

## 1) ქვემოთ მოცემული გამონათქვამებიდან რომელი შეესაბამება ნახატს

ქვემოთ მოცემული გამონათქვამებიდან  
რომელი შეესაბამება ნახატს



აირჩიეთ ერთი:

- a. თითოეული პაციენტის ისტორია ბევრ პაციენტს ეკუთვნის.
- b. თითოეული პაციენტის ისტორია ნულოვან და ერთ პაციენტს ეკუთვნის.
- c. თითოეულ პაციენტს აქვს ერთი და მხოლოდ ერთი ვიზიტი.
- d. თითოეულ პაციენტს აქვს ერთი ან მეტი პაციენტის ისტორია.

სწორი პასუხია: თითოეულ პაციენტს  
აქვს ერთი ან მეტი პაციენტის  
ისტორია.

## 2) რომელ კონსტრუქციაშია შესაძლებელი HAVING- ის გამოყენება?

რომელ კონსტრუქციაშია შესაძლებელი HAVING-  
ის გამოყენება?

აირჩიეთ ერთი:

- a. INSERT
- b. DELETE
- c. JOIN
- d. SELECT

სწორი პასუხია: SELECT

3) რომელი მოქმედების შესრულებაა შეუძლებელი ALTER DATABASE ბრძანებით?

რომელი მოქმედების შესრულებაა შეუძლებელი  
ALTER DATABASE ბრძანებით?

აირჩიეთ ერთი:

- a. მონაცემთა ბაზის წაშლა;
- b. ბაზისათვის ახალი ფაილური ჰგუფის დამატება
- c. ბაზის ატრიბუტების შეცვლა
- d. ბაზის ფაილების მოდიფიკაცია

სწორი პასუხია: მონაცემთა ბაზის წაშლა;

4) ჩამოთვლილთაგან რომელი არ არის კარგი ბიზნეს წესის მახასიათებელი?

ჩამოთვლილთაგან რომელი არ არის კარგი  
ბიზნეს წესის მახასიათებელი?

აირჩიეთ ერთი:

- a. გამომხატველობითი
- b. ატომური
- c. დეკლარაციული
- d. არათანმიმდევრული

სწორი პასუხია: არათანმიმდევრული

5) რომელი კონსტრუქცია გამოიყენება მონაცემთა ბაზის ცხრილში მონაცემის ტიპის შესაცვლელად?

რომელი კონსტრუქცია გამოიყენება მონაცემთა  
ბაზის ცხრილში მონაცემის ტიპის  
შესაცვლელად?

აირჩიეთ ერთი:

- a. ALTER TABLE...ADD DATA
- b. ALTER TABLE ... ALTER COLUMN
- c. MODIFY COLUMN
- d. CHANGE TABLE ... MODIFY

სწორი პასუხია: ALTER TABLE ... ALTER COLUMN

6) რა ფუნქცია აქვს DISTINCT წინადადებას?

რა ფუნქცია აქვს DISTINCT წინადადებას?

აირჩიეთ ერთი:

- a. აკონკრეტებს ძებნის პირობას ჰგუთური ფუნქციის შედეგში;
- b. გამოიყენება რამდენიმე ცხრილის შესაერთებლად;
- c. გამოიყენება მონაცემების დალაგებისათვის
- d. გამოიყენება განსხვავებული მონაცემების დასაბრუნებლად;

სწორი პასუხია: გამოიყენება განსხვავებული მონაცემების დასაბრუნებლად;

7) ჩამოთვლილთაგან, რომელი ბაზის „ბექაპის“ გაკეთებაა საჭირო რეგულარულად?

ჩამოთვლილთაგან, რომელი ბაზის „ბექაპის“ გაკეთებაა საჭირო რეგულარულად?

აირჩიეთ ერთი:

- a. RESOURCE
- b. MASTER
- c. MODEL
- d. TEMPDB

სწორი პასუხია: MASTER

8) რომელი მოთხოვნა დააბრუნებს შეცდომას?

- რომელი მოთხოვნა დააბრუნებს შეცდომას?
- აირჩიეთ ერთი:
- a. select top(5) with ties salary from corporation  
order by salary desc
  - b. select top(5) with ties salary from corporation
  - c. select f\_name from corporation order by f\_name  
asc  
offset 5 row
  - d. select top (7) salary  
from emp  
order by salary desc

სწორი პასუხია: select top(5) with ties salary from corporation

9) ჩამოთვლილი ფუნქციებიდან, რომელი მიეკუთვნება ჯგუფურ (აგრეგატულ) ფუნქციებს ?

- ჩამოთვლილი ფუნქციებიდან, რომელი  
მიეკუთვნება ჯგუფურ (აგრეგატულ) ფუნქციებს?
- აირჩიეთ ერთი:
- a. Join
  - b. Len
  - c. Avg
  - d. Left

სწორი პასუხია: Avg

## 10) რომელი მტკიცებაა არასწორი ?

რომელი მტკიცებაა არასწორი?

აირჩიეთ ერთი:

- a. სქემა განსაზღვრავს ცხრილებს შორის კავშირებს.
- b. ერთ სქემას ჰყავს ერთი მფლობელი;
- c. სქემა წარმოადგენს სახელდებულ სივრცეს ბაზის ობიექტების კოლექციის.
- d. შეუძლებელია ერთ სქემაში ორ სხვადასხვა ცხრილს ჰქონდეს ერთნაირი სახელები;

სწორი პასუხია: სქემა განსაზღვრავს ცხრილებს შორის კავშირებს.

## 11) რომელი კოდი დააბრუნებს სწორ შედეგს მოცემული მოთხოვნისათვის : დააბრუნეთ ყველა 'romance' ჟანრის ფილმი რომელსაც im\_rating აქვს 6-ზე მეტი

რომელი კოდი დააბრუნებს სწორ შედეგს  
მოცემული მოთხოვნისათვის: დააბრუნეთ ყველა  
'romance' ჟანრის ფილმი რომელსაც im\_rating აქვს  
6-ზე მეტი

აირჩიეთ ერთი:

- a. SELECT \* FROM movies WHERE genre = 'romance'  
AND im\_rating >6
- b. SELECT \* FROM movies HAVING genre =  
'romance' AND im\_rating <6
- c. SELECT im\_rating, name FROM movie WHERE  
genre = 'romance' AND HAVING im\_rating <>6
- d. SELECT \* FROM movies WHERE genre = romance  
and im\_rating LIKE '6'

სწორი პასუხია: SELECT \* FROM movies WHERE genre  
= 'romance' AND im\_rating >6

12) წარმოდგენასთან დაკავშირებით რომელი მტკიცებაა არასწორი ?

ნარმოდგენასთან დაკავშირებით  
რომელი მტკიცებაა არასწორი?

აირჩიეთ ერთი:

- a. ნარმოდგენით შესაძლებელია asc და desc პირობით დალეგებული მონაცემების დაბრუნება.
- b. ნარმოდგენაში არ გამოიყენება პარამეტრი,
- c. ნარმოდგენა არ იქმნება დროებით ცხრილებზე;
- d. ნარმოდგენაში არ მუშაობს order by

სწორი პასუხია: ნარმოდგენით შესაძლებელია asc და desc პირობით დალეგებული მონაცემების

13) რომელი მოთხოვნა დააბრუნებს იმ იუზერებს, რომელთა სახელიც იწყება სიმბოლოთი v-დან z-მდე დრაპაზონში

რომელი მოთხოვნა დააბრუნებს იმ იუზერებს, რომელთა სახელიც იწყება სიმბოლოთი v-დან z-მდე დრაპაზონში.

აირჩიეთ ერთი:

- a. SELECT PKUserId, UserName FROM UserProfile WHERE UserName LIKE '[v-z%]'
- b. SELECT UserName FROM UserProfile WHERE UserName between 'A%' and 'D%)'
- c. SELECT PKUserId, UserName FROM UserProfile WHERE UserName LIKE '[v-z]%'
- d. SELECT UserName FROM UserProfile WHERE UserName LIKE '[vz]%'

სწორი პასუხია: SELECT PKUserId, UserName FROM UserProfile WHERE UserName LIKE '[v-z]%'

14) შემდეგი სურათი გვიჩვენებს მაგალითს:

შემდეგი სურათი გვიჩვენებს მაგალითს:

STUDENT	
SID	
Name	
Address	(Street, City, State, ZipCode)

აირჩიეთ ერთი:

- a. ფარდობითი ატრიბუტი.
- b. კომპოზიტური ატრიბუტი.
- c. მიღებული ატრიბუტი.
- d. მრავალფასიანი ატრიბუტი.

15) სტუდენტს შეუძლია დაესწროს ხუთ საგანს, თითოეულ საგანს კითხულობს სხვადასხვა პროფესორი . თითოეულ პროფესორს 30 სტუდენტი ჰყავს . სტუდენტებსა და პროფესორს შორის კავშირი არის:

სტუდენტს შეუძლია დაესწროს ხუთ საგანს, თითოეულ საგანს კითხულობს სხვადასხვა პროფესორი. თითოეულ პროფესორს 30 სტუდენტი ჰყავს. სტუდენტებსა და პროფესორს შორის კავშირი არის:

აირჩიეთ ერთი:

- a. ერთი-ბევრთან
- b. ერთი-ერთთან
- c. ძლიერი
- d. ბევრი-ბევრთან

სწორი პასუხია: ბევრი-ბევრთან

16) რომელი კონსტრუქცია გამოიყენება მონაცემთა ბაზის ცხრილში ველის წასაშლელად?

რომელი კონსტრუქცია გამოიყენება მონაცემთა ბაზის ცხრილში ველის წასაშლელად?

აირჩიეთ ერთი:

- a. ALTER TABLE... DELETE COLUMN
- b. DELETE COLUMN FROM TABLE
- c. UPDATE TABLE ... DROP COLUMN
- d. ALTER TABLE... DROP COLUMN

სწორი პასუხია: ALTER TABLE... DROP COLUMN

17) რომელ ვირტუალურ ცხრილებთან მუშაობენ ტრიგერები?

რომელ ვირტუალურ ცხრილებთან  
მუშაობენ ტრიგერები?

აირჩიეთ ერთი:

- a. CREATED, INSERTED
- b. UPDATED, DELETED;
- c. INSERTED, DELETED
- d. INSERTED, UPDATED



სწორი პასუხია: INSERTED, DELETED

18) რომელი ბრძანებით მოხდება შეერთების ოპერაციის შედეგად ორი ცხრილიდან ჩანაწერების სრულად დაბრუნება .

რომელი ბრძანებით მოხდება შეერთების  
ოპერაციის შედეგად ორი ცხრილიდან  
ჩანაწერების სრულად დაბრუნება.

აირჩიეთ ერთი:

- a. FULL JOIN
- b. LEFT JOIN
- c. UNION ALL
- d. UNION

სწორი პასუხია: FULL JOIN



### 19) რომელი მტკიცებაა არასწორი ?

რომელი მტკიცებაა არასწორი?

აირჩიეთ ერთი:

- a. სქემა განსაზღვრავს ცხრილებს შორის კავშირებს.
- b. შეუძლებელია ერთ სქემაში ორ სხვადასხვა ცხრილს ჰქონდეს ერთნაირი სახელები;
- c. ერთ სქემას ჰყავს ერთი მფლობელი;
- d. სქემა წარმოადგენს სახელდებულ სივრცეს ბაზის ობიექტების კოლექციის.

სწორი პასუხია: სქემა განსაზღვრავს ცხრილებს შორის კავშირებს.

### 20) წესს, რომელშიც ნათქვამია, რომ თითოეული გარე გასარების მნიშვნელობა უნდა ემთხვეოდეს სხვა რრელაციაში პირველადი გასაღების მნიშვნელობას ეწოდება :

წესს, რომელშიც ნათქვამია, რომ თითოეული გარე გასარების მნიშვნელობა უნდა ემთხვეოდეს სხვა რრელაციაში პირველადი გასაღების მნიშვნელობას ეწოდება:

აირჩიეთ ერთი:

- a. არსის გასარებების ჯგუფის წესი.
- b. გარე / პირველადი გასაღების წესი.
- c. გასაღების თანხვედრის წესი.
- d. დამოწმებითი მთლიანობის შეზღუდვა.

სწორი პასუხია: დამოწმებითი მთლიანობის შეზღუდვა.

21) როდესაც ჩვეულებრივი არსის ტიპი შეიცავს მრავალმნიშვნელოვან ატრიბუტს, საჭიროა :

როდესაც ჩვეულებრივი არსის ტიპი  
შეიცავს მრავალმნიშვნელოვან  
ატრიბუტს, საჭიროა:

აირჩიეთ ერთი:

- a. შევქმნათ ორი ახალი რელაცია,  
რომელთაგან ერთი შეიცავს  
მრავალმნიშვნელოვან ატრიბუტს.
- b. მრავალმნიშვნელოვანი  
ატრიბუტის თითოეული  
ეგზემპლარისათვის შევქმნათ  
ერთი რელაცია მრავალი  
სტრიქონით.
- c. შევქმნათ ორი ახალი რელაცია,  
რომელთაგან ორივე შეიცავს  
შეიცავს მრავალმნიშვნელოვან  
ატრიბუტს.
- d. დავშალოთ რელაცია და  
თავიდან შევადგინოთ.

სწორი პასუხია: შევქმნათ ორი ახალი  
რელაცია, რომელთაგან ერთი შეიცავს  
მრავალმნიშვნელოვან ატრიბუტს. 

22) რა ფუნქცია აქვს GROUP BY ფუნქციას?

რა ფუნქცია აქვს GROUP BY  
ფუნქციას?

აირჩიეთ ერთი:

- a. ალაგებს მონაცემებს  
ზრდადობა/კლებადობით;
- b. აკგუფებს ბაზაში სხვადასხვა  
ობიექტებს.
- c. იგი გამოიყენება  
განსხვავებული მონაცემების  
დასაბრუნებლად;
- d. გამოიყენება მონაცემების  
დააკგუფებისათვის

სწორი პასუხია: აკგუფებს ბაზაში  
სხვადასხვა ობიექტებს.

23) რა შედეგი დაბრუნდება შემდეგი მოთხოვნის გაშვების შემდეგ ? SELECT  
SUBSTRING('DOTNET',1,3)

რა შედეგი დაბრუნდება შემდეგი  
მოთხოვნის გაშვების შემდეგ? SELECT  
SUBSTRING('DOTNET',1,3)

აირჩიეთ ერთი:

- a. N
- b. T
- c. OTN
- d. DOT

სწორი პასუხია: DOT

24) რა შედეგს დააბრუნებს მოთხოვნა (ცხრილში არის 12 ჩანაწერი) .....

რა შედეგს დააბრუნებს მოთხოვნა (ცხრილში არის 12 ჩანაწერი) SELECT column1 FROM table1 ORDER BY column1 asc offset 5 row

აირჩიეთ ერთი:

- a. დაბრუნდება მე-5 ჩანაწერი
- b. დაბრუნდება შეცდომა
- c. დაბრუნდება ყველა ჩანაწერი მე-6-დან დაწყებული
- d. დაბრუნდება ბოლო 5 ჩანაწერი.

სწორი პასუხია: დაბრუნდება ყველა ჩანაწერი მე-6-დან დაწყებული

25) მონაცემთა ბაზის პროექტის ფორმას, რომელიც ასახავს კონცეპტუალურ მოთხოვნებს, ეწოდება

მონაცემთა ბაზის პროექტის ფორმას, რომელიც ასახავს კონცეპტუალურ მოთხოვნებს, ეწოდება:

აირჩიეთ ერთი:

- a. ფიზიკური პროექტი
- b. ლოგიკური პროექტი
- c. რეაგირების პროექტი
- d. უსაფრთხოების პროექტი

სწორი პასუხია: ლოგიკური პროექტი

26) ჩამოთვლილთაგან რომელია არსია, რომელიც არსებობს სხვა არსის ტიპებისგან დამოუკიდებლად?

ჩამოთვლილთაგან რომელია არსია,  
რომელიც არსებობს სხვა არსის  
ტიპებისგან დამოუკიდებლად?

აირჩიეთ ერთი:

- a. ვარიანტული
- b. თანადამოკიდებული
- c. ძლიერი
- d. სუსტი

სწორი პასუხია: ძლიერი

27) ჩამოთვლილთაგან რომელი არ არის კარგი ბიზნეს წესის მახასიათებელი ?

ჩამოთვლილთაგან რომელი არ არის კარგი ბიზნეს წესის მახასიათებელი?

აირჩიეთ ერთი:

- a. არათანმიმდევრული
- b. დეკლარაციული
- c. გამომხატველობითი
- d. ატომური

სწორი პასუხია: არათანმიმდევრული

28) რა შედეგს დააბრუნებს მოთხოვნა? SELECT ename FROM test ORDER BY ename ASC offset 5 row FETCH NEXT 3 row only

რა შედეგს დააბრუნებს მოთხოვნა? SELECT ename FROM test ORDER BY ename ASC offset 5 row FETCH NEXT 3 row only

აირჩიეთ ერთი:

- a. დაბრუნდება (დალაგებული შედეგიდან) პირველი ხუთის გამოკლებით, მომდევნო 3 სტრიქონი.
- b. დაბრუნდება პირველი 3 სტრიქონი
- c. ბოლო 5 სტრიქონიდან დაბრუნდება პირველი 3 სტრიქონი
- d. დაბრუნდება შეცდომა

სწორი პასუხია: დაბრუნდება (დალაგებული შედეგიდან) პირველი ხუთის გამოკლებით, მომდევნო 3 სტრიქონი.

29) რა ფუნქცია აქვს UPDATE კონსტრუქციას ?

რა ფუნქცია აქვს UPDATE კონსტრუქციას?

აირჩიეთ ერთი:

- a. შეუძლია ბაზის სტრუქტურის განახლება;
- b. შეუძლია განახლოს ცხრილის სტრუქტურას;
- c. შეუძლია ცხრილში მხოლოდ და მხოლოდ ერთი ჩანაწერის განახლება
- d. შეუძლია ცხრილში რამდენიმე ჩანაწერის განახლება

სწორი პასუხია: შეუძლია ცხრილში რამდენიმე ჩანაწერის განახლება

30) რას განსაზღვრავს მოთხოვნაში საგასაღებო სიტყვა IN ?

რას განსაზღვრავს მოთხოვნაში  
საგასაღებო სიტყვა IN?

აირჩიეთ ერთი:

- a. განსაზღვრავს, ემთხვევა თუ არა მონაცემი რომელიმეს, ქვემოთხოვნაში მოცემული სიიდან;
- b. გამოიყენება ცხრილის გადატანისათვის ერთი სქემიდან მეორეში.
- c. განსაზღვრავს ცხრილებს, საიდანაც ხდება ჩანაწერების დაბრუნება ან მონაცემების წაშლა
- d. გამოიყენება DISTINCT კონსტრუქციაში უნიკალური ჩანაწერების ამოსაღებად;

სწორი პასუხია: განსაზღვრავს,  
ემთხვევა თუ არა მონაცემი  
რომელიმეს, ქვემოთხოვნაში  
მოიამოო სიიდან.



31) არსს, რომელსაც აერთიანებს ერთი ან მეტი არსის ტიპების ეგზემპლარებს და შეიცავს კავშირების თავისებურებებს, ეწოდება :

არსს, რომელიც აერთიანებს ერთი ან მეტი არსის ტიპების ეგზემპლარებს და შეიცავს კავშირების თავისებურებებს, ეწოდება:

აირჩიეთ ერთი:

- a. აგებული არსი
- b. ასოციაციური არსი.
- c. გამანადგურებელი არსი
- d. კარიბჭე არსი

სწორი პასუხია: ასოციაციური არსი.

32) მომხმარებლები, მანქანები და ნაწილები მაგალითებია :

მომხმარებლები, მანქანები და ნაწილები მაგალითებია:

აირჩიეთ ერთი:

- a. არსების
- b. კარდინალურობის
- c. კავშირების
- d. ატრიბუტების

სწორი პასუხია: არსების

33) თვისებას, რომლითაც ქვეტიპის არსები სუპერტიპის ყველა ატრიბუტის მნიშვნელობებს ფლობენ , ეწოდება :

თვისებას, რომლითაც ქვეტიპის არსები სუპერტიპის ყველა ატრიბუტის მნიშვნელობებს ფლობენ, ეწოდება:

აირჩიეთ ერთი:

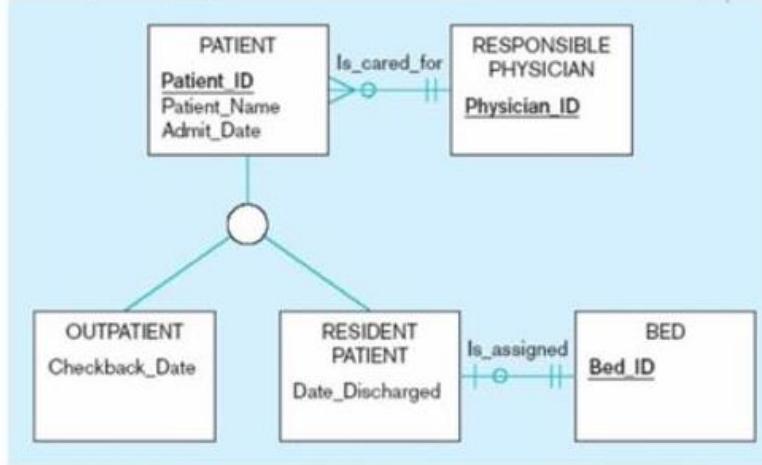
- a. იერარქიის მიღება.
- b. მრავალჯერადი მემკვიდრეობა.
- c. ატრიბუტის მემკვიდრეობა.
- d. კლასის მენეჯმენტი.

სწორი პასუხია: ატრიბუტის მემკვიდრეობა.



34) ქვემოთ მოცემულ ფიგურაში ქვემოთ ჩამოთვლილთაგან რომელი ვრცელდება როგორც OUTPATIENT-ებზე, ასევე RESIDENT\_PATIENT-ზე ?

ქვემოთ მოცემულ ფიგურაში ქვემოთ  
ჩამოთვლილთაგან რომელი  
ვრცელდება როგორც OUTPATIENT-  
ებზე, ასევე RESIDENT\_PATIENT-ზე?



აირჩიეთ ერთი:

- a. XML
- b. Patient\_Name
- c. Checkback\_Date
- d. Date\_Discharged

35) რომელი მტკიცებაა არასწორი? Modify File ბრძანებით შესაძლებელია:

რომელი მტკიცებაა არასწორი? Modify File  
ბრძანებით შესაძლებელია:

აირჩიეთ ერთი:

- a. ახალი ფაილური ჯგუფის დამატება
- b. ფაილის ლოგიკური სახელის ცვლილება;
- c. ფაილის ფიზიკური სახელის შეცვლა;
- d. ფაილის ზრდის ბიჯის ცვლილება;

სწორი პასუხია: ფაილის ლოგიკური სახელის ცვლილება;

36) როდესაც ყველა მრავალმნიშვნელოვანი ატრიბუტი ამოღებულია რელაციიდან, მაშინ ამბობენ რომ ეს არის :

როდესაც ყველა  
მრავალმნიშვნელოვანი ატრიბუტი  
ამოღებულია რელაციიდან, მაშინ  
ამბობბენ რომ ეს არის:



აირჩიეთ ერთი:

- a. ბოის-კოდის ნორმალური ფორმა.
- b. მესამე ნორმალური ფორმა.
- c. პირველი ნორმალური ფორმა.
- d. მეორე ნორმალური ფორმა.

სწორი პასუხია: პირველი ნორმალური ფორმა.



37) რომელი მტკიცებაა სწორი გარე გასაღებთან დაკავშირებით ?

რომელი მტკიცებაა სწორი გარე გასაღებთან  
დაკავშირებით?

აირჩიეთ ერთი:

- a. ცხრილს აქვს იმდენი გარე გასაღები,  
რამდენი შეზღუდვაცაა ცხრილში.
- b. ცხრილში დასაშვებია მხოლოდ ერთი გარე  
გასაღების არსებობა.
- c. ცხრილს შესაძლებელია ჰქონდეს 100-ზე  
მეტი გარე გასაღები.
- d. ცხრილში არის იმდენი გარე გასაღები,  
რამდენი აუცილებლად შევსებადი ველიცაა.

სწორი პასუხია: ცხრილს შესაძლებელია ჰქონდეს  
100-ზე მეტი გარე გასაღები.

38) ჩამოთვლილთაგან, რომელი უზრუნველყობს მონაცემების დიდი  
რაოდენობის კოპირებას ცხრილში ?

ჩამოთვლილთაგან, რომელი  
უზრუნველყობს მონაცემების დიდი  
რაოდენობის კოპირებას ცხრილში?

აირჩიეთ ერთი:

- a. TRANSFER
- b. BULKMOVE
- c. COPY
- d. BULKCOPY

სწორი პასუხია: BULKCOPY

39) რომელი ჩანაწერი მოახდენს ელ ფოსტის მონაცემის განახლებას ?

რომელი ჩანაწერი მოახდენს ელ ფოსტის მონაცემის განახლებას?

აირჩიეთ ერთი:

- a. UPDATE Users SET Email = 'new\_email@yahoo.com' WHERE Email='old\_email@yahoo.com'
- b. ALTER Users SET Email = 'new\_email\_goes\_here@yahoo.com'
- c. UPDATE Users WHERE Email = 'new\_email@yahoo.com'
- d. UPDATE Users HAVING Email = 'new\_email@yahoo.com' WHERE Email='old\_email@yahoo.com'

სწორი პასუხია: UPDATE Users SET Email = 'new\_email@yahoo.com' WHERE Email='old\_email@yahoo.com'

40) მონაცემთა ბაზის რელაციაში არსებულ ატრიბუტს, რომელიც იმავე მონაცემთა ბაზაში სხვა რელაციის ძირითადი გასაღების ფუნქციას ასრულებს, ეწოდება :

მონაცემთა ბაზის რელაციაში  
არსებულ ატრიბუტს, რომელიც იმავე  
მონაცემთა ბაზაში სხვა რელაციის  
ძირითადი გასაღების ფუნქციას  
ასრულებს, ეწოდება:

აირჩიეთ ერთი:

- a. გარე ატრიბუტი.
- b. ბმულის ატრიბუტი.
- c. ბმული გასაღები.
- d. გარე გასაღები.

სწორი პასუხია: გარე გასაღები.

41) არსების ტიპების რაოდენობას, რომლებიც მონაწილეობენ ურთიერთობაში, ეწოდება :

არსების ტიპების რაოდენობას,  
რომლებიც მონაწილეობენ  
ურთიერთობაში, ეწოდება:

აირჩიეთ ერთი:

- a. მრიცხველი.
- b. რიცხვი.
- c. საიდენტიფიკაციო  
მახასიათებლი.
- d. ხარისხი.

სწორი პასუხია: ხარისხი.



