ლაბორატორია 4

ცხრილების კავშირები SQL სერვერზე:

რელაციური მონაცემთა ბაზის სწორად დასაპროექტებლად ძალიან მნიშვნელოვანია კარგად გაიაზროთ ცხრილებს შორის კავშირები. რელაციურ მონაცემთა ბაზაში, თითოეული ცხრილი დაკავშირებულია სხვა ცხრილთან პირველადი-გარე გასაღებების შეზღუდვების გამოყენებით.

პირველი რაც უნდა ვიცოდეთ კავშირების შესახებ არის ის რომ ცხრილთან დაკავშირება შესაძლებელია მხოლოდ იმ შემთხვევაში თუ ცხრილს უკვე აქვს პირველადი (Primary) გასაღები. როგორც ვიცით პირვლადი გასაღები აერთიანებს Unique და Not null შეზღუდვებს, თუმცა საბოლოოდ მას ბევრად მეტი მისია აკისრია.

ცხრილი რომელიც შეიცავს პირველად გასაღებს ერთგვარად გამოთქვავს „მზაობას კავშირისთვის“. თუ ცხრილში არ გვაქვს გასაღები ველი, მასთან დაკავშირება შეუძლებელია! პირველადი გასაღების(Primary Key) შექმნით კი შეიძლება ითქვას რომ ცხრილი ამბობს: „მე მაქვს უნიკალური, ინედქსირებული ჩანაწერების მქონე ველი, რომელშიც არ გვხვდება ცარიელი სტრიქონები, რომელიც იდეალურია ძებნისთვის. და თუ რომელიმე ცხრილს უნდა რომ დამიკავშირდეს, რელაცია უნდა მოხდეს სწორედ ამ ველზე და არცერთ სხვაზე.“

ცხრილი პირველადი გასაღებით ხდება კავშირის ძირითადი ცხრილი, ხოლო მეორე ცხრილი რომელიც აპირებს რომ მას დაუკავშირდეს ქმნის შესაბამის ველს, ადებს მას გარე (Foreign) გასაღებს და მისი საშუალებით უკავშირდება ძირითად ცხრილს. შედეგად იგი გახდება „დამოკიდებული“ ცხრილი, რომელიც შეზღუდულია კავშირის პირობების შესაბამისად.

ქვემოთ მოყვანილ დავალებაში გვაქვს რამოდენიმე ცხრილი. დავიწყოთ სტუდენტის ცხრილის დეტალიზებით. ვთქვათ სტუდენტის შესახებ გვაქვს შემდეგი ინფორმაცია:

- სახელი
- გვარი
- მეილი
- პაროლი
- პირადობის ნომერი
- ფოტო
- დაბადების თარიღი
- ტელეფონი
- მისამართი

სიის პირველი 4 პუნქტი მეტად მნიშვნელოვანი და გამოყენებადი ნაწილია ფაკულტეტის შიდა სისტემაში, დანარჩენი 4 კი წარმოადგენს დამატებით ინფორმაციას. მეტიც ფოტოების

ნაწილი შეიძლება მძიმე და ნელა ჩატვირთვადიც იყოს სიებისთვის. დავყოთ სტუდენტის ცხრილი ორ ნაწილად:

```
CREATE TABLE Students
(StudentId INT IDENTITY (1,1000) CONSTRAINT PK_Students PRIMARY KEY,
StudentName nvarchar(30) NOT NULL,
StudentLastName nvarchar(30) NOT NULL,
Email VARCHAR (30) CHECK (Email LIKE '%@%'),
DirectionId INT NOT NULL,
Password varbinary(250)
)

CREATE TABLE StudentDetails
(
StudentDetailId INT IDENTITY PRIMARY KEY,
PersonalId VARCHAR(11) UNIQUE,
Photo VARBINARY (MAX),
DateOfBirth DATE,
Phone VARCHAR (20),
City NVARCHAR(50) DEFAULT (N'თბილისი')
)
```

იმისთვის რომ კავშირის დამყარება შევძლოთ ორ ცხრილს შორის უნდა ვიცოდეთ:

1. რომელია ძირითადი ცხრილი, (რომელშიც გვაქვს საწყისი მონაცემი)
2. რომელია დამოკიდებული ცხრილი (რომელშიც იქნება ძირითად ცხრილზე დამოკიდებული მონაცემი)
3. გვაქვს თუ არა ძირითად ცხრილში პირველადი გასაღები(ვართ თუ არა მზად კავშირისთვის)
4. გვაქვს თუ არა შესაბამისი კავშირის ველი. წინააღმდეგ შემთვევაში უნდა შეიქმნას დამოკიდებულ ცხრილში „ველი კავშირისთვის“ რომელზეც დაედება გარე გასაღები

ზემოთ აღნიშულ მაგალითში ძირითადი ცხრილი იქნება Students. სტანდარტულად ძირთადი ცხრილი არსებობს დამოუკიდებლად და არ აწუხებს კავშირის ვალდებულება. ხოლო დამოკიდებული ცხრილი იქნება StudentDetails. ადვილი შესამჩნევია რომ ძირითადი ცხრილში იწერება ჩანაწერები და ივსება ისე რომ დამოკდებ ცხრილს არ აქვს ამ პროცესზე გავლენა, თუმცა დამოკიდებული ცხრილი მუდმივად უყურებს ძირითად ცხრილს, რომ ნამდვილად არსებობდეს ისეთი სტუდენტი, რომლის დეტალური ინფორმაციის შეტანაც გვინდა.

ასრულებს თუ არა ჩვენი სკრიპტი ზემოთ აღნიშნულ კრიტერიუმებს?

