES (N'რომანი');
   INSERT INTO #Droebiti_Cxrili_1 VALUES (N'ლიკა');
IF @@ERROR = 0
   COMMIT TRANSACTION Chemi_Trans_1
ELSE
   ROLLBACK TRANSACTION Chemi_Trans_1;
--
SELECT * FROM #Droebiti_Cxrili_1;
```

შედეგი:

	Sveti_1	Sveti_2
1	1	რომანი
2	2	ლიკა

ავტომატური ტრანზაქციები

ჩვეულებრივ, სერვერი მუშაობს *ტრანზაქციის ავტომატურად დაწყების რეჟიმში*

(*autocommit transaction*). ამ რეჟიმში თითოეული ბრძანება განიხილება როგორც ცალკეული ტრანზაქცია. თუ ბრძანება წარმატებით შესრულდა, მაშინ მის მიერ შესრულებული ცვლილებები ფიქსირდება. თუ ბრძანების შესრულების დროს გვაქვს შეცდომა, მაშინ შესრულებული ცვლილებები უქმდება და სისტემა საწყის მდგომარეობას უბრუნდება.

თუ საჭიროა ტრანზაქციის შექმნა, რომელიც რამდენიმე ბრძანებას შეიცავს, მაშინ ტრანზაქცია აშკარად უნდა განვსაზღვროთ.

ჩვეულებრივ, სერვერი მუშაობს ტრანზაქციის განსაზღვრის ავტომატურ ან არააშკარა რეჟიმში. სერვერი არ შეიძლება იმყოფებოდეს ტრანზაქციის მხოლოდ აშკარად განსაზღვრის რეჟიმში.

ტრანზაქციის ავტომატური განსაზღვრის რეჟიმის დასაყენებლად გამოიყენება ბრძანება:

```
SET IMPLICIT_TRANSACTION OFF;
```

არააშკარა ტრანზაქციები

ტრანზაქციის არააშკარა (*ნაგულისხმევი*) დაწყების (*implicit transaction*) რეჟიმში მუშაობის დროს სერვერი ავტომატურად იწყებს ახალ ტრანზაქციას, როგორც კი დამთავრდება წინა. მომხმარებელი არაფერს არ უთითებს ტრანზაქციის დასაწყებად. ტრანზაქცია გრძელდება მანამ, სანამ მომხმარებელი აშკარად არ მიუთითებს ტრანზაქციის უკუქცევის ან დაფიქსირების ბრძანებას. ამის შემდეგ სერვერი ავტომატურად იწყებს ახალ ტრანზაქციას.

მას შემდეგ, რაც შეერთებისთვის დაყენებულია ტრანზაქციის არააშკარა დაწყების რეჟიმი, სერვერი ავტომატურად ამთავრებს მიმდინარე ტრანზაქციას და იწყებს ახალს, თუ გვხვდება შემდეგი ბრძანებებიდან ერთ-ერთი:

- ALTER TABLE - ცხრილის სტრუქტურის შეცვლა;
- CREATE - მონაცემთა ბაზის ობიექტის შექმნა;
- DELETE - ცხრილიდან სტრიქონების წაშლა;
- DROP - მონაცემთა ბაზის ობიექტის წაშლა;
- FETCH - კურსორიდან მითითებული სვეტის მნიშვნელობის მიღება;
- GRANT - მონაცემთა ბაზის ობიექტთან მიმართვის ნების დართვა;
- INSERT - ცხრილში სტრიქონების დამატება;
- OPEN - კურსორის გახსნა;
- REVOKE - მონაცემთა ბაზის ობიექტებთან მიმართვის არააშკარა გადახრა (отклонение);
- SELECT - ცხრილიდან მონაცემების ამორჩევა;
- TRUNCATE TABLE - ცხრილის ჩამოჭრა;
- UPDATE - ცხრილში მონაცემების შეცვლა.

ტრანზაქციის არააშკარა განსაზღვრის რეჟიმის დასაყენებლად გამოიყენება ბრძანება:

```
SET IMPLICIT_TRANSACTION ON
```

განაწილებული ტრანზაქციები

როცა საჭიროა სხვადასხვა მონაცემთა ბაზაში მოთავსებულ რესურსებთან მიმართვა, აუცილებელია *განაწილებული ტრანზაქციის* (*distributed transaction*) გამოყენება. განაწილებული ტრანზაქცია წარმოადგენს რამდენიმე ცალკეულ ტრანზაქციას, რომლებიც ლოკალურად სრულდება ცალკეულ მონაცემთა ბაზაში.

სერვერი განაწილებული ტრანზაქციის შესრულებას იწყებს BEGIN DISTRIBUTED

TRANSACTION ბრძანებით. მისი სინტაქსია:
BEGIN DISTRIBUTED TRAN [SACTION]
[*ტრანზაქციის_სახელი* | @tran_name_variable]

ჩადგმული ტრანზაქციები

ჩადგმული ტრანზაქციები (*nested transaction*) ეწოდება ტრანზაქციებს, რომელთა შესრულება ინიცირდება (იწყება) აქტიური ტრანზაქციის კოდიდან.

ჩადგმული ტრანზაქციების გამოყენება შესაძლებელია მხოლოდ აშკარა ტრანზაქციებში. ავტომატური ან არააშკარა ტრანზაქციების შესრულებისას, ახალი ტრანზაქციის დაწყებამდე წინა ტრანზაქცია დამთავრებული უნდა იყოს. ამიტომ ამ ტრანზაქციებიდან შეუძლებელია ჩადგმული ტრანზაქციების ინიცირება. ჩადგმული ტრანზაქციების ძირითადი დანიშნულებაა იმ ტრანზაქციების უზრუნველყოფა, რომლებიც სრულდება ჩადგმული პროცედურების მიერ. ჩვენ შეგვიძლია მივმართოთ შენახულ პროცედურას როგორც უკვე დაწყებული ტრანზაქციიდან, ისე უშუალოდ (არა ტრანზაქციის კოდიდან).

მაგალითი. ჩადგმული ტრანზაქციების მუშაობის დემონსტრირების მიზნით შევქმნათ პროცედურა, რომელიც დროებით ცხრილს ორ სტრიქონს დაუმატებს. ეს პროცედურა ტრანზაქციის შიგნით გამოვიძახოთ და მისი მუშაობის შედეგი გავაუქმოთ. შემდეგ ისევ გამოვიძახოთ იგივე პროცედურა და მას სხვა პარამეტრები გადავცეთ. მოთხოვნას აქვს სახე:

```
SET QUOTED_IDENTIFIER OFF;
SET NOCOUNT OFF;
USE Shekveta_1;
CREATE TABLE #Droebiti_Cxrili_1
(
Sveti_1 INT PRIMARY KEY,
Sveti_2 NVARCHAR(4) NOT NULL
)
GO
CREATE PROCEDURE Chemi_Proc @Sveti_1 INT, @Sveti_2 NVARCHAR(4) AS
BEGIN TRANSACTION Chemi_Trans_1;
   INSERT INTO #Droebiti_Cxrili_1 VALUES (@Sveti_1, @Sveti_2);
   INSERT INTO #Droebiti_Cxrili_1 VALUES (@Sveti_1 + 1, @Sveti_2);
COMMIT TRANSACTION Chemi_Trans_1;
--      ტრანზაქციის დაწყება და TransProc პროცედურის შესრულება
BEGIN TRANSACTION Chemi_Trans_2;
EXEC Chemi_Proc 1, N'ანა';
--      გარე ტრანზაქციის უკუქცევა
ROLLBACK TRANSACTION Chemi_Trans_2;
EXEC Chemi_Proc 3, N'საბა';
SELECT * FROM #Droebiti_Cxrili_1;
```

შედეგი:

	Sveti_1	Sveti_2
1	3	საბა
2	4	საბა

ჩადგმული ტრანზაქციის შესაქმნელად უნდა დავიწყოთ ახალი ტრანზაქცია წინა ტრანზაქციის დახურვის გარეშე. ზედა დონის ტრანზაქციის დამთავრება გადაიდება ჩადგმული ტრანზაქციების დამთავრებამდე. თუ ყველაზე დაბალი დონის ტრანზაქცია წარუმატებლად დამთავრდა და უკუქცეულია, მაშინ ზედა დონის ყველა ტრანზაქცია უკუიქცევა პირველი დონის ტრანზაქციის ჩათვლით. გარდა ამისა, თუ ქვედა დონის რამდენიმე ტრანზაქცია წარმატებით დამთავრდა (მაგრამ არ დაფიქსირდა), მაგრამ საშუალო დონეზე ტრანზაქცია წარუმატებლად დამთავრდა, მაშინ უკუიქცევა ყველა დონის ტრანზაქცია იმ ტრანზაქციების ჩათვლით, რომლებიც წარმატებით დამთავრდა. როცა ყველა დონის ტრანზაქცია წარმატებით დამრთავდება, მაშინ ყველა ცვლილება დაფიქსირდება იმ შემთხვევაში, როცა წარმატებით დამთავრდება პირველი დონის ტრანზაქცია.

თითოეული COMMIT TRANSACTION ან COMMIT WORK ბრძანება მუშაობს მხოლოდ უკანასკნელად დაწყებულ ტრანზაქციასთან. ჩადგმული ტრანზაქციის დამთავრებისას COMMIT ბრძანება გამოიყენება ყველაზე „ღრმად" ჩადგმული ტრანზაქციის მიმართ. იმ შემთხვევაშიც კი, თუ COMMIT TRANSACTION ბრძანებაში *ტრანზაქციის_სახელი* მიმართავს უფრო მაღალი დონის ტრანზაქციას, ჯერ მაინც დამთავრდება უკანასკნელად დაწყებული ტრანზაქცია.

თუ ROLLBACK TRANSACTION ბრძანება გამოიყენება ჩადგმულობის ნებისმიერ დონეზე ტრანზაქციის სახელის მითითების გარეშე ან გამოიყენება ROLLBACK WORK ბრძანება, მაშინ უკუიქცევა ყველა ჩადგმული ტრანზაქცია, ყველაზე ზედა (პირველი) დონის ტრანზაქციის ჩათვლით. ROLLBACK TRANSACTION ბრძანებაში დასაშვებია მხოლოდ ზედა დონის ტრანზაქციის სახელის მითითება. ნებისმიერი ჩადგმული ტრანზაქციის სახელი იგნორირდება და მისი მითითება გამოიწვევს შეცდომას. ამრიგად, ნებისმიერი დონის ტრანზაქციის უკუქცევისას ნებისმიერ შემთხვევაში ხდება ყველა ტრანზაქციის უკუქცევა.

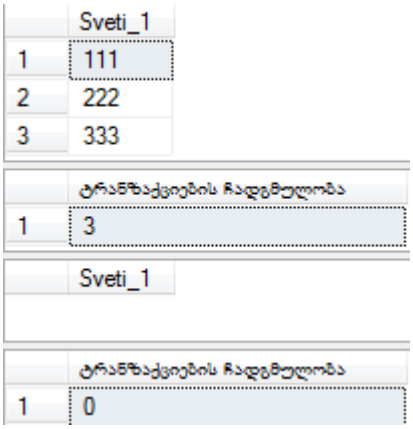
სერვერს აქვს @@TRANCOUNT ფუნქცია, რომელიც განკუთვნილია აქტიური ტრანზაქციების რაოდენობის განსაზღვრისათვის, რომლებიც დაიწყო აქტიურ შეერთებაში. ყოველი BEGIN TRANSACTION ბრძანების შესრულებისას ამ ფუნქციის მიერ დაბრუნებული მნიშვნელობა 1-ით იზრდება. ყოველი ROLLBACK TRANSACTION ან ROLLBACK WORK ბრძანების შესრულება იწვევს ერთი ტრანზაქციის უკუქცევას და 1-ით ამცირებს @@TRANCOUNT ფუნქციის მიერ გაცემულ მნიშვნელობას. ჩადგმული ტრანზაქციების შესრულებისას ROLLBACK TRANSACTION ბრძანების შესრულება ტრანზაქციის სახელის მითითების გარეშე ან ROLLBACK WORK ბრძანების შესრულება გამოიწვევს ყველა ტრანზაქციის უკუქცევას და @@TRANCOUNT მნიშვნელობის განულებას.

მაგალითი. მოყვანილ კოდში იქმნება სამი ჩადგმული ტრანზაქცია, რომელთაგან თითოეული #Droebiti_Cxrili დროებით ცხრილში ახდენს თითო სტრიქონის ჩასმას. ყველა სტრიქონის ჩასმის შემდეგ ხდება ცხრილის შემცველობის ეკრანზე გამოტანა, ტრანზაქციის უკუქცევა, ცხრილის შემცველობის ისევ გამოტანა ეკრანზე და @@TRANCOUNT ფუნქციის მიერ გაცემული მნიშვნელობის ნახვა. მოთხოვნას აქვს სახე:

```
CREATE TABLE #Droebiti_Cxrili (Sveti_1 INT);
BEGIN TRAN;
INSERT INTO #Droebiti_Cxrili VALUES (111);
BEGIN TRAN;
```

```
INSERT INTO #Droebiti_Cxrili VALUES (222);
BEGIN TRAN;
INSERT INTO #Droebiti_Cxrili VALUES (333);
SELECT * FROM #Droebiti_Cxrili;
SELECT N'ტრანზაქციების ჩადგმულობა' = @@TRANCOUNT;
ROLLBACK TRAN;
SELECT * FROM #Droebiti_Cxrili;
SELECT N'ტრანზაქციების ჩადგმულობა' = @@TRANCOUNT;
```

შედეგი:



	Sveti_1
1	111
2	222
3	333

	ტრანზაქციების ჩადგმულობა
1	3

	Sveti_1

	ტრანზაქციების ჩადგმულობა
1	0

ტრანზაქციებში აკრძალული Transact SQL-ის ბრძანებები

ტრანზაქციებში აკრძალულია იმ ოპერაციების შესრულება, რომლებისთვისაც არაა რეალიზებული შესრულებული მოქმედებების უკუქცევა. ასეთი მოქმედებების უკუქცევისათვის საჭიროა მონაცემთა ბაზის დაარქივება ტრანზაქციაში აკრძალული ოპერაციების გამოყენების წინ, რომ გვქონდეს საწყის მდგომარეობაში მონაცემთა ბაზის აღდგენის შესაძლებლობა.

ტრანზაქციის შიგნით აკრძალულია შემდეგი ბრძანებების გამოყენება:

– ALTER DATABASE - მონაცემთა ბაზის კონფიგურაციის შეცვლა;
– BACKUP LOG - ტრანზაქციების ჟურნალის სარეზერვო ასლის შექმნა;
– CREATE DATABASE - მონაცემთა ბაზის შექმნა;
– DROP DATABASE - მონაცემთა ბაზის წაშლა;
– LOAD DATABASE - მონაცემთა ბაზის სარეზერვო ასლის ჩატვირთვა;
– LOAD TRANSACTION - ტრანზაქციების ჟურნალის სარეზერვო ასლის ჩატვირთვა;
– RECONFIGURE - sp_configure შენახული პროცედურის მიერ შესრულებული ცვლილებების გამოყენება;
– RESTORE DATABASE - სარეზერვო ასლიდან მონაცემთა ბაზის აღდგენა;
– RESTORE LOG - სარეზერვო ასლიდან ტრანზაქციების ჟურნალის აღდგენა;
– UPDATE STATISTICS - სტატისტიკის გაახლება.

დაბლოკვის მართვა

დაბლოკვების დაყენებისა და მოხსნის, კონფლიქტების გადაწყვეტის სამუშაოს *დაბლოკვების მენეჯერი* (lock manager) ასრულებს, თუმცა, შეგვიძლია მოთხოვნაში აშკარად

მივუთითოთ ჩვენთვის საჭირო დაბლოკვის ტიპი.

განვიხილოთ დაბლოკვის ალგორითმი. მანამ, სანამ ტრანზაქცია შეძლებს მონაცემების შეცვლას, ის დაბლოკვების მენეჯერისაგან ითხოვს შესაბამისი დაბლოკვის დაწყებას. თუ მოთხოვნილი მონაცემები მოცემულ მომენტში გამოიყენება სხვა პროცესის (შესრულების სტადიაში მყოფი პროგრამა) მიერ, მაშინ დაბლოკვების მენეჯერი ან უარყოფს დაბლოკვის მოთხოვნას, ან მას რიგში აყენებს და მაშინ შეასრულებს, როცა მიმდინარე პროცესი მუშაობას დაამთავრებს და მონაცემებს გაათავისუფლებს. თუ მონაცემები (რესურსი) მოთხოვნის მომენტში თავისუფალია, მაშინ დაბლოკვების მენეჯერი აკმაყოფილებს მოთხოვნას და ბლოკავს მოთხოვნილ მონაცემებს. დაბლოკვის შემდეგ ტრანზაქციას შეუძლია მონაცემების დამუშავება.

რესურსის დაბლოკვის მოხსნის ლოდინის დრო შეგვიძლია ვარეგულიროთ. ამ მიზნით გამოიყენება ბრძანება:

SET LOCK_TIMEOUT *პერიოდი*

პერიოდი მიუთითებს მილიწამებს, რომლის განმავლობაშიც ტრანზაქცია დაელოდება რესურსის გათავისუფლებას. ამ დროის გავლის შემდეგ გაიცემა შეტყობინება შეცდომის შესახებ. მისი ნაგულისხმევი მნიშვნელობაა -1, რაც შეესაბამება უსასრულო ლოდინს. ამ არგუმენტის მიმდინარე მნიშვნელობას გასცემს @@LOCK_TIMEOUT ფუნქცია.

SET ბრძანება მოქმედებს მხოლოდ მიმდინარე შეერთების განმავლობაში. განმეორებითი შეერთების შემთხვევაში ხელახლა უნდა განისაზღვროს ლოდინის მაქსიმალური პერიოდი.

დაბლოკვის სისტემა უზრუნველყოფს მონაცემების დამუშავების მრავალმომხმარებლურ გარემოს. ის გამოირთვება მხოლოდ მაშინ, როცა სერვერი მუშაობს ერთი მომხმარებლის რეჟიმში „მხოლოდ წაკითხვისათვის", რადგან ამ შემთხვევაში ადგილი არ აქვს მიმართვის კონფლიქტს.

დაბლოკვა იკავებს ოპერატიული მეხსიერების უბანს და პროცესორის დროს. დაბლოკვების მაქსიმალური რაოდენობის განსაზღვრისათვის გამოიყენება sp_configure შენახული პროცედურა. მისი სინტაქსია:

sp_configure ['locks', n]

n დაბლოკვების მაქსიმალური რაოდენობაა და იცვლება საზღვრებში 5000÷2 147 843 647. მისი ავტომატური (ნაგულისხმევი) მნიშვნელობაა - 0, რაც ნიშნავს სერვერის ავტომატურ კონფიგურირებას. სერვერმა შეიძლება ავტომატურად გამოყოს ან გაათავისუფლოს ოპერატიული მეხსიერება, რომელიც გამოყოფილია არსებული დაბლოკვების შესახებ ინფორმაციის შესანახად.

დაბლოკვების მონიტორინგისთვის გამოიყენება sp_lock შენახული პროცედურა, რომლის საშუალებით შეგვიძლია მივიღოთ ინფორმაცია კონკრეტული პროცესის მიერ დაყენებული დაბლოკვების შესახებ. ამ პროცედურის სინტაქსია:

sp_lock [[@spid1 =] '*პროცესის_იდენტიფიკატორი_1*'

[, [@spid2 =] '*პროცესის_იდენტიფიკატორი_2*']

'*პროცესის_იდენტიფიკატორი_1*' არგუმენტი შეიცავს პროცესის საიდენტიფიკაციო ნომერს, რომლის შესახებაც გვინდა ინფორმაციის მიღება. '*პროცესის_იდენტიფიკატორი_2*' არგუმენტი დამატებითია და გამოიყენება ერთდროულად ორი პროცესის შესახებ ინფორმაციის მისაღებად. თუ არგუმენტები არ არის მითითებული, მაშინ გაიცემა ინფორმაცია ყველა პროცესის მიერ დაყენებული დაბლოკვის შესახებ.

მაგალითი. დავუშვათ, გვაინტერესებს ინფორმაცია 51-ე და 55-ე პროცესების შესახებ. მოთხოვნას აქვს სახე:

EXEC sp_lock '51', '55';

შედეგი:

	spid	dbid	ObjId	IndId	Type	Resource	Mode	Status
1	55	4	0	0	DB		S	GRANT
2	51	1	85575343	0	TAB		IS	GRANT

პროცესის შესაწყვეტად გამოიყენება KILL ბრძანება. მისი სინტაქსია:

KILL *პროცესის_იდენტიფიკატორი*

სერვერზე აქტიური პროცესების სია ინახება master მონაცემთა ბაზის sysprocesses სისტემურ წარმოდგენაში. ამ წარმოდგენაში ინახება დაწვრილებითი ინფორმაცია პროცესების შესახებ: უკანასკნელად შესრულებული ბრძანება, ტრანზაქციებისა და დაყენებული დაბლოკვების რაოდენობა, ლოდინის დრო, სტატუსი და ა.შ.

საერთო ინფორმაციის მისაღებად სისტემური პროცესის შესახებ შეგვიძლია გამოვიყენოთ sp_who შენახული პროცედურა.

მაგალითი. დავუშვათ, გვაინტერესებს ინფორმაცია 51-ე და 55-ე პროცესების შესახებ. მოთხოვნას აქვს სახე:

```
EXEC sp_who '51';
EXEC sp_who '55';
```

შედეგი:

	spid	ecid	status	loginame	hostname	blk	dbname	cmd
1	51	0	runnable	sa	ROMA	0	master	SELECT

	spid	ecid	status	loginame	hostname	blk	dbname	cmd
1	55	0	sleeping	NT AUTHORITY\SYSTEM	ROMA	0	msdb	AWAITING COMM

თავი 15. ტრიგერები

შესავალი

ტრიგერი (*trigger*) არის შენახული პროცედურის სპეციალური ტიპი, რომელიც სერვერის მიერ ავტომატურად გაიშვება ცხრილში მონაცემების შეცვლის ოპერაციების შესრულების დროს. ტრიგერი სხვადასხვა დანიშნულებით გამოიყენება. მათი საშუალებით შესაძლებელია ბმულ ცხრილებში კასკადური ცვლილებების შესრულება, მონაცემების შემოწმების ალგორითმების რეალიზება და ა.შ.

თითოეული ტრიგერი დაკავშირებულია კონკრეტულ ცხრილთან. მაგალითად, ცხრილში მონაცემების შეცვლის წინ სერვერი ავტომატურად გაუშვებს ტრიგერს. თუ მისი შესრულება წარმატებით დამთავრდა, მაშინ შესრულდება ცხრილში მონაცემების ცვლილება. ტრიგერის მიერ შესრულებული მოქმედებები განიხილება როგორც ერთი ტრანზაქცია. ტრიგერების გამოყენება შეიძლება აგრეთვე ჩვენი შეხედულებისამებრ.

ტრიგერი არ უნდა გამოვიყენოთ მარტივი შემოწმებების შესასრულებლად (რისთვისაც შეიძლება CHECK მთლიანობის შეზღუდვის გამოყენება). ტრიგერი არ უნდა გამოვიყენოთ იმ მოქმედებების შესასრულებლად, რომლებიც შეიძლება შენახული პროცედურებით ან Transact-SQL-ის პაკეტით შესრულდეს. ტრიგერების გამოყენება არ არის სასურველი აგრეთვე, იმიტომ, რომ ისინი ბლოკავენ რესურსს მუშაობის დამთავრებამდე და სხვა მომხმარებლებს არ ეძლევა ამ რესურსთან მიმართვის შესაძლებლობა.

ტრიგერები განსხვავდება იმ ბრძანებების ტიპის მიხედვით, რომლებზეც ისინი რეაგირებენ. არსებობს ტრიგერის სამი ტიპი:

- INSERT TRIGGER. ამ ტიპის ტრიგერი გაიშვება INSERT ბრძანებით მონაცემების ჩასმის მცდელობის დროს.
- UPDATE TRIGGER. ამ ტიპის ტრიგერი გაიშვება UPDATE ბრძანებით მონაცემების შეცვლის მცდელობის დროს.
- DELETE TRIGGER. ამ ტიპის ტრიგერი გაიშვება DELETE ბრძანებით მონაცემების წაშლის მცდელობის დროს.

სამივე ტიპის ტრიგერი წარმოადგენს DML-ტრიგერს (Data Manipulation Language, მონაცემების დამუშავების ენა). არსებობს, აგრეთვე DDL-ტრიგერიც (Data Definition Language, მონაცემების განსაზღვრის ენა), რომელიც გაიშვება მონაცემთა ბაზის ობიექტების შექმნის დროს.

ტრიგერი არ გაიშვება ტექსტური ბლოკების დამუშავების ბრძანებების შესრულებისას, როგორიცაა WRITETXT, READTXT და UPDATETXT.

ტრიგერის ქცევას ორი პარამეტრი განსაზღვრავს:

1. AFTER. ტრიგერი გამოიძახება მისი გამომწვევი ბრძანებების წარმატებით შესრულების შემდეგ. თუ რაიმე მიზეზის გამო ბრძანებების შესრულება წარუმატებლად დამთავრდა, მაშინ ტრიგერი არ გამოიძახება. AFTER-ტრიგერები განისაზღვრება მხოლოდ ცხრილებისთვის და არ განისაზღვრება წარმოდგენებისთვის. თითოეული INSERT, UPDATE და DELETE ბრძანებისათვის შეგვიძლია განვსაზღვროთ რამდენიმე AFTER-ტრიგერი. ასეთ შემთხვევაში sp_settriggerorder შენახული პროცედურის საშუალებით შეგვიძლია განვსაზღვროთ ტრიგერების შესრულების მიმდევრობა. ყველა კასკადური მოქმედება და შეზღუდვების შემოწმება ტრიგერის ამუშავებამდე უნდა დამთავრდეს.
2. INSTEAD OF. ტრიგერი გამოიძახება ბრძანების შესრულების ნაცვლად. INSTEAD OF-ტრიგერები შეგვიძლია განვსაზღვროთ როგორც ცხრილებისთვის, ისე წარმოდგენებისთვის. თითოეული INSERT, UPDATE და DELETE ბრძანებისათვის შეგვიძლია განვსაზღვროთ

მხოლოდ თითო INSTEAD OF-ტრიგერი.

ავტომატურად ყველა ტრიგერი AFTER-ტრიგერია. ერთი ცხრილისთვის შეგვიძლია შევქმნათ რამდენიმე ერთტიპური ტრიგერი.

ტრიგერის შექმნა

ტრიგერის შექმნის წინ საჭიროა გულდასმით გავიაზროთ ბრძანებების შესრულების მიმდევრობა ტრიგერის კოდში. უნდა გვახსოვდეს, რომ ტრიგერები დიდი ხნით ბლოკავენ საჭირო რესურსებს.

ტრიგერების შექმნა არ შეიძლება დროებითი ან სისტემური ცხრილებისათვის. მეორე მხრივ, ტრიგერს შეუძლია მიმართოს დროებით ცხრილს. თუ ტრიგერს სჭირდება სისტემურ ცხრილთან მიმართვა, მაშინ უნდა გამოვიყენოთ წარმოდგენები.

პაკეტში CREATE TRIGGER ბრძანება უნდა იყოს პირველი და უნდა გამოიყენებოდეს მხოლოდ ერთი ცხრილის მიმართ. ტრიგერები შეიძლება შეიქმნას მხოლოდ მიმდინარე მონაცემთა ბაზაში, მაგრამ ტრიგერის შიგნით დასაშვებია სხვა მონაცემთა ბაზებთან მიმართვა, მათ შორის დაშორებულ სერვერზე არსებულ მონაცემთა ბაზებთან.

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] }
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]
}
}

განვიხილოთ არგუმენტების დანიშნულება.

– *ტრიგერის_სახელი* უნდა იყოს უნიკალური მონაცემთა ბაზის ფარგლებში.

– *ცხრილის_სახელი* | *წარმოდგენის_სახელი*. იმ ცხრილის ან წარმოდგენის სახელია, რომელსაც ტრიგერი უკავშირდება.

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

– INSTEAD OF. თუ ეს არგუმენტი მითითებულია, მაშინ ტრიგერი შესრულდება მისი გამომძახებელი მოთხოვნის ნაცვლად.

– [DELETE] [,] [INSERT] [,] [UPDATE]. ეს კონსტრუქცია განსაზღვრავს, თუ რომელ ბრძანებაზე მოახდენს ტრიგერი რეაგირებას. ტრიგერის შექმნისას უნდა მივუთითოთ ერთ-ერთი მათგანი. დასაშვებია ორი ან სამი ბრძანების მითითებაც.

– AS sql_*ბრძანება* [,...n]. ეს არგუმენტი შეიცავს Transact_SQL-ის ბრძანებებს, რომლებიც შესრულდება ტრიგერის გაშვებისას.

– FOR [INSERT] [,] [UPDATE]. განსაზღვრავს ბრძანებას, რომლის შესრულების დროსაც ტრიგერი გაიშვება. შესაძლებელია ტრიგერის დაკავშირება რამდენიმე ბრძანებასთან.

– IF UPDATE (*სვეტის_სახელი*). მისი გამოყენება საშუალებას გვაძლევს ტრიგერი შევასრულოთ ცხრილის კონკრეტული სვეტის მოდიფიცირების დროს. გამოიყენება მხოლოდ INSERT და UPDATE ბრძანებებისთვის. ეს ეხება *ცხრილის_სახელი* არგუმენტით განსაზღვრულ ცხრილს.

– { AND | OR } UPDATE (*სვეტის_სახელი*). ეს კონსტრუქცია გამოიყენება წინა არგუმენტთან ერთად, როცა საჭიროა ტრიგერის გაშვება რამდენიმე სვეტის მოდიფიცირების შემთხვევაში. *სვეტის_სახელი* არგუმენტი შეიცავს იმ სვეტის სახელს, რომლის მოდიფიცირების შემთხვევაშიც უნდა გაიშვას ტრიგერი. AND მიუთითებს, რომ ტრიგერი გაიშვება მაშინ, როცა შესრულდება ორივე სვეტის მოდიფიცირება (მითითებული წინა და ამ არგუმენტში). OR მიუთითებს, რომ ტრიგერი გაიშვება მაშინ, როცა შესრულდება ერთ-ერთი სვეტის მოდიფიცირება.

– IF (COLUMNS_UPDATED()). ეს კონსტრუქცია გვატყობინებს, თუ რომელი სვეტები შეიცვალა ან დაემატა. ის გამოიყენება მხოლოდ UPDATE და INSERT ბრძანებებისათვის. COLUMNS_UPDATED() ფუნქცია გასცემს ორობით რიცხვს, რომლის თითოეული ბიტი შეესაბამება კონკრეტულ სვეტს. თუ ბიტი 1 მდგომარეობაშია, ეს ნიშნავს, რომ შესაბამისი სვეტი შეიცვალა. უმცროსი ბიტი შეესაბამება პირველ სვეტს, მეორე ბიტი მარჯვნიდან შეესაბამება მეორე სვეტს და ა.შ.

– *ბიტობრივი_ოპერატორი*. ეს არის ბიტობრივი დამუშავების ოპერატორი, რომლის გამოყენებით შეგვიძლია განვსაზღვროთ, შეიცვალა თუ არა კონკრეტული სვეტი. მაგალითად, & (ბიტობრივი AND) ოპერატორი შეგვიძლია გამოვიყენოთ იმის დასადგენად, შეიცვალა თუ არა რომელიმე სვეტი. ბიტობრივი ოპერატორები სრულდება COLUMNS_UPDATED() ფუნქციის მიერ გაცემულ ორობით რიცხვსა და *ბიტების_ნიღაბი* მნიშვნელობებზე.

– *ბიტების_ნიღაბი*. ეს არგუმენტი განსაზღვრავს ბიტების ნიღაბს ერთ ან მეტ სვეტში ცვლილებების განსაზღვრისათვის. თითოეული სვეტი წარმოდგენილია ცალკეული ბიტით. უმცროსი ბიტი შეესაბამება პირველ სვეტს, მეორე ბიტი - მეორე სვეტს და ა.შ. მაგალითად, თუ ცხრილს 7 სვეტი აქვს, მაშინ მე-2, მე-3 და მე-6 სვეტების შესამოწმებლად *ბიტების_ნიღაბი* არგუმენტი იღებს 0100110 მნიშვნელობას (რიცხვი 38).

– *შედარების_ოპერატორი*. ის განკუთვნილია შედარების ბიტობრივი ოპერაციის შესრულების შედეგად მიღებული მნიშვნელობის შესადარებლად გარკვეულ კრიტერიუმებთან. ხშირად გამოიყენება =, თუმცა დასაშვებია <, >, != ოპერატორებიც.

– *სვეტის_ნიღაბი*. ის განსაზღვრავს, შეიცვალა თუ არა შესამოწმებელი სვეტები.

ინფორმაცია ტრიგერების შესახებ მოთავსებულია მიმდინარე მონაცემთა ბაზის sys.triggers სისტემურ წარმოდგენაში (SELECT * FROM sys.triggers).

ტრიგერის შიგნით დაუშვებელია შემდეგი ოპერაციების შესრულება:

1. მონაცემთა ბაზების შექმნა, შეცვლა და წაშლა;
2. მონაცემთა ბაზის ან ტრანზაქციების ჟურნალის სარეზერვო ასლის აღდგენა;
3. RECONFIGURE, DISK RESIZE და DISK INIT ბრძანებების შესრულება.

ამ ბრძანებების შესრულება დაუშვებელია, რადგან შეუძლებელია მათი გაუქმება იმ ტრანზაქციის უკუქცევის დროს, რომელშიც ტრიგერი სრულდება.

INSERT და UPDATE ბრძანებებით შეცვლილი სვეტების სიის მისაღებად გამოიყენება ფუნქცია:
COLUMN_UPDATE()

სვეტის ცვლილების ფაქტის დასადასტურებლად გამოიყენება ფუნქცია:
UPDATE(სვეტის_სახელი)

თუ გაიცა TRUE, ეს იმას ნიშნავს, რომ სვეტი შეიცვალა.

ინფორმაციის მისაღებად სტრიქონების რაოდენობის შესახებ, რომლებიც შეიცვლება ტრიგერის წარმატებით დამთავრებისას, შეგვიძლია გამოვიყენოთ @@ROWCOUNT ფუნქცია. უნდა გვახსოვდეს, რომ ტრიგერი არ გაიშვება სტრიქონის შეცვლის მცდელობისას, არამედ გაიშვება ცვლილებების ბრძანების შესრულების მცდელობისას. ერთმა ასეთმა ბრძანებამ შეიძლება შეცვალოს რამდენიმე სტრიქონი, ამიტომ ტრიგერმა უნდა დაამუშაოს ყველა ასეთი სტრიქონი.

ტრიგერი სრულდება როგორც არაცხადად განსაზღვრული ტრანზაქცია, ამიტომ ტრიგერის შიგნით დასაშვებია ტრანზაქციების მართვის ბრძანებების გამოყენება. კერძოდ, შეცდომის შემთხვევაში ტრიგერის მუშაობის შესაწყვეტად და შესრულებული ცვლილებების გასაუქმებლად უნდა გამოვიყენოთ ROLLBACK TRANSACTION ბრძანება. ტრიგერის წარმატებით დამთავრების შემთხვევაში კი - COMMIT TRANSACTION.

როცა სერვერი იწყებს ტრიგერის შესრულებას, ის ქმნის ორ სპეციალურ ცხრილს: inserted და deleted. ამ ცხრილებში მოთავსებულია სტრიქონები, რომლებიც შესაბამისად ჩასმული და წაშლილი იქნება ტრანზაქციის დამთავრებისას. ამ ცხრილების სტრუქტურა იდენტურია იმ ცხრილის სტრუქტურისა, რომლისთვისაც ტრიგერია განკუთვნილი.

თითოეული ტრიგერისთვის იქმნება საკუთარი inserted და deleted ცხრილები. იმ ბრძანებებზე დამოკიდებულებით, რომლებმაც ტრიგერი გამოიწვია, ამ ცხრილების შემცველობა შეიძლება სხვადასხვა იყოს:

– INSERT ბრძანება. inserted ცხრილი შეიცავს ყველა სვეტს, რომელთა ჩასმასაც ვცდილობთ ცხრილში. deleted ცხრილში არ იქნება მოთავსებული არც ერთი სტრიქონი. ტრიგერის დამთავრების შემდეგ inserted ცხრილიდან ყველა სტრიქონი ჩაემატება ცხრილს.
– DELETE ბრძანება. deleted ცხრილში მოთავსდება ის სტრიქონები, რომელთა წაშლასაც ვცდილობთ. ტრიგერს შეუძლია შეამოწმოს თითოეული სტრიქონი და გადაწყვიტოს, ნებადართულია თუ არა მისი წაშლა. inserted ცხრილში არ იქნება მოთავსებული არც ერთი სტრიქონი.
– UPDATE ბრძანება. ამ ბრძანების შესრულების დროს deleted ცხრილში მოთავსებული იქნება სტრიქონების ძველი მნიშვნელობები, რომლებიც წაიშლება ტრიგერის წარმატებით დამთავრებისას. სტრიქონების ახალი მნიშვნელობები მოთავსებულია inserted ცხრილში. ეს სტრიქონები დაემატება საწყის ცხრილს ტრიგერის წარმატებით დამთავრებისას.

inserted და deleted ცხრილებში აკრძალულია ცვლილებების შეტანა. ტრიგერის გაშვების მომენტში საწყისი ცხრილი იმყოფება მდგომარეობაში, რომელშიც მას გადაიყვანდა ტრანზაქციის წარმატებით დამთავრება. inserted და deleted ცხრილები გამოიყენება სტრიქონების ცვლილებისათვის თვალყურის სადევნებლად.

მაგალითები.

ა. მოყვანილ მაგალითში ტრიგერი კრძალავს Personali_1 ცხრილში მონაცემების ჩამატებასა და ცვლილებას. RAISERROR ფუნქციას ახდენს შეცდომის გენერირებას, მონაცემების შეცვლისა და ჩამატების ოპერაცია შეწყდება და გაიცემა შესაბამისი შეტყობინება. მაგალითში იქმნება DML-ტრიგერი. მოთხოვნას აქვს სახე:

```
USE Shekveta;
-- თუ ტრიგერი არსებობს, ის წაიშლება
```

```
IF EXISTS
(
SELECT name
FROM sysobjects
WHERE name = 'Trigeri_1' AND type = 'TR'
)
DROP TRIGGER Trigeri_1;
GO
--        ტრიგერის შექმნა
CREATE TRIGGER Trigeri_1
ON Personali_1 FOR INSERT, UPDATE
AS RAISERROR (N'ცხრილში სრულდება მონაცემების შეცვლა ან ჩამატება', 16, 10);
```

ბ. ტრიგერის მუშაობის დემონსტრირებისათვის შევქმნათ Personali_2 ახალი ცხრილი Shekveta მონაცემთა ბაზის Personali ცხრილის საფუძველზე:

```
USE Shekveta;
SELECT * INTO Personali_2 FROM Personali;
SELECT * FROM Personali_2;
```

შედეგი:

	personaliID	gvari	ganyofileba	qalaqi	raioni	xelfasi	as...	st...
1	1	სამხარაძე საბა	სამედიცინო	თბილისი	დიდუბე	700,5	22	3
2	2	სამხარაძე ანა	სამედიცინო	ბათუმი	NULL	470,4	26	5
3	3	გაჩეჩილაძე ლია	სავაჭრო	თბილისი	საბურთალო	380,55	46	22
4	4	გაჩეჩილაძე ხათუნა	სასოფლო	თბილისი	ვერა	620,25	44	20
5	6	შენგელია ნატა	სავაჭრო	ზუგდიდი	NULL	500,95	28	6
6	7	კაპანაძე ეთერი	სასპორტო	რუსთავი	NULL	650,65	50	17
7	8	ყალაბეგიშვილი გია	სამედიცინო	თბილისი	ვერა	590,8	53	28
8	9	ხოშტარია ცისანა	სასპორტო	ზუგდიდი	NULL	430,5	57	30
9	10	ბარამიძე ზურა	სავაჭრო	ბათუმი	NULL	675,65	32	12
10	14	გაჩეჩილაძე თამი...	სასოფლო	ქუთაისი	NULL	320,8	42	15

ახლა Personali_2 ცხრილისთვის შევქმნათ DELETE ტრიგერი, რომელიც გამოიტანს ინფორმაციას წაშლის მცდელობების შესახებ და წაშლილი სტრიქონების რაოდენობის შესახებ. მოთხოვნას აქვს სახე:

```
USE Shekveta;
--        თუ ტრიგერი არსებობს, მაშინ ის წაიშლება
IF EXISTS
(
SELECT name
FROM sysobjects
WHERE name = 'Personali2_Delete' AND type = 'TR'
)
DROP TRIGGER Personali2_Delete;
GO
```

```
--      ტრიგერის შექმნა
CREATE TRIGGER Personali2_Delete ON Personali_2
FOR DELETE AS
PRINT STR(@@ROWCOUNT) + N' სტრიქონის წაშლის მცდელობა Personali_2 ცხრილიდან';
PRINT N'მომხმარებელი -  ' + CURRENT_USER;
IF CURRENT_USER <> 'dbo'
BEGIN
  PRINT N'წაშლა აკრძალულია';
  ROLLBACK TRANSACTION;
END
ELSE
  PRINT N'წაშლა ნებადართულია';
```

ტრიგერი აგრეთვე გამოიტანს იმ მომხმარებლის სახელს, რომელიც DELETE ბრძანებას ასრულებს. თუ მომხმარებელი არ არის dbo, მაშინ ის სტრიქონებს ვერ წაშლის და გაიცემა შესაბამისი შეტყობინება.

ახლა Personali_2 ცხრილიდან ერთი სტრიქონი წავშალოთ:

```
DELETE FROM Personali_2
WHERE gvari = N'გაჩეჩილაძე' AND saxeli = N'თამილა';
```

შედეგი:

```
      1 სტრიქონის წაშლის მცდელობა Personali_2 ცხრილიდან
 მომხმარებელი -  dbo
 წაშლა ნებადართულია

 (1 row(s) affected)
```

გ. შევქმნათ ტრიგერი, რომელიც asaki სვეტის შეცვლის ნებას აძლევს ყველა მომხმარებელს dbo-ს გარდა. პროგრამულ კოდს აქვს სახე:

```
USE Shekveta;
--      თუ ტრიგერი არსებობს, მაშინ ის წაიშლება
IF EXISTS
(
SELECT name
FROM sysobjects
WHERE name = 'Personali2_ganyofileba' AND type = 'TR'
)
DROP TRIGGER Personali2_ganyofileba;
GO
--      ტრიგერის შექმნა
CREATE TRIGGER Personali2_ganyofileba ON Personali_2
FOR UPDATE AS
SET NOCOUNT ON
PRINT N'მონაცემების შეცვლის მცდელობა Personali_2 ცხრილში'
IF UPDATE(asaki) PRINT N'asaki სვეტის შეცვლა'
IF UPDATE(ganyofileba) PRINT N'ganyofileba სვეტის შეცვლა'
IF UPDATE(qalaqi) PRINT N'qalaqi სვეტის შეცვლა'
```

```
IF ( ( CURRENT_USER = 'dbo' ) AND UPDATE(asaki) )
BEGIN
  PRINT N'dbo მომხმარებელს არ აქვს ასაკის შეცვლის უფლება'
  ROLLBACK TRANSACTION
END
```

ახლა, შევცვალოთ მითითებული თანამშრომლის ასაკი, იმ ქალაქის სახელი, რომელშიც ის ცხოვრობს და განყოფილების სახელი, რომელშიც ის მუშაობს:

```
UPDATE Personali_2 SET asaki = 50, ganyofileba = N'სასპორტო', qalaqi =  N'საგარეჯო'
        WHERE gvari = N'სამხარაძე' AND saxeli = N'რომანი';
```

შედეგი:

```
მოწაცემების შეცვლის მცდელობა Personali_2 ცხრილში
asaki სვეტის შეცვლა
ganyofileba სვეტის შეცვლა
qalaqi სვეტის შეცვლა
dbo მომხმარებელს არ აქვს ასაკის შეცვლის უფლება
```

ახლა შევცვალოთ მითითებული თანამშრომლის იმ ქალაქის სახელი, რომელშიც ის ცხოვრობს და განყოფილების სახელი, რომელშიც ის მუშაობს:

```
UPDATE Personali_2 SET ganyofileba = N'სასპორტო', qalaqi =  N'საგარეჯო'
WHERE gvari = N'სამხარაძე' AND saxeli = N'რომანი';
```

ამ შემთხვევაში, ცვლილებები შესრულდება ganyofileba და qalaqi სვეტებში, რადგან არ ხდება asaki სვეტის ცვლილება.

დ. შევქმნათ ტრიგერი, რომელიც ახალ ხელშეკრულებას არ გაგვაფორმებინებს შემკვეთთან, რომლის სახელი და გვარია 'სამხარაძე' და სახელი 'ბექა'. მოთხოვნას აქვს სახე:

```
USE Shekveta;
--      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_1 FOR INSERT, UPDATE AS
IF EXISTS
( SELECT * FROM Shemkveti_1 WHERE gvari = N'სამხარაძე' AND saxeli = N'ბექა' )
BEGIN
        PRINT N'სამხარაძე ბექასთან ახალი ხელშეკრულების გაფორმება დაუშვებელია '
        ROLLBACK TRANSACTION
END
```

ახლა ვცადოთ ხელშეკრულების გაფორმება შემკვეთთან - 'სამხარაძე ბექა':

```
UPDATE Shemkveti_1 SET gvari = N'სამხარაძე', saxeli = 'ბექა'
WHERE gvari = N'ჭიკაძე' AND saxeli = N'გელა';
```

გაიცემა შეტყობინება:

```
სამხარაძე ბექასთან ახალი ხელშეკრულების გაფორმება დაუშვებელია
```

ე. ამ მაგალითისთვის ჯერ შევქმნათ Shemkveti_3 ცხრილი Shemkveti ცხრილის ბაზაზე:

```
SELECT * INTO Shemkveti_3 FROM Shemkveti;
SELECT * FROM Shemkveti_3;
```

ახლა შევქმნათ ტრიგერი, რომელიც აკრძალავს სტრიქონების წაშლას იმ შემკვეთების შესახებ, რომლებიც იურიდიულ პირს წარმოადგენს. მოთხოვნას აქვს სახე:

```
USE Shekveta;
--      თუ ტრიგერი არსებობს, მაშინ ის წაიშლება
IF EXISTS
 (
SELECT name
FROM sysobjects
WHERE name = 'Trigeri_4' AND type = 'TR'
)
DROP TRIGGER Trigeri_4;
GO
--      ტრიგერის შექმნა
CREATE TRIGGER Trigeri_4 ON Shemkveti_3 FOR DELETE AS
DECLARE @@Result INT
SET @@Result = 1
IF EXISTS
(
SELECT *
FROM deleted
WHERE iuridiuli_fizikuri = N'იურიდიული'
)
BEGIN
 PRINT N'იურიდიული პირის წაშლის მცდელობა'
 SET @@Result = 0
END
IF @@Result = 0
BEGIN
 PRINT N'წაშლა აკრძალულია'
 ROLLBACK TRANSACTION
END
```

@@Result ცვლადი ასრულებს წაშლის წარუმატებელი მცდელობის ალმის როლს. ახლა შევეცადოთ სტრიქონების წაშლას:

```
DELETE FROM Shemkveti_3;
```

გაიცემა შეტყობინება:

```
იურიდიული პირის წაშლის მცდელობა
წაშლა აკრძალულია
```

ტრიგერის წაშლა

ტრიგერის წასაშლელად გამოიყენება DROP TRIGGER ბრძანება. მისი სინტაქსია:

```
DROP TRIGGER [ სქემის_სახელი . ] ტრიგერის_სახელი [,...n]
```

როგორც სინტაქსიდან ჩანს, შესაძლებელია რამდენიმე ტრიგერის ერთდროულად

წაშლა.

მაგალითი. წავშალოთ Trigeri_1 ტრიგერი:

```
DROP TRIGGER Trigeri_1;
```

ტრიგერების შექმნისას გასათვალისწინებელი რეკომენდაციები

ტრიგერის ტანში დასაშვებია Transact_SQL-ის ბრძანებების გამოყენება, აგრეთვე სისტემური და მომხმარებლის მიერ განსაზღვრული შენახული პროცედურების გამოძახება.

უმჯობესია ტრიგერმა არ დააბრუნოს შედეგი, თუმცა ეს აკრძალული არ არის.

რეკომენდებული არ არის SELECT ბრძანების გამოყენება, რომელიც შედეგს აბრუნებს, თუმცა სერვერი მაინც გასცემს შეტყობინებას ბრძანებების ჯგუფის წარმატებით შესრულების შესახებ. ამ შეტყობინების აკრძალვისთვის უნდა შევასრულოთ SET NOCOUNT ბრძანება.

ტრიგერის დაწერისას მინიმუმამდე უნდა დავიყვანოთ მიმართვები გარე ცხრილებთან და მონაცემთა ბაზის სხვა ობიექტებთან. თუ წავშლით იმ ობიექტს, რომელსაც ტრიგერი მიმართავს, მაშინ ტრიგერის გაშვებისას სერვერი გასცემს შეტყობინებას შეცდომის შესახებ. თუ ობიექტის წაშლის შემდეგ შევქმნით ასეთივე სახელის ახალ ობიექტს, რომელსაც ისეთივე სტრუქტურა აქვს, როგორიც ჰქონდა წაშლილ ობიექტს, მაშინ ტრიგერი უპრობლემოდ იმუშავებს. თუ ახალ ობიექტს განსხვავებული სტრუქტურა აქვს, მაშინ ტრიგერში უნდა შევიტანოთ შესაბამისი ცვლილებები.

არ არის რეკომენდებული ტრიგერიდან დროებით ცხრილებთან მიმართვა, თუ არ არის დროებითი ცხრილების სტრუქტურის მუდმივობის გარანტია.

ტრიგერის ტანში დასაშვებია SET ბრძანებების გამოყენება. დაყენებულ მნიშვნელობებს ძალა აქვს მხოლოდ ტრიგერის მუშაობისას.

ტრიგერების მართვა

ტრიგერის სახელის შეცვლა

ტრიგერის სახელის შესაცვლელად გამოიყენება sp_rename სისტემური შენახული პროცედურა.

მაგალითი. Trigeri_1 ტრიგერის სახელი შევცვალოთ ახალი სახელით - Trigeri_2:

```
EXEC sp_rename 'Trigeri_1', 'Trigeri_2', 'OBJECT';
```

ტრიგერის შესახებ ინფორმაციის მიღება

ტრიგერის კოდის მისაღებად უნდა შევასრულოთ sp_helptext სისტემური შენახული პროცედურა. მისი სინტაქსია:

sp_helptext [@objname =] '*ტრიგერის_სახელი*'

'*ტრიგერის_სახელი*' არის იმ ტრიგერის სახელი, რომლის შესახებ გვინდა ინფორმაციის მიღება.

მაგალითი. მივიღოთ ინფორმაცია Trigeri_3 ტრიგერის შესახებ:

```
EXEC sp_helptext 'Trigeri_3';
```

შედეგი:

	Text
1	-- ტრიგერის შექმნა
2	CREATE TRIGGER Trigeri_3 ON Shemkveti_1 FOR INSERT, UPDATE AS
3	IF EXISTS
4	(SELECT * FROM Shemkveti_1 WHERE gvari = N'სამხარაძე ბექა')
5	BEGIN
6	PRINT N'სამხარაძე ბექასთან ახალი ხელშეკრულების გაფორმება დაუშვებელია '
7	ROLLBACK TRANSACTION
8	END

კონკრეტული ცხრილისთვის განსაზღვრული ტრიგერების სიის მისაღებად გამოიყენება sp_helptrigger სისტემური შენახული პროცედურა. მისი სინტაქსია:
sp_helptrigger [@tabname =] '*ცხრილის_სახელი*'
[, [@triggertype =] '*ტრიგერის_ტიპი*']

'*ცხრილის_სახელი*' არის იმ ცხრილის სახელი, რომლის ტრიგერების სიაც გვაინტერესებს. 'ტრიგერის_ტიპი' არის იმ ტრიგერების ტიპი, რომელთა შესახებ გვაინტერესებს ინფორმაცია. ის იღებს მნიშვნელობებს: 'INSERT', 'UPDATE' და 'DELETE'.

მაგალითები.

ა. მივიღოთ ინფორმაცია Shemkveti ცხრილის ტრიგერების შესახებ. მოთხოვნას აქვს სახე:
EXEC sp_helptrigger 'Shemkveti';

შედეგი:

	trigger_name	trigger_owner	isupdate	isdelete	isinsert	isafter	isinsteadof	trigger_schema
1	Trigeri_4	dbo	0	1	0	1	0	dbo

ბ. მივიღოთ ინფორმაცია Shemkveti ცხრილის DELETE-ტრიგერების შესახებ. მოთხოვნას აქვს სახე:
EXEC sp_helptrigger 'Shemkveti', 'DELETE';

შედეგი:

	trigger_name	trigger_owner	isupdate	isdelete	isinsert	isafter	isinsteadof	trigger_schema
1	Trigeri_4	dbo	0	1	0	1	0	dbo

იმ ობიექტების სანახავად, რომლებზეც ტრიგერია დამოკიდებული, უნდა შევასრულოთ sp_depends სისტემური შენახული პროცედურა. მისი სინტაქსია:
sp_depends [@objname =] '*ტრიგერის_სახელი*'

მაგალითი. მივიღოთ ინფორმაცია იმ ობიექტების შესახებ, რომლებზეც Personali2_ganyofileba ტრიგერია დამოკიდებული. მოთხოვნას აქვს სახე:
USE Shekveta;
EXEC sp_depends 'Personali2_ganyofileba';

შედეგი:

	name	type	updat...	select...	column
1	dbo.Personali_2	user table	no	yes	ganyofileba
2	dbo.Personali_2	user table	no	yes	qalaqi
3	dbo.Personali_2	user table	no	yes	asaki

თავი 16. ცხრილური გამოსახულებები

ცხრილური გამოსახულება არის მოთხოვნა, რომელიც ვირტუალურ ცხრილს გასცემს. ის შეგვიძლია მონაცემების დამუშავების ინსტრუქციებში ცხრილების მსგავსად გამოვიყენოთ. SQL სერვერი ცხრილური გამოსახულებების ოთხ ტიპს უზრუნველყოფს: წარმოებული ცხრილები, საერთო ცხრილური გამოსახულებები (common table expression, CTE), წარმოდგენები (view) და შემცვლელი ცხრილური ფუნქციები (inline table-valued function, TVF).

ცხრილური გამოსახულებების გამოყენების ერთ-ერთი უპირატესობაა - გარე მოთხოვნის ნებისმიერ სინტაქსურ ელემენტში იმ ფსევდონიმებთან მიმართვის შესაძლებლობა, რომლებიც სვეტებს შიგა მოთხოვნის SELECT ელემენტში (განყოფილებაში) მიენიჭა. ეს საშუალებას გვაძლევს გვერდი ავუაროთ შეზღუდვას, რომელიც კრძალავს SELECT განყოფილებაში მინიჭებულ ფსევდონიმებთან მიმართვას მოთხოვნის იმ განყოფილებებში, რომლებიც ლოგიკურად SELECT განყოფილებამდე მუშავდება (როგორიცაა WHERE ან GROUP BY).

ცხრილური გამოსახულებები გამოიყენება გამომდინარე მოთხოვნის დამუშავების ლოგიკიდან და არა მწარმოებლურობის მოსაზრებებიდან. ზოგადად, ცხრილური გამოსახულებები არ ახდენს არც უარყოფით და არც დადებით გავლენას მოთხოვნის შესრულების მწარმოებლურობაზე.

წარმოებული ცხრილები

წარმოებულ ცხრილებს სხვანაირად ცხრილური ქვემოთხოვნები ეწოდება და გარე SELECT მოთხოვნის FROM განყოფილებაში განისაზღვრება. როგორც კი გარე მოთხოვნა მთავრდება, წარმოებული ცხრილი იშლება.

ნებისმიერი ტიპის ცხრილური გამოსახულების განსაზღვრა შეიძლება მოთხოვნაში, რომელიც სამ პირობას აკმაყოფილებს:

1. *სტრიქონების მიმდევრობა ნებისმიერი უნდა იყოს.* იგულისხმება, რომ ცხრილური გამოსახულება გასცემს რელაციურ ცხრილს, ხოლო რელაციური ცხრილის სტრიქონებს არ აქვს გარკვეული მიმდევრობა. ამიტომ, ANSI SQL სტანდარტი არ იძლევა ORDER BY განყოფილების გამოყენების უფლებას იმ მოთხოვნებში, რომლებიც ცხრილურ გამოსახულებებს განსაზღვრავენ. T-SQL ენა ამ შეზღუდვას იცავს ხშირ შემთხვევაში. მაგრამ, თუ TOP ელემენტისა და ORDER BY განყოფილების შემცველ მოთხოვნაში მოცემულია ცხრილური გამოსახულება, მაშინ ORDER BY განყოფილება გამოიყენება მხოლოდ ლოგიკური მოწესრიგებისთვის, რომელიც TOP ელემენტს სჭირდება და არა მონაცემების წარმოდგენისთვის.
2. *ყველა სვეტს უნდა ჰქონდეს სახელი.* ეს იმას ნიშნავს, რომ SELECT მოთხოვნაში სვეტის ფსევდონიმი უნდა მივანიჭოთ ყველა გამოსახულებას, რომელიც გამოიყენება ცხრილური გამოსახულების განსაზღვრისთვის.
3. *სვეტების სახელები უნიკალური უნდა იყოს.* თუ ცხრილური გამოსახულების განმსაზღვრელ მოთხოვნაში ორი ცხრილი ერთიანდება, რომლებშიც ერთი და იმავე სახელის მქონე სვეტია, მაშინ მათ სხვადასხვა ფსევდონიმი უნდა მივანიჭოთ.

სვეტებისთვის ფსევდონიმების მინიჭება

დავუშვათ, გვაინტერესებს სხვადასხვა შემკვეთების რაოდენობა, რომლებიც შეკვეთებს ახორციელებდნენ თითოეული წლის განმავლობაში, შეკვეთების თარიღში მითითებულ ყველა წელს. მოყვანილი მოთხოვნა არაკორექტულია:

```
USE Shekveta;
SELECT YEAR(tarigi_dawyebis) AS dawyebisweli, COUNT(DISTINCT shemkvetiID) AS raodshemk
FROM Xelshekruleba
GROUP BY dawyebisweli;
```

ამ მოთხოვნის შესრულების შედეგად, SQL სერვერი გასცემს შეყტობინებას შეცდომის შესახებ, რადგან GROUP BY განყოფილება მიმართავს სვეტის ფსევდონიმს (dawyebisweli), რომელიც SELECT განყოფილებაში იყო მინიჭებული, ხოლო GROUP BY განყოფილება ლოგიკურად SELECT განყოფილებამდე მუშავდება.

ეს ამოცანა შეგვეძლო გადაგვეწყვიტა YEAR(dawyebis_tarigi) გამოსახულების გამოყენებით GROUP BY და SELECT განყოფილებებში. მოთხოვნაში ერთი და იმავე გამოსახულების ორი ასლის ჩასმამ შეიძლება გაართულოს პროგრამული კოდის გაგება და თანხლება, აგრეთვე გაზარდოს შეცდომების ალბათობა. ამ პრობლემის გადასაწყვეტად უნდა გამოვიყენოთ ცხრილური გამოსახულება, რომელშიც გამოყენებული იქნება აღნიშნული გამოსახულების ერთი ასლი. მოყვანილ მოთხოვნაში YEAR(dawyebis_tarigi) გამოსახულებისთვის გამოყენებულია ფსევდონიმის შიგა დანიშვნის ფორმა.

მოყვანილ მოთხოვნაში წარმოებული ცხრილი ფსევდონიმის შიგა დანიშვნის ფორმას იყენებს:

```
USE Shekveta;
SELECT  dawyebisweli, COUNT(DISTINCT shemkvetiID) AS raodshemk
FROM
(
SELECT YEAR(tarigi_dawyebis) AS dawyebisweli, shemkvetiID
FROM Xelshekruleba
) AS T1
GROUP BY dawyebisweli;
```

შედეგი:

	dawyebisweli	raodshemk
1	2000	2
2	2001	1
3	2002	4
4	2003	1
5	2004	2
6	2005	2
7	2006	2
8	2007	1

მოცემულ პროგრამულ კოდში განისაზღვრება T1 წარმოებული ცხრილი. ის მიიღება შიგა მოთხოვნისგან, რომელიც შეკვეთის წელს და შემკვეთის იდენტიფიკატორს გასცემს. შიგა მოთხოვნის სვეტების სიაში YEAR(tarigi_dawyebis) გამოსახულებისთვის dawyebisweli ფსევდონიმის მისანიჭებლად ფსევდონიმების ჩადგმული (შიგა) დანიშვნის ფორმა გამოიყენება. გარე მოთხოვნა შეიძლება მიმართავდეს სვეტის ფსევდონიმს ორივე სინტაქსურ

ელემენტში (SELECT და GROUP BY). ის მიმართავს T1 წარმოებულ ცხრილს, რომლის სვეტებსაც, შესაბამისად, dawyebisweli (შეკვეთის წელი) და shemkvetiID (შემკვეთის ID) ჰქვიათ.

სერვერი ახდენს ცხრილური გამოსახულების განსაზღვრის გახსნას და უშუალოდ მიმართავს მის საბაზო ობიექტებს. გახსნის შემდეგ ზემოთ მოყვანილი მოთხოვნა შემდეგ სახეს მიიღებს:

```
SELECT YEAR(tarigi_dawyebis) AS dawyebisweli, COUNT(DISTINCT shemkvetiID) AS raodshemk
FROM Xelshekruleba
GROUP BY  YEAR(tarigi_dawyebis);
```

ზოგიერთ შემთხვევაში, შეიძლება უპირატესი აღმოჩნდეს სვეტების ფსევდონიმების დანიშვნის მეორე - გარე ფორმა. ამ ფორმაში სვეტებისთვის დასანიშნი ფსევდონიმები ეთითება მრგვალ ფრჩხილებში, რომელიც ცხრილური გამოსახულების სახელს მოსდევს. გარე ფორმის გამოყენებით მოთხოვნა მიიღებს სახეს:

```
USE Shekveta;
SELECT  dawyebisweli, COUNT(DISTINCT shemkvetiID) AS raodshemk
FROM
(
SELECT YEAR(tarigi_dawyebis), shemkvetiID
FROM Xelshekruleba
)
AS T1(dawyebisweli,shemkvetiID)
GROUP BY dawyebisweli;
```

შედეგი:

	dawyebisweli	raodshemk
1	2000	2
2	2001	1
3	2002	4
4	2003	1
5	2004	2
6	2005	2
7	2006	2
8	2007	1

გადასაწყვეტ ამოცანაზე დამოკიდებულებით ხან შიგა ფორმა შეიძლება გამოვიყენოთ, ხან - გარე.

არგუმენტების გამოყენება

მოთხოვნაში, რომელიც წარმოებულ ცხრილს განსაზღვრავს, შეგვიძლია არგუმენტები გამოვიყენოთ. არგუმენტები შეიძლება იყოს ლოკალური ცვლადები და შენახული პროცედურის ან ფუნქციის შესასვლელი პარამეტრები. მოყვანილ პროგრამულ კოდში გამოცხადებული და ინიციალიზებულია @shemkvetiID ლოკალური ცვლადი, ხოლო A წარმოებული ცხრილის განსაზღვრისათვის გამოყენებული მოთხოვნა ამ ლოკალურ ცვლადს WHERE განყოფილებაში მიმართავს:

```
USE Shekveta;
DECLARE @personaliID AS INT = 3;
SELECT dawyebisweli, COUNT(DISTINCT shemkvetiID) AS raodshemk
```

```
FROM
(
SELECT  YEAR(tarigi_dawyebis) AS dawyebisweli,  shemkvetiID
FROM Xelshekruleba
WHERE personaliID = @personaliID
) AS T1
GROUP BY  dawyebisweli;
```

შედეგი:

	dawyebisweli	raodshemk
1	2001	1
2	2002	2
3	2005	1
4	2006	1

მოთხოვნა გასცემს სხვადასხვა შემკვეთის რაოდენობას, რომელთა ხელშეკრულება წლის განმავლობაში იმ თანამშრომელთან გაფორმდა, რომლის იდენტიფიკატორი @personaliID ცვლადში ინახება.

ჩადგმულობა

თუ გვჭირდება წარმოებული ცხრილის განსაზღვრა იმ მოთხოვნის გამოყენებით, რომელიც თვითონ მიმართავს წარმოებულ ცხრილს, მაშინ ადგილი ექნება წარმოებული ცხრილების ჩადგმულობას. წარმოებული ცხრილების ჩადგმა იმის შედეგია, რომ ის გარე მოთხოვნის FROM განყოფილებაში განისაზღვრება, და არა ცალკე. ჩადგმულობა პროგრამული კოდის გართულებას იწვევს და მის კითხვას აძნელებს.

მოყვანილი პროგრამული კოდი გასცემს წლებს, რომლებშიც ხელშეკრულებები გაფორმდა და შემკვეთების რაოდენობას, რომლებმაც თითოეული წლის განმავლობაში გააფორმეს ხელშეკრულება. ამასთან ამოირჩევა მხოლოდ მხოლოდ ის წლები, რომლებშიც ერთზე მეტმა შემკვეთებმა გააფორმა ხელშეკრულება. მოყვანილი მოთხოვნა შედგება ჩადგმული წარმოებული ცხრილებისგან:

```
--      ლისტინგი 16.1
USE Shekveta;
SELECT dawyebisweli, raodshemk
FROM
(
SELECT dawyebisweli, COUNT(DISTINCT shemkvetiID) AS raodshemk
FROM
        (
        SELECT YEAR(tarigi_dawyebis) AS dawyebisweli, shemkvetiID
        FROM Xelshekruleba
        )
AS T1
GROUP BY dawyebisweli
) AS T2
WHERE raodshemk > 1;
```

შედეგი:

	dawyebisweli	raodshemk
1	2000	2
2	2002	4
3	2004	2
4	2005	2
5	2006	2

შიგა მოთხოვნის ამოცანაა - YEAR(tarigi_dawyebis) გამოსახულებას dawyebisweli ფსევდონიმი მიანიჭოს. T1 ცხრილის მიმართ მოთხოვნა მიმართავს dawyebisweli ფსევდონიმს GROUP BY და SELECT განყოფილებებში, და COUNT(DISTINCT shemkvetiID) გამოსახულებას raodshemk ფსევდონიმს ანიჭებს. T1 ცხრილის მიმართ მოთხოვნა გამოიყენება T2 წარმოებული ცხრილის მისაღებად. T2 ცხრილის მიმართ მოთხოვნა raodshemk ფსევდონიმს WHERE განყოფილებაში იმისთვის მიმართავს, რომ ამორჩეული იყოს ის წლები, რომლებშიც ერთზე მეტმა შემკვეთმა გააფორმა შეკვეთა წლის განმავლობაში.

ცხრილური გამოსახულებების გამოყენების ზოგადი მიზანია მოთხოვნების გამარტივება გამოსახულებების ნაცვლად ფსევდონიმების მრავალჯერ გამოყენების გზით. რთული გამოსახულების შემთხვევაში, უმჯობესია მის ნაცვლად ფსევდონიმის გამოყენება. მაგრამ, რიგ შემთხვევებში, გამოსახულების გამოყენებისას (როცა გამოსახულება მარტივია), მთლიანად მოთხოვნა შეიძლება უფრო მარტივი აღმოჩნდეს, ვიდრე ფსევდონიმების გამოყენებისას. მოყვანილ მოთხოვნაში ფსევდონიმის ნაცვლად გამოსახულება გამოიყენება:

```
USE Shekveta;
SELECT YEAR(tarigi_dawyebis) AS dawyebisweli, COUNT(DISTINCT shemkvetiID) AS raodshemk
FROM Xelshekruleba
GROUP BY YEAR(tarigi_dawyebis)
HAVING COUNT(DISTINCT shemkvetiID) > 1;
```

ეს მოთხოვნა უფრო მარტივი და ადვილად აღსაქმელია, ვიდრე ჩადგმული.

შედეგი:

	dawyebisweli	raodshemk
1	2000	2
2	2002	4
3	2004	2
4	2005	2
5	2006	2

მრავლობითი მიმართვები

მეორე პრობლემა, დაკავშირებული წარმოებულ ცხრილებთან, გამოწვეულია იმით, რომ ისინი განსაზღვრულია გარე მოთხოვნის FROM განყოფილებაში, და არა გარე მოთხოვნამდე. გარე მოთხოვნის FROM განყოფილების განხილვამდე წარმოებული ცხრილი არ არსებობს. შედეგად ვერ მივმართავთ წარმოებული ცხრილის ეგზემპლარებს. ამიტომ, იძულებული ვიქნებით განვსაზღვროთ რამდენიმე წარმოებული ცხრილი, დაფუძნებული ერთსა და იმავე მოთხოვნაზე.

მაგალითი. დავუშვათ, გვაინტერესებს შემკვეთების რაოდენობა წლების მიხედვით,

წინა წელს შემკვეთების რაოდენობა და ნაზრდი. მოყვანილ მოთხოვნაში გამოყენებულია რამდენიმე წარმოებული ცხრილი, რომლებიც ერთსა და იმავე მოთხოვნას ეფუძნება:

```
--        ლისტინგი 16.2.
USE Shekveta;
SELECT Mimd.dawyebisweli, Mimd.raodshemk AS mimdraodshemk,
          Wina.raodshemk AS winaraodshemk, Mimd.raodshemk - Wina.raodshemk AS nazrdi
FROM
(
SELECT  YEAR(tarigi_dawyebis) AS dawyebisweli, COUNT(DISTINCT shemkvetiID) AS raodshemk
FROM Xelshekruleba
GROUP BY YEAR(tarigi_dawyebis)
) AS Mimd
LEFT OUTER JOIN
(
SELECT  YEAR(tarigi_dawyebis) AS dawyebisweli, COUNT(DISTINCT shemkvetiID) AS raodshemk
FROM Xelshekruleba
GROUP BY YEAR(tarigi_dawyebis)
) AS Wina
          ON Mimd.dawyebisweli = Wina.dawyebisweli + 1;
```

ეს მოთხოვნა აერთიანებს ცხრილური გამოსახულების ორ ეგზემპლარს ორი წარმოებული ცხრილის შექმნის მიზნით: პირველია Mimd, რომელიც წარმოგვიდგენს მიმდინარე წლებს, მეორეა Wina, რომელიც წარმოგვიდგენს წინა წლებს. Mimd.dawyebisweli = Wina.dawyebisweli + 1 შეერთების პირობა იძლევა იმის გარანტიას, რომ პირველი წარმოებული ცხრილის თითოეული სტრიქონი დაკავშირებული იქნება მეორე ცხრილის წინა წელთან. როდესაც შემოგვაქვს მარცხენა გარე შეერთების ოპერაცია, სტრიქონების შედეგობრივ ნაკრებში შეგვიძლია ჩავრთოთ პირველი წელი Mimd ცხრილიდან, რომელსაც არ აქვს წინა წელი. გარე მოთხოვნის SELECT განყოფილებაში გამოითვლება სხვაობა შემკვეთების რაოდენობებს შორის, რომლებმაც გააფორმეს ხელშეკრულებები მიმდინარე და წინა წლებში.

შედეგი:

	dawyebisweli	mimdraodshemk	winaraodshemk	nazrdi
1	2000	2	NULL	NULL
2	2001	1	2	-1
3	2002	4	1	3
4	2003	1	4	-3
5	2004	2	1	1
6	2005	2	2	0
7	2006	2	2	0
8	2007	1	2	-1

ერთი წარმოებული ცხრილის რამდენიმე ეგზემპლარზე მიმართვის შეუძლებლობა გვაიძულებს შევინახოთ ერთი და იმავე მოთხოვნის განსაზღვრის რამდენიმე ასლი. ეს იწვევს პროგრამული კოდის ზრდას, რთულდება მისი უზრუნველყოფა და იზრდება შეცდომების ალბათობა.

საერთო ცხრილური გამოსახულებები

საერთო ცხრილური გამოსახულებები (Common table expression, CTE, სცგ) - ცხრილური გამოსახულებების მეორე ტიპია, რომელიც ძალიან გავს წარმოებულ ცხრილებს, ოღონდ მასთან შედარებით მნიშვნელოვანი უპირატესობები აქვს. ამ უპირატესობებს ქვემოთ შევეხებით.

საერთო ცხრილური გამოსახულება WITH ინსტრუქციით განისაზღვრება და შემდეგი სინტაქსი აქვს:

WITH <*სცგ_სახელი*> [(<*შედეგის_სვეტების_სია*>)]
AS
(
    <*სცგ-ს_ განმსაზღვრელი_შიგა_მოთხოვნა*>
)
<*გარე_მოთხოვნა_სცგ-ს_მიმართ*> ;

საერთო ცხრილური გამოსახულების განმსაზღვრელი შიგა მოთხოვნა უნდა აკმაყოფილებდეს წინა განყოფილებაში მოყვანილ სამ პირობას. ქვემოთ მოყვანილ მოთხოვნაში განისაზღვრება საერთო ცხრილური გამოსახულება, რომლის სახელია TbilisiShemkveti (შემკვეთები თბილისიდან). ის ეფუძნება მოთხოვნას, რომელიც გასცემს თბილისელ შემკვეთებს. გარე მოთხოვნა ირჩევს ყველა სტრიქონს საერთო ცხრილური გამოსახულებიდან. მოთხოვნას აქვს სახე:

```
USE Shekveta;
WITH TbilisiShemkveti AS
(
SELECT shemkvetiID, firmis_dasaxeleba
FROM Shemkveti
WHERE qalaqi = N'თბილისი'
)
SELECT * FROM TbilisiShemkveti;
```

შედეგი:

	shemkvetiID	firmis_dasaxeleba
1	1	ჯეონეთი
2	2	NULL
3	13	R&Co.
4	15	NULL
5	16	NULL

WITH სინტაქსური განყოფილება T-SQL ენაში სხვადასხვა მიზნებისთვის გამოიყენება. თუ ის საერთო ცხრილური გამოსახულების განსაზღვრისთვის გამოიყენება, მაშინ მის წინ მოთავსებული ბრძანება (ინსტრუქცია) წერტილ-მძიმით უნდა დამთავრდეს.

სვეტებისთვის ფსევდონიმების დანიშვნა

წარმოებული ცხრილების მსგავსად საერთო ცხრილურ გამოსახულებებში შესაძლებელია სვეტებისთვის ფსევდონიმების დანიშვნის ორი ფორმის გამოყენება: ჩადგმული (შიგა) და გარე.

ქვემოთ მოყვანილია სვეტებისთვის ფსევდონიმების დანიშვნის ჩადგმული ფორმის

გამოყენების მაგალითი:

```
USE Shekveta;
WITH T1 AS
(
SELECT YEAR(tarigi_dawyebis) AS dawyebisweli, shemkvetiID
FROM Xelshekruleba
)
SELECT dawyebisweli, COUNT(DISTINCT shemkvetiID) AS raodshemk
FROM T1
GROUP BY dawyebisweli;
```

ახლა იგივე მოთხოვნა ჩავწეროთ გარე ფორმის გამოყენებით:

```
WITH T1 (dawyebisweli, shemkvetiID) AS
(
SELECT YEAR(tarigi_dawyebis), shemkvetiID
FROM Xelshekruleba
)
SELECT dawyebisweli, COUNT(DISTINCT shemkvetiID) AS raodshemk
FROM T1
GROUP BY dawyebisweli;
```

ორივე შემთხვევაში ერთსა და იმავე შედეგს ვიღებთ.

შედეგი:

	dawyebisweli	raodshemk
1	2000	2
2	2001	1
3	2002	4
4	2003	1
5	2004	2
6	2005	2
7	2006	2
8	2007	1

არგუმენტების გამოყენება

მოთხოვნაში, რომელიც საერთო ცხრილური გამოსახულების განსაზღვრისთვის გამოიყენება, შეგვიძლია არგუმენტები მივუთითოთ. მოთხოვნას აქვს სახე:

```
USE Shekveta;
DECLARE @shemkvetiID AS INT = 3;
WITH T1 AS
(
SELECT YEAR(tarigi_dawyebis) AS dawyebisweli, shemkvetiID
FROM Xelshekruleba
WHERE shemkvetiID = @shemkvetiID
)
SELECT dawyebisweli, COUNT(DISTINCT shemkvetiID) AS raodshemk
FROM T1
GROUP BY dawyebisweli;
```

შედეგი:

	dawyebisweli	raodshemk
1	2002	1
2	2003	1
3	2004	1

რამდენიმე საერთო ცხრილური გამოსახულების განსაზღვრა

ის ფაქტი, რომ ჯერ განისაზღვრება საერთო ცხრილური გამოსახულება და შემდეგ გამოიყენება, წარმოებულ ცხრილებთან შედარებით რიგ უპირატესობას იძლევა. ერთ-ერთი მათგანი ისაა, რომ თუ გვინდა ერთი საერთო ცხრილური გამოსახულებიდან მივმართოთ მეორეს, არ არის ერთმანეთში მათი ჩასმის აუცილებლობა, როგორც წარმოებული ცხრილების შემთხვევაში. ამის ნაცვლად, ერთ WITH ინსტრუქციაში განვსაზღვრავთ რამდენიმე საერთო ცხრილურ გამოსახულებას, რომლებიც ერთმანეთისგან მძიმეებით გამოიყოფა. ნებისმიერი საერთო ცხრილური გამოსახულება შეიძლება მიმართავდეს ადრე განსაზღვრულ ყველა საერთო ცხრილური გამოსახულებას, და გარე მოთხოვნა შეიძლება მიმართავდეს ყველა საერთო ცხრილური გამოსახულებას. მაგალითად, ლისტინგ 16.1-ზე მოყვანილი კოდი საერთო ცხრილური გამოსახულების გამოყენებით ასე შეიძლება ჩავწეროთ:

```
WITH T1 AS
(
SELECT YEAR(tarigi_dawyebis) AS dawyebisweli, shemkvetiID
FROM Xelshekruleba
),
T2 AS
(
SELECT dawyebisweli, COUNT(DISTINCT shemkvetiID) AS raodshemk
FROM T1
GROUP BY dawyebisweli
)
SELECT dawyebisweli, raodshemk
FROM T2
WHERE raodshemk > 1;
```

შედეგი:

	dawyebisweli	raodshemk
1	2000	2
2	2002	4
3	2004	2
4	2005	2
5	2006	2

რადგან, ჩვენ საერთო ცხრილური გამოსახულებას განვსაზღვრავთ მის გამოყენებამდე, ამიტომ არ გვჭირდება ჩადგმული საერთო ცხრილური გამოსახულების გამოყენება. პროგრამულ კოდში თითოეული საერთო ცხრილური გამოსახულება გამოჩნდება როგორც მოდული. მოდულური მიდგომა პროგრამულ კოდს უფრო გასაგებს ხდის და ჩადგმულ წარმოებულ ცხრილებთან შედარებით მის თანხლებას აადვილებს.

მრავლობითი მიმართვები

საერთო ცხრილური გამოსახულების წინასწარი განსაზღვრა, მოთხოვნაში მის გამოყენებამდე, იძლევა კიდევ ერთ უპირატესობას. გარე მოთხოვნის FROM განყოფილების დამუშავების ეტაპზე საერთო ცხრილური გამოსახულება უკვე არსებობს. შედეგად შეგვიძლია მივმართოთ ერთი და იმავე საერთო ცხრილური გამოსახულების რამდენიმე ეგზემპლარს. ლისტინგ 16.2-ზე მოყვანილი პროგრამული კოდი საერთო ცხრილური გამოსახულების გამოყენებით შეიძლება ასე ჩავწეროთ:

```
USE Shekveta;
WITH weliRaodenoba AS
(
SELECT YEAR(tarigi_dawyebis) AS dawyebisweli, COUNT(DISTINCT shemkvetiID) AS raodshemk
FROM Xelshekruleba
GROUP BY YEAR(tarigi_dawyebis)
)
SELECT Mimd.dawyebisweli,
Mimd.raodshemk AS Mimdraodshemk, Wina.raodshemk AS Winaraodshemk,
Mimd.raodshemk - Wina.raodshemk AS nazrdi
FROM weliRaodenoba AS Mimd LEFT OUTER JOIN weliRaodenoba AS Wina
ON Mimd.dawyebisweli = Wina.dawyebisweli + 1;
```

შედეგი:

	dawyebisweli	Mimdraodshemk	Winaraodshemk	nazrdi
1	2000	2	NULL	NULL
2	2001	1	2	-1
3	2002	4	1	3
4	2003	1	4	-3
5	2004	2	1	1
6	2005	2	2	0
7	2006	2	2	0
8	2007	1	2	-1

როგორც ვხედავთ, weliRaodenoba (წლიური რაოდენობა) საერთო ცხრილური გამოსახულება გამოცხადებულია ერთხელ, ხოლო გარე მოთხოვნის FROM განყოფილებაში ორჯერ გამოიყენება: როგორც Mimd და როგორც Wina. ამ შემთხვევაში, ჩვენ გვაქვს საერთო ცხრილური გამოსახულების შემცველი მოთხოვნის მხოლოდ ერთი ასლი და არა რამდენიმე, როგორც წარმოებული ცხრილების შემთხვევაში.

APPLY ოპერაცია

APPLY არის ცხრილური ოპერაცია და მოთხოვნის FROM განყოფილებაში გამოიყენება. APPLY ოპერაციის ორი ტიპი არსებობს: CROSS APPLY და OUTER APPLY.

APPLY ოპერაცია ორი შესასვლელი ცხრილის (მარცხენა და მარჯვენა) მიმართ გამოიყენება, რომელთაგან მეორე (მარჯვენა) შეიძლება ცხრილური გამოსახულება იყოს. ჩვეულებრივ, მარჯვენა ცხრილი წარმოებული ცხრილია ან ჩამნაცვლებელი ცხრილური

ფუნქცია. CROSS APPLY ოპერაცია მოთხოვნის ლოგიკური დამუშავების ერთ სტადიას ასრულებს. ის მარჯვენა ცხრილურ გამოსახულებას „მიადგამს“ მარცხენა ცხრილის თითოეულ სტრიქონს და გაერთიანებული შედეგობრივი ნაკრებებიდან შედეგობრივ ცხრილს აფორმირებს.

CROSS APPLY ოპერაცია CROSS JOIN ოპერაციას წააგავს და რიგ შემთხვევაში შეიძლება ერთნაირი შედეგობრივი ნაკრებები გასცენ. მაგალითად, შემდეგი ორი მოთხოვნა ერთნაირ შედეგობრივ ნაკრებს გასცემს:

```
SELECT P.personaliID, S.shemkvetiID
FROM Personali AS P
CROSS JOIN Shemkveti AS S;

SELECT P.personaliID, S.shemkvetiID
FROM Personali AS P
CROSS APPLY Shemkveti AS S;
```

შედეგი:

	personaliID	shemkvetiID
1	1	1
2	1	2
3	1	3
4	1	4
5	1	5
6	1	6
7	1	7
8	1	8
9	1	9
10	1	10
11	1	11
12	1	12
13	1	13
14	1	14
15	1	15
16	1	16
17	1	17
18	2	1
19	2	2
20	2	3
21	2	4
22	2	5
23	2	6

მაგრამ, CROSS APPLY ოპერაციაში CROSS JOIN ოპერაციისგან განსხვავებით მარჯვენა ცხრილური გამოსახულება შეიძლება სტრიქონების სხვადასხვა ნაკრებს წარმოადგენდეს მარცხენა ცხრილის თითოეული სტრიქონისთვის. ამას შეიძლება მივაღწიოთ, თუ მარჯვნივ მივუთითებთ წარმოებულ ცხრილს და მოთხოვნაში, რომელიც წარმოებულ ცხრილს განსაზღვრავს, მივმართავთ მარცხენა ცხრილის სვეტებს.

მაგალითი. მოყვანილი პროგრამული კოდი CROSS APPLY ოპერაციას იყენებს თითოეული შემკვეთის ორი უკანასკნელი შეკვეთის დასაბრუნებლად:

```
SELECT S.shemkvetiID, A.xelshekrulebaID, A.tarigi_dawyebis
FROM Shemkveti AS S
CROSS APPLY
(
SELECT TOP (2) xelshekrulebaID, tarigi_dawyebis
FROM Xelshekruleba AS X
WHERE X.shemkvetiID = S.shemkvetiID
ORDER BY tarigi_dawyebis DESC, xelshekrulebaID DESC
) AS A;
```

შედეგი:

	shemkvetiID	xelshekrulebaID	tarigi_dawyebis
1	15	21	2013-02-01 00:00:00.000
2	1	14	2012-03-03 00:00:00.000
3	1	12	2012-03-03 00:00:00.000
4	2	3	2011-01-01 00:00:00.000
5	2	10	2009-05-07 00:00:00.000
6	3	11	2012-10-10 00:00:00.000
7	3	9	2012-10-10 00:00:00.000
8	4	8	2008-10-17 00:00:00.000
9	5	4	2009-08-20 00:00:00.000
10	5	13	2009-07-09 00:00:00.000
11	6	5	2007-02-01 00:00:00.000
12	7	15	2011-11-14 00:00:00.000
13	7	16	2010-05-21 00:00:00.000
14	12	18	2015-12-01 00:00:00.000
15	13	19	2010-07-10 00:00:00.000
16	14	20	2011-07-10 00:00:00.000

მარჯვენა ცხრილური გამოსახულება (ჩვენს შემთხვევაში, წარმოებული ცხრილი) შეეთანადება თითოეულ სტრიქონს Shemkveti ცხრილიდან. წარმოებული ცხრილი განსაზღვრავს ორ უკანასკნელ შეკვეთას შემკვეთისთვის მარცხენა ცხრილის მიმდინარე სტრიქონიდან. რადგან, წარმოებული ცხრილი გამოიყენება მარცხენა ცხრილის თითოეული სტრიქონის მიმართ, CROSS APPLY ოპერაცია გასცემს ორ უკანასკნელ შეკვეთას თითოეული შემკვეთისთვის.

თუ მარჯვენა ცხრილური გამოსახულება გასცემს ცარიელ ნაკრებს, მაშინ CROSS APPLY ოპერაცია არ გასცემს შესაბამის სტრიქონს მარცხენა ცხრილიდან. მაგალითად, 8, 9 და 10 იდენტიფიკატორის მქონე შემკვეთებისთვის წარმოებული ცხრილი - ცარიელი ნაკრებია. ამიტომ ეს შემკვეთები შედეგში არ ჩაირთვება. თუ გვინდა ამოვირჩიოთ მარცხენა ცხრილის სტრიქონები, რომლებისთვისაც მარჯვენა ცხრილური გამოსახულება გასცემს ცარიელ ნაკრებს, CROSS APPLY ოპერაციის მაგივრად უნდა მივუთითოთ OUTER APPLY ოპერაცია. OUTER APPLY ოპერაცია მოთხოვნის ლოგიკური დამუშავების ორ სტადიას ასრულებს. მეორე სტადია ახდენს მარცხენა ცხრილის იმ სტრიქონების განსაზღვრავს, რომლებისთვისაც მარჯვენა ცხრილური გამოსახულება ცარიელ ნაკრებს გასცემს და ამ სტრიქონებს შედეგობრივ ცხრილში ჩართავს.

მაგალითი. გვაინტერესებს თითოეული შემკვეთის ორი უკანასკნელი შეკვეთა და ის

შემკვეთები, რომლებმაც არც ერთი შეკვეთა არ გააფორმეს. მოთხოვნას აქვს სახე:

```
SELECT S.shemkvetiID, T1.xelshekrulebaID, T1.tarigi_dawyebis
FROM Shemkveti AS S
OUTER APPLY
(
SELECT TOP (2) xelshekrulebaID, personaliID, tarigi_dawyebis
FROM Xelshekruleba AS X
WHERE X.shemkvetiID = S.shemkvetiID
ORDER BY tarigi_dawyebis DESC, xelshekrulebaID DESC
) AS T1;
```

შედეგი:

	shemkvetiID	xelshekrulebaID	tarigi_dawyebis
1	15	21	2013-02-01 00:00:00.000
2	1	14	2012-03-03 00:00:00.000
3	1	12	2012-03-03 00:00:00.000
4	1	1	2011-01-01 00:00:00.000
5	2	3	2011-01-01 00:00:00.000
6	2	10	2009-05-07 00:00:00.000
7	2	2	2009-01-01 00:00:00.000
8	3	11	2012-10-10 00:00:00.000
9	3	9	2012-10-10 00:00:00.000
10	3	6	2012-02-25 00:00:00.000
11	4	8	2008-10-17 00:00:00.000
12	5	4	2009-08-20 00:00:00.000
13	5	13	2009-07-09 00:00:00.000
14	6	5	2007-02-01 00:00:00.000
15	7	15	2011-11-14 00:00:00.000
16	7	16	2010-05-21 00:00:00.000
17	8	NULL	NULL
18	9	NULL	NULL
19	10	NULL	NULL
20	12	18	2015-12-01 00:00:00.000
21	13	19	2010-07-10 00:00:00.000
22	14	20	2011-07-10 00:00:00.000

შედეგში ჩართულია 8, 9 და 10 იდენტიფიკატორის მქონე შემკვეთები, რომლებსაც არც ერთი შეკვეთა არ აქვთ.

ჩამნაცვლებელი ცხრილური ფუნქციები

ჩამნაცვლებელი ცხრილური ფუნქციების (Inline ტიპის ფუნქციები) გამოყენებისას, ადვილი იქნება პროგრამული კოდისთვის თვალყურის დევნება და მისი თანხლება. მაგალითად, მოყვანილი პროგრამული კოდი ქმნის ჩამნაცვლებელ ცხრილურ ფუნქციას Funqcia, რომელიც იღებს შესასვლელ პარამეტრებს: შემკვეთის იდენტიფიკატორი (@shemkvetiID) და რიცხვი (@n), და გასცემს @shemkvetiID შემკვეთის ყველაზე უკანასკნელ შეკვეთებს. პროგრამულ კოდს აქვს სახე:

```
USE Shekveta;
IF OBJECT_ID(N'dbo.Funqcia') IS NOT NULL
DROP FUNCTION dbo.Funqcia;
GO
CREATE FUNCTION dbo.Funqcia (@shemkvetiID AS INT, @n AS INT)
```

```
RETURNS TABLE
AS
RETURN
SELECT TOP (@n) xelshekrulebaID, personaliID, tarigi_dawyebis, tarigi_damtavrebis
FROM Xelshekruleba
WHERE shemkvetiID = @shemkvetiID
ORDER BY tarigi_dawyebis DESC, xelshekrulebaID DESC;
GO
```

ახლა შეგვიძლია წინა მაგალითში წარმოებული ცხრილის გამოყენება შევცვალოთ ფუნქციით:

```
SELECT T2.shemkvetiID, T2.firmis_dasaxeleba,
T1.xelshekrulebaID, T1.personaliID, T1.tarigi_dawyebis, T1.tarigi_damtavrebis
FROM Shemkveti AS T2
CROSS APPLY dbo.Funqcia(T2.shemkvetiID, 3) AS T1;
```

ეს პროგრამული კოდი ადვილად იკითხება, მაგრამ ფიზიკური დამუშავების თვალსაზრისით არაფერი არ შეიცვალა, რადგან ხდება ცხრილური გამოსახულებების გახსნა.

თავი 17. მონაცემების რეორგანიზება და დაჯგუფების ნაკრებები

ამ თავში განვიხილავთ მონაცემების რეორგანიზებისა და დაჯგუფების ნაკრებების დამუშავების საკითხებს. მონაცემების გაშლა ნიშნავს სტრიქონების გარდაქმნას სვეტებად. მონაცემების შეკვეცა ნიშნავს სვეტების გარდაქმნას სტრიქონებად. დამჯგუფებელი ნაკრები არის სვეტების ერთობლიობა, რომლის მიხედვით მონაცემების დაჯგუფება სრულდება.

მონაცემების გაშლა

მონაცემების გაშლა არის მონაცემების მობრუნების მეთოდი, როცა სტრიქონები სვეტებად გარდაიქმნება შემაჯამებელი მონაცემების შესაძლო გამოთვლით. ამის დემონსტრირებისთვის შევასრულოთ პროგრამული კოდი, რომელიც #Xelshekruleba_1 ცხრილს ქმნის და მას სტრიქონებით ავსებს. პროგრამულ კოდს აქვს სახე:

```
CREATE TABLE #Xelshekruleba
(
xelshekrulebaID     INT PRIMARY KEY IDENTITY(1,1),
personaliID         INT,
shemkvetiID         CHAR,
tarigi_dawyebis     DATETIME,
gadasaxdeli_l       FLOAT
);
INSERT INTO #Xelshekruleba VALUES
(2, 'A', '20140903', 350),
(3, 'A', '20140903', 100),
(1, 'B', '20140903', 500),
(2, 'A', '20140903', 300),
(2, 'B', '20140903', 340),
(1, 'C', '20140903', 760),
(3, 'A', '20140903', 500),
(1, 'C', '20140903', 600),
(3, 'C', '20140903', 110),
(2, 'B', '20140903', 320),
(3, 'D', '20140903', 200);

SELECT * FROM #Xelshekruleba;
```

შედეგი:

	xelshekrulebaID	personaliID	shemkvetiID	tarigi_dawyebis	gadasaxdeli_l
1	1	2	A	2014-09-03 00:00:00.000	350
2	2	3	A	2014-09-03 00:00:00.000	100
3	3	1	B	2014-09-03 00:00:00.000	500
4	4	2	A	2014-09-03 00:00:00.000	300
5	5	2	B	2014-09-03 00:00:00.000	340
6	6	1	C	2014-09-03 00:00:00.000	760
7	7	3	A	2014-09-03 00:00:00.000	500
8	8	1	C	2014-09-03 00:00:00.000	600
9	9	3	C	2014-09-03 00:00:00.000	110
10	10	2	B	2014-09-03 00:00:00.000	320
11	11	3	D	2014-09-03 00:00:00.000	200

მონაცემების გაშლის განხილვამდე ჯერ შევასრულოთ მოთხოვნა, რომელიც გასცემს შეკვეთების საერთო თანხებს თითოეული თანამშრომლისა და შემკვეთისთვის:

```
SELECT personaliID, shemkvetiID, SUM(gadasaxdeli_l) AS sumgadasaxdeli_l
FROM #Xelshekruleba
GROUP BY personaliID, shemkvetiID;
```

შედეგი:

	personaliID	shemkvetiID	sumgadasaxdeli_l
1	2	A	650
2	3	A	600
3	1	B	500
4	2	B	660
5	1	C	1360
6	3	C	110
7	3	D	200

დავუშვათ, გვინდა მიღებული შედეგის წარმოდგენა ცხრილი 17.1-ში მოყვანილი სახით.

ცხრილი 17.1. საერთო თანხის შეჯამებული წარმოდგენა.

personaliID	A	B	C	D
1	NULL	500	1360	NULL
2	650	660	NULL	NULL
3	600	NULL	110	200

ცხრილში 17.1 მოყვანილია მონაცემების აგრეგირებული შეჯამებული წარმოდგენა #Xelshekruleba ცხრილიდან თითოეული თანამშრომლისთვის (სტრიქონებში) და თითოეული შემკვეთისთვის (სვეტებში). ამ წარმოდგენის ფორმირების მეთოდს მონაცემების გაშლა ეწოდება.

მონაცემების გაშლის თითოეული მოთხოვნა მოიცავს ლოგიკური დამუშავების სამ სტადიას, რომელთაგან თითოეულს აქვს მასთან დაკავშირებული ელემენტები:

- დაჯგუფების სტადია, დაკავშირებული დამაჯგუფებელ ან სტრიქონულ ელემენტთან;
- გახსნის სტადია, დაკავშირებული გაშლასთან ან სტრიქონულ ელემენტთან;
- შედეგის მიღების სტადია, დაკავშირებული შედეგობრივ ელემენტთან და შეჯამების ან

დაგროვების ფუნქციასთან.

ჩვენს მაგალითში, შედეგობრივ ნაკრებში უნდა მოვახდინოთ ერთადერთი სტრიქონის ფორმირება თანამშრომლის თითოეული იდენტიფიკატორისთვის. ეს იმას ნიშნავს, რომ სტრიქონები Xelshekruleba ცხრილიდან უნდა დაჯგუფდეს personaliID სვეტის მიხედვით. შედეგად, დამაჯგუფებელი ელემენტი ამ შემთხვევაში personaliID სვეტი იქნება.

მონაცემების გაშლის პროცესი გათვლილია შედეგობრივი სვეტების ფორმირებაზე შემკვეთის თითოეული უნიკალური იდენტიფიკატორისთვის, რომლებშიც მოთავსებულია ჯამური თანხა მოცემული შემკვეთისთვის. ეს პროცესი შეგვიძლია განვიხილოთ, როგორც თანხების განაწილება შემკვეთების იდენტიფიკატორებს შორის. მონაცემების გაშლის ელემენტი ჩვენს შემთხვევაში არის personaliID სვეტი.

დაბოლოს, რადგან მონაცემების გაშლა მოიცავს დაჯგუფებას, ჩვენ მოგვიწევს მონაცემების შეჯამება შედეგობრივი მნიშვნელობების ფორმირებისთვის დამაჯგუფებელი და გამშლელი ელემენტების „გადაკვეთებზე“. ამისთვის, საჭიროა მააგრეგირებელი ფუნქციისა (SUM() ფუნქცია) და შედეგობრივი ელემენტის (gadasaxdeli_l სვეტი) მოცემა.

ამრიგად, მონაცემების გაშლა მოიცავს დაჯგუფებას, გაშლას და შედეგის მიღებას. მოცემულ შემთხვევაში, მონაცემებს ვაჯგუფებთ personaliID სვეტის მიხედვით, ვშლით ან ვანაწილებთ (თანხებს) shemkvetiID სვეტის მიხედვით და შედეგს ვიღებთ SUM(gadasaxdeli_l) გამოსახულების საშუალებით.

მონაცემების გაშლა ორი გზით შეიძლება შევასრულოთ: სტანდარტული SQL ენის საშუალებებისა და T-SQL ენის PIVOT ოპერაციის გამოყენებით.

მონაცემების გაშლა სტანდარტული SQL ენის საშულებით

სტანდარტული SQL ენის საშუალებით მონაცემების გაშლის დროს სრულდება სამივე სტადიის რეალიზება.

დაჯგუფების სტადია სრულდება GROUP BY განყოფილების გამოყენებით:

```
GROUP BY personaliID.
```

მონაცემების გაშლის სტადია რეალიზდება CASE ელემენტის შემცველი SELECT განყოფილებით თითოეული შედეგობრივი სვეტისთვის. ჩვენ წინასწარ უნდა ვიცოდეთ გასაშლელი ელემენტის მნიშვნელობა და უნდა შევქმნათ ცალკეული გამოსახულება თითოეული მათგანისთვის. რადგან, ჩვენს მაგალითში ოთხი შემკვეთის (A, B, C და D) შეკვეთების თანხები უნდა „გავანაწილოთ“, ამიტომ მოთხოვნაში ოთხი CASE გამოსახულება იქნება. A შემკვეთისთვის CASE გამოსახულებას ექნება სახე:

```
CASE
WHEN personaliID = 'A' THEN gadasaxdeli_l
END
```

ეს გამოსახულება დაგვიბრუნებს თანხას თითოეული სტრიქონიდან, მხოლოდ იმ შემთხვევაში, თუ მიმდინარე სტრიქონი წარმოადგენს A შემკვეთის შეკვეთას; საწინააღმდეგო შემთხვევაში, გამოსახულება გასცემს NULL მნიშვნელობას. თუ CASE გამოსახულებაში ELSE განშტოება არ არის მოცემული, მაშინ იგულისხმება ELSE NULL. ეს იმას ნიშნავს, რომ შედეგობრივ სვეტში A შემკვეთისთვის გამოჩნდება მხოლოდ მასთან დაკავშირებული შეკვეთების თანხები. სხვა შემთხვევაში სვეტის მნიშვნელობა იქნება NULL.

ბოლოს, სრულდება ჯამური მონაცემების მიღების სტადია აგრეგირების ფუნქციის გამოყენების გზით (ჩვენს შემთხვევაში SUM()) თითოეული CASE გამოსახულების შედეგის მიმართ. მაგალითად, ქვემოთ მოყვანილია გამოსახულება, რომელიც აფორმირებს შედეგობრივ სვეტს A შემკვეთისთვის:

```
SUM(CASE WHEN shemkvetiID = 'A' THEN gadasaxdeli_l END) AS A
```

ამოცანაზე დამოკიდებულებით, შეგვიძლია სხვა აგრეგირების ფუნქციის (MAX(), MIN(), COUNT(), AVG()) გამოყენება.

ქვემოთ მოყვანილია მოთხოვნა, რომელიც გასცემს შეკვეთების სრულ თანხას თითოეული თანამშრომლისთვის (სტრიქონებში) და შემკვეთისთვის (სვეტებში):

```
SELECT personaliID,
SUM(CASE WHEN shemkvetiID = 'A' THEN gadasaxdeli_l END) AS A,
SUM(CASE WHEN shemkvetiID = 'B' THEN gadasaxdeli_l END) AS B,
SUM(CASE WHEN shemkvetiID = 'C' THEN gadasaxdeli_l END) AS C,
SUM(CASE WHEN shemkvetiID = 'D' THEN gadasaxdeli_l END) AS D
FROM #Xelshekruleba
GROUP BY personaliID;
```

ეს მოთხოვნა დაგვიბრუნებს ცხრილში 17.1 მოყვანილ შედეგს:

	personaliID	A	B	C	D
1	1	NULL	500	1360	NULL
2	2	650	660	NULL	NULL
3	3	600	NULL	110	200

მონაცემების გაშლა T-SQL ენის PIVOT ოპერაციის გამოყენებით

PIVOT ოპერაცია გამოიყენება SELECT მოთხოვნის FROM განყოფილებაში. ის ამუშავებს საწყის ცხრილს ან ცხრილურ გამოსახულებას, გამოთვლის ჯამურ მნიშვნელობას და გასცემს შედეგობრივ ცხრილს. PIVOT ოპერაცია მოიცავს ლოგიკური დამუშავების ადრე აღწერილ სტადიებს (დაჯგუფება, გაშლა და შეჯამება) გაშლის იმავე ელემენტებით.

PIVOT ოპერაციის შემცველი მოთხოვნის სინტაქსია:

```
SELECT . . .
  FROM <საწყისი_ცხრილი_ან_ცხრილური_გამოსახულება>
    PIVOT (<აგრეგირების_ფუნქცია> (<შედეგობრივი_ელემენტი>)
      FOR <გასაშლელი_ელემენტი>
        IN (<შედეგობრივი_სვეტების_სია>) ) AS <შედეგობრივი_ცხრილის ფსევდონიმი>
. . . ;
```

PIVOT ცხრილური ოპერაციის ფრჩხილებში მოიცემა აგრეგირების ფუნქცია (SUM()), შედეგობრივი ელემენტი (gadasaxdeli_l), გასაშლელი ელემენტი (shemkvetiID) და შედეგობრივი სვეტების სია (A, B, C, D). PIVOT ოპერაციის მრგვალი ფრჩხილების შემდეგ ეთითება შედეგობრივი ცხრილის ფსევდონიმი.

მნიშვნელოვანია შევნიშნოთ, რომ PIVOT ცხრილურ ოპერაციაში აშკარად არ ეთითება მაჯგუფებელი ელემენტი. ამით გამოირიცხება მოთხოვნაში GROUP BY განყოფილების ჩასმის აუცილებლობა. PIVOT ოპერაცია ირიბად განსაზღვრავს მაჯგუფებელ ელემენტებს, როგორც ყველა სვეტს საწყისი ცხრილიდან (ან ცხრილური გამოსახულებიდან), რომლებიც არ მოიცემა როგორც არც გასაშლელი ელემენტი, არც შედეგობრივი. საწყისი ცხრილი PIVOT ოპერაციისათვის არ უნდა შეიცავდეს არანაირ სვეტს გარდა მაჯგუფებელი, გასაშლელი და შედეგობრივი ელემენტებისა. ამას შეგვიძლია მივაღწიოთ, თუ გამოვიყენებთ PIVOT ცხრილურ ოპერაციას უშუალოდ ცხრილური გამოსახულების მიმართ, რომელიც მხოლოდ საჭირო სვეტებს მოიცავს.

მაგალითი. PIVOT ოპერაციის შემცველ მოთხოვნას აქვს სახე:

```
SELECT personaliID, A, B, C, D
FROM
(
SELECT personaliID, shemkvetiID, gadasaxdeli_l
FROM #Xelshekruleba
) AS T1
PIVOT (SUM(gadasaxdeli_l) FOR shemkvetiID IN ( A, B, C, D ) ) AS T2;
```

ამ მოთხოვნის შესრულების წინ უნდა განვსაზღვროთ SQL სერვერთან მონაცემთა ბაზის თავსებადობის დონე, რისთვისაც ვასრულებთ შემდეგ ბრძანებას:

```
ALTER DATABASE Shekveta
SET COMPATIBILITY_LEVEL = 100;
```

მოთხოვნის შესრულების შედეგი:

	personaliID	A	B	C	D
1	1	NULL	500	1360	NULL
2	2	650	660	NULL	NULL
3	3	600	NULL	110	200

#Xelshekruleba ცხრილის უშუალოდ დამუშავების ნაცვლად PIVOT ოპერაცია გამოიყენება T1 წარმოებული ცხრილისადმი და შეიცავს მხოლოდ გაშლის ელემენტებს: personaliID, shemkvetiID და gadasaxdeli_l. shemkvetiID გამშლელი ელემენტისა და gadasaxdeli_l შედეგობრივი ელემენტის გარდა რჩება personaliID სვეტი, რომელიც განიხილება, როგორც მაჯგუფებელი ელემენტი.

ეს მოთხოვნა დაგვიბრუნებს ცხრილში 17.1 მოყვანილ შედეგს.

განვიხილოთ მონაცემების გაშლის კიდევ ერთი მაგალითი. დავუშვათ, სტრიქონებში თანამშრომლებისა და სვეტებში შემკვეთების დაბრუნების ნაცვლად გვინდა, რომ შევასრულოთ უკუ მოქმედება: მაჯგუფებელი ელემენტია - shemkvetiID, გასაშლელი ელემენტია - personaliID, ხოლო შემაჯამებელი ელემენტი და აგრეგირების ფუნქციაა SUM(gadasaxdeli_l). მოთხოვნას აქვს სახე:

```
SELECT shemkvetiID, [1], [2], [3]
FROM ( SELECT personaliID, shemkvetiID, gadasaxdeli_l
         FROM #Xelshekruleba ) AS T1
  PIVOT ( SUM(gadasaxdeli_l) FOR personaliID IN([1], [2], [3] ) ) AS T2;
```

შედეგი:

	shemkvetiID	1	2	3
1	A	NULL	650	600
2	B	500	660	NULL
3	C	1360	NULL	110
4	D	NULL	NULL	200

თანამშრომლების იდენტიფიკატორები: 1, 2 და 3 არის personaliID სვეტის მნიშვნელობები საწყის ცხრილში. მაგრამ, შედეგის თვალსაზრისით, ეს მნიშვნელობები ხდება შედეგობრივი სვეტების სახელები. შედეგად, PIVOT ოპერაციის IN ელემენტში მათ უნდა მივმართოთ როგორც იდენტიფიკატორებს. თუ იდენტიფიკატორები არასტანდარტულია (მაგალითად, ციფრით იწყება), აუცილებელია მათი შეზღუდვა, დავუშვათ, კვადრატული ფრჩხილებით.

მონაცემების შეკვეცა

მონაცემების შეკვეცა არის მონაცემების მობრუნების მეთოდი, როცა სვეტები სტრიქონებად გარდაიქმნება. ჩვეულებრივ, ის შეიცავს მოთხოვნას გაშლილი შემაჯამებელი მონაცემების მიმართ. ეს მოთხოვნა თითოეული საწყისი სტრიქონიდან რამდენიმე შედეგობრივ სტრიქონს აფორმირებს, რომლებიც საწყისი სვეტის სხვადასხვა მნიშვნელობებს შეიცავს. ეს იმას ნიშნავს, რომ შეჯამებული (გაშლილი) ცხრილის თითოეული საწყისი სტრიქონი შეიძლება სტრიქონების სიმრავლედ გარდაიქმნას - თითო სტრიქონი საწყისი სვეტის ყოველი მოცემული მნიშვნელობისთვის.

შევქმნათ #PersShemXelsh ცხრილი და ის #Xelshekruleba ცხრილის მონაცემებით შევავსოთ. ამისათვის შემდეგი პროგრამული კოდი შევასრულოთ:

```
SELECT personaliID, A, B, C, D
INTO #PersShemXelsh
FROM
(
SELECT personaliID, shemkvetiID, gadasaxdeli_l
FROM  #Xelshekruleba
) AS D
PIVOT (SUM(gadasaxdeli_l) FOR shemkvetiID IN(A, B, C, D) ) AS P;

SELECT * FROM #PersShemXelsh;
```

შედეგი:

	personaliID	A	B	C	D
1	1	NULL	500	1360	NULL
2	2	650	660	NULL	NULL
3	3	600	NULL	110	200

ცხრილში არის სტრიქონი თითოეული თანამშრომლისთვის, სვეტი ოთხი შემკვეთიდან (A, B, C, D) თითოეულისთვის, შეკვეთების თანხები თითოეული თანამშრომლისთვის და თითოეული შემკვეთისათვის. არარსებული შეკვეთები (გადაკვეთები) წარმოდგენილია NULL მნიშვნელობით.

მონაცემების შეკვეცა ორი გზით შეიძლება შევასრულოთ: SQL ენის სტანდარტული საშუალებებისა და T-SQL ენის UNPIVOT ოპერაციის გამოყენებით.

მონაცემების შეკვეცა სტანდარტული SQL ენის საშუალებით

მონაცემების შეკვეცის სტანდარტული გადაწყვეტა ლოგიკური დამუშავების სამი სტადიის რეალიზებას მოიცავს: ასლის ფორმირება, ელემენტების ამოღება და არარსებული გადაკვეთების გამორიცხვა.

პირველი სტადიაზე სრულდება თითოეული საწყისი სტრიქონის ასლების ფორმირება - თითო სტრიქონი ყოველი სვეტისთვის. ჩვენს შემთხვევაში, უნდა შევქმნათ ასლი თითოეული A, B, C და D სვეტისთვის, რომლებიც წარმოადგენენ შემკვეთების იდენტიფიკატორებს. ასლების ფორმირებისთვის გამოიყენება ჯვარედინი შეერთება. ჩვენ

ჯვარედინი შეერთება უნდა გამოვიყენოთ #PersShemXelsh ცხრილისა და იმ ცხრილის მიმართ, რომელიც შეიცავს სტრიქონს თითოეული შემკვეთისთვის.

შეგვიძლია გამოვიყენოთ მნიშვნელობების კონსტრუქტორი VALUES ელემენტის სახით ვირტუალური ცხრილის შესაქმნელად სტრიქონით თითოეული შემკვეთისთვის. მოთხოვნას, რომელიც ახდენს პირველი სტადიის რეალიზებას, აქვს სახე:

```
SELECT *
FROM #PersShemXelsh
CROSS JOIN
(
SELECT 'A' AS shemkvetiID
  UNION ALL SELECT 'B'
  UNION ALL SELECT 'C'
  UNION ALL SELECT 'D'
) AS Pers;
```

მოთხოვნის ეს სახე არ არის სტანდარტული, რადგან სტანდარტული სახე SELECT მოთხოვნაში FROM განყოფილების არსებობას მოითხოვს.

შედეგი:

	personaliID	A	B	C	D	shemkvetiID
1	1	NULL	500	1360	NULL	A
2	1	NULL	500	1360	NULL	B
3	1	NULL	500	1360	NULL	C
4	1	NULL	500	1360	NULL	D
5	2	650	660	NULL	NULL	A
6	2	650	660	NULL	NULL	B
7	2	650	660	NULL	NULL	C
8	2	650	660	NULL	NULL	D
9	3	600	NULL	110	200	A
10	3	600	NULL	110	200	B
11	3	600	NULL	110	200	C
12	3	600	NULL	110	200	D

როგორც ვხედავთ, თითოეული საწყისი სტრიქონისთვის ფორმირებულია ოთხი ასლი - თითო ასლი A, B, C და D შემკვეთებისთვის.

მეორე სტადიაზე სრულდება იმ სვეტის შექმნა (gadasaxdeli_l), რომელიც გასცემს მნიშვნელობას იმ სვეტიდან, რომელიც იმ შემკვეთს შეესაბამება, რომელიც სტრიქონის მიმდინარე ასლშია წარმოდგენილი. უფრო კონკრეტულად, თუ shemkvetiID-ის მიმდინარე მნიშვნელობა A-ს ტოლია, მაშინ gadasaxdeli_l სვეტში უნდა დავაბრუნოთ მნიშვნელობა A სვეტიდან; თუ shemkvetiID B-ს ტოლია, მაშინ gadasaxdeli_l უნდა შეიცავდეს მნიშვნელობას B სვეტიდან, და ა.შ. ამ სტადიის რეალიზება შეიძლება CASE გამოსახულების გამოყენებით:

```
SELECT personaliID, shemkvetiID,
  CASE shemkvetiID
        WHEN 'A' THEN A
        WHEN 'B' THEN B
        WHEN 'C' THEN C
        WHEN 'D' THEN D
  END AS gadasaxdeli_l
FROM #PersShemXelsh
```

CROSS JOIN (VALUES('A'), ('B'), ('C'), ('D')) AS Shems(shemkvetiID);

შედეგი:

	personaliID	shemkvetiID	gadasaxdeli_l
1	1	A	NULL
2	1	B	500
3	1	C	1360
4	1	D	NULL
5	2	A	650
6	2	B	660
7	2	C	NULL
8	2	D	NULL
9	3	A	600
10	3	B	NULL
11	3	C	110
12	3	D	200

გავიხსენოთ, რომ საწყის ცხრილში NULL მნიშვნელობები წარმოადგენენ არარსებულ შეკვეთებს (გადაკვეთებს). არარსებული გადაკვეთების გამოსარიცხად უნდა განვსაზღვროთ ცხრილური გამოსახულება იმ მოთხოვნის საფუძველზე, რომელიც მეორე სტადიის რეალიზებას ახდენს, და შემდეგ, გარე მოთხოვნაში NULL მნიშვნელობა გავფილტროთ. ქვემოთ მოთხოვნა მთლიანად არის მოყვანილი:

```
SELECT *
FROM (SELECT personaliID, shemkvetiID,
  CASE shemkvetiID
        WHEN 'A' THEN A
        WHEN 'B' THEN B
        WHEN 'C' THEN C
        WHEN 'D' THEN D
  END AS gadasaxdeli_l
FROM #PersShemXelsh
  CROSS JOIN (VALUES('A'), ('B'), ('C'), ('D')) AS Shems(shemkvetiID) ) AS T1
WHERE gadasaxdeli_l IS NOT NULL;
```

შედეგი:

	personaliID	shemkvetiID	gadasaxdeli_l
1	1	B	500
2	1	C	1360
3	2	A	650
4	2	B	660
5	3	A	600
6	3	C	110
7	3	D	200

შეკვეცა T-SQL-ის UNPIVOT ოპერაციის საშუალებით

მონაცემების შეკვეცა მოიცავს ორი შედეგობრივი სვეტის ფორმირებას საწყისი სვეტების ნებისმიერი რაოდენობიდან, რომლის შეკვეცასაც ვახდენთ. ჩვენს მაგალითში საჭიროა,

შევკვეცოთ საწყისი A, B, C და D სვეტები, ვქმნით რა ორ შედეგობრივ სვეტს shemkvetiID და gadasaxdeli_l სახელებით. პირველში მოთავსებული იქნება საწყისი სვეტების სახელები ('A', 'B', 'C' და 'D'). მეორეში კი - მნიშვნელობები საწყისი სვეტებიდან (შეკვეთების თანხები). UNPIVOT ოპერაციის შემცველი მოთხოვნის სინტაქსია:

```
SELECT . . .
FROM < საწყისი_ცხრილი_ან_ცხრილური_გამოსახულება >
  UNPIVOT ( < შედეგობრივი_სვეტი_საწყისი_სვეტების_მნიშვნელობების_შესანახად >
    FOR < შედეგობრივი_სვეტი_საწყისი_სვეტების_სახელების_შესანახად >
      IN ( < საწყისი_სვეტების_სია > ) ) AS < შედეგობრივი_ცხრილის_ფსევდონიმი >
. . . ;
```

ისევე როგორც PIVOT, UNPIVOT ოპერაცია რეალიზებულია როგორც ცხრილური ოპერაცია და გამოიყენება FROM განყოფილებაში. ის ამუშავებს საწყის ცხრილს ან ცხრილურ გამოსახულებას (ჩვენს შემთხვევაში #PersShemXelsh). UNPIVOT ოპერაციის მრგვალ ფრჩხილებში ეთითება სახელი, რომელიც ენიშნება იმ სვეტს, რომელშიც მოთავსებული იქნება საწყისი სვეტის (gadasaxdeli_l) მონაცემები: სახელი, რომელიც მიენიჭება იმ სვეტს, რომელშიც მოთავსებული იქნება საწყისი სვეტების სახელები (shemkvetiID), და საწყისი სვეტების სახელების სია (A, B, C და D). მრგვალი ფრჩხილების შემდეგ ეთითება იმ ცხრილის ფსევდონიმი, რომელიც მიიღება ცხრილური ოპერაციის შესრულების შედეგად.

ქვემოთ მოყვანილია მოთხოვნა, რომელიც ახდენს მონაცემების შეკვეცას:

```
SELECT  personaliID, shemkvetiID, gadasaxdeli_l
FROM #PersShemXelsh
  UNPIVOT(gadasaxdeli_l FOR shemkvetiID IN(A, B, C, D)) AS T1;
```

შედეგი:

	personaliID	shemkvetiID	gadasaxdeli_l
1	1	B	500
2	1	C	1360
3	2	A	650
4	2	B	660
5	3	A	600
6	3	C	110
7	3	D	200

UNPIVOT ოპერაცია ახდენს ადრე აღწერილი ლოგიკური დამუშავების ყველა სტადიის რეალიზებას, ასლების ფორმირებას, ელემენტების ამოღებას და NULL-ის ტოლი გადაკვეთების წაშლას.

უნდა გავითვალისწინოთ ასევე ის, რომ გაშლილი შემაჯამებელი ცხრილის შეკვეცა არ გვიბრუნებს საწყის ცხრილს. შეკვეცა ესაა გაშლილი მნიშვნელობების წარმოდგენა ახალი სახით. მაგრამ ცხრილი, რომელიც შეკვეცილი იყო, შეიძლება ისევ გავშალოთ, ვაბრუნებთ რა მის საწყის გაშლილ მდგომარეობას. სხვა სიტყვებით, შეჯამებული მნიშვნელობების მიღება საწყისი გაშლის პროცესში იწვევს დაწვრილებითი ინფორმაციის დაკარგვას.

დაჯგუფების ნაკრებები

დაჯგუფების ნაკრები ესაა სვეტების ერთობლიობა, რომელთა მიხედვით მონაცემების დაჯგუფება სრულდება. ტრადიციულად SQL ენაში ერთი მოთხოვნა, განკუთვნილი

შეჯამებული მონაცემების მისაღებად, დაჯგუფების ერთ ნაკრებს განსაზღვრავს. მაგალითად, მოყვანილი ოთხი მოთხოვნიდან თითოეულში განსაზღვრულია დაჯგუფების თითო ნაკრები:

```
SELECT personaliID, shemkvetiID, SUM(gadasaxdeli_l) AS sum_gadasaxdeli_l
FROM #Xelshekruleba
GROUP BY personaliID, shemkvetiID;

SELECT personaliID, SUM(gadasaxdeli_l) AS sum_gadasaxdeli_l
FROM #Xelshekruleba
GROUP BY personaliID;

SELECT shemkvetiID, SUM(gadasaxdeli_l) AS sum_gadasaxdeli_l
FROM #Xelshekruleba
GROUP BY shemkvetiID;

SELECT SUM(gadasaxdeli_l) AS sum_gadasaxdeli_l
FROM #Xelshekruleba;
```

შედეგი:

Results | Messages

	personaliID	shemkvetiID	sum_gadasaxdeli_l
1	2	A	650
2	3	A	600
3	1	B	500
4	2	B	660
5	1	C	1360
6	3	C	110
7	3	D	200

	personaliID	sum_gadasaxdeli_l
1	1	1860
2	2	1310
3	3	910

	shemkvetiID	sum_gadasaxdeli_l
1	A	1250
2	B	1160
3	C	1470
4	D	200

	sum_gadasaxdeli_l
1	4080

პირველ მოთხოვნაში განსაზღვრულია (personaliID, shemkvetiID) დაჯგუფების ნაკრები, მეორეში - (personaliID), მესამეში - (shemkvetiID), ხოლო მეოთხეში - დაჯგუფების ცარიელი ნაკრები (). შედეგად, გაიცემა ოთხი შედეგობრივი ნაკრები.

დავუშვათ, რომ ოთხი ცალკეული შედეგობრივი ნაკრების ნაცვლად გვინდა მივიღოთ ოთხივე ნაკრების მიხედვით დაჯგუფებული ერთი გაერთიანებული შედეგობრივი ნაკრები შეჯამებული მონაცემებით. ოთხივე მოთხოვნის შედეგობრივი ნაკრების გაერთიანებისთვის უნდა გამოვიყენებთ UNION ALL ოპერაცია. ამ შემთხვევაში შედეგობრივი ნაკრებები თავსებადი უნდა იყოს, ანუ უნდა ჰქონდეს ერთნაირი რაოდენობისა და ტიპის მქონე სვეტები.

ამის მისაღწევად, აუცილებელია მოთხოვნების შეთანხმება, გამოტოვებული სვეტების მაგივრად შემვსებების (მაგალითად, NULL მნიშვნელობა) დამატების გზით. ქვემოთ მოყვანილია შესაბამისი პროგრამული კოდი:

```
SELECT personaliID, shemkvetiID, SUM(gadasaxdeli_l) AS sum_gadasaxdeli_l
FROM #Xelshekruleba
GROUP BY personaliID, shemkvetiID

UNION ALL

SELECT personaliID, NULL, SUM(gadasaxdeli_l) AS sum_gadasaxdeli_l
FROM #Xelshekruleba
GROUP BY personaliID

UNION ALL

SELECT NULL, shemkvetiID, SUM(gadasaxdeli_l) AS sum_gadasaxdeli_l
FROM #Xelshekruleba
GROUP BY shemkvetiID

UNION ALL

SELECT NULL, NULL, SUM(gadasaxdeli_l) AS sum_gadasaxdeli_l
FROM #Xelshekruleba;
```

ეს პროგრამული კოდი დაჯგუფების ოთხივე გაერთიანებული ნაკრების მიხედვით აფორმირებს ერთ შედეგობრივ ნაკრებს შემაჯამებელი ინფორმაციით:

	personaliID	shemkvetiID	sum_gadasaxdeli_l
1	2	A	650
2	3	A	600
3	1	B	500
4	2	B	660
5	1	C	1360
6	3	C	110
7	3	D	200
8	1	NULL	1860
9	2	NULL	1310
10	3	NULL	910
11	NULL	A	1250
12	NULL	B	1160
13	NULL	C	1470
14	NULL	D	200
15	NULL	NULL	4080

ასეთ მიდგომას ორი ნაკლი აქვს: პროგრამული კოდის მოცულობა და მწარმოებლურობა. კერძოდ, დაჯგუფების ნაკრებების დიდი რაოდენობის შემთხვევაში მოთხოვნა შეიძლება ძალიან გრძელი გამოვიდეს. გარდა ამისა, SQL სერვერი საწყის ცხრილს თითოეული მოთხოვნისთვის ცალკე განიხილავს, რაც არაეფექტურია.

GROUPING SETS ჩადგმული ელემენტი

GROUP BY განყოფილების GROUPING SETS ჩადგმულ ელემენტს აქვს მდიდარი ფუნქციური შესაძლებლობები და უმეტესად ანგარიშების შექმნისას და მონაცემთა საცავებში გამოიყენება. ამ ჩადგმული ელემენტის გამოყენებით შესაძლებელია ერთ მოთხოვნაში დაჯგუფების რამდენიმე ნაკრები განისაზღვროს. GROUPING SETS ჩადგმული ელემენტის მრგვალ ფრჩხილებში ვუთითებთ მძიმეებით გამოყოფილ დაჯგუფების ნაკრებებს. დაჯგუფების თითოეული ნაკრები შეიცავს მრგვალ ფრჩხილებში მოთავსებულ და მძიმეებით გამოყოფილ სვეტებს. მაგალითად, მოყვანილ მოთხოვნაში განსაზღვრულია დაჯგუფების ოთხი ნაკრები (personaliID, shemkvetiID), (personaliID), (shemkvetiID) და ():

```
SELECT personaliID, shemkvetiID, SUM(gadasaxdeli_l) AS sum_gadasaxdeli_l
FROM #Xelshekruleba
GROUP BY
  GROUPING SETS
(
(personaliID, shemkvetiID),
(personaliID),
(shemkvetiID),
()
) ;
```

შედეგი:

	personaliID	shemkvetiID	sum_gadasaxdeli_l
1	2	A	650
2	3	A	600
3	NULL	A	1250
4	1	B	500
5	2	B	660
6	NULL	B	1160
7	1	C	1360
8	3	C	110
9	NULL	C	1470
10	3	D	200
11	NULL	D	200
12	NULL	NULL	4080
13	1	NULL	1860
14	2	NULL	1310
15	3	NULL	910

ეს მოთხოვნა წინა მოთხოვნის ლოგიკური ეკვივალენტია. მოცემულ მოთხოვნას წინასთან შედარებით ორი უპირატესობა აქვს: პირველი, მისი პროგრამული კოდი გაცილებით მოკლეა, და მეორე, SQL სერვერი ახდენს საწყისი ცხრილის ნახვის რაოდენობის ოპტიმიზებას და მას განიხილავს ცალ-ცალკე დაჯგუფების თითოეული ნაკრებისთვის.

CUBE ჩადგმული ელემენტი

GROUP BY განყოფილების CUBE ჩადგმული ელემენტი დაჯგუფების რამდენიმე ნაკრების განსაზღვრის შემოკლებულ საშუალებას წარმოადგენს. CUBE ჩადგმული ელემენტის მრგვალ ფჩხილებში ეთითება მძიმეებით გამოყოფილი სვეტების სია, და მის საფუძველზე ყველა შესაძლო დაჯგუფების ნაკრები გაიცემა. მაგალითად, CUBE(a, b, c) არის GROUPING SETS((a, b, c), (a, b), (a, c), (b, c), (a), (b), (c), ())-ის ეკვივალენტური.

GROUPING SETS ჩადგმული ელემენტის ნაცვლად, რომელიც წინა მოთხოვნაში გამოიყენებოდა დაჯგუფების ოთხი ნაკრების განსაზღვრისთვის, შეგვიძლია CUBE(personaliID, shemkvetiID) ელემენტი გამოვიყენოთ. CUBE ელემენტის გამოყენებით მოთხოვნას ექნება სახე:

```
SELECT personaliID, shemkvetiID, SUM(gadasaxdeli_l) AS sum_gadasaxdeli_l
FROM #Xelshekruleba
GROUP BY CUBE(personaliID, shemkvetiID) ;
```

შედეგი:

	personaliID	shemkvetiID	sum_gadasaxdeli_l
1	2	A	650
2	3	A	600
3	NULL	A	1250
4	1	B	500
5	2	B	660
6	NULL	B	1160
7	1	C	1360
8	3	C	110
9	NULL	C	1470
10	3	D	200
11	NULL	D	200
12	NULL	NULL	4080
13	1	NULL	1860
14	2	NULL	1310
15	3	NULL	910

ROLLUP ჩადგმული ელემენტი

ROLLUP ჩადგმული ელემენტი, რომელიც შედის GROUP BY განყოფილების შემადგენლობაში, ასევე წარმოგვიდგენს დაჯგუფების რამდენიმე ნაკრების განსაზღვრის შემოკლებულ საშუალებას. ROLLUP ელემენტი გულისხმობს იერარქიის არსებობას შესასვლელ სვეტებში და დაჯგუფების იმ ნაკრებებს ქმნის, რომლებსაც აზრი იერარქიის გათვალისწინებით აქვს. მაგალითად, ROLLUP(a, b, c) გულისხმობს a > b > c იერარქიის არსებობას, და ქმნის დაჯგუფების ოთხ ნაკრებს. ROLLUP(a, b, c) არის GROUPING SETS((a, b, c), (a, b), (a), ())-ის ეკვივალენტური.

დავუშვათ, გვაინტერესებს საერთო თანხა დაჯგუფების თითოეული ნაკრებისთვის, რომლებიც შეიძლება განსაზღვრული იყოს შემდეგი დროითი იერარქიის საფუძველზე: *შეკვეთის წელი* > *შეკვეთის თვე* > *შეკვეთის დღე*. შეგვიძლია გამოვიყენოთ GROUPING SETS ჩადგმული ელემენტი და აშკარად ჩამოვთვალოთ ოთხივე შესაძლო დაჯგუფის ნაკრები:

```
GROUPING SETS (
(YEAR(tarigi_dawyebis), MONTH(tarigi_dawyebis), DAY(tarigi_dawyebis)),
(YEAR(tarigi_dawyebis), MONTH(tarigi_dawyebis)),
(YEAR(tarigi_dawyebis)),
```

())

იმავე, ჩადგმული ROLLUP ელემენტის გამოყენებით უფრო მოკლედ ჩაიწერება:

ROLLUP (YEAR(tarigi_dawyebis), MONTH(tarigi_dawyebis), DAY(tarigi_dawyebis))

მთლიან მოთხოვნას აქვს სახე:

```
SELECT
  YEAR(tarigi_dawyebis) AS xelshekruleba_year,
  MONTH(tarigi_dawyebis) AS xelshekruleba_month,
  DAY(tarigi_dawyebis) AS xelshekruleba_day,
  SUM(gadasaxdeli_l) AS sum_gadasaxdeli_L
FROM #Xelshekruleba
GROUP BY ROLLUP (YEAR(tarigi_dawyebis), MONTH(tarigi_dawyebis), DAY(tarigi_dawyebis)) ;
```

შედეგი:

	xelshekruleba_year	xelshekruleba_month	xelshekruleba_day	sum_gadasaxdeli_L
1	2014	9	3	4080
2	2014	9	NULL	4080
3	2014	NULL	NULL	4080
4	NULL	NULL	NULL	4080

GROUPING() და GROUPING_ID() ფუნქციები

დავუშვათ, გვაქვს მოთხოვნა, რომელიც დაჯგუფების რამდენიმე ნაკრებს განსაზღვრავს და გვინდა დაჯგუფების ნაკრებებთან შედეგობრივი სტრიქონების დაკავშირება. ე.ი. თითოეული შედეგობრივი სტრიქონისთვის უნდა განისაზღვროს დაჯგუფების რომელ ნაკრებთან არის ის დაკავშირებული. თუ დაჯგუფების სვეტები განსაზღვრულია როგორც NOT NULL (არ უშვებენ NULL მნიშვნელობას), ამის გაკეთება ადვილია. მაგალითად, განვიხილოთ შემდეგი მოთხოვნა:

```
SELECT personaliID, shemkvetiID, SUM(gadasaxdeli_l) AS sum_gadasaxdeli_l
FROM #Xelshekruleba
GROUP BY CUBE (personaliID, shemkvetiID) ;
```

შედეგი:

	personaliID	shemkvetiID	sum_gadasaxdeli_l
1	2	A	650
2	3	A	600
3	NULL	A	1250
4	1	B	500
5	2	B	660
6	NULL	B	1160
7	1	C	1360
8	3	C	110
9	NULL	C	1470
10	3	D	200
11	NULL	D	200
12	NULL	NULL	4080
13	1	NULL	1860
14	2	NULL	1310
15	3	NULL	910

რადგან personaliID და shemkvetiID სვეტები განსაზღვრულია #Xelshekruleba ცხრილში როგორც NOT NULL, ამიტომ NULL მნიშვნელობა ამ სვეტებში შეიძლება იყოს მხოლოდ შემვსები, რომელიც იმაზე მიუთითებს, რომ სვეტი არ მონაწილეობს მიმდინარე დაჯგუფების ნაკრებში. ამრიგად, ყველა სტრიქონი, რომლებშიც personaliID არ არის NULL-ის ტოლი და shemkvetiID არ უდრის NULL-ს, დაკავშირებულია (personaliID, shemkvetiID) დაჯგუფების ნაკრებთან. ყველა სტრიქონი, რომლებშიც personaliID არ უდრის NULL-ს და shemkvetiID უდრის NULL-ს, დაკავშირებულია (personaliID) დაჯგუფების ნაკრებთან და ა.შ. ზოგიერთი შემმუშავებელი NULL მნიშვნელობას ცვლის “ALL” მნიშვნელობით ან მსგავსი აღნიშვნით, თუ საწყის სვეტებში NULL მნიშვნელობა არ არის დაშვებული. ეს გვეხმარება შედეგების გამოტანისას.

მაგრამ, თუ დასაჯგუფებელი სვეტი უშვებს NULL მნიშვნელობას, მაშინ დანამდვილებით ვერ ვიტყვით NULL მნიშვნელობები შედეგობრივ ნაკრებში მონაცემებიდანაა მიღებული თუ გამოიყენება შემვსებებად იმ სვეტისთვის, რომელიც არ შედის დაჯგუფების ნაკრებში. ერთ-ერთი საშულება ცალსახად განვსაზღროთ დაჯგუფების ნაკრებთან კავშირი, მაშინაც კი როცა დასაჯგუფებელ სვეტებში დაშვებულია NULL მნიშვნელობები, არის GROUPING() ფუნქციის გამოყენება. ეს ფუნქცია იღებს სვეტის სახელს და გასცემს 0-ს, თუ ის არის მიმდინარე დაჯგუფების ნაკრების წევრი, და 1-ს საწინააღმდეგო შემთხვევაში.

მაგალითი. მოცემულ მოთხოვნაში GROUPING() ფუნქცია სრულდება დაჯგუფების თითოეული სვეტისთვის:

```
SELECT
  GROUPING (personaliID) AS grppersonaliID,
  GROUPING (shemkvetiID) AS grpshemkvetiID,
personaliID, shemkvetiID, SUM(gadasaxdeli_l) AS sum_gadasaxdeli_l
FROM #Xelshekruleba
GROUP BY CUBE (personaliID, shemkvetiID) ;
```

შედეგი:

	grppersonaliID	grpshemkvetiID	personaliID	shemkvetiID	sum_gadasaxdeli_l
1	0	0	2	A	650
2	0	0	3	A	600
3	1	0	NULL	A	1250
4	0	0	1	B	500
5	0	0	2	B	660
6	1	0	NULL	B	1160
7	0	0	1	C	1360
8	0	0	3	C	110
9	1	0	NULL	C	1470
10	0	0	3	D	200
11	1	0	NULL	D	200
12	1	1	NULL	NULL	4080
13	0	1	1	NULL	1860
14	0	1	2	NULL	1310
15	0	1	3	NULL	910

ახლა ჩვენ აღარ გვჭირდება NULL მნიშვნელობაზე დაყრდნობა იმისათვის, რომ კავშირი დავამყაროთ შედეგობრივ სტრიქონებსა და დაჯგუფების ნაკრებებს შორის. მაგალითად, ყველა სტრიქონი, რომლებშიც personaliID = 0 და shemkvetiID = 0, დაკავშირებულია (personaliID,

shemkvetiID) დაჯგუფების ნაკრებთან. ყველა სტრიქონი, რომლებშიც personaliID = 0, ხოლო shemkvetiID = 1, დაკავშირებულია (personaliID) დაჯგუფების ნაკრებთან და ა.შ.

GROUPING_ID() ფუნქცია კიდევ უფრო ამარტივებს შედეგობრივი სტრიქონებისა და დაჯგუფების ნაკრებების დაკავშირების პროცესს. ჩვენ ფუნქციის შესასვლელ პარამეტრებად ვუთითებთ ყველა ელემენტს, რომელიც მონაწილეობს ნებისმიერ დაჯგუფების ნაკრებში, მაგალითად, GROUPING_ID(a, b, c, d). ფუნქცია დაგვიბრუნებს ორობითი ასახვის მთელრიცხვა მნიშვნელობას, რომელშიც თითოეული ბიტი ერთ-ერთ შესასვლელ ელემენტს წარმოადგენს. უკიდურესი მარჯვენა ბიტი წარმოადგენს უკიდურეს მარჯვენა ელემენტს სიაში. მაგალითად, (a, b, c, d) დაჯგუფების ნაკრები წარმოგვიდგება მთელი რიცხვით 0 (0x8 + 0x4 + 0x2 + 0x1). (a, c) დაჯგუფების ნაკრები წარმოდგება მთელი რიცხვით 5 (0x8 + 1x4 + 0x2 + 1x1) და ა.შ.

დაჯგუფების თითოეული ელემენტისთვის GROUPING ფუნქციის გამოძახების ნაცვლად, როგორც ამას წინა მოთხოვნაში ვაკეთებდით, შეგვიძლია ერთხელ გამოვიძახოთ GROUPING_ID() ფუნქცია და შესასვლელ პარამეტრებად დაჯგუფების ყველა ელემენტი გადავცეთ. მოთხოვნას ექნება სახე:

```
SELECT
  GROUPING_ID (personaliID, shemkvetiID) AS groupingset,
  personaliID, shemkvetiID, SUM(gadasaxdeli_l) AS sum_gadasaxdeli_l
FROM #Xelshekruleba
GROUP BY CUBE (personaliID, shemkvetiID) ;
```

შედეგი:

	groupingset	personaliID	shemkvetiID	sum_gadasaxdeli_l
1	0	2	A	650
2	0	3	A	600
3	2	NULL	A	1250
4	0	1	B	500
5	0	2	B	660
6	2	NULL	B	1160
7	0	1	C	1360
8	0	3	C	110
9	2	NULL	C	1470
10	0	3	D	200
11	2	NULL	D	200
12	3	NULL	NULL	4080
13	1	1	NULL	1860
14	1	2	NULL	1310
15	1	3	NULL	910

ახლა ადვილად შეგვიძლია განვსაზღვროთ თუ დაჯგუფების რომელ ნაკრებთან არის დაკავშირებული თითოეული სტრიქონი. მთელი რიცხვი 0 (ორობითი 00) წარმოადგენს (personaliID, shemkvetiID) დაჯგუფების ნაკრებს; მთელი რიცხვი 1 (ორობითი 01) წარმოადგენს (personaliID) დაჯგუფების ნაკრებს; მთელი რიცხვი 2 (ორობითი 10) წარმოადგენს (shemkvetiID) დაჯგუფების ნაკრებს; მთელი რიცხვი 3 (ორობითი 11) წარმოადგენს () ცარიელ დაჯგუფების ნაკრებს.

თავი 18. მონაცემთა ბაზების სარეზერვო ასლები

სარეზერვო ასლი წარმოადგენს ერთ ან მეტ ფაილს, რომელშიც მთლიანად ან ნაწილობრივ მონაცემთა ბაზაა მოთავსებული. სარეზერვო ასლის შექმნა საჭიროა მონაცემების აღსადგენად კომპიუტერის, დისკის ან მონაცემების დაზიანების შემთხვევაში და ა.შ. სარეზერვო ასლის შესანახად გამოიყენება ინფორმაციის მატარებლები, როგორიცაა: მაგნიტური ლენტები, მაგნიტური დისკები, ZIP დისკები და ა.შ.

მონაცემთა ბაზების სარეზერვო ასლები უნდა შეიქმნას გარკვეული პერიოდულობით, მაგალითად, ყოველდღე, კვირაში ერთხელ ან ორჯერ და ა.შ. საუკეთესო ვარიანტია სარეზერვო ასლის შექმნა დროის რეალურ მასშტაბში.

სარეზერვო ასლის შექმნა იკავებს ოპერაციული სისტემის რესურსებს. ამიტომ, მისი შექმნა უმჯობესია სისტემის ნაკლებად დატვირთვის პერიოდში.

სარეზერვო ასლის ტიპები

არსებობს სარეზერვო ასლის ოთხი ტიპი:

1. მონაცემთა ბაზის სრული სარეზერვო ასლი (Database Backup);
2. ტრანზაქციების ჟურნალის სარეზერვო ასლი (Transaction log Backup);
3. დიფერენცირებული (დანაწევრებული) სარეზერვო ასლი (Differential Database Backup);
4. ფაილებისა და ფაილების ჯგუფის სარეზერვო ასლი (File and Filegroup Backup).

ნებისმიერი ტიპის სარეზერვო ასლი ყოველთვის იქმნება ერთი მონაცემთა ბაზისთვის და წარმოადგენს ერთ ფაილს. სარეზერვო ასლის ფაილი ოპერაციული სისტემის ჩვეულებრივი ფაილია, რომელიც შეგვიძლია სხვა დისკზე ან კატალოგში გადავწეროთ, აგრეთვე, სხვა სერვერზე გადავიტანოთ და იქ აღვადგინოთ. სარეზერვო ასლი შეიძლება აღვადგინოთ იმავე ან სხვა სახელით. მონაცემთა ბაზის სარეზერვო ასლის შექმნის დროს სერვერი ასრულებს, აგრეთვე, მის არქივირებას. მიღებული სარეზერვო ასლი შეგვიძლია დამატებით შევკუმშოთ რომელიმე არქივატორის გამოყენებით.

მონაცემთა ბაზის სარეზერვო ასლის ტიპის არჩევის დროს უნდა გავითვალისწინოთ შემდეგი ფაქტორები:

– რაც უფრო დიდია მონაცემთა ბაზის ზომა, მით მეტ დროს დაიკავებს მისი სარეზერვო ასლის შექმნის პროცესი.

– რაც უფრო ინტენსიურად სრულდება ცვლილებები მონაცემთა ბაზაში, მით უფრო ხშირად უნდა შევასრულოთ სარეზერვო ასლის შექმნა. ტრანზაქციების ოპერატიული დამუშავების სისტემებში (Online Transaction Processing, OLTP) მონაცემების ცვლილებები უწყვეტად სრულდება, ამიტომ სარეზერვო ასლი, პრაქტიკულად, საათში ერთხელ უნდა შეიქმნას.

– თუ მონაცემთა ბაზა გამოიყენება მხოლოდ წაკითხვის რეჟიმში, მაშინ საკმარისია ერთი სარეზერვო ასლის შექმნა.

დამგროვებლის არჩევა

სარეზერვო ასლების შექმნის დროს ერთ-ერთი მთავარი საკითხია დამგროვებლის არჩევა, რომელზეც სარეზერვო ასლი შეინახება. დამგროვებლის არჩევისას შემდეგი მოსაზრებები უნდა გავითვალისწინოთ:

– ერთ ფიზიკურ დისკზე არ უნდა იყოს მოთავსებული მონაცემები და მათი სარეზერვო

ასლი, რადგან დისკის გაფუჭების შემთხვევაში ორივე დაიკარგება.

– ინფორმაციის გარე მატარებლებთან (ZIP დისკები, მაგნიტური ლენტები და ა.შ.) მუშაობის დროს უნდა გვახსოვდეს, რომ მათი ექსპლუატირების ვადა შეზღუდულია.

– იმაში დასარწმუნებლად, რომ მონაცემები არ იქნება შენახული დაზიანებულ უბნებში, მათი გამოყენების წინ საჭიროა დამგროვებლის დაფორმატება ან შემოწმება სპეციალური პროგრამებით. მაგალითად, დისკებისთვის შეგვიძლია გამოვიყენოთ CheckDisk პროგრამა.

– უმჯობესია სარეზერვო ასლი მიმდევრობით ჩავწეროთ რამდენიმე დამგროვებელზე. მაგალითად, თუ გვაქვს შვიდი დისკი, მაშინ მათზე კვირის ყოველ დღეს რიგრიგობით შეგვიძლია ჩავწეროთ სარეზერვო ასლები.

სარეზერვო ასლების შესაქმნელად შეგვიძლია გამოვიყენოთ შემდეგი მოწყობილობები:

– *დამგროვებლები მაგნიტურ ლენტებზე* (*სტრიმერები*). ამ შემთხვევაში, ინფორმაციის მატარებელია კასეტაში მოთავსებული მაგნიტური ლენტი. მასზე ჩაწერასა და წაკითხვას ახდენს მოწყობილობა, რომელსაც სტრიმერი ეწოდება. ერთ კასეტაზე შესაძლებელია რამდენიმე არქივის შენახვა. თუ მონაცემთა ბაზა ძალიან დიდია, მაშინ მისი ასლი შეგვიძლია რამდენიმე კასეტაზე მოვათავსოთ. იმისათვის, რომ სერვერმა მაგნიტურ ლენტზე შეინახოს მონაცემები, ფაილისკენ გზის ნაცვლად უნდა მივუთითოთ ლენტის სახელი, მაგალითად, '\\.\mytape'.

– *დისკური მოწყობილობები*. სერვერს სარეზერვო ასლის ჩაწერა შეუძლია ნებისმიერ მატარებელზე, რომელიც ოპერაციულ სისტემაში წარმოდგენილია ლოგიკური დისკის სახით.

– *ქსელური რესურსები*. თუ ქსელში რამდენიმე სერვერია ჩართული, მაშინ სარეზერვო ასლის მატარებლად შეგვიძლია ქსელური დისკი გამოვიყენოთ. თითოეული სერვერი დამოუკიდებლად ქმნის მონაცემთა ბაზის არქივს, რომელიც ცენტრალური სერვერის დისკზე განლაგდება.

მოწყობილობის განსაზღვრა სარეზერვო ასლის მოსათავსებლად

სარეზერვო ასლის შექმნისათვის სერვერზე შექმნილი ლოგიკური მოწყობილობების სია ინახება master მონაცემთა ბაზის sysdevices სისტემურ წარმოდგენაში. ახალი მოწყობილობის დასამატებლად sp_addumpdevice სისტემური შენახული პროცედურა გამოიყენება. მისი სინტაქსია:

sp_addumpdevice [@devtype =] '*მოწყობილობის_ტიპი*' ,
[@logicalname =] '*ლოგიკური_სახელი*' ,
[@physicalname =] '*ფიზიკური_სახელი*'

განვიხილოთ არგუმენტების დანიშნულება.

– '*მოწყობილობის_ტიპი*'. მოწყობილობის ტიპია და იღებს შემდეგ მნიშვნელობებს: TAPE (ლენტი) და DISK (დისკი);

– '*ლოგიკური_სახელი*'. მოწყობილობის ლოგიკური სახელია;

– '*ფიზიკური_სახელი*'. მოწყობილობის ფიზიკური სახელია.

მაგალითები.

ა. მაგალითში ხდება ChemiDiski სახელის მქონე დისკური მოწყობილობის შექმნა, რომლის ფიზიკური სახელია C:\dump\Asli_1.bak:

```
USE master;
EXEC sp_addumpdevice 'DISK', 'ChemiDiski',
                     'C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Backup\Asli_1.bak';
```

ბ. მაგალითში ხდება QseluriDiski სახელის მქონე დაშორებული (ქსელური) დისკური მოწყობილობის შექმნა, რომლის ფიზიკური სახელის სინტაქსია

\\servername\sharename\path\filename.ext :

```
USE master;
EXEC sp_addumpdevice 'DISK', 'QseluriDiski', '\\Serveri_2\Romani\C:\ChemiKatalogi\Asli_2.bak':
```

გ. მაგალითში ხდება Lenti_1 სახელის მქონე ლენტური მოწყობილობის შექმნა, რომლის ფიზიკური სახელია \\.\tape0:

```
USE master;
EXEC sp_addumpdevice 'TAPE', 'Lenti_1',  '\\.\tape0';
```

ლოგიკური მოწყობილობის წასაშლელად გამოიყენება sp_dropdevice სისტემური შენახული პროცედურა. მისი სინტაქსია:

sp_dropdevice [@logicalname =] '*მოწყობილობის_ლოგიკური_სახელი*'
[, [@delfile =] 'DELFILE']

'DELFILE' მიუთითებს, რომ ფიზიკური ფაილი უნდა წაიშალოს.

მაგალითი.

ა. მაგალითში ხდება Lenti_1 ლენტური მოწყობილობის წაშლა სერვერიდან:

```
sp_dropdevice 'Lenti_1';
```

ბ. მაგალითში ხდება Lenti_1 ლენტური მოწყობილობისა და მასთან დაკავშირებული ფაილის წაშლა სერვერიდან:

```
sp_dropdevice 'Lenti_1', 'DELFILE';
```

ლოგიკური მოწყობილობების შესახებ ინფორმაციის მისაღებად შეგვიძლია გამოვიყენოთ sp_helpdevice სისტემური შენახული პროცედურა ან შევასრულოთ მოთხოვნა:

```
SELECT * FROM sysdevices;
```

მაგალითი.

```
sp_helpdevice 'Lenti_1';
```

	device_name	physical_name	description	status	cntrltype	size
1	ChemiDiski	c:\dump\Asli_1.bak	disk, backup device	16	2	0

მონაცემთა ბაზის სრული და დიფერენცირებული ასლები

მონაცემთა ბაზის სრული ასლი

მონაცემთა ბაზის სრული სარეზერვო ასლის შექმნა გულისხმობს მთელი მონაცემთა ბაზის ასლის შექმნას. ასეთ მიდგომას როგორც დადებითი, ისე უარყოფითი მხარეები აქვს. დადებითი მხარეა ის, რომ საკმარისია მხოლოდ ერთი არქივის აღდგენა, რადგან მთელი ინფორმაცია ერთ ფაილშია მოთავსებული. სრული სარეზერვო ასლი უნდა შევქმნათ მაშინ, როცა მონაცემთა ბაზა იშვიათად იცვლება და მონაცემების დაკარგვას დიდი მნიშვნელობა არ ექნება. უმჯობესია ეს პროცესი კვირაში ერთხელ შევასრულოთ, თუმცა ყველაფერი სისტემის აქტიურობაზეა დამოკიდებული. ასეთი მიდგომის ნაკლია არქივის შექმნის ხანგრძლივი დრო, მასში მცირეოდენი ცვლილებების შეტანის შემთხვევაშიც კი.

გარდა ამისა, თუ ასლს შევქმნით კვირაში ერთხელ, დავუშვათ ორშაბათობით და მონაცემთა ბაზა დაზიანდა შაბათს, მაშინ ერთი კვირის განმავლობაში შესრულებული ცვლილებები დაიკარგება. ამიტომ, მონაცემთა ბაზის სრული სარეზერვო ასლთან ერთად უნდა შევქმნათ სარეზერვო ასლის სხვა ტიპებიც.

სარეზერვო ასლის შექმნის დროს მნიშვნელოვანია ცვლილებებისთვის თვალყურის დევნება. როცა სარეზერვო ასლის შექმნის პროცესი დიდხანს გრძელდება, მაშინ ამ დროის

განმავლობაში მომხმარებელმა შეიძლება მონაცემთა ბაზაში ის მონაცემები შეცვლოს, რომლებიც უკვე გადაიწერა სარეზერვო ასლში. მაგალითად, თუ მომხმარებელმა წაშალა ის სტრიქონი, რომელიც გადაწერილი იყო სარეზერვო ასლში და შემდეგ ეს სტრიქონი დაუმატა ცხრილს ბოლოში, მაშინ ეს სტრიქონი ორჯერ გადაიწერება სარეზერვო ასლში.

ამრიგად, სრული სარეზერვო ასლი შეიცავს მონაცემთა ბაზის ყველა მონაცემს სარეზერვო ასლის შექმნის ოპერაციის დამთავრების მომენტისათვის. ეს იმას ნიშნავს, რომ თუ რომელიმე მომხმარებელმა სარეზერვო ასლის შექმნის პროცესში შეცვალა მონაცემები რომელიმე ცხრილში, მაშინ ეს ცვლილებები დაუყოვნებლივ აისახება სარეზერვო ასლში.

სრული სარეზერვო ასლი არის საბაზო, რადგან ის გამოიყენება დიფერენცირებული და ტრანზაქციების ჟურნალის სარეზერვო ასლის შექმნისას.

სრული და დიფერენცირებული სარეზერვო ასლის შექმნის დროს არ ხდება ტრანზაქციების ჟურნალის ჩამოჭრა (ზომის შემცირება) და დროთა განმავლობაში ამან შეიძლება გამოიწვიოს მისი გადავსება. ამის თავიდან ასაცილებლად რეგულარულად უნდა შევასრულოთ ტრანზაქციების ჟურნალის ჩამოჭრა ხელით, რისთვისაც შეგვიძლია გამოვიყენოთ BACKUP LOG ბრძანება TRUNCATE_ONLY პარამეტრით, ან ეს ავტომატურად უნდა გავაკეთებინოთ სერვერს, რისთვისაც trunc. log on chkpt პარამეტრი უნდა დავაყენოთ ON მდგომარეობაში sp_dboption შენახული პროცედურის გამოყენებით.

მაგალითი. Shekveta მონაცემთა ბაზისთვის ჩავრთოთ ტრანზაქციების ჟურნალის ავტომატური ჩამოჭრის რეჟიმი:

```
sp_dboption 'Shekveta', 'trunc. log on chkpt', 'ON ';
```

მონაცემთა ბაზის დიფერენცირებული ასლი

მონაცემთა ბაზის დიფერენცირებული სარეზერვო ასლი შეიცავს ინფორმაციას იმ ცვლილებების შესახებ, რომლებიც შესრულდა მონაცემთა ბაზაში მისი სრული სარეზერვო ასლის უკანასკნელი შექმნის მომენტიდან. შედეგად, დიფერენცირებული ასლი ნაკლებ ადგილს იკავებს და შექმნისათვის ნაკლებ დროს მოითხოვს. ეს შესაძლებლობას გვაძლევს პერიოდულად შევქმნათ დიდი ზომის მონაცემთა ბაზების სარეზერვო ასლი შედარებით მცირე დროის განმავლობაში. ამრიგად, დიფერენცირებული სარეზერვო ასლის შექმნის წინ, ჯერ უნდა შევქმნათ მონაცემთა ბაზის სრული სარეზერვო ასლი.

დიფერენცირებული ასლების მიმდევრობით შექმნის დროს ყოველ მომდევნო ასლში მთლიანად იქნება მოთავსებული ის სტრიქონები, რომლებიც ჩართული იყო წინა ასლში და ის სტრიქონები, რომლებიც შეიცვალა წინა ასლის შექმნის შემდეგ. ამიტომ, მონაცემთა ბაზის აღსადგენად, უნდა გამოვიყენოთ მხოლოდ უკანასკნელი დიფერენცირებული სარეზერვო ასლი. შესაბამისად, მონაცემთა ბაზის აღდგენის დროს, ჯერ უნდა აღვადგინოთ მონაცემთა ბაზის სრული ასლი, შემდეგ კი - უკანასკნელი დიფერენცირებული ასლი.

დიდი ზომის მონაცემთა ბაზებში, რომლებშიც ცვლილებების რაოდენობა მცირეა, დიფერენცირებული სარეზერვო ასლის შექმნა ყველაზე ოპტიმალური მეთოდია. ადმინისტრატორს შეუძლია კვირაში ერთხელ შექმნას მონაცემთა ბაზის სრული სარეზერვო ასლი, ხოლო ყოველ ღამეს კი - დიფერენცირებული ასლი.

სრული და დიფერენცირებული სარეზერვო ასლების შექმნა

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
   [ [ , ] DESCRIPTION = { 'ტექსტი' | @text_variable } ]
   [ [ , ] DIFFERENTIAL ]
   [ [ , ] EXPIREDATE = { თარიღი | @date_var } |
   [ [ , ] NAME = { არქივის_სახელი | @backup_set_name_var } ]
   [ [ , ] { NOSKIP | SKIP } ]
   [ [ , ] PASSWORD = { პაროლი | @password_variable } ]
   [ [ , ] RESTART ] ]
]
```

განვიხილოთ არგუმენტების დანიშნულება.

– *მონაცემთა_ბაზის_სახელი*. მიუთითებს მონაცემთა ბაზის სახელს, რომლის არქივირებაც უნდა შესრულდეს.

– <*ინფორმაციის_მატარებელი*>. მიუთითებს ინფორმაციის მატარებელს, რომელზეც უნდა მოთავსდეს სარეზერვო ასლი. მისი სინტაქსია:

```
<ინფორმაციის_მატარებელი> :: =
{
{ ინფორმაციის_მატარებლის_სახელი | @logical_backup_device_name_var } |
{ DISK | TAPE } = { 'ფიზიკური_მოწყობილობის_სახელი' | @physical_backup_device_name_var }
}
```

- *ინფორმაციის_მატარებლის_სახელი* არის მოწყობილობის ლოგიკური სახელი, რომელიც მას მიენიჭა sp_addumpdevice შენახული პროცედურის გამოყენებით.
- DISK | TAPE არგუმენტები განსაზღვრავენ შესაქმნელი მოწყობილობის ტიპს.

– DESCRIPTION = '*ტექსტი*'. ეს არგუმენტი უნდა მივუთითოთ იმ შემთხვევაში, როცა არქივთან ერთად უნდა ჩაიწეროს მისი ტექსტური აღწერა. ტექსტის ზომა უნდა იყოს მაქსიმუმ 255 სიმბოლო.

– DIFFERENTIAL. მიუთითებს, რომ უნდა შეიქმნას მონაცემთა ბაზის დიფერენცირებული ასლი. თუ ის არ არის მითითებული, მაშინ შეიქმნება სრული ასლი.

– EXPIREDATE = *თარიღი*. თუ მითითებულია, მაშინ არქივზე გადაწერა არ შესრულდება მითითებული თარიღის გასვლამდე.

– NAME = *არქივის_სახელი*. არქივის სახელია, რომლის სიგრძე არ უნდა აღემატებოდეს 128 სიმბოლოს. თუ სახელი მითითებული არ არის, მაშინ არქივს ცარიელი სახელი მიენიჭება.

– NOSKIP. წინა არგუმენტისაგან განსხვავებით ამოწმებს არქივების სახელებსა და აქტუალურობას, რათა თავიდან ავიცილოთ შემთხვევით მასზე გადაწერა.

– RESTART. თუ მითითებულია, მაშინ სერვერი გააგრძელებს სარეზერვო ასლის შექმნის შეწყვეტილ ოპერაციას.

– SKIP. უნდა მივუთითოთ მაშინ, თუ გადავწყვიტეთ, რომ არქივები, რომლებისთვისაც დაყენებული იყო თარიღის გასვლის ან დროის პერიოდის ამოწურვის პარამეტრები, აღარ გვჭირდება. ასეთ შემთხვევაში, სარეზერვო ასლი გადაეწერება არსებულ ასლებს.

მაგალითი.

ა. შევქმნათ Shekveta მონაცემთა ბაზის სრული ასლი. მოთხოვნას აქვს სახე:

```
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';
```

შედეგი:

```
Processed 248 pages for database 'Shekveta', file 'Shekveta_Data' on file 4.
Processed 1 pages for database 'Shekveta', file 'Shekveta_Log' on file 4.
BACKUP DATABASE successfully processed 249 pages in 0.493 seconds (4.137 MB/sec).
```

ბ. ახლა შევქმნათ Shekveta მონაცემთა ბაზის დიფერენცირებული ასლი:

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

შედეგი:

```
Processed 40 pages for database 'Shekveta', file 'Shekveta_Data' on file 3.
Processed 1 pages for database 'Shekveta', file 'Shekveta_Log' on file 3.
BACKUP DATABASE WITH DIFFERENTIAL successfully processed 41 pages in 0.204 seconds (1.611 MB/sec).
```

სრული და დიფერენცირებული სარეზერვო ასლიდან აღდგენა

სრული და დიფერენცირებული სარეზერვო ასლიდან აღსადგენად გამოიყენება RESTORE DATABASE ბრძანება, რომლის სინტაქსია:

```
RESTORE DATABASE { მონაცემთა_ბაზის_სახელი | @database_name_var }
[ FROM <ინფორმაციის_მატარებელი> [ ,...n ] ]
[ WITH
   [ RESTRICTED_USER ]
   [ [ , ] FILE = { ფაილის_ნომერი | @file_number } ]
   [ [ , ] MOVE 'ფაილის_ლოგიკური_სახელი' TO 'ფაილის_ფიზიკური_სახელი' ] [ ,...n ]
   [ [ , ] { NORECOVERY | RECOVERY | STANDBY = გაუქმების_ფაილის_სახელი } ]
   [ [ , ] PASSWORD = { პაროლი | @password_variable } ]
   [ [ , ] REPLACE ]
   [ [ , ] RESTART ]
]
```

განვიხილოთ არგუმენტების დანიშნულება.

– *მონაცემთა_ბაზის_სახელი*. მონაცემთა ბაზის სახელია, როლის აღდგენა ხდება სრული ან დიფერენცირებული სარეზერვო ასლიდან.

– FROM <*ინფრომაციის_მატარებელი*>. იმ მოწყობილობის სახელია, საიდანაც უნდა შესრულდეს აღდგენა. თუ ეს არგუმენტი არ არის მითითებული, მაშინ სრულდება მონაცემთა ბაზის *რეკონსტრუქცია* (*recovery*). ამ შემთხვევაში, აუცილებლად უნდა მივუთითოთ NORECOVERY, RECOVERY ან STANDBY არგუმენტი.

– FILE = *ფაილის_ნომერი*. მიუთითებს სარეზერვო ასლის ნაკრების (backup set) ნომერს ინფორმაციის მატარებელზე.

– MOVE '*ფაილის_ლოგიკური_სახელი*' TO '*ფაილის_ფიზიკური_სახელი*'. გამოიყენება '*ფაილის_ლოგიკური_სახელის*' მქონე ფაილისთვის '*ფაილის_ფიზიკური_სახელის*' მისანიჭებლად. გაჩუმებით ფაილებს ისეთივე ფიზიკური სახელი აქვთ, როგორიც არქივირების დროს. ამრიგად, სარეზერვო ასლიდან აღდგენისას, სერვერი ეცდება ფაილები მოათავსოს იმავე კატალოგში იმავე სახელით, როგორიც ჰქონდა არქივირებამდე. მაგრამ, თუ

დაგვჭირდა სარეზერვო ასლის გადატანა სხვა კომპიუტერზე ან თუ გვინდა, რომ ერთ კომპიუტერზე გვქონდეს როგორც ძველი, ისე ახალი ფაილები, მაშინ აუცილებელია ამ არგუმენტის გამოყენება.

– NORECOVERY. ეთითება იმისათვის, რომ მომხმარებლებმა ვერ შეძლონ მონაცემთა ბაზასთან მუშაობა. კერძოდ, შესაძლებელია სიტუაცია, როცა საჭირო მდგომარეობამდე მონაცემთა ბაზის აღსადგენად უნდა შევასრულოთ რამდენიმე ტრანზაქციის ჟურნალის მიმდევრობითი აღდგენა. მაშინ ეს არგუმენტი უნდა გამოვიყენოთ ყველა ჟურნალის აღდგენისას, უკანასკნელის გარდა.

– RECOVERY. თუ მითითებულია ეს პარამეტრი, მაშინ აღდგენის შემდეგ სისტემა შეასრულებს ყველა დაუმთავრებელი ტრანზაქციის უკუქცევას.

– REPLACE. თუ სარეზერვო ასლიდან აღდგენის დროს სერვერზე არსებობს მონაცემთა ბაზა ამავე სახელით, მაშინ შესრულდება მასზე გადაწერა.

– RESTART. აგრძელებს არქივიდან აღდგენის პროცესს იმ პოზიციიდან, რომელშიც იმყოფებოდა აღდგენის პროცესი მის შეწყვეტამდე.

მაგალითები.