42) UPDATE ბრძანებით შესაძლებელია

UPDATE ბრძანებით შესაძლებელია

აირჩიეთ ერთი:

- a. წაიშალოს ჩანაწერები ერთზე  
მეტი სტრიქონიდან.
- b. მოხდეს ჩანაწერების  
განახლება ერთზე მეტ სტრიქონში;
- c. წაიშალოს ჩანაწერები  
მხოლოდ ერთი სტრიქონიდან;
- d. ყოველთვის მოხდეს  
ჩანაწერების განახლება მხოლოდ  
ერთ სტრიქონში;

სწორი პასუხია: მოხდეს ჩანაწერების  
განახლება ერთზე მეტ სტრიქონში;



43) რომელი ჩანაწერი მოახდენს ცხრილისათვის budget ველის დამატებას?

რომელი ჩანაწერი მოახდენს ცხრილისათვის budget ველის დამატებას?

აირჩიეთ ერთი:

- a. update table project add budget int
- b. update table project insert budget int
- c. alter table project add budget int
- d. alter table project insert budget int

სწორი პასუხია: update table project add budget int



44) რა ფუნქცია HAVING აქვს ბრძანებას ?

რა ფუნქცია აქვს HAVING ბრძანება?

აირჩიეთ ერთი:

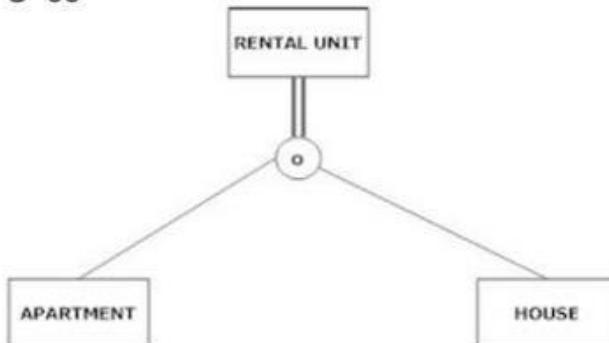
- a. გამოიყენება მონაცემების დალაგებისათვის;
- b. აკონკრეტებს ძებნის პირობას ჰაფური ფუნქციის შედეგში;
- c. იგი გამოიყენება განსხვავებული მონაცემების დასაბრუნებლად;
- d. გამოიყენება რამდენიმე ცხრილის შესაერთებლად;

სწორი პასუხია: აკონკრეტებს ძებნის პირობას ჰაფური ფუნქციის შედეგში;



45) ქვემოთ მოყვანილი დებულებებიდან რომელი შეესაბამება მოცემულ ფიგურას ?

ქვემოთ მოყვანილი დებულებებიდან  
რომელი შეესაბამება მოცემულ  
ფიგურას?



აირჩიეთ ერთი:

- a. საიკარო ერთეული (RENTAL UNIT) შეიძლება იყოს ბინა (APARTMENT) , სახლი (HOUSE) ან უბრალოდ საიკარო ერთეული; ეს შეიძლება არ იყოს ერთზე მეტი.
- b. საიკარო ერთეული (RENTAL UNIT) უნდა იყოს ბინა (APARTMENT) ან სახლი (HOUSE) და შეიძლება იყოს ორივე.
- c. საიკარო ერთეული (RENTAL UNIT) შეიძლება იყოს ბინა (APARTMENT) , სახლი (HOUSE) ან უბრალოდ საიკარო განყოფილება. ეს შეიძლება იყოს ერთდროულად ბინაც და სახლიც.
- d. საიკარო ერთეული (RENTAL UNIT) უნდა იყოს ბინა(APARTMENT) ან სახლი(HOUSE) და არ შეიძლება იყოს ორივე ერთდროულად.

სწორი პასუხია: საიკარო ერთეული (RENTAL UNIT) უნდა იყოს ბინა (APARTMENT) ან სახლი (HOUSE) და შეიძლება იყოს ორივე.

46) რომელი მოთხოვნაა არასწორად ჩაწერილი ?

რომელი მოთხოვნაა არასწორად  
ჩაწერილი?

აირჩიეთ ერთი:

- a. `SELECT dept.deptno, job FROM dept FULL JOIN emp ON dept.deptno=emp.deptno GROUP BY dept.deptno,job ORDER BY dept.deptno`
- b. `SELECT dept.deptno, job, COUNT(ename) FROM dept FULL JOIN emp ON dept.deptno=emp.deptno GROUP BY dept.deptno,job ORDER BY dept.deptno`
- c. `SELECT dept.deptno, job, COUNT(ename) FROM dept FULL JOIN emp ON dept.deptno=emp.deptno GROUP BY ename ORDER BY dept.deptno asc`
- d. `SELECT dept.deptno, job FROM dept FULL JOIN emp ON dept.deptno=emp.deptno GROUP BY dept.deptno,JOB`

სწორი პასუხია: `SELECT dept.deptno, job, COUNT(ename) FROM dept FULL JOIN emp ON dept.deptno=emp.deptno GROUP BY dept.deptno,job ORDER BY dept.deptno`



47) მონაცემთა ბაზის შემუშავება იწყება \_\_\_\_\_, რაც განსაზღვრავს ორგანიზაციული მონაცემთა ბაზის დიაპაზონს და ზოგად შინაარსს.

მონაცემთა ბაზის შემუშავება იწყება \_\_\_\_\_, რაც განსაზღვრავს ორგანიზაციული მონაცემთა ბაზის დიაპაზონს და ზოგად შინაარსს.

აირჩიეთ ერთი:

- a. საწარმოს მონაცემთა მოდელირებით
- b. ორგანიზაციული მონაცემების მოდელირებით
- c. მონაცემთა ბაზის დიზაინით
- d. გადამკვეთი ფუნქციური ანალიზით

სწორი პასუხია: საწარმოს მონაცემთა მოდელირებით



48) ფიზიკური ფაილების და მონაცემთა ბაზების შემუშავების მოთხოვნაა :

ფიზიკური ფაილების და მონაცემთა ბაზების შემუშავების მოთხოვნაა:

აირჩიეთ ერთი:

- a. მონაცემთა ყველა ტიპის დადგენა.
- b. შექმნილი ფიზიკური ცხრილები.
- c. ნორმალიზებული რელაციები
- d. განხორციელების დასრულება.

სწორი პასუხია: ნორმალიზებული რელაციები



49) რომელი ფუნქცია გამოიყენება სტრიქონში კონკრეტული ქვესტრიქონის პოზიციის დასაბრუნებლად?

რომელი ფუნქცია გამოიყენება სტრიქონში კონკრეტული ქვესტრიქონის პოზიციის დასაბრუნებლად?

აირჩიეთ ერთი:

- a. STRING\_POSITION
- b. SUBSTRING
- c. CHARINDEX
- d. INSTR

სწორი პასუხია: CHARINDEX

50) ჩამოთვლილთაგან რომელი ბრძანება ასუფთავებს ცხრილს მონაცემებისაგან?

ჩამოთვლილთაგან რომელი ბრძანება ასუფთავებს ცხრილს მონაცემებისაგან?

აირჩიეთ ერთი:

- a. CLEAR
- b. REMOVE
- c. DROP
- d. DELETE

სწორი პასუხია: DELETE

51) რომელი ტიპის შეზღუდვა არ არსებობს ველისათვის?

რომელი ტიპის შეზღუდვა არ არსებობს ველისათვის?

აირჩიეთ ერთი:

- a. ველისათვის უნიკალურობის განსაზღვრა.
- b. ველისათვის გარე გასაღების მინიჭების შეზრუდვა
- c. ველში ჩანაწერების კლებადობის მიხედვით დალაგების შეზღუდვა.
- d. ველის მნიშვნელობისათვის კონკრეტული დიაპაზონის განსაზღვრა.

სწორი პასუხია: ველში ჩანაწერების კლებადობის მიხედვით დალაგების

52) პიროვნების სახელი, დაბადების დღე და პირადი ნომერი ესენია :

პიროვნების სახელი, დაბადების დღე და პირადი ნომერი ესენია

აირჩიეთ ერთი:

- a. კავშირები
- b. აღნერები.
- c. ატრიბუტები.
- d. არსები

სწორი პასუხია: ატრიბუტები.

53) რომელი ბრძანებით მოხდება ორი ან მეტი მსგავსი select მოთხოვნის შედეგის გაერთიანება ისე, რომ არ დაბრუნდება დუბლირებული ჩანაწერები .

რომელი ბრძანებით მოხდება ორი ან მეტი მსგავსი select მოთხოვნის შედეგის გაერთიანება ისე, რომ არ დაბრუნდება დუბლირებული ჩანაწერები.

აირჩიეთ ერთი:

- a. UNION
- b. LEFT JOIN
- c. FULL JOIN
- d. UNION ALL

სწორი პასუხია: UNION



54) ჩამოთვლილთაგან რომელი მტკიცებაა არასწორი :

ჩამოთვლილთაგან რომელი მტკიცებაა არასწორი:

აირჩიეთ ერთი:

- a. ტრიგერით შეუძლებელია ALTER DATABASE ბრძანების შესრულება.
- b. ტრიგერი რეაგირებს ცხრილებში სამ თპერაციაზე: ჩასმის, განახლების და წაშლის
- c. ტრიგერით შესაძლებელია CREATE DATABASE ბრძანების შესრულება.
- d. არსებობს DML (INSERT, UPDATE, DELETE) და DDL ტრიგერები.

სწორი პასუხია: ტრიგერით შესაძლებელია CREATE DATABASE ბრძანების შესრულება.

55) რომელი მტკიცებაა არასწორი ?

რომელი მტკიცებაა არასწორი?

აირჩიეთ ერთი:

- a. შეუძლებელია ერთ სქემაში ორ სხვადასხვა ცხრილს ჰქონდეს ერთნაირი სახელები;
- b. სქემა ნარმოადგენს სახელდებულ სივრცეს ბაზის ობიექტების კოლექციის.
- c. ერთ სქემას ჰყავს ერთი მფლობელი;
- d. სქემა განსაზღვრავს ცხრილებს შორის კავშირებს.

სწორი პასუხია: სქემა განსაზღვრავს ცხრილებს შორის კავშირებს.

56) რომელი კონსტრუქცია გამოიყენება მონაცემთა ბაზის ცხრილში ველის წასაშლელად?

რომელი კონსტრუქცია გამოიყენება მონაცემთა ბაზის ცხრილში ველის წასაშლელად?

აირჩიეთ ერთი:

- a. ALTER TABLE... DELETE COLUMN
- b. ALTER TABLE... DROP COLUMN
- c. DELETE COLUMN FROM TABLE
- d. UPDATE TABLE ... DROP COLUMN

სწორი პასუხია: ALTER TABLE... DROP COLUMN

57) თვისებას, რომლითაც ქვეტიპის არსები სუპერტიპის ყველა ატრიბუტის მნიშვნელობებს ფლობენ, ეწოდება :

თვისებას, რომლითაც ქვეტიპის არსები სუპერტიპის ყველა ატრიბუტის მნიშვნელობებს ფლობენ, ეწოდება:

აირჩიეთ ერთი:

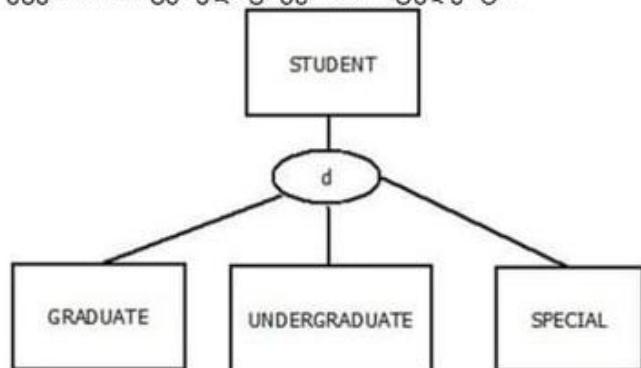
- a. მრავალჯერადი მემკვიდრეობა.
- b. ატრიბუტის მემკვიდრეობა.
- c. იერარქიის მიღება.
- d. კლასის მენეჯმენტი.

სწორი პასუხია: ატრიბუტის მემკვიდრეობა. 



58) ქვემოთ მოცემულ ფიგურაში სტუდენტი :

ქვემოთ მოცემულ ფიგურაში სტუდენტი:



აირჩიეთ ერთი:

- a. უნდა იყოს მინიმუმ სპეციალური სტუდენტი.
- b. უნდა იყოს მაგისტრანტის ან ბაკალავრიატის სტუდენტი.
- c. უნდა იყოს მაგისტრანტი
- d. უნდა იყოს მაგისტრანტი, ბაკალავრიატი, სპეციალური სტუდენტი ან სხვა ტიპის სტუდენტი.

სწორი პასუხია: უნდა იყოს მაგისტრანტი, ბაკალავრიატი, სპეციალური სტუდენტი ან სხვა ტიპის სტუდენტი.

59) ჩამოთვლილთაგან რომელია ტრიგერის არასწორი ტიპი ?

ჩამოთვლილთაგან, რომელია ტრიგერის არასწორი ტიპი?

აირჩიეთ ერთი:

- a. DELETE
- b. SELECT
- c. UPDATE
- d. INSERT

სწორი პასუხია: SELECT

60) რომელი მტკიცებაა არასწორი : წარმოდგენით შესაძლებელია

რომელი მტკიცებაა არასწორი? წარმოდგენით შესაძლებელია:

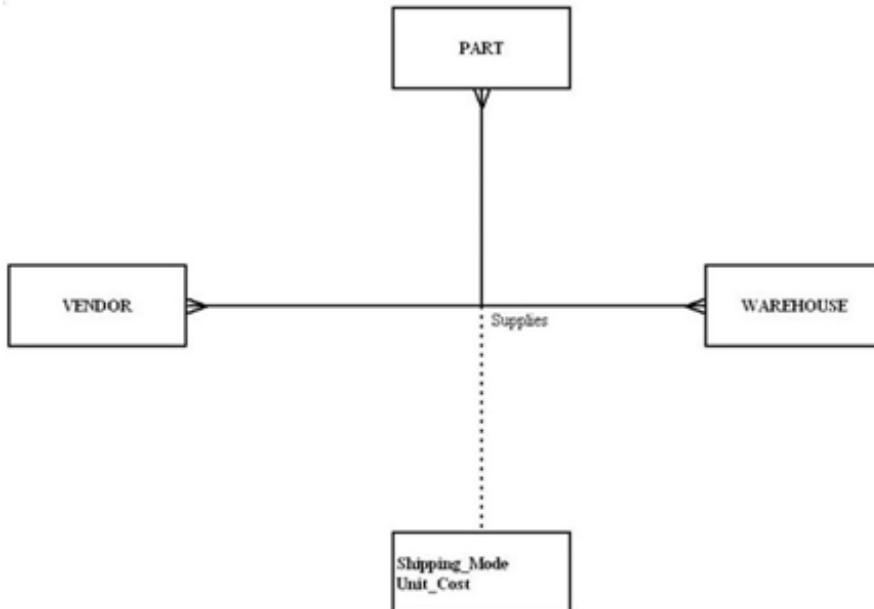
აირჩიეთ ერთი:

- a. სხვადასხვა ცხრილებში არსებული მონაცემების შეერთება და შე
- b. კონკრეტული ცხრილებით ახალი ბაზის შექმნა.
- c. ცხრილის კონკრეტული ველების მონაცემებთან წვდომის შეზღუ
- d. გარდაქმნების შედეგად მიღებული ინფორმაციის წარმოდგენა.

სწორი პასუხია: კონკრეტული ცხრილებით ახალი ბაზის შექმნა. ✓

61) რა ტიპის ურთიერთობაა გამოსახული დიაგრამაზე?

რა ტიპის ურთიერთობაა გამოსახული დიაგრამაზე?



აირჩიეთ ერთი:

- a. კვადრატი
- b. უნარი
- c. ორობითი
- d. სამეული

სწორი პასუხია: სამეული

62) აუთენტიფიკაციის რომელი რეჟიმი არ გვაქვს sql server-ზე

აუთენტიფიკაციის რომელი რეჟიმი არ გვაქვს sql server-ზე

აირჩიეთ ერთი:

- a. Double Mode
- b. Windows Mode
- c. Mixed Mode
- d. Sql Mode

სწორი პასუხია: Double Mode



63) ჩამოთვლილთაგან , რომელი ფუნქცია დააბრუნებს რეალურ დროს ?

---

ჩამოთვლითაგან, რომელია ფუნქცია დააბრუნებს რეალურ დროს?

აირჩიეთ ერთი:

- a. GETDATE
  - b. SYSDATETIME
  - c. Cert\_ID
  - d. SET DATEFIRST
- 

სწორი პასუხია: GETDATE

64) ჩამოთვლილთაგან რომელია ზოგადი არსის ტიპი, რომელსაც აქვს კავშირი ერთ ან მეტ ქვეტიპთან?

ჩამოთვლილთაგან რომელია ზოგადი არსის ტიპი, რომელსაც აქვს კავშირი ერთ ან მეტ ქვეტიპთან?

აირჩიეთ ერთი:

- a. კლასი
  - b. ქვეჯგუფი
  - c. მეგატიპი
  - d. სუპერტიპი
- 

სწორი პასუხია: სუპერტიპი

65) არსის ტიპს, რომლის არსებობა დამოკიდებულია სხვა არსის ტიპზე,  
ეზოდება \_\_\_\_ არსი .

არსის ტიპს, რომლის არსებობა დამოკიდებულია სხვა არსის ტიპზე, ეზოდება \_\_\_\_\_ არსი

აირჩიეთ ერთი:

- a. ვარიანტული
  - b. ძლიერი
  - c. სუსტი
  - d. თანადამოკიდებული
- 

სწორი პასუხია: სუსტი

66) რომელ ნორმალურ ფორმაშია დამოკიდებულება, რომელიც არ შეიცავს  
მრავალმნიშვნელოვან ატრიბუტებს და აქვს არაგასარები ატრიბუტები,  
რომლებიც მხოლოდ პირველად .....  
.....

რომელ ნორმალურ ფორმაშია დამოკიდებულება, რომელიც არ შეიცავს მრავალმნიშვნელოვან ატრიბუტებს და აქვს არაგასარები ატრიბუტები, რომლებიც მხოლოდ პირველად  
გასარჩევა დამოკიდებული, მაგრამ შეიცავს ტარშიტულ დამოკიდებულებებს?

აირჩიეთ ერთი:

- a. პირველი
- b. მეორე
- c. მესამე
- d. მეოთხე

სწორი პასუხია: მეორე **მეორე**

67) ჩამოთვლილთაგან რომელია სწორი ?

ჩამოთვლილთაგან რომელია სწორი?

აირჩიეთ ერთი:

- a. ტრიგერი მუშაობს რამდენიმე ბაზაზე.
- b. ტრიგერი მიეკუთვნება ბაზის ყველა ცხრილს
- c. ტრიგერი შეიძლება მიეკუთვნებოდეს რამდენიმე ცხრილს
- d. ტრიგერი მიეკუთვნება მხოლოდ ერთ ცხრილს

სწორი პასუხია: ტრიგერი მიეკუთვნება მხოლოდ ერთ ცხრილს

68) პირველადი გასაღები, რომელიც შედგება ერთზე მეტი ატრიბუტისგან, ეწოდება :

პირველადი გასაღები, რომელიც შედგება ერთზე მეტი ატრიბუტისგან, ეწოდება:

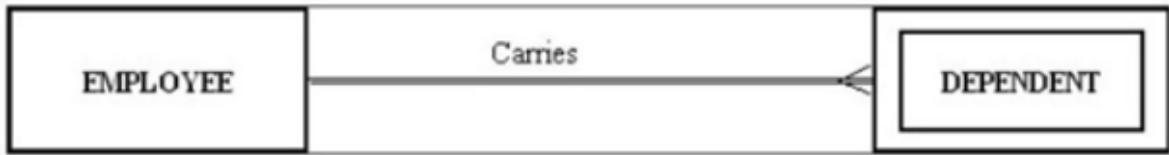
აირჩიეთ ერთი:

- a. კარდინალური გასაღები.
- b. კომპოზიტური გასაღები.
- c. მრავალფასიანი გასაღები.
- d. გარე გასაღები.

სწორი პასუხია: კომპოზიტური გასაღები.

69) ქვემოთ მოცემული სურათი გვიჩვენებს მაგალითს:

ქვემოთ მოცემული სურათი გვიჩვენებს მაგალითს:



აირჩიეთ ერთი:

- a. თანადამოვიდებული კავშირი
- b. ორმაგი ურთიერთობა.
- c. კავშირი „ერთი-ბევრთან“
- d. ძლიერი არსი და მასთან დაკავშირებული სუსტი არსი

სწორი პასუხია: ძლიერი არსი და მასთან დაკავშირებული სუსტი არსი



70) „პროგრამა-მონაცემები“ დამოკიდებულება გამოწვეულია იმით, რომ :

„პროგრამა-მონაცემები“ დამოკიდებულება გამოწვეულია იმით რომ:

აირჩიეთ ერთი:

- a. მონაცემთა აღწერილობა ინახება სერვერზე.
- b. მონაცემთა აღწერილობა იწერება პროგრამირების კოდში.
- c. მონაცემებთან პროგრამებთან თანაცხოვრება.
- d. ფაილის აღწერილობა ინახება მონაცემთა ბაზის თითოეულ პროგრამაში.

სწორი პასუხია:

ფაილის აღწერილობა ინახება მონაცემთა ბაზის თითოეულ პროგრამაში.



71) დააბრუნეთ ყველა ფილმის სახელი - name და რეიტინგი im\_rating სადაც სათაურში ფიგურირებს სიტყვა 'day'

დააბრუნეთ ყველა ფილმის სახელი-name და რეიტინგი im\_rating სადაც სათაურში ფიგურირებს სიტყვა 'day'

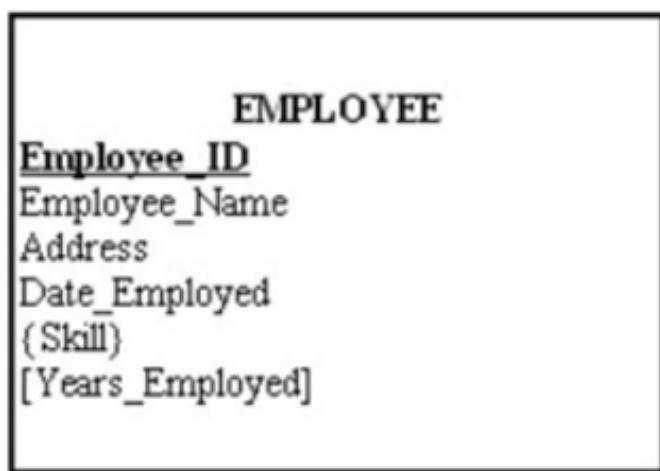
აირჩიეთ ერთი:

- a. SELECT name, im\_rating FROM movies WHERE name LIKE '%day%'
- b. SELECT \* FROM movies WHERE name = "%day%"
- c. SELECT name, im\_rating FROM movies WHERE name = '%day%'
- d. SELECT \* FROM movies WHERE name LIKE '[d-y]%'

სწორი პასუხია: SELECT name, im\_rating FROM movies WHERE name LIKE '%day%'

72) ქვემოთ მოცემულ ფიგურაში რომელი ატრიბუტია წარმოები

ქვემოთ მოცემულ ფიგურაში რომელი ატრიბუტია წარმოები



აირჩიეთ ერთი:

- a. Years\_Employed
- b. Skill
- c. Employee\_ID

სწორი პასუხია: Years\_Employed



73) სუპერტიპის / ქვეტიპის იერარქიაში თითოეულ ქვეტიპს აქვს :

სუპერტიპის / ქვეტიპის იერარქიაში თითოეულ ქვეტიპს აქვს:

აირჩიეთ ერთი:

- a. მაქსიმუმ ორი სუპერტიპი.
- b. მრავალი სუპერტიპი.
- c. მინიმუმ ერთი ქვეტიპი.
- d. მხოლოდ ერთი სუპერტიპი.

სწორი პასუხია: მხოლოდ ერთი სუპერტიპი.

74) რომელი ფუნქცია გამოიყენება სტრიქონში კონკრეტული ქვესტრიქონის პოზიციის დასაბრუნებლად ?

რომელი ფუნქცია გამოიყენება სტრიქონში კონკრეტული ქვესტრიქონის პოზიციის დასაბრუნებლად?

აირჩიეთ ერთი:

- a. STRING\_POSITION
- b. INSTR
- c. CHARINDEX
- d. SUBSTRING

სწორი პასუხია: CHARINDEX



74) რომელი ბრმანებით მოხდება ორი ან მეტი მსგავსი select მოთხოვნის შედეგის გაერთიანება ისე , რომ არ იქნება გათვალისწინებული ...

რომელი ბრმანებით მოხდება ორი ან მეტი მსგავსი select მოთხოვნის შედეგის გაერთიანება ისე, რომ არ იქნება გათვალისწინებული დუბლიკატებიც)

აირჩიეთ ერთი:

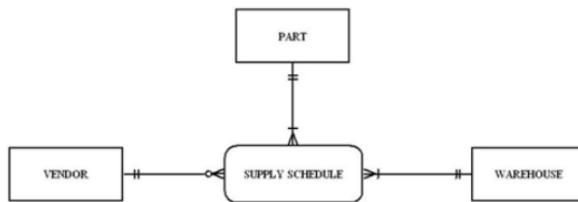
- a. UNION ALL
- b. FULL JOIN
- c. CROSS JOIN
- d. OUTER JOIN

სწორი პასუხია: UNION ALL

75) ქვემოთ მოცემულ ფიგურაში , რომელი ბიზნეს წესები აღიწერება ...

პასუხი\_— თითოეულ გამყიდველს შეუძლია მრავალი ნაწილის მიწოდება  
ნებისმიერი რაოდენობის საწყობიდან, მაგრამ არაა ვალდებული მიაწოდოს  
რაიმე ნაწილი

ქვემოთ მოცემულ ფიგურაში, რომელი ბიზნეს წესები აღიწერება (VENDOR - კამპონტი, PARTS - ნაწილი, WAREHOUSES - საწყობი, SUPPLY SCHEDULE - მიწოდების კანკოგი)



აირჩიეთ ერთი:

- a. თითოეული საწყობი შეიძლება მიეწოდოს წემისმიერი რაოდენობის ნაწილი ერთზე მეტი გამყიდველისგან და თითოეული საწყობი შეიძლება არ მიეწოდოს წაწილი.
- b. თითოეულ გამყიდველს შეუძლია მრავალი ნაწილის მიწოდება წემისმიერი რაოდენობის საწყობიდან, მაგრამ არაა ვალდებული მიაწოდოს რაიმე ნაწილი.
- c. გამყიდველი დაუშვებელია.
- d. თითოეული ნაწილი უნდა მიაწოდოს ზუსტად ერთმა გამყიდველმა წემისმიერი რაოდენობის საწყობში.

სწორი პასუხია: თითოეულ გამყიდველს შეუძლია მრავალი ნაწილის მიწოდება წემისმიერი რაოდენობის საწყობიდან, მაგრამ არაა ვალდებული მიაწოდოს რაიმე ნაწილი.

76) ჩამოთვლილთაგან რომელი არ არის მონაცემთა სახელის კარგი  
მახასიათებელი?