1. ძირითადი ცხრილია Students
2. დამოკიდებული ცხრილია StudentDetails
3. ძირითად ცხრილს აქვს გასაღები StudentId ველზე და მზადაა კავშირისთვის
4. მაგრამ დამოკიდებულ ცხრილში არ არის ველი კავშირისთვის??? როდესაც ცხრილი გავყავით არ გვიფიქრია როგორ უნდა მიხვდეს StudentDetails ცხრილი თუ რომელი სტუდენტის დეტალურ ინფორმაციას ატარებს მისი შესაბამისი

სტრიქონები. აუცილებელია დამოკიდებულ ცხრილშიც იყდეს StudentId ველი, რითაც ცალსახად გასაგები იქება რომელი სტუდენტის დეტალურ ინფორმაციაზეა საუბარი:

```
ALTER TABLE StudentDetails
ADD StudentId INT
```

ამ ეტაპზე ჩვენ უკვე გვაქვს კვაშირის ველი დამოკიდებულ ცხრილში. ველი რომლიც შეიცავს ჩანაწერის ნომრებს ძირითადი ცხრილიდან. შესაძლებელია დაიწეროს ჩვენი პიველი კავშირის სკრიპტი:

როგორც უკვე ვთქვით ამისათვის გვჭირდება ორი გასაღები პირველადი (Primary Key) ძირითად ცხრილში და გარე (Foerign Key) დამოკიდებულ ცხრილში. გარე გასაღების დამატების მომენტში იქმნება კავშირი აღნიშნულ ცხრილებს შორის. შესაბამისად შეიძლება ითქვას რომ კავშირი იქმნება Foreign Key შეზღუდვის დადების მომენტში:

```
ALTER TABLE StudentDetails
    ADD CONSTRAINT FK_StudentDetails_Students FOREIGN KEY (StudentId) REFERENCES
    Students(StudentId)
```

შეზღუდვა სახელად FK_StudentDetails_Students ამბობს რომ ცხრილში StudentDetails დაემატა ახალი შეზღუდვა FOREIGN KEY ველზე StudentId და გადაება Students ცხრილს StudentId ველზე (რომელიც აუცილებლად წარმოადგენს პირველად გასაღებს)

შედეგად დამოკიდებული გახდა ცხრილი რომლის კოდშიც ჩაჯდა FOREIGN KEY შეზღუდვა.

რა შედეგი მოუტანა ცხრილებს ამ შეზღუდვამ?

1. დამოკიდებულ ცხრილში ვერ დაემატება ისეთი ჩანაწერი(ჩვენს შემთხვევაში StudentId) რომელიც არ არსებობს ძირითად ცხრილში (ვერც დამატების და ვერ განახლების დროს)
2. ძირითადი ცხრილიდან ვერ წაიშლება/განახლდება ის ჩანაწერი რომელზეც არის კავშირის შედეგად დამოკიდებული ველი.(თუმცა აღნიშნული პარამეტრი ცვლილება შესაძლებელი იქნება ქვემოთ).

MSSQL Server-ის მონაცემთა ბაზაში გვხვდება ცხრილებს შორის სამი ტიპის კავშირი:

1. 1:1 ერთი-ერთან
2. 1:N ერთი-ბევრთან

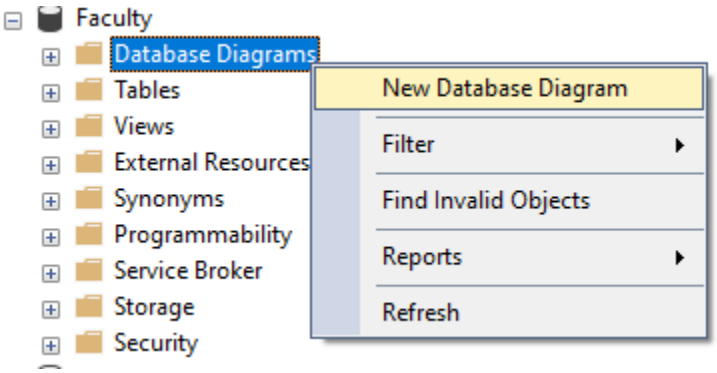
3. N:M ბევრი-ბევრთან

ერთი-ერთან კავშირი

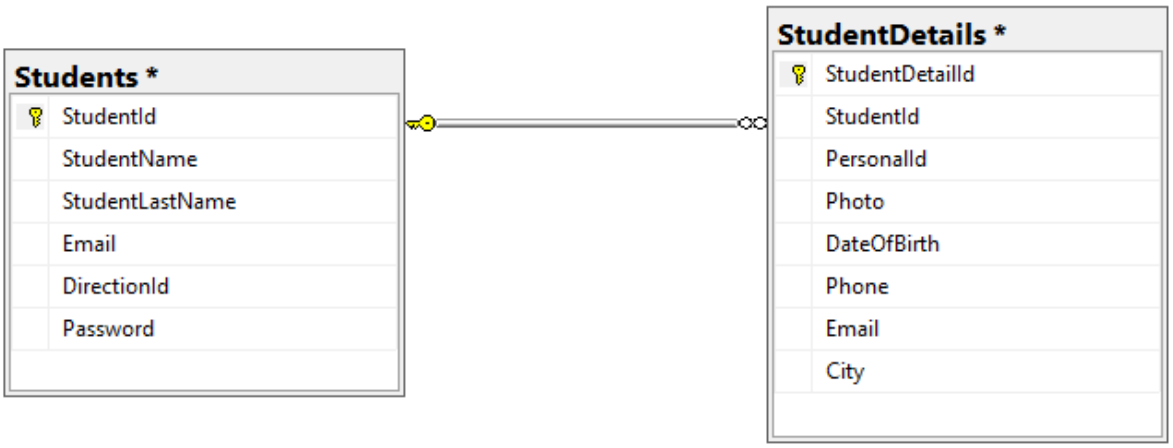
ერთი-ერთთან კავშირის დროს ერთი ცხრილის ერთი ჩანაწერი იქნება დაკავშირებული სხვა ცხრილის ერთ ან არცერთ ჩანაწერთან. მაგალითად, Students ცხრილის თითოეულ ჩანაწერს შეიძლება ჰქონდეს StudentDetails ცხრილში შესაბამისი სტრიქონი, რომელიც ინახავს ამ კონკრეტული სტუდენტის შესახებ დამატებით ინფორმაციას ასე რომ, თითოეულ სტუდენტს ექნება ნული ან ერთი ჩანაწერი ცხრილში StudentDetails. აღნიშნული ტიპის რელაციას ეწოდება ნულოვანი ან ერთი ერთზე კავშირი