ა. მოთხოვნაში სრულდება აღდგენა სრული სარეზერვო ასლიდან. Shekveta მონაცემთა ბაზა არ არსებობს და ის შეიქმნება. მოთხოვნას აქვს სახე:

```
USE Master ;
RESTORE DATABASE Shekveta  FROM Shekveta_Asli;
```

შედეგი:

```
Processed 240 pages for database 'Shekveta', file 'Student_Data' on file 1.
Processed 1 pages for database 'Shekveta', file 'Student_Log' on file 1.
RESTORE DATABASE successfully processed 241 pages in 0.360 seconds (5.484 MB/sec).
```

ბ. მოთხოვნაში სრულდება აღდგენა სრული სარეზერვო ასლიდან. Shekveta მონაცემთა ბაზა უკვე არსებობს და ხდება მასზე გადაწერა ასლიდან. მოთხოვნას აქვს სახე:

```
USE Master;
RESTORE DATABASE Shekveta  FROM Shekveta_Asli WITH REPLACE;
```

შედეგი:

```
Processed 240 pages for database 'Shekveta', file 'Student_Data' on file 1.
Processed 1 pages for database 'Shekveta', file 'Student_Log' on file 1.
RESTORE DATABASE successfully processed 241 pages in 0.394 seconds (5.010 MB/sec).
```

გ. მოთხოვნაში სრულდება აღდგენა ჯერ სრული სარეზერვო ასლიდან შემდეგ კი დიფერენცირებული ასლიდან:

```
USE Master;
RESTORE DATABASE Shekveta FROM Shekveta_Asli WITH NORECOVERY, REPLACE, FILE = 8;
RESTORE DATABASE Shekveta FROM Shekveta_Asli WITH FILE = 9;
```

შედეგი:

```
Processed 240 pages for database 'Shekveta', file 'Shekveta_Data' on file 8.
Processed 1 pages for database 'Shekveta', file 'Shekveta_Log' on file 8.
RESTORE DATABASE successfully processed 241 pages in 0.494 seconds (3.996 MB/sec).
Processed 40 pages for database 'Shekveta', file 'Shekveta_Data' on file 9.
Processed 1 pages for database 'Shekveta', file 'Shekveta_Log' on file 9.
RESTORE DATABASE successfully processed 41 pages in 0.236 seconds (1.421 MB/sec).
```

გ. მოთხოვნაში ხდება RESTART რეჟიმის დემონსტრირება სარეზერვო ასლიდან აღდგენის ოპერაციის შეწყვეტის შემდეგ:

```
-- Shekveta მონაცემთა ბაზის აღდგენა შეწყდა ძაბვის ვარდნის გამო
RESTORE DATABASE Shekveta FROM Shekveta_Asli WITH REPLACE
-- მისი აღდგენა გაგრძელდება, რადგან მითიტებულია RESTORE RESTART არგუმენტი
RESTORE DATABASE Shekveta FROM Shekveta_Asli WITH RESTART
```

ტრანზაქციების ჟურნალის სარეზერვო ასლი

ზოგჯერ საჭირო ხდება მონაცემთა ბაზის აღდგენა შუალედურ მდგომარეობაში. დავუშვათ, მონაცემთა ბაზის სრულ ასლს ვქმნით კვირაში ერთხელ, ხოლო დიფერენცირებულ ასლს - ყოველ ღამე. დავუშვათ, აღმოვაჩინეთ, რომ ერთ ცხრილში სტრიქონების დიდი ნაწილი წაშლილია და გადავწყვიტეთ მონაცემთა ბაზის აღდგენა სრული სარეზერვო ასლიდან. აღდგენის შემდეგ აღმოჩნდა, რომ სარეზერვო ასლი არ შეიცავს წაშლილ სტრიქონებს. ეს იმიტომ მოხდა, რომ სარეზერვო ასლი შეიქმნა ცხრილიდან სტრიქონების წაშლის შემდეგ, მაგრამ, ეს არ არის დიდი პრობლემა, რადგან გვაქვს უფრო „ახალი" დიფერენცირებული ასლი, თუმცა მთელი დღის განმავლობაში შესრულებული ცვლილებები დაკარგული იქნება.

ასეთი სიტუაციიდან გამოსავალია მონაცემთა ბაზის აღდგენა იმ მდგომარეობაში, რომელშიც ის იმყოფებოდა იმ მოთხოვნის შესრულებამდე, რომელმაც მონაცემების წაშლა გამოიწვია. სარეზერვო ასლის ზემოთ აღწერილი ორი ტიპი ამის საშუალებას არ იძლევა. ეს შესაძლებელია ტრანზაქციების ჟურნალის სარეზერვო ასლის გამოყენებით.

სარეზერვო ასლის ამ ტიპს საფუძვლად უდევს სერვერის მიერ ტრანზაქციების ჟურნალის შემოწმება იმ ტრანზაქციების მოძებნის მიზნით, რომლებიც მონაცემთა ბაზაში შესრულდა უკანასკნელი სარეზერვო ასლის შექმნის მომენტიდან. უკანასკნელი სარეზერვო ასლი შეიძლება იყოს სრული, დიფერენცირებული ან ტრანზაქციების ჟურნალის სარეზერვო ასლი. ზოგიერთ შემთხვევაში, შეიძლება გამოყენებული იყოს ფაილის ან ფაილების ჯგუფის სარეზერვო ასლიც.

როგორც ვიცით, ტრანზაქციების ჟურნალში აისახება მონაცემთა ბაზის საწყისი მდგომარეობა, აგრეთვე ის მდგომარეობა, რომელშიც მონაცემთა ბაზა გადავიდა ტრანზაქციის დამთავრების შემდეგ. ამრიგად, ვიცით რა მონაცემთა ბაზის საწყისი მდგომარეობა, შეგვიძლია მიმდევრობით შევასრულოთ ტრანზაქციები. შედეგად მონაცემთა ბაზა თანდათან გადავა იმ მდგომარეობაში, რომელშიც ის იმყოფებოდა ტრანზაქციების ჟურნალის სარეზერვო ასლის შექმნის დამთავრების მომენტში.

ტრანზაქციების ჟურნალის სარეზერვო ასლის შექმნის დროს არქივში მოთავსდება მხოლოდ ის ტრანზაქციები, რომლებიც იყო დამთავრებული და დაფიქსირებული სარეზერვო ასლის შექმნის დამთავრების მომენტში. როცა სარეზერვო ასლში ტრანზაქციების სია ამოიწურება, მაშინ ან უნდა დავამთავროთ მონაცემთა ბაზის აღდგენა, ან აღვადგინოთ ტრანზაქციების ჟურნალის კიდევ ერთი სარეზერვო ასლი.

სარეზერვო ასლიდან მონაცემების მთლიანად აღდგენის ნაცვლად შეგვიძლია

აღვადგინოთ მონაცემების ნაწილი. ამისათვის ცხადად უნდა მივუთითოთ დროის მომენტი, როცა უნდა დავამთავროთ სარეზერვო ასლიდან აღდგენა. ეს მომენტი წარმოადგენს კონკრეტულ თარიღს. მონაცემთა ბაზა აღდგება იმ მდგომარეობაში, რომელშიც ის იყო დროის მითითებულ მომენტამდე. ბუნებრივია, რომ ტრანზაქციების ჟურნალის სარეზერვო ასლი უნდა შეიცავდეს დროის ამ მომენტს.

შეგვიძლია, აგრეთვე მივუთითოთ კონკრეტული ტრანზაქცია, რომლის დაწყებამდე ან დამთავრების შემდეგ უნდა შეწყდეს სარეზერვო ასლიდან მონაცემების აღდგენა. მაგრამ, მანამდე ტრანზაქციების ჟურნალი შესაბამისი გზით უნდა იყოს მონიშნული. როგორც ვიცით, ტრანზაქციის მოსანიშნად BEGIN TRANS ბრძანებაში ტრანზაქციის სახელთან ერთად უნდა მივუთითოთ WITH MARK საკვანძო სიტყვა. შემდეგში, ტრანზაქციების ჟურნალის სარეზერვო ასლიდან აღდგენის დროს უნდა მივუთითოთ ტრანზაქციის სახელი, რითაც განისაზღვრება წერტილი, სადამდეც უნდა შესრულდეს სარეზერვო ასლიდან აღდგენა.

ამრიგად, იმისათვის, რომ შევძლოთ მონაცემთა ბაზის აღდგენა უკანასკნელი სარეზერვო ასლის შექმნის შემდეგ, უნდა გამოვიყენოთ ტრანზაქციების ჟურნალის სარეზერვო ასლი, მაგრამ სერვერზე სრულდება *არაპროტოკოლირებადი ოპერაციებიც* (*nonlogged operation*), რომლებიც ხასიათდება იმით, რომ მათ მიერ შესრულებული მოქმედებები არ აისახება ტრანზაქციების ჟურნალში. ასეთი ოპერაციებია WRITETEXT და UPDATETEXT, აგრეთვე მონაცემების ჩასმა მასობრივი გადაწერის მექანიზმების გამოყენებით. ამიტომ არაპროტოკოლირებადი ოპერაციის შესრულების წინ ჯერ უნდა შევქმნათ სრული ან დიფერენცირებული სარეზერვო ასლი, შემდეგ კი - ტრანზაქციების ჟურნალის სარეზერვო ასლი.

ტრანზაქციების ჟურნალის სარეზერვო ასლის შექმნა შეუძლებელი იქნება, თუ ჩართულია ტრანზაქციების ჟურნალის ავტომატურად ჩამოჭრის რეჟიმი ანუ დაყენებულია trunc. log on chkpt პარამეტრი. ასეთ შემთხვევაში სერვერი პერიოდულად შლის ინფორმაციას ტრანზაქციების ჟურნალიდან უკვე დამთავრებული ტრანზაქციების შესახებ. ამიტომ თუ ვგეგმავთ ტრანზაქციების ჟურნალის სარეზერვო ასლის შექმნას, მაშინ ტრანზაქციების ჟურნალის ავტომატური ჩამოჭრის რეჟიმი უნდა გამოვრთოთ. მხედველობაში უნდა გვქონდეს, აგრეთვე ის, რომ ტრანზაქციების ჟურნალის სარეზერვო ასლის შექმნისას ავტომატურად სრულდება მისი ჩამოჭრაც.

ამრიგად, ტრანზაქციების ჟურნალის სარეზერვო ასლის შექმნისას ან ასლიდან აღდგენისას აუცილებელია გვქონდეს ათვლის წერტილი. ასეთი წერტილი შეიძლება იყოს სრული ან დიფერენცირებული სარეზერვო ასლი. რომელიმე ამ ასლიდან აღდგენის შემდეგ უნდა შევასრულოთ ტრანზაქციების ჟურნალის სარეზერვო ასლიდან აღდგენა. ბუნებრივია, რომ პირველი ტრანზაქცია ტრანზაქციების ჟურნალის სარეზერვო ასლში უნდა იყოს ის ტრანზაქცია, რომელიც შესრულდა პირველი უშუალოდ მას შემდეგ, რაც დამთავრდა შესაბამისი სრული ან დიფერენცირებული ასლის შექმნა.

ტრანზაქციების ჟურნალის სარეზერვო ასლი შეიძლება შეიქმნას ნებისმიერ დროს შესაბამისი სრული ან დიფერენცირებული ასლის შექმნის შემდეგ. ამრიგად, თუ დიფერენცირებულ ასლს ვქმნით ღამე, მაშინ დღის განმავლობაში ნებისმიერ დროს შეგვიძლია შევქმნათ ტრანზაქციების ჟურნალის სარეზერვო ასლი და მონაცემთა ბაზა აღვადგინოთ იმ მდგომარეობაში, რომელშიც ის იყო ტრანზაქციების ჟურნალის სარეზერვო ასლის შექმნამდე.

ეს შესაძლებლობა შეგვიძლია გამოვიყენოთ მონაცემთა ბაზის აღსადგენად აპარატურული შეფერხების შემდეგ. თუ ტრანზაქციების ჟურნალი არ დაზიანდა, მაშინ მისთვის შეგვიძლია შევქმნათ სარეზერვო ასლი და ადრე შექმნილ სრულ ან დიფერენცირბულ ასლთან ერთად აღვადგინოთ მონაცემთა ბაზა იმ მდგომარეობაში, რომელშიც ის იყო უშუალოდ შეფერხების წინ.

უმჯობესია, მონაცემებისა და ტრანზაქციების ჟურნალის ფაილები სხვადასხვა ფიზიკურ დისკებზე მოვათავსოთ. რომელიმე დისკის დაზიანებისას პრაქტიკულად ყოველთვის შევძლებთ მონაცემთა ბაზის აღდგენას.

ძალიან დიდი ზომის მონაცემთა ბაზასთან მუშაობის დროს შეგვიძლია გამოვიყენოთ ტიპური რეკომენდებული გრაფიკი:

– კვირაში ერთხელ კვირა დღის ღამით ვქმნით სრულ სარეზერვო ასლს;
– ყოველ ღამე ვქმნით დიფერენცირებულ ასლს;
– ყოველ სამუშაო დღეს ერთ ან ორ საათში ერთხელ ვქმნით ტრანზაქციების ჟურნალის სარეზერვო ასლს.

შესაძლებელია სარეზერვო ასლების შექმნის პროცესის ავტომატიზება.

ტრანზაქციების ჟურნალის სარეზერვო ასლის შექმნა

ტრანზაქციების ჟურნალის სარეზერვო ასლის შესაქმნელად გამოიყენება BACKUP LOG ბრძანება, რომლის სინტაქსია:

```
BACKUP LOG { მონაცემთა_ბაზის_სახელი | @database_name_var }
{ {
   [ WITH
     { NO_LOG | TRUNCATE_ONLY } ]
}
   TO <ინფორმაციის_მატარებელი> [ ,...n ]
   [ WITH
      [ [ , ] DESCRIPTION = { 'ტექსტი' | @text_variable } ]
      [ [ ,] EXPIREDATE = { თარიღი | @date_var } |
         RETAINDAYS = { დღეების_რაოდენობა | @days_var } ]
      [ [ , ] NAME = { არქივის_სახელი | @backup_set_name_var } ]
      [ [ , ] NO_TRUNCATE ]
      [ [ , ] PASSWORD = { პაროლი | @password_variable } ]
      [ [ , ] RESTART ]  ]
]
}
```

განვიხილოთ არგუმენტების დანიშნულება.

– NO_LOG | TRUNCATE_ONLY. ერთ-ერთი მიეთითება იმ შემთხვევაში, როცა საჭიროა შესრულდეს ტრანზაქციების ჟურნალის მხოლოდ ჩამოჭრა არქივის შექმნის გარეშე. ეს არგუმენტი ძალაშია გაჩუმებით.
– NO_TRUNCATE. აუქმებს ტრანზაქციების ჟურნალის ჩამოჭრას. ეს არგუმენტი ავტომატურად გამოიყენება, თუ ხდება დაზიანებული ტრანზაქციების ჟურნალის არქივირება. ჟურნალის ჩამოჭრა არ ხდება, იმ შემთხვევაშიც, როცა სრულდება იმ მონაცემთა ბაზის არქივირება, რომლის აღდგენა შესრულდა მხოლოდ NORECOVERY არგუმენტის გამოყენებით.

მაგალითები.

ა. მოთხოვნაში იქმნება ტრანზაქციების ჟურნალის ასლი:

```
-- ტრანზაქციების ჟურნალის ასლის შექმნის უფლების მისაცემად Shekveta მონაცემთა ბაზა ისე
-- უნდა შევცვალოთ, რომ მან გამოიყენოს სრული აღდგენის მოდელი
USE master;
GO
ALTER DATABASE Shekveta SET RECOVERY FULL
```

```
-- ვქმნით Shekveta_Asli მოწყობილობას Shekveta მონაცემთა ბაზის ასლისთვის
USE master
GO
EXEC sp_addumpdevice 'DISK', 'Shekveta_Asli',
'C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\BACKUP\Shekveta_Asli.bak'
-- Shekveta_Asli_Log ლოგიკური მოწყობილობის შექმნა ტრანზაქციების ჟურნალის ასლისთვის
USE master
GO
EXEC sp_addumpdevice 'DISK', 'Shekveta_Asli_Log',
'C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\BACKUP\Shekveta_Asli_Log.bak'
-- Shekveta მონაცემთა ბაზის სრული ასლის შექმნა
BACKUP DATABASE Shekveta TO Shekveta_Asli WITH NAME = 'Shekveta_Asli'
-- ტრანზაქციების ჟურნალის ასლის შექმნა
BACKUP LOG Shekveta TO Shekveta_Asli_Log WITH NAME = 'Shekveta_Asli_Log'
```

```
Processed 240 pages for database 'Shekveta', file 'Shekveta_Data' on file 5.
Processed 2 pages for database 'Shekveta', file 'Shekveta_Log' on file 5.
BACKUP DATABASE successfully processed 242 pages in 0.384 seconds (5.144 MB/sec).
Processed 3 pages for database 'Shekveta', file 'Shekveta_Log' on file 3.
BACKUP LOG successfully processed 3 pages in 0.138 seconds (0.140 MB/sec).
```

ბ. მოთხოვნაში ტრანზაქციების ჟურნალის ასლი Shekveta_Asli_Log1 და Shekveta_Asli_Log2 ფაილებში იქმნება:

```
USE master
GO
EXEC sp_addumpdevice 'DISK', 'Shekveta_Asli_Log1',
        'C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\BACKUP\Shekveta_Asli_Log1.bak'
EXEC sp_addumpdevice 'DISK', 'Shekveta_Asli_Log2',
        'C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\BACKUP\Shekveta_Asli_Log2.bak'
BACKUP DATABASE Shekveta TO Shekveta_Asli
BACKUP LOG Shekveta TO Shekveta_Asli_Log1
BACKUP LOG Shekveta TO Shekveta_Asli_Log2
```

```
Processed 240 pages for database 'Shekveta', file 'Shekveta_Data' on file 6.
Processed 2 pages for database 'Shekveta', file 'Shekveta_Log' on file 6.
BACKUP DATABASE successfully processed 242 pages in 1.304 seconds (1.514 MB/sec).
Processed 3 pages for database 'Shekveta', file 'Shekveta_Log' on file 2.
BACKUP LOG successfully processed 3 pages in 0.174 seconds (0.111 MB/sec).
Processed 0 pages for database 'Shekveta', file 'Shekveta_Log' on file 2.
BACKUP LOG successfully processed 0 pages in 0.202 seconds (0.000 MB/sec).
```

ტრანზაქციების ჟურნალიდან აღდგენა

ტრანზაქციების ჟურნალიდან აღსადგენად გამოიყენება ბრძანება:

```
RESTORE LOG { მონაცემთა_ბაზის_სახელი | @database_name_var }
```

[FROM <*ინფორმაციის_მატარებელი*> [,...n]]
[WITH
  [RESTRICTED_USER]
  [[,] FILE = { *ფაილის_ნომერი* | @file_number }]
  [[,] MOVE '*ფაილის_ლოგიკური_სახელი*' TO '*ფაილის_ფიზიკური_სახელი*']
      [,...n]
  [[,] RESTART]
  [[,] STOPAT = { *თარიღი_დრო* | @date_time_var }
    | [,] STOPATMARK = '*ჭდის_სახელი*' [AFTER *თარიღი_დრო*]
    | [,] STOPBEFOREMARK = '*ჭდის_სახელი*' [AFTER *თარიღი_დრო*]
  ]
]

აქ STOPAT = *თარიღი_დრო* არგუმენტი მიუთითებს დროის მომენტს, რომელამდეც უნდა მოხდეს აღდგენა. დრო შეგვიძლია მივუთითოთ სიმბოლური მუდმივას ან ცვლადის სახით, რომლის ტიპია DATETIME ან SMALLDATETIME.

მაგალითები.

ა. მოთხოვნაში ხდება სრული ასლიდან და ტრანზაქციების ჟურნალიდან მონაცემების აღდგენა:

```
RESTORE DATABASE Shekveta  FROM Shekveta_Asli WITH NORECOVERY, REPLACE;
RESTORE LOG Shekveta  FROM Shekveta_Asli_Log WITH RECOVERY;
```

```
Processed 240 pages for database 'Shekveta', file 'Shekveta_Data' on file 1.
Processed 2 pages for database 'Shekveta', file 'Shekveta_Log' on file 1.
RESTORE DATABASE successfully processed 242 pages in 0.466 seconds (4.254 MB/sec).
Processed 0 pages for database 'Shekveta', file 'Shekveta_Data' on file 1.
Processed 3 pages for database 'Shekveta', file 'Shekveta_Log' on file 1.
RESTORE LOG successfully processed 3 pages in 0.025 seconds (0.757 MB/sec).
```

ბ. მოთხოვნაში ხდება სრული ასლიდან და ტრანზაქციების ჟურნალიდან აღდგენა და აღდგენილი მონაცემთა ბაზის გადაწერა C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data\ კატალოგში სხვა სახელით:

```
USE master;
GO
RESTORE DATABASE Shekveta  FROM Shekveta_Asli WITH NORECOVERY, REPLACE,
MOVE
'Shekveta_Data' TO 'C:\Program Files\Microsoft SQL
Server\MSSQL.1\MSSQL\Data\AxaliShekveta.mdf',
MOVE
'Shekveta_Log' TO 'C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data\AxaliShekveta.ldf';
RESTORE LOG Shekveta  FROM Shekveta_Asli_Log WITH RECOVERY, REPLACE;
```

```
Processed 240 pages for database 'Shekveta', file 'Shekveta_Data' on file 1.
Processed 2 pages for database 'Shekveta', file 'Shekveta_Log' on file 1.
RESTORE DATABASE successfully processed 242 pages in 0.426 seconds (4.653 MB/sec).
Processed 0 pages for database 'Shekveta', file 'Shekveta_Data' on file 1.
Processed 3 pages for database 'Shekveta', file 'Shekveta_Log' on file 1.
RESTORE LOG successfully processed 3 pages in 0.040 seconds (0.473 MB/sec).
```

გ. მოთხოვნაში ხდება აღდგენა 2008 წლის 7 აპრილის 12 საათამდე:

```
USE master;
GO
RESTORE DATABASE Shekveta  FROM Shekveta_Asli
            WITH NORECOVERY, REPLACE;
RESTORE LOG Shekveta  FROM Shekveta_Asli_Log WITH RECOVERY, REPLACE,
                                                    STOPAT = 'Apr 7, 2008 12:00 AM';
```

```
Processed 240 pages for database 'Shekveta', file 'Shekveta_Data' on file 1.
Processed 2 pages for database 'Shekveta', file 'Shekveta_Log' on file 1.
RESTORE DATABASE successfully processed 242 pages in 0.418 seconds (4.742 MB/sec).
Processed 0 pages for database 'Shekveta', file 'Shekveta_Data' on file 1.
Processed 3 pages for database 'Shekveta', file 'Shekveta_Log' on file 1.
RESTORE LOG successfully processed 3 pages in 0.039 seconds (0.485 MB/sec).
```

დ. მოთხოვნაში ხდება აღდგენა 2008 წლის 7 აპრილის 12 საათამდე Shekveta_Asli_Log1 და Shekveta_Asli_Log2 ასლებიდან, რომლებიც შეიქმნა წინა განყოფილების ბ მაგალითში:

```
USE master;
GO
RESTORE DATABASE Shekveta  FROM Shekveta_Asli
          WITH NORECOVERY, REPLACE;
RESTORE LOG Shekveta  FROM Shekveta_Asli_Log1 WITH NORECOVERY, REPLACE;
RESTORE LOG Shekveta  FROM Shekveta_Asli_Log2
          WITH RECOVERY, STOPAT = 'Apr 7, 2008 12:00 AM';
```

```
Processed 240 pages for database 'Shekveta', file 'Shekveta_Data' on file 1.
Processed 2 pages for database 'Shekveta', file 'Shekveta_Log' on file 1.
RESTORE DATABASE successfully processed 242 pages in 0.385 seconds (5.149 MB/sec).
Processed 0 pages for database 'Shekveta', file 'Shekveta_Data' on file 1.
Processed 2 pages for database 'Shekveta', file 'Shekveta_Log' on file 1.
RESTORE LOG successfully processed 2 pages in 0.037 seconds (0.235 MB/sec).
RESTORE LOG successfully processed 0 pages in 0.078 seconds (0.000 MB/sec).
```

ფაილებისა და ფაილების ჯგუფის სარეზერვო ასლი

ფაილების ან ფაილების ჯგუფების სარეზერვო ასლი (file and filegroup backup) იქმნება მაშინ, როცა საჭირო ხდება კონკრეტული ცხრილის ან ცალკეული სვეტის სარეზერვო ასლის

შექმნა. როგორც ვიცით, სერვერი საშუალებას გვაძლევს ცხადად მივუთითოთ თუ ფაილების რომელ ჯგუფს უნდა ეკუთვნოდეს ცხრილი, მისი ცალკეული სვეტი ან ინდექსი. შედეგად, ფაილების ჯგუფის სარეზერვო ასლის შექმნის გარდა, შეგვიძლია შევქმნათ სარეზერვო ასლი, რომელშიც მოთავსებული იქნება მხოლოდ ცალკეული ცხრილის, სვეტის, ინდექსის და ა.შ. მონაცემები. შედეგად ასეთი სარეზერვო ასლიდან მონაცემების აღდგენა შეიძლება შესრულდეს დამოუკიდებლად, ე.ი. სხვა დანარჩენი მონაცემების აღდგენის გარეშე.

ამ ტიპის სარეზერვო ასლი შეგვიძლია გამოვიყენოთ, მაგალითად, ცნობარების მიმართ, რომლებშიც ინფორმაცია იშვიათად იცვლება. შეცვლილი მონაცემები შეგვიძლია მოვათავსოთ ფაილების ერთ ან მეტ ჯგუფში. სარეზერვო ასლის შექმნის სტრატეგია ასეთ შემთხვევაში შეიძლება ასეთი იყოს:

– ვქმნით მონაცემთა ბაზის სრულ სარეზერვო ასლს. მასში ვინახავთ იმ მონაცემებს, რომლებიც არ იცვლება. ამიტომ პერიოდულად სრული სარეზერვო ასლის შექმნის აუცილებლობა აღარ იარსებებს.
– პერიოდულად უნდა შევქმნათ ფაილების ჯგუფის სარეზერვო ასლი.

შესაბამისად, მონაცემთა ბაზის აღდგენის დროს, ჯერ უნდა აღვადგინოთ მონაცემთა ბაზის სრული ასლი, შემდეგ კი - ფაილების ჯგუფის უკანასკნელი ასლი.

როგორც აღვნიშნეთ, ფაილების ჯგუფები გამოიყენება ადმინისტრირებისა და მონაცემთა ბაზების ობიექტების ფიზიკური განლაგების მართვის გასამარტივებლად. გარდა ამისა, სარეზერვო ასლის შექმნისას ფაილების ჯგუფები გამოიყენება მონაცემების გასანაწილებლად სარეზერვო ასლის შექმნის პროცესის ოპტიმიზების მიზნით. ამრიგად, ფაილებისა და ფაილების ჯგუფების სარეზერვო ასლის შექმნა შეიძლება ძალიან სასარგებლო იყოს მაშინ, როცა მონაცემთა ბაზა განაწილებულია რამდენიმე ფიზიკურ დისკზე.

ფაილებისა და ფაილების ჯგუფების სარეზერვო ასლების შექმნა

ფაილებისა და ფაილების ჯგუფის სარეზერვო ასლის შესაქმნელად გამოიყენება BACKUP DATABASE ბრძანება, რომლის სინტაქსია:

```
BACKUP DATABASE { მონაცემთა_ბაზის_სახელი | @database_name_var }
   <ფაილი_ან_ფაილების_ჯგუფი> [ ,...n ]
TO <ინფორმაციის_მატარებელი> [ ,...n ]
[ WITH
   [ BLOCKSIZE = { ბლოკის_ზომა | @blocksize_variable } ]
   [ [ , ] DESCRIPTION = { 'ტექსტი' | @text_variable } ]
   [ [ , ] DIFFERENTIAL ]
   [ [ , ] EXPIREDATE = { თარიღი | @date_var } |
         RETAINDAYS = { დღეები | @days_var } ]
   [ [ , ] PASSWORD = { პაროლი | @password_variable } ]
   [ [ , ] RESTART ]  ]
]
```

<ფაილი_ან_ფაილების_ჯგუფი> [,...n] კონსტრუქციის სინტაქსია:

```
<ფაილი_ან_ფაილების_ჯგუფი> ::=
{
FILE = { ფაილის_ლოგიკური_სახელი | @logical_file_name_var }
|
FILEGROUP =
         { ფაილების_ჯგუფის_ლოგიკური_სახელი | @logical_filegroup_name_var }
```

}

აქ FILE არგუმენტი არქივში ჩასაწერი მონაცემთა ბაზის ფაილის ლოგიკური სახელია, FILEGROUP არგუმენტი კი - არქივში ჩასაწერი მონაცემთა ბაზის ფაილების ჯგუფის ლოგიკური სახელი.

მაგალითები.

ა. მოთხოვნა ქმნის Shekveta მონაცემთა ბაზის Shekveta_Data ფაილის ასლს:

```
USE Master;
GO
EXEC sp_addumpdevice 'DISK', 'Shekveta_Asli_F',
'C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\BACKUP\Shekveta_Asli.bak';
BACKUP DATABASE Shekveta FILE = 'Shekveta_Data' TO Shekveta_Asli_F
WITH NAME = 'Shekveta_Asli_F';
```

```
Processed 240 pages for database 'Shekveta', file 'Shekveta_Data' on file 1.
Processed 1 pages for database 'Shekveta', file 'Shekveta_Log' on file 1.
BACKUP DATABASE...FILE=<name> successfully processed 241 pages in 0.548 seconds (3.602 MB/sec).
```

ბ. მაგალითში იქმნება Shekveta მონაცემთა ბაზის PRIMARY ფაილების ჯგუფის ასლი:

```
USE Master;
GO
BACKUP DATABASE Shekveta FILEGROUP = 'PRIMARY' TO Shekveta_Asli_F;
```

```
Processed 240 pages for database 'Shekveta', file 'Shekveta_Data' on file 2.
Processed 1 pages for database 'Shekveta', file 'Shekveta_Log' on file 2.
BACKUP DATABASE...FILE=<name> successfully processed 241 pages in 0.498 seconds (3.964 MB/sec).
```

ფაილებისა და ფაილების ჯგუფიდან აღდგენა

ფაილებისა და ფაილების ჯგუფიდან აღსადგენად გამოიყენება ბრძანება:

```
RESTORE DATABASE { მონაცემთა_ბაზის_სახელი | @database_name_var }
   <ფაილი_ან_ფაილების_ჯგუფი> [ ,...n ]
[ FROM <ინფორმაციის_მატარებელი> [ ,...n ] ]
[ WITH
   [ RESTRICTED_USER ]
   [ [ , ] FILE = { ფაილის_ნომერი | @file_number } ]
   [ [ , ] MOVE 'ფაილის_ლოგიკური_სახელი' TO 'ფაილის_ფიზიკური_სახელი' ] [ ,...n ]
   [ [ , ] PASSWORD = { პაროლი | @password_variable } ]
   [ [ , ] REPLACE ]
   [ [ , ] RESTART ] ]
]
```

მაგალითები.

ა. Shekveta მონაცემთა ბაზის აღდგენა სრულდება ორი ფაილიდან. ერთი ფაილია Shekveta_Data, მეორე კი - ტრანზაქციების ჟურნალი. მოთხოვნას აქვს სახე:

```
USE Master;
GO
RESTORE DATABASE Shekveta
```

```
  FILE = 'Shekveta_Data'
  FROM Shekveta_Asli_F WITH NORECOVERY, REPLACE;
RESTORE LOG Shekveta  FROM Shekveta_Asli_Log;
```

```
Processed 240 pages for database 'Shekveta', file 'Shekveta_Data' on file 1.
Processed 1 pages for database 'Shekveta', file 'Shekveta_Log' on file 1.
RESTORE DATABASE ... FILE=<name> successfully processed 241 pages in 0.444 seconds (4.446 MB/sec).
```

ბ. Shekveta მონაცემთა ბაზის აღდგენა სრულდება PRIMARY ფაილების ჯგუფიდან. მოთხოვნას აქვს სახე:

```
USE Master;
GO
RESTORE DATABASE Shekveta
  FILEGROUP = 'PRIMARY'
  FROM Shekveta_Asli_F WITH NORECOVERY, REPLACE;
RESTORE LOG Shekveta  FROM Shekveta_Asli_Log;
```

```
Processed 240 pages for database 'Shekveta', file 'Shekveta_Data' on file 1.
Processed 1 pages for database 'Shekveta', file 'Shekveta_Log' on file 1.
RESTORE DATABASE ... FILE=<name> successfully processed 241 pages in 0.574 seconds (3.439 MB/sec).
```

შეზღუდვები სარეზერვო ასლის შექმნის ოპერაციაზე

არსებობენ ოპერაციები, რომლებიც არ შეიძლება შესრულდეს სარეზერვო ასლის შექმნის დროს. ეს ოპერაციებია:

– მონაცემთა ბაზის ან მისი ფაილების ზომის ავტომატურად ან ხელით შემცირება (shrinking).
– მონაცემთა ბაზის ფაილების შექმნა და წაშლა.
– ინდექსების შექმნა.
– ოპერაციების შესრულება, რომლებიც არ რეგისტრირდება ტრანზაქციების ჟურნალში.

თუ სარეზერვო ასლის შექმნის დაწყების მომენტში სრულდება ზემოთ ჩამოთვლილი ერთ-ერთი ბრძანება, მაშინ გაიცემა შეტყობინება შეცდომის შესახებ. თუ სარეზერვო ასლის შექმნის მომენტში ვცდილობთ ზემოთ ჩამოთვლილი ერთ-ერთი ბრძანების შესრულებას, მაშინ ეს ბრძანება არ სრულდება და ასლის შექმნა გაგრძელდება.

თავი 19. მონაცემების იმპორტი და ექსპორტი

სერვერს აქვს პროგრამები, რომლებიც იძლევა SQL სერვერსა და მონაცემების დამუშავების სხვა სისტემებს შორის მონაცემების გაცვლის საშუალებას, როგორიცაა Access, FoxPRo, Excel და ა.შ.

მონაცემების იმპორტი და ექსპორტი

არსებობს მონაცემების გაცვლის ორი სახე - იმპორტი და ექსპორტი:

– მონაცემების იმპორტი ესაა პროცესი, როცა ხდება მონაცემების ამოღება გარე წყაროებიდან (მაგალითად, Access), დამუშავება და SQL სერვერის მონაცემთა ბაზების ცხრილებში ჩაწერა.

– მონაცემების ექსპორტი ის პროცესია, როცა ხდება მონაცემების ამოღება SQL სერვერის მონაცემთა ბაზების ცხრილებიდან, დამუშავება და მიმღების (მაგალითად, Access) ცხრილებში ჩაწერა.

მონაცემების იმპორტი საჭიროა მაშინ, როცა დავაყენეთ SQL სერვერის ახალი ვერსია და საჭიროა წინა ვერსიიდან ახალ ვერსიაში მონაცემების გადმოტანა. მონაცემების იმპორტი საჭიროა, აგრეთვე იმ შემთხვევაშიც, როცა მონაცემების გადმოტანა გვინდა სხვა სისტემიდან, მაგალითად, როგორიცაა Access.

მონაცემების დამუშავების ბევრ სისტემას, მაგალითად, Access ან Excel, შეუძლია პირდაპირ მიმართოს მონაცემებს SQL სერვერზე, მათი წინასწარი გარდაქმნის გარეშე.

SELECT INTO ბრძანება საშუალებას გვაძლევს ერთი ან მეტი ცხრილიდან ამორჩეული მონაცემების საფუძველზე შევქმნათ ახალი ცხრილი. ცხრილები შეიძლება მოთავსებული იყოს იმავე ან სხვა მონაცემთა ბაზაში ან სხვა სერვერზე. ეს ბრძანება საშუალებას გვაძლევს, მივმართოთ აგრეთვე, მონაცემების განაწილებულ, ჰეტეროგენულ წყაროებს, რომლებიც წარმოდგენილია OLE DB ტექნოლოგიით. ამისათვის საჭიროა ბმული სერვერების კონფიგურირება შენახული პროცედურების გამოყენებით. INSERT ბრძანების შესაძლებლობები SELECT INTO ბრძანების შესაძლებლობების ანალოგიურია.

მონაცემების იმპორტი SQL სერვერზე

ამ განყოფილებაში განვიხილავთ მონაცემების იმპორტს SQL სერვერზე Access და Excel სისტემებიდან.

მონაცემების იმპორტი Access სისტემიდან

SQL სერვერის Shekveta მონაცემთა ბაზაში Access სისტემიდან შევასრულოთ მონაცემების იმპორტი. ამისათვის, Shekveta ბაზის სახელზე ვხსნით კონტექსტურ მენიუს და ვასრულებთ Tasks ქვემენიუს Import Data ბრძანებას (ნახ. 19.1). გაიხსნება SQL Server Import and Export Wizard ფანჯარა (ნახ. 19.2). ვაჭერთ Next კლავიშს. გახსნილი ფანჯრის Data source სიიდან ვირჩევთ Microsoft Access ელემენტს (ნახ. 19.3). 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. 19.4). სურვილისამებრ New კლავიშის საშუალებით შეგვიძლია შევქმნათ ახალი ბაზა. ვირჩევთ აუტენტიფიცირების რეჟიმს და ვაჭერთ Next კლავიშს. გახსნილ ფანჯარაში (ნახ. 19.5) ვირჩევთ Copy data from one or more tables or views გადამრთველს და ვაჭერთ Next კლავიშს. მომდევნო

ფანჯარაში (ნახ. 19.6) გამოჩნდება db5.mdb მონაცემთა ბაზის ობიექტები. მოვნიშნოთ Personali_1 ცხრილი და დავაჭიროთ Next კლავიშს. გახსნილ ფანჯარაში (ნახ. 19.7) ვაჭერთ Next კლავიშს და უკანასკნელ ფანჯარაში (ნახ. 19.8) ვაჭერთ Finish კლავიშს. დაიწყება მონაცემების იმპორტი (ნახ. 19.9). ოპერაციის დამთავრების შემდეგ ვხურავთ ფანჯარას. იმპორტირებული ცხრილის სანახავად ჯერ უნდა გავხსნათ Shekveta მონაცემთა ბაზის Tables ელემენტი. შემდეგ, Personali_1 სახელზე გავხსნათ კონტექსტური მენიუ და შევასრულოთ Select Top 100 Rows ბრძანება.

მონაცემების იმპორტი Excel სისტემიდან

SQL სერვერზე Excel სისტემიდან მონაცემების იმპორტისათვის ვხსნით ნახ. 19.3-ზე ნაჩვენებ ფანჯარას. Data source სიიდან ვირჩევთ Microsoft Excel ელემენტს (ნახ. 19.10). Browse კლავიშის საშუალებით ვირჩევთ საჭირო ფაილს, მაგალითად, Book1.xls. თუ ცხრილის პირველი სტრიქონი სვეტების სახელებს შეიცავს, მაშინ უნდა ჩავრთოთ First row has column names გადამრთველი. ვაჭერთ Next კლავიშს. გაიხსნება ნახ. 19.5 და ნახ. 19.6-ზე ნაჩვენები ფანჯრები. მომდევნო ფანჯარაში გამოჩნდება Book1.xls ცხრილის ფურცლების სახელები (ნახ. 19.11). ავირჩიოთ 'Sheet1$' ფურცელი და დავაჭიროთ Next კლავიშს. გაიხსნება ნახ. 19.7 და ნახ. 19.9-ზე ნაჩვენები ფანჯრები. Finish კლავიშზე დაჭერა იწყებს იმპორტის ოპერაციას. ბოლოს ვხურავთ უკანასკნელ ფანჯარას.

მონაცემების ექსპორტი SQL სერვერიდან

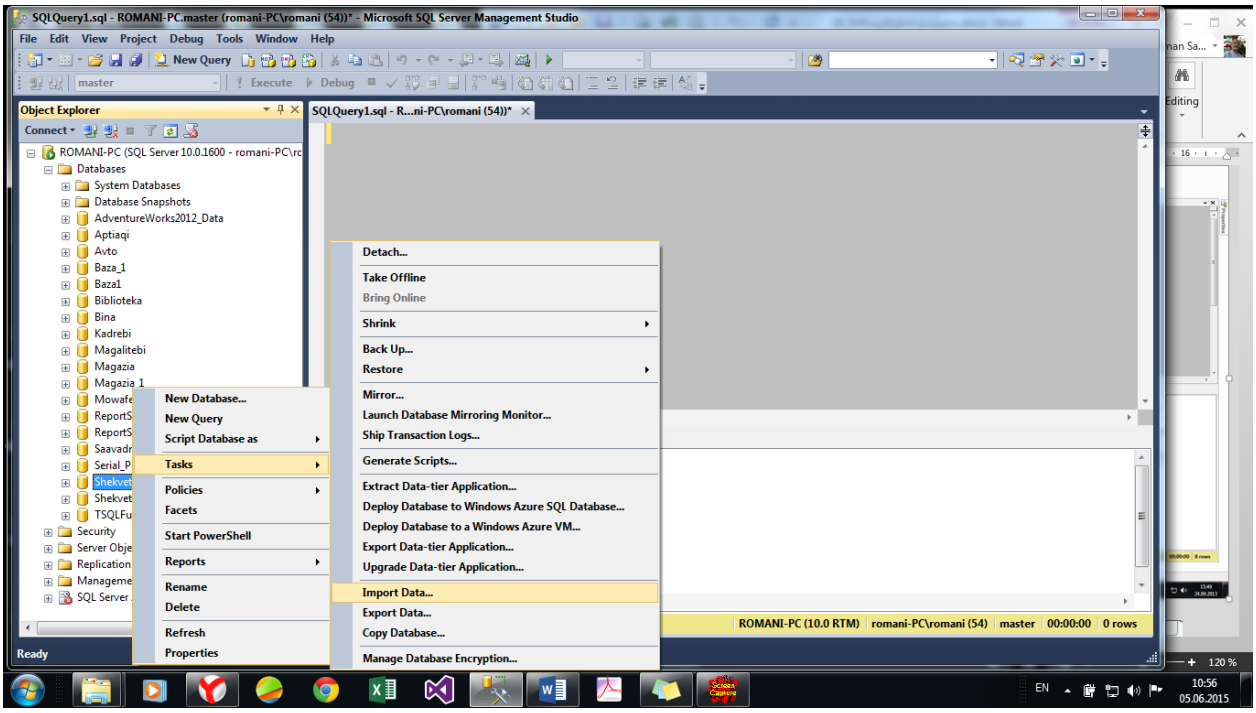
ამ განყოფილებაში განვიხილავთ მონაცემების ექსპორტს SQL სერვერიდან Access და Excel სისტემებში.

მონაცემების ექსპორტი Access სისტემაში

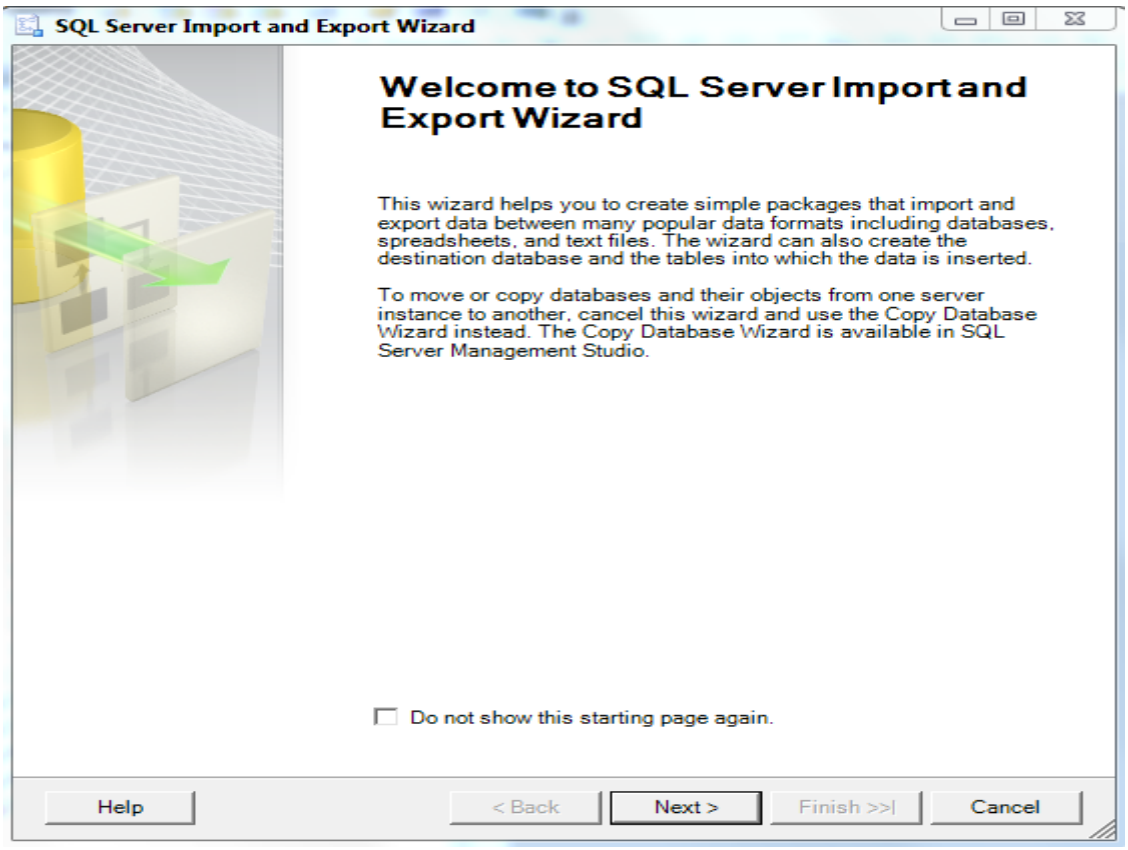
SQL სერვერის რომელიმე მონაცემთა ბაზიდან, მაგალითად, Shemkveti მონაცემთა ბაზიდან Access სისტემაში მონაცემების ექსპორტისათვის Shemkveti ბაზის სახელზე ვხსნით კონტექსტურ მენიუს და ვასრულებთ Tasks ქვემენიუს Export Data ბრძანებას (ნახ. 19.12). გაიხსნება SQL Server Import and Export Wizard ფანჯარა (ნახ. 19.2). ვაჭერთ Next კლავიშს. გახსნილ ფანჯარაში (ნახ. 19.13) ვირჩევთ მონაცემების წყაროს, საიდანაც უნდა შესრულედეს მონაცემების გადაწერა. Database ველში ჩანს Shemkveti მონაცემთა ბაზა, საიდანაც შესრულდება ცხრილების გადაწერა. ვირჩევთ აუტენტიფიცირების რეჟიმს და ვაჭერთ Next კლავიშს. მომდევნო ფანჯარაში (ნახ. 19.14) ვირჩევთ მონაცემების მიმღებს. Destination სიიდან ვირჩევთ Microsoft Access ელემენტს. Browse კლავიშით ვირჩევთ საჭირო ფაილს, მაგალითად, db5.mdb. Next კლავიშზე დაჭერის შემდეგ გაიხსნება ნახ. 19.5-ზე ნაჩვენები ფანჯარა. ისევ ვაჭერთ Next კლავიშს. მომდევნო ფანჯარაში (ნახ. 19.15) გამოჩნდება Shemkveti მონაცემთა ბაზის ობიექტების სია. მოვნიშნოთ Personali, Shemkveti და Xelshekruleba ცხრილები. მომდევნო ფანჯარაში გამოჩნდება ამორჩეული ცხრილები და მათი სვეტები. Next კლავიშზე დაჭერის შემდეგ გაიხსნება ნახ.19.7-19.9-ზე ნაჩვენები ფანჯრები. Finish კლავიშზე დაჭერის შემდეგ დაიწყება მონაცემების ექსპორტი. ექსპორტის შემდეგ გავხსნათ db5.mdb ფაილი. Tables განყოფილებაში დავინახავთ სამივე გადაწერილ ცხრილს.

მონაცემების ექსპორტი Excel სისტემაში

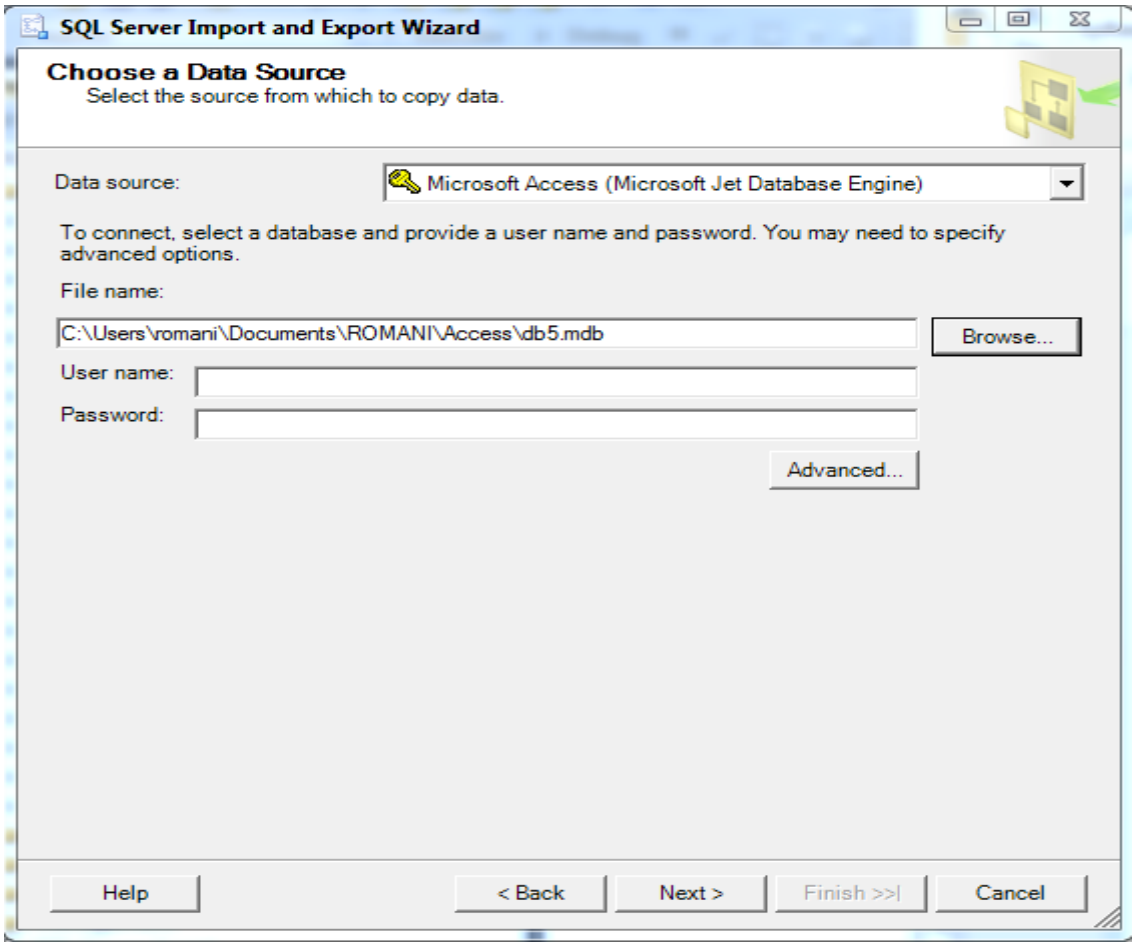
SQL სერვერის რომელიმე მონაცემთა ბაზიდან, მაგალითად, Shemkveti მონაცემთა ბაზიდან Excel სისტემაში მონაცემების ექსპორტისათვის უნდა გავხსნათ ნახ. 19.14-ზე ნაჩვენები ფანჯარა. Destination სიიდან ვირჩევთ Microsoft Excel ელემენტს. Browse კლავიშით ვირჩევთ საჭირო ფაილს, მაგალითად, Book1.xls (ნახ. 19.17). Next კლავიშზე დაჭერის შემდეგ გაიხსნება ნახ. 19.5-ზე ნაჩვენები ფანჯარა. მომდევნო ფანჯარაში გამოჩნდება Shemkveti მონაცემთა ბაზის ობიექტების სია (ნახ. 19.15). მოვნიშნოთ Per_1 ცხრილი (ნახ. 19.18). შემდეგ გაიხსნება ნახ.19.16, 19.7-19.8-ზე ნაჩვენები ფანჯრები. Finish კლავიშზე დაჭერის შემდეგ დაიწყება მონაცემების ექსპორტი. ექსპორტის შემდეგ გავხსნათ Book1.xls ფაილი. დავინახავთ, რომ თითოეული ცხრილი მოთავსებული იქნება ცალკე ფურცელზე.



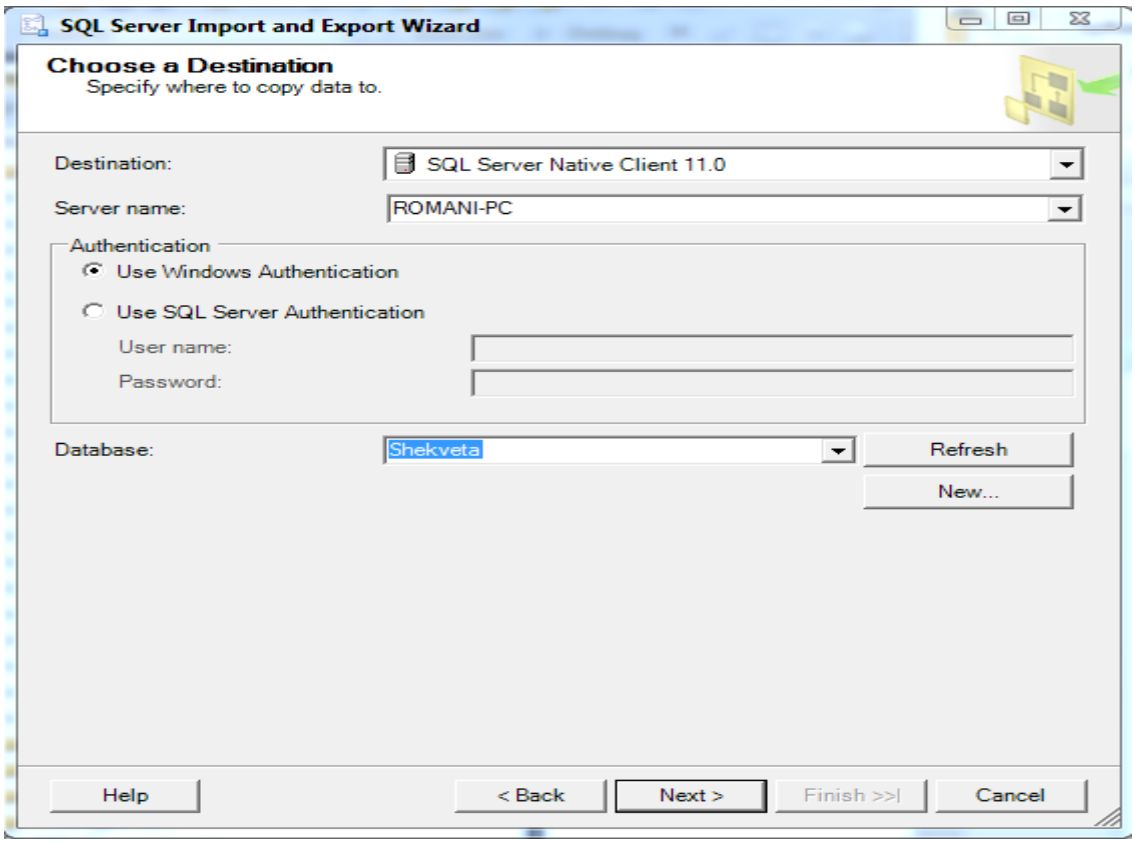
ნახ. 19.1.



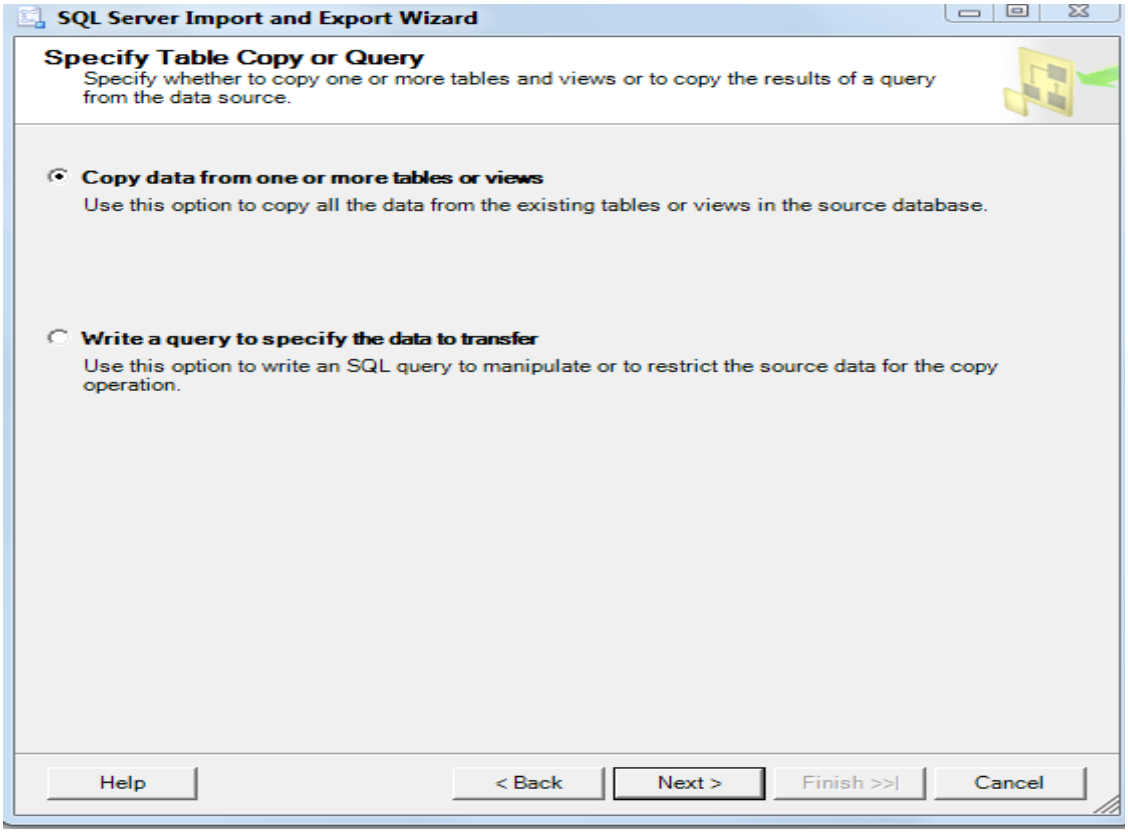
ნახ. 19.2.



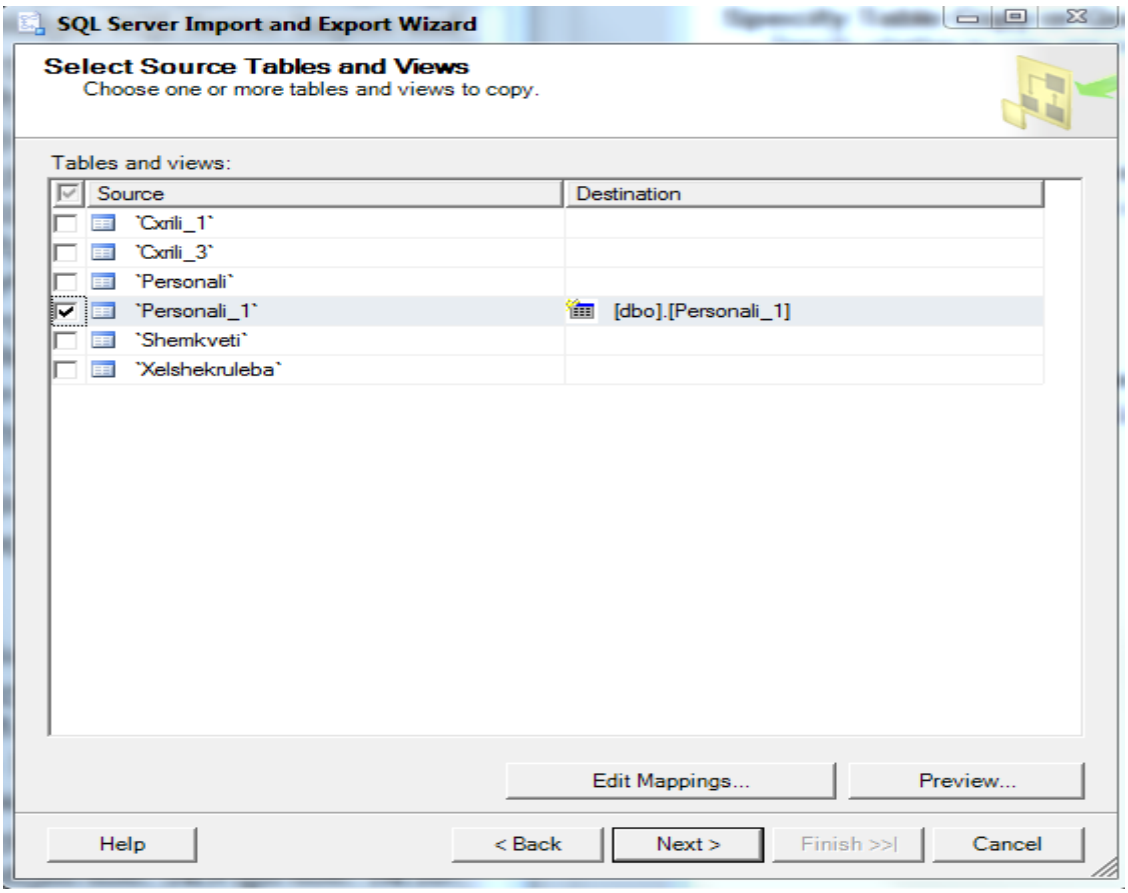
ნახ. 19.3.



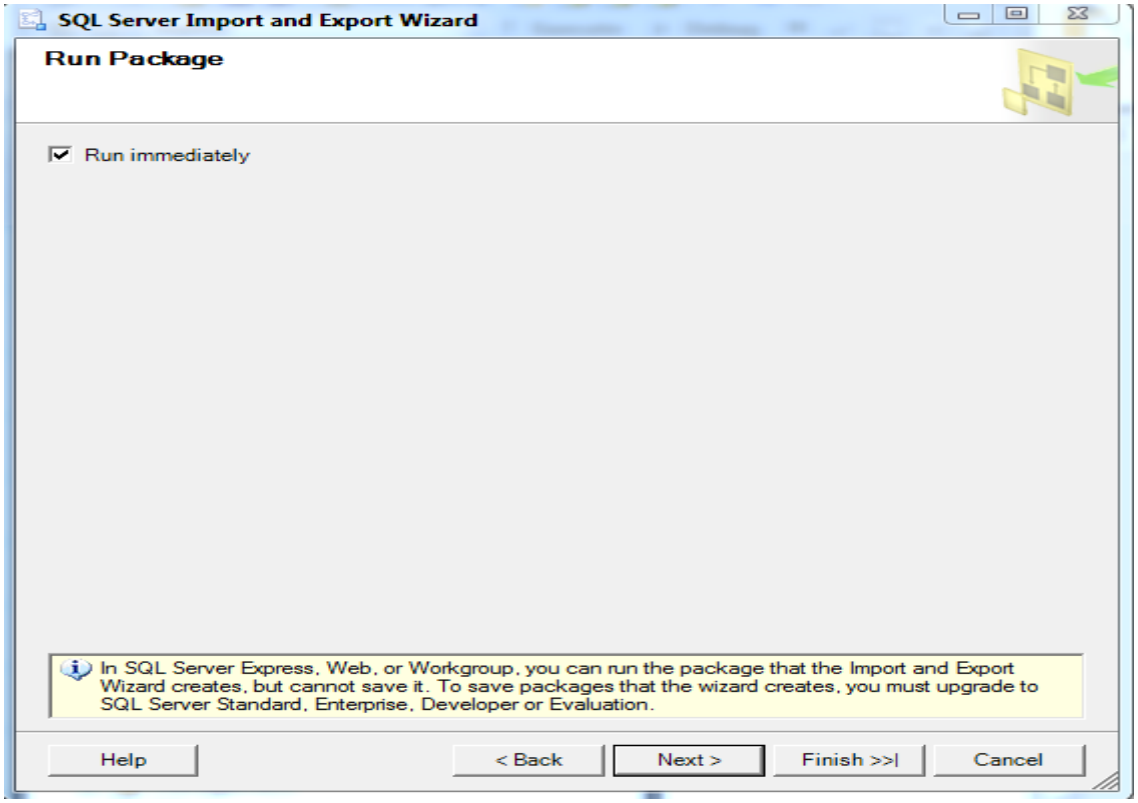
ნახ. 19.4.



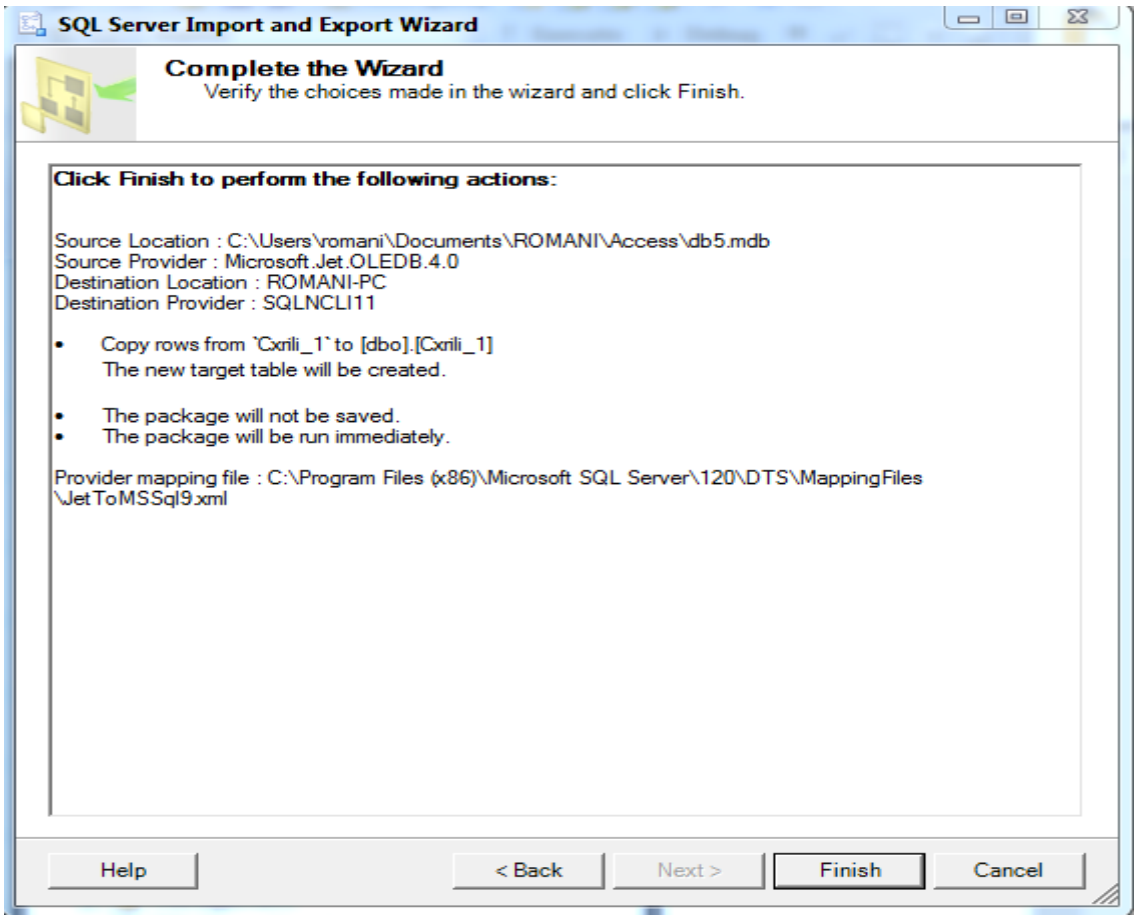
ნახ. 19.5.



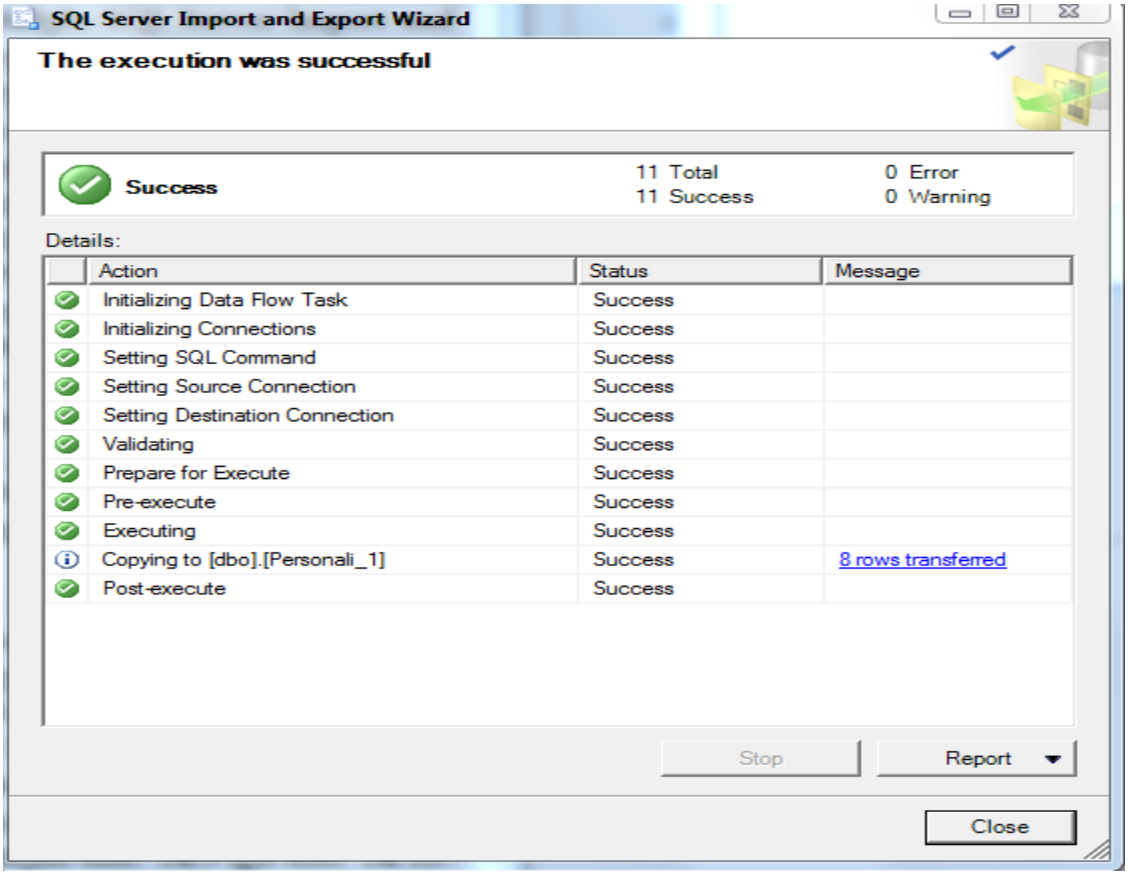
ნახ. 19.6.



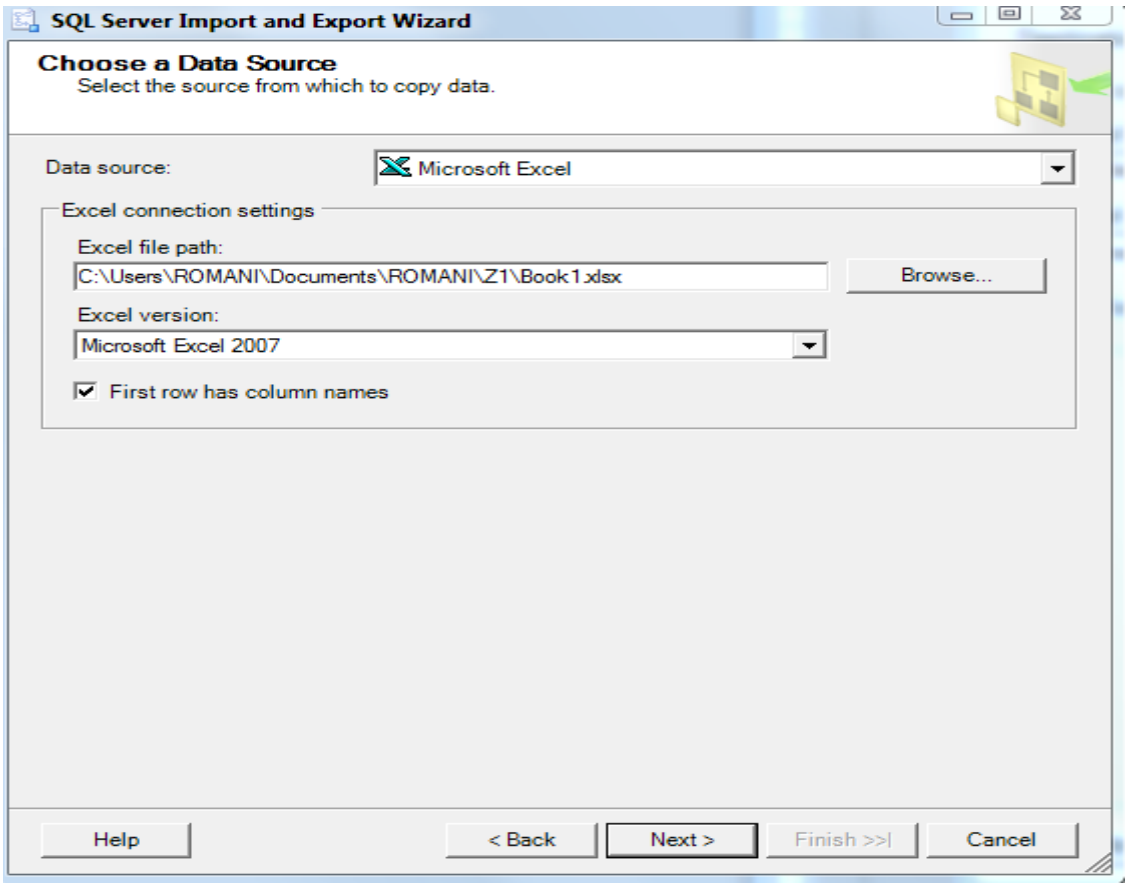
ნახ. 19.7.



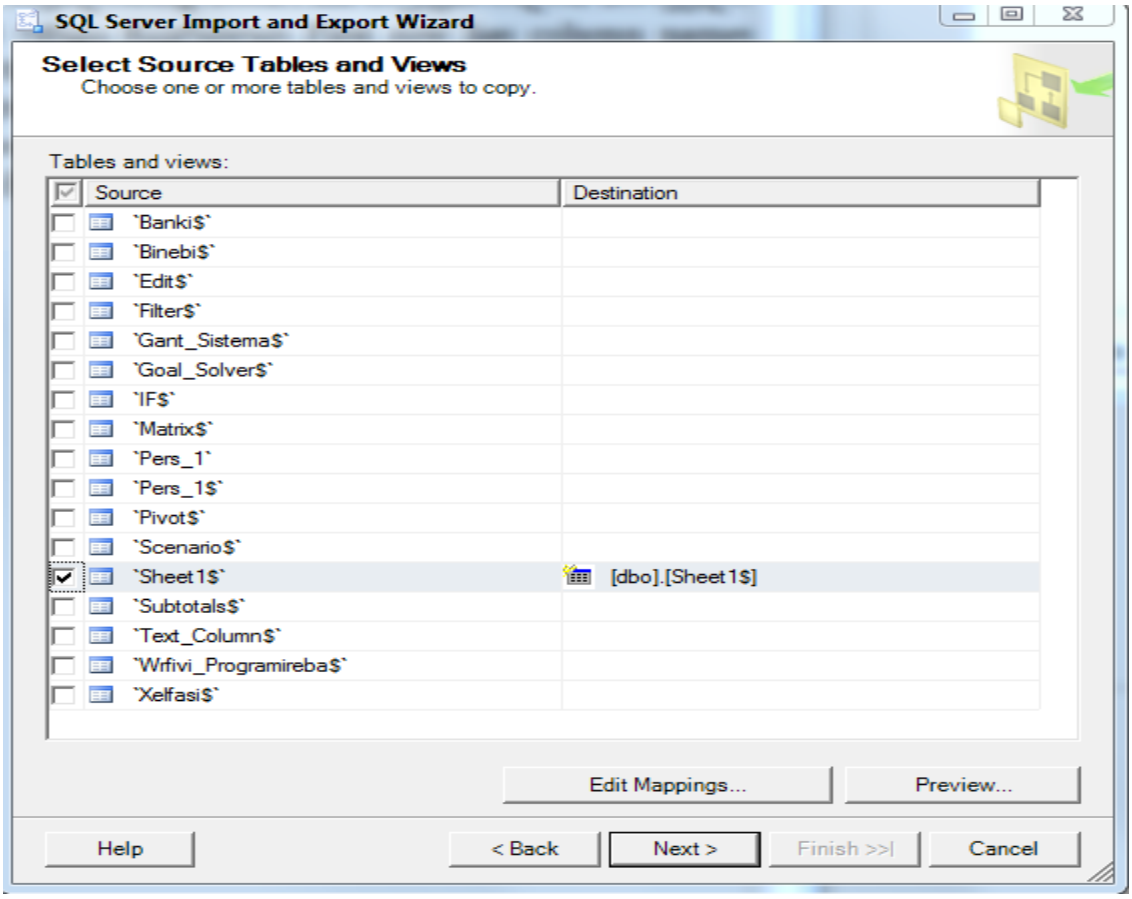
ნახ. 19.8.



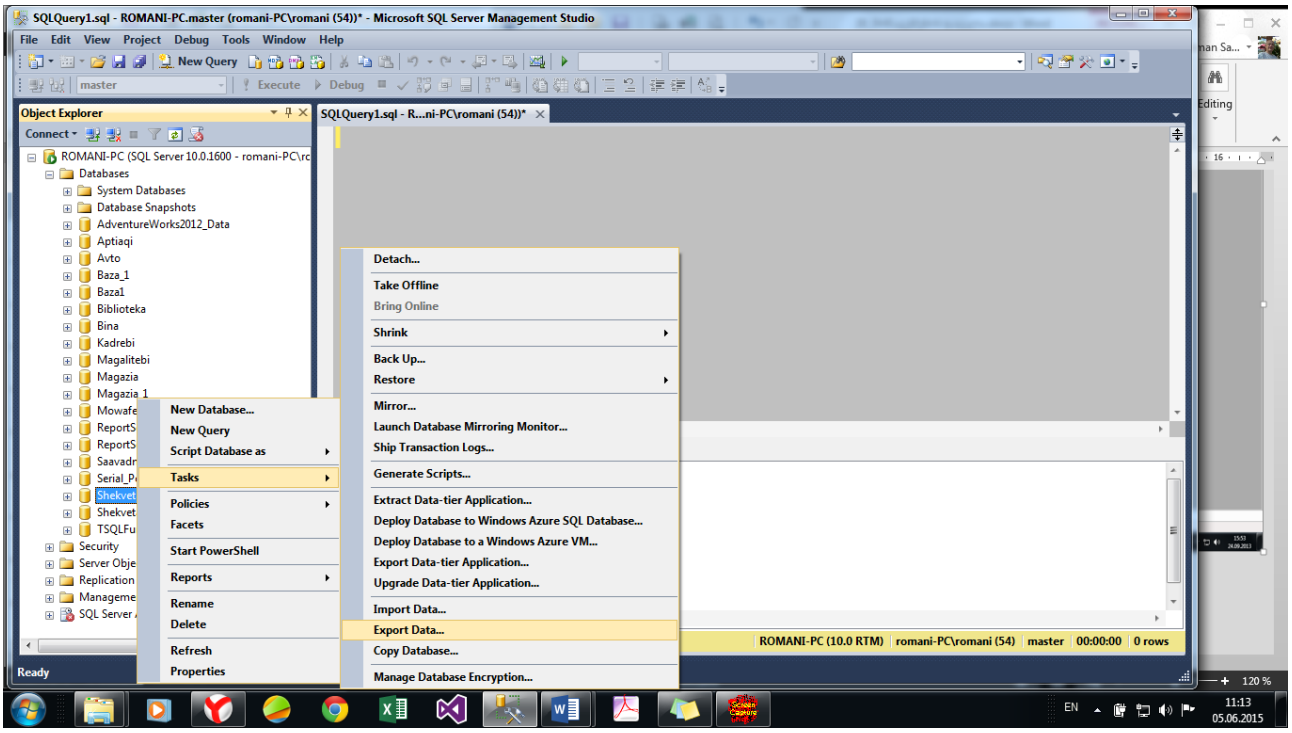
ნახ. 19.9.



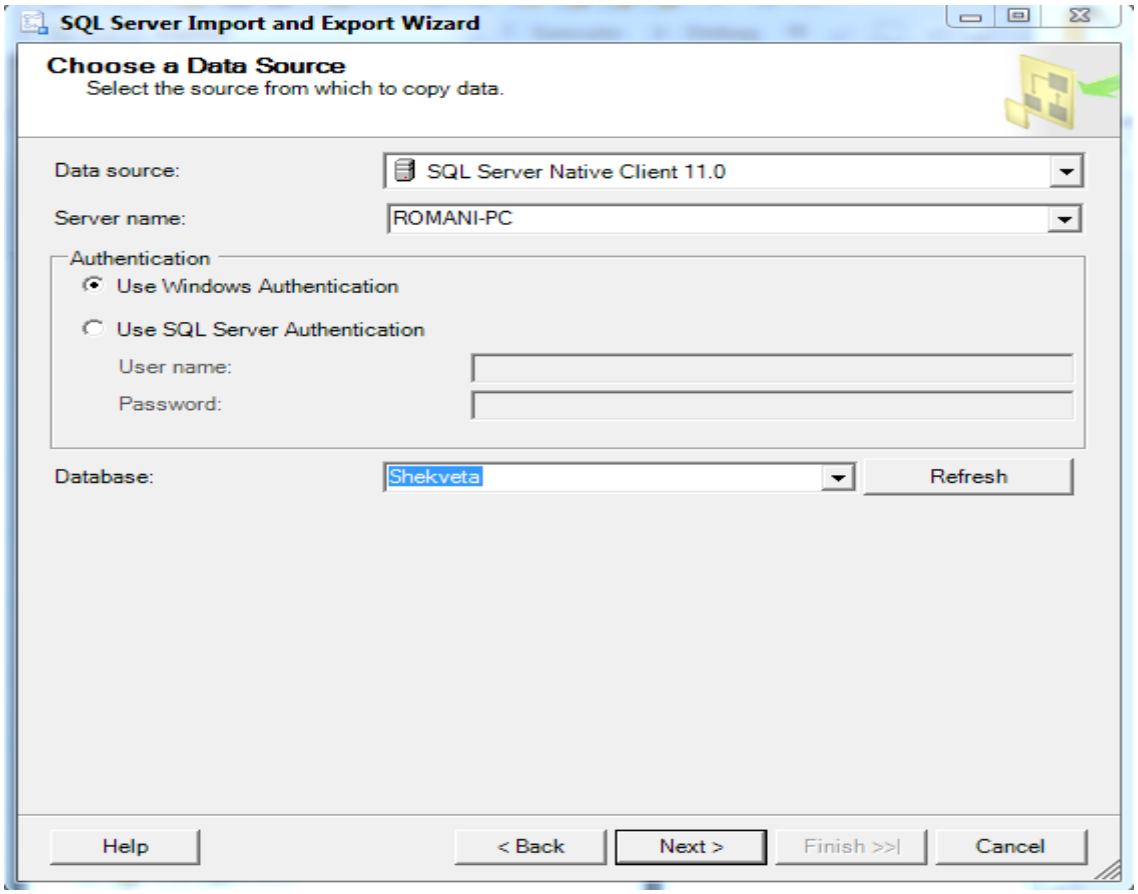
ნახ. 19.10.