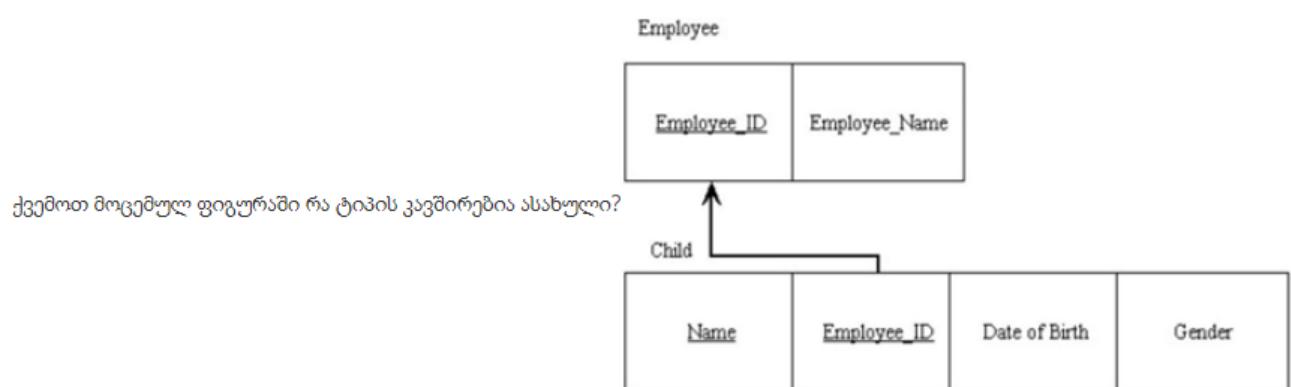
ჩამოთვლილთაგან რომელი არ არის მონაცემთა სახელის კარგი მახასიათებელი?

აირჩიეთ ერთი:

- a. განეკუთვნება ბიზნესის მახასიათებლებს
- b. განმეორებადია
  - c. განეკუთვნება სისტემის ტექნიკურ მახასიათებელს
  - d. იკითხება

სწორი პასუხია: განეკუთვნება სისტემის ტექნიკურ მახასიათებელს ✓

77) ქვემოთ მოცემულ ფიგურაში რა ტიპის კავშირებია ასახული?



აირჩიეთ ერთი:

- a. კომპოზიტური გარე გასაღები
- b. მრავალმნიშვნელოვანება
- c. არსის / სუსტი არსის იდენტიფიკაცია
- d. ერთი-ბევრთან

სწორი პასუხია: არსის / სუსტი არსის იდენტიფიკაცია

78) კავშირი , სადაც მინიმალური და მაქსიმალური კარდინალურობა ერთია, არის

კავშირი, სადაც მინიმალური და მაქსიმალური კარდინალურობა ერთია, არის

აირჩიეთ ერთი:

- a. სურვილისამებრ კავშირი
- b. სავალდებულო ბმული
- c. ცალმხრივი კავშირი
- d. სავალდებულო კავშირი

სწორი პასუხია: სავალდებულო კავშირი

79) ჩამოთვლილთაგან , რომელია DML-ის განსაზღვრა ?

ჩამოთვლილთაგან , რომელია DML-ის განსაზღვრა?

აირჩიეთ ერთი:

- a. Difference Model Level
- b. Data Model Lane
- c. Data Model Language
- d. Data Manipulation Language

სწორი პასუხია: Data Manipulation Language

80) ორ ატრიბუტს შორის შეზღუდვა ეწოდება :

ორ ატრიბუტს შორის შეზღუდვას ეწოდება:

აირჩიეთ ერთი:

- a. ფუნქციური მიმართება.
- b. ატრიბუტის დამოკიდებულება.
- c. ფუნქციონალური დამოკიდებულება.
- d. ფუნქციური მიმართულების შეზღუდვა.

სწორი პასუხია: ფუნქციონალური დამოკიდებულება.

81) რომელი კონსტრუქცია გამოიყენება მონაცემთა ბაზის ცხრილში მონაცემის ტიპის შესაცვლელად?

რომელი კონსტრუქცია გამოიყენება მონაცემთა ბაზის ცხრილში მონაცემის ტიპის შესაცვლელად?

აირჩიეთ ერთი:

- a. ALTER TABLE ... ALTER COLUMN
- b. CHANGE TABLE ... MODIFY
- c. MODIFY COLUMN
- d. ALTER TABLE...ADD DATA

სწორი პასუხია: ALTER TABLE ... ALTER COLUMN

82) სუპერტიპის ერთი ან მეტი ქვეტიპის განსაზღვრისა და კავშირების ფორმირების პროცესს ეწოდება :

სუპერტიპის ერთი ან მეტი ქვეტიპის განსაზღვრისა და კავშირების ფორმირების პროცესს ეწოდება:

აირჩიეთ ერთი:

- a. კლასების შერჩევა.
- b. განზოგადება.
- c. სპეციალიზაცია
- d. უთანხმოების შექმნა.

სწორი პასუხია: სპეციალიზაცია



83) რელაცია, რომელიც შეიცავს მინიმალურ სიჭარბეს და საშუალებას იძლევა მარტივად გამოიყენოთ ის არის :

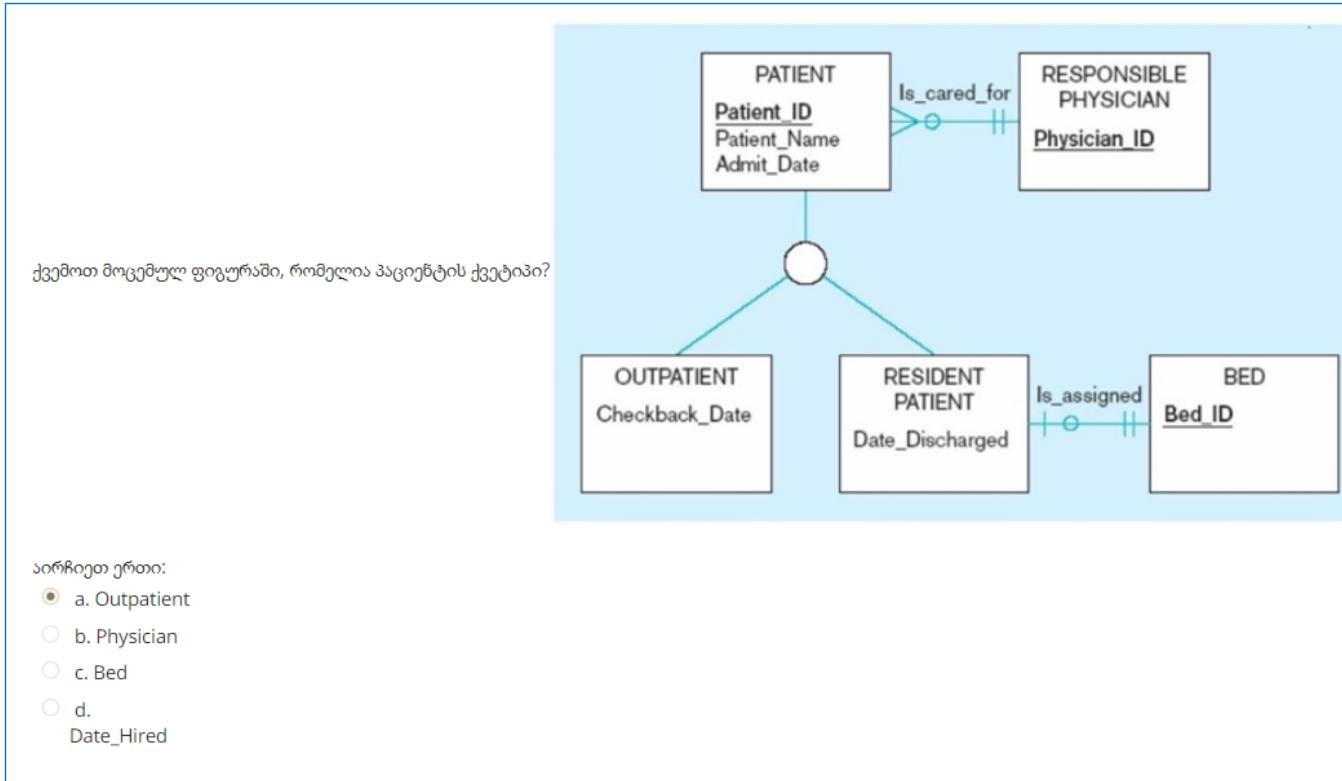
რელაცია, რომელიც შეიცავს მინიმალურ სიჭარბეს და საშუალებას იძლევა მარტივად გამოიყენოთ ის არის:

აირჩიეთ ერთი:

- a. კარგად სტრუქტურირებული.
- b. სუფთა.
- c. კომპლექსური.
- d. მარტივი.

სწორი პასუხია: კარგად სტრუქტურირებული.

84) ქვემოთ მოცემულ ფიგურაში რომელია პაციენტის ქვეტიპი?



სწორი პასუხია: Outpatient

85) ატრიბუტს , რომლის გამოანგარიშება შესაძლებელია ატრიბუტის  
მნიშვნელობებიდან , ეწოდება \_\_\_\_\_ ატრიბუტი

ატრიბუტს , რომლის გამოანგარიშება შესაძლებელია ატრიბუტის მნიშვნელობებიდან , ეწოდება \_\_\_\_\_ ატრიბუტი.

აირჩიეთ ერთი:

- a. წარმოებული
- b. მრავალფასიანი
- c. მარტივი
- d. კომპოზიტური

სწორი პასუხია: წარმოებული

86) ძირითადი გადაწყვეტილება ფიზიკური დიზაინის პროცესში არის :

ძირითადი გადაწყვეტილება ფიზიკური დიზაინის პროცესში არის:

აირჩიეთ ერთი:

- a. E-R დიაგრამების გადაწყვეტა.
- b. მონაცემთა ბაზის ზომის უგულებელყოფა.
- c. სტრუქტურების შერჩევა.
- d. მონიტორის გადაწყვეტილების მიღება.

სწორი პასუხია: სტრუქტურების შერჩევა.



87) როგორც წესი , მეტამონაცემები არ მოიცავს აღწერას :

როგორც წესი, მეტამონაცემები არ მოიცავს  
აღწერას:

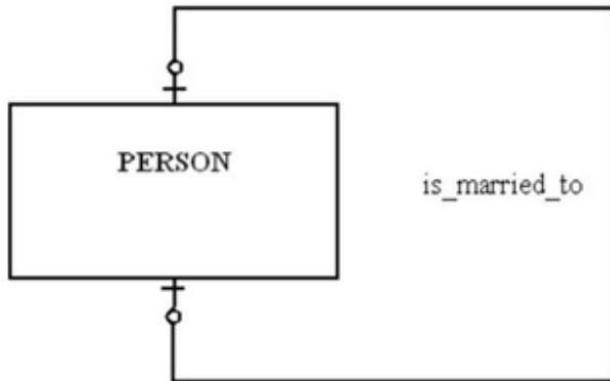
აირჩიეთ ერთი:

- a. დასაშვები მნიშვნელობების.
- b. მონაცემთა სიგრძის
- c. მონაცემთა განმარტებების.
- d. დისკვიზე ადგილობრივობის

სწორი პასუხია: დისკვიზე ადგილობრივობის

88) ქვემოთ მოცემული გამონათქვამებიდან რომელი შეესაბამება ნახატს

ქვემოთ მოცემული გამონათქვამებიდან  
რომელი შეესაბამება ნახატს



აირჩიეთ ერთი:

- a. ადამიანს შეუძლია დაქორწინდეს  
მაქსიმუმ ერთზე.
- b. ადამიანს შეუძლია დაქორწინდეს  
ერთზე მეტი პირზე.
- c. პირს შეუძლია დაქორწინდეს ერთზე  
მეტი ადამიანზე, მაგრამ ამ პირს შეუძლია  
იყოს ქორწინებაში მხოლოდ ერთზე.
- d. ადამიანი უნდა იყოს  
დაქორწინებული.

სწორი პასუხია: ადამიანს შეუძლია  
დაქორწინდეს მაქსიმუმ ერთზე.

89) ატრიბუტს, რომელიც ცალსახად განსაზღვრავს არსს და შედგება  
კომპოზიციური ატრიბუტებისგან, ეწოდება

ატრიბუტს, რომელიც ცალსახად განსაზღვრავს  
არსს და შედგება კომპოზიციური ატრიბუტისგან,  
ენოდება

აირჩიეთ ერთი:

- a. საიდენტიფიკაციო ატრიბუტის.
- b. კომპოზიტური ატრიბუტი.
- c. კომპოზიტური იდენტიფიკატორი.
- d. დამოკიდებულების იდენტიფიკატორი.

სწორი პასუხია: კომპოზიტური იდენტიფიკატორი.

90) რომელი კრიტერიუმია გასათვალისწინებელი იდენტიფიკატორის არჩევისას ?

რომელი კრიტერიუმია  
გასათვალისწინებული იდენტიფიკატორის  
არჩევისას?

აირჩიეთ ერთი:

- a. ნულოვანი იდენტიფიკატორი.
- b. იდენტიფიკატორი, რომელიც არ არის სტაბილური.
- c. იდენტიფიკატორი, რომელსაც არ აქვს ბევრი კომპოზიციური ატრიბუტები.
- d. ყველაზე რთული იდენტიფიკატორი.

სწორი პასუხია: იდენტიფიკატორი,  
რომელსაც არ აქვს ბევრი კომპოზიციური  
ატრიბუტები. 

91) ორ ატრიბუტს შორის შეზღუდვას ეწოდება .

ორ ატრიბუტს შორის შეზღუდვას ეწოდება:

აირჩიეთ ერთი:

- a. ატრიბუტის დამოკიდებულება.
- b. ფუნქციური მიმართულების შეზღუდვა.
- c. ფუნქციონალური დამოკიდებულება.
- d. ფუნქციური მიმართება.

სწორი პასუხია: ფუნქციონალური დამოკიდებულება.

92) რა შედეგს დააბრუნებს მოთხოვნა ? SELECT ename FROM emp OFFSET 5 ROW FETCH NEXT 3 ROW only

რა შედეგს დააბრუნებს მოთხოვნა ? SELECT ename FROM emp OFFSET 5 ROW FETCH NEXT 3 ROW only

აირჩიეთ ერთი:

- a. ბოლო 5 სტრიქონიდან დაბრუნდება პირველი 3 სტრიქონი
- b. დაბრუნდება შეცდომა
- c. დაბრუნდება (დალაგებული შედეგიდან) პირველი ხუთის გამოკლებით, მომდევნო 3 სტრიქონი.
- d. დაბრუნდება პირველი 3 სტრიქონი

სწორი პასუხია: დაბრუნდება შეცდომა ✓

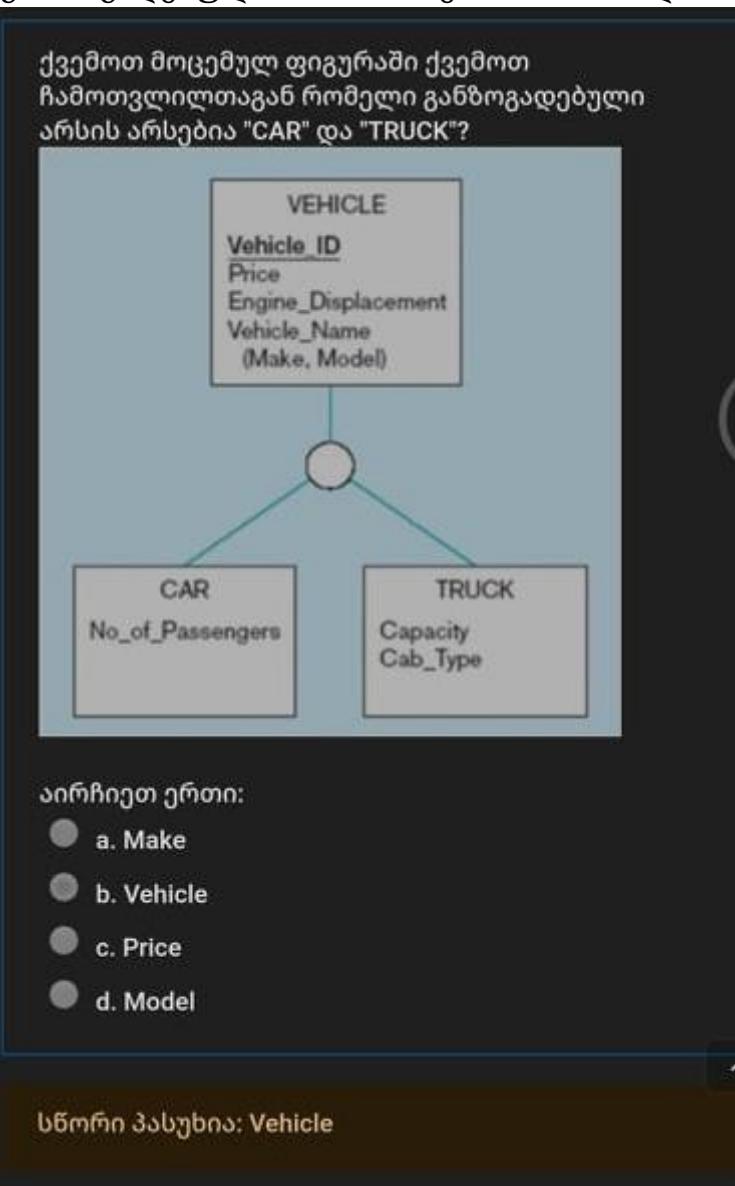
93) რომელი მოქმედების შესრულებაა შეუძლებელი ALTER DATABASE ბრძანებით ?

აირჩიეთ ერთი:

- a. მონაცემთა ბაზის ნაშლა;
- b. ბაზისათვის ახალი ფაილური ჰგუფის დამატება
- c. ბაზის ატრიბუტების შეცვლა
- d. ბაზის ფაილების მოდიფიკაცია

სწორი პასუხია: მონაცემთა ბაზის ნაშლა;

94) ქვემოთ მოცემულ ფიგურაში ქვემოთ ჩამოთვლილთაგან რომელი განზოგადებული არსის არსებია "CAR" და "TRUCK"?



95) არსის ატრიბუტი , რომელიც გამოითვლება არის :

არსის ატრიბუტი, რომელიც გამოითვლება  
არის:

აირჩიეთ ერთი:

- a. ბუნდოვანი ატრიბუტი.
- b. კომპოზიტური ატრიბუტი.
- c. წარმოებული ატრიბუტი.
- d. არასავალდებულო ატრიბუტი.

სწორი პასუხია: წარმოებული ატრიბუტი.

96) რომელი მოთხოვნა დააბრუნებს თითოეული ფირმისათვის გაყიდვებიდან შემოსულ სრულ თანხას .

რომელი მოთხოვნა დააბრუნებს  
თითოეული ფირმისათვის გაყიდვებიდან  
შემოსულ სრულ თანხას

აირჩიეთ ერთი:

- a. SELECT FIRM\_NAME ,SUM(Price) FROM Sales ORDER BY FIRM\_NAME;
- b. SELECT FIRM\_NAME, SUM(Price) FROM Sales GROUP BY FIRM\_NAME;
- c. SELECT FIRM\_NAME, TOTAL(Price) FROM Sales GROUP BY FIRM\_NAME
- d. SELECT TOTAL(Price) FROM Sales WHERE FIRM\_NAME=\*

სწორი პასუხია: SELECT FIRM\_NAME,  
SUM(Price) FROM Sales GROUP BY FIRM\_NAME;

97) შემდეგი სუბიექტების გათვალისწინებით რომელი არჩევანი იქნება ყველაზე რთული ?

შემდეგი სუბიექტების გათვალისწინებით,  
რომელი არჩევანი იქნება ყველაზე რთული?

აირჩიეთ ერთი:

- a. განსახლვრეთ სუპერტიპი, რომელსაც  
ეწოდება სატრანსპორტო საშუალება და  
შექმენით თითოეული არსის ტიპის  
ქვეტიპი.
- b. განსაზღვრეთ ერთი ტიპის  
სატრანსპორტო საშუალების არსის ტიპი,  
ყველა არსის შესანახად
- c. განსაზღვრეთ არსების ფალკეული  
ტიპი.
- d. შეინახეთ მხოლოდ სატვირთო  
მანქანის არსის ტიპის.

სწორი პასუხია: განსაზღვრეთ ერთი ტიპის  
სატრანსპორტო საშუალების არსის ტიპი,  
ყველა არსის შესანახად



98) ჩამოთვლილთაგან რომელი არ არის მონაცემთა სახელის კარგი მახასიათებელი ?

ჩამოთვლილთაგან რომელი არ არის მონაცემთა სახელის კარგი მახასიათებელი?

აირჩიეთ ერთი:

- a. განმეორებადია
- b. განეკუთვნება ბიზნესის მახასიათებლებს
- c. იკითხება
- d. განეკუთვნება სისტემის ტექნიკურ მახასიათებელს

სწორი პასუხია: განეკუთვნება სისტემის ტექნიკურ მახასიათებელს ✓

99) მონაცემთა მოდელირება შეიძლება იყოს სისტემების განვითარების პროცესის ყველაზე მნიშვნელოვანი ნაწილი, რადგან

მონაცემთა მოდელირება შეიძლება იყოს სისტემების განვითარების პროცესის ყველაზე მნიშვნელოვანი ნაწილი, რადგან:

აირჩიეთ ერთი:

- a. ეს ყველაზე მარტივია.
- b. მონაცემები სისტემაში ზოგადად ნაკლებად რთულია, ვიდრე პროცესები და მნიშვნელოვან როლს ასრულებს შემუშავებაში.
- c. მონაცემები ნაკლებად სტაბილურია, ვიდრე პროცესები.
- d. მონაცემთა მახასიათებლები მნიშვნელოვანია პროგრამებისა და სისტემის სხვა კომპონენტების შემუშავებისას.

სწორი პასუხია: მონაცემთა მახასიათებლები მნიშვნელოვანია პროგრამებისა და სისტემის სხვა კომპონენტების შემუშავებისას.

100) კანდიდატი გასაღები უნდა აკმაყოფილებდეს ყველა ქვემოთ ჩამოთვლილ პირობას, გარდა :

კანდიდატი გასაღები უნდა აკმაყოფილებდეს ყველა ქვემოთ ჩამოთვლილ პირობას,

გარდა:

აირჩიეთ ერთი:

- a. გასაღებით უნდა მიეთითოს სტრიქონის პოზიცია ცხრილში.
- b. თითოეული არაგასარები ატრიბუტი ფუნქციურად დამოკიდებული უნდა იყოს მასზე.
- c. გასაღებმა ცალსახად უნდა განსაზღვროს სტრიქონი
- d. გასაღები უნდა არაჭარბი

სწორი პასუხია: გასაღებით უნდა მიეთითოს სტრიქონის პოზიცია ცხრილში.

101) არსების ყველაზე გავრცელებული სახეებია :

არსების ყველაზე გავრცელებული სახეებია:

აირჩიეთ ერთი:

- a. დაშლილი არსები.
- b. სუსტი არსები.
- c. ძლიერი არსები.
- d. ასოციაციური არსები.

სწორი პასუხია: ძლიერი არსები.

102) ქვემოთ მოცემულ ფიგურაში რა ტიპის კავშირებია ასახული?

ქვემოთ მოცემულ ფიგურაში რა ტიპის  
კავშირებია ასახული?

Student

<u>Student_ID</u>	Student_Name	Campus Address	Major
<u>Room_ID</u>	Location	<u>Student_ID</u>	

აირჩიეთ ერთი:

- a. არსის / სუსტი არსის იდენტიფიკაცია
- b. ერთი-ბევრთან
- c. სამმაგი კავშირი
- d. ბევრი-ბევრთან

სწორი პასუხია: ერთი-ბევრთან

103) ჩამოთვლილთაგან, რომელ ველს ახასიათებს ავტომატურად  
გენერირების თვისება ?

ჩამოთვლილთაგან, რომელ ველს ახასიათებს  
ავტომატურად გენერირების თვისება?

აირჩიეთ ერთი:

- a. CANDIDATE COLUMN
- b. INTEGER COLUMN
- c. UNIQUE COLUMN
- d. IDENTITY COLUMN

სწორი პასუხია: IDENTITY COLUMN

104) ორი ან მეტი ატრიბუტი , რომელსაც აქვს სხვადასხვა სახელი, მაგრამ იგივე მნიშვნელობა ეწოდება :

ორი ან მეტი ატრიბუტი, რომელსაც აქვს სხვადასხვა სახელი, მაგრამ იგივე მნიშვნელობა ეწოდება:

აირჩიეთ ერთი:

- a. სინონიმები.
- b. ჰომონიმები.
- c. ალტერნატიული ატრიბუტები.
- d. მეტსახელები.

სწორი პასუხია: სინონიმები.

105) როგორი არ უნდა იყოს არსის სახელი :

როგორი არ უნდა იყოს არსის სახელი:

აირჩიეთ ერთი:

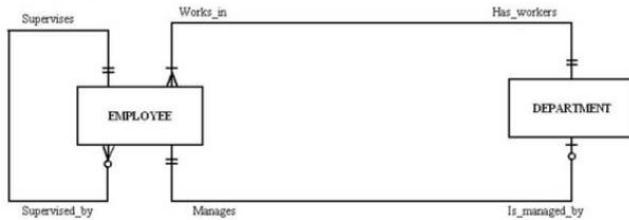
- a. ლაკონური.
- b. რაც შეიძლება მოკლე.
- c. მხოლობითი არსებითი სახელი.
- d. ორგანიზაციისთვის სპეციფიკური.

სწორი პასუხია: რაც შეიძლება მოკლე.



106) ქვემოთ მოცემული გამონათქვამებიდან რომელი შეესაბამება ნახატს

ქვემოთ მოცემული გამონათქვამებიდან  
რომელი შეესაბამება ნახატს



აირჩიეთ ერთი:

- a. თითოეული თანამშრომელი  
გაათავისუფლეს.
- b. თითოეული თანამშრომელი მუშაობს  
ერთზე მეტ განყოფილებაში.
- c. თითოეულ თანამშრომელს შეუძლია  
ზედამხედველობა გაუწიოს ერთ, ბევრ ან  
არცერთ თანამშრომელს.
- d. თითოეულ თანამშრომელს შეუძლია  
მრავალი განყოფილების მართვა.

სწორი პასუხია: თითოეული თანამშრომელს შეუძლია ზედამხედველობა გაუწიოს ერთ, ბევრ ან არცერთ თანამშრომელს.