*კავშირების ვიზუალურად უკეთ აღსაქმელად შეგიძლიათ გამოიყენოთ ბაზის დიაგრამა:

ჩამოშალეთ Faculty ბაზა, ჩანართ Database diagrams დააჭირეთ მარჯვენა ღილაკით და კონტექსტური მენიუდან აარჩიეთ New Database diagram



შემდგომ ახალ შექმნილ დიაგრამაზე დაამატეთ ცხრილები Students და StudentDetails. დამატებით ცხრილების ჩამატება დიაგრამაში შესაძლებელია მარჯვენა ღილაკის დაჭრით და Add Table ბრძანების არჩევით. შედეგად მიიღებთ მსგავს სქემას:

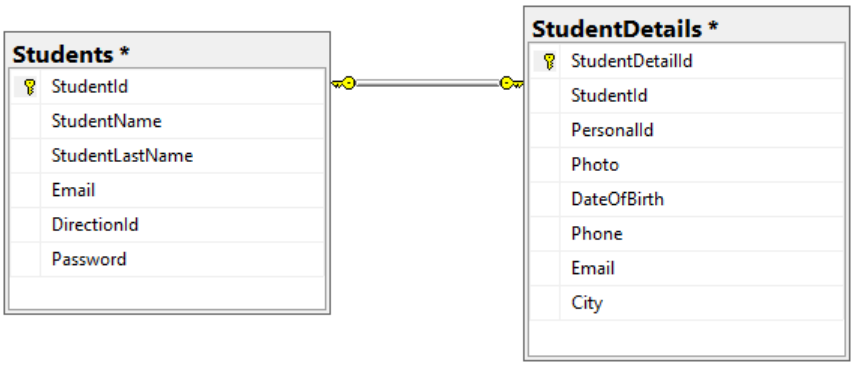


ზემოთ, StudentId სვეტი არის პირველადი (Primary) გასაღები ცხრილში Students, ხოლო StudentDetails ცხრილში წარმოადგენს გარე (Foreign) გასაღების სვეტს. მისი საშუალებით StudentDetails ცხრილი დაკავშირებულია Students ცხრილის შესაბამის ჩანაწერთან.

თუმცაღა თუ დავაკვირდებით სურათს გასაღების აღნიშვნა გვხვდება მხოლოდ სტუდენტის ცხრილის მხარეს, სადაც ველი StudentId წარმოადგენს პირველად გასაღებს და შესაბამისად უნიკალურ ველს, რაც იძლევა გარანტიას რომ სტუდენტი ნომერი 352 აღნიშნულ ცხრილში შეგვხვდება მხოლოდ ერთხელ, რაც შეეხება StudentDetails ცხრილს ამჯერად StudentId სვეტი არც უნიკალურია და არც გასაღები შესაბამისად მას არ ადევს შეზღუდვა რომელიც იქნებოდა მისი უნიკალურობის გარანტი. მარტივად რომ ვთქვათ სტუდენტი ნომერი 352-ის დეტალური ინფორმაცია StudentDetails ცხრილში შეიძლება ჩაიწეროს რამოდენიმეჯერ. 1:1 კავშირის ლოგიკიდან გამომდინარე კი საჭიროა რომ საწყისი ცხრილიდან ერთ ჩანაწერს შეესაბამებოდეს მაქსიმუმ 1 ჩანაწერი კავშირის მეორე ცხრილიდან, საჭიროა სტუდენტ ნომერი 352-ის შესაბამისი ჩანაწერი არ მეორდებოდეს არც StudentDetails ცხრილში.

დავადოთ უნიკალურობის შეზღუდვა StudentId ველს StudentDetails ცხრილში, რაც იქნება გარანტი რომ ჩანაწერები StudentDetails-ს ცხრილშიც უნიკალური იქნება თითოეული სტუდენტისთვის. (

```
ALTER TABLE [dbo].[StudentDetails]
ADD CONSTRAINT UQ_StudentDetails UNIQUE (StudentId)
```



თუ კავშირის ერთ მხარეს იქნება გასაღები ველი, ხოლო მეორე მხარეს უნიკალური ველი, მივიღებთ კავშირს 1:1 რომელიც დიაგრამაზე გამოსახულია ორივე მხარეს გასაღები აღნიშვნით.

ერთი-ბევრთან კავშირი

აღნიშნული ტიპის კავშირები ყველაზე გავრცელებულია ბაზაში.

ერთი ბევრთან კავშირის დროს ძირითადი ცხრილის 1 ჩანაწერზე შესაძლებელია დამოკიდებულ ცხრილში მიბმული გვქონდეს ბევრი სტრიქონი.

დავაკვირდეთ: საწყისად შექმნილი კავშრი (მანამ სანამ დავადებდით უნიკალურობის შეზღუდვას) საშუალებას იძლეოდა რომ ერთ სტუდენტზე გვქონოდა რამოდენიმე დეტალური ინფორმაცია... რითაც იგი ქმნიდა 1:N (ერთი ბევრთან) კავშირს. გამოდის რომ ზედმეტი ძალისხმევის გარეშე როდესაც სტანდარტული(არაუნიკალური) ველი დამოკიდებული ცხრილიდან უკავშირდება გასაღებ (უნიკალურ) ველს ძირითად ცხრილში ვიღებთ ერთი-ბევრთან კავშირს.