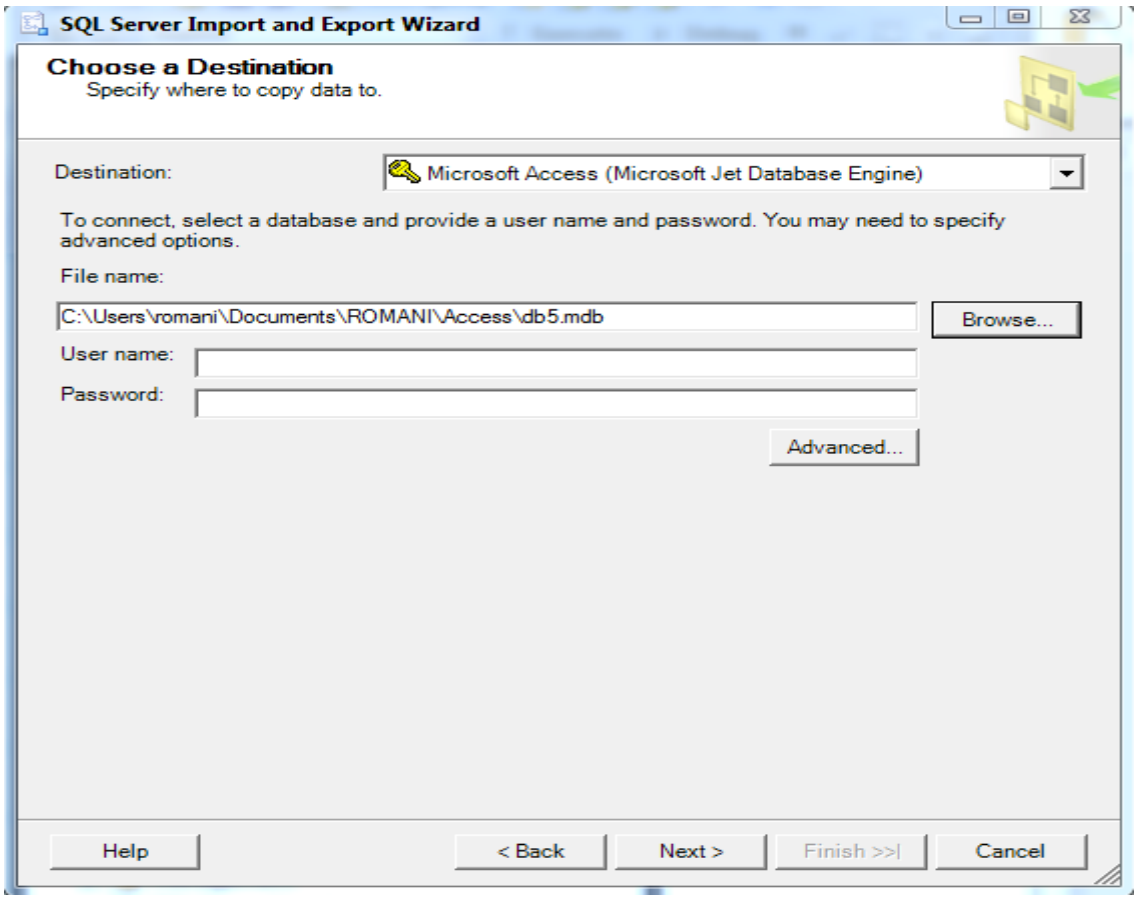
ნახ. 19.11.



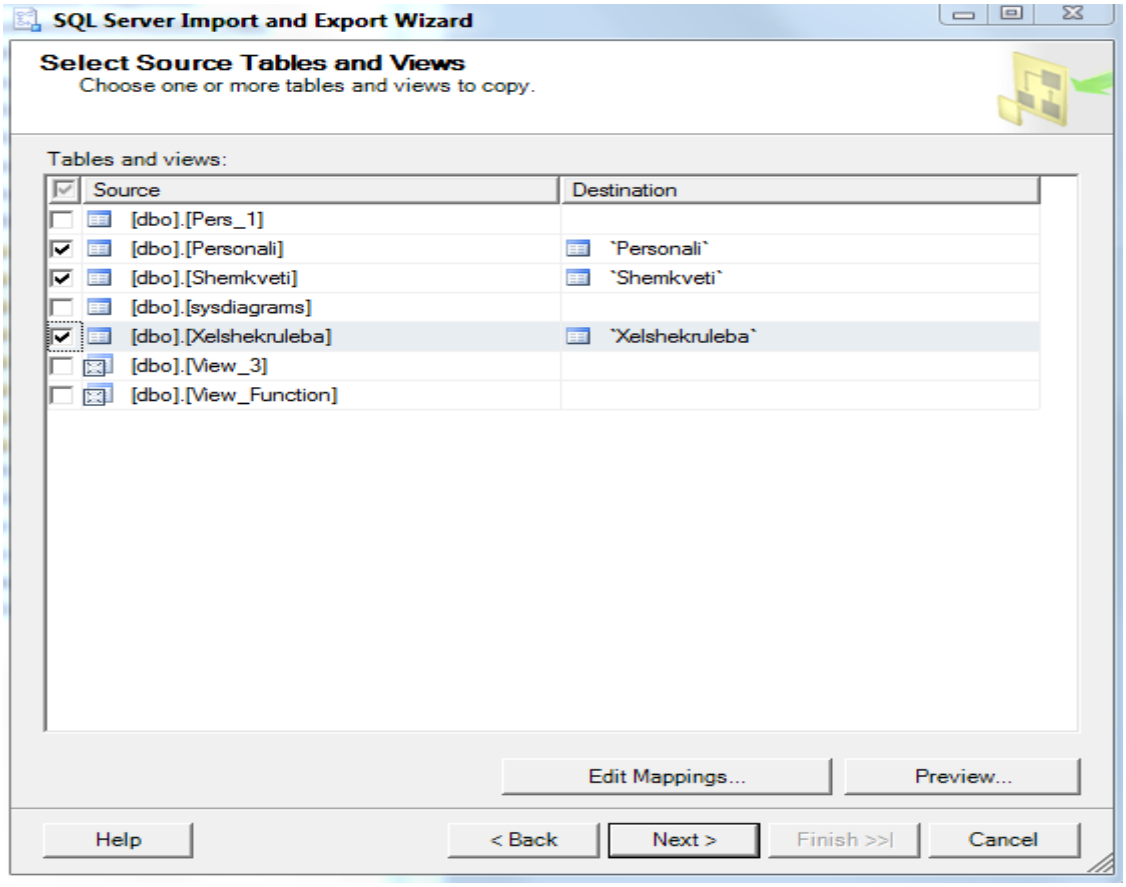
ნახ. 19.12.



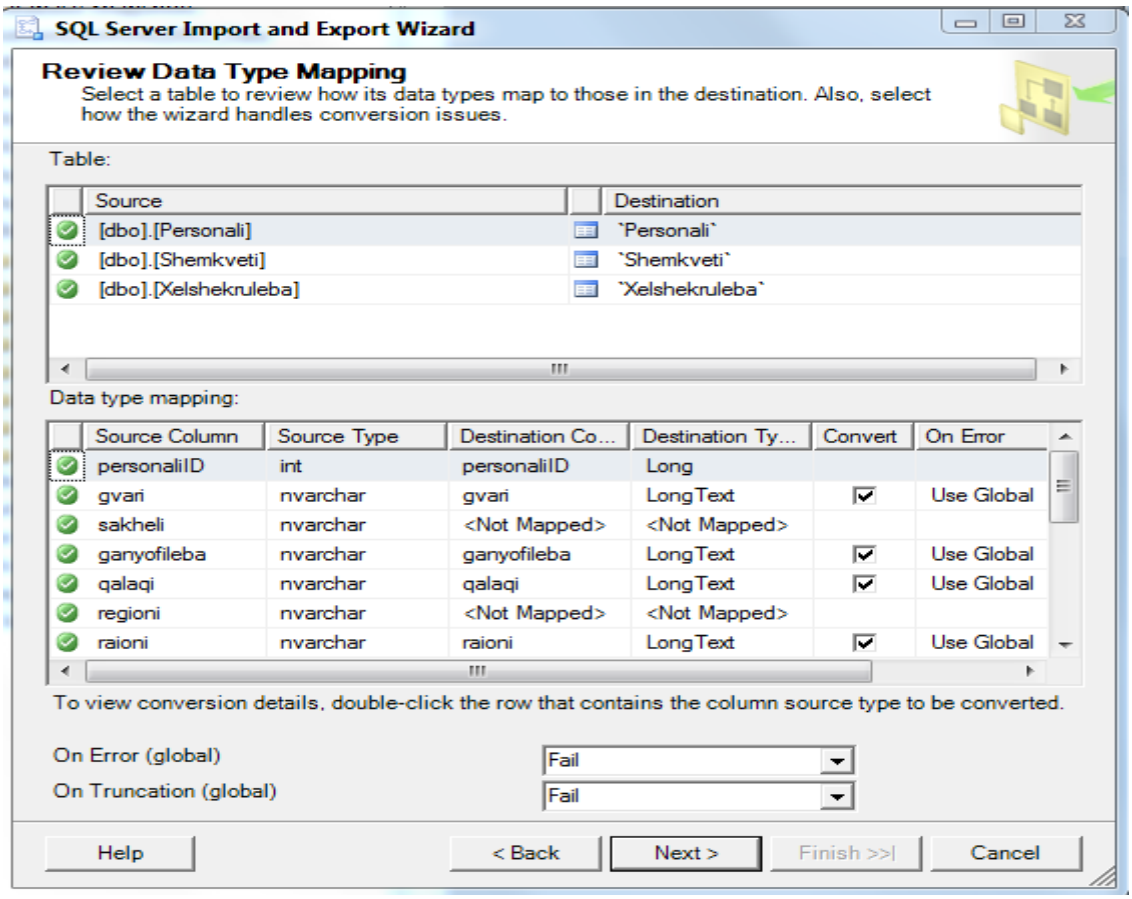
ნახ. 19.13.



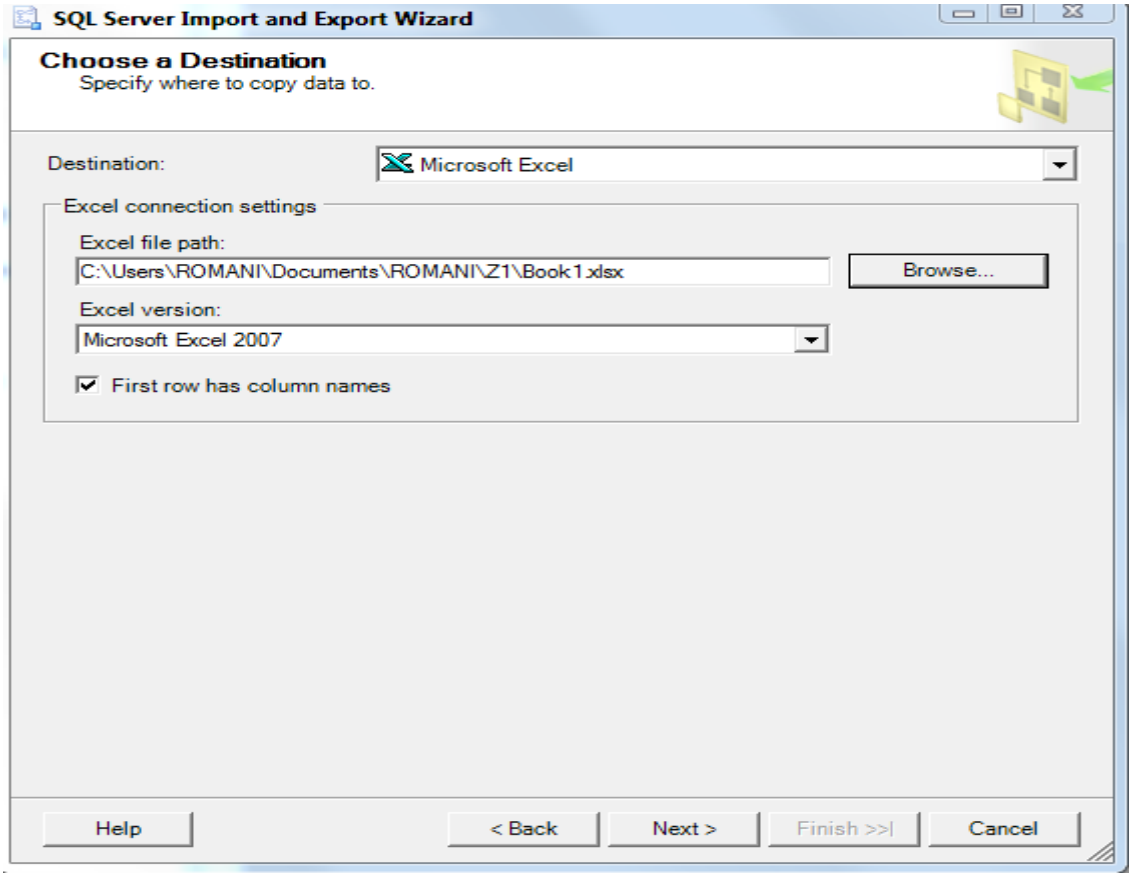
ნახ. 19.14.



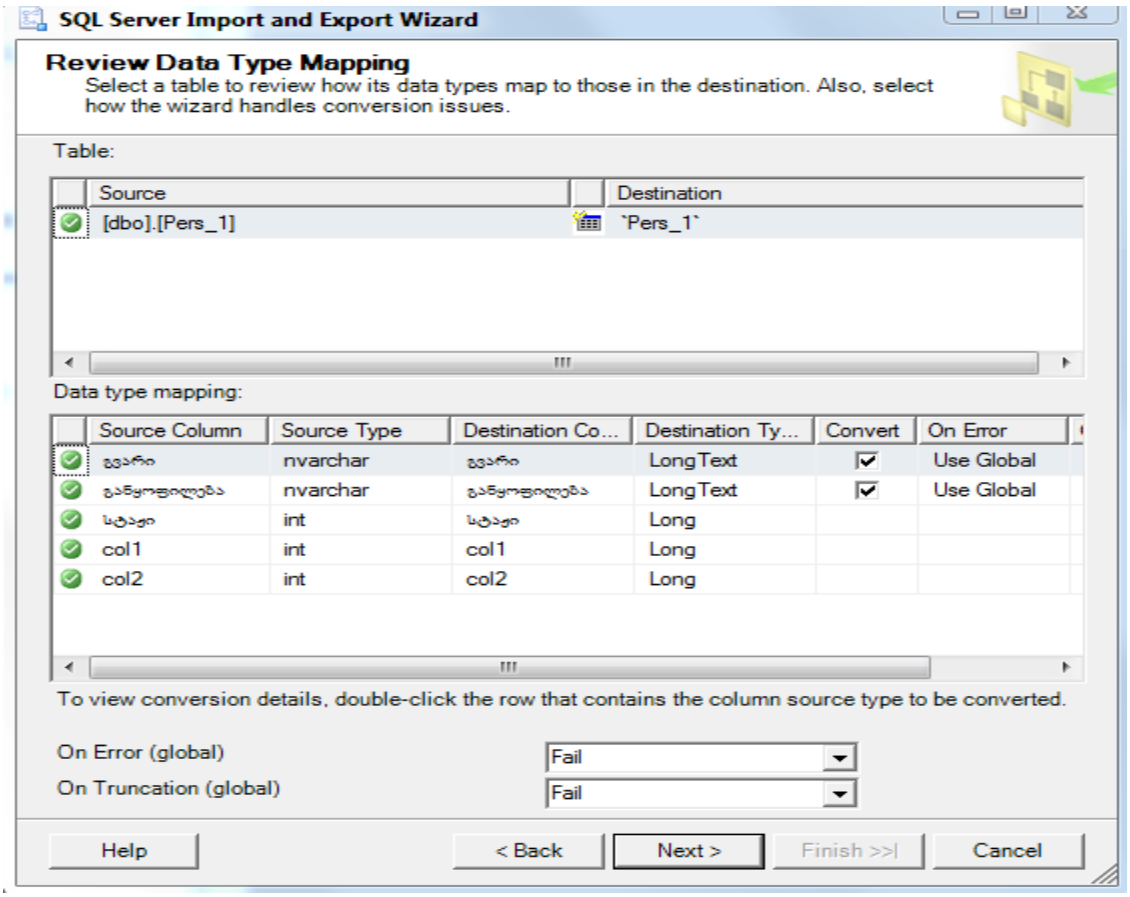
ნახ. 19.15.



ნახ. 19.16.



ნახ. 19.17.


SQL Server Import and Export Wizard
Review Data Type Mapping
Select a table to review how its data types map to those in the destination. Also, select how the wizard handles conversion issues.
Table:
Source
Destination
[dbo].[Pers_1]
`Pers_1`
Data type mapping:
Source Column
Source Type
Destination Co...
Destination Ty...
Convert
On Error
გვარი
nvarchar
გვარი
LongText
Use Global
განყოფილება
nvarchar
განყოფილება
LongText
Use Global
სტაჟი
int
სტაჟი
Long
col1
int
col1
Long
col2
int
col2
Long
To view conversion details, double-click the row that contains the column source type to be converted.
On Error (global)
Fail
On Truncation (global)
Fail
Help
< Back
Next >
Finish >>|
Cancel

ნახ. 19.18.

თავი 20. უსაფრთხოების სისტემა

უფლებამოსილების გამიჯვნის წესები

ხშირად მონაცემთა ბაზასთან რამდენიმე მომხმარებელი მუშაობს. ამ დროს აუცილებელია მასთან მიმართვის უფლებების გამიჯვნა. ეს მოითხოვს უსაფრთხოების სისტემის სწორად დაგეგმვას. დაგეგმვის პროცესში ის მომხმარებლები უნდა განისაზღვროს, რომლებსაც მონაცემთა ბაზასთან მიმართვის უფლება ექნებათ და ის მოქმედებები, რომელთა შესრულების უფლება მათ ექნებათ.

უსაფრთხოების სისტემის პირველი ეტაპი მდგომარებს იმ მომხმარებლების განსაზღვრაში, რომლებსაც მონაცემთა ბაზასთან მიმართვის უფლება ექნებათ. მაგრამ, ჯერ მომხმარებლებს უნდა ჰქონდეთ სერვერთან მიმართვის უფლება. ამისათვის შეგვიძლია სერვერის ან Windows NT-ის უსაფრთხოების სისტემა გამოვიყენოთ. პირველ შემთხვევაში მომხმარებელი უნდა დავარეგისტრიროთ სერვერზე ანუ მისთვის უნდა შევქმნათ სააღრიცხვო ჩანაწერი. მეორე შემთხვევაში მომხმარებელი უნდა დავარეგისტრიროთ Windows NT სისტემაში ანუ მისთვის შევქმნათ სააღრიცხვო ჩანაწერი დომენში. უნდა გვახსოვდეს, რომ თუ მომხმარებელს აქვს სერვერთან მიმართვის უფლება, ეს არ ნიშნავს იმას, რომ მას აქვს რომელიმე მონაცემთა ბაზასთან ან მის ობიექტთან მიმართვის უფლებაც.

უსაფრთხოების სისტემის დაგეგმვის მეორე ეტაპი იმ მოქმედებების განსაზღვრაში მდგომარეობს, რომელთა შესრულების უფლებაც აქვს კონკრეტულ მომხმარებელს მონაცემთა ბაზაში. მონაცემთა ბაზასთან და მის ობიექტებთან სრული მიმართვის უფლება აქვს ადმინისტრატორს. მეორე ადამიანი ადმინისტრატორის შემდეგ არის ობიექტის მფლობელი (dbo, data base owner). ობიექტს მონაცემთა ბაზაში შექმნის დროს ენიშნება მფლობელი, რომელსაც შეუძლია ამ ობიექტთან მიმართვის უფლებები განსაზღვროს. მომხმარებლების მესამე კატეგორიას აქვს მიმართვის ის უფლებები, რომლებიც მათ ადმინისტრატორმა ან ობიექტის მფლობელმა მისცა.

მომხმარებლებისთვის მისაცემი უფლებები ჭკვიანურად უნდა დავგეგმოთ. მაგალითად, არ არის აუცილებებლი, რომ ფირმის დირექტორს ან რიგით თანამშრომელს მივცეთ ხელფასების შესახებ მონაცემების შემცველ ცხრილში მონაცემების შეცვლის უფლება. მათ შეგვიძლია მივცეთ ახალი ინფორმაციის ჩამატების ნებართვა. არასწორი მონაცემების შეტანის შემთხვევაში ფირმა სერიოზულად არ დაზარალდება. მაგრამ, თუ მათ მივცემთ მონაცემების შეცვლის ან წაშლის უფლებას, მაშინ ამან ფირმას შეიძლება დიდი ფინანსური ზიანი მიაყენოს.

სწორად დაგეგმილმა უსაფრთხოების სისტემამ მომხმარებელს არ უნდა მისცეს უფლება შეასრულოს ის მოქმედებები, რომლებიც მის უფლებამოსილებას სცილდება. შეგვიძლია ავკრძალოთ იმ მონაცემების წაშლა, რომელთა შენახვის ვადა არ გასულა, ახალ თანამშრომლებს მივცეთ მონაცემების მხოლოდ წაკითხვის უფლება. შემდგომში მათ შეგვიძლია გავუფართოვოთ უფლებები და მივცეთ მონაცემების შეცვლის, წაშლის ან ჩამატების უფლება.

ადმინისტრატორმა თვალყური უნდა ადევნოს, აგრეთვე, თანამშრომლების გადასვლას ერთი განყოფილებიდან მეორეში. თანამდებობის შეცვლა დაუყოვნებლივ უნდა აისახოს მიმართვის უფლებებზე. დროულად უნდა წავშალოთ ის მომხმარებლები, რომლებიც აღარ მუშაობენ ფირმაში. თუ თანამშრომელი შვებულებაში ან ხანგრძლივ მივლინებაშია, მაშინ მისი სააღრიცხვო ჩანაწერი დროებით უნდა დავბლოკოთ.

პაროლების შექმნისას უნდა დავიცვათ სტანდარტული მოთხოვნები. სასურველია, რომ პაროლი სიმბოლოების გარდა ციფრებსაც შეიცავდეს. პაროლი არ უნდა იყოს ოჯახის წევრის სახელი, დაბადების წელი, პასპორტის ნომერი და ა.შ. პაროლს უნდა ჰქონდეს მოქმედების

შეზღუდული ვადა, რომლის გასვლის შემდეგ მომხმარებელმა ის უნდა შეცვალოს. მაქსიმალური უსაფრთხოების მისაღწევად უნდა გამოვიყენოთ მომხმარებლების აუტენტიფიცირება Windows NT-ის საშუალებებით, რადგან ამ ოჯახის ოპერაციულ სისტემებს დაცვის ეფექტური საშუალებები აქვს.

სერვერის უსაფრთხოების სისტემის არქიტექტურა

სერვერის უსაფრთხოების სისტემა ემყარება *მომხმარებლებსა* და *სააღრიცხვო ჩანაწერებს*. მომხმარებლები გადიან შემოწმების ორ ეტაპს. პირველ ეტაპზე ხდება მომხმარებლის იდენტიფიცირება სააღრიცხვო ჩანაწერის სახელისა და პაროლის მიხედვით, ე.ი. აუტენტიფიცირება. თუ სახელი და პაროლი სწორადაა შეტანილი, მაშინ მომხმარებელი მიეერთდება სერვერს. სერვერთან მიერთება ანუ დარეგისტრირება მომხმარებელს არ აძლევს მონაცემთა ბაზასთან ავტომატურად მიმართვის უფლებას. თითოეული მონაცემთა ბაზისთვის სარეგისტრაციო სახელი (სააღრიცხვო ჩანაწერი, login) უნდა აისახოს მონაცემთა ბაზის მომხმარებლის სახელში (user). მეორე ეტაპზე, მომხმარებლისთვის (როგორც მონაცემთა ბაზის მომხმარებლისათვის) გაცემული უფლებების საფუძველზე, მისი სარეგისტრაციო სახელი (login) შესაბამის მონაცემთა ბაზასთან მიმართვის უფლებას იღებს. სხვადასხვა მონაცემთა ბაზაში ერთი და იმავე მომხმარებლის სარეგისტრაციო სახელს შეიძლება ჰქონდეს ერთნაირი ან სხვადასხვა სახელები (მომხმარებლების სახელები) მიმართვის სხვადასხვა უფლებებით.

მონაცემთა ბაზებთან მიმართვისთვის პროგრამა-დანართებსაც უნდა ჰქონდეს შესაბამისი უფლებები. ჩვეულებრივ, პროგრამა-დანართებს ისეთივე უფლებები ეძლევა, როგორიც მათ გამშვებ მომხმარებლებს აქვს. მაგრამ ზოგიერთი პროგრამა-დანართის მუშაობისთვის საჭიროა მიმართვის უფლებების ისეთი ნაკრები, რომელიც არ არის დამოკიდებული მომხმარებლის მიმართვის უფლებებზე. ასეთი უფლებების მინიჭება შესაძლებელია *როლების* (*ჯგუფების*) გამოყენებით.

სერვერის დონეზე უსაფრთხოების სისტემის განხილვისას შემდეგი ცნებები გამოიყენება:

- აუტენტიფიცირება (authentication);
- სააღრიცხვო ჩანაწერი (login);
- სერვერის ფიქსირებული როლები (fixed server roles).

მონაცემთა ბაზის დონეზე უსაფრთხოების სისტემის განხილვისას შემდეგი ცნებები გამოიყენება:

- მონაცემთა ბაზის მომხმარებელი (database user);
- მონაცემთა ბაზის ფიქსირებული როლი (fixed database role);
- მონაცემთა ბაზის მომხმარებლის როლი (users database role);
- პროგრამა-დანართის როლი (application role).

აუტენტიფიცირების რეჟიმები

მომხმარებლების აუტენტიფიცირების ორი რეჟიმი არსებობს:

- აუტენტიფიცირების რეჟიმი Windows NT-ის საშუალებებით (Windows NT Authentication);
- აუტენტიფიცირების შერეული რეჟიმი (Windows NT Authentication and SQL Server Authentication).

შერეული რეჟიმი საშუალებას გვაძლევს დავრეგისტრირდეთ როგორც სერვერის, ისე Windows NT-ის საშუალებებით.

აუტენტიფიცირების რეჟიმის არჩევის დროს უზრუნველყოფილი უნდა იყოს მაქიმალური უსაფრთხოება და ადმინისტრირების სიმარტივე. ასე, თუ მცირე ზომის ფირმა გვაქვს და ქსელისა და მონაცემთა ბაზების ადმინისტრატორის ფუნქციებს ერთი ადამიანი ითავსებს, მაშინ უმჯობესია გამოვიყენოთ აუტენტიფიცირების რეჟიმი Windows NT-ის საშუალებებით. თუ ფირმა დიდია და ქსელისა და მონაცემთა ბაზების ადმინისტრატორის ფუნქციებს სხვადასხვა ადამიანი ასრულებს, მაშინ უმჯობესია, გამოვიყენოთ აუტენტიფიცირების შერეული რეჟიმი. საწინააღმდეგო შემთხვევაში მონაცემთა ბაზის ადმინისტრატორს მოუწევს ხშირი მიმართვები ქსელის ადმინისტრატორთან თავისი ფუნქციების შესასრულებლად, კერძოდ, ახალი მომხმარებლის შესაქმნელად, პაროლის შესაცვლელად, სხვა ჯგუფში მომხმარებლის გადასაყვანად და ა.შ., რაც ბუნებრივია, არასწორია. მეორე მხრივ, თუ გამოიყენება აუტენტიფიცირების რეჟიმი Windows NT-ის საშუალებებით, მაშინ მონაცემთა ბაზის ადმინისტრატორს შეეძლება არსებული სააღრიცხვო ჩანაწერების გამოყენება.

Windows NT-ის აუტენტიფიცირების რეჟიმი

როცა მომხმარებელი სერვერს უერთდება Windows NT-ის სააღრიცხვო ჩანაწერის გამოყენებით, ამ დროს მყარდება *სანდო შეერთება*. აუტენტიფიცირება Windows NT-ის საშუალებებით გარკვეულ უპირატესობებს იძლევა. მომხმარებლებზე ავტომატურად ვრცელდება უსაფრთხოების პოლიტიკის წესები. მაგალითად, ავტომატურად მოწმდება პაროლის მინიმალური სიგრძე და მისი მოქმედების ვადა. თუ მომხმარებელი რამდენიმეჯერ არასწორად შეიტანს პაროლს, მაშინ გარკვეული დროით ხდება მისი სააღრიცხვო ჩანაწერის დაბლოკვა. გარდა ამისა, რეგისტრირების დროს ხდება პაროლების დაშიფვრა. ყოველივე ეს, სერვერის მონაცემების დაცულობას ზრდის.

როგორც ცნობილია, ოპერაციული სისტემის სააღრიცხვო ჩანაწერები (logins) შეიცავენ ყველა მონაცემს მომხმარებლების შესახებ: სახელს, პაროლს, ჯგუფებში წევრობას და ა.შ. თითოეულ სააღრიცხვო ჩანაწერს *უნიკალური იდენტიფიკატორი* (login ID) ანუ *უსაფრთხოების იდენტიფიკატორი* (Security IDentification, SID) აქვს, რომლის საშუალებით მომხმარებელი ქსელში თავის თავს აიდენტიფიცირებს. იდენტიფიკატორი არის გრძელი თექვსმეტობითი რიცხვი, რომელიც სააღრიცხვო ჩანაწერის შექმნის დროს ოპერაციული სისტემის მიერ შემთხვევითი გზით გენერირდება. ასეთი მიდგომა მომხმარებლების სააღრიცხვო ჩანაწერების გაყალბებას თავიდან გვაცილებს. მომხმარებლის წაშლის შემდეგ, თუ მას განმეორებით შევქმნით იმავე პარამეტრებით, ის მაინც ვერ შეძლებს მიმართოს იმ ობიექტებს, რომლებსაც ის ადრე მიმართავდა, რადგან მას სხვა იდენტიფიკატორი ექნება. ქსელის რესურსებთან მიმართვა კი ამ იდენტიფიკატორის საფუძველზე სრულდება.

Windows NT-ის საშუალებებით აუტენტიფიცირებისას დომენის მომხმარებლის სააღრიცხვო ჩანაწერი (login ID) ინახება სერვერის master მონაცემთა ბაზაში, ხოლო ინფორმაცია მომხმარებლის სახელის, პაროლის და ა.შ. შესახებ კი - დომენის მონაცემთა ბაზაში. ინფორმაცია მომხმარებლის სააღრიცხვო ჩანაწერისა და Windows NT-ის ჯგუფებში წევრობის შესახებ სერვერის მიერ წაიკითხება დომენის უსაფრთხოების სისტემის მონაცემთა ბაზიდან მომხმარებლის მიერთების დროს.

ამრიგად, Windows NT-ის საშუალებებით აუტენტიფიცირების რეჟიმში სერვერის სტანდარტული sysadmin ან securityadmin როლის წევრმა სერვერისთვის უნდა მიუთითოს თუ Windows NT-ის რომელ მომხმარებლებს ან ჯგუფებს აქვთ სერვერთან მიმართვის უფლება. Windows NT-ის საშუალებებით აუტენტიფიცირების დროს მომხმარებელს არ სჭირდება სახელისა და პაროლის მითითება სერვერთან მიმართვის მიზნით, რადგან მომხმარებლის სახელისა და პაროლის შემოწმებას Windows NT-ის დომენის კონტროლერი ასრულებს.

სერვერთან მომხმარებლის მიერთების დროს სერვერი იყენებს სანდო შეერთებას, რომელიც დამყარდება ოპერაციული სისტემის მიერ დომენში მომხმარებლის წარმატებით რეგისტრირების შემდეგ. სერვერი იმ მომხმარებლის სახელს, რომელიც შეერთების დამყარებას ცდილობს, Master მონაცემთა ბაზის syslogins სისტემურ წარმოდგენაში ადარებს იმ სახელებს, რომლებისთვისაც მიმართვა ნებადართულია. პოვნისას მიმართვა ნებადართული იქნება, საწინააღმდეგო შემთხვევაში, ძებნა გრძელდება იმ ჯგუფებში, რომლებსაც ეს მომხმარებელი ეკუთვნის. თუ სახელი ვერ მოიძებნა, მაშინ სერვერი გასცემს შეტყობინებას შეცდომის შესახებ.

Windows NT-ის საშუალებებით აუტენტიფიცირების რეჟიმი სერვერზე დარეგისტრირებულ მომხმარებელს საშუალებას აძლევს შეასრულოს დაშორებული (შორი) პროცედურები და ობიექტებს სხვა სერვერზე მიმართოს პროცედურების შორი გამოძახების (Remote Procedure Call, RPC) მექანიზმის საშუალებით. ამასთან, გამოიყენება დომენში არსებული მომხმარებლის სააღრიცხვო ჩანაწერი.

თუ მომხმარებელი და სერვერი სხვადასხვა დომენში იმყოფება, მაშინ სერვერთან სხვა დომენის მომხმარებლებისათვის Windows NT-ის აუტენტიფიცირების საშუალებებით მიმართვის უფლების მისაცემად ამ დომენებს შორის *სანდო ურთიერთობები* (trusted relationships) უნდა გავაწყოთ (დავამყაროთ).

აუტენტიფიცირების კონკრეტული რეჟიმის არჩევის დროს უნდა გავითვალისწინოთ ის, რომ სერვერის ადმინისტრატორს უნდა ჰქონდეს ადმინისტრაციული უფლებები Windows NT-ის ქსელში. საწინააღმდეგო შემთხვევაში, მას თავისი ფუნქციების შესასრულებლად ხშირად მოუწევს ქსელის ადმინისტრატორთან მიმართვა.

სერვერის აუტენტიფიცირების რეჟიმი

სერვერთან შეერთების დასამყარებლად, რომელიც იმყოფება დომენში, რომელთანაც არ არის დამყარებული სანდო ურთიეთობები, შეგვიძლია გამოვიყენოთ სერვერის აუტენტიფიცირების რეჟიმი. ეს რეჟიმი გამოიყენება, აგრეთვე მაშინ, როცა არ არის დომენში დარეგისტრირების საშუალება, მაგალითად, როცა სერვერს ინტერნეტით ვუკავშირდებით.

სერვერის აუტენტიფიცირების გამოყენების შემთხვევაში სერვერთან მიმართვა ხორციელდება სერვერის სააღრიცხვო ჩანაწერების საფუძველზე.

სერვერის საშუალებებით აუტენტიფიცირებისათვის სერვერის sysadmin ან securityadmin სტანდარტული როლის წევრმა მომხმარებლისთვის უნდა შექმნას სააღრიცხვო ჩანაწერი და შეასრულოს მისი კონფიგურირება. სააღრიცხვო ჩანაწერი შეიცავს სააღრიცხვო ჩანაწერის სახელს, სერვერის უნიკალური იდენტიფიკატორსა და პაროლს. ეს ინფორმაცია შენახული იქნება master მონაცემთა ბაზაში. შექმნილ სააღრიცხვო ჩანაწერს კავშირი არ აქვს Windows NT-ის სააღრიცხვო ჩანაწერებთან. ამ რეჟიმში სერვერი თვითონ ამოწმებს სახელისა და პაროლის სისწორეს.

სერვერის აუტენტიფიცირების რეჟიმი შეგვიძლია გამოვიყენოთ Novell NetWare, Unix და ა.შ. მომხმარებლებისთვის, აგრეთვე სერვერთან ინტერნეტით დაკავშირებისას, როცა დომენში რეგისტრირება არ ხდება.

ჩვეულებრივ, სერვერზე სააღრიცხვო ჩანაწერი იქმნება მიმართვის უფლების მისანიჭებლად, მაგრამ შესაძლებელია სააღრიცხვო ჩანაწერი შეიქმნას მიმართვის აკრძალვის მიზნით. ამისათვის უნდა შევქმნათ Windows NT-ის ჯგუფი, ავუკრძალოთ მას სერვერთან მიმართვა და მასში ჩავრთოთ ის მომხმარებლები, რომლებსაც აკრძალული აქვთ სერვერთან მიმართვა.

უსაფრთხოების სისტემის ელემენტები

სერვერის უსაფრთხოების სისტემის ელემენტებია: *სააღრიცხვო ჩანაწერები* (*login*), *მომხმარებლები* (*user*), *როლები* (*role*) და *ჯგუფები* (*group*).

მომხმარებელმა, რომელიც სერვერს უერთდება, უნდა მოახდინოს თავისი თავის აუტენტიფიცირება სააღრიცხვო ჩანაწერის გამოყენებით. აუტენტიფიცირების წარმატებით გავლის შემდეგ მას შეეძლება სერვერთან მიმართვა. მონაცემთა ბაზასთან მიმართვისათვის მომხმარებლის სააღრიცხვო ჩანაწერი (login) აისახება მონაცემთა ბაზის მომხმარებელში (user). მონაცემთა ბაზის მომხმარებელს შეუძლია მიმართოს ამ მონაცემთა ბაზის ნებისმიერ ობიექტს: ცხრილს, წარმოდგენას, შენახულ პროცედურას და ა.შ. მონაცემთა ბაზის მომხმარებელში შეიძლება აისახოს:

– Windows NT-ის სააღრიცხვო ჩანაწერი;
– Windows NT-ის ჯგუფი;
– სერვერის სააღრიცხვო ჩანაწერი.

სააღრიცხვო ჩანაწერის ასეთი ასახვა მონაცემთა ბაზის მომხმარებელში აუცილებელია თითოეული მონაცემთა ბაზისთვის, რომელთანაც მიმართვა სურს მომხმარებელს. ასახვა ინახება მიმდინარე მონაცემთა ბაზის sysusers სისტემურ წარმოდგენაში. ასეთი მიდგომა უზრუნველყოფს უსაფრთხოების მაღალ დონეს, რადგან თავიდან ვიცილებთ მომხმარებლის ავტომატურ მიმართვას სხვა მონაცემთა ბაზებთან. მონაცემთა ბაზების მომხმარებლები, თავის მხრივ, შეგვიძლია გავაერთიანოთ ჯგუფებში ან როლებში უსაფრთხოების სისტემის მართვის გასამარტივებლად.

იმ შემთხვევაში, თუ სააღრიცხვო ჩანაწერი არ აისახება მონაცემთა ბაზის მომხმარებელში, მაშინ მომხმარებელი მაინც შეძლებს მონაცემთა ბაზასთან მიმართვას guest სტუმრის სახელით, თუ ეს მომხმარებელი, ბუნებრივია, არსებობს მონაცემთა ბაზაში. ჩვეულებრივ, guest მომხმარებელს ეძლევა მინიმალური უფლებები მხოლოდ წაკითხვის რეჟიმში.

სქემები

სქემა არის მასში შემავალი ობიექტების კოლექცია, რომელიც ერთ სახელების სივრცეს ქმნის. სახელების სივრცე არის სიმრავლე, რომლის თითოეულ ელემენტს უნიკალური სახელი აქვს. სქემა შეიძლება შეიცავდეს ცხრილებს, წარმოდგენებს, ფუნქციებს, პროცედურებს, ტრიგერებს, ინდექსებსა და მომხმარებლებს. ერთი სქემის შიგნით ობიექტებს სხვადასხვა სახელი უნდა ჰქონდეს. სხვადასხვა სქემაში შეიძლება ერთნაირი სახელის მქონე ობიექტები იყოს. მაგალითად, Sqema_1 შეიძლება შეიცავდეს სახელებს: Cxrili_1, Procedura_1 და Trigeri_1; Sqema_2 კი სახელებს: Cxrili_1, Cxrili_2, Warmodgena_2 და Procedura_1. როგორც ვხედავთ, თითოეული სქემის შიგნით ობიექტების სახელები განსხვავებულია, თუმცა ორივე სქემაში არის Cxrili_1 სახელის მქონე ობიექტი.

სქემა გამოიყენება როგორც პრეფიქსი ობიექტის სახელში. დავუშვათ, Sqema1 სქემაში არის Xelshekruleba ცხრილი. მაშინ ობიექტის დაზუსტებული სახელი სქემის სახელის ჩართვით იქნება – Sqema1.Xelshekruleba. თუ ობიექტთან მიმართვისას გამოვტოვებთ სქემის სახელს, SQL სერვერი ასრულებს სქემის სახელის გადაწყვეტის პროცესს, რომელიც მოიცავს ობიექტის არსებობის შემოწმებას მომხმარებლის სქემაში. თუ სქემა არ არის მითითებული, მაშინ იგულისხმება dbo. რეკომენდებულია ობიექტთან მიმართვისას პროგრამულ კოდში ყოველთვის გამოვიყენოთ შედგენილი სახელები (შეიძლება სხვადასხვა სქემაში ერთნაირი სახელის ობიეტების იყოს, სახელების გადაწყვეტა კი მოითხოვს გარკვეულ ზედნადებ ხარჯებს და ა.შ.).

ჩვენ შეგვიძლია ვმართოთ მიმართვის უფლებები სქემის დონეზე. მაგალითად, მომხმარებელს შეგვიძლია მისცეთ SELECT ბრძანების შესრულების ნებართვა, ვაძლევთ რა უფლებას მიმართოს სქემის ყველა ობიექტს. ამრიგად, უსაფრთხოება - ერთ-ერთი მნიშვნელოვანი ასპექტია სქემებში ობიექტების ორგანიზების საშუალებების განსაზღვრისას.

სქემის შესაქმნელად გამოიყენება CREATE SCHEMA ბრძანება. მისი სინტაქსია:

CREATE SCHEMA *სქემის_სახელი*

სქემის წასაშლელად გამოიყენება DROP SCHEMA ბრძანება. მისი სინტაქსია:

DROP SCHEMA *სქემის_სახელი*

მაგალითი.

შევქმნათ Sqema1 სქემა. შემდეგ მასში შევქმნათ Cxrili1 ცხრილი. მოთხოვნას აქვს სახე:

```
USE Shekveta_1;
GO
CREATE SCHEMA Sqema1;
GO
CREATE TABLE Sqema1.Cxrili1
(
svet1 INT,
sveti2 INT,
sveti3 INT
);
```

ამის შემდეგ, თუ დაგვჭირდა Sqema1 სქემის წაშლა, ჯერ უნდა წავშალოთ Table1 ცხრილი:

```
DROP TABLE Sqema1.Table1;
```

და შემდეგ Sqema1 სქემა:

```
DROP SCHEMA Sqema1;
```

მონაცემთა ბაზის შექმნის დროს იქმნება სტანდარტული სქემები. ერთ-ერთი მათგანია dbo. თუ ობიექტის შექმნის დროს ან ობიექტთან მიმართვის დროს სქემის სახელი არ არის მითითებული, მაშინ იგულისხმება dbo სქემა. ჩვენ შეგვიძლია შევქმნათ სქემები და მასში მოვათავსოთ სხვადასხვა ობიექტი.

სქემები არსებობენ მათი შემქმნელი მომხმარებლებისაგან დამოუკიდებლად. სქემისაგან მომხმარებლის განცალკევებას შემდეგი უპირატესობები აქვს:

- რამდენიმე მომხმარებელი შეიძლება ფლობდეს ერთ სქემას როლში ან Windows ჯგუფში წევრობის გზით.
- მომხმარებლის წაშლა არ ითხოვს იმ ობიექტების სახელების შეცვლას, რომლებიც სქემაში შედის.
- სახელების გადაწყვეტის მიზნით რამდენიმე მომხმარებელს შეუძლია გაინაწილოს ერთი ნაგულისხმევი სქემა.
- ობიექტების სრულად განსაზღვრული სახელები ოთხი ნაწილისაგან შედგება: *სერვერის_სახელი.მონაცემთა_ბაზის_სახელი.სქემის_სახელი.ობიექტის_სახელი* .

ნაგულისხმევი სქემები გამოიყენება ობიექტების სახელების გადასაწყვეტად (განსაზღვრისათვის). სახელების გადაწყვეტა არის პროცესი, როცა ობიექტის სახელიდან ხდება მისი სრულად განსაზღვრული სახელის ფორმირება და მისი უნიკალურობის შემოწმება. მაგალითად, თუ ვასრულებთ SELECT * FROM Personali მოთხოვნას, მაშინ Personali სახელის ბაზაზე სერვერი ქმნის მის სრულ სახელს: Serveri.Shekveta.dbo.Personali. თუ ეს სახელი უნიკალურია dbo სქემის შიგნით, მაშინ მოთხოვნა შესრულდება. მაგალითად, შევქმნათ ცხრილი, რომლის სახელშიც მითითებულია ჩვენთვის საჭირო სქემა:

```
USE Shekveta;
CREATE TABLE Sqema_1.Cxrili_1 (Sveti_1 INT, Sveti_2 NVARCHAR(10));
SELECT * FROM Sqema_1.Cxrili_1;
```

Cxrili_1 ცხრილის ძებნა შესრულდება Shekveta მონაცემთა ბაზის Sqema_1 სქემაში. თუ სქემას არ მივუთითებთ, მაშინ ცხრილი მოიძებნება dbo სქემაში. შეგვიძლია, აგრეთვე, მონაცემთა ბაზის სახელის მითითებაც:

```
CREATE TABLE Shekveta.Sqema_1.Cxrili_1 (Sveti_1 INT, Sveti_2 NVARCHAR(10));
SELECT * FROM Shekveta.Sqema_1.Cxrili_1;
```

თითოეულ მომხმარებელს აქვს ნაგულისხმევი სქემა. ის არის პირველი სქემა, რომელიც მოიძებნება სერვერის მიერ, როცა ის წყვეტს ობიექტების სახელებს. ნაგულისხმევი სქემა შეიძლება განისაზღვროს და შეიცვალოს CREATE USER და ALTER USER ბრძანებების DEFAULT_SCHEMA რეჟიმის გამოყენებით. თუ DEFAULT_SCHEMA განუსაზღვრელია, მაშინ ნაგულისხმევი სქემა იქნება dbo.

მომხმარებლები

მას შემდეგ, რაც მომხმარებელმა წარმატებით გაიარა აუტენტიფიცირება და მიიღო სააღრიცხვო ჩანაწერის იდენტიფიკატორი (*login ID*), ის ითვლება დარეგისტრირებულად და მას ეძლევა სერვერთან მიმართვის უფლება. მომხმარებლის სააღრიცხვო ჩანაწერი (*login*) ასოცირდება კონკრეტული მონაცემთა ბაზის მომხმარებელთან (*user*). *მომხმარებლები* წარმოადგენენ სერვერის სპეციალურ ობიექტებს, რომელთა საშუალებით განისაზღვრება მიმართვის უფლებები და მონაცემთა ბაზაში ობიქტების მფლობელობა. მომხმარებლის სახელი შეიძლება გამოყენებული იყოს მიმართვის უფლების მისაცემად როგორც ერთი ადამიანისთვის, ისე ადამიანების ჯგუფისთვის.

მონაცემთა ბაზის შექმნის დროს განისაზღვრება ორი სტანდარტული მომხმარებელი: dbo და guest.

თუ სააღრიცხვო ჩანაწერი აშკარად არ არის დაკავშირებული მომხმარებელთან მაშინ მომხმარებელს ეძლევა მონაცემთა ბაზებთან არააშკარა მიმართვის უფლება guest სტუმრის სახელის გამოყენებით. ე.ი. ასეთი სააღრიცხვო ჩანაწერი, რომელმაც მიიღო სერვერთან მიმართვის უფლება, ავტომატურად აისახება guest მომხმარებელში ყველა მონაცემთა ბაზისთვის. თუ ჩვენ მონაცემთა ბაზიდან წავშლით guest მომხმარებელს, მაშინ სააღრიცხვო ჩანაწერები, რომლებსაც არ აქვთ აშკარა ასახვა მომხმარებლის სახელში, ვერ შეძლებს მონაცემთა ბაზასთან მიმართვას. უნდა აღვნიშნოთ, რომ guest მომხმარებელს არ აქვს ავტომატური მიმართვის უფლება მონაცემთა ბაზებთან. მონაცემთა ბაზის მფლობელმა თვითონ უნდა გადაწყვიტოს მისცეს თუ არა guest მომხმარებელს ეს უფლება.

მაქსიმალური უსაფრთხოების უზრუნველყოფისათვის შეგვიძლია წავშალოთ guest მომხმარებელი ყველა მონაცემთა ბაზიდან, გარდა master და tempdb სისტემური ბაზებიდან. master მონაცემთა ბაზაში guest გამოიყენება სისტემური შენახული პროცედურების შესასრულებლად ჩვეულებრივი მომხმარებლების მიერ. tempdb მონაცემთა ბაზაში guest გამოიყენება დროებითი ობიექტების შესაქმნელად ნებისმიერი მომხმარებლის მიერ.

მონაცემთა ბაზის მფლობელი (DataBase Owner, dbo) - სპეციალური მომხმარებელია, რომელსაც მაქსიმალური უფლებები აქვს მონაცემთა ბაზაში. sysadmin როლის ნებისმიერი წევრი ავტომატურად აისახება dbo მომხმარებელში. თუ მომხმარებელი, რომელიც არის sysadmin როლის წევრი, ქმნის რაიმე ობიექტს, მაშინ ამ ობიექტის მფლობელად დაინიშნება არა ეს მომხმარებელი, არამედ dbo. მაგალითად, თუ Saba მომხმარებელი, რომელიც არის ადმინისტრაციული ჯგუფის წევრი, ქმნის Cxrili_1 ცხრილს, მაშინ ცხრილის სრული სახელი

იქნება არა Saba.Cxrili_1, არამედ dbo.Cxrili_1. ამავე დროს, თუ Saba არის არა სერვერის sysadmin როლის წევრი, არამედ მონაცემთა ბაზის მფლობელის db_owner როლის წევრი, მაშინ ცხრილის სახელი იქნება Saba.Cxrili_1.

dbo მომხმარებლის წაშლა არ შეიძლება.

სააღრიცხვო ჩანაწერის (login) დასაკავშირებლად გარკვეულ მომხმარებელთან (user) შეგვიძლია გამოვიყენოთ sp_adduser შენახული პროცედურა. ის მიმდინარე მონაცემთა ბაზას უმატებს ახალ მომხმარებელს. მისი სინტაქსია:

sp_adduser [@loginame =] '*სააღრიცხვო_ჩანაწერის_სახელი*'

[, [@name_in_db =] '*მომხმარებლის_სახელი*']

[, [@grpname =] '*როლის_სახელი*']

განვიხილოთ არგუმენტების დანიშნულება:

– *სააღრიცხვო_ჩანაწერის_სახელი*. იმ სააღრიცხვო ჩანაწერის სახელია, რომელიც უნდა დაუკავშირდეს მონაცემთა ბაზის მომხმარებელს;

– *მომხმარებლის_სახელი*. მონაცემთა ბაზის იმ მომხმარებლის სახელია, რომელთანაც ასოცირდება მოცემული სააღრიცხვო ჩანაწერი;

– *როლის_სახელი*. განსაზღვრავს მონაცემთა ბაზის იმ როლის სახელია, რომელშიც მოხდება მომხმარებლის ჩართვა.

მაგალითები.

ა. მაგალითში ხდება Login_1 მომხმარებლის ჩართვა მიმდინარე მონაცემთა ბაზის Roli_1 როლში Login_1 სააღრიცხვო ჩანაწერის გამოყენებით.

```
USE Baza_8
EXEC sp_adduser 'Login_1', 'Login_1', 'Roli_1'
```

ბ. მაგალითში ხდება Login_3 სააღრიცხვო ჩანაწერის დამატება Shekveta მონაცემთა ბაზისთვის, რომელსაც ჰყავს User_3 მომხმარებელი და ამ მომხმარებელს ამატებს მიმდინარე მონაცემთა ბაზის Roli_1 როლში.

```
USE Shekveta
EXEC sp_adduser 'Login_3', 'User_3', 'Roli_1'
```

მიმდინარე მონაცემთა ბაზაში მომხმარებლის დასამატებლად შეგვიძლია CREATE USER ბრძანება გამოვიყენოთ. მისი გამარტივებული სინტაქსია:

CREATE USER *მომხმარებლის_სახელი*

[{ FOR LOGIN *სააღრიცხვო_ჩანაწერის_სახელი* | WITHOUT LOGIN }]

[WITH DEFAULT_SCHEMA = *სქემის_სახელი*]

განვიხილოთ არგუმენტების დანიშნულება.

- *მომხმარებლის_სახელი* შესაქმნელი მომხმარებლის სახელია მიმდინარე მონაცემთა ბაზაში.
- FOR LOGIN *სააღრიცხვო_ჩანაწერის_სახელი* სერვერის სააღრიცხვო ჩანაწერია, რომლისთვისაც იქმნება მონაცემთა ბაზის მომხმარებელი. სერვერის სააღრიცხვო ჩანაწერი უნდა არსებობდეს. როცა ეს სააღრიცხვო ჩანაწერი დაუკავშირდება მონაცემთა ბაზას, ის მიიღებს ამ მონაცემთა ბაზის ამ (შესაქმნელი) მომხმარებლის სახელსა და იდენტიფიკატორს.
- WITHOUT LOGIN მიუთითებს, რომ მომხმარებელი არ აისახება არსებულ სააღრიცხვო ჩანაწერში. ამ დროს იქმნება მომხმარებელი, რომელსაც მოქმედება შეუძლია მხოლოდ მისი საკუთარი მონაცემთა ბაზის შიგნით. ეს მომხმარებელი ვერ დაუკავშირდება სხვა მონაცემთა ბაზებს, როგორც guest და ვერ აისახება ვერც ერთ სააღრიცხვო ჩანაწერში.
- WITH DEFAULT_SCHEMA = *სქემის_სახელი* განსაზღვრავს პირველ (ნაგულისხმევ) სქემას, რომელიც მოიძებნება სერვერის მიერ მაშინ, როცა ის წყვეტს ობიექტების სახელებს მონაცემთა ბაზის ამ მომხმარებლისთვის. თუ ეს არგუმენტი მითითებული არ არის, მაშინ

მომხმარებლისთვის გამოყენებული იქნება dbo, როგორც მისი ნაგულისხმევი სქემა.

- FOR LOGIN თუ გამოტოვებულია, მაშინ მონაცემთა ბაზის ახალი მომხმარებელი აისახება სერვერის სააღრიცხვო ჩანაწერში იმავე სახელით.

CREATE USER ბრძანებით არ შეიძლება guest მომხმარებლის შექმნა, რადგან ის ყოველთვის არსებობს თითოეულ მონაცემთა ბაზაში. guest მომხმარებლის გააქტიურება შეგვიძლია შემდეგი ბრძანების შესრულების გზით:

```
GRANT CONNECT TO GUEST;
GO
```

მაგალითი. მაგალითში ჯერ იქმნება Login_1 სააღრიცხვო ჩანაწერი, შემდეგ კი შესაბამისი User_1 მომხმარებელი Shekveta მონაცემთა ბაზაში.

```
CREATE LOGIN Login_1 WITH PASSWORD = 'paroli';
USE Shekveta;
CREATE USER User_1;
GO
```

ან

```
CREATE LOGIN Login_1 WITH PASSWORD = 'Paroli';
USE Shekveta;
CREATE USER User_1 FOR LOGIN Login_1
   WITH DEFAULT_SCHEMA = Scema_1;
GO
```

მომხმარებელი, რომელიც მონაცემთა ბაზაში ობიექტს ქმნის, მაგალითად ცხრილს, შენახულ პროცედურას ან წარმოდგენას, ობიექტის მფლობელი ხდება. ობიექტის მფლობელს (database object owner, dbo) აქვს მის მიერ შექმნილ ობიექტთან მიმართვის ყველა უფლება. მომხმარებელმა რომ შეძლოს ობიექტის შექმნა, ამისათვის მას მონაცემთა ბაზის მფლობელმა უნდა მისცეს შესაბამისი უფლებები.

სერვერი საშუალებას გვაძლევს ობიექტის ფლობის უფლება გადავცეთ ერთი მომხმარებლიდან მეორეს. მონაცემთა ბაზიდან ობიექტის მფლობელის წასაშლელად, ჯერ უნდა წავშალოთ მის მიერ შექმნილი ობიექტები, ან მათი ფლობის უფლება გადავცეთ სხვა მომხმარებელს. ამისათვის შეგვიძლია გამოვიყენოთ sp_changeobjectowner შენახული პროცედურა. მისი სინტაქსია:

```
sp_changeobjectowner [ @objname = ] 'ობიექტის_სახელი',
                          [ @newowner = ] 'მფლობელის_სახელი'
```

აქ '*მფლობელის_სახელი*' ობიექტის ახალი მფლობელის სახელია.

მაგალითი. მაგალითში Cxrili_1 ცხრილის ახალი მფლობელი ხდება Login_1.

```
USE Baza_1
GO
sp_changeobjectowner 'Cxrili_1', 'Login_1'
```

სერვერის როლები

როლი საშუალებას გვაძლევს გავაერთიანოთ მომხმარებლები, რომლებიც ერთნაირ ფუნქციებს ასრულებენ. ეს საჭიროა სერვერის უსაფრთხოების სისტემის ადმინისტრირების გასამარტივებლად.

სერვერზე რეალიზებულია ორი სტანდარტული როლი: სერვერის დონეზე და მონაცემთა ბაზის დონეზე. სერვერის დაყენების დროს იქმნება სერვერის ცხრა ფიქსირებული როლი და მონაცემთა ბაზის ცხრა ფიქსირებული როლი. ამ როლებს ჩვენ ვერ წავშლით და არ

შეიძლება მათი უფლებების მოდიფიცირება. იმისათვის, რომ მომხმარებელს მივანიჭოთ ის უფლებები, რომელიც აქვს სერვერის რომელიმე ფიქსირებულ როლს, საჭიროა მომხმარებლის ამ როლში ჩართვა.

სერვერის როლების (server role) გამოყენებით სერვერის ოპერატორებს ენიჭებათ მხოლოდ ის უფლებები, რომლებსაც ადმინისტრატორი ჩათვლის საჭიროდ. სერვერის ნებისმიერ როლში შეგვიძლია ჩავრთოთ სერვერის ან Windows NT-ის სააღრიცხვო ჩანაწერი.

სერვერის სტანდარტული როლები (fixed server role) და მათი უფლებები მოყვანილია ცხრილში 20.1.

ცხრილი 20.1. სერვერის ფიქსირებული როლები

სერვერის ფიქსირებული როლები	დანიშნულება
Sysadmin	სერვერზე შეუძლია ნებისმიერი მოქმედების შესრულება
Serveradmin	ასრულებს სერვერის კონფიგურირებასა და გამორთვას
Setupadmin	მართავს ბმულ სერვერებსა და პროცედურებს, რომლებიც ავტომატურად გაიშვება სერვერის გაშვებისას
Securityadmin	მართავს სააღრიცხვო ჩანაწერებსა და მონაცემთა ბაზის შექმნის წესებს, ასევე, შეუძლია შეცდომების ჟურნალის წაკითხვა
Processadmin	მართავს სერვერის მიერ გაშვებულ პროცესებს
Dbcreator	შეუძლია მონაცემთა ბაზების შექმნა და მოდიფი-ცირება
Diskadmin	მართავს სერვერის ფაილებს
Bulkadmin (Bulk Insert administrators)	შეუძლია მონაცემები ჩასვას მასობრივი გადაწერის საშუალებების გამოყენებით

მონაცემთა ბაზების როლები

მონაცემთა ბაზების როლები (database role) საშუალებას გვაძლევს მომხმარებლები გავაერთიანოთ ერთ ადმინისტრაციულ ერთეულში და მასთან ვიმუშაოთ როგორც ჩვეულებრივ მომხმარებელთან. განვსაზღვრავთ რა მონაცემთა ბაზის ობიექტებთან მიმართვის უფლებებს კონკრეტული როლისთვის, ჩვენ ამით ავტომატურად ვაძლევთ იგივე უფლებებს ამ როლის ყველა წევრს. 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*']

აქ *მფლობელის_სახელი* ახალი როლის მფლობელის სახელია. მფლობელი უნდა იყოს მიმდინარე მონაცემთა ბაზის მომხმარებელი ან მიმდინარე მონაცემთა ბაზის როლი.

მაგალითები. მაგალითში ხდება Shekveta მონაცემთა ბაზისთვის Roli_1 როლის დამატება:

ა. Shekveta მონაცემთა ბაზაში იქმნება Roli_2 როლი, რომლის მფლობელი იქნება dbo.

```
USE Shekveta
GO
sp_addrole 'Roli_2'
```

ბ. Shekveta მონაცემთა ბაზაში იქმნება Roli_3 როლი, რომლის მფლობელი იქნება მიმდინარე მონაცემთა ბაზის User_3 მომხმარებელი.

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

გ. Shekveta მონაცემთა ბაზაში იქმნება Roli_4 როლი, რომლის მფლობელი იქნება მიმდინარე მონაცემთა ბაზის ფიქსირებული db_owner როლი.

```
USE Shekveta
GO
sp_addrole 'Roli_4', 'db_owner'
```

მიმდინარე მონაცემთა ბაზის როლში მონაცემთა ბაზის მომხმარებლის, მონაცემთა ბაზის როლის, Windows-ის სააღრიცხვო ჩანაწერისა და Windows-ის ჯგუფის ჩასართავად გამოიყენება sp_addrolemember შენახული პროცედურა. მისი სინტაქსია:

```
sp_addrolemember [ @rolename = ] 'როლის_სახელი',
                 [ @membername = ] 'უსაფრთხოების_ობიექტის_სახელი'
```

განვიხილოთ აგუმენტების დანიშნულება:

– *'როლის_სახელი'*. მონაცემთა ბაზის როლის სახელია მიმდინარე მონაცემთა ბაზაში;

– *'უსაფრთხოების_ობიექტის_სახელი'* არის უსაფრთხოების სისტემის ობიექტის სახელი, რომელიც როლში უნდა ჩავრთოთ. ასეთი ობიექტი შეიძლება იყოს: მონაცემთა ბაზის მომხმარებელი, მონაცემთა ბაზის როლი, Windows NT-ის სააღრიცხვო ჩანაწერი ან ჯგუფი.

მაგალითები.

ა. მაგალითში ხდება User_1 მომხმარებლის ჩართვა მიმდინარე მონაცემთა ბაზის Roli_1 როლში.

```
EXEC sp_addrolemember 'Roli_1', 'User_1'
```

ბ. მაგალითში ხდება Windows-ის სააღრიცხვო ჩანაწერის - Romani-PC\romani-ის დამატება Shekveta მონაცემთა ბაზისთვის, როგორც User2 მომხმარებელი. User2 დაემატება Roli_1 როლს.

```
USE Shekveta
GO
EXEC sp_grantdbaccess 'Romani-PC\romani', 'User2'
GO
EXEC sp_addrolemember 'Roli_1', 'User2'
```

მონაცემთა ბაზის შექმნისას მისთვის განისაზღვრება მონაცემთა ბაზის სტანდარტული როლები, რომლებიც არ შეიძლება იყოს წაშლილი ან შეცვლილი. მონაცემთა ბაზის სტანდარტული როლები (fixed database role) და მათი უფლებები მოყვანილია ცხრილში 20.2.

ზემოთ ჩამოთვლილი როლების გარდა არსებობს public როლი. მას სპეციალური დანიშნულება აქვს, რადგან მისი წევრია ყველა მომხმარებელი, რომლებსაც შეუძლიათ მონაცემთა ბაზასთან მიმართვა. ამ როლის წევრების ცხადად განსაზღვრა არ შეიძლება, რადგან ყველა მომხმარებელი ავტომატურად ხდება მისი წევრი. ეს როლი უნდა გამოვიყენოთ მიმართვის მინიმალური უფლებების მისანიჭებლად იმ მომხმარებლისთვის, რომლებისთვისაც ობიექტებთან მიმართვის უფლებები ცხადად არ იყო განსაზღვრული. თუ

მონაცემთა ბაზაში ნებადართულია guest მომხმარებელი, მაშინ public როლისთვის მინიჭებული უფლებები ექნება ყველა მომხმარებელს, რომლებსაც აქვთ სერვერთან მიმართვის უფლება. public როლი არის ყველა მონაცემთა ბაზაში, master, tempdb, msdb, model სისტემური მონაცემთა ბაზების ჩათვლით და არ შეიძლება იყოს წაშლილი.

Windows NT-ის ჯგუფები შეიძლება გამოყენებული იყოს სერვერის როლების ანალოგიურად. Windows NT-ის ჯგუფისთვის შეგვიძლია შევქმნათ ერთი სააღრიცხვო ჩანაწერი (login) და შესაბამისი მომხმარებლები როლის ნაცვლად ჩავრთოთ Windows NT-ის ჯგუფში.

ცხრილი 20.2. მონაცემთა ბაზრის ფიქსირებული როლები.

მონაცემთა ბაზის ფიქსირებული როლები	დანიშნულება
db_owner	აქვს ყველა უფლება მონაცემთა ბაზაში
db_accessadmin	შეუძლია მომხმარებლების დამატება და წაშლა
db_securityadmin	მართავს ყველა ნებართვას, ობიექტს, როლს და როლების წევრებს
db_ddladmin	შეუძლია DDL ბრძანებების შესრულება, გარდა GRANT, DENY და REVOKE ბრძანებებისა
db_backupoperator	შეუძლია DBCC, CHECKPOINT და BACKUP ბრძანებების შესრულება
db_datareader	შეუძლია ნახოს ნებისმიერი მონაცემები მონაცემთა ბაზის ნებისმიერ ცხრილში
db_datawriter	შეუძლია შეცვალოს ნებისმიერი მონაცემები მონაცემთა ბაზის ნებისმიერ ცხრილში
db_denydatareader	ეკრძალება ნახოს ნებისმიერი მონაცემები მონაცემთა ბაზის ნებისმიერ ცხრილში
db_denydatawriter	ეკრძალება შეცვალოს ნებისმიერი მონაცემები მონაცემთა ბაზის ნებისმიერ ცხრილში

პროგრამა-დანართების როლები

იმისათვის, რომ პროგრამა-დანართებმა იმუშაონ, მათ სჭირდებათ შესაბამისი უფლებები. ამ უფლებების რეალიზება ხორციელდება პროგრამა-დანართების როლების გამოყენებით, რომლებიც მონაცემთა ბაზის დონეზე იქმნება. პროგრამა-დანართის როლს წევრები არ ჰყავს. როლი აქტიურდება მაშინ, როცა პროგრამა-დანართი შეერთებას ამყარებს. მომხმარებელი, რომელიც მოცემულ მომენტში მუშაობს პროგრამა-დანართთან, არ არის ამ როლის წევრი. დამყარებულ შეერთებას მხოლოდ მისი პროგრამა-დანართი იყენებს.

პროგრამა-დანართის როლის შესაქმნელად გამოიყენება sp_addapprole შენახული პროცედურა. მისი სინტაქსია:

sp_addapprole [@rolename =] '*როლის_სახელი*' , [@password =] '*პაროლი*'

პროგრამა-დანართი შეიძლება ისე იყოს დაპროექტებული, რომ მის გასააქტიურებლად პაროლი შეიტანოს ან მომხმარებელმა ან თვით პროგრამა-დანართმა.

მიერთების პროცესში პროგრამა-დანართმა უნდა გააქტიუროს შესაბამისი როლი. ამისათვის გამოიყენება sp_setapprole შენახული პროცედურა. მისი სინტაქსია:

sp_setapprole [@rolename =] '*როლის_სახელი*',

[@password =] { Encrypt N'*პაროლი*' } | '*პაროლი*'

განვიხილოთ არგუმენტების დანიშნულება:

– *'როლის_სახელი'*. პროგრამა-დანართის როლის სახელია, რომელიც უნდა გააქტიურდეს.

– *'პაროლი'*. ის პროგრამა-დანართმა უნდა გადასცეს სერვერს როლის გასააქტიურებლად.

მაგალითი. მაგალითში ხდება პროგრამა-დანართის App_Roli_2 როლის შექმნა და გააქტიურება პაროლის დაშიფრვის გარეშე:

```
USE Shekveta
GO
EXEC sp_addapprole 'App_Roli_2', 'paroli'
EXEC sp_setapprole 'App_Roli_2', 'paroli'
GO
```

როცა პროგრამა-დანართის როლი აქტიურდება, მაშინ მიმართვის ყველა უფლება მომხმარებლისთვის, ჯგუფებისთვის ან როლებისთვის სეანსის ფარგლებში უქმდება პროგრამა-დანართის მუშაობის დამთავრებამდე. შეერთება იღებს უფლებებს, რომლებიც განსაზღვრულია პროგრამა-დანართის როლისთვის იმ მონაცემთა ბაზაში, რომელშიც ეს როლი არსებობს. შეერთების დამამყარებელი მომხმარებლისთვის მინიჭებული მიმართვის უფლებების დროებითი „დავიწყება", საჭიროა მიმართვის კონფლიქტების აღმოსაფხვრელად. საწინააღმდეგო შემთხვევაში, თუ მომხმარებლისთვის აკრძალულია მონაცემების წაკითხვა, ხოლო პროგრამა-დანართს კი სჭირდება ამ მონაცემების წაკითხვა, მაშინ მიმართვის მოთხოვნა უარყოფილი იქნება, რადგან მიმართვის უფლების აკრძალვას უპირატესობა აქვს მიმართვის უფლების მინიჭებაზე.

რადგან პროგრამა-დანართის როლს უფლებები აქვს მხოლოდ იმ მონაცემთა ბაზაში, რომელშიც ის შეიქმნა, ხოლო ყველა ნებართვა სააღრიცხვო ჩანაწერისთვის, ჯგუფებისთვის და როლებისთვის, რომელთაც ეკუთვნის მომხმარებელი, იკარგება, მაშინ სხვა მონაცემთა ბაზებთან მიმართვა შესაძლებელი იქნება მხოლოდ guest სტუმრის სახელით. შედეგად თუ guest სახელი მონაცემთა ბაზაში არ არის, მაშინ შეერთება ვერ მიმართავს მონაცემებს. საწინააღმდეგო შემთხვევაში, შეერთება მხოლოდ იმ შემთხვევაში მიმართავს მონაცემთა ბაზის ობიექტებს, როცა ნებართვები ცხადად არის გაცემული guest მომხმარებლისთვის.

პროგრამა-დანართის როლის საშუალებით შეერთების დამყარებამდე მომხმარებელი ჯერ უნდა დაუკავშირდეს სერვერს. ამისათვის დასაშვებია მომხმარებლების აუტენტიფიცირების ორივე რეჟიმის გამოყენება.

როლის შეცვლა

როლის სახელის შესაცვლელად გამოიყენება ALTER ROLE ბრძანება. მისი სინტაქსია:

ALTER ROLE *როლის_სახელი* WITH NAME = *ახალი_სახელი*

მონაცემთა ბაზის როლის სახელის შეცვლა არ ცვლის როლის იდენტიფიკატორს, მფლობელს და უფლებებს. მონაცემთა ბაზის როლები მოთავსებულია

მაგალითი. მაგალითში Roli_6 სახელი იცვლება Roli_7 სახელით.

```
USE Shekveta;
ALTER ROLE Roli_6  WITH NAME = Roli_7;
GO
```

უსაფრთხოების სისტემის ადმინისტრირება

სააღრიცხვო ჩანაწერების შექმნა და მართვა

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*']

განვიხილოთ არგუმენტების დანიშნულება:

– '*სააღრიცხვო_ჩანაწერის_სახელი*'. შესაქმნელი სააღრიცხვო ჩანაწერის სახელია, რომელიც გამოყენებული იქნება მომხმარებლის აუტენტიფიცირებისათვის. სერვერის სააღრიცხვო ჩანაწერების სახელები და პაროლები შეიძლება შეიცავდეს 1128 სიმბოლოს ასოების, სიმბოლოებისა და რიცხვების ჩათვლით. გარდაა ამისა, სააღრიცხვო ჩანაწერის სახელი არ უნდა შეიცავდეს ირიბ სლეშს (\), დარეზერვებულ სახელებს, არ უნდა ჰქონდეს NULL მნიშვნელობა და არ უნდა იყოს ცარიელი სტრიქონი ('').

– '*მონაცემთა_ბაზის_სახელი*'. ავტომატურად ნაგულისხმევი მონაცემთა ბაზაა, რომელსაც მომხმარებელი მიუერთდება სერვერთან კავშირის დამყარებისთანავე. უნდა ვეცადოთ, რომ ეს არგუმენტი ყოველთვის მივუთითოთ. საწინააღმდეგო შემთხვევაში, მომხმარებელი მიუერთდება master მონაცემთა ბაზას, რამაც შეიძლება არასასურველ შედეგებამდე მიგვიყვანოს.

– '*ენა*'. არის ნაგულისხმევი ენა, რომელიც ძალაში იქნება მომხმარებლისთვის სერვერთან დაკავშირებისთანავე. სერვერზე არსებული ენები მოთავსებულია მიმდინარე მონაცემთა ბაზის syslanguages სისტემურ წარმოდგენაში. მათ სანახავად უნდა შევასრულოთ მოთხოვნა: SELECT * FROM syslanguages

– *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.

მაგალითები.

ა. შევქმნათ სააღრიცხვო ჩანაწერი, რომლისთვისაც მითითებული იქნება პაროლი.

```
sp_addlogin 'Login_2', 'paroli'
```

ბ. შევქმნათ სააღრიცხვო ჩანაწერი, რომლისთვისაც მითითებული იქნება პაროლი და მონაცემთა ბაზა.

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

გ. შევქმნათ სააღრიცხვო ჩანაწერი, რომლისთვისაც მითითებული იქნება პაროლი, მონაცემთა ბაზა და ენა.

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

დ. შევქმნათ სააღრიცხვო ჩანაწერი, რომლისთვისაც მითითებული იქნება უნიკალური იდენტიფიკატორი.

sp_addlogin 'Login_5', 'paroli', 'Shekveta', 'us_english', 0x0123456789ABCDEF0123456789ABCDEF

უნიკალური იდენტიფიკატორის მისაღებად უნდა შევასრულოთ მოთხოვნა:

```
USE Master
SELECT CONVERT(VARBINARY(32), password)
FROM syslogins  WHERE name = 'Login_1'
```

sp_password შენახული პროცედურა ცვლის სერვერის სააღრიცხვო ჩანაწერის პაროლს. მისი სინტაქსია:

```
sp_password     [ [ @old = ] 'ძველი_პაროლი' , ]
                { [ @new = ] 'ახალი_პაროლი' }
                 [ , [ @loginame = ] 'სააღრიცხვო_ჩანაწერის_სახელი' ]
```

მაგალითები.

ა. მაგალითში Login_1 სააღრიცხვო ჩანაწერის ძველი პაროლი იცვლება ახლით. ძველი პაროლი სააღრიცხვო ჩანაწერის შექმნის დროს მითითებული არ იყო.

```
EXEC sp_password NULL, 'AxaliParoli', 'Login_1'
```

ბ. მაგალითში Login_2 სააღრიცხვო ჩანაწერის ZveliParoli ძველი პაროლი იცვლება ახლით - AxaliParoli.

```
EXEC sp_password 'ZveliParoli', 'AxaliParoli', 'Login_2'
```

მონაცემების დაცვა

მონაცემების დაშიფვრა

დაშიფვრა მეთოდია, რომელსაც სერვერი იყენებს მონაცემების შესაცვლელად არაწაკითხვად ფორმამდე. დაშიფვრა იძლევა იმის გარანტიას, რომ კონფიდენციალურ ინფორმაციას ვერავინ წაიკითხავს. დაშიფრული მონაცემების წასაკითხად ავტორიზებული მომხმარებლებლების მიერ გამოიყენება *გაშიფვრა.*

შეგვიძლია დავშიფროთ:

– ქსელში კლიენტსა და სერვერს შორის გადასაცემი ნებისმიერი მონაცემები;

– სერვერის სააღრიცხვო ჩანაწერების ან პროგრამა-დანართების როლების პაროლები;

– მონაცემთა ბაზის ობიექტების, როგორიცაა: შენახული პროცედურების, წარმოდგენების, ტრიგერების, შექმნისათვის გამოყენებული კოდი.

სააღრიცხვო ჩანაწერებისა და პროგრამა-დანართების როლების პაროლები ინახება სერვერის სისტემურ ცხრილებში დაშიფრული ფორმით. თუ ტრიგერის, შენახული პროცედურის ან წარმოდგენის კოდი შეიცავს მონაცემებს ან ალგორითმს, რომლებიც საიდუმლოა, მაშინ ის უნდა დავშიფვროთ.

სერვერის ფაილებთან მიმართვის შეზღუდვა

სერვერი მუშაობის დროს ქმნის და იყენებს მონაცემთა ბაზების, შეცდომების ჟურნალების, სარეზერვო ასლების, მონაცემების ექსპორტისა და იმპორტის და ა.შ. ფაილებს. სერვერის სამსახურები ოპერაციული სისტემის დონეზე პროცესების სახით სრულდება, რომლებსაც უნდა შეეძლოთ ზემოთ ნახსენებ ფაილებთან მიმართვა. ამის გამო, ოპერაციული

სისტემის დონეზე შესაბამისი უფლებები უნდა მივცეთ სერვერის სააღრიცხვო ჩანაწერებს. უმჯობესია, მიმართვის უფლებები ვმართოთ უშუალოდ ფაილებისა და კატალოგების დონეზე. ამისათვის, სერვერმა უნდა იმუშაოს Windows NT ოპერაციული სისტემის მართვის ქვეშ.

სერვერის ფაილებთან არასანქცირებული მიმართვის შეზღუდვის მიზნით მათთვის უნდა დავაყენოთ წაკითხვის, წაშლისა და მოდიფიცირების აკრძალვა ყველა მომხმარებლისთვის, გარდა უშუალოდ სერვერის მომხმარებლებისა.

მიმართვის უფლებები

როცა მომხმარებლები უერთდება სერვერს, მოქმედებები, რომელთა შესრულებაც მათ შეუძლიათ, განისაზღვრება უფლებებით (ნებართვებით), რომლებიც მიენიჭა მათ სააღრიცხვო ჩანაწერებს, ჯგუფს ან როლს, რომელშიც ისინი შედიან.

სერვერზე უფლებები შეიძლება სამ კატეგორიად დაიყოს:

– მონაცემთა ბაზების ობიექტებთან მიმართვის უფლებები;
– Transact SQL-ის ბრძანებების შესრულების უფლებები;
– არაცხადი უფლებები (ნებართვები).

შექმნის შემდეგ, მომხმარებელს აქვს მხოლოდ მონაცემთა ბაზის public როლისთვის მინიჭებული უფლებები. ამ როლის მიერ განსაზღვრული უფლებები მისაწვდომია ყველა მომხმარებლისათვის.

მომხმარებელს უფლებები ეძლევა ადმინისტრატორის, მონაცემთა ბაზის მფლობელის ან ობიექტის მფლობელის მიერ. მომხმარებლისათვის უფლებების გარკვეული ნაკრების მისანიჭებლად როლები უნდა გამოვიყენოთ. რამდენიმე როლის შექმნისა და მათთვის მიმართვის საჭირო უფლებების მინიჭების შემდეგ, ადმინისტრატორს შეუძლია მომხმარებლები შესაბამის როლებში ჩართოს. მომხმარებელი ავტომატურად იღებს ყველა იმ უფლებას, რომლებიც როლისთვისაა განკუთვნილი. მონაცემთა ბაზის სტანდარტულ როლებს უკვე აქვთ მინიჭებული უფლებების გარკვეული ნაკრები.

მიმართვის უფლებების მართვა შეგვიძლია არამარტო ცხრილების დონეზე, არამედ სვეტების დონეზეც.

იგივე ეხება Transact-SQL-ის ბრძანებების შესრულების ნებართვებს. შეგვიძლია მონაცემთა ბაზა ისე დავაპროექტოთ, რომ კონკრეტული მოქმედებების შესრულება, როგორიცაა: ცხრილების, წარმოდგენების, წესების, სარეზერვო ასლების და ა.შ. შექმნა, შეეძლოთ მხოლოდ მკაცრად განსაზღვრულ მომხმარებლებს.

მონაცემთა ბაზის ობიექტებთან მიმართვის უფლებები

მონაცემებთან მუშაობა და შენახული პროცედურების შესრულება მოითხოვს მიმართვის კლასის არსებობას, რომელსაც მონაცემთა ბაზის ობიექტებთან *მიმართვის უფლებები* ეწოდება. ობიექტებს წარმოადგენენ ცხრილები, სვეტები, წარმოდგენები, შენახული პროცედურები. მონაცემთა ბაზების ობიექტებთან მიმართვის უფლებები მართავენ მომხმარებლების მიერ ბრძანებების (SELECT, INSERT, UPDATE და DELETE) შესრულების შესაძლებლობას ცხრილებისა და წარმოდგენებისათვის. ამრიგად, თუ მომხმარებელს სჭირდება ცხრილში ახალი მონაცემების დამატება, მაშინ მას უნდა მივცეთ INSERT უფლება. თუ მომხმარებელს სჭირდება შენახული პროცედურების შესრულება, მაშინ მას უნდა მივცეთ EXECUTE უფლება.

სხვადასხვა ობიექტისთვის გამოიყენება მათთან მიმართვის უფლებების სხვადასხვა

ნაკრები:

– SELECT და UPDATE უფლებები გამოიყენება ცხრილის ან წარმოდგენის სვეტის მიმართ.