107) ატრიბუტს, რომელიც შეიძლება დაიყოს უფრო მცირე ნაწილებად  
ეწოდება

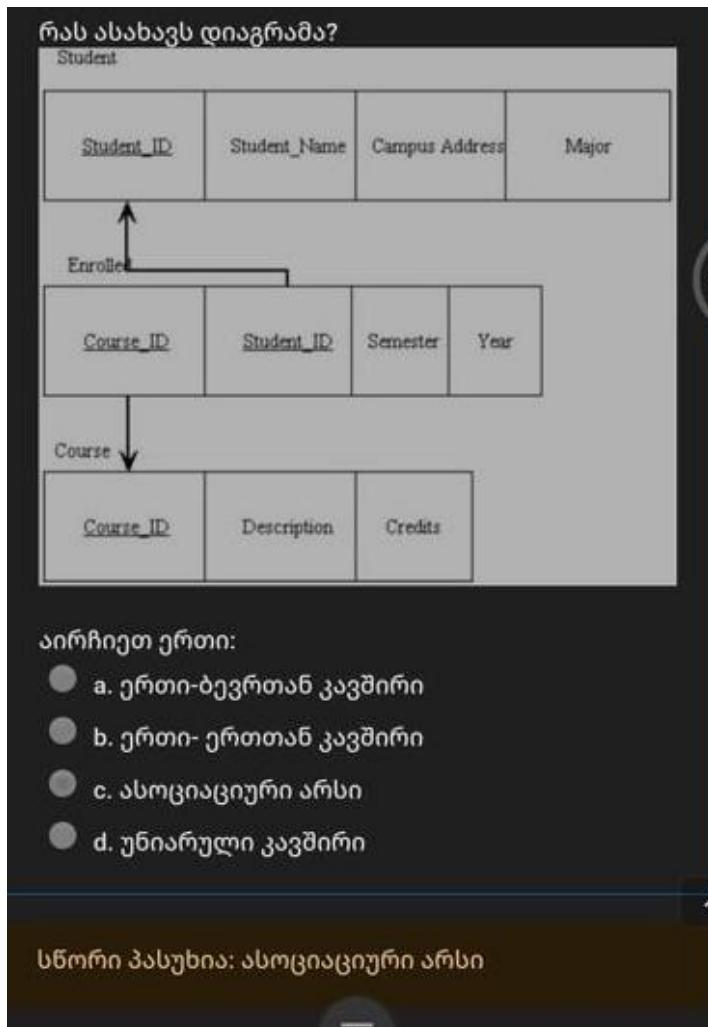
ატრიბუტს, რომელიც შეიძლება დაიყოს უფრო  
მცირე ნაწილებად, ეწოდება

აირჩიეთ ერთი:

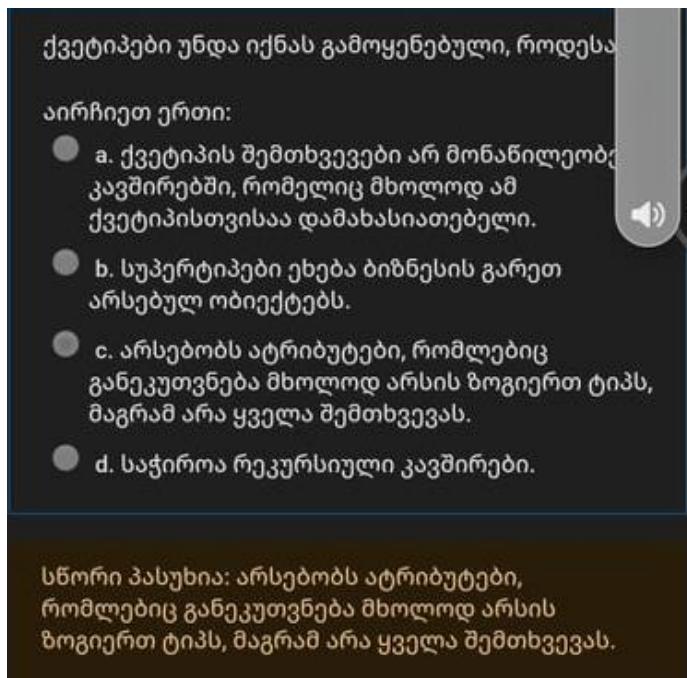
- a. კომპოზიტური
- b. მარტივი
- c. ასოციაციური
- d. რთული

სწორი პასუხია: კომპოზიტური

108) რას ასახავს დიაგრამა ?



109) ქვეტიპი უნდა იქნას გამოყენებული, როდესაც :



110) ველის მნიშვნელობას, რომელსაც მიიღებს ის თუ არაა შეყვანილი გარკვეული მნიშვნელობა ამ ეგზემპლარისათვის არის :

ველის მნიშვნელობას, რომელსაც მიიღებს ის თუ არაა შეყვანილი გარკვეული მნიშვნელობა ამ ეგზემპლარისათვის არის:

აირჩიეთ ერთი:

- a. გარანტია
- b. ნაგულისხმევი მნიშვნელობა.
- c. ნულოვანი მნიშვნელობა.
- d. დიაპაზონის კონტროლი.

სწორი პასუხია: ნაგულისხმევი მნიშვნელობა.

111) ქვემოთ ჩამოთვლილიდან რომელი არაა მონაცემთა ბაზის მართვის სისტემის (DBMS) ძირითადი მიზნები :

ქვემოთ ჩამოთვლილიდან რომელი არაა მონაცემთა ბაზის მართვის სისტემის (DBMS) ძირითადი მიზნები:

აირჩიეთ ერთი:

- a. ინტეგრირებული განვითარების გარემოს უზრუნველყოფა.
- b. მონაცემთა შენახვა.
- c. მონაცემთა განახლება.
- d. მონაცემთა შექმნა.

სწორი პასუხია: ინტეგრირებული განვითარების გარემოს უზრუნველყოფა.

112) ორგანიზაციის მონაცემების ლოგიკურ წარმოდგენას ეწოდება :

ორგანიზაციის მონაცემების ლოგიკურ  
წარმოდგენას ეწოდება:

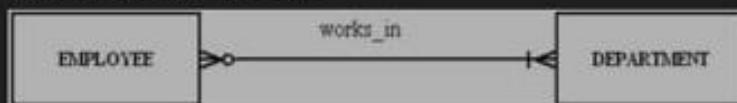
აირჩიეთ ერთი:

- a. მონაცემთა ბაზის მოდელი
- b. არსი-კავშირი მოდელი.
- c. მონაცემთა ბაზის არსების დიაგრამა.
- d. კავშირების სისტემების დაპროექტება.

სწორი პასუხია: არსი-კავშირი მოდელი.

113) ქვემოთ მოცემული გამონათქვამებიდან რომელი შეესაბამება ნახატს

ქვემოთ მოცემული გამონათქვამებიდან რომელი  
შეესაბამება ნახატს



აირჩიეთ ერთი:

- a. დეპარტამენტს უნდა ჰყავდეს მინიმუმ  
ერთი თანამშრომელი.
- b. დასაქმებულს შეუძლია ერთზე მეტ  
განყოფილებაში იმუშაოს, მაგრამ უნდა  
იმუშაოს ერთ კონკრეტულ განყოფილებაში  
მუშაობა.
- c. დეპარტამენტს შეიძლება ჰყავდეს ერთზე  
მეტი თანამშრომელი.
- d. დასაქმებული უნდა მუშაობდეს ერთზე მეტ  
განყოფილებაში.

სწორი პასუხია: დეპარტამენტს შეიძლება  
ჰყავდეს ერთზე მეტი თანამშრომელი.

114) მონაცემთ რომელ მოდელში გამოჩნდება კოდების ცხრილი ?

მონაცემთა რომელი მოდელში გამოჩნდება  
კოდების ცხრილი?

აირჩიეთ ერთი:

- a. კონცეპტუალური
- b. მონაცემთა განლაგება
- c. ლოგიკური
- d. ფიზიკური

სწორი პასუხია: ფიზიკური



115) რომელი შედეგი დაბრუნდება მოცემული მოთხოვნით ? SELECT ename , deptno FROM emp WhERE ename like 'm[^a-n]%'

რომელი შედეგი დაბრუნდება მოცემული  
მოთხოვნით? SELECT ename, deptno FROM  
emp WhERE ename like 'm[^a-n]%'

აირჩიეთ ერთი:

- a. MARTIN, MULLER
- b. MARTIN
- c. დაბრუნდება შეცდომა
- d. MULLER

სწორი პასუხია: MULLER

116) რელაციურ მოდელში მონაცემები წარმოდგენილია შემდეგნაირად :

სწორი პასუხია: ცხრილები.

117) ჩამოთვლილთაგან რომელი არ არის შეზღუდვის ტიპი :

ჩამოთვლილთაგან რომელი არ არის  
შეზღუდვის ტიპი:

აირჩიეთ ერთი:

- a. ORDER BY
- b. CHECK
- c. DEFAULT
- d. NOT NULL

სწორი პასუხია: ORDER BY

118) ჩამოთვლილთაგან რომელია რელაციის თვისებები ?

ჩამოთვლილთაგან რომელია რელაციის  
თვისებები?

აირჩიეთ ერთი:

- a. ყველა სვეტი არის რიცხვითი.
- b. რელაციაში არც ერთი რიგი არ არის  
იდენტური.
- c. რელაციაში არის  
მრავალმნიშვნელოვანი ატრიბუტები.
- d. თითოეულ ატრიბუტს იგივე სახელი  
აქვს.

სწორი პასუხია: რელაციაში არც ერთი რიგი  
არ არის იდენტური.

119) არსის ერთიანობის წესში ნათქვამია :

არსის ერთიანობის წესში ნათქვამია:

აირჩიეთ ერთი:

- a. არცერთი პირველადი გასაღები ატრიბუტი არ შეიძლება იყოს null
- b. პირველადი გასაღებს უნდა ჰქონდეს მხოლოდ ერთი ატრიბუტი.
- c. რეფერენციული მთლიანობა დაცული უნდა იყოს ყველა ობიექტში.
- d. თითოეულ არსს უნდა ჰქონდეს პირველადი გასაღები.

სწორი პასუხია: არცერთი პირველადი გასაღები ატრიბუტი არ შეიძლება იყოს null

120) რომელი მოთხოვნაა სწორად ჩაწერილი ?

რომელი მოთხოვნაა სწორად ჩაწერილი?

აირჩიეთ ერთი:

- a. SELECT SUM(OrderPrice) FROM Sales GROUP BY OrderPrice
- b. SELECT SUM(OrderPrice) FROM Sales
- c. SELECT Name,OrderDate, SUM(OrderPrice) FROM Sales ORDER BY OrderPrice
- d. SELECT SUM(OrderPrice) FROM Sales HAVING OrderPrice>100

სწორი პასუხია: SELECT SUM(OrderPrice) FROM Sales

121) ქვემოთ მოცემულ ფიგურაში რომელი ატრიბუტია  
მრავალმნიშვნელოვანი

ქვემოთ მოცემულ ფიგურაში რომელი ატრიბუტია  
მრავალმნიშვნელოვანი

EMPLOYEE
<u>Employee_ID</u>
Employee_Name
Address
Date_Employed
(Skill)
[Years_Employed]

აირჩიეთ ერთი:

- a. Years\_Employed
- b. Employee\_ID
- c. Skill
- d. Address

სწორი პასუხია: Skill

121) არსების ტიპების რაოდენობას, რომლებიც მონაწილეობენ  
ურთიერთობაში , ეწოდება :

სწორი პასუხია: ხარისხი.

122) რომელი ბრძანებით მოხდება ორი ან მეტი მსგავსი select მოთხოვნის  
შედეგის გაერთიანება ისე, რომ არ იქნება გათვალისწინებული  
დუბლირებული ჩანაწერები? (დაბრუნდება დუბლიკატებიც) .

სწორი პასუხია: UNION ALL

123) მონაცემთა ბაზის შემუშავება იწყება \_\_\_\_\_, რაც განსაზღვრავს ორგანიზაციული მონაცემთა ბაზის დიაპაზონს და ზოგად შინაარსს.

მონაცემთა ბაზის შემუშავება იწყება \_\_\_\_\_, რაც განსაზღვრავს ორგანიზაციული მონაცემთა ბაზის დიაპაზონს და ზოგად შინაარსს.

აირჩიეთ ერთი:

- a. საწარმოს მონაცემთა მოდელირებით
- b. ორგანიზაციული მონაცემების მოდელირებით
- c. მონაცემთა ბაზის დიზაინით
- d. გადამკვეთი ფუნქციური ანალიზით

სწორი პასუხია: საწარმოს მონაცემთა მოდელირებით



124) ფიზიკური ფაილების და მონაცემთა ბაზების შემუშავების მოთხოვნაა :

ფიზიკური ფაილების და მონაცემთა ბაზების შემუშავების მოთხოვნაა:

აირჩიეთ ერთი:

- a. მონაცემთა ყველა ტიპის დადგენა.
- b. შექმნილი ფიზიკური ცხრილები.
- c. ნორმალიზებული რელაციები
- d. განხორციელების დასრულება.

სწორი პასუხია: ნორმალიზებული რელაციები

125) რა ფუნქცია აქვს TRUNCATE ბრძანებას ?

რა ფუნქცია აქვს TRUNCATE ბრძანებას?

აირჩიეთ ერთი:

- a. შლის ცხრილში ყველა ჩანაწერს;
- b. შლის მონაცემთა ბაზას
- c. ამოწმებს ცხრილში ჰირველადი გასაღების არსებობას;
- d. შლის ცხრილს მონაცემთა ბაზიდან;

სწორი პასუხია: შლის ცხრილში ყველა ჩანაწერს;

126) ატრიბუტი, რომელიც უნდა არსებობდეს თითოეული არსის (ან კავშირის) ეგზემპლარისთვის არის

ატრიბუტი, რომელიც უნდა არსებობდეს თითოეული არსის (ან კავშირის) ეგზემპლარისთვის არის

აირჩიეთ ერთი:

- a. მრავალმნიშნელოვანი ატრიბუტი.
- b. სავალდებულო ატრიბუტი.
- c. არასავალდებულო ატრიბუტი.
- d. კომპოზიტური ატრიბუტი.

სწორი პასუხია: სავალდებულო ატრიბუტი.



# მონაცემთა ბაზის შემუშავების ძირითადი ასპექტები

## შესავალი

ბოლო ორი ათეული წლის განმავლობაში, მონაცემები უმრავლესი თანამედროვე ორგანიზაციებისათვის სტრატეგიული აქტივი გახდა. მონაცემთა ბაზები გამოიყენება თითქმის ყველა ტიპის ორგანიზაციაში მონაცემთა შეანხვის, დამუშავების და მოძიებისათვის, მათ შორის ბიზნესის, ჯანმრთელობის დაცვის, განათლების, სამთავრობო ოგრანიზაციებსა და ბიბლიოთეკებში. მონაცემთა ბაზების ტექნოლოგია ჩვეულებრივ გამოიყენება პირად პერსონალურ კომპიუტერებზე და ამა თუ იმ საწარმოოს (ორგანისაციის) თანამშრომლების მიერ, საწარმოთა განაწილებული პროგრამების მომხმარებისას. ასევე მონაცემთა ბაზებზე წვდომა შესაძლებელია მომხმარებელზე ორიენტირებული მრავალფეროვანი ტექნოლოგიების საშუალებით, მაგალითად: ავტომატური გადახდის აპარატები, ვებ-ბრაუზერები, სმარტფონები და სხვადასხვა ინტელექტუალური საყოფაცხოვრებო და საოფისე დანართები. ვებზე დაფუძნებული პროგრამების უმეტესობა მონაცემთა ბაზის დანართებზეა აგებული.

ამდენად მონაცემთა ბაზების მნიშვნელობა ეჭვს არ იწვევს ადამიანის ცხოვრების და მოღვაწეობის სფეროში. იმისათვის, რომ შევძლოთ სრულად გავერკვიოთ მონაცემთა ბაზების აგება/დაგეგმარების (დაპროექტება) და გამოყენება/მართვის (მენეჯმენტის) საკითხებში აუცილებელია მირითად ცნებების და ტერმინების მკაფიოდ განსაზღვრა. ეს მირითადი ტერმინებია: მონაცემები (Data), მონაცემთა ბაზა (Database), მონაცემთა ბაზის მართვის სისტემა (Database Management System), მონაცემთა მოდელი (Data Model), ინფორმაცია (Information), მეტამონაცემები (Metadata), საწარმოს მონაცემთა მოდელი (Enterprise Data Model), არსი (Entity), რელაციური მონაცემთა ბაზა (Relational Database), მონაცემთა ბაზის პროგრამა (Database Application), მონაცემთა საწყობი (Data Warehouse), მონაცემთა დამოუკიდებლობა (Data Independence), საცავი (repository), მომხმარებლის ხედი (User View), საწარმოს მონაცემების მოდელირება (Enterprise Data Modeling), მოქნილი პროგრამული უზრუნველყოფის შემუშავება (Agile Software Development), მონაცემთა მოდელირებისა და დიზაინის ინსტრუმენტები (Data Modeling And Design Tools) კონცეპტუალური სქემა (Conceptual Schema), ლოგიკური სქემა (Logical Schema), ფიზიკური სქემა (Physical Schema).

## ძირითადი ცნებები და განმარტებები

მონაცემთა ბაზის ერთი მკვეთრად ჩამოყალიბებული განმარტება არ არსებობს. სხვადასხვა ავტორები სხვადასხვაგვარად აყალიბებენ მას, მაგრამ არსი ერთია - **მონაცემთა ბაზა** ლოგიკურად დაკავშირებული მონაცემების ორგანიზებული კოლექციაა, რომელიც წინასწარ განსაზღვრული წესს ექვემდებარება. მონაცემთა ბაზა შეიძლიბა იყოს სხვდასხვა ზომის რამდენიმე მეგაბაიტი მონაცემიდან (პესონალური კონტაქტების მონაცემთა ბაზა სმარტფონში) პეტაბაიტებამდე (მსხვილი კორპორაციის მონაცემთა ბაზა, რომელიც დიდი

მონაცემებს წარმოადგენს). არსი ერთია და მათ აუცილებლად ესაჭიროება დალაგება/დაპროეტება და შესატყვისი პროგრამული უზრუნველყოფის შემუშავება.

### მონაცემთა ბაზა ლოგიკურად დაკავშირებული მონაცემების ორგანიზებული კოლექცია

რადგან მთავარი „მოქმედი პირი“ ჩვენს საქმიანობაში მონაცემებია, შევეცადოთ მის განმარტებას. ICTs<sup>1</sup> სფეროში „მონაცემი“ აღწერს არსებთან და მოვლენებთან დაკავშირებულ ფაქტებს, რომელთა ჩაწერა და შენახვა შესაძლებელია კომპიუტერში. მაგალითად უნივერსიტეტის მონაცემთა ბაზა შეიძლება მოიცავდეს სტუდენტის სახელს, მისამართს, ტელეფონს. ასეთი ტიპის მონაცემებს სტრუქტურირებულ მონაცემებს უწოდებენ. სტრუქტურირებული მონაცემაბის ყველაზე მნიშვნელოვანი ტიპებია რიცხვითი, სიმბოლური, თარიღი. სტრუქტურული მონაცემების შენახვა ხდება ტაბულების ფორმით (ცხრილებში, რელაციებში, მასივებში, ელექტრონულ ცხრილებში და სხვ.) და უფრო ხშირად გვხვდებიან ტრადიციულ მონაცემთა ბაზებსა და მონაცემთა საწყობებში (საცავებში).

მონაცემთა ტრადიციული განმარტება მოითხოვს ახალი რეალობის ასახვას: დღესდღეობით მონაცემთა ბაზები სტრუქტურირებულ მონაცემებთან ერთად, გამოიყენება ისეთი არსების შესანახად, როგორიცაა დოკუმენტები, ელექტრონული ფოსტა, სხვადასხვა სოციალური ქსელების შეტყობინებები (Tweeter, Facebook და სხვ.), GPS ინფორმაცია, რუკები, ფოტოები, ბგერითი და ვიდეო სეგმენტები. მაგალითად უნივერსიტეტის მონაცემთა ბაზა მოიცავს სტუდენტის სურათს და მის საკონტაქტო ინფორმაციასაც. ბაზა შეიძლება მოიცავდეს რაიმე ვიდეო ინსტრუქციასაც. ასეთი ტიპის მონაცემები გამოისახება როგორც არასტრუქტურირებული მონაცემები ან როგორც მულტიმედია მონაცემები. დღესდღეობით ხდება სტრუქტურირებული და არასტრუქტურირებული მონაცემების შერწყმა ერთ მონაცემთაბაზაში მულტიმედია გარემოს შექმნის გზით.

თანამედროვე ფართო გაგებით „მონაცემის“ გამნარტება მოიცავს როგორც სტრუქტურირებულ ასევე არასტრუქტურირებულ ტიპს.

**მონაცემი** არის არსებისა და მოვლენების შენახული წარმოდგენა, რომლებსაც გააჩნიათ კონკრეტული მნიშვნელობა და მნიშვნელოვანია მომხმარებლის გარემოში.

ინფორმაცია თანამედროვე ყოფიერების განუყოფელი ნაწილია. ინფორმაცია ეფუძნება მონაცემებს, თუმცა მონაცემები თავისთავად ყოველთვის არ აღიქმება ინფორმაციად. მიუხედავად იმისა, რომ მონაცემები და ინფორმაცია მჭიდროდაა ერთმანეთთან დაკავშირებული, აუცილებელია მათი განსხვავება. ინფორმაცია შეგვიძლია განვმარტოთ,

<sup>1</sup> ICTs - საინფორმაციო და საკომიუნიკაციო ტექნოლოგიები

როგორც მონაცემი, რომელიც დამუშავდა იმგვარად, რომ გაიზარდა ამ მონაცემის გამოყენებლის ცოდნა. სიცხადისთვის განვიხილოთ მაგალითი.

ჩხიტუნიძე საბა	94001056581
ბაბლიძე შოთა	54001056329
ჯანიაშვილი ნათია	36001051107
ცარციძე ნინო	08001111717
წიკლაური სალომე	22001022880
გვასალია ვალერიან	19001105309
გოგია ბაჩანა	19001103122

წარმოდგენილი მაგალითი შეესაბამება მონაცემის განმარტებას, მაგრამ ის ამ ფორმით უსარგებლოა. ჩვენ შეგვიძლია მივხვდეთ, რომ ესაა ადამიანთა სია პირადი ნომრებით, მაგრამ რისი სიაა ამის შესახებ არაფრის თქმა არ შეგვიძლია. კონტექსტის დამატებით შესაძლებელი ბევრი რამის შეცვლა

კურსი: CS302		
სტუდენტი	ID	შფასება
ჩხიტუნიძე საბა	94001056581	71
ბაბლიძე შოთა	54001056329	84
ჯანიაშვილი ნათია	36001051107	52
ცარციძე ნინო	08001111717	69
წიკლაური სალომე	22001022880	93
გვასალია ვალერიან	19001105309	55
გოგია ბაჩანა	19001103122	47

რამდენიმე მონაცემის დამატებით და გარკვეული სტრუქტურის უზრუნველყოფით ჩვენ უკვე შეგვიძლია განვსაზღვროთ საწყისი მონაცემების არსი. ამით ჩვენ მივიღეთ სასარგებლო ინფორმაცია იმ მონაცემებიდან, რომლებიც საწყის ეტაპზე ერთი შეხედვით უსარგებლო იყო.

პრაქტიკაში მონაცემთა ბაზა შიძლე მოიცავდეს როგორ მონაცემებს ასევე ინფორმაციას, ან ნორივეს ერთად.

**ინფორმაცია** არის მონაცემი, რომელიც დამუშავდა იმგვარად, რომ გაიზარდა ამ მონაცემის გამომყენებლის ცოდნა.

ჩვენ აღვნიშნეთ, რომ მონაცემი სასარგებლო ხდება მხოლოდ მაშინ თუ მას შეუსაბამებთ გარკვეულ კონტექსტს (შინაარსს). მონაცემისათვის კონტექსტის უზრუნველყოფის პირველად მექანიზმს წარმოადგენს მეტამონაცემი. **მეტამონაცემი** მონაცემია, რომელიც აღწერს საბოლოო მომხმარებლის მონაცემის თვისებას ან მახასიათებელს და ამ მონაცემის შინაარსს. ზოგიერთი ტიპიური მახასიათებელი აღწერს მონაცემის სახელს, განმარტებას, სიგრძეს (ზომას) და დასაშვებ მნიშვნელობას. მეტამონაცემი არწერს მონაცემის კონტექტსტს მონაცემთა წყაროს ჩათვლით, სადაც მონაცემი ინახება, რასაც მიეკუთვნება და რაშიც გამოიყენება. ხშირად მეტამონაცემები აღიქმება, როგორც „მონაცემები მონაცემთა შესახებ“.

ჩვენს მიერ მოყვანილი მაგალითისათვის მეტამონაცემები შეგვიძლია წარმოვადგინოთ შემდეგი სახით:

მონაცემთა ჯგუფი	მეტამონაცემები					
დასახელება	ტიპი	სიგრძე	min	max	აღწერა	წყარო
საგანი	Alphanumeric	7			საგნის ID	სასწავლო ნაწილი
სტუდენტი	Alphanumeric	30			სტუდენტის გვარი, სახელი	სტუდენტის პირადი ინფორმაცია
ID	Integer	11			სტუდენტის პირადი ნომერი	სტუდენტის პირადი ინფორმაცია
შეფასება	Integer	3	0	100	სტუდენტის შეფასება	სასწავლო ნაწილი

აუცილებლად უნდა გავითვალისწინოთ განსხვავება მონაცემებსა და მეტამონაცემებს შორის. მეტამონაცემები ერთხელ ამოიღება მონაცემებიდან. ეს მეტამონაცემები აღწერს მონაცემების თვისებებს, მაგრამ ამ მონაცემებისგან განცალკევებულია. მეტამონაცემები საშუალებას აძლევს მონაცემთა ბაზის შემმუშავებლებსა და მომხმარებლებს გააცნობიერონ რა მონაცემები არსებობს, რას ნიშნავს მონაცემები და როგორ უნდა განასხვაონ მონაცემთა ერთეულები, რომლებიც ერთი შეხედვით ჰგავს ერთმანეთს. მეტამონაცემების მართვა

ისეთივე მნიშვნელოვანია, როგორც ასოცირებული მონაცემების მართვა, რადგან მონაცემები მკაფიო მნიშვნელობის გარეშე შეიძლება იყოს დამაბნეველი, არასწორად ინტერპრეტირებული ან მცდარი. როგორც ჩესი, მეტამონაცემების დიდი ნაწილი ინახება მონაცემთა ბაზის ნაწილად და მათი მიღება შეიძლება იგივე მიღვომების გამოყენებით, რომლებიც გამოიყენება მონაცემთა ან ინფორმაციის მისაღებად. მონაცემების შენახვა შესაძლებელია ფაილებში (ვთქვათ Excel -ში) ან მონაცემთა ბაზაში.

## ფაილური დამუშავების სისტემები

კომპიუტერზე დაფუძნებული მონაცემთა დამუშავების პირველად ეტაპზე მონაცემთა ბაზა არ არსებობდა. ბიზნესის დანართებისათვის აუცილებელი იყო დიდი მონაცემთა ფაილების შენახვა, მანიპულირება და მოძიება. ამ მიზნით შემუშავდა ფაილების დამუშავების კომპიუტერული სისტემები.