მაგალითად განვიხილოთ მიმართულების ცხრილი:

```
CREATE TABLE Directions
(DirectionId INT NOT NULL PRIMARY KEY IDENTITY,
DirectionName NVARCHAR(150) NOT NULL UNIQUE ,
DirectionHead int NULL
);
```

ცხრილი რომელიც ინახავს ფაკულტეტზე არსებულ მიმართულებებს.

თუმცაღა თუ დაკვირდებით სტუდენტის ცხრილში ასევე გვხვდება DirectionId სვეტი. მარტივი მისახვედრია რომ ეს წარმოადგენს საშუალებას ვიცოდეთ რომელ მიმართულებაზეა სტუდენტი. შესაბამისად ყოველი სტუდენტისთვის გვინდა რეგისტრაციისთანავე ჩავინიშნოთ მისი მიმართულების ნომერი. (ამოვარჩოთ მიმართულებების სიიდან) წარმოიდგინეთ რომ სტუდენტის ცხრილი არ იყოს შეზღუდული და უფლებას იძლეოდეს მიმართულების ნებისმიერი ნომრის ჩაწერის? მაშინ შესაძლებელია სტუდენტებს მიმართულების ნომერში ეწეროს ნომერი 7, ჩვენ კი მხოლოდ 3 მიმართულება გვქონდეს ბაზაში? ან საერთოდ წაიშალოს მიმართულება რომელზეც 200 სტუდენტი გვყავს დარეგისტრირებული. სწორედ მსგავსი შეცდომებისგან გვიცავს გასაღები ველები.

დავამატოთ კავშირი სტუდენტსა და მიმართულებას შორის. ამისათვის თავიდან ვუპასუხოთ წინა კავშირის შემთხვევაში დასმულ კითხვებს:

1. რომელია ძირითადი ცხრილი, (რომელშიც გვაქვს საწყისი მონაცემი)
    - ეს არის მიმართულების ცხრილი ვინაიდან მასში ინახება სია, და სწორედ მიმართულების ცხრილში არსებული მიმართულებებიდან უნდა ამოირჩეს სტუდენტებისთვის ვალიდური მიმართულების ნომერი.
2. რომელია დამოკიდებული ცხრილი (რომელშიც იქნება ძირითად ცხრილზე დამოკიდებული მონაცემი)
    - სტუდენტის ცხრილი. ვინაიდან სტუდენტამდე უნდა არსებობდეს მიმართულება.
3. გვაქვს თუ არა ძირითად ცხრილში პირველადი გასაღები(ვართ თუ არა მზად კავშირისთვის)

- დიახ, ძირითად, მიმართულების ცხრილს აქვს `DirectionId INT PRIMARY KEY`

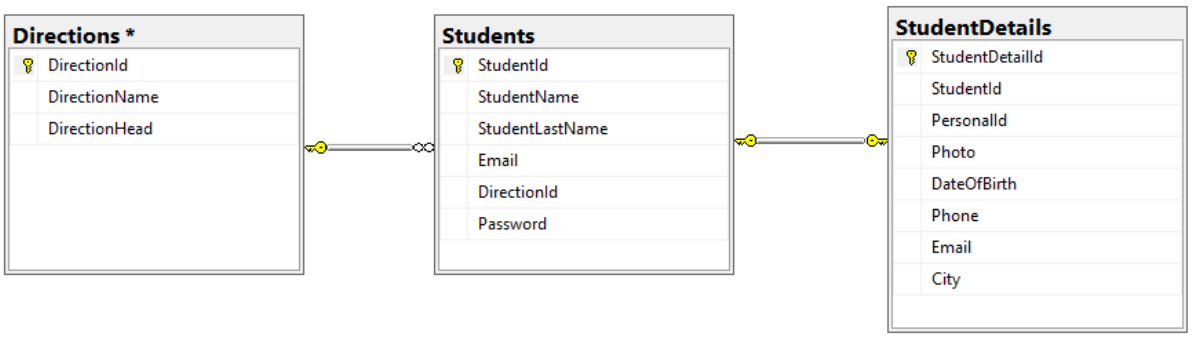
4. გვაქვს თუ არა შესაბამისი კავშირის ველი. წინააღმდეგ შემთვევაში უნდა შეიქმნას დამოკიდებულ ცხრილში „ველი კავშირისთვის“ რომელზეც დაედება გარე გასაღები
    - სტუდენტის ცხრილში ასევე გვაქვს კავშირის ველი `DirectionId INT` რომელზეც მოხდება გარე გასარების შეზღუდვის დადება. რათა სტუდენტის ცხრილში არ მოხვდეს არავალიდური მიმართულების ნომერი.

თუკი ყველა კითხვაზე გვაქვს დამაკმაყოფილებელი პასუხი შესაძლებელია შევქნათ შეზღუდვა:

```
ALTER TABLE [dbo].[Students]
ADD CONSTRAINT FK_Student_Directions FOREIGN KEY ([DirectionId]) REFERENCES
[dbo].[Directions]([DirectionId])
```

სკრიპტის გაშვების შემდეგ, არაუნიკალური ველი სტუდენტების ცხრილიდან დაუკავშირდა გასაღებ ველს მიმართლებებში, შედეგად თითოეულ სტუდენტს შესაძლებელია ამორჩეული ჰქონდეს **ერთი** მიმართულება მიმართულებების ცხრილიდან, ხოლო ერთი მიმართულება შესაძლებელია არჩეული ჰქონდეს **ბევრ** სტუდენტს.