– SELECT, INSERT, UPDATE, DELETE და REFERENCES უფლებები გამოიყენება ცხრილებისა და წარმოდგენებისათვის;

– EXECUTE უფლება გამოიყენება მხოლოდ შენახული პროცედურებისა და ფუნქციების მიმართ.

ცალ-ცალკე აღვწეროთ ეს უფლებები:

– INSERT უფლება საშუალებას გვაძლევს ცხრილში ან წარმოდგენაში ჩავსვათ ახალი სტრიქონები. შედეგად, INSERT უფლება შეიძლება გაიცეს მხოლოდ ცხრილის ან წარმოდგენის დონეზე და არა სვეტის დონეზე.

– UPDATE უფლება გაიცემა ცხრილის დონეზე, რაც საშუალებას გვაძლევს მთელ ცხრილში შევცვალოთ მონაცემები, ან სვეტის დონეზე, რაც საშუალებას გვაძლევს მხოლოდ კონკრეტულ სვეტში შევცვალოთ მონაცემები.

– DELETE უფლება გვაძლევს ცხრილიდან ან წარმოდგენიდან სტრიქონების წაშლის საშუალებას. DELETE უფლება შეიძლება გაიცეს მხოლოდ ცხრილის ან წარმოდგენის დონეზე და არა სვეტის დონეზე.

– SELECT უფლება იძლევა მონაცემების ამორჩევის შესაძლებლობას. შეიძლება გაიცეს როგორც ცხრილის, ისე სვეტის დონეზე.

– REFERENCES უფლება იძლევა მითითებულ ობიექტთან მიმართვის შესაძლებლობას. ცხრილთან მიმართებაში მომხმარებელს უფლებას აძლევს შექმნას ამ ცხრილის გარე გასაღები ან ამ ცხრილის უნიკალური სვეტი. წარმოდგენასთან მიმართებაში მომხმარებელს უფლებას აძლევს წარმოდგენა დააკავშიროს ცხრილების სქემებთან, რომელთა საფუძველზეც იგება ეს წარმოდგენა. ეს საშუალებას გვაძლევს, თვალყური ვადევნოთ საწყისი ცხრილების სტრუქტურის ცვლილებას, რომლებმაც შეიძლება გავლენა იქონიოს წარმოდგენის მუშაობაზე.

მონაცემთა ბაზის მომხმარებლისთვის მონაცემთა ბაზის ყველა ობიექტთან მიმართვის უფლების მიცემა ან წართმევა შეიძლება მონაცემთა ბაზის ჩადგმული როლების საშუალებით. მაგალითად, მონაცემთა ბაზის ყველა ცხრილიდან და წარმოდგენიდან მონაცემების წაკითხვის უფლების მისაცემად საკმარისია, მომხმარებელი ჩავრთოთ db_datareader ფიქსირებულ როლში და არ შევცვალოთ თითოეულ ცხრილთან და წარმოდგენასთან მომხმარებლის მიმართვის უფლებები ცალ-ცალკე.

GRANT ბრძანება გამოიყენება მონაცემთა ბაზის ობიექტებთან მომხმარებლის მიმართვის უფლებების მართვისთვის. 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* }]

განვიხილოთ არგუმენტების დანიშნულება.

– ALL. მისი გამოყენება მხოლოდ sysadmin როლის წევრებს შეუძლიათ. არგუმენტი არ იძლევა ყველა შესაძლო უფლებას. ის შემდეგი უფლებების გაცემის ეკვივალენტურია:

- მონაცემთა ბაზისთვის 'ALL' ნიშნავს BACKUP DATABASE, BACKUP LOG, CREATE DATABASE, CREATE DEFAULT, CREATE FUNCTION, CREATE PROCEDURE, CREATE

RULE, CREATE TABLE და CREATE VIEW.

- Scalar ფუნქციისთვის 'ALL' ნიშნავს EXECUTE და REFERENCES.
- Inline და Multi-statement ფუნქციებისთვის 'ALL' ნიშნავს DELETE, INSERT, REFERENCES, SELECT და UPDATE.
- შენახული პროცედურისთვის 'ALL' ნიშნავს DELETE, EXECUTE, INSERT, SELECT და UPDATE.
- ცხრილისთვის 'ALL' ნიშნავს DELETE, INSERT, REFERENCES, SELECT და UPDATE.
- წარმოდგენისთვის 'ALL' ნიშნავს DELETE, INSERT, REFERENCES, SELECT და UPDATE.

– *ნებართვა*. მისაწვდომი უფლებების სიაა, რომელიც მომხმარებელს ეძლევა (SELECT, INSERT, UPDATE,DELETE, EXECUTE). თუ მომხმარებელს რამდენიმე უფლებას ვაძლევთ, მაშინ ისინი მძიმეებით უნდა გამოვყოთ ერთმანეთისაგან.

– *უსაფრთხოების_სისტემის_ობიექტის_სახელი*. უსაფრთხოების სისტემის ობიექტის სახელია, რომელიც როლში უნდა იყოს ჩართული. ასეთი ობიექტი შეიძლება იყოს როგორც სერვერის სააღრიცხვო ჩანაწერი, ისე Windows NT-ის მომხმარებლები ან ჯგუფები, რომლებსაც მინიჭებული აქვთ სერვერთან მიმართვის უფლება.

– *ცხრილის_სახელი, წარმოდგენის_სახელი, სვეტის_სახელი, შენახული_პროცედურის_სახელი, მომხმარებლის_ფუნქციის_სახელი*. ეს არგუმენტები შეიცავს მიმდინარე მონაცემთა ბაზის ობიექტების სახელებს, რომლებთანაც მიმართვა სრულდება.

– WITH GRANT OPTION. ეს არგუმენტი მომხმარებელს უფლებას აძლევს ობიექტთან მიმართვის უფლებები სხვა მომხმარებლებს დაუნიშნოს.

– AS { *ჯგუფის_სახელი* | *როლის_სახელი* }. ეს არგუმენტი მიუთითებს მომხმარებლის წევრობაზე იმ როლში, რომელსაც უფლება აქვს უფლებები სხვა მომხმარებლებს მისცეს.

მაგალითები.

ა. 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 User_2 მომხმარებელს.

ბ. 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.

მიმართვის ერთადერთი უფლება, რომელიც შეიძლება გაიცეს შენახული პროცედურისთვის, არის მისი შესრულების უფლება (EXECUTION). ბუნებრივია, მის მფლობელს შეუძლია ნახოს და შეცვალოს შენახული პროცედურის კოდი.

ფუნქციისთვის შეიძლება გაიცეს როგორც მისი შესრულების უფლება, ისე, REFERENCES უფლება, რაც უზრუნველყოფს იმ ობიექტებთან ფუნქციის დაკავშირების შესაძლებლობას, რომელსაც ფუნქცია მიმართავს. ასეთი კავშირი კრძალავს ფუნქციასთან დაკავშირებული ობიექტის სტრუქტურის შეცვლას, რადგან ეს გამოიწვევს ფუნქციის მუშაობის დარღვევას.

Transact-SQL-ის ბრძანებების შესრულების უფლებები

უფლებების ეს კლასი მართავს მონაცემთა ბაზის შექმნის, მონაცემთა ბაზის ობიექტებისა და სარეზერვო ასლის შექმნის პროცედურის შესრულების შესაძლებლობას. მაგალითად, თუ მომხმარებელს სურს წარმოდგენის შექმნა, მაშინ ადმინისტრატორმა მას უნდა მისცეს CREATE VIEW ბრძანების შესრულების უფლება. ბრძანებების უფლებები მოყვანილია ცხრილში 16.3.

ცხრილი 16.3. ბრძანებების უფლებები.

Transact_SQL ბრძანება	დანიშნულება
CREATE DATABASE	მონაცემთა ბაზის შექმნა
CREATE TABLE	ცხრილის შექმნა
CREATE VIEW	წარმოდგენის შექმნა
CREATE DEFAULT	ავტომატური მნიშვნელობის შექმნა
CREATE PROCEDURE	შენახული პროცედურის შექმნა
BACKUP DATABASE	მონაცემთა ბაზის სარეზერვო ასლის შექმნა
BACKUP LOG	ტრანზაქციების ჟურნალის სარეზერვო ასლის შექმნა
ALL	გაიცემა ყველა ზემოთ ჩამოთვლილი უფლება

სერვერის sysadmin ფიქსირებული როლის და მონაცემთა ბაზის db_owner ფიქსირებული როლის წევრებს ავტომატურად აქვთ ALL უფლება. სხვა ფიქსირებული როლის წევრებს აქვთ შესაბამისი უფლებების ნაკრები.

Transact-SQL-ის ბრძანებების შესრულების უფლების მისაცემად გამოიყენება ბრძანება:
GRANT { ALL | *ბრძანება* [,...n] } TO *უსაფრთხოების_სისტემის_ობიექტის_სახელი* [,...n]

ბრძანება არგუმენტი იღებს ცხრილში 15.3 მოყვანილი მნიშვნელობებიდან ერთ-ერთს.

მაგალითები.

ა. CREATE TABLE უფლება მივცეთ Roli_1 როლის წევრებს:
GRANT CREATE TABLE TO Roli_1

ბ. CREATE VIEW და BACKUP DATABASE უფლებები მივცეთ Roli_1 და Roli_2 როლის წევრებს:
GRANT CREATE VIEW, BACKUP DATABASE TO Roli_1, Roli_2

არაცხადი უფლებები

ზოგიერთი მოქმედების შესრულებას არ სჭირდება ცხადად უფლებების მინიჭება და მისაწვდომია ავტომატურად. ეს მოქმედებები შეიძლება შესრულდეს მხოლოდ სერვერის როლის წევრების მიერ ან ობიექტების მფლობელების მიერ მონაცემთა ბაზაში.

არაცხადი უფლებები მომხმარებელს არ გადაეცემა უშუალოდ. ის მათ იღებს მხოლოდ გარკვეულ გარემოებებში. მაგალითად, მომხმარებელი შეიძლება გახდეს მონაცემთა ბაზის ობიექტის მფლობელი, თუ ის თვითონ შექმნის ამ ობიექტს ან თუ სხვა მფლობელი გადასცემს მას თავისი ობიექტის ფლობის უფლებას. ამრიგად, ობიექტის მფლობელი ავტომატურად იღებს ამ ობიექტზე ნებისმიერი მოქმედების შესრულების უფლებას, მათ შორის, სხვა მომხმარებლისთვის ამ ობიექტთან მიმართვის გადაცემის უფლებას. ეს უფლებები არსად ცხადად არ ეთითება. მხოლოდ ობიექტის ფლობის ფაქტი მომხმარებელს საშუალებას აძლევს შეასრულოს ნებისმიერი მოქმედებები ამ ობიექტზე.

ანალოგიური სიტუაციაა სერვერის როლებთან დაკავშირებით. მაგალითად, ჩვენ არ შეგვიძლია კონკრეტულ მომხმარებელს მივცეთ სერვერის პროცესების მართვის უფლება. ეს შესაძლებელია მომხმარებლის ჩართვით processadmin როლში.

კონკრეტული მოქმედებისთვის, რომელიც უფლებებით იმართება, არსებობს მიმართვის მდგომარეობის სამი ვარიანტი: მინიჭება, აკრძალვა და არაცხადი უარყოფა.

მიმართვის აკრძალვა

სერვერის უსაფრთხოების სისტემას აქვს იერარქიული სტრუქტურა. ეს მონაცემთა ბაზის როლებს საშუალებას აძლევს შეიცავდეს Windows NT-ის სააღრიცხვო ჩანაწერებსა და ჯგუფებს, სერვერის მომხმარებლებსა და როლებს. მომხმარებელი, თავის მხრივ, შეიძლება რამდენიმე როლში მონაწილეობდეს. უსაფრთხოების სისტემის იერარქიული სტრუქტურის შედეგია ის, რომ ერთ მომხმარებელს ერთდროულად შეიძლება ჰქონდეს მიმართვის სხვადასხვა უფლებები სხვადასხვა როლებში. თუ ერთ-ერთ როლს, რომელშიც მომხმარებელი იმყოფება, აქვს მონაცემებთან მიმართვის უფლება, მაშინ მომხმარებელსაც ავტომატურად ექნება იგივე უფლებები. და მაინც, შეიძლება საჭირო გახდეს მონაცემებთან მიმართვის უფლების აკრძალვა. როცა მომხმარებელს ან Transact_SQL-ის ბრძანებებს ვუკრძალავთ მონაცემებთან მიმართვას (deny access), ამით ანულირდება მიმართვის ყველა უფლება, რომელიც მომხმარებელმა მიიღო იერარქიის ნებისმიერ დონეზე. ამასთან, მიმართვა დარჩება აკრძალული, მიუხედავად უფრო მაღალ დონეზე გაცემული ნებართვებისა.

დავუშვათ, მონაცემთა ბაზაში ვქმნით ცხრილს, რომელთანაც მიმართვა შეუძლია ყველა მომხმარებელს, მაგრამ გვინდა დროებით ავუკრძალოთ ზოგიერთ მომხმარებელს ცხრილთან მიმართვა. ამ შემთხვევაში, უმჯობესია, შევქმნათ როლი, რომელსაც აკრძალული ექნება ცხრილთან მიმართვა, და მასში ჩავრთოთ მომხმარებლები. მომხმარებლების დიდი რაოდენობისას ასეთი მიდგომა უსაფრთხოების სისტემის ადმინისტრირებას აადვილებს.

DENY ბრძანება გამოიყენება მომხმარებლისათვის მონაცემთა ბაზის ობიექტთან მიმართვის აკრძალვისათვის. მისი სინტაქსია:

DENY
{ ALL [PRIVILEGES] | *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]
[CASCADE]

DENY ბრძანების არგუმენტები GRANT ბრძანების არგუმენტების ანალოგიურია. CASCADE არგუმენტი საშუალებას გვაძლევს უფლებები ჩამოვართვათ არა მხოლოდ მოცემულ მომხმარებელს, არამედ იმ მომხმარებლებსაც, რომლებსაც ამ მომხმარებელმა გადასცა უფლებები.

მაგალითები.

ა. დავუშვათ User_3 მომხმარებელს ცხრილის შექმნის უფლება გადაეცა:

GRANT CREATE TABLE TO User_3 WITH GRANT OPTION

შემდეგ, User_3 მომხმარებელმა ანალოგიური უფლება სხვა მომხმარებლებლებს გადასცა. თუ შემდეგ დაგვჭირდა User_3 მომხმარებლისათვის უფლებების ჩამორთმევა, მაშინ უნდა შევასრულოთ ბრძანება:

DENY CREATE TABLE TO User_3 CASCADE

შედეგად, უფლებები ჩამოერთმევა იმ მომხმარებლებს, რომლებსაც User_3 მომხმარებელმა გადასცა უფლებები.

ბ. Saba და Ana მომხამრებლებს ავუკრძალოთ CREATE DATABASE და CREATE TABLE ბრძანებების შესრულება.

DENY CREATE DATABASE, CREATE TABLE TO Saba, Ana

გ. მაგალითში ცხრილის შექმნა ეკრძალება Roli_1 როლის წევრებს.

DENY CREATE TABLE TO Roli_1

დ. ობიექტთან მიმართვის აკრძალვა უფლებების იერარქიის შიგნით. მაგალითი გვიჩვენებს უფლებების მინიჭების მიმდევრობას. ჯერ public როლს ენიჭება SELECT უფლება, შემდეგ კი SELECT, INSERT, UPDATE და DELETE უფლებები ჩამოერთმევა Saba მომხმარებელსა და Roli_5 როლის წევრებს. მათ აღარ შეეძლებათ Personali ცხრილთან მიმართვა.

```
USE Shekveta
GO
GRANT SELECT ON Personali TO public
GO
DENY SELECT, INSERT, UPDATE, DELETE
ON Personali TO Saba, Roli_5
```

მიმართვის არაცხადი უარყოფა

მიმართვის მინიჭებისა და უარყოფის ოპერაციების გარდა, არსებობს კიდევ მიმართვის არაცხადი უარყოფის (revoke access) ოპერაცია. არაცხადი უარყოფა მიმართვის აკრძალვის მსგავსია, იმ განსხვავებით, რომ ის მოქმედებს მხოლოდ იმ დონეზე, რომელზეც განსაზღვრულია. თუ მომხმარებლისთვის განსაზღვრულ დონეზე უარყოფილია მიმართვა, მან ის მაინც შეიძლება მიიღოს იერარქიის სხვა დონეზე როლში წევრობის გზით, რომელსაც აქვს მიმართვის უფლება. ავტომატურად, მონაცემებთან მომხმარებლის მიმართვა არაცხადად უარყოფილია. ეს შეიძლება განვიხილოთ როგორც „შუალედური მდგომარეობა" მიმართვის უფლების მინიჭებასა და აკრძალვას შორის.

REVOKE ბრძანება გამოიყენება მონაცემთა ბაზის ობიექტებთან მიმართვის არაცხადი უარყოფისათვის:

REVOKE [GRANT OPTION FOR]
{ ALL [PRIVILEGES] | *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 | FROM } *უსაფრთხოების_სისტემის_ობიექტის_სახელი* [,...n]
[CASCADE]
[AS { *ჯგუფის_სახელი* | *როლის_სახელი* }]

Transact_SQL-ის ბრძანების შესრულების უფლების არაცხადი უარყოფისათვის გამოიყენება ბრძანება:

REVOKE { ALL | *ბრძანება* [,...n] } FROM *უსაფრთხოების_სისტემის_ობიექტის_სახელი* [,...n]

არგუმენტების დანიშნულება ისეთივეა, როგორიც GRANT და DENY ბრძანებებში. GRANT OPTION FOR არგუმენტი გამოიყენება მაშინ, როცა გვინდა იმ უფლების უარყოფა, რომელიც გაიცა GRANT ბრძანების WITH GRANT OPTION არგუმენტის საშუალებით. ამასთან, მომხმარებებლი ინარჩუნებს ობიექტთან მიმართვის უფლებას, მაგრამ კარგავს შესაძლებლობას ეს უფლება სხვა მომხმარებლებს გადასცეს.

დავუშვათ, User_4 მომხმარებელს, რომელიც არის Role_2 როლის წევრი, მიენიჭა Cxrili_2 ცხრილთან მიმართვის უფლებები. თუ REVOKE ბრძანებით უარვყავით Cxrili_2 ცხრილთან მიმართვა Role_2 როლისთვის, User_4 მომხმარებელი მაინც შესძლებს Cxrili_2 ცხრილთან მიმართვას, რადგან მისთვის მიმართვის უფლებები ცხადად არის განსაზღვრული. User_4 მომხმარებელი მხოლოდ მაშინ დაკარგავს Cxrili_2 ცხრილთან მიმართვის უფლებას, როცა REVOKE ბრძანებას გამოვიყენებთ პერსონალურად მის მიმართ.

მიმართვის არაცხადი უარყოფა იძლევა უსაფრთხოების სისტემის უფრო მოქნილად მართვის შესაძლებლობას.

მაგალითები.

ა. მაგალითში ხდება CREATE TABLE უფლების არაცხადი უარყოფა, რომელიც მინიჭებული ჰქონდათ Roli_6 და Roli_5 როლის წევრებს.

REVOKE CREATE TABLE FROM Roli_6, Roli_5

ბ. მაგალითში ხდება CREATE TABLE და CREATE DEFAULT უფლების არაცხადი უარყოფა Ana და Lika მომხმარებლებისთვის.

REVOKE CREATE TABLE, CREATE DEFAULT FROM Ana, Lika

მიმართვის კონფლიქტები

როლისთვის ან ჯგუფისთვის მინიჭებული უფლებები მემკვიდრეობით გადაეცემა მათ წევრებს, თუმცა მომხმარებელს ერთ როლში შეიძლება მიენიჭოს მიმართვის უფლებები, მეორე როლში - აეკრძალოს.

ცხრილი 16.4. შენახული პროცედურები და Transact_SQL-ის ბრძანებები, გამოყენებული უსაფრთხოების სისტემის მართვისათვის

Transact_SQL	დანიშნულება
sp_addapprole	როლის შექმნა პროგრამა-დანართისთვის
sp_addlogin	სერვერის ახალი სააღრიცხვო ჩანაწერის შექმნა
sp_addrole	ახალი როლის შექმნა მონაცემთა ბაზაში
sp_addrolemember	მონაცემთა ბაზის როლში წევრის დამატება
sp_addsrvrolemember	სერვერის ფიქსირებულ როლში ახალი წევრის დამატება
sp_approlepassword	პროგრამა-დანართის როლისთვის პაროლის შეცვლა
sp_defaultdb	სააღრიცხვო ჩანაწერისთვის ავტომატურად განსაზღვრული მონაცემთა ბაზის შეცვლა
sp_defaultlanguage	სააღრიცხვო ჩანაწერისთვის ავტომატურად განსაზღვრული ენის შეცვლა
sp_denylogin	Windows NT-ის მომხმარებლის ან ჯგუფისთვის მიმართვის აკრძალვა
sp_dropapprole	პროგრამა-დანართის როლის წაშლა
sp_droplinkedsrvlogin	სხვა სერვერისთვის სააღრიცხვო ჩანაწერის ასახვის წაშლა
sp_droplogin	სერვერის სააღრიცხვო ჩანაწერის წაშლა
sp_droprole	მონაცემთა ბაზის როლის წაშლა
sp_droprolemember	მონაცემთა ბაზის როლიდან მომხმარებლის წაშლა
sp_dropsrvrolemember	სერვერის როლიდან წევრის წაშლა
sp_dropuser	მონაცემთა ბაზიდან მომხმარებლის წაშლა
sp_grantdbaccess	სერვერის სააღრიცხვო ჩანაწერის მონაცემთა ბაზასთან მიმართვის უფლების მიცემა Windows NT-ის მომხმარებლის ან ჯგუფისთვის
sp_grantlogin	სერვერთან მიმართვის უფლების მიცემა
sp_helpdbfixedrole	მონაცემთა ბაზაში ფიქსირებული როლების სიის გაცემა
sp_helplogins	სააღრიცხვო ჩანაწერის შესახებ ინფორმაციის მიღება
sp_helpntgroup	Windows NT-ის ჯგუფების შესახებ ინფორმაციის ნახვა, რომლებსაც აქვთ სერვერთან მიმართვის უფლება
sp_helprole	ინფორმაციის მიღება მონაცემთა ბაზაში განსაზღვრული როლების შესახებ
sp_helpsrvrole	სერვერის ფიქსირებული როლების სიის გაცემა
sp_helpsrvrolemember	ინფორმაციის გაცემა სერვერის როლის წევრის შესახებ
sp_helpuser	მომხმარებლის შესახებ ინფორმაციის გაცემა
sp_password	ცვლის სერვერის სააღრიცხვო ჩანაწერის პაროლს
sp_revokelogin	შლის Windows NT-ის მომხმარებელს ან ჯგუფს
sp_setapprole	პროგრამა-დანართის როლის ინიციალიზება
GRANT	მიმართვის უფლების მიცემა
DENY	მიმართვის აკრძალვა
REVOKE	მიმართვის არაცხადი უარყოფა

მიმრთვის კონფლიქტის გადაწყვეტის დროს სერვერი ხელმძღვანელობს პრინციპით, რომელიც იმაში მდგომარეობს, რომ მიმართვის მინიჭების უფლებას აქვს ყველაზე დაბალი პრიორიტეტი, ხოლო მიმართვის აკრძალვის უფლებას - ყველაზე მაღალი. ეს იმას ნიშნავს, რომ მონაცემებთან მიმართვის უფლება შეიძლება მივიღოთ მხოლოდ მისი ცხადად მითითების გზით, მიმართვის აკრძალვის არარსებობისას უსაფრთხოების სისტემის იერარქიის ნებისმიერ დონეზე. თუ მიმართვა ცხადად არ არის მინიჭებული, მაშინ მომხმარებელი მონაცემებთან ვერ იმუშავებს.

უსაფრთხოების სისტემასთან სამუშაო პროცედურები და ბრძანებები

ცხრილში 16.4 მოყვანილია შენახული პროცედურებისა და Transact_SQL-ის ბრძანებების სია, განკუთვნილი უსაფრთხოების სისტემის მართვისთვის.

დანართი 1. სავარჯიშოები თავების მიხედვით

თავი 2. მონაცემთა ბაზები

2.1. შექმენით Baza_1 მონაცემთა ბაზა, რომელიც შედგება მონაცემების ერთი და ტრანზაქციების ჟურნალის ერთი ფაილებისგან. მონაცემების ფაილის საწყისი ზომაა 10 მბ, მაქსიმალური ზომა 50 მბ, ნაზრდი - 5 მბ. ტრანზაქციების ჟურნალის საწყისი ზომაა 5 მბ, მაქსიმალური ზომა 20 მბ, ნაზრდი - 3 მბ.
2.2. შექმენით Baza_2 მონაცემთა ბაზა, რომელიც შედგება მონაცემების ერთი და ტრანზაქციების ჟურნალის ერთი ფაილებისგან. მონაცემების ფაილისა და ტრანზაქციების ჟურნალის ზომა შეზღუდული არ არის.
2.3. შექმენით Baza_3 მონაცემთა ბაზა, რომელიც შედგება მონაცემების ერთი და ტრანზაქციების ჟურნალის ერთი ფაილებისგან. მონაცემების ფაილის საწყისი ზომაა 15 მბ. ტრანზაქციების ჟურნალის საწყისი ზომაა 7 მბ.
2.4. შექმენით Baza_4 მონაცემთა ბაზა, რომელიც შედგება მონაცემების ერთი და ტრანზაქციების ჟურნალის ერთი ფაილებისგან. მონაცემების ფაილის ნაზრდია 8 მბ. ტრანზაქციების ჟურნალის ნაზრდია 4 მბ.
2.5. შექმენით Baza_5 მონაცემთა ბაზა, რომელიც შედგება მონაცემების სამი და ტრანზაქციების ჟურნალის ორი ფაილებისგან. მონაცემების სამივე ფაილის საწყისი ზომაა 10 მბ, მაქსიმალური ზომა 50 მბ, ნაზრდი - 5 მბ. ტრანზაქციების ჟურნალის ორივე ფაილის საწყისი ზომაა 5 მბ, მაქსიმალური ზომა 20 მბ, ნაზრდი - 3 მბ.
2.6. შევქმნათ Baza_6 მონაცემთა ბაზა ფაილების ჯგუფების მითითების გზით. Baza6 შედგება ორი ჯგუფისაგან. თითოეულ ჯგუფში ორ-ორი ფაილია.
2.7. შექმენით Baza_7 მონაცემთა ბაზა. მისი სახელი შეცვალეთ Baza_71 სახელით (გამოიყენეთ sp_renamedb პროცედურა).
2.8. Baza_8 მონაცემთა ბაზა ნაგულისხმევი მნიშვნელობებით ჯერ შექმენით, შემდეგ კი - წაშალეთ.
2.9. შეასრულეთ Baza_2 მონაცემთა ბაზის გამორთვა.
2.10. შეასრულეთ Baza_2 მონაცემთა ბაზის მიერთება.
2.11. მიიღეთ ინფორმაცია Baza_3 მონაცემთა ბაზის შესახებ.
2.12. შექმენით მონაცემის ტიპი ფირმის თანამშრომლის ასაკის აღწერისათვის. მომხმარებლის ტიპი უნდა ინახავდეს NULL მნიშვნელობას.
2.13. შექმენით მომხმარებლის ტიპი ფირმის თანამშრომლის ხელფასის აღწერისათვის. მომხმარებლის ტიპი არ უნდა ინახავდეს NULL მნიშვნელობას.
2.14. წინა სავარჯიშოში შექმნილ მონაცემის ტიპს ახალი სახელი დაარქვით.

თავი 3. ცხრილები

3.1. შექმენით Avto მონაცემთა ბაზა ავტომანქანებისთვის. მასში მოათავსეთ Manqana_1 ცხრილი, რომლის პირველადი გასაღები იქნება სვეტი-მთვლელი. მთვლელის საწყისი მნიშვნელობა და ნაზრდი 1-ის ტოლია. ცხრილი უნდა შეიცავდეს შემდეგ ინფორმაციას: მანქანის მოდელი, გამომშვები ფირმის დასახელება, ქვეყანა, გამოშვების თარიღი, ფერი, კარების რაოდენობა, ძრავის ტიპი, ფასი.
3.2. Avto მონაცემთა ბაზაში მოათავსეთ Manqana_2 ცხრილი, რომლის პირველადი გასაღები იქნება სვეტი-მთვლელი. მთვლელის საწყისი მნიშვნელობა და ნაზრდი 1-ის ტოლია. პირველადი გასაღების მნიშვნელობები დახარისხებული უნდა იყოს კლებადობით. შექმენით

შეზღუდვა ცხრილის დონეზე. ცხრილი უნდა შეიცავდეს შემდეგ ინფორმაციას: მანქანის მარკა, გამომშვები ფირმის დასახელება, ქვეყანა, გამოშვების თარიღი, ფერი, კარების რაოდენობა, ძრავის ტიპი, ფასი.

3.3. შექმენით Aptiaqi მონაცემთა ბაზა აფთიაქისთვის. მასში მოათავსეთ Camlebi (წამლები) და Gayidvebi (გაყიდვები) ცხრილები. Camlebi ცხრილი შემდეგ ინფორმაციას უნდა შეიცავდეს: წამლის დასახელება, მწარმოებელი ფირმა, ქვეყანა, გამოშვების თარიღი, ვადა, რაოდენობა, წამლის დანიშნულება და ფასი. Gayidvebi ცხრილი შემდეგ ინფორმაციას უნდა შეიცავდეს: გაყიდული რაოდენობა, თანხა და გაყიდვის თარიღი. Gayidvebi დამოკიდებული ცხრილის გარე გასაღები Camlebi მთავარი ცხრილის პირველად გასაღებს დავუკავშირეთ. გარე გასაღები ერთი სვეტისგან შედგება.

3.4. Aptiaqi მონაცემთა ბაზაში მოათავსეთ Camlebi_1 (წამლები) და Gayidvebi_1 (გაყიდვები) ცხრილები. Camlebi_1 ცხრილი შემდეგ ინფორმაციას უნდა შეიცავდეს: წამლის დასახელება, მწარმოებელი ფირმა, ქვეყანა, გამოშვების თარიღი, ვადა, რაოდენობა, წამლის დანიშნულება და ფასი. Gayidvebi_1 ცხრილი შემდეგ ინფორმაციას უნდა შეიცავდეს: გაყიდული რაოდენობა, თანხა და გაყიდვის თარიღი. Gayidvebi_1 დამოკიდებული ცხრილის გარე გასაღები ორი სვეტისგან შედგება. ისინი დაუკავშირეთ Camlebi_1 მთავარი ცხრილის პირველად გასაღებს, რომელიც ასევე ორი სვეტისგან შედგება. შექმენით შეზღუდვა ცხრილის დონეზე, რომელიც განსაზღვრავს რეჟიმს, როცა მთავარი ცხრილიდან სტრიქონის წაშლისას დამოკიდებულ ცხრილში სტრიქონები წაიშლება.

3.5. შექმენით Kadrebi მონაცემთა ბაზა კადრების განყოფილებისთვის. მასში მოათავსეთ Tanamshromlebi_1 ცხრილი. ცხრილი შემდეგ ინფორმაციას უნდა შეიცავდეს: თანამშრომლის სახელი, გვარი, დაბადების თარიღი, სტაჟი, განყოფილების დასახელება, ხელფასი, ქალაქი, მისამართი, მობილური ტელეფონი. ცხრილის ზოგიერთი სვეტისთვის განსაზღვრეთ ნაგულისხმევი მნიშვნელობები.

3.6. Kadrebi მონაცემთა ბაზაში მოათავსეთ Tanamshromlebi_2 ცხრილი. ცხრილი შემდეგ ინფორმაციას უნდა შეიცავდეს: თანამშრომლის სახელი, გვარი, დაბადების თარიღი, სტაჟი, განყოფილების დასახელება, ხელფასი, ქალაქი, მისამართი, მობილური ტელეფონი. ცხრილის ზოგიერთი სვეტისთვის განსაზღვრეთ შეზღუდვები.

3.7. შექმენით Bina მონაცემთა ბაზა. მასში მოათავსეთ Binebi_1 ცხრილი. ის შემდეგ ინფორმაციას უნდა შეიცავდეს: ქალაქი, სადაც ბინა მდებარეობს, ქალაქის რაიონი, ქუჩის დასახელება, ქუჩის ნომერი, ქალაქის ტელეფონი, ფასი დოლარებში, ფასი ლარებში, კურსი, ბინის პროექტი, მფლობელის გვარი, მფლობელის სახელი. „ფასი ლარებში“ უნდა იყოს გამოთვლადი სვეტი.

3.8. Bina მონაცემთა ბაზაში მოათავსეთ Binebi_2 ცხრილი. ის შემდეგ ინფორმაციას უნდა შეიცავდეს: ქალაქი, სადაც ბინა მდებარეობს, ქალაქის რაიონი, ქუჩის დასახელება, ქუჩის ნომერი, ქალაქის ტელეფონი, ფასი დოლარებში, ფასი ლარებში, კურსი, პროექტის დასახელება, მფლობელის გვარი, მფლობელის სახელი. „მფლობელის გვარი“ სვეტი არ უნდა უშვებდეს NULL მნიშვნელობებს.

3.9. Kadrebi მონაცემთა ბაზაში მოათავსეთ Tanamshromlebi_3 ცხრილი. ცხრილი შემდეგ ინფორმაციას უნდა შეიცავდეს: თანამშრომლის სახელი, გვარი, დაბადების თარიღი, სტაჟი, განყოფილების დასახელება, ხელფასი, ქალაქი, მისამართი, მობილური ტელეფონი, სახლის ტელეფონი. „მობილური ტელეფონი“ და „სახლის ტელეფონი“ სვეტები უნიკალური უნიკალური უნდა იყოს.

3.10. შექმენით Magazia_1 მონაცემთა ბაზა პროდუქტების მაღაზიისთვის. მასში მოათავსეთ Produqtebi (პროდუქტები) და Gayidvebi (გაყიდვები) ცხრილები. Produqtebi ცხრილი უნდა შეიცავდეს შემდეგ ინფორმაციას: პროდუქტის დასახელება, რაოდენობა, მწარმოებელი ფირმა,

ქვეყანა, კურსი, ფასი დოლარებში, ფასი ლარებში. Gayidvebi ცხრილი უნდა შეიცავდეს შემდეგ ინფორმაციას: გაყიდვის თარიღი, თანხა და რაოდენობა. განსაზღვრეთ რეჟიმი, როცა Produqtebi მთავარი ცხრილიდან სტრიქონის წაშლისას Gayidvebi დამოკიდებულ ცხრილში სტრიქონები წაიშლება.

3.11. Magazia_1 მონაცემთა ბაზაში მოათავსეთ Produqtebi_1 (პროდუქტები) და Gayidvebi_1 (გაყიდვები) ცხრილები. Produqtebi_1 ცხრილი უნდა შეიცავდეს შემდეგ ინფორმაციას: პროდუქტის დასახელება, რაოდენობა, მწარმოებელი ფირმა, ქვეყანა, კურსი, ფასი დოლარებში, ფასი ლარებში. Gayidvebi_1 ცხრილი უნდა შეიცავდეს შემდეგ ინფორმაციას: გაყიდვის თარიღი, თანხა და რაოდენობა. განსაზღვრეთ რეჟიმი, როცა Produqtebi_1 მთავარი ცხრილიდან სტრიქონის წაშლისას Gayidvebi_1 დამოკიდებულ ცხრილში სტრიქონები არ წაიშლება.

3.12. Magazia_1 მონაცემთა ბაზაში მოათავსეთ Produqtebi_2 (პროდუქტები) და Gayidvebi_2 (გაყიდვები) ცხრილები. Produqtebi_2 ცხრილი უნდა შეიცავდეს შემდეგ ინფორმაციას: პროდუქტის დასახელება, რაოდენობა, მწარმოებელი ფირმა, ქვეყანა, კურსი, ფასი დოლარებში, ფასი ლარებში. Gayidvebi_2 ცხრილი უნდა შეიცავდეს შემდეგ ინფორმაციას: გაყიდვის თარიღი, თანხა და რაოდენობა. განსაზღვრეთ რეჟიმი, როცა Produqtebi_2 მთავარ ცხრილში პირველადი გასაღების მნიშვნელობის შეცვლისას შეიცვლება Gayidvebi_2 დამოკიდებული ცხრილის გარე გასაღების მნიშვნელობებიც.

3.13. Magazia_1 მონაცემთა ბაზაში მოათავსეთ Produqtebi_3 (პროდუქტები) და Gayidvebi_3 (გაყიდვები) ცხრილები. Produqtebi_3 ცხრილი უნდა შეიცავდეს შემდეგ ინფორმაციას: პროდუქტის დასახელება, რაოდენობა, მწარმოებელი ფირმა, ქვეყანა, კურსი, ფასი დოლარებში, ფასი ლარებში. Gayidvebi_3 ცხრილი უნდა შეიცავდეს შემდეგ ინფორმაციას: გაყიდვის თარიღი, თანხა და რაოდენობა. განსაზღვრეთ რეჟიმი, როცა Produqtebi_3 მთავარ ცხრილში პირველადი გასაღების მნიშვნელობის შეცვლისას Gayidvebi_3 დამოკიდებული ცხრილის გარე გასაღების მნიშვნელობები არ შეიცვლება.

3.14. Bina მონაცემთა ბაზაში მოათავსეთ Binebi_3 ცხრილი. ის შემდეგ ინფორმაციას უნდა შეიცავდეს: ქალაქი, სადაც ბინა მდებარეობს, ქალაქის რაიონი, ქუჩის დასახელება, ქუჩის ნომერი, ქალაქის ტელეფონი, ფასი დოლარებში, ფასი ლარებში, კურსი, პროექტის დასახელება, მფლობელის გვარი, მფლობელის სახელი. ცხრილს უნდა ჰქონდეს პირველადი გასაღები, რომლის მნიშვნელობების კლებადობით იქნება დალაგებული. შექმენით შეზღუდვა სვეტის დონეზე.

3.15. შექმენით Saavadmyofo მონაცემთა ბაზა საავადმყოფოსთვის. მასში მოათავსეთ Eqimi (ექიმები) და Avadmyofi (ავადმყოფები) ცხრილები. Eqimi ცხრილში მოათავსეთ შემდეგი ინფორმაცია: ექიმის გვარი და სახელი, განყოფილება, რომელშიც ის მუშაობს, თანამდებობა, სტაჟი, ხელფასი, დაბადების თარიღი, მობილური, ქალაქი, სადაც ის ცხოვრობს და მისამართი. Avadmyofi ცხრილში მოათავსეთ შემდეგი ინფორმაცია: ავადმყოფის გვარი და სახელი, განყოფილება, რომელშიც ის წევს, გადახდილი თანხა, დაბადების თარიღი, მობილური, ქალაქი, სადაც ის ცხოვრობს, მისამართი, დიაგნოზი და პალატის ნომერი. შექმენით შეზღუდვა სვეტის დონეზე, რომელიც განსაზღვრავს რეჟიმს, როცა Eqimi მთავარი ცხრილიდან სტრიქონის წაშლისას Avadmyofi დამოკიდებულ ცხრილში სტრიქონები წაიშლება.

3.16. Saavadmyofo მონაცემთა ბაზაში მოათავსეთ Eqimi_1 (ექიმები) და Avadmyofi_1 (ავადმყოფები) ცხრილები. Eqimi_1 ცხრილში მოათავსეთ შემდეგი ინფორმაცია: ექიმის გვარი და სახელი, განყოფილება, რომელშიც ის მუშაობს, თანამდებობა, სტაჟი, ხელფასი, დაბადების თარიღი, მობილური, ქალაქი, სადაც ის ცხოვრობს და მისამართი. Avadmyofi_1 ცხრილში მოათავსეთ შემდეგი ინფორმაცია: ავადმყოფის გვარი და სახელი, განყოფილება, რომელშიც ის წევს, გადახდილი თანხა, დაბადების თარიღი, მობილური, ქალაქი, სადაც ის ცხოვრობს, მისამართი, დიაგნოზი და პალატის ნომერი. შექმენით შეზღუდვა სვეტის დონეზე, რომელიც

განსაზღვრავს რეჟიმს, როცა Eqimi_1 მთავარი ცხრილიდან სტრიქონის წაშლისას Avadmyofi_1 დამოკიდებულ ცხრილში სტრიქონები არ წაიშლება.
3.17. Saavadmyofo მონაცემთა ბაზაში მოათავსეთ Eqimi_2 (ექიმები) და Avadmyofi_2 (ავადმყოფები) ცხრილები. Eqimi_2 ცხრილში მოათავსეთ შემდეგი ინფორმაცია: ექიმის გვარი და სახელი, განყოფილება, რომელშიც ის მუშაობს, თანამდებობა, სტაჟი, ხელფასი, დაბადების თარიღი, მობილური, ქალაქი, სადაც ის ცხოვრობს და მისამართი. Avadmyofi_2 ცხრილში მოათავსეთ შემდეგი ინფორმაცია: ავადმყოფის გვარი და სახელი, განყოფილება, რომელშიც ის წევს, გადახდილი თანხა, დაბადების თარიღი, მობილური, ქალაქი, სადაც ის ცხოვრობს, მისამართი, დიაგნოზი და პალატის ნომერი. შექმენით შეზღუდვა სვეტის დონეზე, რომელიც განსაზღვრავს რეჟიმს, როცა Eqimi_2 მთავარ ცხრილში პირველადი გასაღების მნიშვნელობის შეცვლისას შეიცვლება Avadmyofi_2 დამოკიდებული ცხრილის გარე გასაღების მნიშვნელობებიც.
3.18. Saavadmyofo მონაცემთა ბაზაში მოათავსეთ Eqimi_3 (ექიმები) და Avadmyofi_3 (ავადმყოფები) ცხრილები. Eqimi_3 ცხრილში მოათავსეთ შემდეგი ინფორმაცია: ექიმის გვარი და სახელი, განყოფილება, რომელშიც ის მუშაობს, თანამდებობა, სტაჟი, ხელფასი, დაბადების თარიღი, მობილური, ქალაქი, სადაც ის ცხოვრობს და მისამართი. Avadmyofi_3 ცხრილში მოათავსეთ შემდეგი ინფორმაცია: ავადმყოფის გვარი და სახელი, განყოფილება, რომელშიც ის წევს, გადახდილი თანხა, დაბადების თარიღი, მობილური, ქალაქი, სადაც ის ცხოვრობს, მისამართი, დიაგნოზი და პალატის ნომერი. შექმენით შეზღუდვა სვეტის დონეზე, რომელიც განსაზღვრავს რეჟიმს, როცა Eqimi_3 მთავარ ცხრილში პირველადი გასაღების მნიშვნელობის შეცვლისას არ შეიცვლება Avadmyofi_3 დამოკიდებული ცხრილის გარე გასაღების მნიშვნელობები.
3.19. შექმენით Biblioteka მონაცემთა ბაზა ბიბლიოთეკისათვის. მასში მოათავსეთ Cignebi (წიგნები) და Abonentebi (აბონენტები) ცხრილები. Cignebi ცხრილში მოათავსეთ შემდეგი ინფორმაცია: წიგნის დასახელება, ავტორის გვარი, გამოშვების წელი, ISBN კოდი, ტირაჟი, გამომცემლობა, ფასი, რეცენზენტი, ანოტაცია, გვერდების რაოდენობა. Abonentebi ცხრილში მოათავსეთ შემდეგი ინფორმაცია: გვარი, სახელი, ორგანიზაცია, სადაც მუშაობს, დაბადების თარიღი, პირადი ნომერი, მობილური, ქალაქი, მისამართი. შექმენით შეზღუდვა სვეტის დონეზე, რომელიც განსაზღვრავს რეჟიმს, როცა Cignebi მთავარ ცხრილში პირველადი გასაღების მნიშვნელობის შეცვლისას Abonentebi დამოკიდებულ ცხრილში გარე გასაღების მნიშვნელობები შესაბამისად შეიცვლება, ხოლო მთავარი ცხრილიდან სტრიქონის წაშლისას კი - დამოკიდებულ ცხრილში შესაბამისი სტრიქონები წაიშლება.
3.20. Biblioteka მონაცემთა ბაზაში მოათავსეთ Cignebi_1 (წიგნები) და Abonentebi_1 (აბონენტები) ცხრილები. Cignebi_1 ცხრილში მოათავსეთ შემდეგი ინფორმაცია: წიგნის დასახელება, ავტორის გვარი, გამოშვების წელი, ISBN კოდი, ტირაჟი, გამომცემლობა, ფასი, რეცენზენტი, ანოტაცია, გვერდების რაოდენობა. Abonentebi_1 ცხრილში მოათავსეთ შემდეგი ინფორმაცია: გვარი, სახელი, ორგანიზაცია, სადაც მუშაობს, დაბადების თარიღი, პირადი ნომერი, მობილური, ქალაქი, მისამართი. შექმენით შეზღუდვა ცხრილის დონეზე, რომელიც განსაზღვრავს რეჟიმს, როცა Cignebi_1 მთავარ ცხრილში პირველადი გასაღების მნიშვნელობის შეცვლისას Abonentebi_1 დამოკიდებულ ცხრილში გარე გასაღების მნიშვნელობები შესაბამისად შეიცვლება, ხოლო მთავარი ცხრილიდან სტრიქონის წაშლისას კი - დამოკიდებულ ცხრილში შესაბამისი სტრიქონები წაიშლება.
3.21. Kadrebi მონაცემთა ბაზის Tanamshromlebi_3 ცხრილს Tanamshromlebi_4 სახელი დაარქვით.
3.22. Kadrebi მონაცემთა ბაზის Tanamshromlebi_4 ცხრილი წაშალეთ.

თავი 4. მონაცემების მართვა

Transact SQL-ის საფუძვლები

4.1. ჩაწერეთ მოთხოვნა, რომელიც ‘Table’ ცხრილიდან გამოიტანს ‘SET’ და ‘firmis saxeli’ სვეტების მნიშვნელობებს.
4.2. გამოაცხადეთ მთელი, წილადი, სტრიქონული და თარიღის ტიპის მქონე ცვლადები და მიანიჭეთ მათ მნიშვნელობები.
4.3. ცვლადს მივანიჭოთ Personali ცხრილის სტაჟი სვეტის მინიმალური მნიშვნელობა SELECT ბრძანების გამოყენებით.
4.4. ცვლადს მივანიჭოთ Personali ცხრილის სტაჟი სვეტის მაქსიმალური მნიშვნელობა SET ბრძანების გამოყენებით.

INSERT ბრძანება

4.1.1. Shekveta ბაზის Shemkveti ცხრილს ერთი სტრიქონი დაუმატეთ. ჩასამატებელი მნიშვნელობების მიმდევრობა უნდა შეესაბამებოდეს ცხრილში სვეტების მიმდევრობას.
4.1.2. Shemkveti ცხრილს ერთი სტრიქონი დაუმატეთ. ჩასამატებელი მნიშვნელობები თქვენთვის საჭირო მიმდევრობით შეიტანეთ.
4.1.3. Shemkveti ცხრილს ერთი სტრიქონი დაუმატეთ. ზოგიერთ სვეტს NULL მნიშვნელობა მიანიჭეთ.
4.1.4. Shemkveti ცხრილს ერთი სტრიქონი დაუმატეთ. ზოგიერთ სვეტს DEFAULT (ნაგულისხმევი) მნიშვნელობა მიანიჭეთ.
4.1.5. Shemkveti ცხრილს ერთი სტრიქონი დაუმატეთ. ყველა სვეტს DEFAULT (ნაგულისხმევი) მნიშვნელობა მიანიჭეთ.
4.1.6. Shemkveti ცხრილს ორი სტრიქონი დაუმატეთ. გამოიყენეთ ორი INSERT ბრძანება.
4.1.7. ერთი INSERT ბრძანების გამოყენებით ცხრილს სამი სტრიქონი ჩაუმატეთ.
4.1.8. Shemkveti ცხრილიდან Shemkveti_1 ცხრილს ჩავუმატოთ მონაცემები თბილისელი შემკვეთების შესახებ. Shemkveti_1 ცხრილის shemkvetiID სვეტი არ უნდა იყოს პირველადი გასაღები და მის IDENTITY თვისებას უნდა ჰქონდეს NO მნიშვნელობა. Shemkveti_1 ცხრილი წინასწარ უნდა შექმნათ.
4.1.9 Shemkveti ცხრილიდან მონაცემები ბათუმელი შემკვეთების შესახებ მოვათავსოთ #Droebiti_Shemkveti დროებით ცხრილში. #Droebiti_Shemkveti დროებითი ცხრილი წინასწარ უნდა შექმნათ.

SELECT ... INTO ბრძანება

4.2.1. შევქმნათ Shemkveti_2 ცხრილი და მასში Shemkveti ცხრილიდან ჩავწეროთ gvari, saxeli, iuridiuli_fizikuri და firmis_dasaxeleba სვეტების მნიშვნელობები.
4.2.2. შევქმნათ Shemkveti_3 ცხრილი და მასში Shemkveti ცხრილიდან ჩავწეროთ თბილისელი თანამშრომლების გვარები.

SELECT ბრძანება

SELECT განყოფილება

4.3.1. Xelshekruleba ცხრილიდან გამოიტანეთ shemkvetiID სვეტის მნიშვნელობები.
4.3.2. Xelshekruleba ცხრილიდან გამოიტანეთ shemkvetiID სვეტის არაგამეორებადი მნიშვნელობები.
4.3.3. Shemkveti ცხრილიდან გამოიტანეთ ინფორმაცია პირველი 7 შემკვეთის შესახებ.

4.3.4. Shemkveti ცხრილიდან გამოიტანეთ სტრიქონების 60%.
4.3.5. Xelshekruleba ცხრილიდან გამოიტანეთ ხელშეკრულების ნომერი და გაფორმების წელი.
4.3.6. Shemkveti და Xelshekruleba ცხრილებიდან გამოიტანეთ შემკვეთების გვარები, სახელები და მათი ხელშეკრულებების ნომრები.

WHERE განყოფილება

4.4.1. გვაინტერესებს ინფორმაცია მევალეების შესახებ: გვარი, ფირმა, ტელეფონი.
4.4.2. გვაინტერესებს ინფორმაცია სამედიცინო განყოფილების თანამშრომლების შესახებ.
4.4.3. გვაინტერესებს ინფორმაცია თბილისელი თანამშრომლების შესახებ.
4.4.4. გვაინტერესებს ინფორმაცია დიდუბის რაიონში მცხოვრები თანამშრომლების შესახებ.
4.4.5. გვაინტერესებს ინფორმაცია იმ თანამშრომლების შესახებ, რომელთა ასაკი აღემატება 40 წელს.
4.4.6. გვაინტერესებს ინფორმაცია იმ თანამშრომლების შესახებ, რომელთა სტაჟი აღემატება 20 წელს.
4.4.7. გვაინტერესებს ინფორმაცია იმ თანამშრომლების შესახებ, რომელთა სტაჟი აღემატება 20 წელს და რომელთა ასაკი აღემატება 45 წელს.
4.4.8. გვაინტერესებს ინფორმაცია იმ თანამშრომლების შესახებ, რომლებიც მუშაობენ სამედიცინო განყოფილებაში და რომელთა ასაკი აღემატება 30 წელს.
4.4.9. გვაინტერესებს ინფორმაცია იმ თანამშრომლების შესახებ, რომლებიც მუშაობენ თბილისში და რომელთა სტაჟი აღემატება 15 წელს.
4.4.10. გვაინტერესებს ინფორმაცია იმ თანამშრომლების შესახებ, რომლებიც მუშაობენ სამედიცინო განყოფილებაში და რომელთა ხელფასი აღემატება 1500 ლარს.
4.4.11. გვაინტერესებს ის ხელშეკრულებები, რომელთა თანხა აღემატება 6500 ლარს.
4.4.12. გვაინტერესებს ის ხელშეკრულებები, რომელთა თანხა არ აღემატება 6500 ლარს.
4.4.13. გვაინტერესებს თბილისელ შემკვეთებს აქვთ თუ არა ვალი.
4.4.14. გვაინტერესებს ინფორმაცია იურიდიული პირების შესახებ.
4.4.15. გვაინტერესებს ინფორმაცია იმ თანამშრომლების შესახებ, რომელთა ასაკი მეტია 50-ზე ან ნაკლებია 30-ზე.
4.4.16. გვაინტერესებს ინფორმაცია იმ თანამშრომლების შესახებ, რომლებიც არ ცხოვრობენ თბილისში ან ბათუმში.
5.4.17. გვაინტერესებს ინფორმაცია იმ თანამშრომლების შესახებ, რომელთა სტაჟი 20 წელს აღემატება.
4.4.18. გვაინტერესებს ინფორმაცია იმ შემკვეთების შესახებ, რომლებსაც აქვთ ვალი.
4.4.19. გვაინტერესებს დაგვიანებით შესრულებული ხელშეკრულებები.
4.4.20. გვაინტერესებს ადრე შესრულებული ხელშეკრულებები.
4.4.21. გვაინტერესებს დროულად შესრულებული ხელშეკრულებები.
4.4.22. გვაინტერესებს ადრე დამთავრებული ხელშეკრულებები 2012 წლის განმავლობაში.
4.4.23. გვაინტერესებს დაგვიანებით დამთავრებული ხელშეკრულებები 2013 წლის განმავლობაში.
4.4.24. გვაინტერესებს დროულად დამთავრებული ხელშეკრულებები 2012 წლის განმავლობაში.
4.4.25. გვაინტერესებს ადრე დამთავრებული ხელშეკრულებები რომელიმე წლის განმავლობაში.
4.4.26. გვაინტერესებს დაგვიანებით დამთავრებული ხელშეკრულებები რომელიმე წლის განმავლობაში.
4.4.27. გვაინტერესებს დროულად დამთავრებული ხელშეკრულებები რომელიმე წლის

განმავლობაში.
4.4.28. გვაინტერესებს ადრე დამთავრებული ხელშეკრულებები რომელიმე განყოფილებაში.
4.4.29. გვაინტერესებს დაგვიანებით დამთავრებული ხელშეკრულებები რომელიმე განყოფილებაში.
4.4.30. გვაინტერესებს დროულად დამთავრებული ხელშეკრულებები რომელიმე განყოფილებაში.
4.4.31. გვაინტერესებს დამთავრებული შეკვეთები.
5.4.32. გვაინტერესებს დაუმთავრებელი შეკვეთები.
5.4.33. გვაინტერესებს შეუსრულებელი (დამთავრების ვადა (tarigi_damtavrebis < GETDATE() AND shesruleba = N'არა') გასულია და ხელშეკრულება არ დამთავრდა) შეკვეთები.
5.4.34. გვაინტერესებს დამთავრებული (დამთავრების ვადა (tarigi_damtavrebis < GETDATE() AND shesruleba = N'კი') გასულია და ხელშეკრულება დამთავრებულია) შეკვეთები მოცემული წლისთვის.
5.4.35. გვაინტერესებს დაუმთავრებელი (tarigi_damtavrebis > GETDATE() AND shesruleba = N'არა') შეკვეთები.
5.4.36. გვაინტერესებს შეუსრულებელი შეკვეთები მოცემული წლისთვის.
5.4.37. გვაინტერესებს ხელშეკრულებების ნომრები და მათი შესრულების დაგვიანების (თვეების მიხედვით) ხანგრძლივობა.
5.4.38. გვაინტერესებს ხელშეკრულებების ნომრები და რამდენი თვით ადრე შესრულდა თითოეული მათგანი.

BETWEEN ოპერატორი

4.5.1. გვაინტერესებს 40-50 ასაკის მქონე თანამშრომლები.
4.5.2. გვაინტერესებს 2005 წლის მარტის თვეში დაბადებული თანამშრომლები.
4.5.3. გვაინტერესებს 10-20 წლის სტაჟის მქონე თანამშრომლები.
4.5.4. გვაინტერესებს 1000-2000 ლარი ხელფასის მქონე თანამშრომლები.
4.5.5. გვაინტერესებს 500-1000 ლარი ვალის მქონე ხელშეკრულებები.
4.5.6. გვაინტერესებს ის ხელშეკრულებები, სადაც გადასახდელი თანხაა 4500-7000 ლარი.
4.5.7. გვაინტერესებს ის შემკვეთები, რომლებსაც გადასახდელი აქვთ 4500-7000 ლარი.
4.5.8. გვაინტერესებს ის შემკვეთები, რომლებსაც გადასახდელი ვალი აქვთ 500-1000 ლარი.
4.5.9. გვაინტერესებს ის თანამშრომლები, რომლებსაც გაფორმებული აქვთ ხელშეკრულება 3500-5000 ლარის ფარგლებში.
4.5.10. გვაინტერესებს ის თანამშრომლები, რომელთა მევალეების ვალია 500-1000 ლარის ფარგლებში.
4.5.11. გვაინტერესებს ის თანამშრომლები, რომლებსაც გაფორმებული აქვთ ხელშეკრულება 01.01.2000-დან 01.01.2005-მდე.
4.5.12. გვაინტერესებს ის შემკვეთები, რომლებსაც გაფორმებული აქვთ ხელშეკრულება 01.01.2000-დან 01.01.2005-მდე.
4.5.13. გვაინტერესებს დამთავრებული (დამთავრების ვადა (tarigi_damtavrebis < GETDATE() AND shesruleba = N'კი') გასულია და ხელშეკრულება დამთავრებულია) შეკვეთები მოცემული წლისთვის.
4.5.14. გვაინტერესებს ადრე დამთავრებული ხელშეკრულებები 2012 წლის განმავლობაში.
4.5.15. გვაინტერესებს დაგვიანებით დამთავრებული ხელშეკრულებები 2013 წლის განმავლობაში.
4.5.16. გვაინტერესებს დროულად დამთავრებული ხელშეკრულებები 2012 წლის განმავლობაში.

4.5.17. გვაინტერესებს ადრე დამთავრებული ხელშეკრულებები რომელიმე წლის განმავლობაში.
4.5.18. გვაინტერესებს დაგვიანებით დამთავრებული ხელშეკრულებები რომელიმე წლის განმავლობაში.
4.5.19. გვაინტერესებს დროულად დამთავრებული ხელშეკრულებები რომელიმე წლის განმავლობაში.
4.5.20. გვაინტერესებს 45-45 ასაკის მქონე თანამშრომელი ქალები.
4.5.21. გვაინტერესებს 30-40 ასაკის მქონე თანამშრომელი კაცები.
4.5.22. გვაინტერესებს 40-45 ასაკის მქონე თბილისელი თანამშრომელი ქალები.
4.5.23. გვაინტერესებს 30-40 ასაკის მქონე თელაველი თანამშრომელი კაცები.
4.5.24. გვაინტერესებს 5-10 სტაჟის მქონე თანამშრომელი ქალები.
4.5.25. გვაინტერესებს 3-7 სტაჟის მქონე თანამშრომელი კაცები.
4.5.26. გვაინტერესებს 9-20 სტაჟის მქონე თბილისელი თანამშრომელი ქალები.
4.5.27. გვაინტერესებს 2-10 სტაჟის მქონე თელაველი თანამშრომელი კაცები.
4.5.28. გვაინტერესებს 1000-2000 ლარი ხელფასის მქონე თანამშრომელი ქალები.
4.5.29. გვაინტერესებს 1000-2000 ლარი ხელფასის მქონე თანამშრომელი კაცები.
4.5.30. გვაინტერესებს 1000-2000 ლარი ხელფასის მქონე თანამშრომელი ქალები, რომელთა სტაჟი 6 წელს აღემატება.
4.5.31. გვაინტერესებს 1000-2000 ლარი ხელფასის მქონე თანამშრომელი კაცები, რომელთა სტაჟი 5 წელს აღემატება.

IN ოპერატორი

4.6.1. გვაინტერესებს ინფორმაცია ბათუმელი, ქობულეთელი ან გორელი თანამშრომლების შესახებ.
4.6.2. გვაინტერესებს ინფორმაცია ბათუმელი ან ქობულეთელი თანამშრომლების შესახებ.
4.6.3. გვაინტერესებს ინფორმაცია დიდუბის, ვაკისა ან საბურთალოს რაიონებში მცხოვრები თანამშრომლების შესახებ.
4.6.4. გვაინტერესებს ინფორმაცია ბათუმელი, ქობულეთელი ან გორელი შემკვეთების შესახებ.
4.6.5. გვაინტერესებს ინფორმაცია იმ თანამშრომლების შესახებ, რომელთა სტაჟია 5, 10 ან 15 წელი.
4.6.6. გვაინტერესებს ინფორმაცია იმ თანამშრომლების შესახებ, რომელთა ასაკია 25, 30 ან 35 წელი.
4.6.7. გვაინტერესებს ინფორმაცია იმ თანამშრომლების შესახებ, რომელთა მობილურია 593-277285, 555-123456 ან 599-012345.
4.6.8. გვაინტერესებს ინფორმაცია იმ თანამშრომლების შესახებ, რომელთა სახლის ტელეფონებია 221-11-00, 239-88-44 ან 234-79-92.
4.6.9. გვაინტერესებს ინფორმაცია იმ შემკვეთების შესახებ, რომლებიც მუშაობენ ფირმებში: 'Gia&Co.', 'Luka&Co.', 'Avto&Co.'.

LIKE ოპერატორი

4.7.1. გვაინტერესებს ის თანამშრომლები, რომელთა გვარის მესამე სიმბოლოა 'მ'. მოთხოვნას ექნება სახე.
4.7.2. გვაინტერესებს შემკვეთები, რომელთა ფირმების სახელები 'G' ასოთი იწყება.
4.7.3. გვაინტერესებს ის შემკვეთები, რომლებიც იმ ქალაქებში ცხოვრობენ, რომელთა სახელების პირველი ასოებია 'ხ', 'რ' ან 'ბ'.

4.7.4. გვაინტერესებს ის ფირმები, რომლებიც იმ ქალაქებში მდებარეობენ, რომელთა დასახელება არ იწყება 'ხ' და 'რ' ასოებით.
4.7.5. გვაინტერესებს ის შემკვეთები, რომლებიც ცხოვრობენ იმ ქალაქებში, რომელთა სახელების პირველი სიმბოლო მოთავსებულია 'მ' და 'ჯ' ასოებს შორის. მოთხოვნას ექნება სახე.
4.7.6. გვაინტერესებს ის შემკვეთები, რომელთა ელ. ფოსტა აქვთ გახსნილი სერვერზე @geonet.ge.
4.7.7. გვაინტერესებს ის შემკვეთები, რომელთა მობილური ტელეფონის ნომრები 574-ით იწყება. მოთხოვნას ექნება სახე.
4.7.8. გვაინტერესებს ის შემკვეთები, რომელთა სახლის ტელეფონის ნომრები 234-ით იწყება მოთხოვნას ექნება სახე.
4.7.9. გვაინტერესებს ის თანამშრომლები, რომელთა გვარები 'ძე'-ზე მთავრდება.
4.7.10. გვაინტერესებს ის თანამშრომლები, რომელთა გვარები "შვილი'-ზე მთავრდება.
4.7.11. გვაინტერესებს ის თანამშრომელი ქალები, რომელთა გვარები 'ძე'-ზე მთავრდება.
4.7.12. გვაინტერესებს ის თანამშრომელი კაცები, რომელთა გვარები "შვილი'-ზე მთავრდება.
4.7.13. გვაინტერესებს ის შემკვეთები, რომელთა გვარები 'ძე'-ზე მთავრდება.
4.7.14. გვაინტერესებს ის კაცი ფიზიკური პირები, რომელთა გვარები 'ძე'-ზე მთავრდება.
4.7.12. გვაინტერესებს ის ქალი ფიზიკური პირები, რომელთა გვარები "შვილი'-ზე მთავრდება.

აგრეგირების ფუნქციები
4.8.1. გვაინტერესებს ვალის მაქსიმალური და მინიმალური მნიშვნელობები.
4.8.2. გვაინტერესებს ფირმის თანამშრომლების საშუალო ასაკი.
4.8.3. გვაინტერესებს შემკვეთების რაოდენობა.
4.8.4. გვაინტერესებს შემკვეთების მთლიანი ვალი.
4.8.5. გვაინტერესებს შემკვეთების მაქსიმალური ვალი.
4.8.6. გვაინტერესებს შემკვეთების მინიმალური ვალი.
4.8.7. გვაინტერესებს ფირმის თანამშრომლების მაქსიმალური ასაკი.
4.8.8. გვაინტერესებს ფირმის თანამშრომლების მაინიმალური ასაკი.
4.8.9. გვაინტერესებს ფირმის თანამშრომლების მაქსიმალური სტაჟი.
4.8.10. გვაინტერესებს ფირმის თანამშრომლების მინიმალური სტაჟი.
4.8.11. გვაინტერესებს ფირმის თანამშრომლების საშუალო სტაჟი.
4.8.12. გვაინტერესებს დაგვიანებით შესრულებული ხელშეკრულებების რაოდენობა.
4.8.13. გვაინტერესებს ადრე შესრულებული ხელშეკრულებების რაოდენობა.
4.8.14. გვაინტერესებს დროულად შესრულებული ხელშეკრულებების რაოდენობა.
4.8.15. გვაინტერესებს დაგვიანებით შესრულებული ხელშეკრულებების რაოდენობა 2013 წლის განმავლობაში.
4.8.16. გვაინტერესებს ადრე შესრულებული ხელშეკრულებების რაოდენობა 2012 წლის განმავლობაში.
4.8.17. გვაინტერესებს დროულად შესრულებული ხელშეკრულებების რაოდენობა 2012 წლის განმავლობაში.
4.8.18. გვაინტერესებს დაგვიანებით შესრულებული ხელშეკრულებების საშუალო თანხა.
4.8.19. გვაინტერესებს ადრე შესრულებული ხელშეკრულებების საშუალო თანხა.
4.8.20. გვაინტერესებს დროულად შესრულებული ხელშეკრულებების საშუალო თანხა.
4.8.21. გვაინტერესებს დაგვიანებით შესრულებული ხელშეკრულებების მთლიანი თანხა.
4.8.22. გვაინტერესებს ადრე შესრულებული ხელშეკრულებების მთლიანი თანხა.
4.8.23 გვაინტერესებს დროულად შესრულებული ხელშეკრულებების მთლიანი თანხა.
4.8.24. გვაინტერესებს დაგვიანებით შესრულებული ხელშეკრულებების მინიმალური თანხა.

4.8.25. გვაინტერესებს ადრე შესრულებული ხელშეკრულებების მინიმალური თანხა.
4.8.26 გვაინტერესებს დროულად შესრულებული ხელშეკრულებების მინიმალური თანხა.
4.8.27. გვაინტერესებს დაგვიანებით შესრულებული ხელშეკრულებების მაქსიმალური თანხა.
4.8.28. გვაინტერესებს ადრე შესრულებული ხელშეკრულებების მაქსიმალურ თანხა.
4.8.29 გვაინტერესებს დროულად შესრულებული ხელშეკრულებების მაქსიმალურ თანხა.
4.8.30. გვაინტერესებს რომელიმე წლის განმავლობაში დაგვიანებით შესრულებული ხელშეკრულებების საშუალო თანხა.
4.8.31. გვაინტერესებს რომელიმე წლის განმავლობაში ადრე შესრულებული ხელშეკრულებების საშუალო თანხა.
4.8.32. გვაინტერესებს რომელიმე წლის განმავლობაში დროულად შესრულებული ხელშეკრულებების საშუალო თანხა.
4.8.33. გვაინტერესებს რომელიმე წლის განმავლობაში დაგვიანებით შესრულებული ხელშეკრულებების მთლიანი თანხა.
4.8.34. გვაინტერესებს რომელიმე წლის განმავლობაში ადრე შესრულებული ხელშეკრულებების მთლიანი თანხა.
4.8.35 გვაინტერესებს თანხა რომელიმე წლის განმავლობაში დროულად შესრულებული ხელშეკრულებების მთლიანი.
4.8.36. გვაინტერესებს რომელიმე წლის განმავლობაში დაგვიანებით შესრულებული ხელშეკრულებების მინიმალური თანხა.
4.8.37. გვაინტერესებს რომელიმე წლის განმავლობაში ადრე შესრულებული ხელშეკრულებების მინიმალური თანხა.
4.8.38 გვაინტერესებს რომელიმე წლის განმავლობაში დროულად შესრულებული ხელშეკრულებების მინიმალური თანხა.
4.8.49. გვაინტერესებს რომელიმე წლის განმავლობაში დაგვიანებით შესრულებული ხელშეკრულებების მაქსიმალური თანხა.
4.8.40. გვაინტერესებს რომელიმე წლის განმავლობაში ადრე შესრულებული ხელშეკრულებების მაქსიმალურ თანხა.
4.8.41. გვაინტერესებს რომელიმე წლის განმავლობაში დროულად შესრულებული ხელშეკრულებების მაქსიმალურ თანხა.
4.8.42. გვაინტერესებს სამედიცინო განყოფილებაში დაგვიანებით შესრულებული ხელშეკრულებების რაოდენობა.
4.8.43. გვაინტერესებს სამედიცინო განყოფილებაში ადრე შესრულებული ხელშეკრულებების რაოდენობა.
4.8.44. გვაინტერესებს სამედიცინო განყოფილებაში დროულად შესრულებული ხელშეკრულებების რაოდენობა.
4.8.45. გვაინტერესებს რომელიმე განყოფილებაში დაგვიანებით შესრულებული ხელშეკრულებების საშუალო თანხა.
4.8.50. გვაინტერესებს რომელიმე განყოფილებაში ადრე შესრულებული ხელშეკრულებების საშუალო თანხა.
4.8.51 გვაინტერესებს რომელიმე განყოფილებაში დროულად შესრულებული ხელშეკრულებების საშუალო თანხა.
4.8.52. გვაინტერესებს რომელიმე განყოფილებაში დაგვიანებით შესრულებული ხელშეკრულებების მთლიანი თანხა.
4.8.53. გვაინტერესებს რომელიმე განყოფილებაში ადრე შესრულებული ხელშეკრულებების მთლიანი თანხა.
4.8.54. გვაინტერესებს რომელიმე განყოფილებაში დროულად შესრულებული

ხელშეკრულებების მთლიანი თანხა.
4.8.55. გვაინტერესებს რომელიმე განყოფილებაში დაგვიანებით შესრულებული ხელშეკრულებების მინიმალური თანხა.
4.8.56. გვაინტერესებს რომელიმე განყოფილებაში ადრე შესრულებული ხელშეკრულებების მინიმალური თანხა.
4.8.57. გვაინტერესებს რომელიმე განყოფილებაში დროულად შესრულებული ხელშეკრულებების მინიმალური თანხა.
4.8.58. გვაინტერესებს რომელიმე განყოფილებაში დაგვიანებით შესრულებული ხელშეკრულებების მაქსიმალური თანხა.
4.8.59. გვაინტერესებს რომელიმე განყოფილებაში ადრე შესრულებული ხელშეკრულებების მაქსიმალურ თანხა.
4.8.60. გვაინტერესებს რომელიმე განყოფილებაში დროულად შესრულებული ხელშეკრულებების მაქსიმალური თანხა.
4.8.61. გვაინტერესებს დამთავრებული შეკვეთების შეკვეთების რაოდენობა.
4.8.62. გვაინტერესებს დაუმთავრებელი შეკვეთების შეკვეთების რაოდენობა.
4.8.63. გვაინტერესებს შეუსრულებელი შეკვეთების შეკვეთების რაოდენობა.
4.8.64. გვაინტერესებს დამთავრებული შეკვეთების რაოდენობა მოცემული წლისთვის.
4.8.65. გვაინტერესებს დაუმთავრებელი შეკვეთების რაოდენობა მოცემული წლისთვის.
4.8.66. გვაინტერესებს შეუსრულებელი შეკვეთების რაოდენობა მოცემული წლისთვის.
4.8.67. გვაინტერესებს მოცემული განყოფილებისთვის დამთავრებული შეკვეთების რაოდენობა.
4.8.68. გვაინტერესებს მოცემული განყოფილებისთვის დაუმთავრებელი შეკვეთების რაოდენობა.
4.8.69. გვაინტერესებს მოცემული განყოფილებისთვის შეუსრულებელი შეკვეთების რაოდენობა.
4.8.70. გვაინტერესებს ფირმის თანამშრომლების მაქსიმალური ხელფასი.
4.8.71. გვაინტერესებს ფირმის თანამშრომლების მინიმალური ხელფასი.
4.8.72. გვაინტერესებს ფირმის თანამშრომლების საშუალო ხელფასი.
4.8.73. გვაინტერესებს საშუალო ვალი.
4.8.74. გვაინტერესებს შეკვეთის მინიმალური თანხა.
4.8.75. გვაინტერესებს შეკვეთის მაქსიმალური თანხა.
4.8.76. გვაინტერესებს შეკვეთის საშუალო თანხა.
4.8.77. გვაინტერესებს შეკვეთის მთლიანი თანხა.
4.8.78. გვაინტერესებს შეკვეთების რაოდენობა.
4.8.79. გვაინტერესებს თანამშრომლების რაოდენობა.
4.8.80. გვაინტერესებს შემკვეთების რაოდენობა.
4.8.81. გვაინტერესებს თანამშრომელი ქალების რაოდენობა.
4.8.82. გვაინტერესებს თანამშრომელი კაცების რაოდენობა.
4.8.83. გვაინტერესებს შემკვეთების რაოდენობა.
4.8.84. გვაინტერესებს ფიზიკური პირები კაცების რაოდენობა.
4.8.85. გვაინტერესებს ფიზიკური პირები ქალების რაოდენობა.

GROUP BY განყოფილება

4.9.1. გვაინტერესებს შეკვეთის მინიმალური თანხა თითოეული თანამშრომლისთვის.
4.9.2. გვაინტერესებს შეკვეთის მაქსიმალური თანხა თითოეული თანამშრომლისთვის.
4.9.3. გვაინტერესებს შეკვეთის საშუალო თანხა თითოეული თანამშრომლისთვის.

4.9.4. გვაინტერესებს შეკვეთის მთლიანი თანხა თითოეული თანამშრომლისთვის.
5.9.5. გვაინტერესებს შეკვეთების რაოდენობა თითოეული თანამშრომლისთვის.
4.9.6. გვაინტერესებს თანამშრომლების საშუალო ხელფასი განყოფილებების მიხედვით.
4.9.7. გვაინტერესებს თანამშრომლების მაქსიმალური ხელფასი განყოფილებების მიხედვით.
4.9.8. გვაინტერესებს თანამშრომლების მინიმალური ხელფასი განყოფილებების მიხედვით.
4.9.9. გვაინტერესებს თანამშრომლების ჯამური ხელფასი განყოფილებების მიხედვით.
4.9.10. გვაინტერესებს თანამშრომლების რაოდენობა განყოფილებების მიხედვით.
4.9.11. გვაინტერესებს თანამშრომლების მინიმალური ასაკი განყოფილებების მიხედვით.
4.9.12. გვაინტერესებს თანამშრომლების მაქსიმალური ასაკი განყოფილებების მიხედვით.
4.9.13. გვაინტერესებს თანამშრომლების საშუალო ასაკი განყოფილებების მიხედვით.
4.9.14. გვაინტერესებს თანამშრომლების მინიმალური სტაჟი განყოფილებების მიხედვით.
4.9.15. გვაინტერესებს თანამშრომლების მაქსიმალური სტაჟი განყოფილებების მიხედვით.
4.9.16. გვაინტერესებს თანამშრომლების საშუალო სტაჟი განყოფილებების მიხედვით.
4.9.17. გვაინტერესებს თანამშრომლების რაოდენობა რეგიონების მიხედვით.
4.9.18. გვაინტერესებს თანამშრომლების რაოდენობა ქალაქების მიხედვით.
4.9.19. გვაინტერესებს განყოფილებების მიხედვით თანამშრომელი ქალების რაოდენობა.
4.9.20. გვაინტერესებს განყოფილებების მიხედვით თანამშრომელი ქალების მაქსიმალური ხელფასი.
4.9.21. გვაინტერესებს განყოფილებების მიხედვით თანამშრომელი ქალების მინიმალური ხელფასი.
4.9.22. გვაინტერესებს განყოფილებების მიხედვით თანამშრომელი ქალების საშუალო ხელფასი.
4.9.23. გვაინტერესებს განყოფილებების მიხედვით თანამშრომელი ქალების ჯამური ხელფასი.
4.9.24. გვაინტერესებს განყოფილებების მიხედვით თანამშრომელი ქალების მაქსიმალური სტაჟი.
4.9.25. გვაინტერესებს განყოფილებების მიხედვით თანამშრომელი ქალების მინიმალური სტაჟი.
4.9.26. გვაინტერესებს განყოფილებების მიხედვით თანამშრომელი ქალების საშუალო სტაჟი.
4.9.27. გვაინტერესებს განყოფილებების მიხედვითთანამშრომელი ქალების მაქსიმალური ასაკი.
4.9.28. გვაინტერესებს განყოფილებების მიხედვით თანამშრომელი ქალების მინიმალური ასაკი.
4.9.29. გვაინტერესებს განყოფილებების მიხედვით თანამშრომელი ქალების საშუალო ასაკი.
4.9.30. გვაინტერესებს განყოფილებების მიხედვით თანამშრომელი კაცების რაოდენობა.
4.9.31. გვაინტერესებს განყოფილებების მიხედვით თანამშრომელი კაცების მაქსიმალური ხელფასი.
4.9.32. გვაინტერესებს განყოფილებების მიხედვით თანამშრომელი კაცების მინიმალური ხელფასი.
4.9.33. გვაინტერესებს განყოფილებების მიხედვით თანამშრომელი კაცების საშუალო ხელფასი.
4.9.34. გვაინტერესებს განყოფილებების მიხედვით თანამშრომელი კაცების ჯამური ხელფასი.
4.9.35. გვაინტერესებს განყოფილებების მიხედვით თანამშრომელი კაცების მაქსიმალური სტაჟი.
4.9.36. გვაინტერესებს განყოფილებების მიხედვით თანამშრომელი კაცების მინიმალური სტაჟი.
4.9.37. გვაინტერესებს განყოფილებების მიხედვით თანამშრომელი კაცების საშუალო სტაჟი.
4.9.38. გვაინტერესებს განყოფილებების მიხედვით თანამშრომელი კაცების მაქსიმალური

ასაკი.
4.9.39. გვაინტერესებს განყოფილებების მიხედვით თანამშრომელი კაცების მინიმალური ასაკი.
4.9.40. გვაინტერესებს განყოფილებების მიხედვით თანამშრომელი კაცების საშუალო ასაკი.
4.9.41. გვაინტერესებს შემკვეთების რაოდენობა ქალაქების მიხედვით.
4.9.42. გვაინტერესებს ფიზიკური პირი ქალების რაოდენობა ქალაქების მიხედვით.
4.9.43. გვაინტერესებს კაცი ფიზიკური პირების რაოდენობა ქალაქების მიხედვით.
4.9.44. გვაინტერესებს ფიზიკური პირების რაოდენობა ქალაქების მიხედვით.
4.9.45. გვაინტერესებს იურიდიული პირების რაოდენობა ქალაქების მიხედვით.
4.9.46. გვაინტერესებს ფიზიკური პირი ქალების რაოდენობა ქალაქების მიხედვით.
4.9.47. გვაინტერესებს ფიზიკური პირი კაცების რაოდენობა ქალაქების მიხედვით.
4.9.48. გვაინტერესებს შემკვეთების მაქსიმალური ასაკი ქალაქების მიხედვით.
4.9.49. გვაინტერესებს შემკვეთების მინიმალური ასაკი ქალაქების მიხედვით.
4.9.50. გვაინტერესებს შემკვეთების საშუალო ასაკი ქალაქების მიხედვით.
4.9.51. გვაინტერესებს შემკვეთების რაოდენობა რეგიონების მიხედვით.
4.9.52. გვაინტერესებს შემკვეთი ქალების რაოდენობა რეგიონების მიხედვით.
4.9.53. გვაინტერესებს შემკვეთი კაცების რაოდენობარეგიონების მიხედვით.
4.9.54. გვაინტერესებს ფიზიკური პირების რაოდენობა რეგიონების მიხედვით.
4.9.55. გვაინტერესებს იურიდიული პირების რაოდენობა რეგიონების მიხედვით.
4.9.56. გვაინტერესებს ფიზიკური პირი ქალების რაოდენობა რეგიონების მიხედვით.
4.9.57. გვაინტერესებს ფიზიკური პირი კაცების რაოდენობა რეგიონების მიხედვით.
4.9.58. გვაინტერესებს შემკვეთების მაქსიმალური ასაკი რეგიონების მიხედვით.
4.9.59. გვაინტერესებს შემკვეთების მინიმალური ასაკი რეგიონების მიხედვით.
4.9.60. გვაინტერესებს შემკვეთების საშუალო ასაკი რეგიონების მიხედვით.
4.9.61. გვაინტერესებს თითოეული შემკვეთისთვის შეკვეთის საშუალო თანხა.
4.9.62. გვაინტერესებ თითოეული შემკვეთისთვისს შეკვეთის მაქსიმალური თანხა.
4.9.63. გვაინტერესებს თითოეული შემკვეთისთვის შეკვეთის მინიმალური თანხა.
4.9.64. გვაინტერესებს თითოეული შემკვეთისთვის შეკვეთის ჯამური თანხა.
4.9.65. გვაინტერესებს იურიდიული პირებისთვის შეკვეთის საშუალო თანხა.
4.9.66. გვაინტერესებს იურიდიული პირებისთვის შეკვეთის მაქსიმალური თანხა.
4.9.67. გვაინტერესებს იურიდიული პირებისთვის შეკვეთის მინიმალური თანხა.
4.9.68. გვაინტერესებს იურიდიული პირებისთვის შეკვეთის ჯამური თანხა.
4.9.69. გვაინტერესებს ფიზიკური პირებისთვის შეკვეთის საშუალო თანხა.
4.9.70. გვაინტერესებს ფიზიკური პირებისთვის შეკვეთის მაქსიმალური თანხა.
4.9.71. გვაინტერესებს ფიზიკური პირებისთვის შეკვეთის მინიმალური თანხა.
4.9.72. გვაინტერესებს ფიზიკური პირებისთვის შეკვეთის ჯამური თანხა.
4.9.73. გვაინტერესებს განყოფილებების მიხედვით შეუსრულებელი შეკვეთების რაოდენობა.
4.9.74. გვაინტერესებს წლების მიხედვით შეუსრულებელი შეკვეთების რაოდენობა.
4.9.75. გვაინტერესებს განყოფილებების მიხედვით დაუმთავრებელი შეკვეთების რაოდენობა.
4.9.76. გვაინტერესებს წლების მიხედვით დაუმთავრებელი შეკვეთების რაოდენობა.
4.9.77. გვაინტერესებს განყოფილებების მიხედვით დამთავრებული შეკვეთების რაოდენობა.
4.9.78. გვაინტერესებს წლების მიხედვით დამთავრებული შეკვეთების რაოდენობა.