მიუხედავად იმისა, რომ ეს სისტემები დროთა განმავლობაში ვითარდებოდა, მათი ძირითადი სტრუქტურა და დანიშნულება შეიცვალა რამდენიმე ათწლეულის განმავლობაში.

ბიზნესის აპლიკაციების გართულებისთანავე, აშკარა გახდა, რომ ფაილების დამუშავების ტრადიციულ სისტემებს მთელი რიგი ხარვეზები და შეზღუდვები გააჩნდათ. შედეგად უმეტეს ბიზნეს პროგრამებში, ეს სისტემები შეიცვალა მონაცემთა ბაზის დამუშავების სისტემებით. მიუხედავად ამისა ფაილების დამუშავების სისტემების ცოდნა სასარგებლოა ასეთ სისტემებში არსებული პრობლემებისა და შეზღუდვების გაგებისათვის, რაც დაგვეხმარება თავიდან აიცილოთ იგივე პრობლემები მონაცემთა ბაზის სისტემების შექმნისას. უნდა აღინიშნოს, რომ Excel ფაილი, ზოგადად, იმავე კატეგორიაში შედის, როგორც ფაილური სისტემა და აქვთ იგივე ნაკლოვანებები, რაც ფაილურ სისტემებს.

ფაილური სისტემის პრობლემების შეზღუდვების ცხადი სახით წარმოდგენისათის განვიხილოთ მაგალითი: ფაილის დამუშავების სისტემები **Pine Valley Furniture** ავეჯის კომპანიაში.

კომპანიაში არსებული პროგრამებმა იყენებდა ფაილის დამუშავების ტრადიციული მიდგომას. საინფორმაციო სისტემების დიზაინის ეს მიდგომა აკმაყოფილებდა ცალკეული განყოფილებების მონაცემთა დამუშავების მოთხოვნებს, და არა ორგანიზაციის მთლიან ინფორმაციულ საჭიროებებს. ინფორმაციული სისტემების ჯგუფი, როგორც ჩესი, პასუხობდა მომხმარებელთა მოთხოვნებს ახალი სისტემების შესახებ ახალი ინდივიდუალური პროგრამების შემუშავებით (ან შემწით), როგორიცაა ინვენტარის კონტროლი, დებიტორული დავალიანება ან ადამიანური რესურსების მართვა. არ იყო საერთო რუკა, გეგმა ან მოდელი პროგრამის ზრდისათვის. ფაილების დამუშავების მიდგომაზე დაფუძნებული კომპიუტერული პროგრამებიდან სამი ნაჩვენებია ნახაზზე **1-2**. - შეკვეთის შევსება, ზედნადების შედგენა და ხელფასი.

ნახაზზე ასევე მოცემულია მონაცემთა ძირითადი ფაილები, რომლებიც დაკავშირებულია თითოეულ პროგრამასთან. ფაილი ეს არის დაკავშირებული ჩანაწერების კრებული. მაგალითად, შეკვეთების შევსების სისტემას გააჩნია სამი ფაილი: Customer Master, Inventory

Master და Back Order. ყურადღება მივაჭიოთ იმ ფაქტს, რომ ხდება სამი დანართის მიერ გამოყენებული ფაილების დუბლირება, რაც დამახასიათებელია ფაილების დამუშავების სისტემებისთვის.

**ფაილური დამუშავების სისტემების უარყოფითი მხარეები**

რამდენიმე უარყოფითი მხარე, რომელიც დაკავშირებულია ჩვეულებრივ ფაილების დამუშავების სისტემებთან, ჩამოთვლილია ცხრილში 1-2. მნიშვნელოვანია ამ საკითხების გაგება და გააზრება, რადგან მისი ცოდნა გავლენას ახდენს თანამედროვე მონაცემთა ბაზის მართვის პრაქტიკაზე და ეს ნაკლოვანებები შეიძლება გახდეს თქვენს მიერ შემუშავებული მონაცემთა ბაზის პრობლემაც.

#### „პროგრამა-მონაცემი“ დამოკიდებულება

ფაილის აღწერა ინახება იმ თითოეული მონაცემთა ბაზის დანართი პროგრამის ფარგლებში, რომელიც მიმართავს მოცემულ ფაილს. მაგალითად, ზედნადების შედგენის სისტემაში ნახაზ 1-2-ში, A პროგრამა მიამრთავს ინვენტარის ფასების ფაილს მომხმარებელთა მთავარ ფაილის საფუძველზე. იმის გამო, რომ პროგრამა შეიცავს ამ ფაილების დეტალურ აღწერას, ფაილის სტრუქტურის ნებისმიერი ცვლილება მოითხოვს ფაილის აღწერილობის შეცვლას ყველა იმ პროგრამისთვის, რომლებიც იყენებს ამ ფაილს.

**მონაცემთა ბაზის დანართი - გამოყენებითი პროგრამა (ან მასთან დაკავშირებული პროგრამების ნაკრები), რომელიც გამოიყენება მონაცემთა ბაზის მომხმარებელთა სახელით მონაცემთა ბაზის მთელი რიგი აქტივობების შესასრულებლად (შექმნა, წაკითხვა, განახლება და წაშლა).**

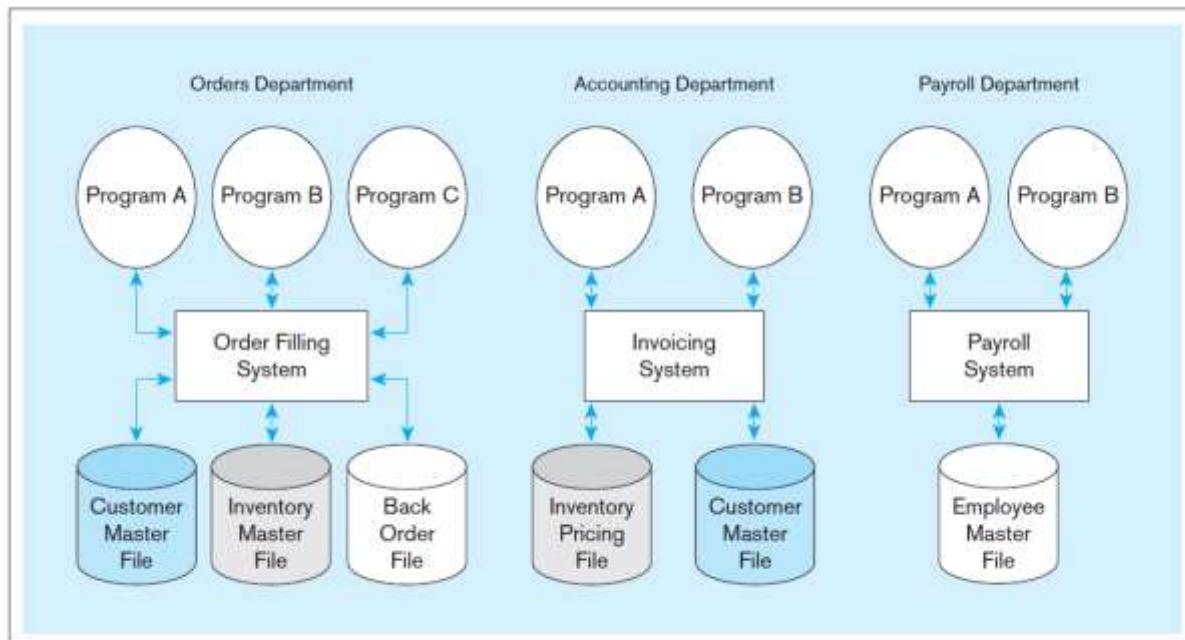


FIGURE 1-2 Old file processing systems at Pine Valley Furniture Company

1-2 ნახაზზე მომხმარებელთა ძირითადი ფაილი გამოიყენება შეკვეთების შევსების სისტემაში და ზედნადებების სისტემაში. დავუშვათ, ამ ფაილში არსებულ ჩანაწერებში გადაწყდა მომხმარებლის მისამართის ველის სიგრძის შეცვლა 30-დან 40 სიმბოლომდე. შესაბამისად უნდა შეიცვალოს ამ მონაცემის გამოიყენებელი თითოეული პროგრამის (ხუთამდე პროგრამა) ფაილის აღწერილობა. ხშირად ძნელია თუნდაც ყველა პროგრამის მოქმედნა, რომლებმაც ეს ცვლილებებმა განიცადეს. უფრო მეტიც, შეცდომები ხშირად ასეთი ცვლილებების შეტანისას ჩნდება.

#### მონაცემთა დუბლირება

იმის გამო, რომ ფაილების დამუშავების სისტემებში პროგრამები ხშირად დამოუკიდებლად ვითარდება, მონაცემების ფაილების დუბლირება უფრო კანონზომიერებას წარმოადგენს, ვიდრე გამონაკლისს.

მაგალითად, ნახ. 1-2-ში, შეკვეთის შევსების სისტემა შეიცავს ინვენტარის მთავარ ფაილს, ხოლო ზედნადების შედგენის სისტემა შეიცავს ინვენტარის ფასების ფაილს. ეს ფაილები შეიცავს Pine Valley ავეჯის კომპანიის პროდუქტების აღწერის მონაცემებს, როგორიცაა პროდუქტის აღწერა, ერთეულის ფასი და ხელთ არსებული რაოდენობა. ასეთი დუბლირება ფლანგველობაა, რადგან ის მოითხოვს დამატებით ადგილს შენახვისათვის და გაზრდილ ძალისხმევას ფაილის განახლებისათვის. ასევე შესაძლოა მონაცემთა ფორმატები იყოს არათანამიმდევრული ან მონაცემთა მნიშვნელობები არ ეთანხმებოდეს ერთმანეთს (ან ორივე). ფაილების დამუშავების სისტემებში საიმედო მეტამონაცემების დადგენა ძალიან რთულია. მაგალითად, ერთი და იგივე მონაცემთა ერთეულს შეიძლება ჰქონდეს სხვადასხვა სახელები სხვადასხვა ფაილებში ან, პირიქით, იგივე სახელი შეიძლება გამოყენებულ იქნას სხვადასხვა ფაილების მონაცემთა სხვადასხვა ერთეულებისთვის.

#### მონაცემთა გაზიარების აშეზღუდულობა

ფაილების დამუშავების ტრადიციული მიდგომის გამოყენებისას, თითოეულ აპლიკაციას აქვს საკუთარი პირადი ფაილები და მომხმარებლებს მცირე შესაძლებლობა აქვთ გაზიარონ მონაცემები საკუთარი პროგრამების მიღმა. მაგალითად, ნახ. 1-2-ში ბუღალტრული აღრიცხვის დეპარტამენტის მომხმარებლებს აქვთ წვდომა ზედნადებების სისტემაზე და მის ფაილებზე, მაგრამ მათ, ალბათ, არ აქვთ წვდომა ანგარიშის შევსების სისტემაზე ან სახელფასო სისტემაზე და მათ ფაილებზე. მენეჯერები ხშირად ხვდებიან, რომ მათვის სასურველი ანგარიშის მიღება მოითხოვს დიდ ძალისხმევას დაპროგრამიების მხრივ, რადგან მონაცემები უნდა ამოკრიფონ რამდენიმე შეუთავსებელი ფაილიდან. როდესაც სხვადასხვა ორგანიზაციული ერთეულები ფლობენ ასეთ განსხვავებულ ფაილებს, აუცილებელია დამატებითი ბარიერების გადალახვა.

#### შემუსავება/განვითარებისათვის საჭირო ხანგრძლივი დრო

ტრადიციული ფაილების დამუშავების სისტემაში, თითოეული ახალი დანართი მოითხოვს, რომ შემუშავებელმა დაიწყოს ნულიდან - ფაილის ახალი ფორმატისა და აღწერილობის შექმნით, და ყოველი ახალი დანართისათვის ფაილზე წვდომის ლოგიკის დაწერით. ყოველივე ამის განსახორციელებლად საჭირო დრო არ შეესაბამება თანამედროვე ბიზნეს გარემოს, რომლისათვისაც ბაზარზე გასვლის დრო (ან ინფორმაციის სისტემისთვის წარმოების დრო) ბიზნესის წარმატების მთავარი ფაქტორია.

## პროგრამის მომსახურებისათვის საჭირო გადაჭარბებული რესურსები

ყველა წინა ფაქტორები ქმნიან პროგრამის ქმედითობის მხარდაჭერას საჭირო მომსახურებისათვის აუცილებელი რესურსებზე დიდ დატვირთვას ორგანიზაციებში, რომლებიც ეყრდნობოდნენ ფაილების დამუშავების ტრადიციულ სისტემებს. სინამდვილეში, საინფორმაციო სისტემის განვითარების ბიუჯეტის 80 პროცენტი შეიძლება დაეთმოს პროგრამების შენარჩუნებას ასეთ ორგანიზაციებში. ეს თავის მხრივ ნიშნავს, რომ რესურსები (დრო, ხალხი და ფული) არ იხარჯება ახალი პროგრამების შემუშავებაზე.

მნიშვნელოვანია აღინიშნოს, რომ ფაილის დამუშავების მრავალი უარყოფითი მხარე შეიძლება იყოს მონაცემთა ბაზის შეზღუდვაც, თუ ორგანიზაცია სათანადოდ არ იყენებს მონაცემთა ბაზის მიდგომას. მაგალითად, თუ ორგანიზაცია შეიმუშავებს მრავალ ცალკეულ მართულ მონაცემთა ბაზას (ვთქვათ, თითოეული განყოფილებისთვის ან ბიზნესის ფუნქციისთვის) მეტამონაცემების მცირე ან საერთოდ არა კოორდინაციით, შეიძლება მოხდეს მონაცემთა უკონტროლო დუბლირება, მონაცემთა გაზიარების შეზღუდვა, განვითარების და მხარდაჭერის ხარჯების და დაროის სიჭარბე. ამრიგად, მონაცემთა ბაზის მიდგომა, ისეთივე ორგანიზაციული მონაცემების მართვის საშუალებაა, როგორც ამ მონაცემთა განსაზღვრის, შექმნის, შენარჩუნებისა და გამოყენების ტექნოლოგიების ერთობლიობა.

## **მონაცემთა ბაზა - თანამედროვე მიდგომა**

მიდგომა, რომელიც გულისხმობს მონაცემთა დამუშავებისათვის მონაცემთა ბაზის გამოყენებას ICTs ერთერთი ყველაზე ფართოდ გამოყენებადი არეა. ამ მიდგომის სრული აღქმისა და გაგებისათვის აღვწეროთ, როგორ შეუძლია მას გადალახოს ის შეზღუდვები, რაც დაკავშირებულია მონაცემთა დამუშავების ფაილურ მიდგომასთან.

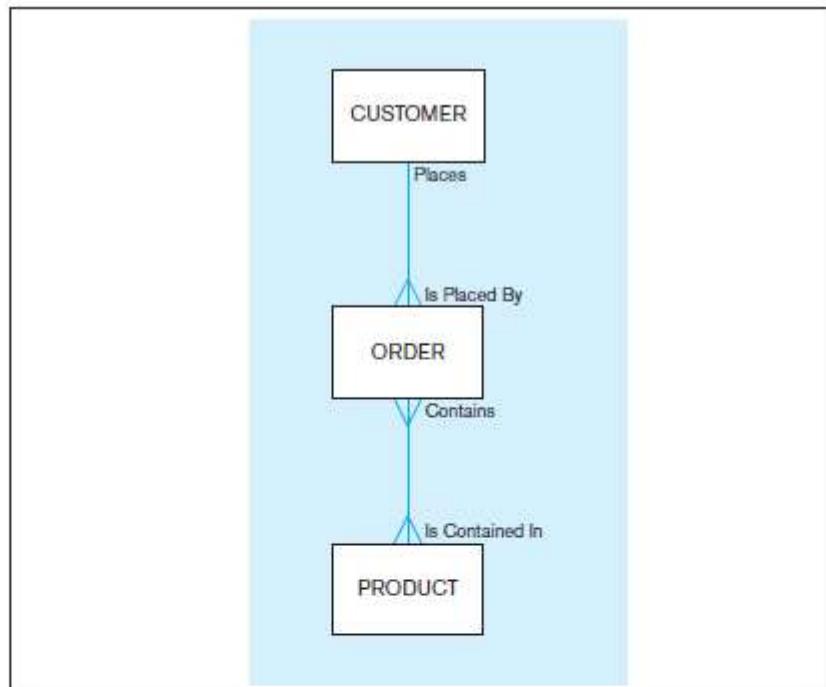
### **მონაცემთა მოდელი**

მონაცემთა ბაზის სწორად შემუშავება ფუნდამენტურია ისეთი მონაცემთა ბაზის შესაქმნელად, რომელიც დააკმაყოფილებს მომხმარებელთა საჭიროებებს. მონაცემთა მოდელები ასახავს მონაცემების ბუნებას და მათ შორის კავშირებს და გამოიყენება აბსტრაქციის სხვადასხვა დონეზე მონაცემთა ბაზის კონცეპტუალიზაციის და დაპროექტების პროცესში. მონაცემთა ბაზის ეფექტურობა პირდაპირ კავშირშია მონაცემთა ბაზის სტრუქტურასთან. არსებობს სხვადასხვა გრაფიკული სისტემა, რომლებიც ამ სტრუქტურის გადმოსცემის საშუალებას იძლევა და გამოიყენება მონაცემთა მოდელების შესაქმნელად, რომელთა გაგებაც შეუძლიათ საბოლოო მომხმარებლებს, სისტემების ანალიტიკოსებს და მონაცემთა ბაზის შემუშავებლებს. მონაცემთა ტიპიური მოდელი შედგება არსებისგან (entities) (არსებისაგან), ატრიბუტებისგან ( attributes) და დამოკიდებულებებისაგან (relationships) (კავშირისაგან). მონაცემთა მოდელირების ყველაზე გავრცელებული წარმოდგენა არის „არსი-დამოკიდებულება“ („არსი-კავშირი“, „entity-relationship ) მოდელი.

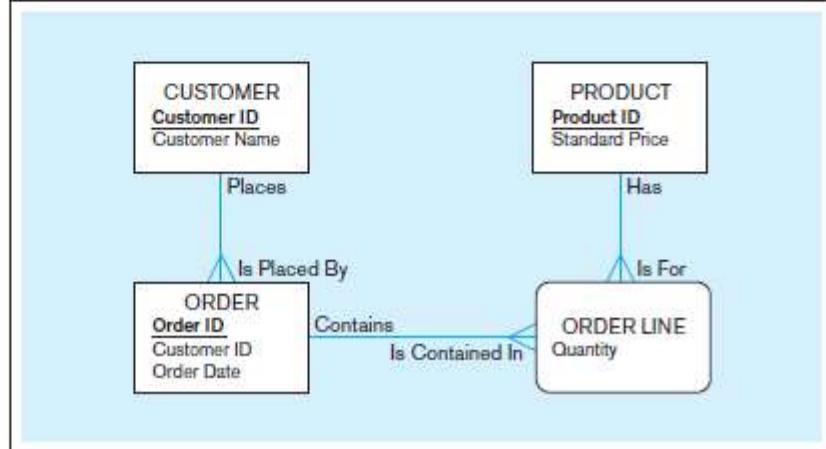
**მონაცემთა მოდელი** - გრაფიკული სისტემები,  
რომელიც გამოიყენება მონაცემების  
მახასიათებლებისა და ურთიერთკავშირის  
გამოსახატავად.

**არსები (არსები)** - ნახ. 1-3 მოცემულ მაგალითში მომხმარებელი-CUSTOMER და **შეკვეთა-ORDER** არიან არსები, რომელთა შესახებ ინფორმაციაც ბიზნესს აინტერესებს და ესაჭიროება. ისინი მოიხსენიებიან როგორც "არსები" (ან არსები). არსი (არსი) არსებით სახელს ჰგავს იმით, რომ ის აღწერს პიროვნებას, ადგილს, საგანს, მოვლენას ან კონცეფციას ბიზნესის გარემოში, რომლთა შესახებ ინფორმაცია უნდა ჩაიწეროს და შენარჩუნდეს. CUSTOMER და ORDER არსებია ნახაზზე 1-ზა. მონაცემებს, რომელთა მიღებაც გვსურს ამ არსებთან დაკავშირებით (მაგალითად, მომხმარებლის სახელი) ატრიბუტს უწოდებენ. მონაცემები იწერება მრავალი მომხმარებლისთვის. თითოეული მომხმარებლის ინფორმაცია მოიხსენიება როგორც მომხმარებელთა ეგზემპლიარი.

**FIGURE 1.3** Comparison of enterprise and project-level data models  
 (a) Segment of an enterprise data model



(b) Segment of a project data model



**არსები (არსები)** - არსები, რომელთა შესახებ  
 ინფორმაციაც ბიზნესს აინტერესებს და ესაჭიროება.

**დამოკიდებულება (კავშირი)** - კარგად სტრუქტურირებული მონაცემთა ბაზა ამყარებს ურთიერთდამოკიდებულებას (კავშირს). ორგანიზაციულ მონაცემებში არსებულ არსებს (არსებს) შორის ისე, რომ შესაძლებელი იყოს სასურველი ინფორმაციის მიღება. კავშირების უმეტესობა არის ერთი-ბევრთან (1: M) ან ბევრი-ბევრთან (M:N) ტიპის. მაგალითის სახით ეს შეიძლება ასე განიმარტოს „მომხმარებელს შეუძლია გააკეთოს ერთზე მეტი შეკვეთა კომპანიაში. ამასთან, თითოეული შეკვეთა, როგორც წესი, დაკავშირებულია მხოლოდ კონკრეტულ მომხმარებელთან“.

დიაგრამა 1-3a აჩვენებს მომხმარებელთა 1:M კავშირს, რომლებსაც შეუძლიათ გააკეთონ ერთი ან მეტი შეკვეთა; კავშირი 1:M აღინიშნება „ბატის ფეხით“, რომელიც ერთვის მართვულების (ერთეულს), რომელსაც ეწოდა ORDER. როგორც ჩანს, ეს კავშირი მსგავსია ნახატებში 1-3a და 1-3b. ამასთან, შეკვეთასა (ORDER ) და პროდუქტს (PRODUCT) შორის კავშირი არის M: N. შეკვეთა შეიძლება იყოს ერთი ან მეტი პროდუქტისთვის, და პროდუქტი შეიძლება მოთავსდეს ერთზე მეტ შეკვეთზე. საგულისხმოა, რომ სურათი 1-3a წარმოადგენს საწარმოს დონის მოდელს, რომელშიც საჭიროა მხოლოდ უფრო მაღალი დონის დეტალიზაციის დამატება მომხმარებლების, შეკვეთების და პროდუქტების კავშირების შესახებ. 1-3 ბ ნახატზე ნაჩვენები პროექტის დონის დიაგრამა შეიცავს დეტალების დამატებით დონეს, მაგალითად შეკვეთის შემდგომ დეტალებს.

მონაცემთა ბაზების თანამედროვე მიდგომებში დიდი ადგილი უკავია რელაციურ მონაცემთა ბაზებს.

**რელაციური მონაცემთა ბაზები** ადგენენ კავშირებს არსებს შორის ფაილში შეტანილი საერთო ველების საშუალებით. ამ დამოკიდებულებას ეწოდება რელაცია. 1-3 ნახაზზე გამოსახულ მონაცემთა მოდელებში, კავშირი მომხმარებელსა და მომხმარებელის შეკვეთას შორის, დამყარებულია მომხმარებლის ნომრის მომხმარებლის შეკვეთები ჩასმით. ამრიგად, მომხმარებლის საიდენტიფიკაციო ნომერი ჩაისმის ფაილში (ან რელაციაში), სადაც ინახება მომხმარებლის ინფორმაცია, როგორიცაა სახელი, მისამართი და ა.შ. ყოველთვის, როდესაც მომხმარებელი აკეთებს შეკვეთას, მომხმარებლის საიდენტიფიკაციო ნომერი ასევე ჩაისმის იმ დამოკიდებულებაში, რომელიც შეიცავს შეკვეთის ინფორმაციას. მონაცემთა ბაზები იყენებენ საიდენტიფიკაციო ნომერს მომხმარებელსა და შეკვეთას შორის კავშირის დასადგენად.

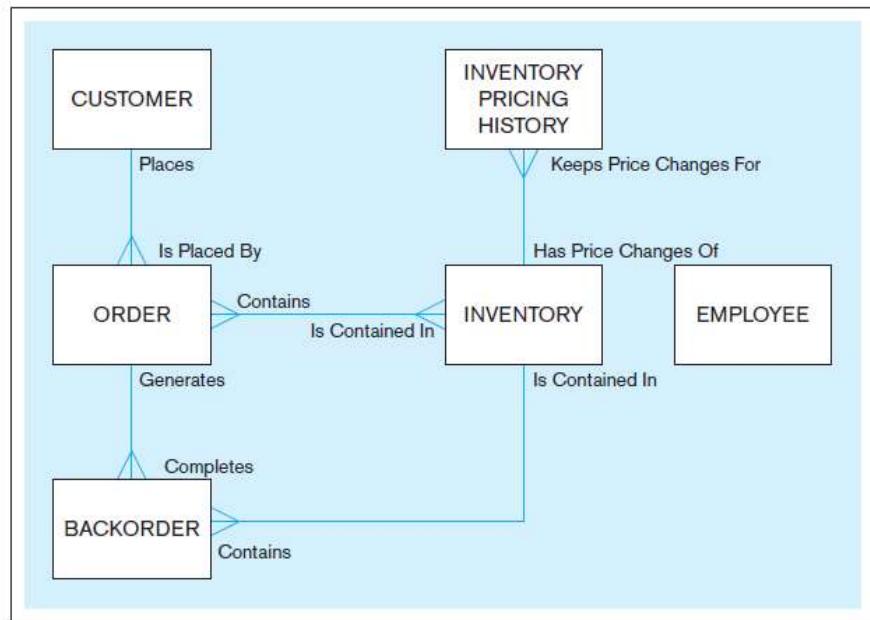
**რელაციური მონაცემთა ბაზა -** მონაცემთა ბაზა, რომელიც მონაცემებს წარმოადგენს ცხრილების ერთობლიობის სახით, რომელშიც მონაცემთა ყველა დამოკიდებულება წარმოდგენილია შემაკავშირებელი ცხრილების საერთო მნიშვნელობებით.

მონაცემთა ბაზების გამოყენების სარგებლიანობას და წარმატებულობას გარდა მონაცემთა მოდელებისა განსაზღვრავენ მონაცემთა ბაზის მართვის სისტემები (Database Management System - DBMS).

მონაცემთა ბაზის მართვის სისტემა (DBMS) არის პროგრამული უზრუნველყოფა, რომელიც მონაცემების დამუშავებისათვის მონაცემთა ბაზის მიდგომის გამოყენების შესაძლებლობას იძლევა. DBMS-ის ძირითადი დანიშნულებაა მონაცემთა ბაზაში შენახული მონაცემთა შექმნის, განახლების, შენახვისა და მოძიების სისტემური მეთოდების მიწოდება/მხარდაჭერა. ის საშუალებას აძლევს საბოლოო მომხმარებლებს და პროგრამისტებს მოახდინონ მონაცემების გაზიარება და ამ მონაცემების ერთობლივი გამოყენება სხვადასხვა დანართი პროგრამებისთვის, განსხვავებით ფაილური მიდგომისაგან, როდესაც ყოველი ახალი დანართისათვის საჭიროა ახალი ფაილის შექმნა. DBMS ასევე უზრუნველყოფს მონაცემთა წვდომის კონტროლის, მონაცემთა მთლიანობის

დაცვის, ერთდროული კონტროლის მართვისა და მონაცემთა ბაზის აღდგენის შესაძლებლობებს.