შედეგად მივიღეთ ერთი ბევრთან კავშირი, რომელიც ასევე დავამატეთ დიაგრამაზე:



დავამატოთ საგნების ცხრილი:

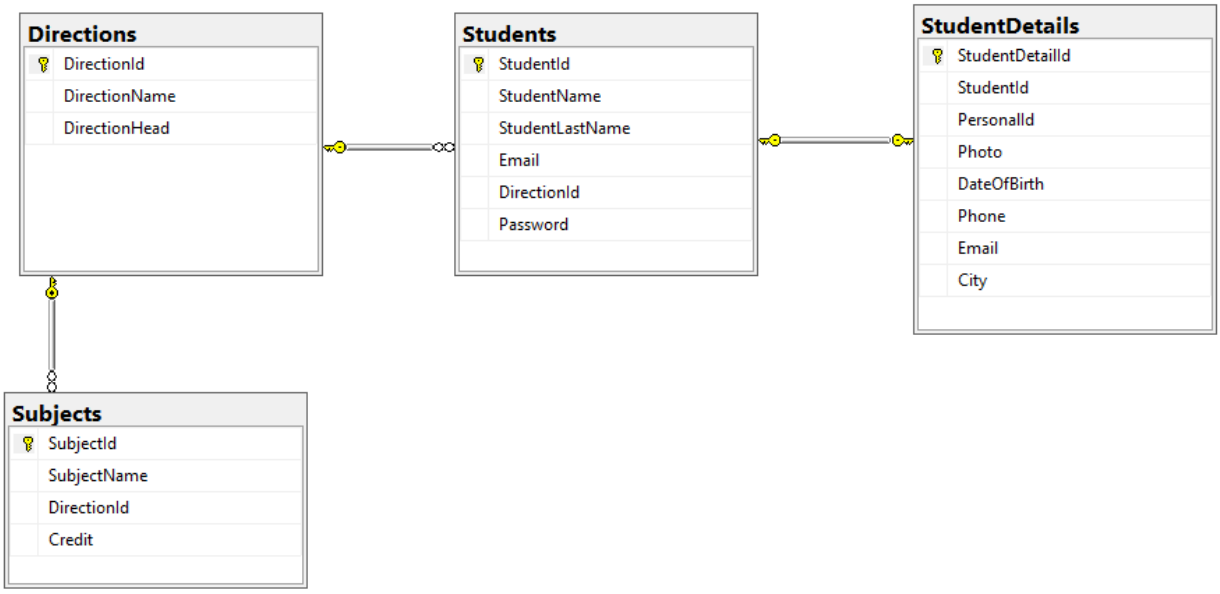
საჭირო ველები იქნება:

ნომერი, დასახელება, კრედიტების რაოდენობა და მიმართულების ნომერი რომელსაც ეკუთვნის საგანი, აქვე გასაგებია რომ მიმართულების ნომერი უნდა იყოს დამოკიდებული მიმართულებების ცხრილზე, შესაბამისად შექმნისთანავე შეგვიძლია გარე გასარების სკრიპტის გათვალისწინება:

```
CREATE TABLE Subjects
(
SubjectId INT IDENTITY (1,1) PRIMARY KEY ,
SubjectName NVARCHAR(150) NOT NULL,
DirectionId INT NOT NULL FOREIGN KEY REFERENCES [dbo].[Directions]([DirectionId]),
Credit INT,
```

)

შედეგად ბაზაში დაგვემატება საგნების ცხრილი, რომელიც დაკავშირებულია მიმართულების ცხრილთან კავშირით ერთი-ბევრთან, ვინაიდან თითოეული საგანი ეკუთვნის **ერთ** მიმართულებას, ხოლო ერთ მიმართულებაზე შეიძლება იყოს **ბევრი** საგანი:



ბევრი-ბევრთან კავშირი

როგორც უკვე ვნახეთ გარე გასაღების (Foreign key) შეზღუდვით შესაძლებელია შეიქმნას 1:1 ან 1:N კავშირები, ვინდაინდან კავშირის ცალ მხარეს ყოველთვის დგას პირველადი გასაღები, უნიკალური ველი (primary key) ანუ 1. დამოკიდებული ცხრილის მხარეს კი იმის მიხედვით თუ იდგება უნიკალური ველი, იქაც გვექნება 1 და გამოვა კავშირი 1:1 თუ არადა სტანდარტული არაუნიკალური ველის შემთქვევაში გვექნება N და გამოვა კავშირი 1:N.

ვფიქრობ ამ ეტაპზე უკვე გაგიჩნდებოდათ კითხვა: „როგორ შევქმნათ ამ მოცემულობით კავშირი N:M(ბევრი-ბევრთან), თუ ცალ მხარეს ყოველთვის 1 -ია?“

პასუხი მარტივია, ერთი Foreign key შეზღუდვით, მსგავსი ტიპის კავშირის შექმნა შეუძლებლია! ამისათვის საჭიროა ორი 1:N და N:1 კავშირი.