HAVING განყოფილება

4.10.1. გვაინტერესებს თითოეული თანამშრომლის შეკვეთების მინიმალური თანხა, რომელიც 5000-ზე ნაკლებია.

4.10.2. გვაინტერესებს თითოეული თანამშრომლის შეკვეთების მაქსიმალური თანხა, რომელიც 6000-ზე მეტია.
4.10.3. გვაინტერესებს თითოეული თანამშრომლის შეკვეთების საშუალო თანხა, რომელიც 4000-ზე ნაკლებია.
4.10.4. გვაინტერესებს თანამშრომლების მინიმალური ხელფასი განყოფილებების მიხედვით, რომელიც 1000-ზე მეტია.
4.10.5. გვაინტერესებს თანამშრომლების მაქსიმალური ხელფასი განყოფილებების მიხედვით, რომელიც 5000-ზე ნაკლებია.
4.10.6. გვაინტერესებს თანამშრომლების საშუალო ხელფასი განყოფილებების მიხედვით, რომელიც 3000-ზე მეტია.
4.10.7. გვაინტერესებს თანამშრომლების რაოდენობა განყოფილებების მიხედვით, რომელიც 5-ზე მეტია.
4.10.8. გვაინტერესებს თანამშრომლების მინიმალური ასაკი განყოფილებების მიხედვით, რომელიც 30-ზე ნაკლებია.
4.10.9. გვაინტერესებს თანამშრომლების მაქსიმალური ასაკი განყოფილებების მიხედვით, რომელიც 30-ზე მეტია.
4.10.10. გვაინტერესებს თანამშრომლების საშუალო ასაკი განყოფილებების მიხედვით, რომელიც 35-ზე მეტია.
4.10.11. გვაინტერესებს თანამშრომლების მინიმალური სტაჟი განყოფილებების მიხედვით, რომელიც 20-ზე მეტია.
4.10.12. გვაინტერესებს თანამშრომლების მაქსიმალური სტაჟი განყოფილებების მიხედვით, რომელიც 25-ზე ნაკლებია.
4.10.13. გვაინტერესებს თანამშრომლების საშუალო სტაჟი განყოფილებების მიხედვით, რომელიც 40-ზე მეტია.
4.10.14. გვაინტერესებს თანამშრომლების რაოდენობა ქალაქების მიხედვით, რომელიც 10-ზე მეტია.
4.10.15. გვაინტერესებს იმ თანამშრომლების რაოდენობა, რომელთა სტაჟი ნაკლებია ბათუმელი თანამშრომლების საშუალო სტაჟზე.
4.10.16. გვაინტერესებს შემკვეთების რაოდენობა ქალაქების მიხედვით, რომელიც ერთს აღემატება.
4.10.17. გვაინტერესებს ქალი შემკვეთების რაოდენობა ქალაქების მიხედვით, რომელიც ორს აღემატება.
4.10.18. გვაინტერესებს კაცი შემკვეთების რაოდენობა ქალაქების მიხედვით, რომელიც ორს აღემატება.
4.10.19. გვაინტერესებს ფიზიკური პირების რაოდენობა რეგიონების მიხედვით, რომელიც სამს აღემატება.
4.10.20. გვაინტერესებს იურიდიული პირების რაოდენობა რეგიონების მიხედვით, რომელიც სამს აღემატება.
4.10.21. გვაინტერესებს შემკვეთების მაქსიმალური ასაკი რეგიონების მიხედვით, რომელიც 50-ზე ნაკლებია.
4.10.22. გვაინტერესებს შემკვეთების რაოდენობა სქესის მიხედვით, რომელიც ერთს აღემატება.
4.10.23. გვაინტერესებს შეკვეთის საშუალო თანხა თითოეული შემკვეთისთვის, რომელიც 5000 ლარს აღემატება.
4.10.24. გვაინტერესებს შეკვეთის მაქსიმალური თანხა თითოეული შემკვეთისთვის, რომელიც 7500 ლარს აღემატება.
4.10.25. გვაინტერესებს შეკვეთის მინიმალური თანხა თითოეული შემკვეთისთვის, რომელიც

3000 ლარს აღემატება.
4.10.26. გვაინტერესებს ქალი თანამშრომლების მაქსიმალური ხელფასი განყოფილებების მიხედვით, რომელიც 2000 ლარზე ნაკლებია.
4.10.27. გვაინტერესებს კაცი თანამშრომლების რაოდენობა განყოფილებების მიხედვით, რომელიც სამზე ნაკლებია.
4.10.28. გვაინტერესებს კაცი თანამშრომლების საშუალო ასაკი განყოფილებების მიხედვით, რომელიც 40 წელზე ნაკლებია.
4.10.29. გვაინტერესებს ქალი თანამშრომლების მინიმალური სტაჟი განყოფილებების მიხედვით, რომელიც 15 წელზე ნაკლებია.

ORDER BY განყოფილება
4.11.1. Personali ცხრილის „gvari“ და „saxeli“ სვეტები დაალაგეთ ზრდადობით, „asaki“ სვეტი - კლებადობით.
4.11.2. Personali ცხრილის „gvari“ და „saxeli“ სვეტები დაალაგეთ კლებადობით, „staji“ სვეტი - ზრდადობით.
4.11.3. Personali ცხრილის „gvari“ და „saxeli“ სვეტები დაალაგეთ ზრდადობით, „asaki“ და „staji“ სვეტები - კლებადობით.
4.11.4. Personali ცხრილის „qalaqi“ და „ganyofileba“ სვეტები დაალაგეთ კლებადობით, „staji“ სვეტი - ზრდადობით.
4.11.5. Shemkveti ცხრილის „qalaqi“ და „firmis_dasaxeleba“ სვეტები დაალაგეთ ზრდადობით, „raioni“ სვეტი - კლებადობით.
4.11.6. Shemkveti ცხრილის „qalaqi“ და „raioni“ სვეტები დაალაგეთ ზრდადობით, „firmis_dasaxeleba“ სვეტი - კლებადობით.
4.11.7. Xelshekruleba ცხრილის „dawyebis_tarigi“ სვეტი დაალაგეთ ზრდადობით, „gadasaxdeli_l“ სვეტი - კლებადობით.
4.11.8. Xelshekruleba ცხრილის „dawyebis_tarigi“ სვეტი დაალაგეთ კლებადობით, „gadasaxdeli_l“ სვეტი - ზრდადობით.
4.11.9. Xelshekruleba ცხრილის „shesruleba“ სვეტი დაალაგეთ ზრდადობით, „vali_l“ სვეტი - კლებადობით.
4.11.10. Shemkveti ცხრილის „regioni“ და „iuridiuli_fizikuri“ სვეტები დაალაგეთ ზრდადობით, „raioni“ სვეტი - კლებადობით.

COMPUTE განყოფილება
4.12.1. გვაინტერესებს ყველა იმ ხელშეკრულებით გადასახდელი თანხების ჯამი და ხელშეკრულებების რაოდენობა, რომლებშიც გადასახდელი თანხა 2000-ს აღემატება.
4.12.2. გვაინტერესებს ყველა იმ ხელშეკრულებით გადასახდელი თანხების ჯამი და ხელშეკრულებების რაოდენობა, რომლებშიც გადასახდელი თანხა 2000-ს აღემატება, გამოთვლის შედეგი უნდა დაჯგუფდეს.
4.12.3. გვაინტერესებს ყველა იმ ხელშეკრულებით გადასახდელი თანხების ჯამი და ხელშეკრულებების რაოდენობა, რომლებიც გაფორმდა 01.01.2000-დან 01.01.2005-მდე.
4.12.4. გვაინტერესებს ყველა იმ ხელშეკრულებით გადასახდელი თანხების საშუალო, მაქსიმალური და მინიმილური მნიშვნელობები, რომლებიც გაფორმდა 01.01.2000-დან 01.01.2005-მდე.
4.12.5. გვაინტერესებს ყველა იმ ხელშეკრულებით გადასახდელი თანხების საშუალო, მაქსიმალური და მინიმილური მნიშვნელობები, რომლებიც გაფორმდა 01.01.2000-დან 01.01.2005-მდე. დაჯგუფება შეასრულეთ personaliID სვეტის მოხედვით.

4.12.6. გვაინტერესებს თანამშრომლების საშუალო, მაქსიმალური და მინიმალური ხელფასი.
4.12.7. გვაინტერესებს თანამშრომლების საშუალო, მაქსიმალური და მინიმალური ხელფასი დაჯგუფება შეასრულეთ განყოფილებების მიხედვით.
4.12.8. გვაინტერესებს თანამშრომლების საშუალო, მაქსიმალური და მინიმალური ასაკი.
4.12.9. გვაინტერესებს თანამშრომლების საშუალო, მაქსიმალური და მინიმალური სტაჟი.
4.12.10. გვაინტერესებს თანამშრომლების საშუალო, მაქსიმალური და მინიმალური სტაჟი დაჯგუფება შეასრულეთ განყოფილების მიხედვით.
4.12.11. გვაინტერესებს ვალის მაქსიმალური და მინიმალური მნიშვნელობები.

OVER ელემენტი

4.13.1. გამოთვალეთ vali_l სვეტის მნიშვნელობების ჯამი და შედეგი მოათავსეთ saertovali_l სვეტში.
4.13.2. გამოთვალეთ vali_d სვეტის მნიშვნელობების ჯამი და შედეგი მოათავსეთ saertovali_d სვეტში.
4.13.3. გამოთვალეთ ხელშეკრულების თანხების (ლარებში) საერთო ჯამი და ხელშეკრულების თანხების (ლარებში) ჯამი თითოეული თანამშრომლისთვის.
4.13.4. გამოთვალეთ თითოეული თანამშრომლისთვის ხელშეკრულების თანხების (ლარებში) ჯამი.
4.13.5. გამოთვალეთ თითოეული თანამშრომლისთვის vali_l სვეტის მნიშვნელობების ჯამი.
4.13.6. გამოთვალეთ თითოეული შემკვეთისთვის vali_l სვეტის მნიშვნელობების ჯამი.
4.13.7. გამოთვალეთ თითოეული თანამშრომლისთვის vali_d სვეტის მნიშვნელობების ჯამი.
4.13.8. გამოთვალეთ თითოეული შემკვეთისთვის vali_d სვეტის მნიშვნელობების ჯამი.
4.13.9. გამოთვალეთ ვალის (ლარებში) პროცენტული წილი საერთო შედეგში (ჯამური ვალი).
4.13.10. გამოთვალეთ თითოეული შემსრულებლისთვის ვალის (ლარებში) პროცენტული წილი.
4.13.11. გამოთვალეთ თითოეული შემსრულებლისთვის ვალის (ლარებში) პროცენტული წილი საერთო შედეგში (ჯამური ვალი) და ვალის პროცენტული წილი.
4.13.12. გამოთვალეთ ვალის (დოლარებში) პროცენტული წილი საერთო შედეგში (ჯამური ვალი).
4.13.13. გამოთვალეთ თითოეული თანამშრომლისთვის ვალის (დოლარებში) პროცენტული წილი.
4.13.14. გამოთვალეთ თითოეული თანამშრომლისთვის ვალის (დოლარებში) პროცენტული წილი საერთო შედეგში (ჯამური ვალი) და ვალის (დოლარებში) პროცენტული წილი.
4.13.15. სამედიცინო განყოფილების თითოეული თანამშრომლისთვის გამოთვალეთ ხელშეკრულების თანხების (ლარებში) საერთო ჯამი და ხელშეკრულების თანხების (ლარებში) ჯამი.
4.13.16. სასპორტო განყოფილებისთვის გამოთვალეთ ხელშეკრულების თანხების (ლარებში) საერთო ჯამი.
4.13.17. სასოფლო განყოფილების თითოეული თანამშრომლისთვის გამოთვალეთ ხელშეკრულების თანხების (ლარებში) ჯამი.
4.13.18. სამედიცინო განყოფილების თითოეული თანამშრომლისთვის გამოთვალეთ ხელშეკრულების თანხების (დოლარებში) საერთო ჯამი და ხელშეკრულების თანხების (დოლარებში) ჯამი.
4.13.19. სასპორტო განყოფილებისთვის გამოთვალეთ ხელშეკრულების თანხების (დოლარებში) საერთო ჯამი.
4.13.20. სასოფლო განყოფილების თითოეული თანამშრომლისთვის გამოთვალეთ

ხელშეკრულების თანხების (დოლარებში) ჯამი.
4.13.21. სამედიცინო განყოფილებისთვის გამოთვალეთ ვალის (ლარებში) პროცენტული წილი საერთო შედეგში (ჯამური ვალი).
4.13.22. სამედიცინო განყოფილების თითოეული შემსრულებლისთვის გამოთვალეთ ვალის (ლარებში) პროცენტული წილი.
4.13.23. სამედიცინო განყოფილების თითოეული შემსრულებლისთვის გამოთვალეთ ვალის (ლარებში) პროცენტული წილი საერთო შედეგში (ჯამური ვალი) და ვალის პროცენტული წილი.
4.13.24. სამედიცინო განყოფილებისთვის გამოთვალეთ ვალის (დოლარებში) პროცენტული წილი საერთო შედეგში (ჯამური ვალი).
4.13.25. სამედიცინო განყოფილების თითოეული თანამშრომლისთვის გამოთვალეთ ვალის (დოლარებში) პროცენტული წილი.
4.13.26. სამედიცინო განყოფილების თითოეული თანამშრომლისთვის გამოთვალეთ ვალის (დოლარებში) პროცენტული წილი საერთო შედეგში (ჯამური ვალი) და ვალის (დოლარებში) პროცენტული წილი.

NULL მნიშვნელობასთან მუშაობა

4.14.1. გვაინტერესებს ინფორმაცია თანამშრომლების შესახებ ყველა რეგიონიდან ქართლის გარდა.
4.14.2. გვაინტერესებს ინფორმაცია თანამშრომლების შესახებ ყველა რეგიონიდან აჭარის გარდა.
4.14.3. გვაინტერესებს ინფორმაცია თანამშრომლების შესახებ ყველა რაიონიდან ვერის გარდა.
4.14.4. გვაინტერესებს ინფორმაცია თანამშრომლების შესახებ ყველა რაიონიდან ვაკის გარდა.
4.14.5. გვაინტერესებს ინფორმაცია შემკვეთების შესახებ ყველა რაიონიდან საბურთალოს გარდა.
4.14.6. გვაინტერესებს ინფორმაცია შემკვეთების შესახებ ყველა რაიონიდან დიდუბის გარდა.
4.14.7. გვაინტერესებს ის შემკვეთები, რომლებსაც firmis_dasaxeleba სვეტი ცარიელი არ აქვთ.
4.14.8. გვაინტერესებს ის შემკვეთები, რომლებსაც firmis_dasaxeleba სვეტი ცარიელი აქვთ.

UPDATE ბრძანება

4.15.1. თითოეული თანამშრომლის ხელფასი გაზარდეთ 200 ლარით.
4.15.2. Xelshekruleba ცხრილში გადასახდელი თანხა შეცვალეთ 8000-ით იმ ხელშეკრულებებისათვის, რომელთა ნომრებია 3 და 9.
4.15.3. გამოთვალეთ გადასახდელი თანხა და ვალი დოლარებში, ლარებში მათი მნიშვნელობების კურსზე გაყოფის გზით.

DELETE ბრძანება

4.16.1. Xelshekruleba ცხრილი გადაწერეთ Xelshekruleba_1 ცხრილში. Xelshekruleba_1 ცხრილიდან წაშალეთ ის ხელშეკრულებები, რომლებშიც ვალი ნულის ტოლია.
4.16.2. Xelshekruleba_1 ცხრილიდან წაშალეთ ყველა სტრიქონი.

მონაცემების მასობრივი გადაწერა

4.17.1. Shekveta მონაცემთა ბაზის Shemkveti ცხრილიდან ტექსტურ ფაილში გადაწერეთ ყველა სტრიქონი. მონაცემები უნდა დამუშავდეს Unicode ფორმატში. გამოვიყენოთ Windows NT-ის სააღრიცხვო ჩანაწერი.
4.17.2. Shekveta მონაცემთა ბაზის Personali ცხრილიდან ტექსტურ ფაილში გადაწერეთ

მონაცემები იმ თანამშრომლების შესახებ, რომელთა სტაჟი 20-ს აღემატება. გამოვიყენოთ SELECT მოთხოვნა და Windows NT-ის სააღრიცხვო ჩანაწერი.
4.17.3. შეასრულეთ ტექსტური ფაილიდან სტრიქონების ჩასმა Shekveta.dbo.Shemkveti_1 ცხრილში. ამისათვის, ჯერ შექმენით Shemkveti_1 ცხრილი, შემდეგ კი შეასრულეთ სტრიქონების ჩასმა. მიუთითეთ სერვერის სახელი და პაროლი.
4.17.4. Shekveta მონაცემთა ბაზის Personali ცხრილიდან ტექსტურ ფაილში გადაწერეთ სტრიქონები, რომელთა ნომრებია 3-6. მივუთითოთ სერვერის სახელი და პაროლი.
4.17.5. შეასრულეთ ტექსტური ფაილიდან 2-5 ნომრის მქონე სტრიქონების ჩასმა Shekveta.dbo.Shemkveti_1 ცხრილში. მივუთითოთ სერვერის სახელი და პაროლი.

თავი 5. შეერთებები

ჯვარედინი შეერთებები

5.1.1. Shemkveti და Xelshekruleba ცხრილებს შორის CROSS JOIN კავშირი დაამყარეთ.
5.1.2. შეასრულეთ Shemkveti ცხრილის ორი ეგზემპლარის ჯვარედინი თვითშეერთება.

შიგა შეერთებები

5.2.1. Shemkveti და Xelshekruleba ცხრილებს შორის INNER კავშირი დაამყარეთ.
5.2.2. გვაინტერესებს, ინფორმაცია მევალეების შესახებ.
5.2.3. გვაინტერესებს, ინფორმაცია ფიზიკურ პირებთან გაფორმებული ხელშეკრულებების შესახებ.
5.2.4. გვაინტერესებს, ინფორმაცია იურიდიულ პირებთან გაფორმებული ხელშეკრულებების შესახებ.
5.2.5. გვაინტერესებს, ინფორმაცია თბილისელ შემკვეთებთან გაფორმებული ხელშეკრულებების შესახებ.
5.2.6. გვაინტერესებს, ინფორმაცია ბათუმელ თანამშრომლებთან გაფორმებული ხელშეკრულებების შესახებ.
5.2.7. გვაინტერესებს, ინფორმაცია სამედიცინო განყოფილების თანამშრომლებთან გაფორმებული ხელშეკრულებების შესახებ.
5.2.8. გვაინტერესებს, სამედიცინო განყოფილების თანამშრომლებთან გაფორმებული ხელშეკრულებების რაოდენობა.
5.2.9. გვაინტერესებს, ბათუმელ თანამშრომლებთან გაფორმებული ხელშეკრულებების თანხებს შორის მაქსიმალური.
5.2.10. გვაინტერესებს, თბილისელ შემკვეთებთან გაფორმებული ხელშეკრულებების თანხების ჯამი.

გარე შეერთებები

5.3.1. Shemkveti და Xelshekruleba ცხრილებს შორის LEFT OUTER კავშირი დაამყარეთ.
5.3.2. Shemkveti და Xelshekruleba ცხრილებს შორის RIGHT OUTER კავშირი დაამყარეთ.
5.3.3. Shemkveti და Xelshekruleba ცხრილებს შორის FULL OUTER კავშირი დაამყარეთ.
5.3.4. Personali და Xelshekruleba ცხრილებს შორის დაამყარეთ LEFT OUTER კავშირი.
5.3.5. Personali და Xelshekruleba ცხრილებს შორის დაამყარეთ RIGHT OUTER კავშირი.
5.3.6. Personali და Xelshekruleba ცხრილებს შორის დაამყარეთ FULL OUTER კავშირი.

თავი 6. მმართველი კონსტრუქციები

IF...ELSE

6.1.1. გვაინტერესებს, აღემატება თუ არა მითითებული გვარის მქონე თანამშრომლის ასაკი 30 წელს.
6.1.2. გვაინტერესებს, აღემატება თუ არა მითითებული გვარის მქონე თანამშრომლის სტაჟი 17 წელს.
6.1.3. გვაინტერესებს, აღემატება თუ არა მითითებული თანამშრომლის ხელფასი 1500 ლარს.
6.1.4. გვაინტერესებს, არის თუ არა მითითებული გვარის თანამშრომელი დაბადებული მითითებულ თარიღში.
6.1.5. გვაინტერესებს, ცხოვრობს თუ არა მითითებული გვარის თანამშრომელი მითითებულ რაიონში.
6.1.6. გვაინტერესებს, მუშაობს თუ არა მითითებული გვარის თანამშრომელი მითითებულ განყოფილებაში.
6.1.7. გვაინტერესებს, არის თუ არა მითითებული მობილური მითითებული გვარის თანამშრომლის ტელეფონი.
6.1.8. გვაინტერესებს, ცხოვრობს თუ არა მითითებული გვარის თანამშრომელი მითითებულ ქალაქში.
6.1.9. გვაინტერესებს, მითითებული გვარის შემკვეთი იურიდიული პირია თუ ფიზიკური.
6.1.10. გვაინტერესებს, მითითებული გვარის შემკვეთი მუშაობს თუ არა მითითებულ ფირმაში.

CASE...END

6.2.1. გვაინტერესებს, ცხოვრობს თუ არა მითითებული გვარის თანამშრომელი მითითებულ რაიონში.
6.2.2. გვაინტერესებს, ცხოვრობს თუ არა მითითებული გვარის თანამშრომელი მითითებულ ქალაქში.
6.2.3. გვაინტერესებს, რომელ თვეში დაიბადა მითითებული გვარის თანამშრომელი.
6.2.4. გვაინტერესებს, რომელ თვეში დაიბადა თითოეული თანამშრომელი.
6.2.5. გვაინტერესებს, რომელ წელს დაიბადა თითოეული თანამშრომელი.
6.2.6. გვაინტერესებს, რომელ განყოფილებაში მუშაობს თითოეული თანამშრომელი.
6.2.7. გვაინტერესებს, თანამშრომლების მისამართები რაიონების მიხედვით.
6.2.8. გვაინტერესებს, თანამშრომლის ასაკის მიხედვით გამოიტანოს მისი სტაჟი.
6.2.9. გვაინტერესებს, თანამშრომლების ტელეფონები რაიონების მიხედვით.
6.2.10. გვაინტერესებს, თანამშრომლების მისამართები ქალაქების მიხედვით.

WHILE...BREAK & CONTINUE

6.3.1. 10%-ით გაზარდეთ თანამშრომლების ხელფასი მანამ, სანამ მათი საშუალო ხელფასი 3000 ლარზე ნაკლებია.
6.3.2. 15%-ით გაზარდეთ თანამშრომლების ხელფასი მანამ, სანამ მათი მაქსიმალური ხელფასი 500 ლარზე ნაკლებია.
6.3.3. 15%-ით გაზარდეთ თანამშრომლების ხელფასი მანამ, სანამ მათი მინიმალური ხელფასი 400 ლარზე ნაკლებია.
6.3.4. 10%-ით გაზარდეთ თანამშრომლების ხელფასი მანამ, სანამ მათი ჯამი 4000 ლარზე ნაკლებია.
6.3.5. 20%-ით გაზარდეთ თანამშრომლების ხელფასი მანამ, სანამ მათი ჯამი 6000 ლარზე ნაკლებია.

თავი 7. ქვემოთხოვნები

ჩადგმული მოთხოვნები

7.1.1. დავუშვათ, ვიცით ფირმის თანამშრომლის გვარი და სახელი - 'სამხარაძე საბა', მაგრამ არ ვიცით მისი კოდი - personaliID და გვაინტერესებს მისი ყველა შემკვეთი.
7.1.2. გვაინტერესებს ის თანამშრომლები, რომლებიც იმ ქალაქში ცხოვრობენ, რომელშიც ცხოვრობს 'სამხარაძე საბა'.
7.1.3. გვაინტერესებს ინფორმაცია მინიმალური ვალის მქონე ხელშეკრულების შესახებ.
7.1.4. გვაინტერესებს ხელშეკრულებები, რომლებიც გააფორმეს თბილისელმა თანამშრომლებმა.
7.1.5. გვაინტერესებს ხელშეკრულებები, რომლებიც არ გააფორმეს თბილისელმა თანამშრომლებმა.
7.1.6. გვაინტერესებს ხელშეკრულებები, რომლებიც გააფორმეს თბილისელმა შემკვეთებებმა.
7.1.7. გვაინტერესებს ხელშეკრულებები, რომლებიც არ გააფორმეს თბილისელმა შემკვეთებებმა.
7.1.8. გვაინტერესებს ხელშეკრულებები, რომლებიც გააფორმდა სამედიცინო განყოფილებაში.
7.1.9. გვაინტერესებს ხელშეკრულებები, რომლებიც არ გააფორმდა სამედიცინო განყოფილებაში.
7.1.10. დავუშვათ, ვიცით შემკვეთის გვარი და სახელი - 'სეხნიაშვილი გია', მაგრამ არ ვიცით მისი კოდი - shemkvetiID და გვაინტერესებს მისი შემსრულებლები.
7.1.11. გვაინტერესებს ის შემკვეთები, რომლებიც იმ ქალაქში ცხოვრობენ, რომელშიც ცხოვრობს 'სეხნიაშვილი გია'.
7.1.12. გვაინტერესებს ინფორმაცია მაქსიმალური ვალის მქონე ხელშეკრულების შესახებ.
7.1.13. დავუშვათ, ვიცით ფირმის დასახელება და არ ვიცით მისი კოდი. გვაინტერესებს ინფორმაცია მის მიერ გაფორმებული ხელშეკრულებების შესახებ.

ბმული მოთხოვნები

7.2.1. გვაინტერესებს ის თანამშრომლები, რომლებმაც ხელშეკრულება 2011 წლის 1 იანვარს გააფორმეს.
7.2.2. გვაინტერესებს ის თანამშრომლები, რომლებსაც სამზე ნაკლები შემკვეთი ჰყავს.
7.2.3. გვაინტერესებს ის თანამშრომლები, რომლებსაც ორზე მეტი შემკვეთი ჰყავს.
7.2.4. გვაინტერესებს ის შემკვეთები, რომლებსაც ორზე ნაკლები შემსრულებელი ჰყავს.
7.2.5. გვაინტერესებს ის შემკვეთები, რომლებსაც ორზე მეტი შემსრულებელი ჰყავს.
7.2.6. გვაინტერესებს ის შემკვეთები, რომლებმაც ხელშეკრულება 2011 წლის 1 იანვარს გააფორმეს.

ALL ოპერატორი

7.3.1. გვაინტერესებს, ყველა თანამშრომლის ასაკი აღემატება თუ არა 30 წელს.
7.3.2. გვაინტერესებს, ყველა ხელშეკრულება შესრულდა თუ არა. თუ არა, მაშინ რომელი ხელშეკრულებები არ შესრულდა.
7.3.3. გვაინტერესებს, ყველა თანამშრომლის სტაჟი აღემატება თუ არა 20 წელს.
7.3.4. გვაინტერესებს, ყველა თანამშრომლის ხელფასი აღემატება თუ არა 700 ლარს.
7.3.5. გვაინტერესებს, ყველა თანამშრომლის ასაკი აღემატება თუ არა თანამშრომლების საშუალო ასაკს.

7.3.6. გვაინტერესებს, ყველა თანამშრომლის ხელფასი აღემატება თუ არა თანამშრომლების საშუალო ხელფასს.
7.3.7. გვაინტერესებს, ყველა თანამშრომლის სტაჟი აღემატება თუ არა თანამშრომლების საშუალო სტაჟს.
7.3.8. გვაინტერესებს, ყველა ხელშეკრულების თანხა აღემატება თუ არა 5000 ლარს.
7.3.9. გვაინტერესებს, ყველა ხელშეკრულების თანხა აღემატება თუ არა ხელშეკრულებების საშუალო თანხას.

SOME და ANY ოპერატორები

7.4.1. გვაინტერესებს, რომელიმე თანამშრომლის ასაკი აღემატება თუ არა 20 წელს.
7.4.2. გვაინტერესებს, რომელიმე ხელშეკრულება შესრულდა თუ არა.
7.4.3. გვაინტერესებს, რომელიმე თანამშრომლის სტაჟი აღემატება თუ არა 50 წელს.
7.4.4. გვაინტერესებს, რომელიმე თანამშრომლის ხელფასი აღემატება თუ არა 800 ლარს.
7.4.5. გვაინტერესებს, რომელიმე თანამშრომლის ასაკი აღემატება თუ არა თანამშრომლების საშუალო ასაკს.
7.4.6. გვაინტერესებს, რომელიმე თანამშრომლის ხელფასი აღემატება თუ არა თანამშრომლების საშუალო ხელფასს.
7.4.7. გვაინტერესებს, რომელიმე თანამშრომლის სტაჟი აღემატება თუ არა თანამშრომლების საშუალო სტაჟს.
7.4.8. გვაინტერესებს, რომელიმე ხელშეკრულების თანხა აღემატება თუ არა 4000 ლარს.
7.4.9. გვაინტერესებს, რომელიმე ხელშეკრულების თანხა აღემატება თუ არა ხელშეკრულებების საშუალო თანხას.

EXISTS ოპერატორი

7.5.1. მივიღოთ ინფორმაცია ყველა შემკვეთის შესახებ, თუ მათ შორის არის ერთი ფიზიკური პირი მაინც.
7.5.2. მივიღოთ ინფორმაცია ყველა შემკვეთის შესახებ, თუ მათ შორის არის ერთი თბილისელი მაინც.
7.5.3. მივიღოთ ინფორმაცია ყველა თანამშრომლის შესახებ, თუ მათ შორის არის ერთი თბილისელი მაინც.
7.5.4. მივიღოთ ინფორმაცია ყველა თანამშრომლის შესახებ, თუ მათ შორის არის ერთი მაინც, რომლის ასაკი 30 წელს აღემატება.
7.5.5. მივიღოთ ინფორმაცია ყველა თანამშრომლის შესახებ, თუ მათ შორის არის ერთი მაინც, რომლის სტაჟი 15 წელს აღემატება.
7.5.6. მივიღოთ ინფორმაცია ყველა თანამშრომლის შესახებ, თუ მათ შორის არის ერთი მაინც, რომლის ხელფასი 1500 ლარს აღემატება.
7.5.7. მივიღოთ ინფორმაცია ყველა თანამშრომლის შესახებ, თუ მათ შორის არის ერთი მაინც, რომელიც სამედიცინო განყოფილებაში მუშაობს.
7.5.8. მივიღოთ ინფორმაცია ყველა თანამშრომლის შესახებ, თუ მათ შორის არის ერთი ვაკელი მაინც.
7.5.9. გამოვიტანოთ შესაბამისი შეტყობინება, თუ ერთი თანამშრომელი მაინც ცხოვრობს ვაკეში.

თავი 8. ოპერაციები სიმრავლეებზე

UNION ოპერაცია

8.1.1. გვაინტერესებს თანამშრომლებისა და შემკვეთების ერთმანეთისაგან განსხვავებული ადგილმდებარეობები (ქალაქი, რეგიონი).

8.1.2. გვაინტერესებს თანამშრომლებისა და შემკვეთების ადგილმდებარეობები (ქალაქი, რეგიონი).

INTERSECT ოპერაცია

8.2.1. გვაინტერესებს თანამშრომლებისა და შემკვეთების ერთმანეთისაგან განსხვავებული საერთო ადგილმდებარეობა (ქალაქი, რეგიონი).

8.2.2. გვაინტერესებს თანამშრომლებისა და შემკვეთების ერთმანეთისაგან განსხვავებული საერთო გვარები.

8.2.3. გვაინტერესებს თანამშრომლებისა და შემკვეთების ერთმანეთისაგან განსხვავებული საერთო ქალაქები.

8.2.4. გვაინტერესებს თანამშრომლებისა და შემკვეთების ერთმანეთისაგან განსხვავებული საერთო რაიონები.

8.2.5. გვაინტერესებს თანამშრომლებისა და შემკვეთების ერთმანეთისაგან განსხვავებული საერთო რეგიონები.

EXCEPT ოპერაცია

8.3.1. გვაინტერესებს თანამშრომლების სხვადასხვა ადგილმდებარეობა, რომელიც არ ემთხვევა შემკვეთების ადგილმდებარეობას (ქალაქი, რეგიონი).

8.3.2. გვაინტერესებს თანამშრომლების სხვადასხვა ადგილმდებარეობა, რომელიც არ ემთხვევა შემკვეთების ადგილმდებარეობას (ქალაქი).

8.3.3. გვაინტერესებს თანამშრომლების სხვადასხვა ადგილმდებარეობა, რომელიც არ ემთხვევა შემკვეთების ადგილმდებარეობას (რაიონი).

8.3.4. გვაინტერესებს თანამშრომლების სხვადასხვა ადგილმდებარეობა, რომელიც არ ემთხვევა შემკვეთების ადგილმდებარეობას (რეგიონი).

8.3.5. გვაინტერესებს შემკვეთების სხვადასხვა ადგილმდებარეობა, რომელიც არ ემთხვევა თანამშრომლების ადგილმდებარეობას (ქალაქი, რეგიონი).

8.3.6. გვაინტერესებს შემკვეთების სხვადასხვა ადგილმდებარეობა, რომელიც არ ემთხვევა თანამშრომლების შემკვეთების ადგილმდებარეობას (ქალაქი).

8.3.7. გვაინტერესებს შემკვეთების სხვადასხვა ადგილმდებარეობა, რომელიც არ ემთხვევა თანამშრომლების ადგილმდებარეობას (რაიონი).

8.3.8. გვაინტერესებს შემკვეთების სხვადასხვა ადგილმდებარეობა, რომელიც არ ემთხვევა თანამშრომლების ადგილმდებარეობას (რეგიონი).

8.3.9. გვაინტერესებს იმ შემკვეთების ადგილმდებარეობა, რომლებიც იურიდიულ პირებს წარმოადგენენ და რომელთა ადგილმდებარეობა არ ემთხვევა თანამშრომლებისა და იმ შემკვეთების საერთო ადგილმდებარეობას, რომლებიც ფიზიკურ პირებს წარმოადგენენ (ქალაქი, რაიონი).

8.3.10. გვაინტერესებს იმ შემკვეთების ადგილმდებარეობა, რომლებიც იურიდიულ პირებს წარმოადგენენ და რომელთა ადგილმდებარეობა ემთხვევა თანამშრომლებისა და იმ შემკვეთების საერთო ადგილმდებარეობას, რომლებიც ფიზიკურ პირებს წარმოადგენენ (ქალაქი, რაიონი).

8.3.11. გვაინტერესებს იმ შემკვეთების ადგილმდებარეობა, რომლებიც ფიზიკურ პირებს წარმოადგენენ და რომელთა ადგილმდებარეობა ემთხვევა თანამშრომლებისა და იმ შემკვეთების საერთო ადგილმდებარეობას, რომლებიც იურიდიულ პირებს წარმოადგენენ (ქალაქი, რაიონი).
8.3.12. გვაინტერესებს იმ შემკვეთების ადგილმდებარეობა, რომლებიც ფიზიკურ პირებს წარმოადგენენ და რომელთა ადგილმდებარეობა ემთხვევა სამედიცინო განყოფილებაში მომუშავე თანამშრომლებისა და იმ შემკვეთების საერთო ადგილმდებარეობას, რომლებიც იურიდიულ პირებს წარმოადგენენ (ქალაქი, რაიონი).
8.3.13. გვაინტერესებს იმ შემკვეთების ადგილმდებარეობა, რომლებიც ფიზიკურ პირებს წარმოადგენენ და რომელთა ადგილმდებარეობა არ ემთხვევა სამედიცინო განყოფილებაში მომუშავე თანამშრომლებისა და იმ შემკვეთების საერთო ადგილმდებარეობას, რომლებიც იურიდიულ პირებს წარმოადგენენ (ქალაქი, რაიონი).
8.3.14. გვაინტერესებს იმ შემკვეთების ადგილმდებარეობა, რომლებიც იურიდიულ პირებს წარმოადგენენ და რომელთა ადგილმდებარეობა არ ემთხვევა თანამშრომლებისა და იმ შემკვეთების საერთო ადგილმდებარეობას, რომლებიც ფიზიკურ პირებს წარმოადგენენ (ქალაქი, რეგიონი).
8.3.15. გვაინტერესებს იმ შემკვეთების ადგილმდებარეობა, რომლებიც იურიდიულ პირებს წარმოადგენენ და რომელთა ადგილმდებარეობა ემთხვევა თანამშრომლებისა და იმ შემკვეთების საერთო ადგილმდებარეობას, რომლებიც ფიზიკურ პირებს წარმოადგენენ (ქალაქი, რეგიონი).
8.3.16. გვაინტერესებს იმ შემკვეთების ადგილმდებარეობა, რომლებიც ფიზიკურ პირებს წარმოადგენენ და რომელთა ადგილმდებარეობა ემთხვევა თანამშრომლებისა და იმ შემკვეთების საერთო ადგილმდებარეობას, რომლებიც იურიდიულ პირებს წარმოადგენენ (ქალაქი, რეგიონი).
8.3.17. გვაინტერესებს იმ შემკვეთების ადგილმდებარეობა, რომლებიც ფიზიკურ პირებს წარმოადგენენ და რომელთა ადგილმდებარეობა ემთხვევა სამედიცინო განყოფილებაში მომუშავე თანამშრომლებისა და იმ შემკვეთების საერთო ადგილმდებარეობას, რომლებიც იურიდიულ პირებს წარმოადგენენ (ქალაქი, რეგიონი).
8.3.18. გვაინტერესებს იმ შემკვეთების ადგილმდებარეობა, რომლებიც ფიზიკურ პირებს წარმოადგენენ და რომელთა ადგილმდებარეობა არ ემთხვევა სამედიცინო განყოფილებაში მომუშავე თანამშრომლებისა და იმ შემკვეთების საერთო ადგილმდებარეობას, რომლებიც იურიდიულ პირებს წარმოადგენენ (ქალაქი, რეგიონი).
8.3.19. გვაინტერესებს იმ შემკვეთების ადგილმდებარეობა, რომლებიც იურიდიულ პირებს წარმოადგენენ და რომელთა ადგილმდებარეობა არ ემთხვევა თანამშრომლებისა და იმ შემკვეთების საერთო ადგილმდებარეობას, რომლებიც ფიზიკურ პირებს წარმოადგენენ (ქალაქი).
8.3.20. გვაინტერესებს იმ შემკვეთების ადგილმდებარეობა, რომლებიც იურიდიულ პირებს წარმოადგენენ და რომელთა ადგილმდებარეობა ემთხვევა თანამშრომლებისა და იმ შემკვეთების საერთო ადგილმდებარეობას, რომლებიც ფიზიკურ პირებს წარმოადგენენ (რაიონი).
8.3.21. გვაინტერესებს იმ შემკვეთების ადგილმდებარეობა, რომლებიც ფიზიკურ პირებს წარმოადგენენ და რომელთა ადგილმდებარეობა ემთხვევა თანამშრომლებისა და იმ შემკვეთების საერთო ადგილმდებარეობას, რომლებიც იურიდიულ პირებს წარმოადგენენ (რეგიონი).
8.3.22. გვაინტერესებს იმ შემკვეთების ადგილმდებარეობა, რომლებიც ფიზიკურ პირებს წარმოადგენენ და რომელთა ადგილმდებარეობა ემთხვევა 37 წელზე მეტი ასაკის მქონე

თანამშრომლებისა და იმ შემკვეთების საერთო ადგილმდებარეობას, რომლებიც იურიდიულ პირებს წარმოადგენენ (ქალაქი).
8.3.23. გვაინტერესებს იმ შემკვეთების ადგილმდებარეობა, რომლებიც ფიზიკურ პირებს წარმოადგენენ და რომელთა ადგილმდებარეობა არ ემთხვევა 7 წელზე მეტი სტაჟის მქონე თანამშრომლებისა და იმ შემკვეთების საერთო ადგილმდებარეობას, რომლებიც იურიდიულ პირებს წარმოადგენენ (ქალაქი).
8.3.24. გვაინტერესებს იმ შემკვეთების ადგილმდებარეობა, რომლებიც იურიდიულ პირებს წარმოადგენენ და რომელთა ადგილმდებარეობა არ ემთხვევა იმ შემკვეთების ადგილმდებარეობას, რომლებიც ფიზიკურ პირებს წარმოადგენენ, მაგრამ ემთხვევა თანამშრომლების ადგილმდებარეობას (ქალაქი, რაიონი).
8.3.25. გვაინტერესებს იმ შემკვეთების ადგილმდებარეობა, რომლებიც იურიდიულ პირებს წარმოადგენენ და რომელთა ადგილმდებარეობა არ ემთხვევა იმ შემკვეთების ადგილმდებარეობას, რომლებიც ფიზიკურ პირებს წარმოადგენენ, მაგრამ ემთხვევა თანამშრომლების ადგილმდებარეობას, რომლებიც სამედიცინო განყოფილებაში მუშაობენ (ქალაქი, რაიონი).
8.3.26. გვაინტერესებს იმ შემკვეთების ადგილმდებარეობა, რომლებიც იურიდიულ პირებს წარმოადგენენ და რომელთა ადგილმდებარეობა ემთხვევა იმ შემკვეთების ადგილმდებარეობას, რომლებიც ფიზიკურ პირებს წარმოადგენენ, მაგრამ ემთხვევა თანამშრომლების ადგილმდებარეობას (ქალაქი, რაიონი).
8.3.27. გვაინტერესებს იმ შემკვეთების ადგილმდებარეობა, რომლებიც იურიდიულ პირებს წარმოადგენენ და რომელთა ადგილმდებარეობა ემთხვევა იმ შემკვეთების ადგილმდებარეობას, რომლებიც ფიზიკურ პირებს წარმოადგენენ, მაგრამ ემთხვევა თანამშრომლების ადგილმდებარეობას, რომლებიც სამედიცინო განყოფილებაში მუშაობენ (ქალაქი, რაიონი).
8.3.28. გვაინტერესებს იმ შემკვეთების ადგილმდებარეობა, რომლებიც იურიდიულ პირებს წარმოადგენენ და რომელთა ადგილმდებარეობა ემთხვევა იმ შემკვეთების ადგილმდებარეობას, რომლებიც ფიზიკურ პირებს წარმოადგენენ, მაგრამ ემთხვევა თანამშრომლების ადგილმდებარეობას, რომელთა ასაკი 50 წელს აღემატება (ქალაქი).
8.3.29. გვაინტერესებს იმ შემკვეთების ადგილმდებარეობა, რომლებიც იურიდიულ პირებს წარმოადგენენ და რომელთა ადგილმდებარეობა ემთხვევა იმ შემკვეთების ადგილმდებარეობას, რომლებიც ფიზიკურ პირებს წარმოადგენენ, მაგრამ ემთხვევა თანამშრომლების ადგილმდებარეობას, რომლებიც სამედიცინო განყოფილებაში მუშაობენ (რაიონი).
8.3.30. გვაინტერესებს იმ შემკვეთების ადგილმდებარეობა, რომლებიც იურიდიულ პირებს წარმოადგენენ და რომელთა ადგილმდებარეობა ემთხვევა იმ შემკვეთების ადგილმდებარეობას, რომლებიც ფიზიკურ პირებს წარმოადგენენ, მაგრამ ემთხვევა თანამშრომლების ადგილმდებარეობას (რეგიონი).

თავი 9. ფუნქციები

Scalar ტიპის ფუნქციები

9.1.1. გამოთვალეთ მითითებული განყოფილების საშუალო ხელფასი
9.1.2. გამოთვალეთ მითითებული განყოფილების მინიმალური ხელფასი.
9.1.3. გამოთვალეთ მითითებული განყოფილების მინიმალური ასაკი.
9.1.4. გამოთვალეთ მითითებული განყოფილების საშუალო ასაკი.
9.1.5. გამოთვალეთ მითითებული განყოფილების საშუალო სტაჟი.
9.1.6. გამოთვალეთ მითითებული განყოფილების მაქსიმალური სტაჟი.

9.1.7. გამოთვალეთ მითითებულ ქალაქში მცხოვრები თანამშრომლების მაქსიმალური სტაჟი.
9.1.8. გამოთვალეთ მითითებული განყოფილების მინიმალური ხელფასი.
9.1.9. გამოთვალეთ მითითებულ ქალაქში მცხოვრები თანამშრომლების საშუალო სტაჟი.
9.1.10. შეადგინეთ Scalar ტიპის ფუნქცია, რომელიც გამოთვლის და გასცემს ხელშეკრულების მაქსიმალურ თანხას, რომელიც გაფორმდა 2002 წლის 1 იანვრიდან 2005 წლის 1 იანვრამდე. ფუნქციას პარამეტრად გადაეცემა ორივე თარიღი.
9.1.11. შეადგინეთ Scalar ტიპის ფუნქცია, რომელიც გასცემს მითითებულ ქალაქში მცხოვრები თანამშრომლების მინიმალურ ხელფასს. ფუნქციას პარამეტრად გადაეცემა ქალაქი.
9.1.12. შეადგინეთ Scalar ტიპის ფუნქცია, რომელიც გამოთვლის და გასცემს ხელშეკრულების მინიმალურ თანხას, რომელიც გაფორმდა მითითებული თარიღის შემდეგ და რომელთა თანხა ნაკლებია პარამეტრით მითითებულ თანხაზე. ფუნქციას პარამეტრად გადაეცემა თარიღი და თანხა.

Inline ტიპის ფუნქციები

9.2.1. გამოთვალეთ საშუალო ხელფასი განყოფილებების მიხედვით.
9.2.2. გამოთვალეთ მაქსიმალური ხელფასი განყოფილებების მიხედვით.
9.2.3. გამოთვალეთ მინიმალური ხელფასი განყოფილებების მიხედვით.
9.2.4. გამოთვალეთ მინიმალური ასაკი განყოფილებების მიხედვით.
9.2.5. გამოთვალეთ მაქსიმალური ასაკი განყოფილებების მიხედვით.
9.2.6. გამოთვალეთ საშუალო ასაკი განყოფილებების მიხედვით.
9.2.7. გამოთვალეთ საშუალო სტაჟი განყოფილებების მიხედვით.
9.2.8. გამოთვალეთ მაქსიმალური სტაჟი განყოფილებების მიხედვით.
9.2.9. გამოთვალეთ მინიმალური სტაჟი განყოფილებების მიხედვით.
9.2.10. შეადგინეთ Inline ტიპის ფუნქცია, რომელიც გასცემს ინფორმაციას მითითებულ ქალაქში მცხოვრები თანამშრომლების შესახებ. ფუნქციას პარამეტრად გადაეცემა ქალაქი.
9.2.11. შეადგინეთ Inline ტიპის ფუნქცია, რომელიც გასცემს ინფორმაციას იმ ხელშეკრულებების შესახებ, რომლებიც გაფორმდა მითითებულ წელს და რომელთა თანხა ნაკლებია პარამეტრით მითითებულ თანხაზე. ფუნქციას პარამეტრად გადაეცემა თანხა და თარიღი.

Multi-statement ტიპის ფუნქციები

9.3.1. შეადგინეთ Multi-statement ტიპის ფუნქცია, რომელსაც 2 პარამეტრი აქვს: წილადი 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 წილადი ტიპი, მესამე სვეტს კი სტრიქონული. დასაბრუნებელ ცხრილს დაუმატეთ 3 სტრიქონი. მეორე და მესამე სვეტებში უნდა ჩაიწეროს ფუნქციის პარამეტრები. ყოველი ჩაწერის წინ წილადი პარამეტრის მნიშვნელობა 25,5-ით გაზარდეთ.
9.3.2. 9.3.1. სავარჯიშოში შედგენილი ფუნქცია შეცვალეთ ისე, რომ მან გასცეს ინფორმაცია თანამშრომლების შესახებ, რომელთა ასაკი აღემატება პარამეტრით გადაცემულ მნიშვნელობას და რომელთა სტაჟი ნაკლებია პარამეტრით მითითებულ მნიშვნელობაზე. ფუნქციას პარამეტრებად გადაეცემა სტაჟი და ასაკი.

ჩადგმული ფუნქციები

მათემატიკის ფუნქციები

9.4.1. Personali ცხრილის xelfasi სვეტის მნიშვნელობა დაამრგვალეთ ქვევით, ზევით, დამრგვალებისას წერტილის შემდეგ ორი თანრიგი გამოჩნდეს, დამრგვალებისას წერტილამდე ორი თანრიგი გამოჩნდეს.

9.4.2. Xelshekruleba ცხრილის gadasaxdeli_1 სვეტის მნიშვნელობა დაამრგვალეთ ქვევით, ზევით, დამრგვალებისას წერტილის შემდეგ ორი თანრიგი გამოჩნდეს, დამრგვალებისას წერტილამდე ორი თანრიგი გამოჩნდეს.

9.4.3. Personali ცხრილის xelfasi სვეტის მნიშვნელობა აიყვანეთ კუბში. ამავე სვეტიდან ამოიღეთ კვადრატული ფესვი.

9.4.4. წარმოქმენით 15 შემთხვევითი რიცხვი.

თარიღებთან სამუშაო ფუნქციები

9.5.1. გვაინტერესებს თანამშრომლების ასაკი.

9.5.2. გვაინტერესებს თანამშრომლების მაქსიმალური ასაკი.

9.5.3. გვაინტერესებს თანამშრომლების მინიმალური ასაკი.

9.5.4. გვაინტერესებს თანამშრომლების საშუალო ასაკი.

9.5.5. გვაინტერესებს თანამშრომელი კაცების მაქსიმალური ასაკი.

9.5.6. გვაინტერესებს თანამშრომელი ქალების მაქსიმალური ასაკი.

9.5.7. გვაინტერესებს თანამშრომელი კაცების მინიმალური ასაკი.

9.5.8. გვაინტერესებს თანამშრომელი ქალების მინიმალური ასაკი.

9.5.9. გვაინტერესებს თანამშრომელი კაცების საშუალო ასაკი.

9.5.10. გვაინტერესებს თანამშრომელი ქალების საშუალო ასაკი.

სტრიქონებთან სამუშაო ფუნქციები

9.6.1. Personali ცხრილის saxeli და gvari სვეტებს შორის გამოტანისას მოათავსეთ ხუთი ინტერვალი.

9.6.2. Personali ცხრილის xelfasi სვეტის მნიშვნელობა ისე გამოიტანეთ, რომ მან დაიკავოს შვიდი თანრიგი, წილადმა ნაწილმა კი - ორი.

9.6.3. Personali ცხრილის gvari სვეტის მნიშვნელობა გამოტანისას მოათავსეთ კვადრატულ ფრჩხილებში, saxeli სვეტის მნიშვნელობა - ბრჭყალებში, ganyofileba სვეტის მნიშვნელობა - ფიგურულ ფრჩხილებში, qalaqi სვეტის მნიშვნელობა - მრგვალ ფრჩხილებში, raioni სვეტის მნიშვნელობა - აპოსტროფებში.

9.6.4. Personali ცხრილიდან saxeli სვეტის გამოტანისას პირველი სიმბოლო შეცვალეთ ‘გ’ ასოთი იმ თანამშრომლებისთვის, რომელთა გვარია - ‘გაჩეჩილაძე’.

9.6.5. Personali ცხრილიდან gvari სვეტის გამოტანისას სიმბოლოები - ‘ძე’ შეცვალეთ სიმბოლოებით - ‘შვილი’.

9.6.6. Personali ცხრილიდან qalaqi სვეტის გამოტანისას მისი მნიშვნელობა სამჯერ გაიმეორეთ ‘თბილისი’ მნიშვნელობისთვის.

9.6.7. Personali ცხრილის saxeli სვეტში იპოვეთ ‘გი’ სიმბოლოების პოზიცია დაწყებული პირველი პოზიციიდან და დაწყებული მეორე პოზიციიდან იმ თანამშრომლებისთვის, რომელთა სახელი ‘გ’ ასოთი იწყება.

9.6.8. გამოიტანეთ Personali ცხრილის saxeli სვეტის სიგრძე და 4310 კოდის მქონე სიმბოლო იმ თანამშრომლისთვის, რომლის სახელი და გვარია - ‘სამხარაძე ანა’.

9.6.9. გამოიტანეთ Personali ცხრილის gvari სვეტის მარცხენა ექვსი სიმბოლო და მარეჯვენა ორი სიმბოლო იმ თანამშრომლისთვის, რომლის სახელი და გვარია ‘სამხარაძე საბა’.

თავი 10. შენახული პროცედურები

10.1.1. შექმენით შენახული პროცედურა, რომელსაც გადავცემთ განყოფილების სახელს და მივიღებთ ინფორმაციას ამ განყოფილებაში მომუშავე თანამშრომლების შესახებ.
10.1.2. შექმენით შენახული პროცედურა, რომელსაც გადავცემთ ქალაქის სახელს და მივიღებთ ინფორმაციას ამ ქალაქში მომუშავე თანამშრომლების შესახებ.
10.1.3. შეადგინეთ შენახული პროცედურა, რომელიც გასცემს მითითებული განყოფილების თანამშრომლების სიას. ავტომატურ მნიშვნელობად მითითებულია 'სასპორტო'.
10.1.4. შეადგინეთ შენახული პროცედურა, რომელიც გასცემს მითითებული ქალაქის თანამშრომლების სიას. ავტომატურ მნიშვნელობად მითითებულია 'ბათუმი'.
10.1.5. შეადგინეთ შენახული პროცედურა, რომელიც გამოთვლის და დაგვიბრუნებს მითითებული განყოფილების მაქსიმალურ ხელფასს.
10.1.6. შეადგინეთ შენახული პროცედურა, რომელიც გამოთვლის და დაგვიბრუნებს მითითებული განყოფილების მინიმალურ ასაკს.
10.1.7. შეადგინეთ შენახული პროცედურა, რომელიც გასცემს ინფორმაციას მითითებულ ქალაქში მომუშავე და 40 წელს გადაცილებული თანამშრომლების შესახებ.
10.1.8. შეადგინეთ შენახული პროცედურა, რომელიც გასცემს ინფორმაციას მითითებულ განყოფილებაში მომუშავე და 15 წელზე მეტი სტაჟის მქონე თანამშრომლების შესახებ
10.1.9. შენახული პროცედურის პარამეტრი შეიცავს შაბლონს, რომლის მიხედვით მოხდება 's' ასოთი დაწყებული გვარების ძებნა.
10.1.10. შეადგინეთ შენახული პროცედურა, რომელიც გასცემს ინფორმაციას მითითებულ ქალაქში მცხოვრები შემკვეთების შესახებ. პროცედურას პარამეტრად გადაეცემა ქალაქი.
10.1.11. შეადგინეთ შენახული პროცედურა, რომელიც გასცემს ინფორმაციას მითითებულ ქალაქში მცხოვრები თანამშრომლების შესახებ, რომელთა ასაკი ნაკლებია პარამეტრით მითითებულ მნიშვნელობაზე. პროცედურას პარამეტრად გადაეცემა ქალაქი და ასაკი.
10.1.12. შეადგინეთ შენახული პროცედურა, რომელიც გასცემს ინფორმაციას იმ ხელშეკრულებების შესახებ, რომლებიც გაფორმდა მითითებული წლის შემდეგ და რომელთა თანხა ნაკლებია პარამეტრით მითითებულ მნიშვნელობაზე. პროცედურას პარამეტრად გადაეცემა თანხა და თარიღი.
10.1.13. შეადგინეთ შენახული პროცედურა, რომელიც გასცემს ინფორმაციას მითითებული განყოფილების თანამშრომლების შესახებ, რომელთა ხელფასი არ აღემატება პარამეტრით მითითებულ თანხას. პროცედურას პარამეტრებად გადაეცემა განყოფილება და ხელფასი.

თავი 11. ინდექსები

11.1.1. Shemkveti ცხრილის gvari სვეტისთვის შექმენით არა კლასტერული ინდექსი. სვეტის მონაცემები უნდა დალაგდეს ზრდადობით.
11.1.2. Personali ცხრილის gvari სვეტისთვის შევქმნათ უნიკალური არაკლასტერული ინდექსი. სვეტის მონაცემები უნდა დალაგდეს კლებადობით.
11.1.3. Personali ცხრილის email სვეტისთვის შევქმნათ უნიკალური ინდექსი.
11.1.4. Personali ცხრილის mobiluri სვეტისთვის შევქმნათ უნიკალური არაკლასტერული ინდექსი.
11.1.5. Personali ცხრილის ganyofileba სვეტისთვის შევქმნათ არაკლასტერული ინდექსი. გამოიყენეთ DROP_EXISTING არგუმენტი.
11.1.6. შევასრულოთ Personali ცხრილის Index_mobiluri ინდექსის გადაწყობა.

11.1.7. შევასრულოთ Personali ცხრილის ყველა ინდექსის გადაწყობა
11.1.8. Personali ცხრილის firmis_dasaxeleba სვეტისთვის შევქმნათ უნიკალური არაკლასტერული ინდექსი
11.1.9. Shemkveti ცხრილის warmomadgeneli სვეტისთვის შევქმნათ არაკლასტერული ინდექსი.
11.1.10. Shemkveti ცხრილის firmis_dasaxeleba სვეტისთვის შექმენით ინდექსი. სვეტის მონაცემები უნდა დალაგდეს ზრდადობით. შეასრულეთ ინდექსის გადაწყობა.
11.1.11. Personali ცხრილის gvari და asaki სვეტებისთვის შექმენით არაკლასტერული ინდექსი. gvari სვეტის მონაცემები უნდა დალაგდეს კლებადობით, asaki სვეტის კი - ზრდადობით. შეასრულეთ ინდექსის გადაწყობა.
11.1.12. შეცვალეთ 11.1.10 სავარჯიშოში შექმნილი ინდექსი ისე, რომ დუბლირებული მნიშვნელობების შექმნას, სერვერმა გასცეს შეტყობინება შეცდომის შესახებ და გააუქმოს მხოლოდ იმ სტრიქონების ცვლილება, რომლებმაც დუბლირება გამოიწვიეს.

თავი 12. წარმოდგენები

წარმოდგენის შექმნა

12.1.1. შექმენით წარმოდგენა, რომელშიც გამოჩნდება Xelshekruleba ცხრილის ყველა სვეტი. წარმოდგენაში უნდა გამოჩნდეს ინფორმაცია მევალეების შესახებ.
12.1.2. შექმენით წარმოდგენა, რომელშიც გამოჩნდება Personali ცხრილის gvari სვეტი და Shemkveti ცხრილის gvari სვეტი. Personali ცხრილი არის მთავარი, Shemkveti ცხრილი - კი დამოკიდებული. გამოვიტანოთ ინფორმაცია იმ თანამშრომლების შესახებ, რომელთა ასაკი 40-ზე მეტია.
12.1.3. შექმენით წარმოდგენა, რომელშიც გამოჩნდება Personali ცხრილის gvari სვეტი და Shemkveti ცხრილის gvari სვეტი. Personali ცხრილი არის მთავარი, Shemkveti ცხრილი - კი დამოკიდებული. გამოიტანეთ ინფორმაცია იმ თანამშრომლების შესახებ, რომელთა სტაჟი 10 წელზე მეტია.
12.1.4. გამოიტანეთ ინფორმაცია იმ თანამშრომლების შესახებ, რომელთა სტაჟი 10 წელზე ნაკლებია. გამოვიყენთ ENCRYPTION არგუმენტი.
12.1.5. გვაინტერესებს მაქსიმალური ვალის მნიშვნელობა თითოეული თანამშრომლისთვის. წარმოდგენის შიგნით გამოიყენეთ ფუნქცია.
12.1.6. გამოიტანეთ თანამშრომლის საშუალო ასაკის მნიშვნელობა წარმოდგენის საშუალებით.
12.1.7. გამოიტანეთ ინფარმაცია იმ თანამშრომლების შესახებ, რომელთა ასაკია 25-დან 45 წლამდე.
12.1.8. გამოიტანეთ თანამშრომლის საშუალო სტაჟის მნიშვნელობა წარმოდგენის საშუალებით.
12.1.9. შექმენით წარმოდგენა, რომელშიც გამოჩნდება Personali და Shemkveti ცხრილების ყველა სვეტი. Personali ცხრილი არის მთავარი, Shemkveti ცხრილი - კი დამოკიდებული. გამოიტანეთ ინფორმაცია იმ თანამშრომლების შესახებ, რომელთა სტაჟი 20-ზე მეტია. თანამშრომლების გვარები ზრდადობით დაალაგეთ.
12.1.10. შექმენით წარმოდგენა, რომელშიც გამოჩნდება Personali ცხრილის gvari და asaki სვეტები, ხოლო Xelshekruleba ცხრილიდან კი - gadasaxdeli_l და tarigi_dawyebis სვეტები. Personali ცხრილი არის მთავარი, Xelshekruleba ცხრილი - კი დამოკიდებული. გამოიტანეთ ინფორმაცია იმ ხელშეკრულებების შესახებ, რომლებიც 2002-2005 წლებში გაფორმდა. თანამშრომლების გვარები დაალაგეთ ანბანურად ზრდადობით, თარიღები კი - კლებადობით.
12.1.11 შეცვალეთ 12.1.9 სავარჯიშოში შედგენილი წარმოდგენა ისე, რომ გამოჩნდეს ინფორმაცია იმ თანამშრომლების შესახებ, რომელთა ასაკი 40-ზე ნაკლებია. თანამშრომლების გვარები კლებადობით დაალაგეთ.

თავი 13. კურსორები

13.1. შექმენით სტატიკური კურსორი.
13.2. შექმენით გადახვევადი კურსორი.
13.3. შექმენით მხოლოდ წაკითხვადი კურსორი.
13.4. შექმენით კურსორი, რომელშიც შესაძლებელი იქნება მონაცემების ცვლილება.
13.5. შექმენით ლოკალური კურსორი, რომელშიც შესაძლებელი იქნება მონაცემების ცვლილება.
13.6. შექმენით გლობალური კურსორი, რომელშიც შესაძლებელი იქნება მონაცემების ცვლილება.
13.7. შექმენით მიმდევრობითი კურსორი.
13.8. შექმენით საგასაღებო კურსორი.
13.9. შექმენით დინამიკური კურსორი.
13.10. შეასრულეთ შექმნილი კურსორის ოპტიმიზება.
13.11. კურსორიდან წაიკითხეთ მე-5 სტრიქონი.
13.12. კურსორიდან წაიკითხეთ პირველი სტრიქონი.
13.13. კურსორიდან წაიკითხეთ უკანასკნელი სტრიქონი.
13.14. კურსორიდან წაიკითხეთ მომდევნო სტრიქონი.
13.15. კურსორიდან წაიკითხეთ წინა სტრიქონი.
13.16. კურსორიდან წაიკითხეთ გადასახდელი თანხა, ვალი და ხელშეკრულების გაფორმების თარიღი.
13.17. გაზარდეთ კურსორის გამოყენებით Personali ცხრილში სავაჭრო განყოფილების თანამშრომლების ხელფასი 200-ით.
13.18. წაშალეთ კურსორის გამოყენებით Personali ცხრილიდან მონაცემები ზუგდიდელი თანამშრომლების შესახებ.
13.19. გადანომრეთ კურსორის გამოყენებით Xelshekruleba ცხრილში xelshekrulebaID სვეტის მნიშვნელობები დაწყებული ერთიდან.

თავი 14. ტრანზაქციები და დაბლოკვები

14.1. ტრანზაქციის გამოყენებით Personali ცხრილში გააორმაგეთ ხელფასები.
14.2. გააუქმეთ Personali ცხრილში შეტანილი ცვლილებები.
14.3. შექმენით პროცედურა, რომელიც ცხრილში სტრიქონებს შეცვლის. ეს პროცედურა გამოვიძახოთ ტრანზაქციის შიგნით და გავაუქმოთ მისი მუშაობის შედეგები. შემდეგ ისევ გამოვიძახოთ იგივე პროცედურა და მას სხვა პარამეტრები გადავცეთ.
14.4. დაუმატეთ ტრანზაქციის გამოყენებით #Personali და #Personali დროებით ცხრილებს თითო სტრიქონი.
14.5. შეცვალეთ ტრანზაქციის გამოყენებით #Personali და #Personali დროებით ცხრილებში თითო სტრიქონი.
14.6. წაშალეთ ტრანზაქციის გამოყენებით #Personali და #Personali დროებითი ცხრილებიდან თითო სტრიქონი.

თავი 15. ტრიგერები

15.1. შექმენით Personali_1 ცხრილი. მისთვის შექმენით UPDATE ტრიგერი, რომელიც აკრძალავს სტრიქონების შეცვლას და გამოიტანს ინფორმაციას სტრიქონების შეცვლის მცდელობის შესახებ, აგრეთვე შეცვლილი სტრიქონების რაოდენობას.
15.2. შექმენით ტრიგერი, რომელიც ახალ ხელშეკრულებას არ გაგვაფორმებინებს შემკვეთთან, რომლის სახელი და გვარია - 'სამხარაძე რომანი'.
15.3. შექმენით ტრიგერი, რომელიც აკრძალავს Shemkveti ცხრილიდან სტრიქონების წაშლას თბილისელი შემკვეთების შესახებ.
15.4. შექმენით ტრიგერი, რომელიც xelfasi სვეტის შეცვლის ნებას აძლევს მხოლოდ dbo მომხმარებელს.
15.5. შექმენით Personali_2 ცხრილი. მისთვის შექმენით UPDATE ტრიგერი, რომელიც გამოიტანს შეცვლილი სტრიქონების რაოდენობას.

თავი 16. ცხრილური გამოსახულებები

წარმოებული ცხრილები

16.1.1. წლების მიხედვით გვაინტერესებს ხელშეკრულების რაოდენობა.
16.1.2. გვაინტერესებს ხელშეკრულების რაოდენობა, წლების მიხედვით, რომლებიც იმ თანამშრომელთან გაფორმდა, რომლის იდენტიფიკატორი @personaliID ცვლადში ინახება.
16.1.3. გვაინტერესებს წლები, რომლებშიც ხელშეკრულებები გაფორმდა და შემსრულებლების რაოდენობა, რომლებმაც თითოეული წლის განმავლობაში გააფორმეს ხელშეკრულება. ამასთან გვაინტერესებს მხოლოდ ის წლები, რომლებშიც ერთზე მეტმა შემსრულებელმა გააფორმა ხელშეკრულება.
16.1.4. გვაინტერესებს ის წლები, რომლებშიც ერთზე მეტი ხელშეკრულება გაფორმდა.

საერთო ცხრილური გამოსახულებები

16.2.1. განსაზღვრეთ საერთო ცხრილური გამოსახულება, რომლის სახელია TbilisiPersonali (შემსრულებლები თბილისიდან). ის ეფუძნება მოთხოვნას, რომელიც გასცემს თბილისელ შემსრულებლებს.
16.2.2. გვაინტერესებს ხელშეკრულების რაოდენობა, წლების მიხედვით, რომლებიც იმ თანამშრომელთან გაფორმდა, რომლის იდენტიფიკატორი @personaliID ცვლადში ინახება.
16.2.3. გვაინტერესებს წლები, რომლებშიც ხელშეკრულებები გაფორმდა და შემსრულებლების რაოდენობა, რომლებმაც თითოეული წლის განმავლობაში გააფორმეს ხელშეკრულება. ამასთან, გვაინტერესებს მხოლოდ ის წლები, რომლებშიც ერთზე მეტმა შემსრულებელმა გააფორმა ხელშეკრულება.
16.2.4. გვაინტერესებს ის წლები, რომლებშიც ერთზე მეტი ხელშეკრულება გაფორმდა.

APPLY ოპერაცია

16.3.1. გვაინტერესებს თითოეული შემსრულებლის სამი უკანასკნელი შეკვეთა.
16.3.2. გვაინტერესებს თითოეული შემსრულებლის სამი პირველი შეკვეთა.
16.3.3. გვაინტერესებს თითოეული შემსრულებლის სამი უკანასკნელი შეკვეთა და ის შემსრულებლები, რომლებმაც არც ერთი შეკვეთა არ გააფორმეს.
16.3.4. გვაინტერესებს თითოეული შემსრულებლის სამი პირველი შეკვეთა და ის შემსრულებლები, რომლებმაც არც ერთი შეკვეთა არ გააფორმეს.

თავი 18. მონაცემთა ბაზების სარეზერვო ასლები

18.1. მონაცემთა ბაზის მოსათავსებლად შექმენით დისკური მოწყობილობა.

18.2. მონაცემთა ბაზის მოსათავსებლად შექმენით ლენტური მოწყობილობა.

18.3. მონაცემთა ბაზის მოსათავსებლად შექმენით ქსელური მოწყობილობა.

18.4. წაშალეთ 18.2. სავარჯიშოში შექმნილი ლენტური მოწყობილობა. უნდა წაიშალოს ფიზიკური ფაილიც.

18.5. შექმენით მონაცემთა ბაზის სრული სარეზერვო ასლი, რომელიც ინფორმაციის მატარებელზე მოთავსთდება როგორც პირველი ფაილი.

18.6. შექმენით მონაცემთა ბაზის სრული სარეზერვო ასლი, რომელიც დაემატება ინფორმაციის მატარებელზე არსებულ ფაილებს.

18.7. შექმენით მონაცემთა ბაზის სრული სარეზერვო ასლი, რომელიც ზემოდან გადაეწერება არსებულ ასლებს.

18.8. შექმენით მონაცემთა ბაზის სრული სარეზერვო ასლი. შეამოწმეთ არქივების სახელები და აქტუალურობა, რათა თავიდან ავიცილოთ შემთხვევით ზემოდან გადაწერა.

18.9. შექმენით მონაცემთა ბაზის სრული სარეზერვო ასლი. განსაზღვრეთ არქივის აქტუალურობა დღეების მიხედვით.

18.10. შექმენით მონაცემთა ბაზის სრული სარეზერვო ასლი. არქივზე გადაწერა არ შეასრულოთ მითითებული თარიღის გასვლამდე.

18.11. შექმენით მონაცემთა ბაზის დიფერენცირებული სარეზერვო ასლი.

18.12. შეასრულეთ მონაცემთა ბაზის აღდგენა სრული სარეზერვო ასლიდან. მონაცემთა ბაზა არ არსებობს.

18.13. შეასრულეთ მონაცემთა ბაზის აღდგენა სრული სარეზერვო ასლიდან. მონაცემთა ბაზა უკვე არსებობს და ხდება მასზე გადაწერა ასლიდან.

18.14. შეასრულეთ მონაცემთა ბაზის აღდგენა ჯერ სრული სარეზერვო, შემდეგ კი დიფერენცირებული ასლიდან.