**FIGURE 1-4** Enterprise model for Figure 1-3 segments



მონაცემთა ბაზის მიდგომის მირითადი ელემენტების ცოდნა საშუალებას გვაძლევს ცხადი სახით წარმოვიდგინოთ განსხვავებები მონაცემთა ბაზის მიდგომასა და ფაილებზე დაფუძნებულ მიდგომას შორის. განვიხილოთ ნახ. 1-2 და 1-4. დიაგრამა 1-4 იძლევა მონაცემთა ბაზაში მონაცემთა შენახვის წარმოდგენას (არსების - არსების), თუ როგორ შეიძლება მათი წარმოჩენა. გავითვალისწინოთ, რომ ნახ. 1-2-ისგან განსხვავებით, ნახაზში 1-4, მხოლოდ ერთი ადგილია, სადაც ინახება კლიენტის (CUSTOMER) ინფორმაცია (განსხვავებით ნახ. 1-2-ის ორი Customer Master Files ფაილისა). შეკვეთების შევსების სისტემაც (Order Filling System) და ზედნადებების სისტემაც (Order Filling System) მიიღებენ მონაცემებს ერთი არსიდან კლიენტი (CUSTOMER). გარდა ამისა სავარაუდოდ, არცერთ სისტემასთან არ არის დაკავშირებული თუ კლიენტის (CUSTOMER) რა ინფორმაცია ინახება, როგორ ინახება და როგორ ხდება მასზე წვდომა.

**მონაცემთა ბაზის მართვის სისტემა (Database Management System - DBMS)** - პროგრამული სისტემა, რომელიც გამოიყენება მომხმარებლის მონაცემთა ბაზების შესაქმნელად, შენარჩუნებისა და კონტროლირებადი წვდომის უზრუნველსაყოფად.

ჩვენ მოვიყვანეთ მხოლოდ ერთი, მაგრამ მნიშვნელოვანი უპირატესობა მონაცემთა დამუშავებაში მონაცემთა ბაზის მიდგომის გამოყენებით. მონაცემთა ბაზის მიდგომის უპირატესობები უფრო მეტია:

- მონაცემთა პროგრამისაგან დამოუკიდებლობა

- მონაცემთა დაგეგმილი სიჭარბე
- მონაცემების გაუმჯობესებული შეჯერებულობა
- მონაცემთა გაუმჯობესებული გაზიარება
- დანართების შემუშავების პროდუქტიულობის ზრდა
- სტანდარტების დაცვა
- მონაცემთა ხარისხის გაუმჯობესება
- მონაცემებზე წვდომის და რეაგირების სისწრაფის ზრდა
- პროგრამების მხარდაჭერის რესურსების შემცირება
- გადაწყვეტილების მიღების მხარდაჭერის გაუმჯობესება

განვიხილოთ თითოეულიცალ-ცალკე.

**მონაცემთა პროგრამისაგან დამოუკიდებლობა** - მონაცემთა აღწერილობის (მეტამონაცემების) განცალკევებას პროგრამებისაგან, რომლებიც იყენებენ ამ მონაცემებს, მონაცემთა დამოუკიდებლობა ეწოდება. მონაცემთა ბაზის მიღებით, მონაცემთა აღწერილობა ინახება ცენტრალურ ადგილას, რომელსაც ეწოდება საცავი (*repository*). მონაცემთა ბაზის სისტემების ეს თვისება საშუალებას იძლევა ორგანიზაციის მონაცემები შეიცვალოს და განვითარდეს (გარკვეულ საზღვრებში) მონაცემთა დამუშავების პროგრამების ცვლილების გარეშე.

**მონაცემთა დამოუკიდებლობა** - მონაცემთა  
აღწერილობის განცალკევება პროგრამებიდან,  
რომლებიც იყენებენ ამ მონაცემებს.

**მონაცემთა დაგეგმილი სიჭარბე** - კარგი დიზაინის მონაცემთა ბაზის შემუშავებისას ცდილობები ადრე ცალკეულად არსებული (და ჭარბი) მონაცემთა ფაილების ინტეგრირებას ერთ, ლოგიკურ სტრუქტურაში. იდეალურ შემთხვევაში, მონაცემთა ბაზაში თითოეული პირველადი ფაქტი ფიქსირდება მხოლოდ ერთ ადგილზე. მაგალითად, პროდუქტის შესახებ ფაქტები, როგორიცაა Pine Valley მუხის კომპიუტერის მაგიდა, მისი ზომები, ფასი და ა.შ., აღირიცხება პროდუქტის ცხრილში ერთ ადგილას, რომელიც შეიცავს მონაცემებს Pine Valley- ს თითოეული პროდუქტის შესახებ. მონაცემთა ბაზის მიღებისა არ აღმოფხვრის სიჭარბეს მთლიანად, მაგრამ ის საშუალებას აძლევს შემმუშავებელს გააკონტროლოს ზედმეტობის ტიპი და რაოდენობა.

**მონაცემების გაუმჯობესებული შეჯერებულობა** - მონაცემთა სიჭარბის აღმოფხვრით ან კონტროლით, მნიშვნელოვნად ვამცირებთ შეუსაბამობის შესაძლებლობებს. მაგალითად, თუ მომხმარებლის მისამართი მხოლოდ ერთგანაა შენახული, ჩვენ არ შეგვიძლია არ ვენდოთ მომხმარებლის მისამართს. როდესაც მომხმარებლის მისამართი შეიცვლება, ახალი მისამართის ჩაწერა მნიშვნელოვნად გამარტივდება, რადგან მისამართი ინახება მხოლოდ ერთ ადგილზე. და ბოლოს, ჩვენ თავიდან ავიცილებთ შენახვის ადგილის ჭარბ გამოყენებას, რაც გადაჭარბებული მონაცემების შენახვის შედეგად წარმოიქმნება.

**მონაცემთა გაუმჯობესებული გაზიარება** - მონაცემთა ბაზები იქმნება, როგორც საერთო კორპორატიული რესურსი. ავტორიზებულ შიდა და გარე მომხმარებლებს ეძლევათ მონაცემთა ბაზის გამოყენების ნებართვა და თითოეულ მომხმარებელს (ან მომხმარებელთა

ჯგუფს) ამ მონაცემთა დამუშავების გასაადვილებლად ეძლევა ერთი ან მეტი მომხმარებლის „ხედი“ (**user view**) მონაცემთა ბაზაში. მომხმარებლის ხედი არის მონაცემთა ბაზის ზოგიერთი ნაწილის ლოგიკური აღწერა, რომელსაც მომხმარებელი მოითხოვს გარკვეული დავალების შესასრულებლად. მომხმარებლის ხედი ხშირად იქმნება იმ ფორმების ან ანგარიშების იდენტიფიცირებით, რომელიც მომხმარებელს რეგულარულად სჭირდება. მაგალითად, ადამიანურ რესურსებში მომუშავე თანამშრომელს დასჭირდება თანამშრომლის კონფიდენციალური მონაცემები; მომხმარებელს სჭირდება წვდომა პროდუქტის კატალოგზე, რომელიც ხელმისაწვდომია Pine Valley-ის ვებ-გვერდზე. ადამიანური რესურსების თანამშრომლისა და მომხმარებლისათვის მომხმარებლის ხედი იქნება სრულიად განსხვავებული სფეროებიდან ერთიანი მონაცემთა ბაზის სივრცეში.

**მომხმარებლის ხედი** - მონაცემთა ბაზის ზოგიერთი ნაწილის ლოგიკური აღწერა, რომელსაც მომხმარებელი მოითხოვს გარკვეული დავალების შესასრულებლად.

**დანართების შემუშავების პროდუქტიულის ზრდა** - მონაცემთა ბაზის მიდგომის მთავარი უპირატესობა ისაა, რომ იგი მნიშვნელოვნად ამცირებს ხარჯებს და დროს ახალი ბიზნეს პროგრამების (დანართტების) შესაქმნელად. არსებობს სამი მნიშვნელოვანი მიზეზი, რომელის განაპირობებს მონაცემთა ბაზის დანართების ხშირად შემუშავებას ბევრად უფრო სწრაფად, ვიდრე ჩვეულებრივი ფაილური პროგრამები:

1. ვთქვათ, რომ უკვე შემუშავებული და დანერგილია მონაცემთა ბაზა და მასთან დაკავშირებული მონაცემთა ამოდების და მომსახურების დანართი პროგრამები, დანართების შემუშავებელს შეუძლია კონცენტრირება მოახდინოს ახალ პროგრამისთვის საჭირო სპეციფიკურ ფუნქციებზე ფაილის დიზაინზე ან დაბალი დონის განხორციელების დეტალებზე აქცენტირების გარეშე;
2. მონაცემთა ბაზის მართვის სისტემა უზრუნველყოფს მაღალი დონის წარმადობის მრავალ ინსტრუმენტს, ისეთები როგორიცაა ფორმების და ანგარიშების შემქმნელები და მაღალი დონის ენები, რომლებიც ახდენენ მონაცემთა ბაზის დაგეგმვისა და განხორციელებისათვის საჭირო რიგი ქმედებების ავტომატიზაციას.
3. დანართების შემქმნელთა პროდუქტიულობის მნიშვნელოვანი გაუმჯობესება, სავარაუდოდ 60 პროცენტამდე, ამჟამად ხორციელდება ვებ-სერვისების გამოყენებით, სტანდარტული ინტერნეტ პროტოკოლებისა და საყოველთაოდ მიღებული მონაცემების ფორმატის საფუძველზე (XML).

**სტანდარტების დაცვა** - როდესაც მონაცემთა ბაზის მიდგომა ხორციელდება მენეჯმენტის სრული მხარდაჭერით, მონაცემთა ბაზის ადმინისტრირების ფუნქციას უნდა მიენიჭოს ერთიანი უფლებამოსილება და პასუხისმგებლობა მონაცემთა სტანდარტების დამკვიდრებასა და შესრულებაზე. ეს სტანდარტები მოიცავს მონაცემების დასახელების, განახლებისა და დაცვის ერთიან პროცედურებს. მონაცემთა საცავი უზრუნველყოფს მონაცემთა ბაზის ადმინისტრატორებს ამ სტანდარტების შემუშავებისა და დანერგვის მძლავრი ინსტრუმენტებით.

**მონაცემთა ხარისხის გაუმჯობესება** - მონაცემთა ბაზების დაგეგმვისა და ადმინისტრირების პროცესისათვის მონაცემთა ხარისხი ერთ-ერთი მნიშვნელოვანი საკითხია. მონაცემთა დაბალი ხარისხი რიგი მსხვილი კორპორატიული და სახელმწიფო მნიშვნელობის ბაზებისათვის, შესაძლოა ეკონომიკური ზარალის მიზეზიც კი შეიძლება გახდეს. მონაცემთა ბაზის მიდგომა ითვალისწინებს უამრავ ინსტრუმენტს და პროცესს მონაცემთა ხარისხის გასაუმჯობესებლად. ამათგან ორი ყველაზე მნიშვნელოვანა:

1. მონაცემთა ბაზის დამპროექტებლებს შეუძლიათ მიუთითონ (დააკონკრეტონ) მთლიანობის შეზღუდვები, რომლებსაც იყენებენ DBMS. შეზღუდვა არის წესი, რომლის დარღვევა არ შეუძლებლიათ მონაცემთა ბაზის მომხმარებლებს. ასე მაგალითად, თუ მომხმარებელი აკეთებს შეკვეთას, იმ შეზღუდვას, რომელიც უზრუნველყოფს მომხმარებლისა და შეკვეთის კავშირს, ეწოდება "კავშირის მთლიანობის შეზღუდვა" და ეს ხელს უშლის შეკვეთის დაფიქსირებას იმის მითითების გარეშე, თუ ვინ გააკეთა შეკვეთა.
2. მონაცემთა საწყობის გარემოს ერთ-ერთი მიზანია მუშა მონაცემების გასუფთავება (ან "გაფხეკა" - "scrub") მონაცემთა საწყობში განთავსებამდე.

**მონაცემებზე წვდომის და რეაგირების სისწრაფის ზრდა** - რელაციური მონაცემთა ბაზის საშუალებით, საბოლოო მომხმარებლებს, პროგრამირების გამოცდილების გარეშეც, ხშირად შეუძლიათ მონაცემების მოძიება და ჩვენება, მაშინაც კი, როდესაც ისინი სცდებიან თავიანთი „განყოფილების“ საზღვრებს. მაგალითად, თანამშრომელს შეუძლია ამოიღოს ინფორმაცია კომპიუტერის მაგიდების შესახებ Pine Valley Furniture Company- ში შემდეგი მოთხოვნით:

```
SELECT *
FROM Product_T
WHERE ProductDescription = "Computer Desk";
```

ამ მოთხოვნაში გამოყენებულ ენას ეწოდება **სტრუქტურირებული შეკითხვის ენა** (Structured Query Language) ან SQL. მართალია, მოთხოვნები შეიძლება ბევრად უფრო რთული სახით იყოს წარმოდგენილი, მოთხოვნის მირითადი სტრუქტურა ადვილი აღსაქმელია. თუ მომხმარებელს ესმის მონაცემთა ბაზაში შემავალი მონაცემთა სტრუქტურა და სახელები, მას შეუძლია სწრაფად მიიღოს პასუხები სხვა ახალ მოთხოვნებზე, პროფესიონალური პროგრამის შემმუშავებლის გარეშეც. მაგრამ აუცილებელია სითრთხილის გამოჩენა - მოთხოვნები საფუძვლიანად უნდა შემოწმდეს.

**პროგრამების მხარდაჭერის რესურსების შემცირება** - შენახული მონაცემები ხშირად უნდა შეიცვალოს სხვადასხვა მიზეზების გამო: ემატება მონაცემთა ერთეულის ახალი ტიპები, იცვლება მონაცემთა ფორმატები და ა.შ. ამ პრობლემის ცნობილი მაგალითა საყოველთაოდ ცნობილი "2000 წლის" პრობლემა, რომელშიც ორნიშნა წლის ველები ოთხნიშნაზე გადავიდა, რადგან უზრუნველყოფილიყო 1999 წლიდან 2000 წელზე გადასვლა.

ფაილების დამუშავების გარემოში მონაცემების აღწერილობა და მონაცემებზე წვდომის ლოგიკა ცალკეულ პროგრამულ დანართებშია ჩასმული. შედეგად, მონაცემთა ფორმატებში ცვლილებების შეტანისა და წვდომის მეთოდების შეცვლა აუცილებლად იწვევს

აპლიკაციის პროგრამების შეცვლის აუცილებლობას. მონაცემთა ბაზის გარემოში მონაცემები უფრო დამოუკიდებელია იმ გამოყენებითი პროგრამებისგან, რომლებიც მათ იყენებს. გარკვეულ ფარგლებში, ჩვენ შეგვიძლია შეცვალოთ მონაცემები ან პროგრამები, რომლებიც იყენებენ მონაცემებს, სხვა ფაქტორის შეცვლის აუცილებლობის გარეშე. შედეგად, მონაცემთა ბაზის თანამედროვე გარემოში პროგრამის შენარჩუნებისათვის საჭირო რესურსები შეიძლება მნიშვნელოვნად შემცირდეს.

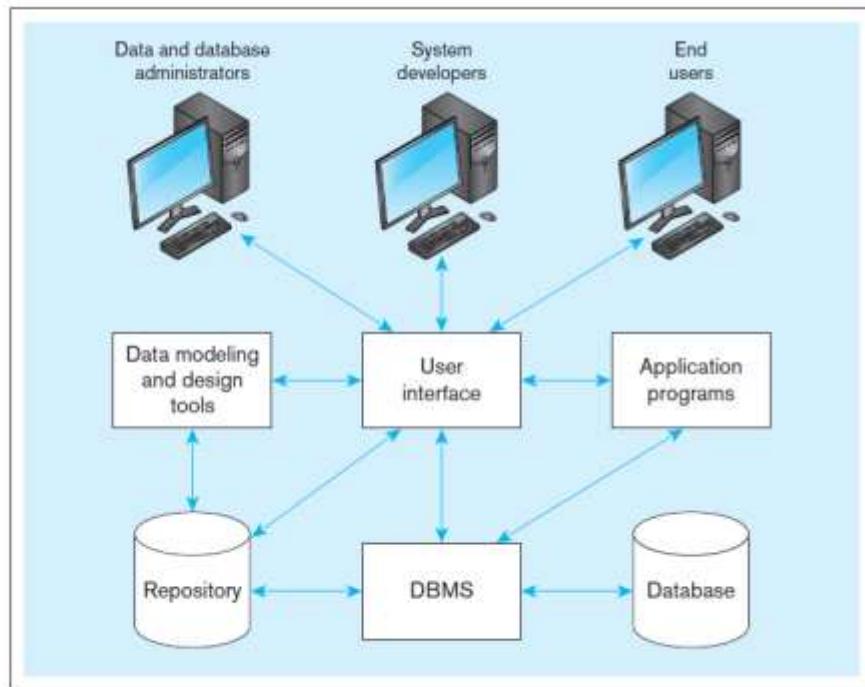
**გადაწყვეტილების მიღების მხარდაჭერის გაუმჯობესება** - ზოგიერთი მონაცემთა ბაზა შექმნილია გადაწყვეტილების მიღების მხარდაჭერი პროგრამებისთვის. მაგალითად, ზოგიერთი მონაცემთა ბაზა განკუთვნილია მომხმარებელთან კავშირის მენეჯმენტის დასახმარებლად, ზოგი კი ფინანსური ანალიზის ან მიწოდების ჯაჭვის მენეჯმენტის მხარდასაჭერად.

მიუხედავად იმისა, რომ მონაცემთა ბაზის მიდგომას მონაცემთა დამუშავება/გამოყენების თვალსაზრისით ბევრი დადებითი მახასიათებელი გააჩნია, ამ მიდგომის გამოყენება ყოველთვის არ იძლევა მოსალოდნელ შედეგს. ამის მიზეზი არაცალხასაა - მონაცემთა ძველი მოდელები და მონაცემთა ბაზის მართვის პროგრამული უზრუნველყოფის შეზღუდვები; ცუდი ორგანიზაციული დაგეგმვა და მონაცემთა ბაზის ცუდი დანერგვის პროცესი და სხვ.

### **მონაცემთა ბაზის გარემოს მთავარი კომპონენტები**

როგორც უკვე აღვნიშნეთ, მონაცემთა მონაცემთა ბაზა თანამედროვე ICTs-ს ერთერთი საუკეთესო მიდგომა მონაცემთა დამუშავება/ანალიზის თვალსაზრისით. იმისათვის, რომ დავეუფლოთ ამ მიდგომის პრაქტიკულ გამოყენებას აუცილებელია გავეცნოთ მის გარემოს - მირეულ კომპონენტებს და მათ ურთიერთკავშირს (ნახ.1-5).

**FIGURE 1-5** Components of the database environment



1. **მონაცემთა მოდელირებისა და დაგეგმვის ხელსაწყოები** - მონაცემთა მოდელირებისა და დაგეგმვის ხელსაწყოები არის ავტომატიზირებული საშუალებები, რომლებიც გამოიყენება მონაცემთა ბაზებისა და პროგრამების შესაქმნელად. ეს ხელსაწყოები ხელს უწყობს მონაცემთა მოდელების შექმნას და ზოგიერთ შემთხვევაში ასევე შეუძლია დაეხმაროს შემმუშავებელს მონაცემთა ბაზის შესაქმნელად საჭირო "კოდის" ავტომატურად გენერირებაში.
2. **საცავი (repository)** - საცავი არის ცენტრალიზებული ცოდნის ბაზა ყველა მონაცემთა, მონაცემთა კავშირის, ეკრანისა და ანგარიშის ფორმატისა და სისტემის სხვა კომპონენტების განსაზღვრისათვის. საცავი შეიცავს მეტამონაცემების გაფართოებულ ნაკრებებს, რომლებიც მნიშვნელოვანია მონაცემთა ბაზების, აგრეთვე ინფორმაციული სისტემის სხვა კომპონენტების მართვისთვის.
3. **მონაცემთა ბაზის მართვის სისტემა (DBMS)** - მონაცემთა ბაზის მართვის სისტემა (DBMS) არის პროგრამული უზრუნველყოფა, რომელიც მონაცემების დამუშავებისათვის მონაცემთა ბაზის მიღების გამოყენების შესაძლებლობას იძლევა.
4. **მონაცემთა ბაზა (Database)** - მონაცემთა ბაზა არის ლოგიკურად დაკავშირებული მონაცემების ორგანიზებული კოლექცია, რომელიც ჩვეულებრივ შექმნილია ორგანიზაციის მრავალი მომხმარებლის ინფორმაციული საჭიროებების დასაკმაყოფილებლად. მნიშვნელოვანია განვასხვაოთ მონაცემთა ბაზა და საცავი. საცავი შეიცავს მონაცემთა განმარტებებს, ხოლო მონაცემთა ბაზა შეიცავს მოვლენების მონაცემებს.
5. **გამოყენებადი პროგრამები** - კომპიუტერზე დაფუძნებული პროგრამები, რომლებიც გამოიყენება მონაცემთა ბაზის შესაქმნელად და შენარჩუნებისთვის და მომხმარებლებისთვის ინფორმაციის მიწოდების მიზნით.

6. **მომხმარებლის ინტერფეისი** - მომხმარებლის ინტერფეისი მოიცავს ენებს, მენიუებსა და სხვა საშუალებებს, რომლითაც მომხმარებლები ურთიერთქმედებენ სისტემის სხვადასხვა კომპონენტებთან, როგორიცაა მონაცემთა მოდელირებისა და დაპროექტების ხელსაწყოები, გამოყენებითო პროგრამები, DBMS და საცავი.
7. **მონაცემთა და მონაცემთა ბაზის ადმინისტრატორები** - მონაცემთა ადმინისტრატორები არიან პირები, რომლებიც პასუხისმგებელნი არიან ორგანიზაციაში მონაცემთა რესურსების საერთო მართვაზე. მონაცემთა ბაზის ადმინისტრატორები პასუხისმგებელნი არიან მონაცემთა ბაზის ფიზიკურ დაპროექტებაზე და მონაცემთა ბაზის ტექნიკური საკითხების მართვაზე.
8. **სისტემის შემსუშავებელი** - სისტემის შემსუშავებლები არიან პირები, როგორიცაა სისტემების ანალიტიკოსები და პროგრამისტები, რომლებიც ქმნიან ახალ პროგრამებს.
9. **საბოლოო მომხმარებლები** - საბოლოო მომხმარებლები არიან ორგანიზაციის მასშტაბით ის პირები, რომლებიც ახდენენ მონაცემთა ბაზაში მონაცემების დამატებას, წაშლას და შეცვლას, ასევე ითხოვენ ან იღებენ ინფორმაციას მისგან. მომხმარებლის ყველა ურთიერთქმედება მონაცემთა ბაზასთან უნდა მოხდეს DBMS- ის საშუალებით.

მონაცემთა ბაზის ქმედების გარემო არის აპარატურის, პროგრამული უზრუნველყოფისა და ადამიანების ინტეგრირებული სისტემა, რომელიც შექმნილია ინფორმაციის რესურსის შენახვის, მოძიებისა და კონტროლისა და ორგანიზაციის პროდუქტიულობის გასაუმჯობესებლად.

## მონაცემთა ბაზის შემსუშავების პროცესი

უმეტესწილად მონაცემთა ბაზის შემსუშავება იწყება საწარმოს (ან საქმიანი გარემოს) მონაცემთა მოდელირებით, რომელიც ადგენს ორგანიზაციული (ან ბაზის აქტივობის გარემოს) მონაცემთა ბაზის დიაპაზონს და ზოგად შინაარსს. მისი მიზანია შექმნას ორგანიზაციული მონაცემების საერთო სურათი ან განმარტება და არა კონკრეტული მონაცემთა ბაზის დიზაინი. კონკრეტულ მონაცემთა ბაზაში მოცემულია მონაცემები ერთი ან მეტი საინფორმაციო სისტემისთვის, ხოლო საწარმოს (ბიზნესის) მონაცემთა მოდელი, რომელიც შეიძლება მოიცავდეს მრავალ მონაცემთა ბაზას, აღწერს მონაცემებს ორგანიზაციის (საქმიანი გარემოს) ფარგლებში. საწარმოს მონაცემთა მოდელირებისას განიხილავენ მიმდინარე სისტემებს, აანალიზებენ ბიზნესის მხარდაჭერის სფეროებს, აღწერს აბსტრაქციის ძალიან მაღალ დონეზე საჭირო მონაცემებს და გეგმავენ მონაცემთა ბაზის განვითარების ერთ ან მეტ პროექტს.

ნახ. 1-3a ნაჩვენებია საწარმოს მონაცემთა მოდელის ფრაგმენტი Pine Valley Furniture Company- სთვის, აღნიშვნების გამარტივებული ვერსიის გამოყენებით. არსის ტიპების ამგვარი გრაფიკული გამოსახულების გარდა, საწარმოს მონაცემთა სრულყოფილი მოდელი ასევე მოიცავს ბიზნესზე ორიენტირებულ აღწერილობას თითოეული ტიპის არსისათვის და ასევე სხვადასხვა დებულების კრებულს ბიზნესის წარმოების შესახებ, ბიზნესის საქმიანობის წესებს, რომლებიც მართავს მონაცემებს. ურთიერთკავშირი ბიზნეს არსებს (საქმიანი ფუნქციები, განყოფილებები, პროგრამები და ა.შ.) და მონაცემებს შორის ხშირად ფიქსირდება მატრიცების გამოყენებით, რომელსაც ავსებენ საწარმოს მონაცემთა მოდელიდან ამოღებული ინფორმაციით. ნახ. 1-6 გვიჩვენებს ასეთი მატრიცის მაგალითს.