მაგალითისთვის დავამატოთ კავშირი საგანსა და სტუდენტებს შორის, პირველ რიგში განვსაზსღვროთ როგორი ტიპის კავშირი იქნება იგი:

- ერთი სტუდენტი რეგისტრირდება **ბევრ** საგანზე
- ერთ საგანზე რეგისტრირდება **ბევრი** სტუდენტი

ანუ დასაკავშირებელი გვაქვს ბევრი სტუდენტი და ბევრი საგანი, და სჭიროა შეიქმნას ჩვენი პირველი ბევრი-ბევრთან კავშირი

სად შევინახოთ აღნიშნული ინფორმაცია, ფაქტია რომ წინასწარ არ ვიცით რამდენ საგანზე შეიძლება დარეგისტრირდეს სტუდენტი და რომც ვიცოდეთ ფიზიკურად შეუძლებელია სტუდენტის ცხრილმა დაიტიოს ეს ინფორმაცია, ანალოგიური შეიძლება ითქვას საგნების ცხრილზე, ვინაიდან ვერც ის შეინახავს ინფორმაციას მასზე დარეგისტრირებული ყველა სტუდენტის შესახებ. საჭიროა შეიქმნას **ახალი ცხრილი.** ახალ ცხრილში ჩაიწერება სტუდენტისა და საგნის წყვილები:

სტუდენტი ნომერი 1 დარეგისტრირდა საგან ნომერ 2-ზე

სტუდენტი ნომერი 3 დარეგისტრირდა საგან ნომერ 2-ზე

სტუდენტი ნომერი 4 დარეგისტრირდა საგან ნომერ 2-ზე

სტუდენტი ნომერი 1 დარეგისტრირდა საგან ნომერ 3-ზე

სტუდენტი ნომერი 3 დარეგისტრირდა საგან ნომერ 3-ზე და ა.შ

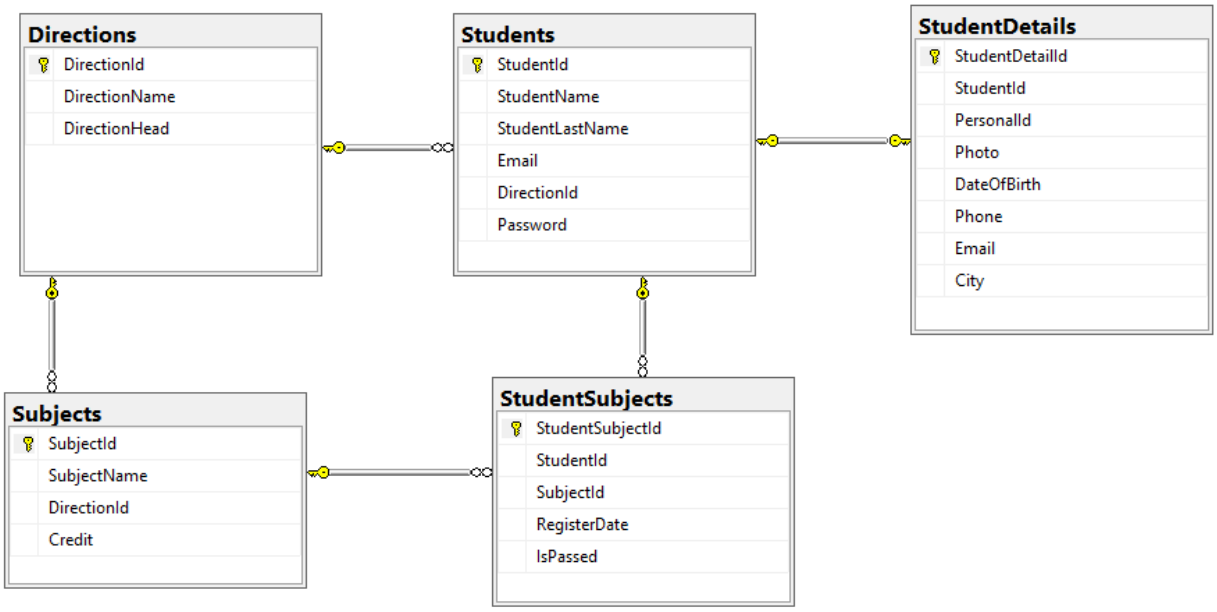
მარტივად რომ ვთქვათ ცხრილი რომელიც შეინახავს სტუდენტებისა და საგნების წყვილების ნომრებს:

სჭირო ველები იქნება:

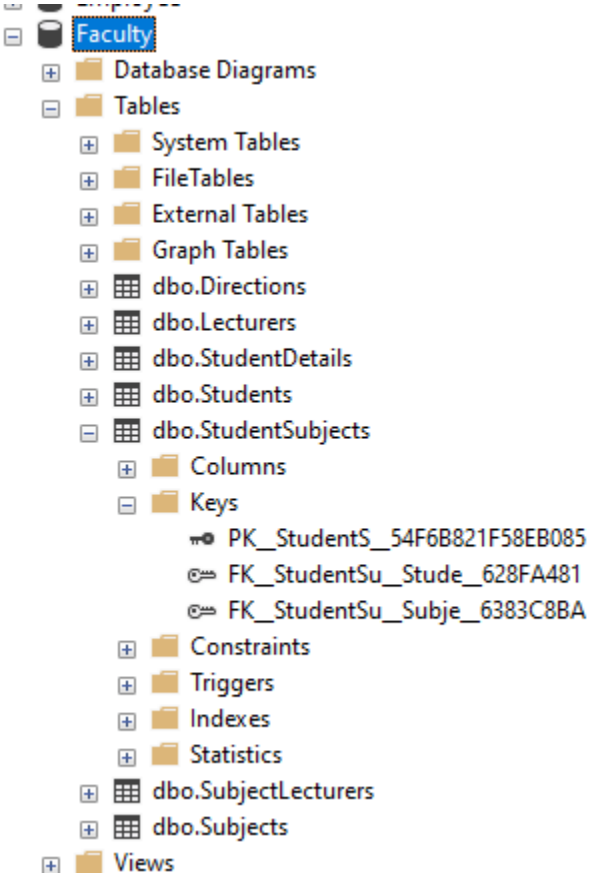
- საკუთარი უნიკალური საძიებო გასაღები ველი(სასურველია)
- სტუდენტის აიდი, რომელიც დაკავშირებულია სტუდენტის ცხრილთან, რათა არ მოხდეს არარსებული სტუდენტის ნომრის ჩაწერა.
- საგნის აიდი, რომელიც დაკავშირებულია საგნის ცხრილთან, რათა არ მოხდეს არარსებული საგნის ნომრის ჩაწერა.
- თუ კი კავშირს აქვს ატრიბუტები, მაგალითად როდის მოხდა რეგისტრაცია, ან წარმატებით ჩააბარა სტუდენტმა საგანი თუ არა, შესაძლებელია დამატებით ველების ჩასმაც.

```
CREATE TABLE StudentSubjects
(StudentSubjectId INT IDENTITY PRIMARY KEY,
StudentId INT NOT NULL FOREIGN KEY REFERENCES [dbo].[Students]([StudentId]),
SubjectId INT NOT NULL FOREIGN KEY REFERENCES [dbo].[Subjects]([SubjectId]),
RegisterDate DATETIME NOT NULL DEFAULT (GETDATE()),
IsPassed BIT NOT NULL DEFAULT(0)
)
```