დანართი 2. სავარჯიშოების ამოხსნები

თავი 2. მონაცემთა ბაზები

2.1.

```
USE Master;
 -- თუ Baza_1 ბაზა არსებობს, მაშინ ის წაიშლება
IF DB_ID (N'Baza_1') IS NOT NULL
DROP DATABASE Baza_1;
-- Baza_1 მონაცემთა ბაზის შექმნა
CREATE DATABASE Baza_1
ON
(
NAME = Baza_1,
FILENAME =
'C:\Program Files\Microsoft SQL Server\MSSQL10.MSSQLSERVER\MSSQL\DATA\Baza_1.mdf',
SIZE = 10MB,
MAXSIZE = 50MB,
FILEGROWTH = 5MB
)
LOG ON
(
NAME = Baza_1_log,
FILENAME =
'C:\Program Files\Microsoft SQL Server\MSSQL10.MSSQLSERVER\MSSQL\DATA\Baza_1_Log.ldf',
SIZE = 5MB,
MAXSIZE = 20MB,
FILEGROWTH = 3MB
);
```

2.2.

```
USE Master;
 -- თუ Baza_2 ბაზა არსებობს, მაშინ ის წაიშლება
IF DB_ID (N'Baza_2') IS NOT NULL
DROP DATABASE Baza_2;

CREATE DATABASE Baza_2
ON
(
NAME = Baza_2,
FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL10.MSSQLSERVER\MSSQL\DATA\Baza_2.mdf',
MAXSIZE = UNLIMITED
)
LOG ON
(
NAME = Baza_2_log,
```

```
FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL10.MSSQLSERVER\MSSQL\DATA\Baza_2_Log.ldf',
MAXSIZE = UNLIMITED
);
```
2.3.
```
USE Master;
 -- თუ Baza_3 ბაზა არსებობს, მაშინ ის წაიშლება
IF DB_ID (N'Baza_3') IS NOT NULL
DROP DATABASE Baza_3;

CREATE DATABASE Baza_3
ON
(
NAME = Baza_3,
FILENAME =
'C:\Program Files\Microsoft SQL Server\MSSQL10.MSSQLSERVER\MSSQL\DATA\Baza_3.mdf',
SIZE = 15MB
)
LOG ON
(
NAME = Baza_3_log,
FILENAME =
'C:\Program Files\Microsoft SQL Server\MSSQL10.MSSQLSERVER\MSSQL\DATA\Baza_3_Log.ldf',
SIZE = 7MB
);
```
2.4.
```
USE Master;
 -- თუ Baza_4 ბაზა არსებობს, მაშინ ის წაიშლება
IF DB_ID (N'Baza_4') IS NOT NULL
DROP DATABASE Baza_4;

CREATE DATABASE Baza_4
ON
(
NAME = Baza_4,
FILENAME =
'C:\Program Files\Microsoft SQL Server\MSSQL10.MSSQLSERVER\MSSQL\DATA\Baza_4.mdf',
FILEGROWTH = 8MB
)
LOG ON
(
NAME = Baza_4_log,
FILENAME =
'C:\Program Files\Microsoft SQL Server\MSSQL10.MSSQLSERVER\MSSQL\DATA\Baza_4_Log.ldf',
FILEGROWTH = 4MB
);
```

2.5.

```
USE master;
-- თუ Baza_5 ბაზა არსებობს, მაშინ ის წაიშლება
IF DB_ID (N'Baza_5') IS NOT NULL
DROP DATABASE Baza_5;
-- იმ კატალოგისკენ გზის განსაზღვრა, რომელშიც იქმნება Baza_5 მონაცემთა ბაზა
DECLARE @data_path NVARCHAR(256);
SET @data_path =
(
SELECT SUBSTRING(physical_name, 1, CHARINDEX(N'master.mdf', LOWER(physical_name)) - 1)
FROM master.sys.master_files
WHERE database_id = 1 AND file_id = 1
);
-- Baza_5 ბაზის შექმნა
EXEC ('CREATE DATABASE Baza_5
ON
PRIMARY
(
NAME = Baza_51,
FILENAME = '''+ @data_path + 'Baza_51.mdf'',
SIZE = 50MB,
MAXSIZE = 100,
FILEGROWTH = 10
),
(
NAME = Baza_52,
FILENAME = '''+ @data_path + 'Baza_52.ndf'',
SIZE = 50MB,
MAXSIZE = 100,
FILEGROWTH = 10
)
LOG ON
(
NAME = Baza_5_log1,
FILENAME = '''+ @data_path + 'Baza_5_log1.ldf'',
SIZE = 50MB,
MAXSIZE = 100,
FILEGROWTH = 10
),
(
NAME = Baza_5_log2,
FILENAME = '''+ @data_path + 'Baza_5_log2.ldf'',
SIZE = 50MB,
MAXSIZE = 100,
FILEGROWTH = 10
)'
```

```
);
2.6.
USE master;
-- თუ Baza_6 ბაზა არსებობს, მაშინ ის წაიშლება
IF DB_ID (N'Baza_6') IS NOT NULL
DROP DATABASE Baza_6;
-- იმ კატალოგისკენ გზის განსაზღვრა, რომელშიც იქმნება Baza_6 მონაცემთა ბაზა
DECLARE @data_path NVARCHAR(256);
SET @data_path =
(
SELECT SUBSTRING(physical_name, 1, CHARINDEX(N'master.mdf', LOWER(physical_name)) - 1)
FROM master.sys.master_files
WHERE database_id = 1 AND file_id = 1
);
-- Baza_6 ბაზის შექმნა
EXEC ('CREATE DATABASE Baza_6
ON PRIMARY
(
NAME = File61,
FILENAME = '''+ @data_path + 'File61.mdf'',
SIZE = 20MB,
MAXSIZE = 50MB,
FILEGROWTH = 15%
),
(
NAME = File62,
FILENAME = '''+ @data_path + 'File62.ndf'',
SIZE = 20MB,
MAXSIZE = 50MB,
FILEGROWTH = 15%
),
FILEGROUP Group1
(
NAME = File63,
FILENAME = '''+ @data_path + 'File63.ndf'',
SIZE = 20MB,
MAXSIZE = 50MB,
FILEGROWTH = 5MB
),
(
NAME = File64,
FILENAME = '''+ @data_path + 'File64.ndf'',
SIZE = 20MB,
MAXSIZE = 50MB,
FILEGROWTH = 5MB
)
```

```
LOG ON
(
NAME = Baza6_log,
FILENAME = '''+ @data_path + 'Baza6_log.ldf'',
SIZE = 10MB,
MAXSIZE = 30MB,
FILEGROWTH = 5MB
)'
);
```

2.7.

```
CREATE DATABASE Baza_7;
EXEC sp_renamedb @dbname = N'Baza_7', @newname = N'Baza_71' ;
```

2.8.

```
CREATE DATABASE Baza_8;
DROP DATABASE Baza_8;
```

2.9.

```
sp_detach_db 'Baza_2';
```

2.10.

```
sp_attach_db @dbname = 'Baza_2',
@filename1 = 'C:\Program Files\Microsoft SQL
                Server\MSSQL10.MSSQLSERVER\MSSQL\DATA\Baza_2.mdf',
@filename2 = 'C:\Program Files\Microsoft SQL
                Server\MSSQL10.MSSQLSERVER\MSSQL\DATA\Baza_2_log.ldf';
```

2.11.

```
sp_helpdb Baza_3;
```

2.12.

```
USE Baza_4 ;
GO
sp_addtype tanamshromlis_asaki, INT, 'NULL' ;
GO
```

2.13.

```
USE Baza_4;
GO
sp_addtype tanamshromlis_xelfasi, FLOAT, 'NOT NULL' ;
```

2.14.

```
USE Baza_4 ;
GO
sp_rename N'tanamshromlis_xelfasi', N'tanamshromlis_anazgaureba', N'USERDATATYPE';
```

თავი 3. ცხრილები

3.1.

```
CREATE DATABASE Avto;
GO
USE Avto;
--      თუ ცხრილი არსებობს, მაშინ ის წაიშლება
```

```
IF OBJECT_ID(N'Manqana_1',N'U') IS NOT NULL
   DROP TABLE Manqana_1;
--       ცხრილის შექმნა
CREATE TABLE Manqana_1
(
Manqana_1ID          INT PRIMARY KEY IDENTITY (1,1),
modeli               NVARCHAR(20),
firma                NVARCHAR(20),
qveyana              NVARCHAR(20),
tarigi               DATETIME,
feri                 NVARCHAR(20),
kari_raodenoba       INT,
dzravis_tipi         NVARCHAR(20),
fasi                 FLOAT
);
```

3.2.

```
USE Avto;
--       თუ ცხრილი არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID(N'Manqana_2',N'U') IS NOT NULL
   DROP TABLE Manqana_2;
--       ცხრილის შექმნა
CREATE TABLE Manqana_2
(
Manqana_2ID     INT CONSTRAINT FK_C1 PRIMARY KEY (Manqana_2ID DESC) IDENTITY (1,1),
marka           NVARCHAR(20),
firma           NVARCHAR(20),
qveyana         NVARCHAR(20),
tarigi          DATETIME,
feri            NVARCHAR(20),
kari_raodenoba  INT,
dzravis_tipi    NVARCHAR(20),
fasi            FLOAT
);
```

3.3.

```
CREATE DATABASE Aptiaqi;
GO
USE Aptiaqi;
--       თუ ცხრილი არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID(N'Camlebi',N'U') IS NOT NULL
   DROP TABLE Camlebi;
--       ცხრილის შექმნა
CREATE TABLE Camlebi
(
camlebiID            INT PRIMARY KEY IDENTITY (1,1),
dasaxeleba           NVARCHAR(20),
firma                NVARCHAR(20),
```

```
qveyana                 NVARCHAR(20),
tarigi_gamoshvebis      DATETIME,
vada                    DATETIME,
raodenoba               INT,
danishnuleba            NVARCHAR(20),
fasi                    FLOAT
);
--      თუ ცხრილი არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID(N'Gayidvebi', N'U') IS NOT NULL
   DROP TABLE Gayidvebi;
--      ცხრილის შექმნა
CREATE TABLE Gayidvebi
(
gayidvebiID     INT PRIMARY KEY IDENTITY (1,1),
camlebiID       INT NULL REFERENCES Camlebi(camlebiID),
raodenoba       INT,
tanxa           FLOAT,
tarigi_gayidvis DATETIME
);
```

3.4.

```
USE Aptiaqi;
--      თუ ცხრილი არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID(N'Camlebi_1',N'U') IS NOT NULL
   DROP TABLE Camlebi_1;
--      ცხრილის შექმნა
CREATE TABLE Camlebi_1
(
camlebi_1ID             INT,
sveti1                  INT,
dasaxeleba              NVARCHAR(20),
firma                   NVARCHAR(20),
qveyana                 NVARCHAR(20),
tarigi_gamoshvebis      DATETIME,
vada                    DATETIME,
raodenoba               INT,
danishnuleba            NVARCHAR(20),
fasi                    FLOAT,
CONSTRAINT PK_C1 PRIMARY KEY (camlebi_1ID, sveti1)
);
--      თუ ცხრილი არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID(N'Gayidvebi_1', N'U') IS NOT NULL
   DROP TABLE Gayidvebi_1;
--      ცხრილის შექმნა
CREATE TABLE Gayidvebi_1
(
gayidvebi_1ID  INT PRIMARY KEY IDENTITY (1,1),
```

```
camlebi_1ID             INT NULL,
sveti1                  INT,
raodenoba               INT,
tanxa                   FLOAT,
tarigi_gayidvis         DATETIME,
CONSTRAINT FK_C2  FOREIGN KEY (camlebi_1ID, sveti1)
                        REFERENCES Camlebi_1(camlebi_1ID, sveti1)
                        ON DELETE CASCADE
);
```

3.5.

```
CREATE DATABASE Kadrebi;
GO
USE Kadrebi;
--      თუ ცხრილი არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID(N'Tanamshromlebi_1',N'U') IS NOT NULL
   DROP TABLE Tanamshromlebi_1;
--      ცხრილის შექმნა
CREATE TABLE Tanamshromlebi_1
(
tanamshromlebi_1ID   INT PRIMARY KEY IDENTITY (1,1),
gvari                NVARCHAR(20) DEFAULT N'სამხარაძე',
saxeli               NVARCHAR(20) DEFAULT N'რომანი',
tarigi_dabadebis     DATETIME DEFAULT GETDATE(),
staji                INT  DEFAULT 5,
ganyofileba          NVARCHAR(20) DEFAULT N'სამედიცინო',
xelfasi              FLOAT  DEFAULT 500.00,
qalaqi               NVARCHAR(20) DEFAULT N'თბილისი',
misamarti            NVARCHAR(20),
mobiluri             NVARCHAR(12)
);
```

3.6.

```
USE Kadrebi;
--      თუ ცხრილი არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID(N'Tanamshromlebi_2',N'U') IS NOT NULL
   DROP TABLE Tanamshromlebi_2;
--      ცხრილის შექმნა
CREATE TABLE Tanamshromlebi_2
(
tanamshromlebi_2ID   INT PRIMARY KEY IDENTITY (1,1),
gvari                NVARCHAR(20),
saxeli               NVARCHAR(20),
tarigi_dabadebis     DATETIME CHECK
                     ( tarigi_dabadebis >= '01.01.1990' AND tarigi_dabadebis <= '01.01.1955' ),
staji                INT CHECK ( staji > 5 AND staji < 30 ),
ganyofileba          NVARCHAR(20),
xelfasi              FLOAT,
```

```
qalaqi                  NVARCHAR(20),
misamarti               NVARCHAR(20),
mobiluri                NVARCHAR(12) CHECK (mobiluri LIKE '5[0-9][0-9]')
);
```

3.7.

```
CREATE DATABASE Bina;
GO
USE Bina;
--      თუ ცხრილი არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID(N'Binebi_1',N'U') IS NOT NULL
   DROP TABLE Binebi_1;
--      ცხრილის შექმნა
CREATE TABLE Binebi_1
(
binebi_1ID              INT PRIMARY KEY IDENTITY (1,1),
qalaqi                  NVARCHAR(20),
raioni                  NVARCHAR(20),
quchis_saxeli           NVARCHAR(20),
quchis_nomeri           NVARCHAR(7),
teleponi_qalaqis        NVARCHAR(9),
kursi                   FLOAT,
fasi_dolari             FLOAT,
fasi_lari               AS fasi_dolari * kursi,
binis_proeqti           NVARCHAR(12),
mflobelis_gvari         NVARCHAR(15),
mflobelis_saxeli        NVARCHAR(15),
tarigi_ashenebis        AS GETDATE()
);
```

3.8.

```
USE Bina;
--      თუ ცხრილი არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID(N'Binebi_2',N'U') IS NOT NULL
   DROP TABLE Binebi_2;
--      ცხრილის შექმნა
CREATE TABLE Binebi_2
(
binebi_2ID              INT PRIMARY KEY IDENTITY (1,1),
qalaqi                  NVARCHAR(20),
raioni                  NVARCHAR(20),
quchis_saxeli           NVARCHAR(20),
quchis_nomeri           NVARCHAR(7),
teleponi_qalaqis        NVARCHAR(9),
kursi                   FLOAT,
fasi_dolari             FLOAT,
fasi_lari               FLOAT,
binis_proeqti           NVARCHAR(12),
```

```
mflobelis_gvari         NVARCHAR(15) NOT NULL,
mflobelis_saxeli        NVARCHAR(15),
tarigi_ashenebis        DATETIME
);
```

3.9.

```
USE Kadrebi;
--      თუ ცხრილი არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID(N'Tanamshromlebi_3',N'U') IS NOT NULL
   DROP TABLE Tanamshromlebi_3;
--      ცხრილის შექმნა
CREATE TABLE Tanamshromlebi_3
(
tanamshromlebi_3ID  INT PRIMARY KEY IDENTITY (1,1),
gvari               NVARCHAR(20),
saxeli              NVARCHAR(20),
tarigi_dabadebis    DATETIME,
staji               INT  DEFAULT 5,
ganyofileba         NVARCHAR(20),
xelfasi             FLOAT,
qalaqi              NVARCHAR(20),
misamarti           NVARCHAR(20),
teleponi_saxlis     NVARCHAR(9) UNIQUE,
mobiluri            NVARCHAR(12) UNIQUE
);
```

3.10.

```
CREATE DATABASE Magazia_1;
GO
USE Magazia_1;
--      თუ ცხრილი არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID(N'Produqtebi',N'U') IS NOT NULL
   DROP TABLE Produqtebi;
--      ცხრილის შექმნა
CREATE TABLE Produqtebi
(
produqtebiID  INT PRIMARY KEY IDENTITY (1,1),
dasaxeleba    NVARCHAR(20),
raodenoba     FLOAT,
firma         NVARCHAR(20),
qveyana       NVARCHAR(20),
kursi         FLOAT,
pasi_lari     FLOAT,
pasi_dolari   FLOAT
);
--
IF OBJECT_ID(N'Gayidvebi',N'U') IS NOT NULL
   DROP TABLE Gayidvebi;
```

```
--      ცხრილის შექმნა
CREATE TABLE Gayidvebi
(
gayidvebiID     INT PRIMARY KEY IDENTITY (1,1),
produqtebiID    INT NULL REFERENCES Produqtebi(produqtebiID)
                                ON DELETE CASCADE,
raodenoba       INT,
tanxa           FLOAT,
tarigi_gayidvis DATETIME
);
```

3.11.

```
USE Magazia_1;
--      თუ ცხრილი არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID(N'Produqtebi_1',N'U') IS NOT NULL
   DROP TABLE Produqtebi_1;
--      ცხრილის შექმნა
CREATE TABLE Produqtebi_1
(
produqtebi_1ID      INT PRIMARY KEY IDENTITY (1,1),
dasaxeleba          NVARCHAR(20),
raodenoba           FLOAT,
firma               NVARCHAR(20),
qveyana             NVARCHAR(20),
kursi               FLOAT,
pasi_lari           FLOAT,
pasi_dolari         FLOAT
);
--
IF OBJECT_ID(N'Gayidvebi_1',N'U') IS NOT NULL
   DROP TABLE Gayidvebi_1;
--      ცხრილის შექმნა
CREATE TABLE Gayidvebi_1
(
gayidvebi_1ID   INT PRIMARY KEY IDENTITY (1,1),
produqtebi_1ID  INT NULL REFERENCES Produqtebi_1(produqtebi_1ID)
                                ON DELETE NO ACTION,
raodenoba       INT,
tanxa           FLOAT,
tarigi_gayidvis DATETIME
);
```

3.12.

```
USE Magazia_1;
--      თუ ცხრილი არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID(N'Produqtebi_2',N'U') IS NOT NULL
   DROP TABLE Produqtebi_2;
--      ცხრილის შექმნა
```

```
CREATE TABLE Produqtebi_2
(
produqtebi_2ID        INT PRIMARY KEY IDENTITY (1,1),
dasaxeleba            NVARCHAR(20),
raodenoba             FLOAT,
firma                 NVARCHAR(20),
qveyana               NVARCHAR(20),
kursi                 FLOAT,
pasi_lari             FLOAT,
pasi_dolari           FLOAT
);
--
IF OBJECT_ID(N'Gayidvebi_2',N'U') IS NOT NULL
   DROP TABLE Gayidvebi_2;
--       ცხრილის შექმნა
CREATE TABLE Gayidvebi_2
(
gayidvebi_2ID    INT PRIMARY KEY IDENTITY (1,1),
produqtebi_2ID   INT NULL REFERENCES Produqtebi_2(produqtebi_2ID)
                          ON UPDATE CASCADE,
raodenoba        INT,
tanxa            FLOAT,
tarigi_gayidvis  DATETIME
);
3.13.
USE Magazia_1;
--       თუ ცხრილი არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID(N'Produqtebi_3',N'U') IS NOT NULL
   DROP TABLE Produqtebi_3;
--       ცხრილის შექმნა
CREATE TABLE Produqtebi_3
(
produqtebi_3ID        INT PRIMARY KEY IDENTITY (1,1),
dasaxeleba            NVARCHAR(20),
raodenoba             FLOAT,
firma                 NVARCHAR(20),
qveyana               NVARCHAR(20),
kursi                 FLOAT,
pasi_lari             FLOAT,
pasi_dolari           FLOAT
);
--
IF OBJECT_ID(N'Gayidvebi_3',N'U') IS NOT NULL
   DROP TABLE Gayidvebi_3;
--       ცხრილის შექმნა
CREATE TABLE Gayidvebi_3
```

```
(
gayidvebi_3ID   INT PRIMARY KEY IDENTITY (1,1),
produqtebi_3ID  INT NULL REFERENCES Produqtebi_3(produqtebi_3ID)
                              ON UPDATE NO ACTION,
raodenoba       NT,
tanxa           FLOAT,
tarigi_gayidvis DATETIME
);
```

3.14.

```
USE Bina;
--      თუ ცხრილი არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID(N'Binebi_3',N'U') IS NOT NULL
   DROP TABLE Binebi_3;
--      ცხრილის შექმნა
CREATE TABLE Binebi_3
(
binebi_3ID          INT CONSTRAINT FK_C1 PRIMARY KEY (binebi_3ID DESC),
qalaqi              NVARCHAR(20),
raioni              NVARCHAR(20),
quchis_saxeli       NVARCHAR(20),
quchis_nomeri       NVARCHAR(7),
teleponi_qalaqis    NVARCHAR(9),
kursi               FLOAT,
fasi_dolari         FLOAT,
fasi_lari           FLOAT,
binis_proeqti       NVARCHAR(12),
mflobelis_gvari     NVARCHAR(15) NOT NULL,
mflobelis_saxeli    NVARCHAR(15),
tarigi_ashenebis    DATETIME
);
```

3.15.

```
CREATE DATABASE Saavadmyofo;
GO
USE Saavadmyofo;
--      თუ ცხრილი არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID(N'Eqimi',N'U') IS NOT NULL
   DROP TABLE Eqimi;
--      ცხრილის შექმნა
CREATE TABLE Eqimi
(
eqimiID             INT PRIMARY KEY IDENTITY (1,1),
gvari               NVARCHAR(20),
saxeli              NVARCHAR(20),
ganyofileba         NVARCHAR(20),
tanamdeboba         NVARCHAR(7),
staji               INT,
```

```
xelfasi                 FLOAT,
tarigi_dabadebis        DATETIME,
mobiluri                NVARCHAR(12),
qalaqi                  NVARCHAR(15),
misamarti               NVARCHAR(20)
);
--      თუ ცხრილი არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID(N'Avadmyofi', N'U') IS NOT NULL
   DROP TABLE Avadmyofi;
--      ცხრილის შექმნა
CREATE TABLE Avadmyofi
(
avadmyofiID             INT PRIMARY KEY IDENTITY (1,1),
eqimiID                 INT NULL CONSTRAINT FK_C2  REFERENCES Eqimi(eqimiID)
                                ON DELETE CASCADE,
gvari                   NVARCHAR(20),
saxeli                  NVARCHAR(20),
ganyofileba             NVARCHAR(20),
tanxa                   FLOAT,
dabadebis_tarigi        DATETIME,
mobiluri                NVARCHAR(12),
qalaqi                  NVARCHAR(15),
misamarti               NVARCHAR(20),
diagnozi                NVARCHAR(300),
palatis_nomeri          NVARCHAR(7)
);
```

3.16.

```
USE Saavadmyofo;
--      თუ ცხრილი არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID(N'Eqimi_1',N'U') IS NOT NULL
   DROP TABLE Eqimi_1;
--      ცხრილის შექმნა
CREATE TABLE Eqimi_1
(
eqimi_1ID               INT PRIMARY KEY IDENTITY (1,1),
gvari                   NVARCHAR(20),
saxeli                  NVARCHAR(20),
ganyofileba             NVARCHAR(20),
tanamdeboba             NVARCHAR(7),
staji                   INT,
xelfasi                 FLOAT,
tarigi_dabadebis        DATETIME,
mobiluri                NVARCHAR(12),
qalaqi                  NVARCHAR(15),
misamarti               NVARCHAR(20)
);
```

```
--      თუ ცხრილი არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID(N'Avadmyofi_1', N'U') IS NOT NULL
   DROP TABLE Avadmyofi_1;
--      ცხრილის შექმნა
CREATE TABLE Avadmyofi_1
(
avadmyofi_1ID       INT PRIMARY KEY IDENTITY (1,1),
eqimi_1ID           INT NULL CONSTRAINT FK_C3 REFERENCES Eqimi_1(eqimi_1ID)
                        ON DELETE NO ACTION,
gvari               NVARCHAR(20),
saxeli              NVARCHAR(20),
ganyofileba         NVARCHAR(20),
tanxa               FLOAT,
dabadebis_tarigi    DATETIME,
mobiluri            NVARCHAR(12),
qalaqi              NVARCHAR(15),
misamarti           NVARCHAR(20),
diagnozi            NVARCHAR(300),
palatis_nomeri      NVARCHAR(7)
);
```

3.17.

```
USE Saavadmyofo;
--      თუ ცხრილი არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID(N'Eqimi_2',N'U') IS NOT NULL
   DROP TABLE Eqimi_2;
--      ცხრილის შექმნა
CREATE TABLE Eqimi_2
(
eqimi_2ID           INT PRIMARY KEY IDENTITY (1,1),
gvari               NVARCHAR(20),
saxeli              NVARCHAR(20),
ganyofileba         NVARCHAR(20),
tanamdeboba         NVARCHAR(7),
staji               INT,
xelfasi             FLOAT,
tarigi_dabadebis    DATETIME,
mobiluri            NVARCHAR(12),
qalaqi              NVARCHAR(15),
misamarti           NVARCHAR(20)
);
--      თუ ცხრილი არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID(N'Avadmyofi_2', N'U') IS NOT NULL
   DROP TABLE Avadmyofi_2;
--      ცხრილის შექმნა
CREATE TABLE Avadmyofi_2
(
```

```
avadmyofi_2ID       INT PRIMARY KEY IDENTITY (1,1),
eqimi_2ID           INT NULL CONSTRAINT FK_C4 REFERENCES Eqimi_2(eqimi_2ID)
                                ON UPDATE CASCADE ,
gvari               NVARCHAR(20),
saxeli              NVARCHAR(20),
ganyofileba         NVARCHAR(20),
tanxa               FLOAT,
dabadebis_tarigi    DATETIME,
mobiluri            NVARCHAR(12),
qalaqi              NVARCHAR(15),
misamarti           NVARCHAR(20),
diagnozi            NVARCHAR(300),
palatis_nomeri      NVARCHAR(7)
);
```

3.18.

```
USE Saavadmyofo;
--      თუ ცხრილი არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID(N'Eqimi_3',N'U') IS NOT NULL
   DROP TABLE Eqimi_3;
--      ცხრილის შექმნა
CREATE TABLE Eqimi_3
(
eqimi_3ID           INT PRIMARY KEY IDENTITY (1,1),
gvari               NVARCHAR(20),
saxeli              NVARCHAR(20),
ganyofileba         NVARCHAR(20),
tanamdeboba         NVARCHAR(7),
staji               INT,
xelfasi             FLOAT,
tarigi_dabadebis    DATETIME,
mobiluri            NVARCHAR(12),
qalaqi              NVARCHAR(15),
misamarti           NVARCHAR(20)
);
--      თუ ცხრილი არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID(N'Avadmyofi_3', N'U') IS NOT NULL
   DROP TABLE Avadmyofi_3;
--      ცხრილის შექმნა
CREATE TABLE Avadmyofi_3
(
avadmyofi_3ID       INT PRIMARY KEY IDENTITY (1,1),
eqimi_3ID           INT NULL CONSTRAINT FK_C5 REFERENCES Eqimi_3(eqimi_3ID)
                                ON UPDATE NO ACTION ,
gvari               NVARCHAR(20),
saxeli              NVARCHAR(20),
ganyofileba         NVARCHAR(20),
```

```
tanxa                FLOAT,
dabadebis_tarigi     DATETIME,
mobiluri             NVARCHAR(12),
qalaqi               NVARCHAR(15),
misamarti            NVARCHAR(20),
diagnozi             NVARCHAR(300),
palatis_nomeri       NVARCHAR(7)
);
```

3.19.

```
CREATE DATABASE Biblioteka;
GO
USE Biblioteka;
--      თუ ცხრილი არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID(N'Cignebi',N'U') IS NOT NULL
   DROP TABLE Cignebi;
--      ცხრილის შექმნა
CREATE TABLE Cignebi
(
cignebiID            INT PRIMARY KEY IDENTITY (1,1),
avtoris_gvari        NVARCHAR(20),
dasaxeleba           NVARCHAR(50),
weli                 INT,
ISBN_kodi            NVARCHAR(25),
tiraji               INT,
gamomcemloba         NVARCHAR(50),
fasi                 FLOAT,
recenzenti           NVARCHAR(20),
anotacia             NVARCHAR(300),
gverdebis_raod       INT
);
--      თუ ცხრილი არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID(N'Abonentebi', N'U') IS NOT NULL
   DROP TABLE Abonentebi;
--      ცხრილის შექმნა
CREATE TABLE Abonentebi
(
abonentebiID         INT PRIMARY KEY IDENTITY (1,1),
cignebi              INT NULL CONSTRAINT FK_C6  REFERENCES Cignebi(cignebiID)
                              ON UPDATE CASCADE
                              ON DELETE CASCADE,
gvari                NVARCHAR(20),
saxeli               NVARCHAR(20),
organizacia          NVARCHAR(20),
piradi_nomeri        NVARCHAR(11),
dabadebis_tarigi     DATETIME,
mobiluri             NVARCHAR(12),
```

```
qalaqi                  NVARCHAR(15),
misamarti               NVARCHAR(20)
);
```

3.20.

```
USE Biblioteka;
--      თუ ცხრილი არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID(N'Cignebi_1',N'U') IS NOT NULL
   DROP TABLE Cignebi_1;
--      ცხრილის შექმნა
CREATE TABLE Cignebi_1
(
cignebi_1ID             INT PRIMARY KEY IDENTITY (1,1),
avtoris_gvari           NVARCHAR(20),
dasaxeleba              NVARCHAR(50),
weli                    INT,
ISBN_kodi               NVARCHAR(25),
tiraji                  INT,
gamomcemloba            NVARCHAR(50),
fasi                    FLOAT,
recenzenti              NVARCHAR(20),
anotacia                NVARCHAR(300),
gverdebis_raod          INT
);
--      თუ ცხრილი არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID(N'Abonentebi_1', N'U') IS NOT NULL
   DROP TABLE Abonentebi_1;
--      ცხრილის შექმნა
CREATE TABLE Abonentebi_1
(
abonentebi_1ID          INT PRIMARY KEY IDENTITY (1,1),
cignebi_1ID             INT NULL,
gvari                   NVARCHAR(20),
saxeli                  NVARCHAR(20),
organizacia             NVARCHAR(20),
piradi_nomeri           NVARCHAR(11),
dabadebis_tarigi        DATETIME,
mobiluri                NVARCHAR(12),
qalaqi                  NVARCHAR(15),
misamarti               NVARCHAR(20),
CONSTRAINT FK_C7  FOREIGN KEY (cignebi_1ID) REFERENCES Cignebi_1(cignebi_1ID)
                        ON UPDATE CASCADE
                        ON DELETE CASCADE
);
```

3.21.

```
USE Kadrebi;
GO
```

```
sp_rename 'Tanamshromlebi_3', 'Tanamshromlebi_4';
```
3.22.
```
USE Kadrebi;
DROP TABLE Tanamshromlebi_4;
```

თავი 4. მონაცემების მართვა

Transact SQL-ის საფუძვლები

4.1.
```
USE Shekveta;
SET QUOTED_IDENTIFIER ON;
SELECT [SET], [firmis saxeli]
FROM [Table];
```
4.2.
```
DECLARE @mteli INT, @ciladi FLOAT, @striqoni NVARCHAR(10), @tarigi DATETIME;
SET @mteli     = 5;
SET @ciladi    = 5.5;
SET @striqoni = N'რომანი';
SET @tarigi    = '01.01.2016';
```
4.3.
```
USE Shekveta;
DECLARE @minimaluri_staji INT;
SELECT @minimaluri_staji = MIN(staji) FROM Personali;
SELECT @minimaluri_staji AS [მინიმალური სტაჟი];
```
4.4.
```
USE Shekveta;
DECLARE @maximaluri_staji INT;
SET @maximaluri_staji =
(
SELECT MAX(staji)
FROM Personali
);
SELECT @maximaluri_staji AS [მაქსიმალური სტაჟი];
```

INSERT ბრძანება

4.1.1.
```
USE Shekveta;
SET DATEFORMAT DMY;
INSERT INTO Shemkveti VALUES (N'ფიზიკური', N'სამხარაძე', N'რომანი', N'თბილისი', NULL, N'დიდუბე', N'522-222-222', NULL, N'კაცი', NULL, N'სტალინის ქ.5', N'VTB ბანკი', N'rs@geonet.ge', N'ბენაშვილი', N'სანდრო', N'521-212-121');
SELECT * FROM Shemkveti;
```
4.1.2.
```
USE Shekveta;
SET DATEFORMAT DMY;
INSERT INTO Shemkveti (saxeli, gvari, iuridiuli_fizikuri, qalaqi, raioni, misamarti,
```

mobiluri_direqtoris, firmis_dasaxeleba, email, banki, sqesi, regioni, gvari_warmomadgenlis, saxeli_warmomadgenlis, mobiluri_warmomadgenlis, mobiluri)
VALUES (N'ნინო', N'ყანჩაველი', N'იურიდიული', N'ბათუმი', NULL, N'ც. დადიანის 2', '500-099-900', N'Nino&Co.', N'nino@mail.ru', N'სტანდარტ ბანკი', N'ქალი', N'აჭარა', N'დვალიშვილი', N'მარინა', '510-199-910', NULL);
SELECT * FROM Shemkveti;
4.1.3.
USE Shekveta;
SET DATEFORMAT DMY;
INSERT INTO Shemkveti VALUES (N'ფიზიკური', N'სამხარაძე', N'რომანი', N'თბილისი', NULL, NULL, N'522-222-222', NULL, NULL, N'სტალინის ქ.5', N'VTB ბანკი', N'rs@geonet.ge', NULL, NULL, NULL, NULL);
SELECT * FROM Shemkveti;
4.1.4.
USE Shekveta;
INSERT INTO Shemkveti VALUES (N'ფიზიკური', N'სამხარაძე', N'რომანი', N'თბილისი', DEFAULT, DEFAULT, N'522-222-222', DEFAULT, DEFAULT, N'სტალინის ქ.5', N'VTB ბანკი', N'rs@geonet.ge', DEFAULT, DEFAULT, DEFAULT, DEFAULT);
SELECT * FROM Shemkveti;
4.1.5.
USE Shekveta;
INSERT INTO Shemkveti DEFAULT VALUES;
SELECT * FROM Shemkveti;
4.1.6.
USE Shekveta;
SET DATEFORMAT DMY;
INSERT INTO Shemkveti VALUES (N'ფიზიკური', N'სამხარაძე', N'რომანი', N'თბილისი', NULL, N'ვაკე', NULL, NULL, N'კაცი', NULL, N'სტალინის ქ.5', NULL, N'rs@geonet.ge', NULL, NULL, NULL);
INSERT INTO Shemkveti VALUES (N'იურიდიული', N'ღავთაძე', N'მანანა', N'ქუთაისი', NULL, NULL, NULL, '524-242-224', N'ქალი', NULL, N'აღმაშენებლის ქ.15', N'TBC ბანკი', NULL, N'ბენაშვილი', N'სანდრო', '521-212-121');
SELECT * FROM Shemkveti;
4.1.7.
USE Shekveta;
INSERT INTO Shemkveti VALUES (N'ფიზიკური', N'სამხარაძე', N'რომანი', N'თბილისი', NULL, N'ვაკე', N'533-232-232', NULL, N'კაცი', NULL, N'სტალინის ქ.5', NULL, N'rs@geonet.ge', NULL, NULL, NULL),
(N'იურიდიული', N'ღავთაძე', N'მანანა', N'ქუთაისი', NULL, NULL, NULL, N'524-242-224', N'ქალი',
NULL, N'აღმაშენებლის ქ.15', N'TBC ბანკი', NULL, N'ბენაშვილი', N'სანდრო', N'521-212-121'),
(N'ფიზიკური', N'მზარელუა', N'მიხეილი', N'ზუგდიდი', NULL, NULL, N'554-252-254', NULL, N'კაცი', NULL, N'ვეძინის ქ.12', NULL, NULL, N'მზარელუა', N'ციალა', N'526-612-621');
SELECT * FROM Shemkveti;

4.1.8.

```
USE Shekveta;
CREATE TABLE Shemkveti_1
(
shemkvetiID             INT,
iuridiuli_fizikuri      NVARCHAR(10),
gvari                   NVARCHAR(30),
xaseli                  NVARCHAR(15),
mobiluri_direqtoris     NVARCHAR(12),
firmis_dasaxeleba       NVARCHAR(30),
misamarti               NVARCHAR(50),
banki                   NVARCHAR(15)
);
INSERT INTO Shemkveti_1
SELECT  shemkvetiID, iuridiuli_fizikuri, gvari, saxeli, mobiluri_warmomadgenlis,
                firmis_dasaxeleba, misamarti, banki
FROM Shemkveti
WHERE qalaqi = N'თელავი';
SELECT * FROM Shemkveti_1;
```

4.1.9.

```
USE Shekveta;
IF EXISTS (
SELECT *
FROM tempdb..sysobjects
WHERE id = OBJECT_ID(N'tempdb..#Droebiti_Shemkveti')
)
DROP TABLE #Droebiti_Shemkveti;
--      ცხრილის შექმნა
CREATE TABLE #Droebiti_Shemkveti
(
gvari               NVARCHAR (30),
ganyofileba         NVARCHAR (30),
qalaqi              NVARCHAR (30),
firmis_dasaxeleba NVARCHAR (30)
);
--      სტრიქონების ჩამატება და ეკრანზე გამოტანა
INSERT INTO #Droebiti_Shemkveti
SELECT gvari, saxeli, qalaqi, firmis_dasaxeleba
FROM Shemkveti
WHERE qalaqi = N'ბათუმი';
SELECT * FROM #Droebiti_Shemkveti;
```

SELECT ... INTO ბრძანება

4.2.1.

```
USE Shekveta;
SELECT gvari AS [gvari], saxeli AS [saxeli], iuridiuli_fizikuri AS [iuridiuli_fizikuri],
```

```
        firmis_dasaxeleba AS [firmis_dasaxeleba]
INTO Shemkveti_2
FROM Shemkveti;
SELECT * FROM Shemkveti_2;
4.2.2.
USE Shekveta;
SELECT *
INTO Shemkveti_3
FROM Shemkveti
WHERE qalaqi = N'თბილისი';
SELECT * FROM Shemkveti_3;
```

SELECT ბრძანება
SELECT განყოფილება.

```
4.3.1.
Use Shekveta;
SELECT shemkvetiID
FROM Xelshekruleba;
ან
SELECT ALL shemkvetiID
FROM Xelshekruleba;
4.3.2.
Use Shekveta;
SELECT DISTINCT shemkvetiID
FROM Xelshekruleba;
4.3.3.
Use Shekveta;
SELECT TOP 7 *
FROM Shemkveti;
4.3.4.
Use Shekveta;
SELECT TOP 60 PERCENT *
FROM Shemkveti;
4.3.5.
Use Shekveta;
SELECT xelshekrulebaID AS [ხელშეკრულების ნომერი], tarigi_dawyebis AS [დაწყების თარიღი]
FROM Xelshekruleba;
4.3.6.
Use Shekveta;
SELECT  Shemkveti.gvari  AS [შემკვეთის გვარი],
        Shemkveti.saxeli AS [შემკვეთის სახელი],
        Xelshekruleba.xelshekrulebaID AS [ხელშეკრულების ნომერი]
FROM    Shemkveti, Xelshekruleba
WHERE  Shemkveti.shemkvetiID = Xelshekruleba.shemkvetiID;
```

WHERE განყოფილება

4.4.1.

```
USE Shekveta;
SELECT Shemkveti.gvari, Shemkveti.firmis_dasaxeleba, Shemkveti.mobiluri,
       Xelshekruleba.vali_l, Xelshekruleba.gadasaxdeli_l, Xelshekruleba.gadaxdili_l,
       Xelshekruleba.tarigi_dawyebis
FROM   Shemkveti, Xelshekruleba
WHERE Xelshekruleba.vali_l > 0 AND Shemkveti.shemkvetiID = Xelshekruleba.shemkvetiID;
```

4.4.2.

```
USE Shekveta;
SELECT *
FROM Personali
WHERE ganyofileba = N'სამედიცინო';
```

4.4.3.

```
USE Shekveta;
SELECT *
FROM Personali
WHERE qalaqi = N'თბილისი';
```

4.4.4.

```
USE Shekveta;
SELECT *
FROM Personali
WHERE raioni = N'დიდუბე';
```

4.4.5.

```
USE Shekveta;
SELECT *
FROM Personali
WHERE asaki >= 40;
```

4.4.6.

```
USE Shekveta;
SELECT *
FROM Personali
WHERE staji >= 20;
```

4.4.7.

```
USE Shekveta;
SELECT *
FROM Personali
WHERE staji >= 20 AND asaki >= 45;
```

4.4.8.

```
USE Shekveta;
SELECT *
FROM Personali
WHERE ganyofileba = N'სამედიცინო' AND asaki >= 30;
```

4.4.9.

```
USE Shekveta;
```

```
SELECT *
FROM Personali
WHERE qalaqi = N'თბილისი' AND staji >= 15;
4.4.10.
USE Shekveta;
SELECT *
FROM Personali
WHERE ganyofileba = N'სამედიცინო' AND xelfasi >= 1500;

4.4.11.
USE Shekveta;
SELECT gadasaxdeli_l AS [გადასახდელი თანხა]
FROM Xelshekruleba
WHERE gadasaxdeli_l > 6500;
4.4.12.
USE Shekveta;
SELECT gadasaxdeli_l AS [გადასახდელი თანხა]
FROM Xelshekruleba
WHERE NOT gadasaxdeli_l > 6500;
--          ან
USE Shekveta;
SELECT gadasaxdeli_l AS [გადასახდელი თანხა]
FROM Xelshekruleba
WHERE gadasaxdeli_l <= 6500;
4.4.13.
USE Shekveta;
SELECT  S.gvari AS [შემკვეთის გვარი], X.vali_l AS [ვალი]
FROM    Shemkveti S, Xelshekruleba X
WHERE S.qalaqi = N'თბილისი' AND X.vali_l > 0 AND S.shemkvetiID = X.shemkvetiID;
4.4.14.
USE Shekveta;
SELECT *
FROM Shemkveti
WHERE iuridiuli_fizikuri = N'იურიდიული';
4.4.15.
USE Shekveta;
SELECT *
FROM Personali
WHERE asaki > 50 OR asaki < 30;
4.4.16.
USE Shekveta;
SELECT *
FROM Personali
WHERE NOT ( qalaqi =N'თბილისი' OR qalaqi = N'ზათუმი' );
4.4.17.
USE Shekveta;
```

```
SELECT * FROM Personali
WHERE staji > 20;
```
4.4.18.
```
USE Shekveta;
SELECT Shemkveti.gvari, Xelshekruleba.vali_l
FROM Shemkveti, Xelshekruleba
WHERE Shemkveti.shemkvetiID = Xelshekruleba.shemkvetiID
    AND Xelshekruleba.vali_l > 0;
```
4.4.19.
```
USE Shekveta_1;
SELECT *
FROM Xelshekruleba
WHERE shesruleba = N'ჰ' AND tarigi_damtavrebis < tarigi_shesrulebis;
```
4.4.20.
```
USE Shekveta_1;
SELECT *
FROM Xelshekruleba
WHERE shesruleba = N'ჰ' AND tarigi_damtavrebis > tarigi_shesrulebis;
```
4.4.21.
```
USE Shekveta_1;
SELECT *
FROM Xelshekruleba
WHERE shesruleba = N'ჰ' AND tarigi_damtavrebis = tarigi_shesrulebis;
```
4.4.22.
```
USE Shekveta;
SET DATEFORMAT DMY;
SELECT *
FROM Xelshekruleba
WHERE shesruleba = N'ჰ'
    AND tarigi_damtavrebis > tarigi_shesrulebis
    AND tarigi_damtavrebis >= '01.01.2012' AND tarigi_damtavrebis <= '31.12.2012';
```
4.4.23.
```
USE Shekveta;
SET DATEFORMAT DMY;
SELECT *
FROM Xelshekruleba
WHERE shesruleba = N'ჰ'
    AND tarigi_damtavrebis < tarigi_shesrulebis
    AND tarigi_damtavrebis >= '01.01.2013' AND tarigi_damtavrebis <= '31.12.2013';
```
4.4.24.
```
USE Shekveta;
SET DATEFORMAT DMY;
SELECT *
FROM Xelshekruleba
WHERE shesruleba = N'ჰ'
    AND tarigi_damtavrebis = tarigi_shesrulebis
```

```
    AND tarigi_damtavrebis >= '01.01.2012' AND tarigi_damtavrebis <= '31.12.2012';
```
4.4.25.
```
USE Shekveta;
SET DATEFORMAT DMY;
DECLARE @tarigi_1 DATETIME, @tarigi_2 DATETIME;
SET @tarigi_1 = N'01.01.2014';
SET @tarigi_2 = N'31.12.2014';
SELECT *
FROM Xelshekruleba
WHERE shesruleba = N'ჯი'
    AND tarigi_damtavrebis > tarigi_shesrulebis
    AND tarigi_damtavrebis >= @tarigi_1 AND tarigi_damtavrebis <= @tarigi_2;
```

4.4.26.
```
USE Shekveta;
SET DATEFORMAT DMY;
DECLARE @tarigi_1 DATETIME, @tarigi_2 DATETIME;
SET @tarigi_1 = N'01.01.2014';
SET @tarigi_2 = N'31.12.2014';
SELECT *
FROM Xelshekruleba
WHERE shesruleba = N'ჯი'
    AND tarigi_damtavrebis < tarigi_shesrulebis
    AND tarigi_damtavrebis >= @tarigi_1 AND tarigi_damtavrebis <= @tarigi_2;
```
4.4.27.
```
USE Shekveta;
SET DATEFORMAT DMY;
DECLARE @tarigi_1 DATETIME, @tarigi_2 DATETIME;
SET @tarigi_1 = N'01.01.2014';
SET @tarigi_2 = N'31.12.2014';
SELECT *
FROM Xelshekruleba
WHERE shesruleba = N'ჯი'
    AND tarigi_damtavrebis = tarigi_shesrulebis
    AND tarigi_damtavrebis >= @tarigi_1 AND tarigi_damtavrebis <= @tarigi_2;
```
4.4.32.
```
USE Shekveta;
SELECT *
FROM Xelshekruleba
WHERE shesruleba = N'ჯი';
```
4.4.33.
```
USE Shekveta;
SET DATEFORMAT DMY;
SELECT *
FROM Xelshekruleba
WHERE tarigi_damtavrebis < GETDATE() AND shesruleba = N'არა';
```

4.4.34.
```
USE Shekveta;
SET DATEFORMAT DMY;
SELECT *
FROM Xelshekruleba
WHERE shesruleba = N'კი'
    AND tarigi_damtavrebis >= '01.01.2012' AND tarigi_damtavrebis <='31.12.2012';
```
4.4.35.
```
USE Shekveta;
SET DATEFORMAT DMY;
SELECT *
FROM Xelshekruleba
WHERE shesruleba = N'არა' AND tarigi_damtavrebis > GETDATE();
```
4.4.37.
```
USE Shekveta;
SET DATEFORMAT DMY;
SELECT xelshekrulebaID, DATEDIFF(month, tarigi_shesrulebis, tarigi_damtavrebis)
FROM Xelshekruleba
WHERE tarigi_shesrulebis > tarigi_damtavrebis AND shesruleba = N'კი';
```
4.4.38.
```
USE Shekveta;
SET DATEFORMAT DMY;
SELECT xelshekrulebaID, DATEDIFF(month, tarigi_shesrulebis, tarigi_damtavrebis)
FROM Xelshekruleba
WHERE tarigi_shesrulebis < tarigi_damtavrebis AND shesruleba = N'კი';
```

BETWEEN ოპერატორი

4.5.1.
```
USE Shekveta;
SELECT *
FROM Personali
WHERE asaki BETWEEN 40 AND 50;
```
4.5.2.
```
USE Shekveta;
SET DATEFORMAT DMY;
SELECT *
FROM Personali
WHERE tarigi_dabadebis BETWEEN '1.03.1985' AND '31.03.1985';
```
4.5.3.
```
USE Shekveta;
SELECT *
FROM Personali
WHERE staji BETWEEN 10 AND 20;
```
4.4.4.
```
USE Shekveta;
SELECT *
```

```
FROM Personali
WHERE xelfasi BETWEEN 1000 AND 2000;
```
4.4.5.
```
USE Shekveta;
SELECT *
FROM Xelshekruleba
WHERE vali_l BETWEEN 500 AND 1000;
```
4.4.6.
```
USE Shekveta;
SELECT *
FROM Xelshekruleba
WHERE gadasaxdeli_l BETWEEN 4500 AND 7000;
```
4.4.7.
```
USE Shekveta;
SELECT Shemkveti.gvari, Xelshekruleba.gadasaxdeli_l
FROM Shemkveti, Xelshekruleba
WHERE Xelshekruleba.gadasaxdeli_l BETWEEN 4500 AND 7000
        AND Shemkveti.shemkvetiID = Xelshekruleba.shemkvetiID;
```
4.4.8.
```
USE Shekveta;
SELECT Shemkveti.gvari, Xelshekruleba.vali_l
FROM Shemkveti, Xelshekruleba
WHERE Xelshekruleba.vali_l BETWEEN 500 AND 1000
   AND Shemkveti.shemkvetiID = Xelshekruleba.shemkvetiID;
```
4.4.9.
```
USE Shekveta;
SELECT Personali.gvari, Xelshekruleba.vali_l
FROM Personali, Xelshekruleba
WHERE Xelshekruleba.gadasaxdeli_l BETWEEN 3500 AND 5000
    AND Personali.personaliID = Xelshekruleba.personaliID;
```
4.4.10.
```
USE Shekveta;
SELECT  Personali.gvari, Xelshekruleba.vali_l
FROM    Personali, Xelshekruleba
WHERE Xelshekruleba.vali_l BETWEEN 500 AND 1000
    AND Personali.personaliID = Xelshekruleba.personaliID;
```
4.5.11.
```
USE Shekveta;
SELECT   Personali.gvari, Personali.saxeli, Xelshekruleba.tarigi_dawyebis
FROM     Personali, Xelshekruleba
WHERE  Xelshekruleba.tarigi_dawyebis BETWEEN '01.01.2010' AND '01.01.2015'
    AND  Personali.personaliID = Xelshekruleba.personaliID;
```
4.5.12.
```
USE Shekveta;
SELECT  Shemkveti.gvari, Shemkveti.saxeli, Xelshekruleba.tarigi_dawyebis
FROM     Shemkveti, Xelshekruleba
```

```
WHERE  Xelshekruleba.tarigi_dawyebis BETWEEN '01.01.2010' AND '01.01.2015'
   AND  Shemkveti.shemkvetiID = Xelshekruleba.shemkvetiID;
```

4.4.13.

```
USE Shekveta;
SET DATEFORMAT DMY;
SELECT *
FROM Xelshekruleba
WHERE shesruleba = N'ვი'
    AND tarigi_damtavrebis > tarigi_shesrulebis
    AND tarigi_damtavrebis BETWEEN '01.01.2012' AND '31.12.2012';
```

4.4.14.

```
USE Shekveta;
SET DATEFORMAT DMY;
SELECT *
FROM Xelshekruleba
WHERE shesruleba = N'ვი'
    AND tarigi_damtavrebis < tarigi_shesrulebis
    AND tarigi_damtavrebis BETWEEN '01.01.2013' AND '31.12.2013';
```

4.4.15.

```
USE Shekveta;
SET DATEFORMAT DMY;
SELECT *
FROM Xelshekruleba
WHERE shesruleba = N'ვი'
    AND tarigi_damtavrebis = tarigi_shesrulebis
    AND tarigi_damtavrebis BETWEEN '01.01.2012' AND '31.12.2012';
```

4.4.16.

```
USE Shekveta;
SET DATEFORMAT DMY;
DECLARE @tarigi_1 DATETIME, @tarigi_2 DATETIME;
SET @tarigi_1 = N'01.01.2014';
SET @tarigi_2 = N'31.12.2014';
SELECT *
FROM Xelshekruleba
WHERE shesruleba = N'ვი'
    AND tarigi_damtavrebis > tarigi_shesrulebis
    AND tarigi_damtavrebis BETWEEN @tarigi_1 AND @tarigi_2;
```

4.4.17.

```
USE Shekveta;
SET DATEFORMAT DMY;
DECLARE @tarigi_1 DATETIME, @tarigi_2 DATETIME;
SET @tarigi_1 = N'01.01.2014';
SET @tarigi_2 = N'31.12.2014';
SELECT *
FROM Xelshekruleba
```

```
WHERE shesruleba = N'კი'
    AND tarigi_damtavrebis < tarigi_shesrulebis
    AND tarigi_damtavrebis BETWEEN @tarigi_1 AND @tarigi_2;
```

4.4.18.

```
USE Shekveta;
SET DATEFORMAT DMY;
DECLARE @tarigi_1 DATETIME, @tarigi_2 DATETIME;
SET @tarigi_1 = N'01.01.2014';
SET @tarigi_2 = N'31.12.2014';
SELECT *
FROM Xelshekruleba
WHERE shesruleba = N'კი'
    AND tarigi_damtavrebis = tarigi_shesrulebis
    AND tarigi_damtavrebis BETWEEN @tarigi_1 AND @tarigi_2;
```

4.4.19.

```
USE Shekveta;
SET DATEFORMAT DMY;
SELECT *
FROM Xelshekruleba
WHERE shesruleba = N'კი'
    AND tarigi_damtavrebis BETWEEN '01.01.2012' AND '31.12.2012';
```

4.5.20.

```
USE Shekveta;
SELECT *
FROM Personali
WHERE sqesi = N'ქალი'
    AND asaki BETWEEN 45 AND 55;
```

4.5.22.

```
USE Shekveta;
SELECT *
FROM Personali
WHERE sqesi  = N'ქალი'
   AND qalaqi = N'თბილისი'
   AND asaki BETWEEN 40 AND 55;
```

4.5.24.

```
USE Shekveta;
SELECT *
FROM Personali
WHERE sqesi  = N'ქალი'
    AND staji BETWEEN 5 AND 10;
```

4.5.26.

```
USE Shekveta;
SELECT *
FROM Personali
WHERE sqesi  = N'ქალი'
    AND qalaqi = N'თბილისი'
```

```
    AND staji BETWEEN 9 AND 20;
```
4.5.28.
```
USE Shekveta;
SELECT *
FROM Personali
WHERE sqesi = N'ქალი'
   AND xelfasi BETWEEN 1000 AND 2000;
```
4.5.30.
```
USE Shekveta;
SELECT *
FROM Personali
WHERE sqesi = N'ქალი'
    AND staji > 6
    AND xelfasi BETWEEN 1000 AND 2000;
```

IN ოპერატორი

4.6.1.
```
USE Shekveta;
SELECT * FROM Personali WHERE qalaqi IN (N'ბათუმი', N'ქობულეთი', N'გორი');
```
4.6.2.
```
USE Shekveta;
SELECT * FROM Personali WHERE ganyofileba IN (N'სამედიცინო', N'სასპორტო');
```
4.6.3.
```
USE Shekveta;
SELECT * FROM Personali WHERE raioni IN (N'დიდუბე', N'ვაკე', N'საბურთალო');
```
4.6.4.
```
USE Shekveta;
SELECT * FROM Shemkveti WHERE qalaqi IN (N'ბათუმი', N'ქობულეთი', N'გორი');
```
4.6.5.
```
USE Shekveta;
SELECT * FROM Personali WHERE staji IN (5, 10, 15);
```
4.6.6.
```
USE Shekveta;
SELECT * FROM Personali WHERE asaki IN (25, 30, 35);
```
4.6.7.
```
USE Shekveta;
SELECT * FROM Personali WHERE mobiluri IN ('593-277285', '555-123456', '599-012345');
```
4.6.8.
```
USE Shekveta;
SELECT * FROM Personali WHERE teleponi_saxlis IN ('221-11-00', '239-88-44', '234-79-92');
```
4.6.9.
```
USE Shekveta;
SELECT * FROM Shemkveti WHERE firmis_dasaxeleba IN ('Gia&Co.', 'Luka&Co.', 'Avto&Co.');
```

LIKE ოპერატორი

4.7.1.

```
USE Shekveta;
SELECT gvari, qalaqi
FROM Personali
WHERE gvari LIKE N'__ð%'
```

4.7.2.

```
USE Shekveta;
SELECT firmis_dasaxeleba, gvari, qalaqi
FROM Shemkveti
WHERE firmis_dasaxeleba LIKE N'G%';
```

4.7.3.

```
USE Shekveta;
SELECT *
FROM Shemkveti
WHERE qalaqi LIKE N'[ხრბ]%';
```

4.7.4.

```
USE Shekveta;
SELECT qalaqi, firmis_dasaxeleba, gvari
FROM Shemkveti
WHERE qalaqi LIKE N'[^რხ]%';
```

4.7.5.

```
USE Shekveta;
SELECT qalaqi, gvari, firmis_dasaxeleba
FROM Shemkveti
WHERE qalaqi LIKE N'[ი-ღ]%';
```

4.7.6.

```
USE Shekveta;
SELECT gvari, email, mobiluri
FROM Personali
WHERE email LIKE '%@geonet.ge';
```

4.7.7.

```
USE Shekveta;
SELECT *
FROM Shemkveti
WHERE mobiluri LIKE '574%';
```

4.7.8.

```
USE Shekveta;
SELECT *
FROM Shemkveti
WHERE teleponi_saxlis LIKE '234%';
```

4.7.9.

```
USE Shekveta;
SELECT *
FROM Personali
WHERE gvari LIKE N'%ძე%';
```

აგრეგირების ფუნქციები

4.8.1.

```
USE Shekveta;
SELECT MIN(vali_l) AS [შემკვეთების მინიმალური ვალი],
MAX(vali_l) AS [შემკვეთების მაქსიმალური ვალი]
FROM Xelshekruleba
WHERE vali_l > 0;
```

4.8.2.

```
USE Shekveta;
SELECT AVG(asaki) AS [საშუალო ასაკი]
FROM Personali;
```

4.8.3.

```
USE Shekveta;
SELECT COUNT(*) AS [შემკვეთების რაოდენობა]
FROM Shemkveti;
```

4.8.4.

```
USE Shekveta;
SELECT SUM(vali_l) AS [შემკვეთების მთლიანი ვალი]
 FROM Xelshekruleba;
```

4.8.5.

```
USE Shekveta;
SELECT MAX(vali_l) AS [შემკვეთების მაქსიმალური ვალი]
FROM Xelshekruleba;
```

4.8.6.

```
USE Shekveta;
SELECT MIN(vali_l) AS [შემკვეთების მინიმალური ვალი]
FROM Xelshekruleba
WHERE vali_l > 0;
```

4.8.12.

```
USE Shekveta;
SELECT COUNT(*)
FROM Xelshekruleba
WHERE shesruleba = N'კი' AND tarigi_damtavrebis < tarigi_shesrulebis;
```

4.8.13.

```
USE Shekveta;
SELECT COUNT(*)
FROM Xelshekruleba
WHERE shesruleba = N'კი' AND tarigi_damtavrebis > tarigi_shesrulebis;
```

4.8.14.

```
USE Shekveta;
SELECT COUNT(*)
FROM Xelshekruleba
WHERE shesruleba = N'კი' AND tarigi_damtavrebis = tarigi_shesrulebis;
```

4.8.15.

```
USE Shekveta;
```

```
SET DATEFORMAT DMY;
SELECT COUNT(*)
FROM Xelshekruleba
WHERE shesruleba = N'კი' AND tarigi_damtavrebis < tarigi_shesrulebis
AND tarigi_damtavrebis BETWEEN '01.01.2013' AND '31.12.2013';
```

4.8.16.

```
USE Shekveta;
SET DATEFORMAT DMY;
SELECT COUNT(*)
FROM Xelshekruleba
WHERE shesruleba = N'კი' AND tarigi_damtavrebis > tarigi_shesrulebis
AND tarigi_damtavrebis BETWEEN '01.01.2015' AND '31.12.2015';
```

4.8.17.

```
USE Shekveta;
SET DATEFORMAT DMY;
SELECT COUNT(*)
FROM Xelshekruleba
WHERE shesruleba = N'კი' AND tarigi_damtavrebis = tarigi_shesrulebis
AND tarigi_damtavrebis BETWEEN '01.01.2014' AND '31.12.2014';
```

4.8.42.

```
USE Shekveta;
SET DATEFORMAT DMY;
SELECT COUNT(P.personaliID)
FROM Personali P, Xelshekruleba X
WHERE P.personaliID = X.personaliID
    AND X.shesruleba = N'კი'
    AND X.tarigi_damtavrebis < X.tarigi_shesrulebis
    AND P.ganyofileba = N'სამედიცინო';
```

4.8.43.

```
USE Shekveta;
SET DATEFORMAT DMY;
SELECT COUNT(P.personaliID)
FROM Personali P, Xelshekruleba X
WHERE P.personaliID = X.personaliID
    AND X.shesruleba = N'კი'
    AND X.tarigi_damtavrebis > X.tarigi_shesrulebis
    AND P.ganyofileba = N'სამედიცინო';
```

4.8.44.

```
USE Shekveta;
SET DATEFORMAT DMY;
SELECT COUNT(P.personaliID)
FROM Personali P, Xelshekruleba X
WHERE P.personaliID = X.personaliID
    AND X.shesruleba = N'კი'
    AND X.tarigi_damtavrebis = X.tarigi_shesrulebis
    AND P.ganyofileba = N'სამედიცინო';
```

4.8.45.

```
USE Shekveta;
SET DATEFORMAT DMY;
DECLARE @ganyofileba NVARCHAR(15);
SET @ganyofileba = N'სავაჭრო';
SELECT COUNT(P.personaliID)
FROM Personali P, Xelshekruleba X
WHERE P.personaliID = X.personaliID
    AND X.shesruleba = N'კი'
    AND X.tarigi_damtavrebis < X.tarigi_shesrulebis
    AND P.ganyofileba = @ganyofileba;
```

4.8.50.

```
USE Shekveta;
SET DATEFORMAT DMY;
DECLARE @ganyofileba NVARCHAR(15);
SET @ganyofileba = N'სასოფლო';
SELECT COUNT(P.personaliID)
FROM Personali P, Xelshekruleba X
WHERE P.personaliID = X.personaliID
    AND X.shesruleba = N'კი'
    AND X.tarigi_damtavrebis > X.tarigi_shesrulebis
    AND P.ganyofileba = @ganyofileba;
```

4.8.51.

```
USE Shekveta;
SET DATEFORMAT DMY;
DECLARE @ganyofileba NVARCHAR(15);
SET @ganyofileba = N'სასოფლო';
SELECT COUNT(P.personaliID)
FROM Personali P, Xelshekruleba X
WHERE P.personaliID = X.personaliID
    AND X.shesruleba = N'კი'
    AND X.tarigi_damtavrebis = X.tarigi_shesrulebis
    AND P.ganyofileba = @ganyofileba;
```

4.8.70.

```
USE Shekveta_1;
SELECT MAX(xelfasi)
FROM Personali;
```

4.8.73.

```
USE Shekveta;
SELECT AVG(vali_l)
FROM Xelshekruleba;
```

4.8.74.

```
USE Shekveta;
SELECT MIN(gadasaxdeli_l)
FROM Xelshekruleba;
```

4.8.78.

```
USE Shekveta;
SELECT COUNT(*)
FROM Xelshekruleba;
4.8.79.
USE Shekveta;
SELECT COUNT(*)
FROM Personali
4.8.80.
USE Shekveta;
SELECT COUNT(*)
FROM Shemkveti
4.8.84.
USE Shekveta;
SELECT COUNT(*)
FROM Shemkveti
WHERE sqesi = N'კაცი' AND iuridiuli_fizikuri = N'ფიზიკური';
```

GROUP BY განყოფილება

```
4.9.1.
USE Shekveta;
SELECT personaliID, MIN(gadasaxdeli_l) AS [მინიმალური გადასახდელი თანხა]
FROM   Xelshekruleba
GROUP BY personaliID;
4.9.6.
USE Shekveta;
SELECT ganyofileba, AVG(xelfasi) AS [საშუალო ხელფასი]
FROM   Personali
GROUP BY ganyofileba;
4.9.10.
USE Shekveta;
SELECT ganyofileba, COUNT(*) AS [თანამშრომლების რაოდენობა]
FROM   Personali
GROUP BY ganyofileba;

4.9.12.
USE Shekveta;
SELECT ganyofileba, MAX(asaki) AS [თანამშრომლების მაქსიმალური ასაკი]
FROM   Personali
GROUP BY ganyofileba;
4.9.16.
USE Shekveta;
SELECT ganyofileba, AVG(staji) AS [თანამშრომლების საშუალო სტაჟი]
FROM   Personali
GROUP BY ganyofileba;
4.9.21.
USE Shekveta;
```

```
SELECT   ganyofileba, MIN(xelfasi) AS [მინიმალური ხელფასი]
FROM     Personali
WHERE  sqesi = N'ქალი'
GROUP BY ganyofileba;
```

4.9.24.

```
USE Shekveta;
SELECT   ganyofileba, MAX(staji) AS [მაქსიმალური სტაჟი]
FROM     Personali
WHERE  sqesi = N'ქალი'
GROUP BY ganyofileba;
```

4.9.27.

```
USE Shekveta;
SELECT   ganyofileba, MAX(asaki) AS [მაქსიმალური სტაჟი]
FROM     Personali
WHERE  sqesi = N'ქალი'
GROUP BY ganyofileba;
```

4.9.43.

```
USE Shekveta;
SELECT   qalaqi, COUNT(*) AS [კაცი ფიზიკური პირების რაოდენობა]
FROM     Shemkveti
WHERE  sqesi = N'კაცი' AND iuridiuli_fizikuri = N'ფიზიკური'
GROUP BY qalaqi;
```

4.9.44.

```
USE Shekveta;
SELECT  qalaqi, COUNT(*) AS [ფიზიკური პირების რაოდენობა]
FROM    Shemkveti
WHERE iuridiuli_fizikuri = N'ფიზიკური'
GROUP BY qalaqi;
```

4.9.48.

```
USE Shekveta;
SELECT qalaqi, MAX(DATEDIFF(year, tarigi_dabadebis, GETDATE()))
FROM    Shemkveti
GROUP BY qalaqi;
```

4.9.55.

```
USE Shekveta;
SELECT regioni, COUNT(*)
FROM    Shemkveti
WHERE iuridiuli_fizikuri = N'იურიდიული' AND regioni IS NOT NULL
GROUP BY regioni;
```

4.9.62.

```
USE Shekveta;
SELECT shemkvetiID, MAX(gadasaxdeli_d)
FROM   Xelshekruleba
GROUP BY shemkvetiID;
```

4.9.66.

```
USE Shekveta;
```

```
SELECT X.shemkvetiID, MAX(X.gadasaxdeli_d)
FROM    Shemkveti S, Xelshekruleba X
WHERE S.shemkvetiID = X.shemkvetiID
    AND S.iuridiuli_fizikuri = N'იურიდიული'
GROUP BY X.shemkvetiID;
```

4.9.70.

```
USE Shekveta;
SELECT X.shemkvetiID, MAX(X.gadasaxdeli_d)
FROM   Shemkveti S, Xelshekruleba X
WHERE S.shemkvetiID = X.shemkvetiID AND S.iuridiuli_fizikuri = N'ფიზიკური'
GROUP BY X.shemkvetiID;
```

4.9.73.

```
USE Shekveta;
SELECT   P.ganyofileba, COUNT(*)
FROM     Personali P, Xelshekruleba X
WHERE  P.personaliID = X.personaliID AND X.shesruleba = N'არა'
GROUP BY P.ganyofileba;
```

4.9.74.

```
USE Shekveta;
SELECT YEAR(tarigi_damtavrebis), COUNT(*)
FROM    Xelshekruleba
WHERE tarigi_damtavrebis < GETDATE() AND shesruleba = N'არა'
GROUP BY YEAR(tarigi_damtavrebis);
```

HAVING განყოფილება

4.10.1.

```
USE Shekveta;
SELECT personaliID, MIN(gadasaxdeli_l) AS [გადასახდელი თანხა]
FROM Xelshekruleba
GROUP BY personaliID
HAVING MIN(gadasaxdeli_l) < 5000;
```

4.10.4.

```
USE Shekveta;
SELECT ganyofileba, MIN(xelfasi) AS [მინიმალური ხელფასი]
FROM Personali
GROUP BY ganyofileba
HAVING MIN(xelfasi) > 1000;
```

4.10.5.

```
USE Shekveta;
SELECT ganyofileba, MAX(xelfasi) AS [მაქსიმალური ხელფასი]
FROM Personali
GROUP BY ganyofileba
HAVING MAX(xelfasi) < 5000;
```

4.10.6.

```
USE Shekveta;
```

```
SELECT ganyofileba, AVG(xelfasi) AS [საშუალო ხელფასი]
FROM Personali
GROUP BY ganyofileba
HAVING AVG(xelfasi) > 3000;
```

4.10.15.

```
USE Shekveta;
SELECT staji, COUNT(DISTINCT PersonaliID)
FROM Personali
GROUP BY staji
HAVING staji <
(
SELECT AVG(staji) FROM Personali WHERE qalaqi = N'ბათუმი'
);
```

ORDER BY განყოფილება

4.11.1.

```
USE Shekveta;
SELECT gvari, saxeli, asaki
FROM Personali
ORDER BY gvari, saxeli, asaki DESC;
```

4.11.2.

```
USE Shekveta;
SELECT gvari, saxeli, asaki
FROM Personali
ORDER BY gvari DESC, saxeli DESC, staji;
```

4.11.3.

```
USE Shekveta;
SELECT gvari, saxeli, asaki, staji
FROM Personali
ORDER BY gvari, saxeli, asaki DESC, staji DESC;
```

4.11.4.

```
USE Shekveta;
SELECT gvari, saxeli, qalaqi, ganyofileba, staji
FROM Personali
ORDER BY qalaqi, ganyofileba DESC, staji;
```

4.11.5.

```
USE Shekveta;
SELECT gvari, saxeli, qalaqi, raioni
FROM   Shemkveti
ORDER BY qalaqi, firmis_dasaxeleba, raioni DESC;
```

COMPUTE განყოფილება

4.12.1.

```
USE Shekveta;
SELECT personaliID, gadasaxdeli_l, vali_l, shesruleba
```

```
FROM   Xelshekruleba WHERE gadasaxdeli_l > 2000
ORDER BY personaliID
COMPUTE SUM(gadasaxdeli_l), COUNT(gadasaxdeli_l);
```

4.12.2.

```
USE Shekveta;
SELECT  personaliID, gadasaxdeli_l, vali_l, shesruleba
FROM    Xelshekruleba
WHERE gadasaxdeli_l > 2000
ORDER BY personaliID
COMPUTE SUM(gadasaxdeli_l), COUNT(gadasaxdeli_l) BY personaliID;
```

4.12.3.

```
USE Shekveta;
SET DATEFORMAT DMY;
SELECT  personaliID, gadasaxdeli_l, tarigi_dawyebis, shesruleba
FROM    Xelshekruleba
WHERE tarigi_dawyebis >= '01.01.2000' AND tarigi_dawyebis <= '01.01.2005'
ORDER BY personaliID
COMPUTE SUM(gadasaxdeli_l), COUNT(gadasaxdeli_l);
```

4.12.4.

```
USE Shekveta;
SET DATEFORMAT DMY;
SELECT  personaliID, gadasaxdeli_l, tarigi_dawyebis, shesruleba
FROM    Xelshekruleba
WHERE tarigi_dawyebis >= '01.01.2000' AND tarigi_dawyebis <= '01.01.2005'
ORDER BY personaliID
COMPUTE AVG(gadasaxdeli_l), MAX(gadasaxdeli_l), MIN(gadasaxdeli_l);
```

4.12.5.

```
USE Shekveta;
SET DATEFORMAT DMY;
SELECT personaliID, gadasaxdeli_l, tarigi_dawyebis, shesruleba
FROM   Xelshekruleba
WHERE tarigi_dawyebis >= '01.01.2000' AND tarigi_dawyebis <= '01.01.2005'
ORDER BY personaliID
COMPUTE AVG(gadasaxdeli_l), MAX(gadasaxdeli_l), MIN(gadasaxdeli_l) BY personaliID;
```

4.12.6.

```
USE Shekveta;
SELECT personaliID, gvari, xelfasi
FROM   Personali
ORDER BY gvari
COMPUTE AVG(xelfasi), MAX(xelfasi), MIN(xelfasi);
```

4.12.7.

```
USE Shekveta;
SELECT gvari, ganyofileba, xelfasi
FROM   Personali
ORDER BY ganyofileba
COMPUTE AVG(xelfasi), MAX(xelfasi), MIN(xelfasi) BY ganyofileba;
```

4.12.8.
```
USE Shekveta;
SELECT gvari, ganyofileba, xelfasi, asaki
FROM   Personali
ORDER BY ganyofileba
COMPUTE AVG(asaki), MAX(asaki), MIN(asaki);
```
4.12.9.
```
USE Shekveta;
SELECT gvari, ganyofileba, xelfasi, asaki, staji
FROM   Personali
ORDER BY ganyofileba
COMPUTE AVG(staji), MAX(staji), MIN(staji);
```
4.12.10.
```
USE Shekveta;
SELECT gvari, ganyofileba, xelfasi, asaki, staji
FROM   Personali
ORDER BY ganyofileba
COMPUTE AVG(staji), MAX(staji), MIN(staji) BY ganyofileba;
```
4.12.11.
```
USE Shekveta;
SELECT vali_l, shesruleba
FROM   Xelshekruleba WHERE vali_l > 0
ORDER BY vali_l DESC
COMPUTE MAX(vali_l), MIN(vali_l);
```

OVER ელემენტი

4.13.1.
```
USE Shekveta;
SELECT xelshekrulebaID, personaliID, shemkvetiID, vali_l,
        SUM(vali_l) OVER() AS saertovali_l
FROM Xelshekruleba;
```
4.13.3.
```
USE Shekveta;
SELECT xelshekrulebaID, personaliID, gadasaxdeli_l,
        SUM(gadasaxdeli_l) OVER() AS saertotanxa,
        SUM(gadasaxdeli_l) OVER(PARTITION BY personaliID) AS personalisaertotanxa
FROM Xelshekruleba;
```
4.13.5.
```
USE Shekveta;
SELECT xelshekrulebaID, personaliID, shemkvetiID, vali_l,
        SUM(vali_l) OVER(PARTITION BY personaliID) AS personalivali_l
FROM Xelshekruleba;
```
4.13.6.
```
USE Shekveta;
SELECT xelshekrulebaID, personaliID, shemkvetiID, vali_l,
        SUM(vali_l) OVER(PARTITION BY shemkvetiID) AS shemkvetivali_l
FROM Xelshekruleba;
```

4.13.9.

```
USE Shekveta;
SELECT xelshekrulebaID, personaliID, shemkvetiID, vali_l,
        100. * vali_l / SUM(vali_l) OVER() AS procentimtlianivali
FROM Xelshekruleba
WHERE vali_l > 0;
```

4.13.10.

```
USE Shekveta;
SELECT xelshekrulebaID, personaliID, shemkvetiID, vali_l,
        100. * vali_l / SUM(vali_l) OVER(PARTITION BY personaliID) AS procentipersonali
FROM Xelshekruleba
WHERE vali_l > 0;
```

4.13.11.

```
USE Shekveta;
SELECT xelshekrulebaID, personaliID, shemkvetiID, vali_l,
        100. * vali_l / SUM(vali_l) OVER() AS procentisaertovali,
        100. * vali_l / SUM(vali_l) OVER(PARTITION BY shemkvetiID) AS procentipersonalivali
FROM Xelshekruleba
WHERE vali_l > 0;
```

4.13.15.

```
USE Shekveta;
SELECT X.xelshekrulebaID, X.personaliID, X.shemkvetiID, X.gadasaxdeli_l,
        100. * X.gadasaxdeli_l / SUM(X.gadasaxdeli_l) OVER() AS mtlianigadasaxdeli_l,
        100. * X.gadasaxdeli_l / SUM(X.gadasaxdeli_l) OVER(PARTITION BY shemkvetiID) AS
         personaligadasaxdeli_l
FROM    Personali P, Xelshekruleba X
WHERE P.ganyofileba = N'სამედიცინო' AND
        P.personaliID = X.personaliID;
```

4.13.16.

```
USE Shekveta;
SELECT X.xelshekrulebaID, X.personaliID, X.shemkvetiID, X.gadasaxdeli_l,
        100. * X.gadasaxdeli_l / SUM(X.gadasaxdeli_l) OVER() AS mtlianigadasaxdeli_l
FROM    Personali P, Xelshekruleba X
WHERE P.ganyofileba = N'სასპორტო' AND
        P.personaliID = X.personaliID;
```

4.13.17.

```
USE Shekveta;
SELECT X.xelshekrulebaID, X.personaliID, X.shemkvetiID, X.gadasaxdeli_l,
        100. * X.gadasaxdeli_l / SUM(X.gadasaxdeli_l) OVER(PARTITION BY shemkvetiID) AS
         personaligadasaxdeli_l
FROM    Personali P, Xelshekruleba X
WHERE P.ganyofileba = N'სასოფლო' AND
        P.personaliID = X.personaliID;
```

NULL მნიშვნელობასთან მუშაობა

4.14.1.

```
USE Shekveta;
SELECT  personaliID, regioni, qalaqi, raioni
FROM     Personali
WHERE  regioni <> N'ქართლი' OR regioni IS NULL;
```

4.14.2.

```
USE Shekveta;
SELECT   personaliID, regioni, qalaqi, raioni
FROM     Personali
WHERE  regioni <> N'აჭარა' OR regioni IS NULL;
```

4.14.3.

```
USE Shekveta;
SELECT  personaliID, regioni, qalaqi, raioni
FROM    Personali
WHERE  raioni <> N'ვერა' OR raioni IS NULL;
```

4.14.4.

```
USE Shekveta;
SELECT  personaliID, regioni, qalaqi, raioni
FROM    Personali
WHERE  raioni <> N'ვაკე' OR raioni IS NULL;
```

4.14.5.

```
USE Shekveta;
SELECT   shemkvetiID, regioni, qalaqi, raioni
FROM     Shemkveti
WHERE  raioni <> N'საბურთალო' OR raioni IS NULL;
```

4.14.6.

```
USE Shekveta;
SELECT   shemkvetiID, regioni, qalaqi, raioni
FROM     Shemkveti
WHERE  raioni <> N'დიდუბე' OR raioni IS NULL;
```

4.14.7.

```
SELECT   iuridiuli_fizikuri, firmis_dasaxeleba
FROM     Shemkveti
WHERE  firmis_dasaxeleba IS NOT NULL;
```

4.14.8.

```
SELECT   iuridiuli_fizikuri, firmis_dasaxeleba
FROM     Shemkveti
WHERE  firmis_dasaxeleba IS NULL;
```

UPDATE ბრძანება

4.15.1.

```
USE Shekveta;
UPDATE Personali SET xelfasi = xelfasi + 200;
SELECT *
```

```
FROM Personali;
```
4.15.2.
```
USE Shekveta;
UPDATE Xelshekruleba SET gadasaxdeli_l = 8000
WHERE xelshekrulebaID = 3 OR xelshekrulebaID = 9;
SELECT *
FROM Xelshekruleba;
```
4.15.3.
```
USE Shekveta;
UPDATE Xelshekruleba SET gadasaxdeli_d = ROUND(gadasaxdeli_l / kursi, 2),
                                         vali_d = ROUND(vali_l / kursi, 2);
SELECT *
FROM Xelshekruleba;
```

DELETE ბრძანება

4.16.1.
```
USE Shekveta;
SELECT * INTO Xelshekruleba_1
FROM Xelshekruleba;
DELETE FROM Xelshekruleba_1  WHERE vali_l = 0;
SELECT *
FROM Xelshekruleba_1;
```
4.16.2.
```
USE Shekveta;
DELETE FROM Xelshekruleba_1;
SELECT *
FROM Xelshekruleba_1;
```

მასობრივი გადაწერა

4.17.1.
```
bcp "Shekveta.dbo.Shemkveti" out "C:\Documents and Settings\romani\My Documents\SQL Server Management Studio\Shem_1.txt" -w -T
```
4.17.2.
```
bcp "SELECT * FROM Shekveta.dbo.Personali WHERE staji>20" queryout "C:\Documents and Settings\romani\My Documents\SQL Server Management Studio\Personali4.txt" -w -T
```
4.17.3.
```
bcp "Shekveta.dbo.Shemkveti_1" in "C:\Documents and Settings\romani\My Documents\SQL Server Management Studio\Shem_1.txt" -w –S"Serveri-pc" –U"sa" –P"paroli"
```
4.17.4.
```
bcp "Shekveta.dbo.Shemkveti" out "C:\Documents and Settings\romani\My Documents\SQL Server Management Studio\Shem_1.txt" –F3 –L6 –w –S"Serveri-pc" –U"sa" –P"paroli"
```
4.17.5.
```
bcp "Shekveta.dbo.Shemkveti" in "C:\Documents and Settings\romani\My Documents\SQL Server Management Studio\Shem_1.txt" –F2 –L5 –w –S"Serveri-pc" –U"sa" –P"paroli"
```

თავი 5. შეერთებები

ჯვარედინი შეერთებები

5.1.1.

```
USE Shekveta;
SELECT S.gvari AS [შემკვეთის გვარი], X.gadasaxdeli_l AS [გადასახდელი თანხა]
FROM Shemkveti S CROSS JOIN Xelshekruleba X
ORDER BY S.gvari, X.gadasaxdeli_l;
```

5.1.2.

```
USE Shekveta;
SELECT P1.shemkvetiID, P1.gvari, P2.shemkvetiID, P2.gvari
FROM   Shemkveti AS P1 CROSS JOIN Shemkveti AS P2;
```

შიდა შეერთებები

5.2.1.

```
USE Shekveta;
SELECT Shemkveti.gvari, Shemkveti.qalaqi, Xelshekruleba.gadasaxdeli_l, Xelshekruleba.vali_l
FROM   Shemkveti INNER JOIN Xelshekruleba
          ON Shemkveti.shemkvetiID = Xelshekruleba.shemkvetiID;
```

5.2.2.

```
USE Shekveta;
SELECT Shemkveti.gvari, Shemkveti.saxeli, Shemkveti.iuridiuli_fizikuri, Xelshekruleba.vali_l,
           Xelshekruleba.gadasaxdeli_l, Xelshekruleba.gadaxdili_l
FROM    Shemkveti INNER JOIN Xelshekruleba
           ON Shemkveti.shemkvetiID = Xelshekruleba.shemkvetiID
WHERE Xelshekruleba.vali_l > 0;
```

5.2.3.

```
USE Shekveta;
SELECT S.gvari, S.saxeli, S.iuridiuli_fizikuri, X.*
FROM   Shemkveti S INNER JOIN Xelshekruleba X
            ON S.shemkvetiID = X.shemkvetiID
WHERE  S.iuridiuli_fizikuri = N'ფიზიკური'
ORDER BY S.gvari, S.saxeli;
```

5.2.5.

```
USE Shekveta;
SELECT   S.gvari, S.saxeli, S.qalaqi, X.*
FROM     Shemkveti S INNER JOIN Xelshekruleba X
            ON S.shemkvetiID = X.shemkvetiID
WHERE  S.qalaqi = N'თბილისი'
ORDER BY S.gvari, S.saxeli;
```

5.2.6.

```
USE Shekveta;
SELECT  P.gvari, P.saxeli, P.qalaqi, X.*
FROM     Personali P INNER JOIN Xelshekruleba X
            ON P.personaliID = X.personaliID
WHERE  P.qalaqi = N'ბათუმი'
```

```
ORDER BY P.gvari, P.saxeli;
```
5.2.7.
```
USE Shekveta;
SELECT   P.gvari, P.saxeli, P.ganyofileba, X.*
FROM     Personali P INNER JOIN Xelshekruleba X
         ON P.personaliID = X.personaliID
WHERE  P.ganyofileba = N'სამედიცინო'
ORDER BY P.gvari, P.saxeli;
```
5.2.8.
```
USE Shekveta;
SELECT  COUNT(*)
FROM     Personali P INNER JOIN Xelshekruleba X
         ON P.personaliID = X.personaliID
WHERE  P.ganyofileba = N'სასპორტო';
```
5.2.9.
```
USE Shekveta;
SELECT  MAX(X.gadasaxdeli_l)
FROM     Personali P INNER JOIN Xelshekruleba X
         ON P.personaliID = X.personaliID
WHERE  P.qalaqi = N'ბათუმი';
```
5.2.10.
```
USE Shekveta;
SELECT   SUM(X.gadasaxdeli_l)
FROM     Shemkveti S INNER JOIN Xelshekruleba X
         ON S.shemkvetiID = X.shemkvetiID
WHERE  S.qalaqi = N'თბილისი';
```

გარე შეერთებები

5.3.1.
```
USE Shekveta;
SELECT Shemkveti.gvari, Shemkveti.qalaqi, Xelshekruleba.gadasaxdeli_l, Xelshekruleba.vali_l
FROM   Shemkveti LEFT OUTER JOIN Xelshekruleba
       ON Shemkveti.shemkvetiID = Xelshekruleba.shemkvetiID;
```
5.3.2.
```
USE Shekveta;
SELECT Shemkveti.gvari, Shemkveti.qalaqi, Xelshekruleba.gadasaxdeli_l, Xelshekruleba.vali_l
FROM   Shemkveti RIGHT OUTER JOIN Xelshekruleba
       ON Shemkveti.shemkvetiID = Xelshekruleba.shemkvetiID;
```
5.3.3.
```
USE Shekveta;
SELECT Shemkveti.gvari, Shemkveti.qalaqi, Xelshekruleba.gadasaxdeli_l, Xelshekruleba.vali_l
FROM   Shemkveti FULL OUTER JOIN Xelshekruleba
       ON Shemkveti.ShemkvetiID = Xelshekruleba.ShemkvetiID;
```
5.3.4.
```
USE Shekveta;
SELECT Personali.gvari, Personali.saxeli, Personali.qalaqi,
```

```
        Xelshekruleba.gadasaxdeli_l, Xelshekruleba.vali_l
FROM  Personali LEFT OUTER JOIN Xelshekruleba
      ON Personali.personaliID = Xelshekruleba.personaliID;
```

5.3.5.

```
USE Shekveta;
SELECT Personali.gvari, Personali.saxeli, Personali.qalaqi,
        Xelshekruleba.gadasaxdeli_l, Xelshekruleba.vali_l
FROM   Personali RIGHT OUTER JOIN Xelshekruleba
       ON Personali.personaliID = Xelshekruleba.personaliID;
```

5.3.6.

```
USE Shekveta;
SELECT Shemkveti.gvari, Shemkveti.saxeli, Shemkveti.qalaqi,
        Xelshekruleba.gadasaxdeli_l, Xelshekruleba.vali_l
FROM    Shemkveti FULL OUTER JOIN Xelshekruleba
        ON Shemkveti.personaliID = Xelshekruleba.personaliID;
```

თავი 6. Transact SQL-ის საფუძვლები

IF...ELSE

6.1.1.

```
USE Shekveta;
DECLARE @asaki INT;
SET @asaki =
(
SELECT asaki
FROM   Personali
WHERE gvari = N'სამხარაძე' AND saxeli = N'საბა'
);
SELECT @asaki;
IF  @asaki > 30
   SELECT N'თანამშრომლის ასაკი 30 წელს აღემატება'
ELSE
   SELECT N'თანამშრომლის ასაკი 30 წელს არ აღემატება';
```

6.1.2.

```
USE Shekveta;
DECLARE @staji INT;
SET @staji =
(
SELECT staji
FROM   Personali
WHERE gvari = N'სამხარაძე' AND saxeli = N'საბა'
);
SELECT @staji;
IF  @staji > 15
   SELECT N'თანამშრომლის სტაჟი 15 წელს აღემატება'
ELSE
```

```
    SELECT N'თანამშრომლის სტაჟი 15 წელს აღემატება';
```
6.1.3.
```
USE Shekveta;
DECLARE @xelfasi FLOAT;
SET @xelfasi =
(
SELECT xelfasi
FROM   Personali
WHERE gvari = N'სამხარაძე' AND saxeli = N'საბა'
);
SELECT @xelfasi;
IF  @xelfasi > 15
    SELECT N'თანამშრომლის ხელფასი 1500 ლარს აღემატება'
ELSE
    SELECT N'თანამშრომლის ხელფასი 1500 ლარს აღემატება';
```
6.1.4.
```
USE Shekveta;
SET DATEFORMAT DMY;
DECLARE @dabadebis_tarigi DATE;
SET @dabadebis_tarigi =
(
SELECT tarigi_dabadebis
FROM    Personali
WHERE gvari = N'სამხარაძე' AND saxeli = N'საბა'
);
SELECT @dabadebis_tarigi;
IF  @dabadebis_tarigi = '19.03.1985'
    SELECT N'თანამშრომლის დაბადების თარიღია:  ', @dabadebis_tarigi
ELSE
    SELECT N'თანამშრომლის დაბადების თარიღი არაა: ', @dabadebis_tarigi;
```
6.1.5.
```
USE Shekveta;
SET DATEFORMAT DMY;
DECLARE @raioni NVARCHAR(20);
SET @raioni =
(
SELECT raioni
FROM   Personali
WHERE gvari = N'სამხარაძე' AND saxeli = N'საბა'
)
SELECT @raioni;
IF  @raioni = N'დიდუბე'
    SELECT N'თანამშრომელი ცხოვრობს დიდუბის რაიონში:  ', @raioni
ELSE
    SELECT N'თანამშრომელი არ ცხოვრობს დიდუბის რაიონში: ', @raioni;
```
6.1.6.

```
USE Shekveta;
SET DATEFORMAT DMY;
DECLARE @ganyofileba NVARCHAR(20);
SET @ganyofileba =
(
SELECT ganyofileba
FROM   Personali
WHERE gvari = N'სამხარაძე' AND saxeli = N'საბა'
);
SELECT @ganyofileba;
IF  @ganyofileba = N'სამედიცინო'
    SELECT N'თანამშრომელი მუშაობს სამედიცინო განყოფილებაში:  ', @ganyofileba
ELSE
    SELECT N'თანამშრომელი არ მუშაობს სამედიცინო განყოფილებაში: ', @ganyofileba;
```

6.1.7.

```
USE Shekveta;
SET DATEFORMAT DMY;
DECLARE @mobiluri NVARCHAR(12), @gvari NVARCHAR(20), @saxeli NVARCHAR(15);
SET @gvari = N'სამხარაძე';
SET @saxeli = N'საბა';
SET @mobiluri =
(
SELECT mobiluri
FROM   Personali
WHERE gvari = @gvari AND saxeli = @saxeli
);
SELECT @mobiluri;
IF  @mobiluri = N'893-277285'
    SELECT @gvari + N'ს მობილურია:  ', @mobiluri
ELSE
    SELECT N'თანამშრომლის მობილური არაა: ', @mobiluri;
```

6.1.8.

```
USE Shekveta;
SET DATEFORMAT DMY;
DECLARE @qalaqi NVARCHAR(12), @gvari NVARCHAR(20), @saxeli NVARCHAR(15);
SET @gvari = N'სამხარაძე';
SET @saxeli = N'საბა';
SET @qalaqi =
(
SELECT qalaqi
FROM Personali
WHERE gvari = @gvari AND saxeli = @saxeli
);
SELECT @qalaqi;
IF  @qalaqi = N'თბილისი'
    SELECT @gvari + N' ცხოვრობს ' + @qalaqi + N' ქალაქში'
```

```
ELSE
   SELECT @gvari + N' არ ცხოვრობს მითითებულ ქალაქში';
```
6.1.9.
```
USE Shekveta;
SET DATEFORMAT DMY;
DECLARE @iuridiuli_fizikuri NVARCHAR(12), @gvari NVARCHAR(20),
            @saxeli NVARCHAR(15);
SET @gvari = N'სამხარაძე';
SET @saxeli = N'არჩილი';
SET @iuridiuli_fizikuri =
(
SELECT iuridiuli_fizikuri
FROM   Shemkveti
WHERE gvari = @gvari AND saxeli = @saxeli
);
SELECT @iuridiuli_fizikuri;
IF  @iuridiuli_fizikuri = N'ფიზიკური'
   SELECT @gvari + N' არის ' + @iuridiuli_fizikuri + N' პირი'
ELSE
   SELECT @gvari + N' არის იურიდიული პირი';
```
6.1.10.
```
USE Shekveta;
SET DATEFORMAT DMY;
DECLARE @firmis_dasaxeleba NVARCHAR(12), @gvari NVARCHAR(20),
            @saxeli NVARCHAR(15);
SET @gvari = N'სეხნიაშვილი';
SET @saxeli = N'გია';
SET @firmis_dasaxeleba =
(
SELECT firmis_dasaxeleba
FROM   Shemkveti
WHERE gvari = @gvari AND saxeli = @saxeli
);
SELECT @firmis_dasaxeleba;
IF  @firmis_dasaxeleba = N'Gia&Co.'
   SELECT @gvari + N' მუშაობს ' + @firmis_dasaxeleba + N' ფირმაში'
ELSE
   SELECT @gvari + N' არ მუშაობს მითითებულ ფირმაში';
```