Data Entity Types		Customer	Product	Raw Material	Order	Work Center	Work Order	Invoice	Equipment	Employee
Business Functions	Customer	X	X						X	X
	Product		X	X	X				X	
	Raw Material			X	X	X	X		X	
	Order			X	X	X	X	X	X	X
	Work Center				X	X				
	Work Order					X	X	X	X	X
	Invoice						X	X	X	X
	Equipment							X	X	X
	Employee								X	X

X = data entity is used within business function

**FIGURE 1-6** Example business function-to-data entity matrix

საწარმოთა მონაცემთა მოდელირება, როგორც ინფორმაციული სისტემების დაგეგმვისა და განვითარების „ზემოდან-ქვებოთ“ მიდგომის კომპონენტი, წარმოადგენს მონაცემთა ბაზის დაპროექტების ერთ-ერთ წყაროს. ასეთ პროექტებში ხშირად ქმნიან ახალ მონაცემთა ბაზას ორგანიზაციის სტრატეგიული მიზნების მისაღწევად, მაგალითად როგორიცაა მომხმარებელთა დახმარების გაუმჯობესება, წარმოებისა და მარაგების უკეთესი მენეჯმენტი, ან გაყიდვების უფრო ზუსტი პროგნოზირება. თუმცა მონაცემთა ბაზის ბევრი პროექტი წარმოიქმნება უფრო „ქვემოდან-ზემოთ“. ამ შემთხვევაში, პროექტების მოთხოვნა მოდის ინფორმაციული სისტემების მომხმარებლებისაგან, რომელთაც სჭირდებათ გარკვეული ინფორმაცია სამუშაოს შესასრულებლად, ან სხვა ინფორმაციული სისტემების პროფესიონალებისაგან, რომლებიც ორგანიზაციაში მონაცემთა მართვის გაუმჯობესების აუცილებლობას ხედავენ.

მონაცემთა ბაზის შემუშავების „ქვემოდა-ზევით“ ტიპი, როგორც წესი, ფოკუსირებულია მხოლოდ ერთი მონაცემთა ბაზის შექმნაზე. მონაცემთა ბაზის ზოგიერთი პროექტი კონცენტრირებულია მხოლოდ მონაცემთა ბაზის განსაზღვრაზე, პროექტზე და განხორციელებაზე, როგორც ინფორმაციული სისტემების შემდგომი განვითარების საფუძველი. თუმცა უმეტეს შემთხვევაში, მონაცემთა ბაზა და მასთან დაკავშირებული ინფორმაციის დამუშავების ფუნქციების შემუშავება ხდება ერთიანადა, როგორც საინფორმაციო სისტემების პროექტის ნაწილი.

### სისტემის შემუშავების სასიცოცხლო ციკლი

ინფორმაციული სისტემების პროექტის შექმნის ტრადიციულ პროცეს სისტემების შემუშავების სასიცოცხლო ციკლი (Systems Development Life Cycle - SDLC) ეწოდება. SDLC არის ბიჯების სრული ნაკრები, რომელთა გავლაც უზრუნველყოფს საინფორმაციო სისტემის განსაზღვრას, შემუსავებას, ქმედითუნარიოანობის მხარდაჭერას და

ცვლილებებს. როგორც წესი გამოიყენება სასიცოცხლო ციკლის მრავალი ვარიაცია და ის შეიძლება განსაზღვროს 3-დან 20 სხვადასხვა ფაზამდე.

SDLC-ის სხვადასხვა ბიჯები და მათი ასოცირებული მიზნები ასახულია ნახაზზე 1-7. როგორც ვხედავთ პროცესი ციკლურია და მიზნად ისახავს სისტემების შემუშავების პროექტების განმეორებითი ხასიათის გადმოცემას. ბიჯები შეიძლება დროში გადაიფაროს, შეიძლება განხორციელდეს პარალელურად და შესაძლებელია წინა ბიჯებზე უკან დაბრუნება, როდესაც საჭიროა წინასწარი გადაწყვეტილებების გადახედვა.

ნახ. 1-7 ასევე გვთავაზობს მონაცემთა ბაზის შემუშავების აქტივობას, რომელიც ჩვეულებრივ შედის SDLC-ის თითოეულ ფაზაში. გასათვალისწინებელია ის ფაქტი, რომ SDLC ფაზებსა და მონაცემთა ბაზის შემუშავების საფეხურებს შორის ყოველთვის არ არის ცალსახა ურთიერთკავშირი. მაგალითად, მონაცემთა კონცეპტუალური მოდელირება ხდება როგორც დაგეგმვის, ისე ანალიზის ფაზებში.

**საწარმოს (საქმიანი გარემოს) გეგმის მოდელირება** - მონაცემთა ბაზის შემუშავების პროცესი იწყება საწარმოს (საქმიანი გარემოს) მოდელირების კომპონენტების მიმოხილვით, რომლებიც შემუშავდა ინფორმაციული სისტემების დაგეგმვის პროცესში. ამ ეტაპის განმავლობაში, ანალიტიკოსები განიხილავენ მიმდინარე მონაცემთა ბაზებსა და ინფორმაციულ სისტემებს; ანალიზებენ ბიზნესის სფეროს ხასიათს, რომელიც პროექტის შემუშავების საგანია; ზოგადად აღწერენ მონაცემებს, რომლებიც საჭიროა თითოეული განსახილველი საინფორმაციო სისტემისთვის. განსაზღვრავენ რა მონაცემებია უკვე ხელმისაწვდომი მონაცემთა ბაზაში და რა ახალი მონაცემების დამატებაა შემოთავაზებული ახალი პროექტის მხარდასაჭერად. თითოეული პროექტის ორგანიზაციისთვის საპროგნოზო მნიშვნელობის გათვალისწინების შემდეგ შერჩეული პროექტები გადადიან მომდევნო ფაზაში.

**დაგეგმვა** - მონაცემთა კონცეპტუალური მოდელი - ინიცირებული საინფორმაციო სისტემების პროექტისთვის აუცილებელია შემოთავაზებული ინფორმაციული სისტემის საერთო მოთხოვნების გაანალიზება. ეს კეთდება ორ ეტაპად. პირველ ეტაპზე, დაგეგმვის ფაზის განმავლობაში, ანალიტიკოსი შეიმუშავებს დიაგრამას, რომელიც ნახ. 1-3a- ს მსგავსია, ასევე სხვა დოკუმენტაციას, რათა განისაზღვროს ამ კონკრეტულ პროექტში ჩართული მონაცემების მოცულობა იმის გათვალისწინებით, თუ რა მონაცემთა ბაზა არსებობს უკვე. ამ ეტაპზე ჩართულია მონაცემთა მაღალი დონის კატეგორიები (არსები - არსები) და ძირითადი ურთიერთკავშირები. SDLC-ის ამ ბიჯს გადამწყვეტი მნიშვნელობა აქვს შემუშავების წარმატებული პროცესის შანსების გასაუმჯობესებლად. რაც უფრო უკეთა განსაზღვრავრული ორგანიზაციის სპეციფიკური საჭიროებები, მით უფრო ახლოს უნდა იდგეს კონცეპტუალური მოდელი ინგენიერის მოთხოვნილებების დასაკმაყოფილებლად და მით უფრო ნაკლებად იქნება SDLC-ის განმეორებით გამოყენების საჭიროება.

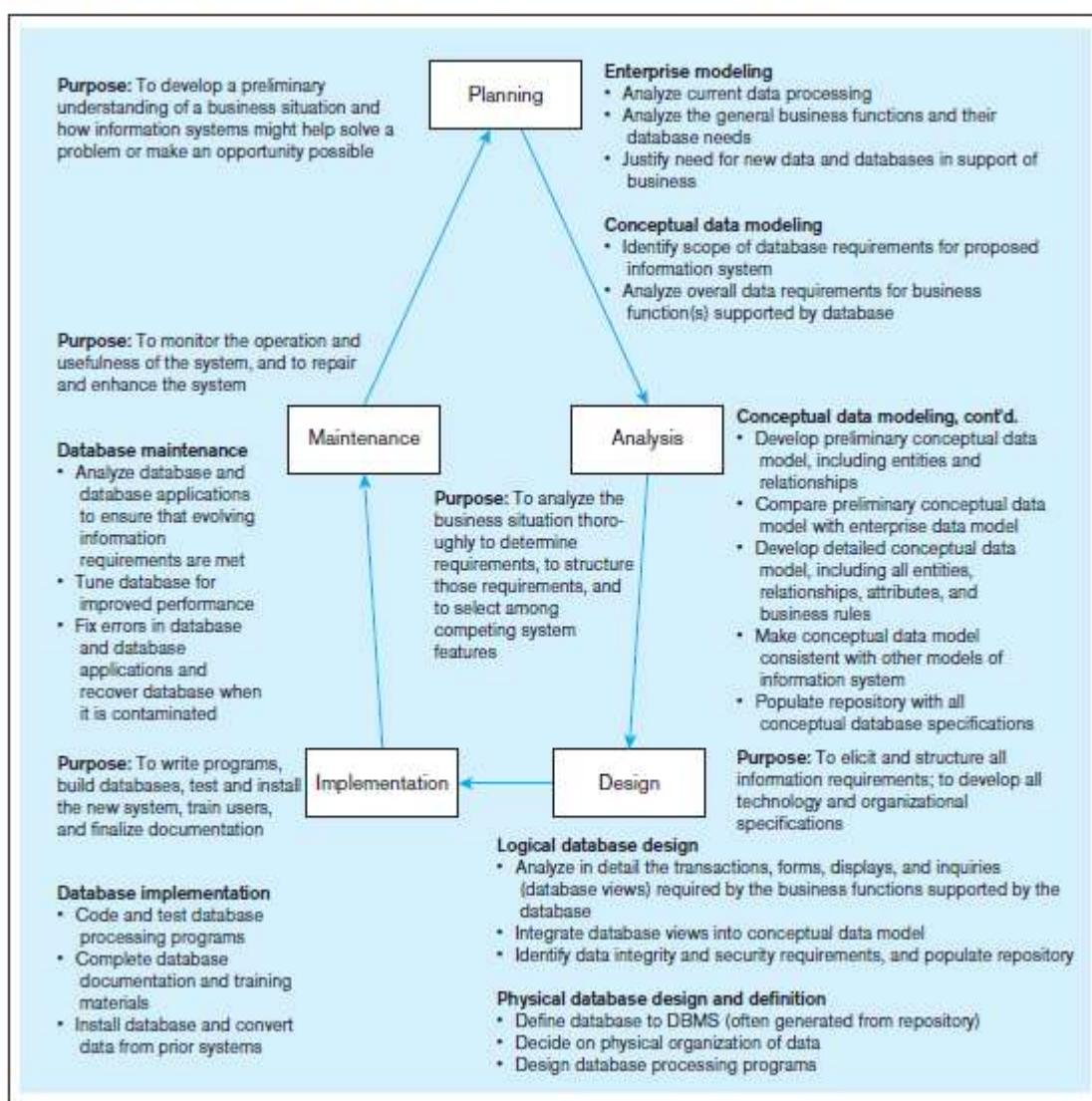
**ანალიზი** - მონაცემთა კონცეპტუალური მოდელი - SDLC- ის ანალიზის ფაზის განმავლობაში, ანალიტიკოსი ქმნის მონაცემთა დეტალურ მოდელს, რომელიც განსაზღვრავს ყველა იმ ორგანიზაციულ მონაცემებს, რომელთა მართვაც უნდა მოხდეს ამ ინფორმაციული სისტემისთვის. განსაზღვრულია მონაცემთა ყველა ატრიბუტი, ჩამოთვლილია მონაცემთა ყველა კატეგორია, წარმოდგენილია ყველა ბიზნეს-კავშირი მონაცემთა არსებს შორის, მითითებულია ყველა წესი, რომელიც უზრუნველყოფს

მონაცემთა მთლიანობას. ანალიზის ფაზაში ასევე მოწმდება მონაცემთა კონცეპტუალური მოდელის შესაბამისობა სხვა ტიპის მოდელებთან, რომლებიც მიზნად ისახავს სამიზნე ინფორმაციული სისტემის სხვა განზომილებებს, როგორიცაა დამუშავების ეტაპები, მონაცემთა დამუშავების წესები და ქმედებების ვადები. თუმცა ეს დეტალური მონაცემების კონცეპტუალური მოდელიც არის წინასწარი, რადგან SDLC-ს შემდგომმა აქტივობებმა შეიძლება აღმოაჩინოს დაკარგული ელემენტები ან შეცდომები კონკრეტული ტრანზაქციების, ანგარიშების, წარმოჩენის და მოთხოვნების შემუშავებისას.

**დაპროექტება - მონაცემთა ბაზის ლოგიკური დაპროექტება** - მონაცემთა ბაზის ლოგიკური დაპროექტება ორი თვალსაზრისით უდგება მონაცემთა ბაზის შექმნას. პირველი, კონცეპტუალური სქემა უნდა გარდაიქმნას ლოგიკურ სქემაში, რომელიც აღწერს მონაცემებს მონაცემთა მართვის ტექნოლოგიის თვალსაზრისით, რომელიც გამოყენებული იქნება მონაცემთა ბაზის განსახორციელებლად. მაგალითად, თუ გამოყენებული იქნება რელაციური ტექნოლოგია, მონაცემთა კონცეპტუალური მოდელი გარდაიქმნება და წარმოდგება რელაციური მოდელის ელემენტების გამოყენებით, რომლებიც მოიცავს ცხრილებს, სვეტებს, მწკრივებს, პირველად გასაღებებს, გარე გასაღებებსა და შეზღუდვებს. ამ წარმოდგენას ლოგიკურ სქემად მოიხსენიებენ.

ამის შემდგომ, საინფორმაციო სისტემაში თითოეული დანართის შეიქმნისას, მათ შორის შეყვანისა და გამოტანის ფორმატის პროგრამებში, ანალიტიკოსი ახორციელებს მონაცემთა ბაზის მიერ მხარდაჭერილი ოპერაციების, ანგარიშების, წარმოჩენების და გამოკვლევების დეტალურ მიმოხილვას. ამ ე.წ. „ქვემოდან-ზემოთ“ განხორციელებული ანალიზის დროს, ანალიტიკოსი ზუსტად ამოწმებს თუ რა მონაცემები უნდა იყოს დაცული მონაცემთა ბაზაში და ა იმ მონაცემების ხასიათს, რაც საჭიროა თითოეული ტრანზაქციისთვის, ანგარიშისა და ა.შ. შეიძლება საჭირო გახდეს მონაცემთა კონცეპტუალური მოდელის დახვეწა, რადგან ამ დროს ხდება თითოეული ანგარიშის, ბიზნეს ოპერაციის და მომხმარებლის სხვა ხედვის ანალიზი. ამ შემთხვევაში აუცილებელია საწყისი მონაცემების კონცეპტუალური მოდელის და ცალკეული მომხმარებლის წარმოჩენების დაკავშირება ან ინტეგრირება მონაცემთა ბაზის ლოგიკურ პროექტში. ასევე შესაძლებელია ინფორმაციის დამუშავების დამატებითი მოთხოვნების იდენტიფიცირება საინფორმაციო სისტემების ლოგიკური დაპროექტების დროს, ამ შემთხვევაში ეს ახალი მოთხოვნები უნდა იყოს ინტეგრირებული ადრე განსაზღვრულ მონაცემთა ბაზის ლოგიკურ პროექტში.

**FIGURE 1-7** Database development activities during the systems development life cycle (SDLC)



მონაცემთა ბაზის ლოგიკური დაპროექტების საბოლოო ეტაპია მონაცემთა კომბინირებული და შეჯერებული სპეციფიკაციების გარდაქმნა ძირითად, ან ატომარულ ელემენტებად კარგად სტრუქტურირებული მონაცემების სპეციფიკაციების კარგად დადგენილი წესების შესაბამისად. დღეს მონაცემთა ბაზების უმეტესობისთვის ეს წესები გამომდინარეობს რელაციური მონაცემთა ბაზის თეორიიდან და პროცესიდან, რომელსაც ეწოდება ნორმალიზაცია. შედეგად მიიღება მონაცემთა ბაზის სრული სურათი, ამ მონაცემთა მართვის კონკრეტული მონაცემთა სისტემის მითითების გარეშე. მონაცემთა ბაზის საბოლოო ლოგიკური პროექტის შემუშავებისთანავე, ანალიტიკოსი იწყებს კონკრეტული კომპიუტერული პროგრამების და მოთხოვნების ლოგიკის განსაზღვრას მონაცემთა ბაზის შინაარსის და შესაბამისობის შესანარჩუნებლად.

**დაპროექტება - მონაცემთა ბაზის ფიზიკური დაპროექტება და განსაზღვრა - ფიზიკური სქემა არის სპეციფიკაციების ერთობლიობა, რომელშიც აღწერილია, თუ როგორ ინახება**

ლოგიკური სქემის მონაცემები კომპიუტერის მეორად მეხსიერებაში მონაცემთა ბაზის მართვის სპეციფიკური სისტემის მიერ. თითოეული ლოგიკური სქემისთვის არსებობს ერთი ფიზიკური სქემა. მონაცემთა ბაზის ფიზიკური განხორციელება მოითხოვს კონკრეტული DBMS-ის ცოდნას, რომელიც გამოყენებული იქნება მონაცემთა ბაზის განსახორციელებლად. მონაცემთა ბაზის ფიზიკური დაპროექტებისა და განსაზღვრისას, ანალიტიკის ილებს გადაწყვეტილებას ფიზიკური ჩანაწერების ორგანიზების, ფაილური ორგანიზების არჩევანის, ინდექსების გამოყენების და ა.შ. შესახებ. ამისათვის მონაცემთა ბაზის შემმუშავებელმა უნდა მონიშნოს პროგრამები ტრანზაქციების დამუშავების და მოსალოდნელი სამართავი ინფორმაციის და გადაწყვეტილებების მხარდამჭერი ანგარიშების შესაქმნელად. ამის მიზანია ისეთი მონაცემთა ბაზის შექმნა, რომელიც ეფექტურად და საიმედოდ გაუმკლავდება მის წინშე არსებულ მონაცემთა ყველა დამუშავების მოთხოვნებს. ამრიგად, მონაცემთა ბაზის ფიზიკური შექმნა ხორციელდება ფიზიკური ინფორმაციის სისტემის ყველა სხვა ასპექტის: პროგრამების, კომპიუტერული ტექნიკის, ოპერაციული სისტემებისა და მონაცემთა საკომუნიკაციო ქსელების მჭიდრო კოორდინაციით.

**დანერგვა - მონაცემთა ბაზის დანერგვა** - მონაცემთა ბაზის დანერგვისას, დამპროექტებელი წერს, ცდის და აყენებს პროგრამებს/სკრიპტებს, რომლებიც უზრუნველყოფენ მონაცემთა ბაზაში წვდომას, შექმნას ან შეცვლის. დამპროექტებელმა შეიძლება ეს გააკეთოს სტანდარტული დაპროგრამების ენების გამოყენებით (მაგ. Java, C # ან Visual Basic.NET) ან მონაცემთა ბაზის დამუშავების სპეციალურ ენებით (მაგ., SQL) ან გამოიყენოს სპეციალური დანიშნულების არაპროცესორული ენები სტილიზებული ანგარიშებისა და წარმოჩენების შესაქმნელად, მათ შორის გრაფიკების ჩათვლით. განხორციელების პროცესში, დამპროექტებელი ასრულებს მონაცემთა ბაზის ყველა დოკუმენტაციას, ამზადებს მომხმარებლებს და ადგენს პროცედურებს ინფორმაციული სისტემის (და მონაცემთა ბაზის) მომხმარებლების მუდმივი მხარდაჭერისთვის. ბოლო ბიჯი არის არსებული ინფორმაციის წყაროებიდან მონაცემების ჩატვირთვა (ფაილები და მონაცემთა ბაზები „სამერკვიდრე“ პროგრამებიდან, ასევე ახალი მონაცემები) ჩატვირთვა ხშირად ხდება არსებული ფაილებისა და მონაცემთა ბაზებიდან ნეიტრალურ ფორმატში (მაგალითად, ორობითი ან ტექსტური ფაილების) მონაცემების გადმოტვირთვის და შემდეგ ამ მონაცემების ახალ მონაცემთა ბაზაში ჩატვირთვით. დაბოლოს, მონაცემთა ბაზა და მასთან დაკავშირებული პროგრამები მზადდება მონაცემთა მომხმარებლების მიერ მონაცემთა შენარჩუნებისა და აღსადგენად. წარმოების დროს, მონაცემთა ბაზაში პერიოდულად უნდა შეიქმნას სარეზერვო ასლი და მოხდეს აღდგენა „დაბინძურების“ ან განადგურების შემთხვევაში.

**მომსახურება - მონაცემთა ბაზის მომსახურება** - მონაცემთა ბაზა ვითარდება მონაცემთა ბაზის მომსახურების დროს. ამ ეტაპზე, დამპროექტებელი ამატებს, შლის ან ცვლის მონაცემთა ბაზის სტრუქტურის მახასიათებლებს ბიზნესის შეცვლის პირობების დაკმაყოფილების, მონაცემთა ბაზის პროექტის შეცდომების გამოსწორების ან მონაცემთა ბაზის პროგრამების დამუშავების სიჩქარის გასაუმჯობესებლად. დამპროექტებელს შესაძლოა ასევე დასჭირდეს მონაცემთა ბაზის აღდგენა, თუ იგი „დაბინძურდება“ ან განადგურდება პროგრამის ან კომპიუტერული სისტემის გაუმართაობის გამო. ეს, როგორც წესი, მონაცემთა ბაზის განვითარების ყველაზე გრძელი ბიჯია, რადგან ის გრძელდება მონაცემთა ბაზისა და მასთან დაკავშირებული პროგრამების არსებობის განმავლობაში.

ყოველ ჯერზე, როდესაც მონაცემთა ბაზა ვითარდება, ხდება მისი გადახედვა, როგორც მონაცემთა ბაზის შემუშავების შემცირებული პროცესის, რომელშიც ხდება კონცეპტუალური მონაცემების მოდელირება, მონაცემთა ბაზის ლოგიკური და ფიზიკური პროექტირება და მონაცემთა ბაზის დანერგვა, შემოთავაზებული ცვლილებების განსახორციელებლად.

### მონაცემთა ბაზის შემუშავების სამ სქემიანი არქიტექტურა

ჩვენ განვმარტეთ მონაცემთა ბაზის შემუშავების პროცესთან დაკავშირებული რამდენიმე განსხვავებული, მაგრამ მასთან დაკავშირებული მონაცემთა ბაზის შემუშავების მოდელები. მონაცემების ეს მოდელები და SDLC-ის ძირითადი ფაზები, რომლებშიც ხდება მათი შემუშავება, შეიძლება მოკლედ აღვწეროთ როგორც:

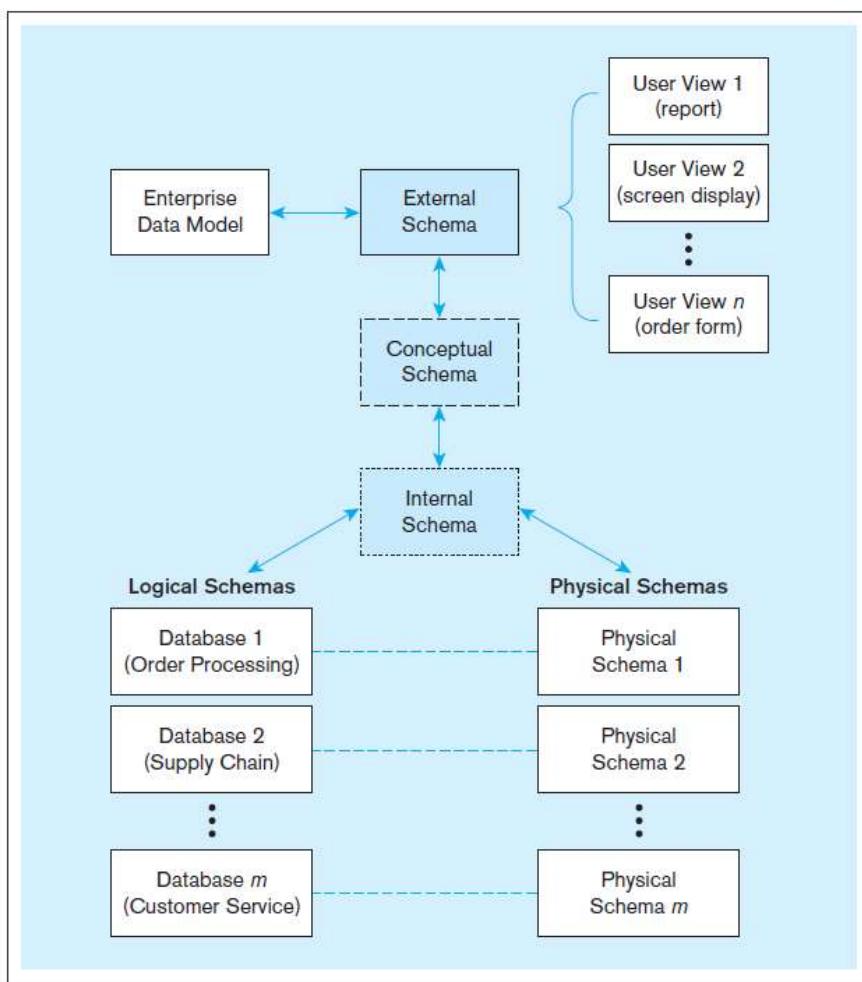
- საწარმოს მონაცემთა მოდელი (ინფორმაციული სისტემების დაგეგმვის ეტაპზე);
- გარე სქემა ან მომხმარებლის ხედი (ანალიზისა და ლოგიკური დაპროექტების ფაზების განმავლობაში);
- კონცეპტუალური სქემა (ანალიზის ეტაპზე);
- ლოგიკური სქემა (ლოგიკური დაპროექტების ეტაპზე);
- ფიზიკური სქემა (ფიზიკური დაპროექტების ეტაპზე).

1978 წელიდან მოქმედებაშია უკვე საერთაშორისოდ აღიარებული ANSI / SPARC სტანდარტი, რომელშიც აღწერილი იყო სამსქემინი (ხშირად მოიხსენიება, როგორც სამშრიანი ან სამდონიანი) არქიტექტურა, რომელიც მონაცემთა სტრუქტურის აღსაწერად მოიცავს გარე, კონცეპტუალური და შიდა სქემების (შრეებს). ნახ. 1-9 გვიჩვენებს ურთიერთკავშირს სამსქემანი არქიტექტურის დროს შემუშავებულ სხვადასხვა სქემებს შორის. მნიშვნელოვანია გავითვალისწინოთ, რომ ყველა ეს სქემა წარმოადგენს სხვადასხვა დაინტერესებული მხარეების მიერ ერთი და იგივე მონაცემთა ბაზის სტრუქტურის ვიზუალიზაციის განსხვავებული გზებით.

ამ სტანდარტით განსაზღვრული სამდონიანი სქემაის თოთოეული შემადგენელი სქემა შეიძლება ასე განისაზღვროს:

1. **გარე სქემა** - ეს არის მონაცემთა ბაზის მომხმარებელი მენეჯერებისა და სხვა თანამშრომლების წარმოჩენები (ხედები), როგორც ნაჩვენებია ნახ. 1-9-ში, გარე სქემა შეიძლება წარმოდგენილი იყოს როგორც საწარმოს მონაცემთა მოდელის კომბინაცია (ზემოდან-ქვემოთ ხედი) და დეტალური (ან ქვემოდან-ზემოთ) მომხმარებლის ხედების კრებული.
2. **კონცეპტუალური სქემა** - ეს სქემა აერთიანებს სხვადასხვა გარე წარმოჩენას საწარმოს მონაცემების ერთ, თანმიმდევრულ და ყოვლისმომცველ განსაზღვრებად. კონცეპტუალური სქემა წარმოადგენს მონაცემთა არქიტექტორის ან მონაცემთა ადმინისტრატორის ხედვას.
3. **შიდა სქემა** - შიდა სქემა შედგება ორი ცალკეული სქემისგან: ლოგიკური სქემა და ფიზიკური სქემა. ლოგიკური სქემა წარმოადგენს მონაცემთა წარმოების ტექნოლოგიის ტიპის მონაცემების წარმოდგენას (მაგალითად, რელაციური). ფიზიკური სქემა აღწერს, თუ როგორ ხდება მონაცემების წარმოდგენა და შენახვა მეორად საცავში, კონკრეტული DBMS-ის (მაგ., Oracle) გამოყენებით.