ვიღებთ კავშირის ცხრილს, რომიც ამოცნობაც ადვილია ორი გარე გასაღების შეზღუდვის საშუალებით:



Object Explorer ფანჯარაში შესაძლებელია თითოეული ცხრილის გასაღები ველების, და მათი დასახელებების ნახვა:



თუ საჭირო გახდება შეზღუდვის წაშლა, წინა ლექციაში განხილული ბრძანებით Drop Constraint, რომლის აუცილებელი პარამეტრია შეზღუდვის სახელი, ჩვენ კი არ განგვისაზღვრავს აღნიშნული კავშირების სახელები, აუცილებლად დაგვჭირდება სისტემის მიერ დაგენერირებული სახელების ნახვა.

```
ALTER TABLE [dbo].[StudentSubjects]
 DROP CONSTRAINT [PK__StudentS__54F6B821F58EB085]
```

იქნება სკრიპტი, ძირითადი გასაღების შეზღუდვის წაშლის სურვილის შემთხვევაში.

გარე გასაღების (Foreign key) შეზღუდვის პარამეტრები:

როგორც უკვე აღვნიშნეთ გარე გასაღების (Foreign key) შეზღუდვა ზღუდავს არამარტო დამოკიდებულ ცხრილს არავალიდური მონაცემების შეტანა/განახლებისგან, არამედ შეზღუდვა ედება ძირითად ცხრილსაც, მას აღარ შეუძლია წაშალოს/განაახლოს ველი რომელზეც აქვს დამოკიდებულება. მაგალითად ვერ წავშლით მიმართულებას თუ მასზე მიბმული გვაქვს სტუდენტები ან საგნები, მეტიც ვერ შევცვლით ამ მიმართულების აიდის ნომერს, ვინაიდან თუ მიმართულების ნომერი იყო 3 და ყველა სტუდენტს და საგანს მითითებული აქვს რომ მე-3 მიმართულებას ეკუთვნის, შეუძლებელია ამ მიმართულების ნომერი გადავაკეთოთ 7-ად.

თუმცა ეს მხოლოდ იმიტომ რომ აღნიშნულ კავშირს აქვს ორი პარამეტრი

- `ON UPDATE` როოგრც მოიქცეს დამოკიდეული ცხრილი თუ ძირითად ცხრილში ველის ნომერი ახლდება?
- `ON DELETE` როოგრც მოიქცეს დამოკიდეული ცხრილი თუ ძირითად ცხრილში ველის ნომერი იშლება?

ორივე პარამეტრის მნიშვნელობა გაჩუმების პრინციპით არის `NO ACTION` , ანუ აღნიშნულ თავში, ყველგან სადაც დავწერეთ Foreign key შეზღუდვა და არ მივუთითეთ აღნიშნული პარამერები ყველგან იგულისხმებოდა:

```
StudentId INT NOT NULL FOREIGN KEY REFERENCES [dbo].[Students]([StudentId]) ON UPDATE  NO
ACTION ON DELETE  NO ACTION
```

რაც ნიშნავს რომ დამოკიდებულების არსებობის შემთხვევაში ძირითად ცხრილს ეზღუდებოდა როგორც წაშლა ის განახლება.

გარდა აღნიშნულისა პარამეტრები შეიძლება იყოს

- `ON UPDATE CASCADE ON DELETE CASCADE`-კასკადური წაშლა, თუ წაიშლება მიმართულება ძირითად ცხრილში წაიშლება ყველა მასზე მიბმული საგანი/ სტუდენტი. ანალოგიურად თუ განახლდება მიმართულების ნომერი, განახლდება ყველა დამოკიდებულ ცხრილში შესაბამისი ჩანაწერები.

- `ON UPDATE SET NULL ON DELETE SET NULL`- ნულით შევსება, თუ წაიშლება ან განახლდება მიმართულება ძირითად ცხრილში ყველა მასზე მიბმულ საგანის თუ სტუდენტის ჩანაწერს მიმართულების აიდის ველში ჩაეწერება NULL ცარიელი მნიშვნელობა.
- `ON UPDATE SET DEFAULT ON DELETE SET DEFAULT`- გაჩუმების პრინციპით შევსება, თუ წაიშლება ან განახლდება მიმართულება ძირითად ცხრილში ყველა მასზე მიბმულ საგანის თუ სტუდენტის ჩანაწერს მიმართულების აიდის ველში ჩაეწერება მათთვის წინასწარ გაჩუმების პრინციპით გათვლილი მნიშვნელობა.