CASE...END

6.2.1.
```
USE Shekveta;
DECLARE @raioni NVARCHAR(20), @shedegi NVARCHAR(50);
SET @raioni =
(
SELECT  raioni
```

```
FROM    Personali
WHERE gvari = N'სამხარაძე' AND saxeli = N'საბა'
);
SELECT @raioni;
SET @shedegi =
CASE @raioni
WHEN N'დიდუბე'    THEN N'თანამშრომელი ცხოვრობს დიდუბის რაიონში'
WHEN N'ვაკე'      THEN N'თანამშრომელი ცხოვრობს ვაკის რაიონში'
WHEN N'ვერა'      THEN N'თანამშრომელი ცხოვრობს ვერის რაიონში'
WHEN N'ჩუღურეთი' THEN N'თანამშრომელი ცხოვრობს ჩუღურეთის რაიონში'
ELSE    N'რაიონი უცნობია'
END;
SELECT @shedegi AS [შედეგი];
```

6.2.2.

```
USE Shekveta;
DECLARE @qalaqi NVARCHAR(20), @shedegi NVARCHAR(50);
SET @qalaqi =
(
SELECT  qalaqi
FROM    Personali
WHERE gvari = N'სამხარაძე' AND saxeli = N'საბა'
);
SELECT @qalaqi;
SET       @shedegi =
CASE    @qalaqi
WHEN N'თბილისი'    THEN N'თანამშრომელი ცხოვრობს თბილისში'
WHEN N'ბათუმი'     THEN N'თანამშრომელი ცხოვრობს ბათუმში'
WHEN N'ქუთაისი'    THEN N'თანამშრომელი ცხოვრობს ქუთაისში'
WHEN N'ქობულეთი'  THEN N'თანამშრომელი ცხოვრობს ქობულეთში'
ELSE    N'ქალაქი უცნობია'
END;
SELECT @shedegi AS [შედეგი];
```

6.2.3.

```
USE Shekveta;
DECLARE @tarigi_dabadebis DATETIME;
SET @tarigi_dabadebis  =
(
SELECT tarigi_dabadebis
FROM   Personali
WHERE gvari = N'სამხარაძე' AND saxeli = N'ანა'
);
SELECT @tarigi_dabadebis;
SELECT gvari, saxeli,
CASE    MONTH(@tarigi_dabadebis)
   WHEN  1 THEN N'იანვარი'
   WHEN  2 THEN N'თებერვალი'
```

```
   WHEN  3 THEN N'მარტი'
   WHEN  4 THEN N'აპრილი'
   WHEN  5 THEN N'მაისი'
   WHEN  6 THEN N'ივნისი'
   WHEN  7 THEN N'ივლისი'
   WHEN  8 THEN N'აგვისტო'
   WHEN  9 THEN N'სექტემბერი'
   WHEN 10 THEN N'ოქტომბერი'
   WHEN 11 THEN N'ნოემბერი'
   WHEN 12 THEN N'დეკემბერი'
   ELSE N''
END
FROM Personali
ORDER BY gvari;
```

6.2.4.

```
USE Shekveta;
SELECT gvari, saxeli,
CASE    MONTH(tarigi_dabadebis)
   WHEN  1 THEN N'დაიბადა იანვარში'
   WHEN  2 THEN N'თებერვალი'
   WHEN  3 THEN N'მარტი'
   WHEN  4 THEN N'იანვარი'
   WHEN  5 THEN N'თებერვალი'
   WHEN  6 THEN N'მარტი'
   WHEN  7 THEN N'იანვარი'
   WHEN  8 THEN N'თებერვალი'
   WHEN  9 THEN N'მარტი'
   WHEN 10 THEN N'იანვარი'
   WHEN 11 THEN N'თებერვალი'
   WHEN 12 THEN N'მარტი'
   ELSE N'უცნობი'
END
FROM Personali;
```

6.2.5.

```
USE Shekveta;
SELECT gvari, saxeli,
CASE     YEAR(tarigi_dabadebis)
   WHEN  1980 THEN N'დაიბადა 1980 წელს'
   WHEN  1985 THEN N'დაიბადა 1985 წელს'
   WHEN  1990 THEN N'დაიბადა 1990 წელს'
   WHEN  1995 THEN N'დაიბადა 1995 წელს'
   WHEN  2000 THEN N'დაიბადა 2000 წელს'
   WHEN  2005 THEN N'დაიბადა 2005 წელს'
   WHEN  1975 THEN N'დაიბადა 1975 წელს'
   WHEN  1970 THEN N'დაიბადა 1970 წელს'
   WHEN  1965 THEN N'დაიბადა 1965 წელს'
```

```
   WHEN  1960 THEN N'დაიბადა 1960 წელს'
   WHEN  1955 THEN N'დაიბადა 1955 წელს'
   WHEN  1950 THEN N'დაიბადა 1950 წელს'
   ELSE ''
END
FROM Personali
ORDER BY gvari;
```

6.2.6.

```
USE Shekveta;
SELECT gvari, saxeli,
CASE    ganyofileba
   WHEN  N'სამედიცინო' THEN N'მუშაობს სამედიცინო განყოფილებაში'
   WHEN  N'სავაჭრო'       THEN N'სავაჭრო'
   WHEN  N'სასპორტო'    THEN N'სასპორტო'
   WHEN  N'სასოფლო'     THEN N'სასოფლო'
   ELSE ''
END
FROM Personali
ORDER BY ganyofileba, gvari, saxeli;
```

6.2.7.

```
USE Shekveta;
SELECT gvari,
CASE    raioni
   WHEN  N'დიდუბე'        THEN misamarti
   WHEN  N'ვაკე'              THEN misamarti
   WHEN  N'საბურთალო' THEN misamarti
   WHEN  N'ვერა'              THEN misamarti
   WHEN  N'ჩუღურეთი'   THEN misamarti
   ELSE ''
END
FROM Personali
ORDER BY raioni, gvari, saxeli;
```

6.2.8.

```
USE Shekveta;
SELECT gvari,
CASE    asaki
   WHEN  20 THEN staji
   WHEN  46 THEN staji
   WHEN  26 THEN staji
   WHEN  50 THEN staji
   WHEN  22 THEN staji
   WHEN  57 THEN staji
   WHEN  45 THEN staji
   ELSE 0
END
FROM Personali
```

```
ORDER BY gvari;
```
6.2.9.
```
USE Shekveta;
SELECT gvari, saxeli,
CASE     raioni
   WHEN  N'დიდუბე'          THEN teleponi_saxlis
   WHEN  N'ვაკე'            THEN teleponi_saxlis
   WHEN  N'საბურთალო'       THEN teleponi_saxlis
   WHEN  N'ვერა'            THEN teleponi_saxlis
   WHEN  N'ჩუღურეთი'        THEN teleponi_saxlis
   ELSE ''
END
FROM Personali
ORDER BY gvari, saxeli;
```
6.2.10.
```
USE Shekveta;
SELECT gvari, saxeli, qalaqi,
CASE     qalaqi
   WHEN  N'თბილისი'    THEN misamarti
   WHEN  N'ბათუმი'     THEN misamarti
   WHEN  N'ქუთაისი'    THEN misamarti
   WHEN  N'ზუგდიდი'    THEN misamarti
   WHEN  N'რუსთავი'    THEN misamarti
   WHEN  N'ქობულეთი' THEN misamarti
   ELSE ''
END
FROM Personali
ORDER BY qalaqi, gvari;
```

WHILE...BREAK & CONTINUE

6.3.1.
```
USE Shekveta;
WHILE
(
SELECT AVG(xelfasi)
FROM   Personali
) < 3000
UPDATE Personali SET xelfasi = xelfasi * 1.1
SELECT * FROM Personali;
```
6.3.2.
```
USE Shekveta;
WHILE
(
SELECT MAX(xelfasi)
FROM Personali
) < 200
```

```
UPDATE Personali SET xelfasi = xelfasi * 1.15;
SELECT * FROM Personali;
```

6.3.3.

```
USE Shekveta;
WHILE
(
SELECT MIN(xelfasi)
FROM Personali
) < 300
UPDATE Personali SET xelfasi = xelfasi * 1.15
SELECT * FROM Personali
```

6.3.4.

```
USE Shekveta;
WHILE
(
SELECT SUM(xelfasi)
FROM Personali
) < 6000
UPDATE Personali_1 SET xelfasi = xelfasi * 1.10;
SELECT * FROM Personali;
```

6.3.5.

```
USE Shekveta;
DECLARE @sashualo_xelfasi FLOAT;
WHILE 1 = 1
BEGIN
UPDATE Personali SET xelfasi = xelfasi * 1.1
SET @sashualo_xelfasi =
(
SELECT AVG(xelfasi)
FROM Personali
)
IF    @sashualo_xelfasi < 50 CONTINUE
 ELSE BREAK
END
SELECT * FROM Personali;
```

თავი 7. ქვემოთხოვნები

ჩადგმული მოთხოვნები

7.1.1.

```
USE Shekveta;
SELECT  *
FROM    Xelshekruleba
WHERE personaliID =
(
SELECT  personaliID
```

```
FROM    Personali
WHERE gvari = N'სამხარაძე' AND saxeli = N'საბა'
);
```
7.1.2.
```
USE Shekveta;
SELECT  *
FROM    Personali
WHERE qalaqi =
(
SELECT  qalaqi
FROM    Personali
WHERE gvari = N'სამხარაძე' AND saxeli = N'საბა'
);
```
7.1.3.
```
USE Shekveta;
SELECT xelshekrulebaID, personaliID, shemkvetiID, tarigi_dawyebis, vali_l
FROM Xelshekruleba
WHERE vali_l =
(
SELECT MIN(X.vali_l)
FROM    Xelshekruleba AS X
WHERE X.vali_l > 0
);
```
7.1.4.
```
USE Shekveta;
SELECT   *
FROM     Xelshekruleba
WHERE  personaliID IN
(
SELECT   P.personaliID
FROM     Personali AS P
WHERE  P.qalaqi = N'თბილისი'
);
```
7.1.5.
```
SELECT *
FROM     Xelshekruleba
WHERE  personaliID NOT IN
(
SELECT   P.personaliID
FROM     Personali AS P
WHERE  P.qalaqi = N'თბილისი'
);
```
7.1.6.
```
USE Shekveta;
SELECT *
FROM     Xelshekruleba
```

```
WHERE  shemkvetiID IN
(
SELECT   shemkvetiID
FROM     Shemkveti
WHERE  qalaqi = N'თბილისი'
);
```

7.1.7.

```
USE Shekveta;
SELECT   *
FROM     Xelshekruleba
WHERE  shemkvetiID NOT IN
(
SELECT   shemkvetiID
FROM     Shemkveti
WHERE  qalaqi = N'თბილისი'
);
```

7.1.10.

```
USE Shekveta;
SELECT   *
FROM     Xelshekruleba
WHERE  shemkvetiID =
(
SELECT   shemkvetiID
FROM     Shemkveti
WHERE  gvari = N'სეხნიაშვილი' AND saxeli = N'გია'
);
```

7.1.11.

```
USE Shekveta;
SELECT   *
FROM     Shemkveti
WHERE  qalaqi =
(
SELECT   qalaqi
FROM     Shemkveti
WHERE  gvari = N'სეხნიაშვილი' AND saxeli = N'გია'
);
```

7.1.13.

```
USE Shekveta;
SELECT * FROM Xelshekruleba
WHERE  shemkvetiID IN
(
SELECT shemkvetiID
FROM   Shemkveti
WHERE firmis_dasaxeleba = N'Gia&Co.'
);
```

7.2.1.

```
USE Shekveta;
SET DATEFORMAT DMY;
SELECT   *
FROM     Personali P
WHERE  '01.01.2011' IN
(
SELECT   X.tarigi_dawyebis
FROM     Xelshekruleba X
WHERE  X.personaliID = P.personaliID
);
```

7.2.2.

```
USE Shekveta;
SELECT  personaliID, gvari, saxeli
FROM    Personali P
WHERE 2 >
(
SELECT  COUNT(*)
FROM     Xelshekruleba X
WHERE  X.personaliID = P.personaliID
GROUP BY X.personaliID
);
```

7.2.4.

```
USE Shekveta;
SELECT  shemkvetiID, gvari, saxeli
FROM    Shemkveti S
WHERE 2 >
(
SELECT   COUNT(*)
FROM     Xelshekruleba X
WHERE  X.shemkvetiID = S.shemkvetiID
GROUP BY X.shemkvetiID
);
```

7.2.5.

```
USE Shekveta;
SELECT  shemkvetiID, gvari, saxeli
FROM    Shemkveti S
WHERE 2 <
(
SELECT   COUNT(*)
FROM     Xelshekruleba X
WHERE  X.shemkvetiID = S.shemkvetiID
GROUP BY X.shemkvetiID
);
```

7.2.6.

```
USE Shekveta;
SET DATEFORMAT DMY;
SELECT   *
FROM     Shemkveti S
WHERE  '01.01.2011' IN
(
SELECT   X.tarigi_dawyebis
FROM     Xelshekruleba X
WHERE  X.shemkvetiID = S.shemkvetiID
);
```

ALL ოპერატორი

7.3.1.

```
USE Shekveta;
IF 30 < ALL
(
SELECT asaki
FROM Personali
)
        SELECT N'ყველა თანამშრომლის ასაკი აღემატება 30 წელს'
ELSE
        SELECT N'ყველა თანამშრომლის ასაკი არ აღემატება 30 წელს';
```

7.3.2.

```
USE Shekveta;
IF N'კი' = ALL
(
SELECT shesruleba
FROM Xelshekruleba
)
        SELECT N'ყველა ხელსშეკრულება შესრულდა'
ELSE
        SELECT N'ყველა ხელშეკრულება არ შესრულდა';
SELECT xelshekrulebaID [არ შესრულდა ხელსშეკრულებები]
FROM Xelshekruleba
WHERE shesruleba = N'არა';
```

7.3.5.

```
USE Shekveta;
DECLARE @sashualo_asaki FLOAT;
SET @sashualo_asaki =
(
SELECT AVG(asaki)
FROM Personali
);
SELECT @sashualo_asaki;
IF @sashualo_asaki < ALL
```

```
(
SELECT asaki
FROM Personali
)
        SELECT N'ყველა თანამშრომლის ასაკი აღემატება საშუალო ასაკს'
ELSE
        SELECT N'ყველა თანამშრომლის ასაკი არ აღემატება საშუალო ასაკს';
```

7.3.8.

```
USE Shekveta;
IF 5000 < ALL
(
SELECT gadasaxdeli_l
FROM Xelshekruleba
)
        SELECT N'ყველა ხელსშეკრულების თანხა აღემატება 5000 ლარს'
ELSE
        SELECT N'ყველა ხელსშეკრულების თანხა არ აღემატება 5000 ლარს';
```

7.3.9.

```
USE Shekveta;
DECLARE @sashualo_tanxa FLOAT;
SET @sashualo_tanxa =
(
SELECT AVG(gadasaxdeli_l)
FROM Xelshekruleba
);
SELECT @sashualo_tanxa;
IF 5000 < ALL
(
SELECT gadasaxdeli_l
FROM Xelshekruleba
)
        SELECT N'ყველა ხელშეკრულების თანხა აღემატება 5000 ლარს'
ELSE
        SELECT N'ყველა ხელშეკრულების თანხა არ აღემატება 5000 ლარს';
```

SOME და ANY ოპერატორები

7.4.1.

```
USE Shekveta;
IF 20 < ANY
(
SELECT staji
FROM Personali
)
        SELECT N'ზოგიერთი თანამშრომლის სტაჟი აღემატება 20 წელს'
ELSE
        SELECT N'არც ერთი თანამშრომლის სტაჟი არ აღემატება 20 წელს';
```

7.4.2.

```
USE Shekveta;
IF N'კი' = ANY
(
SELECT shesruleba
FROM Xelshekruleba
)
	SELECT N'ზოგიერთი ხელშეკრულება შესრულდა'
ELSE
	SELECT N'ზოგიერთი ხელშეკრულება არ შესრულდა';
SELECT xelshekrulebaID [არ შესრულდა ხელშეკრულებები]
FROM Xelshekruleba
WHERE shesruleba = N'არა';
```

7.4.5.

```
USE Shekveta;
DECLARE @sashualo_asaki FLOAT;
SET @sashualo_asaki =
(
SELECT AVG(asaki)
FROM Personali
);
SELECT @sashualo_asaki;
IF @sashualo_asaki < ANY
(
SELECT asaki
FROM Personali
)
	SELECT N'ზოგიერთი თანამშრომლის ასაკი აღემატება საშუალო ასაკს'
ELSE
	SELECT N'ზოგიერთი თანამშრომლის ასაკი არ აღემატება საშუალო ასაკს';
```

7.5.8.

```
USE Shekveta;
IF 4000 < ANY
(
SELECT gadasaxdeli_l
FROM Xelshekruleba
)
	SELECT N'ზოგიერთი ხელშეკრულების თანხა აღემატება 5000 ლარს'
ELSE
	SELECT N'ზოგიერთი ხელშეკრულების თანხა არ აღემატება 5000 ლარს';
```

7.5.9.

```
USE Shekveta;
DECLARE @sashualo_tanxa FLOAT;
SET @sashualo_tanxa =
(
SELECT AVG(gadasaxdeli_l)
```

```
FROM Xelshekruleba
);
SELECT @sashualo_tanxa;
IF 5000 < ANY
(
SELECT gadasaxdeli_l
FROM Xelshekruleba
)
        SELECT N'ზოგიერთი ხელშეკრულების თანხა აღემატება 5000 ლარს'
ELSE
        SELECT N'ზოგიერთი ხელშეკრულების თანხა არ აღემატება 5000 ლარს';
```

EXISTS ოპერატორი

7.5.1.

```
USE Shekveta;
SELECT gvari, qalaqi, iuridiuli_fizikuri
FROM   Shemkveti
WHERE EXISTS
(
SELECT  *
FROM    Shemkveti
WHERE iuridiuli_fizikuri = N'ფიზიკური'
);
```

7.5.2.

```
USE Shekveta;
SELECT gvari, qalaqi, iuridiuli_fizikuri
FROM   Shemkveti
WHERE EXISTS
(
SELECT  *
FROM    Shemkveti
WHERE qalaqi = N'თბილისი'
);
```

7.5.3.

```
USE Shekveta;
IF EXISTS
(
SELECT *
FROM Personali
WHERE qalaqi = N'თბილისი'
)
SELECT *
FROM Personali;
```

7.5.4.

```
USE Shekveta;
IF EXISTS
```

```
(
SELECT *
FROM Personali
WHERE asaki > 30
)
SELECT *
FROM Personali;
```

7.5.5.

```
USE Shekveta;
IF EXISTS
(
SELECT *
FROM Personali
WHERE staji >= 15
)
SELECT *
FROM Personali;
```

7.5.6.

```
USE Shekveta;
IF EXISTS
(
SELECT *
FROM Personali
WHERE xelfasi >= 1500
)
SELECT *
FROM Personali;
```

7.5.7.

```
USE Shekveta;
IF EXISTS
(
SELECT *
FROM Personali
WHERE ganyofileba = N'სამედიცინო'
)
SELECT *
FROM Personali;
```

7.5.8.

```
USE Shekveta;
SELECT gvari, qalaqi, xelfasi
FROM   Personali
WHERE EXISTS
(
SELECT  *
FROM    Personali
WHERE raioni = N'ვაკე'
```

);
7.5.9.
USE Shekveta;
IF EXISTS
(
SELECT *
FROM Personali
WHERE raioni = N'ვაკე'
)
PRINT N'ზოგიერთი თანამშრომელი ცხოვრობს დიდუბის რაიონში';

თავი 8. ოპერაციები სიმრავლეებზე

UNION ოპერაცია

8.1.1.
USE Shekveta;
SELECT qalaqi, regioni FROM Personali
UNION
SELECT qalaqi, regioni FROM Shemkveti;
8.1.2.
USE Shekveta;
SELECT qalaqi, regioni FROM Personali
UNION ALL
SELECT qalaqi, regioni FROM Shemkveti;

INTERSECT ოპერაცია

8.2.1.
USE Shekveta;
SELECT qalaqi, regioni FROM Personali
INTERSECT
SELECT qalaqi, regioni FROM Shemkveti;
8.2.2.
USE Shekveta;
SELECT gvari FROM Personali
INTERSECT
SELECT gvari FROM Shemkveti;
8.2.3.
USE Shekveta;
SELECT qalaqi FROM Personali
INTERSECT
SELECT qalaqi FROM Shemkveti;

EXCEPT ოპერაცია

8.3.1.
USE Shekveta;
SELECT qalaqi, regioni FROM Personali

```
EXCEPT
SELECT qalaqi, regioni FROM Shemkveti;
```
8.3.5.
```
USE Shekveta;
SELECT qalaqi, regioni FROM Shemkveti
EXCEPT
SELECT qalaqi, regioni FROM Personali;
```
8.3.9.
```
USE Shekveta;
SELECT qalaqi, raioni FROM Shemkveti WHERE iuridiuli_fizikuri = N'იურიდიული'
EXCEPT
SELECT qalaqi, raioni FROM Personali
INTERSECT
SELECT qalaqi, raioni FROM Shemkveti WHERE iuridiuli_fizikuri = N'ფიზიკური';
```
8.3.10.
```
USE Shekveta;
SELECT qalaqi, raioni FROM Shemkveti WHERE iuridiuli_fizikuri = N'იურიდიული'
UNION
SELECT qalaqi, raioni FROM Personali
INTERSECT
SELECT qalaqi, raioni FROM Shemkveti WHERE iuridiuli_fizikuri = N'ფიზიკური';
```
8.3.11.
```
USE Shekveta;
SELECT qalaqi, raioni FROM Shemkveti WHERE iuridiuli_fizikuri = N'ფიზიკური'
UNION
SELECT qalaqi, raioni FROM Personali
INTERSECT
SELECT qalaqi, raioni FROM Shemkveti WHERE iuridiuli_fizikuri = N'იურიდიული';
```
8.3.12.
```
USE Shekveta;
SELECT qalaqi, raioni FROM Shemkveti WHERE iuridiuli_fizikuri = N'ფიზიკური'
UNION
SELECT qalaqi, raioni FROM Personali WHERE ganyofileba = N'სამედიცინო'
INTERSECT
SELECT qalaqi, raioni FROM Shemkveti WHERE iuridiuli_fizikuri = N'იურიდიული';
```
8.3.13.
```
USE Shekveta;
SELECT qalaqi, raioni FROM Shemkveti WHERE iuridiuli_fizikuri = N'ფიზიკური'
EXCEPT
SELECT qalaqi, raioni FROM Personali WHERE ganyofileba = N'სამედიცინო'
INTERSECT
SELECT qalaqi, raioni FROM Shemkveti WHERE iuridiuli_fizikuri = N'იურიდიული';
```
8.3.24.
```
USE Shekveta;
(
SELECT qalaqi, raioni FROM Shemkveti WHERE iuridiuli_fizikuri = N'იურიდიული'
```

```
EXCEPT
SELECT qalaqi, raioni FROM Shemkveti WHERE iuridiuli_fizikuri = N'ფიზიკური'
)
INTERSECT
SELECT qalaqi, raioni FROM Personali;
```

8.3.25.

```
USE Shekveta;
(
SELECT qalaqi, raioni FROM Shemkveti WHERE iuridiuli_fizikuri = N'იურიდიული'
EXCEPT
SELECT qalaqi, raioni FROM Shemkveti WHERE iuridiuli_fizikuri = N'ფიზიკური'
)
INTERSECT
SELECT qalaqi, raioni FROM Personali WHERE ganyofileba = N'სამედიცინო';
```

8.3.26.

```
USE Shekveta;
(
SELECT qalaqi, raioni FROM Shemkveti WHERE iuridiuli_fizikuri = N'იურიდიული'
UNION
SELECT qalaqi, raioni FROM Shemkveti WHERE iuridiuli_fizikuri = N'ფიზიკური'
)
INTERSECT
SELECT qalaqi, raioni FROM Personali;
```

8.3.27.

```
USE Shekveta;
(
SELECT qalaqi, raioni FROM Shemkveti WHERE iuridiuli_fizikuri = N'იურიდიული'
UNION
SELECT qalaqi, raioni FROM Shemkveti WHERE iuridiuli_fizikuri = N'ფიზიკური'
)
INTERSECT
SELECT qalaqi, raioni FROM Personali WHERE ganyofileba = N'სამედიცინო';
```

თავი 9. ფუნქციები

Scalar ტიპის ფუნქციები

9.1.1.

```
USE Shekveta;
GO
CREATE FUNCTION Sashualo_Xelfasi( @ganyofileba NVARCHAR(30) )
RETURNS FLOAT
AS
BEGIN
DECLARE @sashualo_xelfasi FLOAT;
SET @sashualo_xelfasi =
(
```

```
SELECT AVG(xelfasi)
FROM Personali
WHERE ganyofileba = @ganyofileba
GROUP BY ganyofileba
);
RETURN @Sashualo_xelfasi
END
-- ფუნქციის გამოძახება:
SELECT dbo.Sashualo_Xelfasi(N'სამედიცინო')
        AS [სამედიცინო განყოფილების საშუალო ხელფასი];
```

9.1.2.

```
USE Shekveta;
GO
CREATE FUNCTION Minimialuri_Xelfasi( @ganyofileba NVARCHAR(30) )
RETURNS FLOAT
AS
BEGIN
DECLARE @minimaluri_xelfasi FLOAT;
SET @minimaluri_xelfasi =
(
SELECT MIN(xelfasi)
FROM Personali
WHERE ganyofileba = @ganyofileba
GROUP BY ganyofileba
);
RETURN @minimaluri_xelfasi
END
-- ფუნქციის გამოძახება:
SELECT dbo.Minimialuri_Xelfasi(N'სამედიცინო')
        AS [სამედიცინო განყოფილების მინიმალური ხელფასი];
```

9.1.3.

```
USE Shekveta;
GO
CREATE FUNCTION Minimialuri_Asaki( @ganyofileba NVARCHAR(30) )
RETURNS FLOAT
AS
BEGIN
DECLARE @minimaluri_asaki FLOAT;
SET @minimaluri_asaki =
(
SELECT MIN(asaki)
FROM Personali
WHERE ganyofileba = @ganyofileba
GROUP BY ganyofileba
);
```

```
RETURN @minimaluri_asaki
END
-- ფუნქციის გამოძახება:
SELECT dbo.Minimialuri_Asaki(N'სამედიცინო')
        AS [სამედიცინო განყოფილების მინიმალური ასაკი];
```

9.1.4.

```
USE Shekveta;
GO
CREATE FUNCTION Sashualo_Asaki( @ganyofileba NVARCHAR(30) )
RETURNS FLOAT
AS
BEGIN
DECLARE @sashualo_asaki FLOAT;
SET @sashualo_asaki =
(
SELECT MIN(asaki)
FROM Personali
WHERE ganyofileba = @ganyofileba
GROUP BY ganyofileba
);
RETURN @sashualo_asaki
END
-- ფუნქციის გამოძახება:
SELECT dbo.Sashualo_Asaki(N'სამედიცინო') AS [სამედიცინო განყოფილების საშუალო ასაკი];
```

9.1.5.

```
USE Shekveta;
GO
CREATE FUNCTION Sashualo_Staji( @ganyofileba NVARCHAR(30) )
RETURNS FLOAT
AS
BEGIN
DECLARE @sashualo_staji FLOAT;
SET @sashualo_staji =
(
SELECT MIN(staji)
FROM Personali
WHERE ganyofileba = @ganyofileba
GROUP BY ganyofileba
);
RETURN @sashualo_staji
END
-- ფუნქციის გამოძახება:
SELECT dbo.Sashualo_Staji(N'სამედიცინო') AS [სამედიცინო განყოფილების საშუალო სტაჟი]
```

9.1.6.

```
USE Shekveta;
GO
```

```
CREATE FUNCTION Maximaluri_Staji( @ganyofileba NVARCHAR(30) )
RETURNS FLOAT
AS
BEGIN
DECLARE @maximaluri_staji FLOAT;
SET @maximaluri_staji =
(
SELECT MIN(staji)
FROM Personali
WHERE ganyofileba = @ganyofileba
GROUP BY ganyofileba
);
RETURN @maximaluri_staji
END
-- ფუნქციის გამოძახება:
SELECT dbo.Maximaluri_Staji(N'სამედიცინო')
        AS [სამედიცინო განყოფილების მაქსიმალური სტაჟი];
```

9.1.7.

```
USE Shekveta;
GO
CREATE FUNCTION Qalaqi_Max_Staji( @qalaqi NVARCHAR(30) )
RETURNS FLOAT
AS
BEGIN
DECLARE @qalaqi_max_staji FLOAT;
SET @qalaqi_max_staji =
(
SELECT MAX(staji)
FROM Personali
WHERE qalaqi = @qalaqi
GROUP BY qalaqi
);
RETURN @qalaqi_max_staji
END
-- ფუნქციის გამოძახება:
SELECT dbo.Qalaqi_Max_Staji(N'თბილისი')
        AS [თბილისში მცხოვრები თანამშრომლების მაქსიმალური სტაჟი];
```

9.1.8.

```
USE Shekveta;
GO
CREATE FUNCTION Min_Xelfasi( @ganyofileba NVARCHAR(30) )
RETURNS FLOAT
AS
BEGIN
DECLARE @sashualo_xelfasi FLOAT;
SET @sashualo_xelfasi =
```

```
(
SELECT MIN(xelfasi)
FROM Personali
WHERE ganyofileba = @ganyofileba
GROUP BY ganyofileba
);
RETURN @Sashualo_xelfasi
END
-- ფუნქციის გამოძახება:
SELECT dbo.Min_Xelfasi(N'სამედიცინო')
            AS [სამედიცინო განყოფილების მინიმალური ხელფასი];
9.1.9.
USE Shekveta;
GO
CREATE FUNCTION Qalaqi_Sashualo_Staji( @qalaqi NVARCHAR(30) )
RETURNS FLOAT
AS
BEGIN
DECLARE @qalaqi_sashualo_staji FLOAT;
SET @qalaqi_sashualo_staji =
(
SELECT AVG(staji)
FROM Personali
WHERE qalaqi = @qalaqi
GROUP BY qalaqi
);
RETURN @qalaqi_sashualo_staji
END
-- ფუნქციის გამოძახება:
SELECT dbo.Qalaqi_Sashualo_Staji(N'თბილისი') AS [საშუალო სტაჟი];
```

Inline ტიპის ფუნქციები

```
9.2.1.
USE Shekveta;
GO
CREATE FUNCTION Sashualo_Xelfasi()
RETURNS TABLE
AS
RETURN
SELECT ganyofileba AS [განყოფილება], AVG(xelfasi) AS [საშუალო ხელფასი]
FROM Personali
GROUP BY ganyofileba;
--        ფუნქციის გამოძახება
SELECT * FROM Sashualo_Xelfasi() ORDER BY 2;
9.2.2.
USE Shekveta;
```

```
GO
CREATE FUNCTION Maximaluri_Xelfasi()
RETURNS TABLE
AS
RETURN
SELECT ganyofileba AS [განყოფილება], MAX(xelfasi) AS [საშუალო ხელფასი]
FROM Personali
GROUP BY ganyofileba;
--	ფუნქციის გამოძახება
SELECT * FROM Maximaluri_Xelfasi();
```

9.2.3.

```
USE Shekveta;
GO
CREATE FUNCTION Minimaluri_Xelfasi()
RETURNS TABLE
AS
RETURN
SELECT ganyofileba AS [განყოფილება], MIN(xelfasi) AS [საშუალო ხელფასი]
FROM Personali
GROUP BY ganyofileba;
--	ფუნქციის გამოძახება
SELECT * FROM Minimaluri_Xelfasi();
```

9.2.4.

```
USE Shekveta;
GO
CREATE FUNCTION Minimaluri_Asaki()
RETURNS TABLE
AS
RETURN
SELECT ganyofileba AS [განყოფილება], MIN(asaki) AS [მინიმალური ასაკი]
FROM Personali
GROUP BY ganyofileba;
--	ფუნქციის გამოძახება
SELECT * FROM Minimaluri_Asaki();
```

9.2.5.

```
USE Shekveta;
GO
CREATE FUNCTION Maximaluri_Asaki()
RETURNS TABLE
AS
RETURN
SELECT ganyofileba AS [განყოფილება], MAX(asaki) AS [საშუალო ხელფასი]
FROM Personali
GROUP BY ganyofileba;
--	ფუნქციის გამოძახება
SELECT * FROM Maximaluri_Asaki();
```

9.2.6.

```
USE Shekveta;
GO
CREATE FUNCTION Sashualo_Asaki()
RETURNS TABLE
AS
RETURN
SELECT ganyofileba AS [განყოფილება], AVG(asaki) AS [საშუალო ხელფასი]
FROM Personali
GROUP BY ganyofileba;
--        ფუნქციის გამოძახება
SELECT * FROM Sashualo_Asaki();
```

9.2.7.

```
USE Shekveta;
GO
CREATE FUNCTION Sashualo_Staji()
RETURNS TABLE
AS
RETURN
SELECT ganyofileba AS [განყოფილება], AVG(staji) AS [საშუალო სტაჟი]
FROM Personali
GROUP BY ganyofileba;
--        ფუნქციის გამოძახება
SELECT * FROM Sashualo_Staji();
```

9.2.8.

```
USE Shekveta;
GO
CREATE FUNCTION MAX_Staji()
RETURNS TABLE
AS
RETURN
SELECT ganyofileba AS [განყოფილება], MAX(staji) AS [მაქსიმალური სტაჟი]
FROM Personali
GROUP BY ganyofileba;
--        ფუნქციის გამოძახება
SELECT * FROM Max_Staji();
```

9.2.9.

```
USE Shekveta;
GO
CREATE FUNCTION Min_Staji()
RETURNS TABLE
AS
RETURN
SELECT ganyofileba AS [განყოფილება], MIN(staji) AS [მინიმალური სტაჟი]
FROM Personali
GROUP BY ganyofileba;
```

```
--        ფუნქციის გამოძახება
SELECT * FROM Min_Staji();
```

Multi-statement ტიპის ფუნქციები

9.3.1.

```
USE Shekveta;
GO
CREATE FUNCTION Funqcia_2 (@ricxvi FLOAT, @striqoni NVARCHAR(20))
RETURNS @Cxrili_1 TABLE
(
id1 INT PRIMARY KEY IDENTITY (1,1) NOT NULL,
Sveti_2 FLOAT,
Sveti_3 NVARCHAR(20)
)
AS
BEGIN
SET @ricxvi = @ricxvi + 25.5;
INSERT INTO @Cxrili_1 VALUES(@ricxvi, @striqoni);
SET @ricxvi = @ricxvi + 25.5;
INSERT INTO @Cxrili_1 VALUES(@ricxvi, @striqoni);
SET @ricxvi = @ricxvi + 25.5;
INSERT INTO @Cxrili_1 VALUES(@ricxvi, @striqoni);
RETURN
END
GO
--                ფუნქციის გამოძახება:
USE Shekveta;
SELECT *
FROM Funqcia_2(55.5, N'ანა');
```

9.3.2.

```
USE Shekveta;
GO
ALTER FUNCTION [dbo].[TanamSromlebi]( @staji INT, @asaki INT )
RETURNS TABLE
AS
RETURN
SELECT *
FROM Personali
WHERE staji < @staji AND asaki > @asaki;
--                ფუნქციის გამოსაძახებლად შეგვაქვს მოთხოვნა:
SELECT *
FROM TanamSromlebi(20, 30);
```

ჩადგმული ფუნქციები

მათემატიკის ფუნქციები

10.4.1.

```
USE Shekveta;
SELECT FLOOR(xelfasi) AS [დამრგვალება ქვევით],
        CEILING(xelfasi) AS [დამრგვალება ზევით],
        ROUND(xelfasi, 2) AS [დამრგვალება წერტილის შემდეგ],
        ROUND(xelfasi, -2) AS [დამრგვალება წერტილამდე]
FROM Personali;
```

9.4.3.

```
USE Shekveta;
SELECT POWER(xelfasi, 3) AS [რიცხვის ხარისხი],
        SQRT(xelfasi) AS [კვადრატული ფესვი რიცხვიდან]
FROM Personali;
```

9.4.4.

```
USE Shekveta;
DECLARE @i INT = 1;
PRINT N'შემთხვევითი რიცხვები'
WHILE @i <= 15
BEGIN
PRINT RAND();
SET @i = @i + 1;
END
```

თარიღებთან სამუშაო ფუნქციები

9.5.1.

```
USE Shekveta;
SET DATEFORMAT DMY;
SELECT gvari, saxeli, DATEDIFF(year, tarigi_dabadebis, GETDATE()) AS [ასაკი]
FROM Personali;
```

9.5.2.

```
USE Shekveta;
SET DATEFORMAT DMY;
SELECT MAX(DATEDIFF(year, tarigi_dabadebis, GETDATE())) AS [მაქსიმალური ასაკი]
FROM Personali;
```

9.5.3.

```
USE Shekveta;
SET DATEFORMAT DMY;
SELECT MIN(DATEDIFF(year, tarigi_dabadebis, GETDATE())) AS [მინიმალური ასაკი]
FROM Personali;
```

9.5.4.

```
USE Shekveta;
SET DATEFORMAT DMY;
SELECT AVG(DATEDIFF(year, tarigi_dabadebis, GETDATE())) AS [საშუალო ასაკი]
FROM Personali;
```

9.5.5.
```
USE Shekveta;
SET DATEFORMAT DMY;
SELECT MAX(DATEDIFF(year, tarigi_dabadebis, GETDATE()))
FROM Personali
WHERE sqesi = N'კაცი';
```
9.5.6.
```
USE Shekveta;
SET DATEFORMAT DMY;
SELECT MAX(DATEDIFF(year, tarigi_dabadebis, GETDATE()))
FROM Personali
WHERE sqesi = N'ქალი';
```

სტრიქონებთან სამუშაო ფუნქციები

9.6.1.
```
USE Shekveta;
SELECT saxeli + SPACE(5) + gvari
FROM   Personali;
```
9.6.2.
```
USE Shekveta;
SELECT STR(xelfasi, 7, 2)
FROM   Personali;
```
9.6.3.
```
USE Shekveta;
SELECT  QUOTENAME(gvari),       QUOTENAME(saxeli, '"'), QUOTENAME(ganyofileba, '{'),
          QUOTENAME(qalaqi, '('), QUOTENAME(raioni, '''')
FROM Personali;
```
9.6.4.
```
USE Shekveta;
SELECT STUFF(saxeli, 1, 1, N'გ')
FROM    Personali
WHERE  gvari = N'გაჩეჩილაძე';
```
9.6.5.
```
USE Shekveta;
SELECT REPLACE(gvari, N'ძე', N'შვილი')
FROM    Personali;
```
9.6.6.
```
USE Shekveta;
SELECT REPLICATE(qalaqi, 3)
FROM    Personali
WHERE qalaqi = N'თბილისი';
```
9.6.7.
```
USE Shekveta;
SELECT CHARINDEX(N'გი', saxeli),
          CHARINDEX(N'გი', saxeli, 2)
FROM    Personali
```

```
WHERE  saxeli LIKE N'გ%';
```
9.6.8.
```
USE Shekveta;
SELECT LEN(gvari) AS [სტრიქონის სიგრძე],
          NCHAR(4310) AS [უნიკოდ-სიმბოლო]
FROM    Personali
WHERE gvari = N'სამხარაძე' AND saxeli = N'ანა';
```
9.6.9.
```
USE Shekveta;
SELECT LEFT(gvari, 6) AS [პირველი 6 სიმბოლო],
          RIGHT(gvari, 2) AS [უკანასკნელი 2 სიმბოლო]
FROM    Personali
WHERE gvari = N'სამხარაძე' AND saxeli = N'საბა';
```

თავი 10. შენახული პროცედურები

10.1.1.
```
USE Shekveta;
--        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';
```
10.1.2.
```
USE Shekveta;
--        თუ შენახული პროცედურა არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID ('[dbo].[Qalaqi_PAR]', 'P' ) IS NOT NULL
   DROP PROCEDURE [dbo].[Qalaqi_PAR];
GO
--        შენახული პროცედურის შექმნა
CREATE PROCEDURE [dbo].[Qalaqi_PAR] @qalaqi NVARCHAR(30)
AS
SELECT *
FROM Personali
WHERE qalaqi = @qalaqi;
          -- გამოვიძახოთ ეს პროცედურა:
EXEC [dbo].[Qalaqi_PAR] N'თბილისი';
```
10.1.3.
```
USE Shekveta;
```

```
--        თუ შენახული პროცედურა არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID ( '[dbo].[Ganyofileba_PAR]', 'P' ) IS NOT NULL
   DROP PROCEDURE [dbo].[Ganyofileba_PAR];
GO
--        შენახული პროცედურის შექმნა
CREATE PROCEDURE [dbo].[Ganyofileba_PAR] @ganyofileba NVARCHAR(30) = N'სასპორტო'
AS
SELECT *
FROM Personali
WHERE ganyofileba = @ganyofileba;
-- გამოვიძახოთ ეს პროცედურა:
EXEC [dbo].[Ganyofileba_PAR] DEFAULT;
10.1.4.
USE Shekveta;
--        თუ შენახული პროცედურა არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID ( '[dbo].[Qalaqi_PAR]', 'P' ) IS NOT NULL
   DROP PROCEDURE [dbo].[Qalaqi_PAR];
GO
--        შენახული პროცედურის შექმნა
CREATE PROCEDURE [dbo].[Qalaqi_PAR] @qalaqi NVARCHAR(30) = N'ბათუმი'
AS
SELECT *
FROM Personali
WHERE qalaqi = @qalaqi;
-- გამოვიძახოთ ეს პროცედურა:
EXEC [dbo].[Qalaqi_PAR] DEFAULT;
10.1.5.
USE Shekveta;
--        თუ შენახული პროცედურა არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID ( '[dbo].[Max_Proc]', 'P' ) IS NOT NULL
   DROP PROCEDURE [dbo].[Max_Proc];
GO
--        შენახული პროცედურის შექმნა
CREATE PROC Max_Proc @ganyofileba NVARCHAR(30), @max_xelfasi FLOAT OUTPUT
AS
SET @max_xelfasi =
(
SELECT MAX(xelfasi)
FROM Personali
WHERE  @ganyofileba = ganyofileba
);
-- გამოვიძახოთ ეს პროცედურა:
DECLARE @ganyofileba NVARCHAR(30), @max_xelfasi FLOAT;
SET @ganyofileba = N'სავაჭრო';
EXEC Max_Proc @ganyofileba, @max_xelfasi OUTPUT;
SELECT @ganyofileba AS [განყოფილება], ROUND(@max_xelfasi,2) AS [მაქსიმალური ხელფასი];
```

10.1.6.

```
USE Shekveta;
--	თუ შენახული პროცედურა არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID ( '[dbo].[Min_Asaki_Proc]', 'P' ) IS NOT NULL
   DROP PROCEDURE [dbo].[Min_Asaki_Proc];
GO
--	შენახული პროცედურის შექმნა
CREATE PROC Min_Asaki_Proc @ganyofileba NVARCHAR(30), @min_asaki INT OUTPUT
AS
SET @min_asaki =
(
SELECT MIN(asaki)
FROM Personali
WHERE  @ganyofileba = ganyofileba
)
-- გამოვიძახოთ ეს პროცედურა:
DECLARE @ganyofileba NVARCHAR(30), @min_asaki INT;
SET @ganyofileba = N'სავაჭრო';
EXEC Min_Asaki_Proc @ganyofileba, @min_asaki OUTPUT;
SELECT @ganyofileba AS [განყოფილება], @min_asaki AS [მინიმალური ასაკი];
```

10.1.7.

```
USE Shekveta;
--	თუ შენახული პროცედურა არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID ( '[dbo].[qalaqi_asaki]', 'P' ) IS NOT NULL
   DROP PROCEDURE [dbo].[qalaqi_asaki];
GO
--	შენახული პროცედურის შექმნა
CREATE PROC [dbo].[qalaqi_asaki] @qalaqi NVARCHAR(30), @asaki INT
AS
SELECT *
FROM Personali
WHERE  @qalaqi = qalaqi AND @asaki > asaki;
--	გამოვიძახოთ ეს პროცედურა:
EXEC dbo.qalaqi_asaki N'თბილისი', 40;
```

10.1.8.

```
USE Shekveta;
--	თუ შენახული პროცედურა არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID ( '[dbo].[Ganyofileba_Staji]', 'P' ) IS NOT NULL
   DROP PROCEDURE [dbo].[Ganyofileba_Staji];
GO
--	შენახული პროცედურის შექმნა
CREATE PROC Ganyofileba_Staji @ganyofileba NVARCHAR(30), @staji INT
AS
SELECT *
FROM Personali
WHERE  @ganyofileba = ganyofileba AND @staji > staji;
```

```
-- გამოვიძახოთ ეს პროცედურა:
EXEC Ganyofileba_Staji N'სავაჭრო', 15;
```
10.1.9.
```
USE Shekveta;
--	თუ შენახული პროცედურა არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID ( '[dbo].[Procedura1]', 'P' ) IS NOT NULL
   DROP PROCEDURE Chemi_Procedura1;
GO
--	შენახული პროცედურის შექმნა
CREATE PROCEDURE Procedura1 @gvari NVARCHAR(30) = N'გ%'
AS
SELECT * FROM Shemkveti  WHERE gvari LIKE @gvari;
-- ამ პროცედურას შეგვიძლია გადავცეთ პარამეტრი
EXEC Procedura1  N'ს%';
```
10.1.10.
```
USE Shekveta;
--	თუ შენახული პროცედურა არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID ('Shemkvet_1', 'P' ) IS NOT NULL
   DROP PROCEDURE Shemkvet_1;
--		შენახული პროცედურის შექმნა
CREATE PROCEDURE Shemkvet_3 @qalaqi NVARCHAR(30)
AS
SELECT * FROM Shemkveti WHERE qalaqi = @qalaqi;
--	გამოვიძახოთ ეს პროცედურა:
EXEC Shemkvet_3 N'თბილისი';
```
10.1.11.
```
USE Shekveta;
--		თუ შენახული პროცედურა არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID ('Shemkvet_1', 'P' ) IS NOT NULL
   DROP PROCEDURE Shemkvet_1;
GO
--		შენახული პროცედურის შექმნა
CREATE PROCEDURE Shemkvet_1 @qalaqi NVARCHAR(30), @asaki INT
AS
SELECT *
FROM Personali
WHERE qalaqi = @qalaqi AND asaki < @asaki;
	გამოვიძახოთ ეს პროცედურა:
EXEC Shemkvet_1 N'თბილისი', 45;
```
10.1.12.
```
USE Shekveta;
--		თუ შენახული პროცედურა არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID ('Shemkvet_4', 'P' ) IS NOT NULL
   DROP PROCEDURE Shemkvet_4;
GO
--		შენახული პროცედურის შექმნა
```

```
CREATE PROCEDURE Shemkvet_4 @tanxa FLOAT, @tarigi INT
AS
SELECT *
FROM Xelshekruleba
WHERE @tarigi <= YEAR(tarigi_dawyebis) AND @tanxa > gadasaxdeli_l;
--      გამოვიძახოთ ეს პროცედურა:
EXEC Shemkvet_4 4500.00, 2002;
10.1.13.
USE Shekveta;
GO
ALTER PROCEDURE Shemkvet_1 @ganyofileba NVARCHAR(30), @xelfasi FLOAT
AS
SELECT *
FROM Personali
WHERE ganyofileba = @ganyofileba AND xelfasi < @xelfasi;
--      გამოვიძახოთ ეს პროცედურა:
EXEC Shemkvet_1 N'სამედიცინო', 3500;
```

თავი 11. ინდექსები

```
11.1.1.
USE Shekveta;
--      თუ ინდექსი არსებობს, მაშინ ის წაიშლება
IF EXISTS
(
SELECT name
FROM sys.indexes
WHERE name = N'Index_Gvari'
)
DROP INDEX Index_Gvari ON Personali;
GO
--      ინდექსის შექმნა
CREATE NONCLUSTERED INDEX Index_Gvari
ON [dbo].[Personali] ([gvari]);
11.1.2.
USE Shekveta;
--      თუ ინდექსი არსებობს, მაშინ ის წაიშლება
IF EXISTS
(
SELECT name
FROM sys.indexes
WHERE name = N'Index_Gvari1'
)
DROP INDEX Index_Gvari1 ON Personali;
GO
--      ინდექსის შექმნა
```

```
CREATE UNIQUE NONCLUSTERED INDEX Index_Gvari1
ON [dbo].[Personali] ([gvari DESC]);
```
11.1.3.
```
USE Shekveta;
--        თუ ინდექსი არსებობს, მაშინ ის წაიშლება
IF EXISTS
(
SELECT name
FROM sys.indexes
WHERE name = N'Index__mail'
)
DROP INDEX Index_mail ON Personali;
GO
--        ინდექსის შექმნა
CREATE UNIQUE INDEX Index__mail
ON [dbo].[Personali] ([email]);
```
11.1.4.
```
USE Shekveta;
--        თუ ინდექსი არსებობს, მაშინ ის წაიშლება
IF EXISTS
(
SELECT name
FROM sys.indexes
WHERE name = N'Index_Mobiluri'
)
DROP INDEX Index_Mobiluri ON Personali;
GO
--        ინდექსის შექმნა
CREATE UNIQUE NONCLUSTERED INDEX Index_mobiluri
ON [dbo].[Personali] ([mobiluri]);
```
11.1.5.
```
USE Shekveta;
GO
--        ინდექსის შექმნა
CREATE NONCLUSTERED INDEX Index_Ganyofileba
ON Personali(ganyofileba)
WITH (DROP_EXISTING = ON);
```
11.1.6.
```
USE Shekveta;
DBCC DBREINDEX ('Personali', Index_mobiluri, 80);
```
11.1.7.
```
USE Shekveta;
DBCC DBREINDEX ('Personali', '');
```
11.1.8.
```
USE Shekveta;
--        თუ ინდექსი არსებობს, მაშინ ის წაიშლება
```

```
IF EXISTS
(
SELECT name
FROM sys.indexes
WHERE name = N'Index_firmis_dasaxeleba'
)
DROP INDEX Index_firmis_dasaxeleba ON Shemkveti;
GO
--        ინდექსის შექმნა
CREATE UNIQUE NONCLUSTERED INDEX Index_firmis_dasaxeleba
ON [dbo].[Shemkveti] ([firmis_dasaxeleba]);
```

11.1.9.

```
USE Shekveta;
--        თუ ინდექსი არსებობს, მაშინ ის წაიშლება
IF EXISTS
(
SELECT name
FROM sys.indexes
WHERE name = N'Index_warmomadgeneli'
)
DROP INDEX Index_warmomadgeneli ON Shemkveti;
GO
--        ინდექსის შექმნა
CREATE NONCLUSTERED INDEX Index_warmomadgeneli
ON [dbo].[Shemkveti] ([warmomadgeneli]);
```

11.1.10.

```
USE Shekveta;
GO
--        ინდექსის შექმნა
CREATE INDEX Ind_gvari_S
ON Shemkveti (firmis_dasaxeleba)
WITH ( DROP_EXISTING = ON);
```

11.1.11.

```
USE Shekveta;
GO
--        ინდექსის შექმნა
CREATE NONCLUSTERED INDEX Ind_gvari_P
ON Personali (gvari DESC, asaki ASC)
WITH ( DROP_EXISTING = ON);
```

11.1.12.

```
USE Shekveta;
GO
--        ინდექსის შეცვლა
ALTER INDEX Ind_gvari_S ON Shemkveti
REBUILD
WITH ( IGNORE_DUP_KEY = ON);
```

თავი 12. წარმოდგენები

12.1.1.

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

12.1.2.

```
USE Shekveta;
--        თუ წარმოდგენა არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID('Gvari_40', 'VIEW') IS NOT NULL
DROP VIEW Gvari_40;
GO
--        წარმოდგენის შექმნა
CREATE VIEW Gvari_40
AS
SELECT  dbo.Personali.gvari  AS [თანამშრომლის გვარი],
          dbo.Shemkveti.gvari  AS [შემკვეთის გვარი]
FROM dbo.Personali INNER JOIN dbo.Shemkveti
        ON dbo.Personali.PersonaliID = dbo.Shemkveti.PersonaliID
WHERE dbo.Personali.asaki > 40;
-- წარმოდგენის  გახსნა:
SELECT * FROM Gvari_40;
```

12.1.3.

```
USE Shekveta;
--        თუ წარმოდგენა არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID('Staji_10', 'VIEW') IS NOT NULL
DROP VIEW Staji_10;
GO
--        წარმოდგენის შექმნა
CREATE VIEW Staji_10
AS
SELECT    gvari, ganyofileba, qalaqi, asaki, staji
FROM      dbo.Personali
WHERE    staji > 10;
-- წარმოდგენის  გახსნა:
SELECT * FROM Staji_10;
```

12.1.4.

```
USE Shekveta;
GO
--	თუ წარმოდგენა არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID('Staji_10', 'VIEW') IS NOT NULL
DROP VIEW Staji_10;
--	წარმოდგენის შექმნა
CREATE VIEW Staji_10 WITH ENCRYPTION
AS
SELECT	gvari, ganyofileba, qalaqi, asaki, staji
FROM	dbo.Personali
WHERE	staji < 10;
-- წარმოდგენის  გახსნა:
SELECT * FROM Staji_10;
```

12.1.5.

```
USE Shekveta;
--	თუ წარმოდგენა არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID('View_Function', 'VIEW') IS NOT NULL
DROP VIEW View_Function;
GO
--	წარმოდგენის შექმნა
CREATE VIEW View_Function
AS
SELECT PersonaliID, MAX(vali_l) AS [მაქსიმალური ვალი]
FROM Xelshekruleba
GROUP BY PersonaliID;
-- წარმოდგენის  გახსნა:
SELECT * FROM View_Function;
```

12.1.6.

```
USE Shekveta;
--	თუ წარმოდგენა არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID('View_Asaki', 'VIEW') IS NOT NULL
DROP VIEW View_Asaki;
GO
--	წარმოდგენის შექმნა
CREATE VIEW View_Asaki
AS
SELECT AVG(asaki) AS [საშუალო ასაკი]
FROM Personali
GROUP BY PersonaliID;
-- წარმოდგენის  გახსნა:
SELECT * FROM View_Asaki;
```

12.1.7.

```
USE Shekveta;
--	თუ წარმოდგენა არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID('View_Asaki', 'VIEW') IS NOT NULL
DROP VIEW View_Asaki;
```

```
GO
--	წარმოდგენის შექმნა
CREATE VIEW View_Asaki
AS
SELECT *
FROM Personali
WHERE asaki BETWEEN 25 AND 45;
-- წარმოდგენის  გახსნა:
SELECT * FROM View_Asaki;
```
12.1.8.
```
USE Shekveta;
--	თუ წარმოდგენა არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID('View_Staji', 'VIEW') IS NOT NULL
DROP VIEW View_Asaki;
GO
--	წარმოდგენის შექმნა
CREATE VIEW View_Staji
AS
SELECT AVG(staji) AS [საშუალო სტაჟი]
FROM Personali
GROUP BY PersonaliID;
-- წარმოდგენის  გახსნა:
SELECT * FROM View_Staji;
```
12.1.9.
```
USE Shekveta;
--	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.Personali.staji,
                                dbo.Shemkveti.gvari AS [შემკვეთის გვარი]
FROM dbo.Personali INNER JOIN
        dbo.Shemkveti ON dbo.Personali.PersonaliID = dbo.Shemkveti.PersonaliID
WHERE dbo.Personali.staji > 20
ORDER BY dbo.Personali.gvari ASC;
--	წარმოდგენის  გახსნა:
SELECT * FROM View_2;
```
12.1.10.
```
USE Shekveta;
--	თუ წარმოდგენა არსებობს, მაშინ ის წაიშლება
IF OBJECT_ID('View_1', 'VIEW') IS NOT NULL
DROP VIEW View_1;
GO
```

```
--      წარმოდგენის შექმნა
CREATE VIEW View_1
AS
SELECT TOP 100 PERCENT dbo.Personali.gvari, dbo.Personali.asaki,
                                dbo.Xelshekruleba.gadasaxdeli_l, Xelshekruleba.tarigi_dawyebis
FROM dbo.Personali INNER JOIN dbo.Xelshekruleba
        ON dbo.Personali.PersonaliID = dbo.Xelshekruleba.PersonaliID
WHERE dbo.Xelshekruleba.tarigi_dawyebis BETWEEN '2002-01-01' AND '2005-12-31'
ORDER BY dbo.Personali.gvari ASC, dbo.Xelshekruleba.tarigi_dawyebis DESC;
        წარმოდგენის  გახსნა:
SELECT * FROM View_1;
12.1.11.
USE Shekveta;
GO
--      წარმოდგენის შეცვლა
ALTER VIEW View_2
AS
SELECT TOP 100 PERCENT dbo.Personali.gvari, dbo.Personali.asaki,
                                dbo.Shemkveti.gvari AS [შემკვეთის გვარი]
FROM dbo.Personali INNER JOIN dbo.Shemkveti
        ON dbo.Personali.PersonaliID = dbo.Shemkveti.PersonaliID
WHERE dbo.Personali.asaki < 40
ORDER BY dbo.Personali.gvari DESC;
        წარმოდგენის  გახსნა:
SELECT * FROM View_2;
```

თავი 13. კურსორები

```
13.1.
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;
13.2.
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;
```
13.3.
```
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;
```
13.4.
```
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;
```
13.5.
```
USE Shekveta;
DECLARE Xelshekruleba_Cursori CURSOR LOCAL
FOR
        SELECT *
        FROM Xelshekruleba
        WHERE vali_l > 0
FOR UPDATE;
OPEN Xelshekruleba_Cursori;
FETCH NEXT FROM Xelshekruleba_Cursori;
CLOSE Xelshekruleba_Cursori;
DEALLOCATE Xelshekruleba_Cursori;
```
13.6.
```
USE Shekveta;
DECLARE Xelshekruleba_Cursori CURSOR GLOBAL
FOR
        SELECT *
        FROM Xelshekruleba
```

```
        WHERE vali_l > 0
FOR UPDATE;
OPEN Xelshekruleba_Cursori;
FETCH NEXT FROM Xelshekruleba_Cursori;
CLOSE Xelshekruleba_Cursori;
DEALLOCATE Xelshekruleba_Cursori;
```
13.7.
```
USE Shekveta;
DECLARE Xelshekruleba_Cursori CURSOR FORWARD_ONLY
FOR
        SELECT *
        FROM Xelshekruleba
        WHERE vali_l > 0;
OPEN Xelshekruleba_Cursori;
FETCH NEXT FROM Xelshekruleba_Cursori;
CLOSE Xelshekruleba_Cursori;
DEALLOCATE Xelshekruleba_Cursori;
```
13.8.
```
USE Shekveta;
DECLARE Xelshekruleba_Cursori CURSOR KEYSET
FOR
        SELECT *
        FROM Xelshekruleba
        WHERE vali_l > 0;
OPEN Xelshekruleba_Cursori;
FETCH NEXT FROM Xelshekruleba_Cursori;
CLOSE Xelshekruleba_Cursori;
DEALLOCATE Xelshekruleba_Cursori;
```
13.9.
```
USE Shekveta;
DECLARE Xelshekruleba_Cursori CURSOR DYNAMIC
FOR
        SELECT *
        FROM Xelshekruleba
        WHERE vali_l > 0;
OPEN Xelshekruleba_Cursori;
FETCH NEXT FROM Xelshekruleba_Cursori;
CLOSE Xelshekruleba_Cursori;
DEALLOCATE Xelshekruleba_Cursori;
```
13.10.
```
USE Shekveta;
DECLARE Xelshekruleba_Cursori CURSOR FAST_FORWARD
FOR
        SELECT *
        FROM Xelshekruleba
        WHERE vali_l > 0;
```

```
OPEN Xelshekruleba_Cursori;
FETCH NEXT FROM Xelshekruleba_Cursori;
CLOSE Xelshekruleba_Cursori;
DEALLOCATE Xelshekruleba_Cursori;
```
13.11.
```
USE Shekveta;
--	კურსორის შექმნა
DECLARE Kursori_1 CURSOR GLOBAL SCROLL KEYSET TYPE_WARNING
FOR
	SELECT  gadasaxdeli_l
	FROM Xelshekruleba X;
--	კურსორის გახსნა
OPEN Kursori_1;
DECLARE  @@Striqonebis_raodenoba INT, @@Jami FLOAT, @@gadasaxdeli_l FLOAT;
FETCH ABSOLUTE 5 FROM Kursori_1 INTO @@gadasaxdeli_l;
SELECT  @@gadasaxdeli_l AS [გადასახდელი თანხაა:];
--	კურსორის დახურვა და გათავისუფლება
CLOSE Kursori_1;
DEALLOCATE Kursori_1;
```
13.12.
```
USE Shekveta;
--	კურსორის შექმნა
DECLARE Kursori_1 CURSOR GLOBAL SCROLL KEYSET TYPE_WARNING
FOR
	SELECT  gadasaxdeli_l
	FROM Xelshekruleba X;
--	კურსორის გახსნა
OPEN Kursori_1;
DECLARE  @@Striqonebis_raodenoba INT, @@Jami FLOAT, @@gadasaxdeli_l FLOAT;
FETCH FIRST FROM Kursori_1 INTO @@gadasaxdeli_l;
SELECT  @@gadasaxdeli_l AS [გადასახდელი თანხაა:];
--	კურსორის დახურვა და გათავისუფლება
CLOSE Kursori_1;
DEALLOCATE Kursori_1;
```
13.13.
```
USE Shekveta;
--	კურსორის შექმნა
DECLARE Kursori_1 CURSOR GLOBAL SCROLL KEYSET TYPE_WARNING
FOR
	SELECT  gadasaxdeli_l
	FROM Xelshekruleba X  ;
--	კურსორის გახსნა
OPEN Kursori_1;
DECLARE  @@Striqonebis_raodenoba INT, @@Jami FLOAT, @@gadasaxdeli_l FLOAT;
FETCH LAST FROM Kursori_1 INTO @@gadasaxdeli_l;
SELECT  @@gadasaxdeli_l AS [გადასახდელი თანხაა:];
```

```
--	კურსორის დახურვა და გათავისუფლება
CLOSE Kursori_1;
DEALLOCATE Kursori_1;
13.14.
USE Shekveta;
--	კურსორის შექმნა
DECLARE Kursori_1 CURSOR GLOBAL SCROLL KEYSET TYPE_WARNING
FOR
	SELECT  gadasaxdeli_l
	FROM Xelshekruleba X;
--	კურსორის გახსნა
OPEN Kursori_1;
DECLARE  @@Striqonebis_raodenoba INT, @@Jami FLOAT, @@gadasaxdeli_l FLOAT;
FETCH FIRST FROM Kursori_1 INTO @@gadasaxdeli_l;
FETCH NEXT FROM Kursori_1 INTO @@gadasaxdeli_l;
SELECT  @@gadasaxdeli_l AS [გადასახდელი თანხაა:];
--	კურსორის დახურვა და გათავისუფლება
CLOSE Kursori_1;
DEALLOCATE Kursori_1;
13.15.
USE Shekveta;
--	კურსორის შექმნა
DECLARE Kursori_1 CURSOR GLOBAL SCROLL KEYSET TYPE_WARNING
FOR
	SELECT  gadasaxdeli_l
	FROM Xelshekruleba X;
--	კურსორის გახსნა
OPEN Kursori_1;
DECLARE  @@Striqonebis_raodenoba INT, @@Jami FLOAT, @@gadasaxdeli_l FLOAT;
FETCH LAST FROM Kursori_1 INTO @@gadasaxdeli_l;
FETCH PRIOR FROM Kursori_1 INTO @@gadasaxdeli_l;
SELECT  @@gadasaxdeli_l AS [გადასახდელი თანხაა:];
--	კურსორის დახურვა და გათავისუფლება
CLOSE Kursori_1;
DEALLOCATE Kursori_1;
13.16.
USE Shekveta;
--	კურსორის შექმნა
DECLARE Kursori_1 CURSOR GLOBAL SCROLL KEYSET TYPE_WARNING
FOR
	SELECT  gadasaxdeli_l, vali_l, tarigi_dawyebis
	FROM Xelshekruleba X;
--	კურსორის გახსნა
OPEN Kursori_1;
DECLARE  @@vali_l FLOAT, @@gadasaxdeli_l FLOAT, @@tarigi_dawyebis DATETIME;
FETCH ABSOLUTE 5  FROM Kursori_1
```

```
                        INTO @@gadasaxdeli_l, @@vali_l, @@tarigi_dawyebis;
SELECT  @@gadasaxdeli_l AS [გადასახდელი თანხაა], @@vali_l AS [ვალი],
          @@tarigi_dawyebis AS [გაფორმების თარიღი]
--        კურსორის დახურვა და გათავისუფლება  INTO @variable_name
CLOSE Kursori_1;
DEALLOCATE Kursori_1;
13.17.
USE Shekveta;
DECLARE @n INT, @raod INT;
SET @n = 1;
DECLARE Kursori_2 CURSOR GLOBAL SCROLL KEYSET TYPE_WARNING
FOR
        SELECT  gvari, xelfasi, asaki, ganyofileba, staji
        FROM Personali
        WHERE ganyofileba = N'სავაჭრო'
FOR UPDATE;
OPEN Kursori_2;
--        მონაცემების შეცვლა
SELECT @raod = COUNT(*)
FROM Personali
WHERE ganyofileba = N'სავაჭრო';
WHILE @n <= @raod
BEGIN
FETCH ABSOLUTE @n FROM Kursori_2;
UPDATE Personali SET xelfasi = xelfasi + 200  WHERE CURRENT OF Kursori_2;
SET @n = @n + 1;
END
--        კურსორის დახურვა და გათავისუფლება
CLOSE Kursori_2;
DEALLOCATE Kursori_2;
13.18.
USE Shekveta;
DECLARE @n INT, @raod INT;
SET @n = 1;
DECLARE Kursoti_3 CURSOR DYNAMIC GLOBAL SCROLL TYPE_WARNING
FOR
        SELECT  *
        FROM Personali
        WHERE qalaqi = N'ბათუმი'
FOR UPDATE;
OPEN Kursoti_3;
--        სტრიქონების წაშლა
SELECT @raod = COUNT(*)
FROM Personali
WHERE qalaqi = N'ბათუმი';
WHILE @n <= @raod
```

```
BEGIN
FETCH FIRST FROM Kursoti_3;
DELETE Personali WHERE CURRENT OF Kursoti_3;
SET @n = @n + 1;
END
--        კურსორის დახურვა და გათავისუფლება
CLOSE Kursoti_3 ;
DEALLOCATE Kursoti_3;
```

თავი 14. ტრანზაქციები და დაბლოკვები

14.1.

```
USE Shekveta;
DECLARE @TranSaxeli NVARCHAR(20);
SELECT @TranSaxeli = N'ChemiTranzaqcia';
BEGIN TRANSACTION @TranSaxeli;
UPDATE Personali SET xelfasi = xelfasi * 2;
COMMIT TRANSACTION ChemiTranzaqcia;
SELECT * FROM Personali;
```

14.2.

```
USE Shekveta;
DECLARE @TranSaxeli NVARCHAR(20);
SELECT @TranSaxeli = N'ChemiTranzaqcia';
BEGIN TRANSACTION @TranSaxeli;
UPDATE Personali SET xelfasi = xelfasi * 2;
ROLLBACK TRANSACTION ChemiTranzaqcia;
SELECT * FROM Personali;
```

14.3.

```
USE Shekveta;
DECLARE @Sveti_1 INT, @Sveti_2 NVARCHAR(20);
CREATE TABLE #Droebiti_Cxrili_2
(
Sveti_1 INT,
Sveti_2 NVARCHAR(4)
);
GO
CREATE PROCEDURE Chemi_Proc @Sveti_1 INT, @Sveti_2 NVARCHAR(4) AS
BEGIN TRANSACTION Chemi_Trans_1;
UPDATE #Droebiti_Cxrili_2 SET Sveti_1 = @Sveti_1 + 50;
UPDATE #Droebiti_Cxrili_2 SET Sveti_2 = @Sveti_2;
COMMIT TRANSACTION Chemi_Trans_1;
--        ტრანზაქციის დაწყება და TransProc პროცედურის შესრულება
BEGIN TRANSACTION Chemi_Trans_2;
EXEC Chemi_Proc 10, N'ანა';
--        გარე ტრანზაქციების უკუქცევა
ROLLBACK TRANSACTION Chemi_Trans_2;
```

```
EXEC Chemi_Proc 30, N'საბა';
SELECT * FROM #Droebiti_Cxrili_2;
```

თავი 15. ტრიგერები

15.1.

```
USE Shekveta;
--      თუ ტრიგერი არსებობს, მაშინ ის წაიშლება
IF EXISTS
(
SELECT name
FROM sysobjects
WHERE name = 'Personali_Update' AND type = 'TR'
)
DROP TRIGGER Personali_Update;
GO
--      ტრიგერის შექმნა
CREATE TRIGGER Personali_Update ON Personali_1
FOR UPDATE AS
PRINT STR(@@ROWCOUNT) + N' სტრიქონის შეცვლის მცდელობა Personali_1 ცხრილში';
PRINT N'მომხმარებელი -  ' + CURRENT_USER;
IF CURRENT_USER <> 'dbo'
BEGIN
  PRINT N'შეცვლა აკრძალულია'
  ROLLBACK TRANSACTION
END
ELSE
  PRINT N'შეცვლა ნებადართულია';
      ახლა Personali_1 ცხრილში სტრიქონები შევცვალოთ:
UPDATE Personali_1 SET xelfasi = xelfasi + 300 WHERE ganyofileba = N'სამედიცინო';
```

15.2.

```
USE Shekveta;
--      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' AND saxeli = N'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', saxeli = N'რომანი'
WHERE gvari = N'ნონიაშვილი' AND saxeli = N'ზურა';
15.3.
USE Shekveta;
--	თუ ტრიგერი არსებობს, მაშინ ის წაიშლება
IF EXISTS ( SELECT name FROM sysobjects WHERE name = 'Trigeri_4' AND type = 'TR' )
DROP TRIGGER Trigeri_4;
GO
--	ტრიგერის შექმნა
CREATE TRIGGER Trigeri_4 ON Shemkveti FOR DELETE AS
DECLARE @@Result INT;
SET @@Result = 1;
IF EXISTS
(
SELECT *
FROM deleted
WHERE qalaqi = N'თბილისი'
)
BEGIN
 PRINT N'თბილისელი შემკვეთების წაშლის მცდელობა';
 SET @@Result = 0;
END
IF @@Result = 0
BEGIN
 PRINT N'წაშლა აკრძალულია';
 ROLLBACK TRANSACTION;
END
	ახლა შევეცადოთ სტრიქონების წაშლას:
DELETE FROM Shemkveti WHERE qalaqi = N'თბილისი';
15.5.
USE Shekveta;
--	თუ ტრიგერი არსებობს, მაშინ ის წაიშლება
IF EXISTS
(
SELECT name
FROM sysobjects
WHERE name = 'Personali_Update' AND type = 'TR'
```

```
)
DROP TRIGGER Personali_Update;
GO
--        ტრიგერის შექმნა
CREATE TRIGGER Personali_Update ON Personali_2
FOR UPDATE AS
PRINT STR(@@ROWCOUNT) + N' სტრიქონი შეიცვალა Personali_2 ცხრილში';
PRINT N'მომხმარებელი -  ' + CURRENT_USER;
--        ახლა Personali_1 ცხრილში სტრიქონები შევცვალოთ:
UPDATE Personali_2 SET xelfasi = xelfasi + 300 WHERE ganyofileba = N'სამედიცინო';
```

თავი 16. ცხრილური გამოსახულებები

წარმოებული ცხრილები

16.1.1.

```
USE Shekveta;
SELECT  dawyebisweli, COUNT(xelshekrulebaID) AS raodenobaXelshekrulebis
FROM
(
SELECT YEAR(tarigi_dawyebis) AS dawyebisweli, xelshekrulebaID
FROM Xelshekruleba
) AS T1
GROUP BY dawyebisweli;
```

16.1.2.

```
USE Shekveta;
DECLARE @personaliID AS INT = 3;
SELECT  dawyebisweli, COUNT(xelshekrulebaID) AS raodenobaXelshekrulebis
FROM
(
SELECT YEAR(tarigi_dawyebis) AS dawyebisweli, xelshekrulebaID
FROM Xelshekruleba
WHERE personaliID = @personaliID
) AS T1
GROUP BY dawyebisweli;
```

16.1.3.

```
USE Shekveta;
SELECT dawyebisweli, raodshems
FROM
(
SELECT dawyebisweli, COUNT(DISTINCT personaliID) AS raodshems
FROM
(
SELECT YEAR(tarigi_dawyebis) AS dawyebisweli, personaliID
FROM Xelshekruleba
)
```

```
AS T1
GROUP BY dawyebisweli
) AS T2
WHERE raodshems > 1;
```

16.1.4.

```
USE Shekveta;
SELECT dawyebisweli, raodxelshekruleba
FROM
(
SELECT dawyebisweli, COUNT(xelshekrulebaID) AS raodxelshekruleba
FROM
(
SELECT YEAR(tarigi_dawyebis) AS dawyebisweli, xelshekrulebaID
FROM Xelshekruleba
)
AS T1
GROUP BY dawyebisweli
) AS T2
WHERE raodxelshekruleba > 1;
```

საერთო ცხრილური გამოსახულებები

16.2.1.

```
USE Shekveta;
WITH TbilisiPersonali AS
(
SELECT personaliID, gvari, saxeli, ganyofileba
FROM Personali
WHERE qalaqi = N'თბილისი'
)
SELECT * FROM TbilisiPersonali;
```

16.2.2.

```
USE Shekveta;
DECLARE @personaliID AS INT = 3;
WITH TbilisiPersonali AS
(
SELECT personaliID, gvari, saxeli, ganyofileba
FROM Personali
WHERE personaliID = @personaliID
)
SELECT * FROM TbilisiPersonali;
```

16.2.3.

```
USE Shekveta;
WITH T3 AS
(
SELECT dawyebisweli, raodpersonali
FROM
```

```
(
SELECT dawyebisweli, COUNT(DISTINCT personaliID) AS raodpersonali
FROM
(
SELECT YEAR(tarigi_dawyebis) AS dawyebisweli, personaliID
FROM Xelshekruleba
)
AS T1
GROUP BY dawyebisweli
) AS T2
WHERE raodpersonali > 1
)
SELECT  dawyebisweli, raodpersonali FROM T3;
```

16.2.4.

```
USE Shekveta;
WITH T3 AS
(
SELECT dawyebisweli, raodXelshekruleba
FROM
(
SELECT dawyebisweli, COUNT(xelshekrulebaID) AS raodXelshekruleba
FROM
(
SELECT YEAR(tarigi_dawyebis) AS dawyebisweli, xelshekrulebaID
FROM Xelshekruleba
)
AS T1
GROUP BY dawyebisweli
) AS T2
WHERE raodXelshekruleba > 1
)
SELECT  dawyebisweli, raodxelshekruleba FROM T3;
```

APPLY ოპერაცია

16.3.1.

```
USE Shekveta;
SELECT P.personaliID, A.xelshekrulebaID, A.tarigi_dawyebis
FROM Personali AS P
CROSS APPLY
(
SELECT TOP (3) xelshekrulebaID, tarigi_dawyebis
FROM Xelshekruleba AS X
WHERE X.personaliID = P.personaliID
ORDER BY tarigi_dawyebis DESC, xelshekrulebaID DESC
) AS A;
```

16.3.2.

```
USE Shekveta;
SELECT P.personaliID, A.xelshekrulebaID, A.tarigi_dawyebis
FROM Personali AS P
CROSS APPLY
(
SELECT TOP (3) xelshekrulebaID, tarigi_dawyebis
FROM Xelshekruleba AS X
WHERE X.personaliID = P.personaliID
ORDER BY tarigi_dawyebis DESC, xelshekrulebaID
) AS A;
```

16.3.3.

```
USE Shekveta;
SELECT P.personaliID, T1.xelshekrulebaID, T1.tarigi_dawyebis
FROM Personali AS P
OUTER APPLY
(
SELECT TOP (3) xelshekrulebaID, personaliID, tarigi_dawyebis
FROM Xelshekruleba AS X
WHERE X.personaliID = P.personaliID
ORDER BY tarigi_dawyebis DESC, xelshekrulebaID DESC
) AS T1;
```

16.3.4.

```
USE Shekveta;
SELECT P.personaliID, T1.xelshekrulebaID, T1.tarigi_dawyebis
FROM Personali AS P
OUTER APPLY
(
SELECT TOP (3) xelshekrulebaID, personaliID, tarigi_dawyebis
FROM Xelshekruleba AS X
WHERE X.personaliID = P.personaliID
ORDER BY tarigi_dawyebis DESC, xelshekrulebaID
) AS T1;
```

თავი 18. მონაცემთა ბაზების სარეზერვო ასლები

18.1.

```
USE master;
EXEC sp_addumpdevice 'DISK', 'LokaluriDiski',
'C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Backup\Asli_3.bak';
```

18.2.

```
USE master;
EXEC sp_addumpdevice 'TAPE', 'Lenti_2',  '\\.\tape0';
```

18.3.

```
USE master;
EXEC sp_addumpdevice 'DISK', 'QseluriDiski_2', '\\Serveri_2\Romani\C:\ChemiKatalogi\Asli_2.bak';
```

18.4.

```
EXEC sp_dropdevice 'Lenti_2', 'DELFILE';
18.5.
USE master;
BACKUP DATABASE Baza_3 TO Baza_3_Device WITH NAME = 'Baza_3_Asli', INIT;
18.6.
USE master;
BACKUP DATABASE Baza_3 TO Baza_3_Device WITH NAME = 'Baza_3_Asli', NOINIT;
18.7.
USE master;
BACKUP DATABASE Baza_3 TO Baza_3_Device WITH NAME = 'Baza_3_Asli', SKIP;
18.8.
USE master;
BACKUP DATABASE Baza_3 TO Baza_3_Device WITH NAME = 'Baza_3_Asli', NOSKIP;
18.9.
USE master;
BACKUP DATABASE Baza_3 TO Baza_3_Device
WITH NAME = 'Baza_3_Asli', INIT, RETAINDAYS = 15;
18.10.
USE master;
EXEC sp_addumpdevice 'DISK', 'Baza_31_Device',
' C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Backup \Baza_31_Asli.bak';
BACKUP DATABASE Baza_3 TO Baza_31_Device
WITH NAME = 'Baza_31_Asli', EXPIREDATE = '12.31.2009', INIT;
18.11.
USE master;
EXEC sp_addumpdevice 'DISK', 'Baza_32_Device',
' C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Backup \Baza_32_Asli.bak';
BACKUP DATABASE Baza_3 TO Baza_32_Device
WITH NAME = 'Baza_32_Asli';
GO
--
USE master;
BACKUP DATABASE Shekveta TO Shekveta_Asli
WITH DIFFERENTIAL, NAME = 'Shekvata_32_Dif';
18.12.
USE master;
RESTORE DATABASE Shekveta  FROM Shekveta_Asli;
18.13.
USE Master;
RESTORE DATABASE Shekveta  FROM Shekveta_Asli WITH REPLACE;
18.14.
USE Master;
RESTORE DATABASE Shekveta FROM Shekveta_Asli WITH NORECOVERY, REPLACE, FILE = 8;
RESTORE DATABASE Shekveta FROM Shekveta_Asli WITH FILE = 9;
```

გამოყენებული ლიტერატურა

1. SAMs Teach Yourself Database Programming with Visual C++ 6 in 21 Days. ttp://www.pbs.mcp.com/ebooks/0672313502/fm/fm.htm (1 of 3) [9/22/1999 1:42:57 AM]
2. А.В. Фролов, Г.В. Фролов. Базы Данных в Интернете.
3. Администрирование SQL Server 2000. Учебный курс MCSA/MCSE, MCDBA/Пер. с англ. — М.: Издательско_торговый дом «Русская Редакция», 2002. — 816 стр.: ил.
4. Основы реляционных баз данных. Пер. с англ. — М.: Издательско_торговый дом «Русская Редакция», 2001. - 384 стр.: ил.
5. Р. Саукап. Основы Microsoft SQL Server 6.5. Пер. с англ. — М.: Издательский отдел "Русская редакция" ТОО «Channel Trading Ltd.». - 1999. - 704 стр.: ил.
6. Реализация баз данных Microsoft SQL Server 7.0. Учебный курс.
Пер. с англ. — М.: Издательско_торговый дом «Русская Редакция», 2000. - 528 стр.: ил.
7. Малкольм Г. Программирование для Microsoft SQL Server 2000 с использованием XML. М.: Издательско_торговый дом «Русская Редакция», 2002. - 320 стр.: ил.
8. Проектирование и реализация баз данных. М.: Издательско_торговый дом «Русская Редакция», 2001. - 704 стр.: ил.
9. Г. Дейтел. Введение в операционные системы. В 2-х томах. Пер. с англ. - М.: Мир, 1987.
10. Э. Таненбаум. Современные операционные системы. - СПб.: Питер, 2002. - 1040 с.: ил.
11. А. Джонс. Руководство системного администратора Windows.
12. П. Нильсен. SQL Server 2005. Библия пользователя. Пер. с англ. - М.: ООО «И.Д. Вильямс», 2008. – 1232 с. : ил.

კომპიუტერული უზრუნველყოფა ლ. გაჩეჩილაძის

იბეჭდება ავტორების მიერ წარმოდგენილი სახით

გადაეცა წარმოებას 20.05.2016. ხელმოწერილია დასაბეჭდად 21.05.2016. ქაღალდის ზომა 60X84 1/8. პირობითი ნაბეჭდი თაბახი 28,13. ტირაჟი 100 ეგზ. შეკვეთა #

საგამომცემლო სახლი „ტექნიკური უნივერსიტეტი", თბილისი, კოსტავას 77