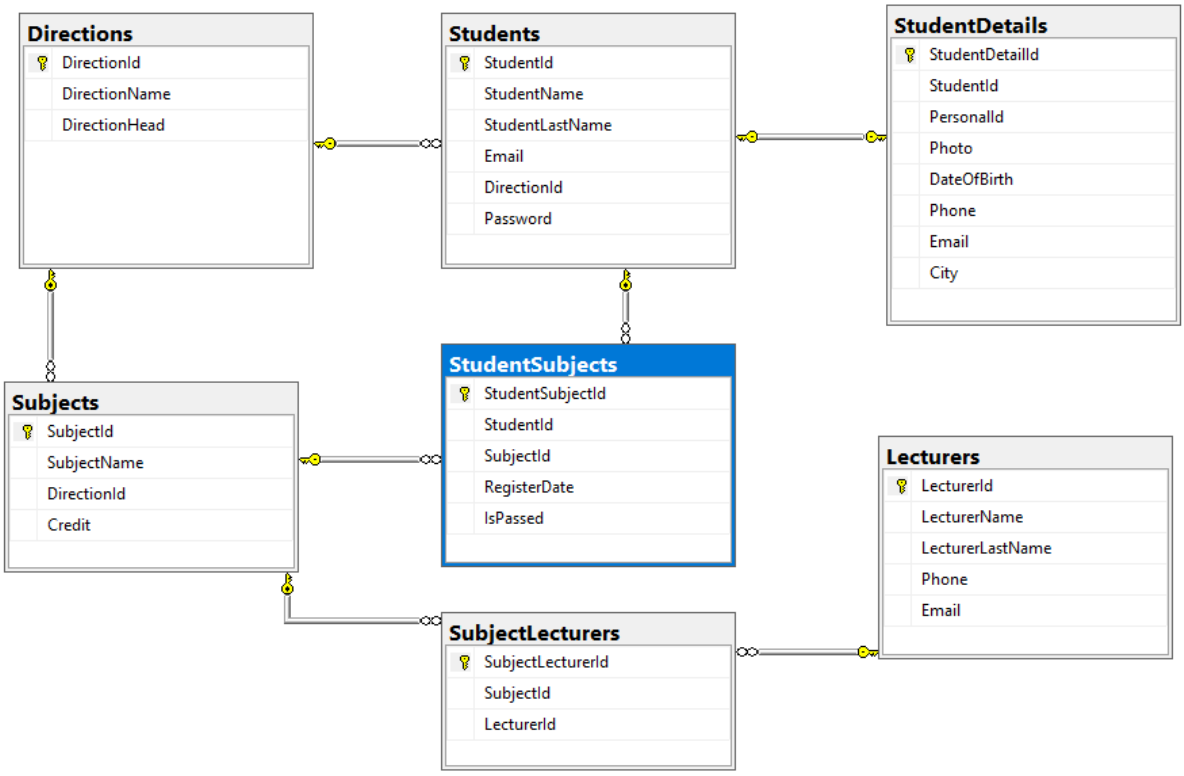
დავამატოთ ლექტორის ცხრილი:

```
CREATE TABLE Lecturers
(LecturerId INT IDENTITY PRIMARY KEY ,
LecturerName nvarchar(30) NOT NULL,
LecturerLastName nvarchar(30) NOT NULL,
Phone VARCHAR (20),
Email VARCHAR (30) CHECK (Email LIKE '%@%')
)
```

დავაკავშიროთ ლექტორები საგნებთან, ბევრი ბევრთან კავშირით, ვინაიდან ერთი ლექტორი შეიძლება კითხულობდეს **ბევრ** საგანს, ასევე ერთ საგანს შეიძლება ჰყავდეს **ბევრი** ლექტორი. როგორც უკვე ვიცით გვჭირდება კავშირის შუალედური ცხრილი, ორი გარე გასაღებით. თუმცა ამჯერად გავითვალისწინოთ კავშირის პარამეტრები, ვინიდან პარამეტრების გაწერის გარეშე ვერ წავშლით ლექტორს თუ კი მასზე დაკავშირებულია საგანი და პირიქით. ანუ კავშირის ცხრილი შეგვიშლის განახლებისა და წაშლის ოპერაციებში. თუკი წინასწარ გავთვლით კასკადურ წაშლას და განახლებას ლექტორის წაშლის შემთხვევაში ავტომატურად ჩაიხსნება კავშირი საგანთან, ასევე მაგალითად საგნის ნომრის განახლების შემთხვევაში ავტომატურად განხლდება კავშირის ცხრილებში შესაბამისი საგნის ნომერი. დამატებით გავითვალისწინოთ შედგენილი უნიკალურობის შეზღუდვა, რათა ერთი და იგივე ლექტორი რამდენჯერმე არ დაემატოს ერთსა და იმავე საგანზე. მივიღებთ სკრიპტს:

```
CREATE TABLE SubjectLecturers
(SubjectLecturerId INT IDENTITY PRIMARY KEY,
SubjectId INT NOT NULL FOREIGN KEY REFERENCES [dbo].[Subjects]([SubjectId])
ON UPDATE CASCADE ON DELETE CASCADE,
LecturerId INT NOT NULL FOREIGN KEY REFERENCES [dbo].[Lecturers]([LecturerId])
ON UPDATE CASCADE ON DELETE CASCADE,
CONSTRAINT UQ_SubjectLecturers UNIQUE(SubjectId,LecturerId)
)
```

ხოლო საბოლოო დიაგრამას ექნება სახე:


Directions
DirectionId
DirectionName
DirectionHead
Students
StudentId
StudentName
StudentLastName
Email
DirectionId
Password
StudentDetails
StudentDetailId
StudentId
PersonalId
Photo
DateOfBirth
Phone
Email
City
Subjects
SubjectId
SubjectName
DirectionId
Credit
StudentSubjects
StudentSubjectId
StudentId
SubjectId
RegisterDate
IsPassed
Lecturers
LecturerId
LecturerName
LecturerLastName
Phone
Email
SubjectLecturers
SubjectLecturerId
SubjectId
LecturerId