**FIGURE 1-9** Three-schema architecture



მონაცემთა ბაზა შემუშავების პროცესი მრავალკომპონენტიანი პროექტის გახორციელებას გულისხმობს. პროექტი მიზნის მისაღწევად დაგეგმილი ურთიერდაკავშირებული ქმედებებია, რომელსაც აქვს დასაწყისი და დასასრული. პროექტი იწყება პროექტის ინიცირებისა და დაგეგმვის ფაზის პირველი ნაბიჯებით და მთავრდება განხორციელების ფაზის ბოლო ნაბიჯებით.

პროექტი იწყება და იგეგმება დაგეგმვის ეტაპზე; ხორციელდება ანალიზის, ლოგიკური დიზაინის, ფიზიკური დაპროექტიების და განხორციელების ფაზების დროს; და ბოლოს იხურება ყველა ფაზის დასრულებისას. ინიცირების დროს იქმნება პროექტის განმხორციელებელი გუნდი, რომლის შემადგენლობაში შედის სხვადასხვა ფუნქციის მქონე წევრები. თითო პოზიციაზე შეიძლება იმყოფებოდეს რამდენიმე პირი. ეს პოზიციებია:

**ბიზნესის ანალიტიკურები** - პირები რომლებიც თანამშრომლობენ როგორც მმართველ რგოლთან, ასევე მომხმარებლებთან ბიზნეს სიტუაციის გასაანალიზებლად და

პროექტისათვის დეტალური სისტემისა და პროგრამის სპეციფიკაციების შემუშავებისთვის.

სისტემების ანალიტიკოსები - ამ პირებს შეუძლიათ შეასრულონ ბიზნესის ანალიტიკოსის საქმიანობაც, მაგრამ ასევე განსაზღვრონ კომპიუტერული სისტემების მოთხოვნები და, როგორც წესი, აქვთ უფრო მეტი გამოცდილება სისტემების შემუშავებაში, ვიდრე ბიზნეს ანალიტიკოსებს.

მონაცემთა ბაზის ანალიტიკოსები და მონაცემთა მოდელის შემდგენები - ეს პირები კონცენტრირებულნი არიან ინფორმაციული სისტემის მონაცემთა ბაზის კომპონენტის მოთხოვნების და პროექტირების განსაზღვრაზე.

მომხმარებლები - მომხმარებლები ახდენენ თავიანთი ინფორმაციული საჭიროებების შეფასებას და აკონტროლებენ, რომ შემუშავებული სისტემა აკმაყოფილებს მათ მოთხოვნებს.

პროგრამისტები - პირები რომლებიც ქმნიან და წერენ კომპიუტერულ პროგრამებს, რომლებიც მოიცავენ მონაცემთა ბაზაში ჩასმულ ბრძანებებს მონაცემთა შენარჩუნებისა და წვდომისათვის.

მონაცემთა ბაზის არქიტექტორები - პირები რომლებიც ადგენენ მონაცემთა სტანდარტებს ბიზნესის ერთეულებში.

მონაცემთა ადმინისტრატორები - პირები რომლებთაც ეკისრებათ პასუხისმგებლობა არსებულ და მომავლ მონაცემთა ბაზაზე და რომლებმაც უნდა უზრუნველყონ მონაცემთა ბაზაში თანმიმდევრულობა და მთლიანობა, და როგორც მონაცემთა ბაზის ტექნოლოგიის ექსპერტები, ისინი კონსულტაციასა და ტრენინგებს უტარებენ პროექტის გუნდის სხვა წევრებს.

პროექტის მენეჯერები - პროექტის მენეჯერები აკონტროლებენ პროექტებს, მათ შორის გუნდური შემადგენლობას პოზიციონირებას, ანალიზს, დიზაინს, განხორციელებას და პროექტების მხარდაჭერას.

სხვა ტექნიკური პერსონალი - სხვა საჭირო პირები ისეთ სფეროებში, როგორიცაა ქსელი, ოპერაციული სისტემები, ტესტირება, მონაცემთა შენახვა და დოკუმენტაცია.

## ლექცია - 2

### მონაცემთა მოდელირება მონაცემთა ბაზის ფუნქციონირების გარემოსათვის (ორგანიზაციისათვის)

მონაცემთა ბაზის განვითარების პირველი ნაბიჯი არის მონაცემთა ბაზის ანალიზი, რომელშიც ჩვენ განვსაზღვრავთ მომხმარებლის მოთხოვნებს მონაცემთა მიმართ და ვამუშავებთ მონაცემთა მოდელებს, ამ მოთხოვნების გამოსახატავად. ეს პროცესი იწყება მონაცემთა მოდელის შექმნით.

მონაცემთა მოდელების საფუძველი ბიზნესის (საქმიანი გარემოს) წესებია, რომელიც გამომდინარეობს ამ საქმიანობის პოლიტიკის, პროცედურების, ღონისძიებების, ფუნქციების და სხვა ბიზნეს მიმდინარეობისაგან და ასახავს ორგანიზაციის (ან/და საქმიანობის) შეზღუდვებს. ბიზნესის წესები წარმოადგენს ორგანიზაციის ენას და ფუნდამენტურ სტრუქტურას. ბიზნესის წესები ახდენენ ორგანიზაციის მფლობელების, მენეჯერებისა და ლიდერების მიერ ორგანიზაციის აღმის ფორმულირებას ინფორმაციული სისტემების არქიტექტორების გასაგები ფორმით.

ბიზნესის წესები მნიშვნელოვანია მონაცემთა მოდელირებისთვის, რადგან ისინი განსაზღვრავენ თუ რომელი მონაცემები უნდა დამუშავდეს და შეინახოს. ძირითადი ბიზნეს წესების მაგალითებია მონაცემთა სახელები და მათი განმარტებები. მონაცემების კონცეპტუალური მოდელირების თვალსაზრისით, სახელები და განმარტებები უნდა მიეთითოს მონაცემთა ძირითადი ობიექტებისთვის: არსის (სუბიექტის) ტიპები (მაგ., მომხმარებელი - Customer), ატრიბუტები (მომხმარებლის სახელი Customer Name), და დამოკიდებულებები (კავშირები) (მომხმარებელი განათავსებს შეკვეთას Customer Places Order). სხვა ბიზნეს წესებში შეიძლება დაწესდეს შეზღუდვები ამ მონაცემთა ობიექტებზე. ამ შეზღუდვების დაფიქსირება შესაძლებელია მონაცემთა მოდელში, მაგალითად, „არსი-კავშირი“ სქემა და მასთან დაკავშირებული დოკუმენტაცია. დამატებითი ბიზნეს წესები არეგულირებს ორგანიზაციის ხალხს, ადგილებს, მოვლენებს, პროცესებს, ქსელებსა და მიზნებს, რომლებიც დაკავშირებულია მონაცემთა მოთხოვნებთან სხვა სისტემის დოკუმენტაციის საშუალებით.

ათწლეულების გამოყენების მიუხედავად E-R მოდელი კონცეპტუალური მონაცემების მოდელირების მთავარ მიდგომად რჩება. მისი პოპულარობა გამომდინარეობს ისეთი ფაქტორებიდან, როგორიცაა გამოყენების შედარებითი სიმარტივე, კომპიუტერული ინჟინერიის ინსტრუმენტი პროგრამების (CASE) ფართო მხარდაჭერა და რწმენა იმისა, რომ „არსები“ და „კავშირები“ რეალურ სამყაროში ბუნებრივი მოდელირების ცნებებია.

E-R მოდელი უფრო ხშირად გამოიყენება, როგორც მონაცემთა ბაზის დიზაინერებსა და საბოლოო მომხმარებლებს შორის კომუნიკაციის საშუალება მონაცემთა ბაზის შემუშავების ანალიზის ფაზაში. E-R მოდელი გამოიყენება კონცეპტუალური მონაცემების მოდელის შესაქმნელად, რომელიც წარმოადგენს პროგრამული უზრუნველყოფისგან დამოუკიდებელ მონაცემთა ბაზის სტრუქტურასა და შეზღუდვებს, მაგალითად როგორიცაა მონაცემთა ბაზის მართვის სისტემა.

ზოგიერთ ავტორს E-R მოდელირების განხილვისას შემოაქვს ტერმინები და ცნებები, რომლებიც ეხება რელაციური მონაცემების მოდელის შექმნას. რელაციური მონაცემების მოდელი წარმოადგენს უმეტესი თანამედროვე მონაცემთა ბაზის მართვის სისტემის საფუძველს. კერძოდ, ისინი რეკომენდაციას იძლევიან, რომ E-R მოდელი იყოს მთლიანად ნორმალიზებული, პირველადი და გარე გასაღებების სრული რეზოლუციით. ამან შეიძლება ნაადრევად გადაგვიყვანოს რელაციურ მოდელზე მაშინ, როცა დღევანდელ მონაცემთა ბაზის გარემოში, მონაცემთა ბაზა შეიძლება განხორციელდეს ობიექტზე ორიენტირებული ტექნოლოგიით ან ობიექტზე ორიენტირებული და რელაციური ტექნოლოგიის შერევითაც.

E-R მოდელი შემოიტანა ჩენმა 1976 წელს სტატიით - Chen, P. P.-S. 1976. "The Entity-Relationship Model—Toward a Unified View of Data." *ACM Transactions on Database Systems* 1,1 (March): 9–36., სადაც მან აღწერა E-R მოდელის ძირითადი კონსტრუქციები - „არსები“ და „კავშირები“ და მათთან დაკავშირებული ატრიბუტები. შემდგომში მოდელი გაფართოვდა ჩენის და სხვათა მიერ დამატებით კონსტრუქციების შემოტანით; E-R მოდელი აგრძელებს განვითარებას, მაგრამ, სამწუხაროდ, ჯერ არ არსებობს სტანდარტული აღნიშვნა E-R მოდელირებისთვის. სხვადასხვა პროგრამული ინსტრუმენტები იყენებენ სხვადასხვა აღნიშვნებს, მაგრამ ძირითადი აზრი და პრინციპი ყველასთვის ერთი რჩება.

მიუხედავად ყველაფრისა მონაცემთა მოდელირება სისტემების განვითარების პროცესის ყველაზე მნიშვნელოვანი ნაწილია შემდეგი მიზეზების გამო:

1. მონაცემების მოდელირების დროს მიღებული მონაცემების მახასიათებლებს გადამწყვეტი მნიშვნელობა აქვს მონაცემთა ბაზების, პროგრამებისა და სისტემის სხვა კომპონენტების შემუშავებისას. მონაცემთა მოდელირების პროცესში დაფიქსირებული ფაქტები და წესები არსებითია საინფორმაციო სისტემაში მონაცემთა მთლიანობის უზრუნველსაყოფად;
2. მონაცემები და არა პროცესები მრავალი თანამედროვე საინფორმაციო სისტემის ყველაზე რთული ასპექტი და, შესაბამისად, საჭიროებს ცენტრალურ ადგილს სისტემის მოთხოვნების ჩამოყალიბებაში. ხშირად მონაცემთა მოდელის მიზანია მონაცემთა მდიდარი რესურსის მიწოდების უზრუნველყოფა, რომელიც შეიძლება გამოყენებულ იქნას ნებისმიერი ტიპის ინფორმაციის გამოკვლევის, ანალიზის და შეჯამების მხარდასაჭერად;
3. მონაცემები როგორც წესი უფრო სტაბილურინი არიან, ვიდრე ბიზნეს პროცესები, რომლებიც იყენებენ ამ მონაცემებს. ამრიგად, საინფორმაციო სისტემის დიზაინს, რომელიც ემყარება მონაცემთა ორიენტაციას, უნდა ჰქონდეს უფრო ხანგრძლივი სარგებლიანობა, ვიდრე პროცესზე ორიენტაციის საფუძველზე შექმნილ პროექტს.

წარმოვადგინოთ E-R მოდელირების ძირითადი მახასიათებლები, საერთო აღნიშვნებისა და შეთანხმებების გამოყენებით. დავიწყოთ E-R დიაგრამის ნიმუშით, მათ შორის E-R მოდელის ძირითადი კონსტრუქციებით - **არსები, ატრიბუტები და კავშირები**, შემდეგ კი შემოვიტანოთ ბიზნესის წესების კონცეფცია, რომელიც წარმოადგენს მონაცემთა მოდელირების ყველა კონსტრუქციის საფუძველს. განვსაზღვროთ არსების ტიპები, რომლებიც გავრცელებულია E-R მოდელირებაში: ძლიერი არსები, სუსტი არსები და ასოციაციური არსები; კიდევ რამდენიმე არსის ტიპის განისაზღვრა. ასევე განვსაზღვროთ ატრიბუტების რამდენიმე მნიშვნელოვან ტიპი, მათ შორის სავალდებულო და არასავალდებულო ატრიბუტები, ერთმნიშვნელოვანი და მრავალმნიშვნელოვანი ატრიბუტები, წარმოებული ატრიბუტები და კომპოზიციური ატრიბუტები. შემოვიტანოთ და განვმარტოთ „კავშირთან“ (**დამოკიდებულებასთან**)

დაკავშირებული სამი მნიშვნელოვანი ცნება: კავშირის (ურთიერთობის) ხარისხი, კავშირის (ურთიერთობის) კარდინალურობა და კავშირში (ურთიერთობაში) მონაწილეობის შეზღუდვა.

## არსი-კავშირი მოდელი

არსი-კავშირი მოდელი (E-R მოდელი) არის მონაცემთა დეტალური, ლოგიკური წარმოდგენა ორგანიზაციის ან ბიზნესის სფეროსთვის. E-R მოდელი გამოიხატება ბიზნესის გარემოში არსებული სუბიექტების (არსების), ამ სუბიექტებს შორის ურთიერთობების (კავშირების), ასევე სუბიექტებისა და კავშირების ატრიბუტების (ან თვისებების) საშუალებით. E-R მოდელი ჩვეულებრივ არსების ურთიერთკავშირის სქემა (E-R სქემა, ან ERD), რომელიც წარმოადგენს E-R მოდელის გრაფიკულ გამოსახულებას.

## არსი-კავშირი დიაგრამა

E-R დიაგრამების გაგების გამარტივების მიზნით მოვიყვანოთ მაგალითად მცირე გამარტივებული E-R დიაგრამა (ნახაზზე 2-1) ავეჯის მწარმოებელი კომპანიისთვის. ასეთ სურათს, რომელიც არ შეიცავს ატრიბუტებს, ხშირად საწარმოს მონაცემთა მოდელს უწოდებენ. ნახ. 2-1-ის მიხედვით შევვიძლია ვთქვათ, რომ მრავალი მომწოდებელი (supplier) აწვდის და აგზავნის სხვადასხვა ნივთებს (item) Pine Valley Furniture-ს. საგნები იკრიბება პროდუქტებად (product), რომლებიც მიეყიდება იმ მომხმარებლებს (customer), რომელმაც მოახდინა პროდუქციის შეკვეთა (order). თითოეული მომხმარებლის შეკვეთა შეიძლება შეიცავდეს ერთ ან მეტ სტრიქონს, რომელიც შეესაბამება ამ შეკვეთის პროდუქტებს.

ნახ. 2-1-ზე მოცემულია ამ კომპანიის არსები (სუბიექტები) და კავშირები (ურთიერთობები) ატრიბუტები გამოტოვებულია დიაგრამის გასამარტივებლად. არსები (ორგანიზაციის ობიექტები) წარმოდგენილია მართვულთხედის სიმბოლოთი, ხოლო არსებს (სუბიექტებს) შორის კავშირები (ურთიერთობები) წარმოდგენილია არსების დამაკავშირებელი ხაზებით. ნახ. 2-1-ში მოცემული არსები მოიცავს შემდეგს:

**CUSTOMER** მომხმარებელი - პირი ან ორგანიზაცია, რომელმაც შეუკვეთა ან შესაძლოა შეუკვეთოს პროდუქტები. მაგალითი: შ.პ.ს. „ბიბო“.

**PRODUCT** Pine Valley Furniture ქარხნის მიერ დამზადებული ავეჯის სახეობა, რომლის შეკვეთა შეუძლია მომხმარებელს.

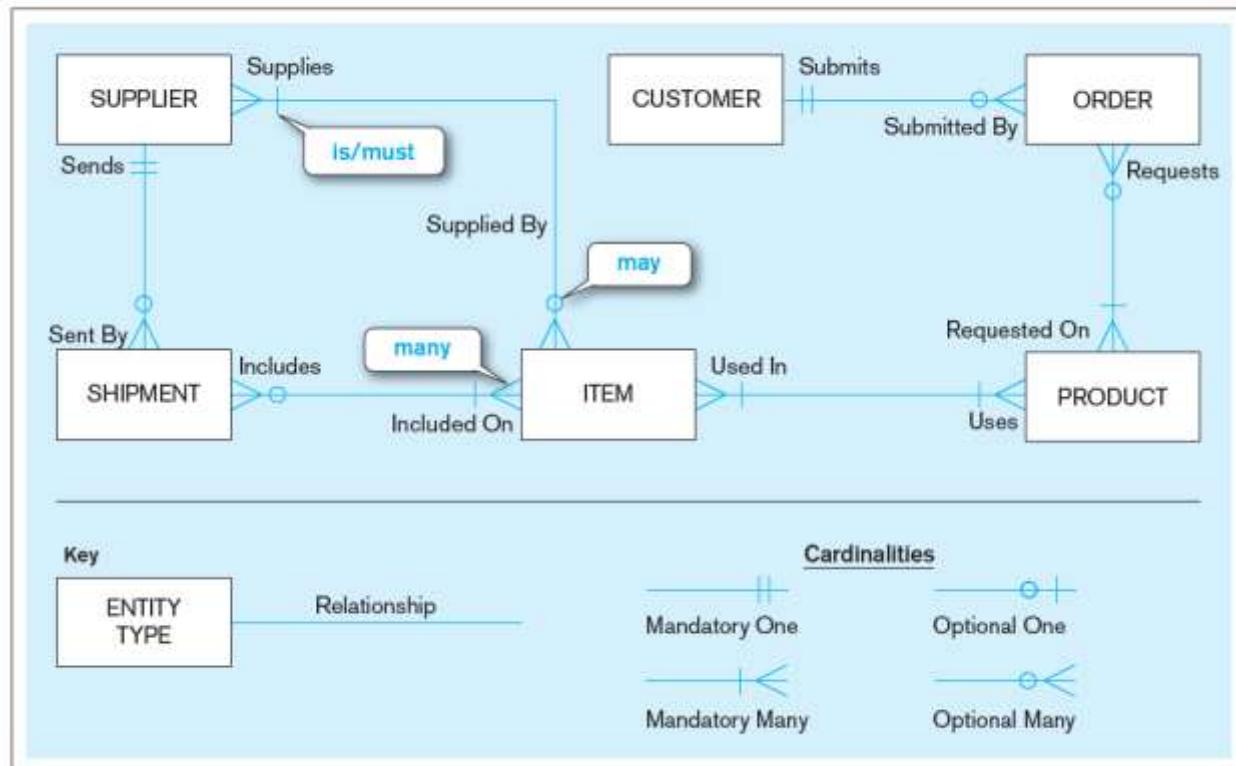
გავითვალისწინოთ, რომ პროდუქტი არ არის კონკრეტული „კარადა“, რადგან ცალკეულ კარადაზე თვალის დევნება არა საჭირო. მაგალითი: მუხის წიგნის კარადა, სახელწოდებით O600.

**ORDER** შეკვეთა, რომელიც დაკავშირებულია მომხმარებელზე ერთი ან მეტი პროდუქტის რეალიზაციასთან და იდენტიფიცირებულია გაყიდვების ან ბუღალტრული აღრიცხვის გარიგების ნომრით. მაგალითი: 2015 წლის 10 სექტემბერს შ.პ.ს. „ბიბომ“ შეიძინა ერთი პროდუქტი O600 და ოთხი პროდუქტი O623.

**ITEM** კომპონენტის ტიპი, რომელიც შედის ერთი ან მეტი პროდუქტის წარმოებაში და მათი მომარაგება შეიძლება ერთი ან მეტი მომწოდებლის მიერ. მაგალითი: საკეტი მექანიზმი, სახელწოდებით I-27-4375.

- SUPPLIER სხვა კომპანია, რომელსაც შეუძლია მიაწოდოს სხვადასხვა საჭირო საქონელი  
Pine Valley Furniture- ს. მაგალითი: შ.პ.ს. „მუხა“
- SHIPMENT ტრანზაქცია (გადაზიდვა) რომელიც დაკავშირებილია Pine Valley Furniture  
ქარხნისათვის სხვადასხვა კომპონენტების (ნივთების) მიმწოდებლისგან  
მიღების გარიგებასთან. ტვირთის ყველა ნივთი ერთ სარეგისტრაციო  
დოკუმენტზე აისახება. მაგალითად: 300 I-27- ის მიღება.

**FIGURE 2-1** Sample E-R diagram



### ნახ.2-1 გამარტივებული E-R დიაგრამა

გავითვალისწინოთ, რომ მნიშვნელოვანია თითოეული არსის მკაფიოდ განსაზღვრა, როგორც მეტამონაცემები. მაგალითად, მნიშვნელოვანია ვიცოდეთ, რომ მომხმარებელთა (CUSTOMER) არსი მოიცავს პირებს ან ორგანიზაციებს, რომლებსაც ჯერ არ აქვთ ნაყიდი პროდუქტები Pine Valley Furniture-დან. ასევე ჩვეულებრივი მოვლენაა, რომ ერთი ორგანიზაციის სხვადასხვა დეპარტამენტისთვის ერთი და იგივე ტერმინს შეიძლება განსხვავებული მნიშვნელობა ჰქონდეს (ომონიმები). მაგალითად, ბუღალტერიამ შეიძლება მომხმარებლად მიიჩნიოს მხოლოდ ის პირები ან ორგანიზაციები, რომლებმაც ოდესმე შესყიდვა შეასრულეს, ამით გამორიცხავენ პოტენციურ მომხმარებლებს, ხოლო მარკეტინგი მომხმარებელად მიიჩნევს ყველას, ვინც დაუკავშირდა ქარხანას საქონლის შეძენის მიზნით, ან რომელმაც შეიძინა საქონელი Pine Valley Furniture-დან ან აქვს ინფორმაცია ცნობილი კონკურენტებიდან. ზუსტი და საფუძვლიანი ERD დიაგრამა მკაფიო მეტამონაცემების გარეშე, სხვადასხვა ადამიანის მიერ შეიძლება სხვადასხვა გვარად განიმარტოს.

ERD-ზე თითოეული ხაზის ბოლოს მითითებული სიმბოლოები განსაზღვრავს კავშირის სიძლიერეს (მას კარდინალურობასაც უწოდებენ), რომლებიც ასახავს თუ ერთი სახის რამდენი ერთეული უკავშირდება სხვა სახის რამდენ ერთეულს. ნახ. 2-1-ის შესწავლისას ვხედავთ, რომ ეს კარდინალური სიმბოლოები გამოხატავს შემდეგ ბიზნეს წესებს:

1. მიმწოდებელმა (SUPPLIER) შეიძლება მიაწოდოს მრავალი ნივთი (ITEM) ("შეიძლება მიწოდება"-ში იგილისხმება, რომ მიმწოდებელმა შეიძლება არ მიაწოდოს რაიმე ნივთი). თითოეული ნივთის მოწოდება ხდება ნებისმიერი რაოდენობის მიმწოდებლის მიერ („მოწოდევა ხდება“ იგულისხმება, რომ ნივთი უნდა მოაწოდოს მინიმუმ ერთმა მიმწოდებელმა მაინც).
2. თითოეული ნივთი (ITEM) უნდა იქნას გამოყენებული მინიმუმ ერთი პროდუქტის (PRODUCT) წარმოებაში და ის შეიძლება გამოყენებულ იქნას მრავალ პროდუქტში. და პირიქით, თითოეულმა პროდუქტმა უნდა გამოიყენოს ერთი ან მეტი ნივთი.
3. მიმწოდებელს (SUPPLIER) შეუძლია მოახდინოს მრავალი გადაზიდვის (SHIPMENT) ტრანზაქცია. ამასთან, თითოეული გადაზიდვა უნდა განხორციელდეს ზუსტად ერთი მიმწოდებლის (SUPPLIER) მიერ. გასათვალისწინებელია, რომ გზავნილები და ნივთები ცალკეული ცნებებია. მიმწოდებელს (SUPPLIER) შეუძლია შეძლოს ნივთის (ITEM) მომარაგება, მაგრამ მას ჯერ არ მოუხდენია ამ ნივთის გადაზიდვები.
4. გადაზიდვა (SHIPMENT) უნდა შეიცვადეს ერთ (ან მეტ) ნივთს (ITEM). ნივთი (ITEM) შეიძლება იყოს შეტანილი რამდენიმე გადაზიდვაში (SHIPMENT).
5. მომხმარებელს (CUSTOMER) შეუძლია გააკეთოს ნებისმიერი რაოდენობის შეკვეთა (ORDER). ამასთან, თითოეული შეკვეთა (ORDER) უნდა გააკეთოს ზუსტად ერთმა მომხმარებელმა (CUSTOMER). იმის გათვალისწინებით, რომ მომხმარებელს (CUSTOMER) შეიძლება არ ჰქონდეს გაკეთებული რაიმე შეკვეთა (ORDER), ზოგიერთი მომხმარებელი (CUSTOMER) უნდა იყოს პოტენციური, არააქტიური ან სხვა მომხმარებელი, რაიმე შესაბამისი შეკვეთის (ORDER) გარეშე.
6. შეკვეთაში (ORDER) უნდა მოითხოვებოდეს ერთი (ან მეტი) პროდუქტი (PRODUCT). მოცემული პროდუქტის მოთხოვნა არ შეიძლება იყოს ნებისმიერ შეკვეთაში ან შეიძლება მოითხოვნილი იყოს ერთი ან მეტი შეკვეთით.

თითოეული კავშირისათვის (ურთიერთობისთვის), ფაქტობრივად, არსებობს ორი ბიზნესის წესი, სათითაო მიმართულებით ერთი საგანიდან მეორეზე. გავითვალისწინოთ, რომ თითოეული ბიზნესის წესები ასრულებს გარკვეულ გრამატიკას:

**<entity> <minimum cardinality> <relationship> <maximum cardinality> <entity>**

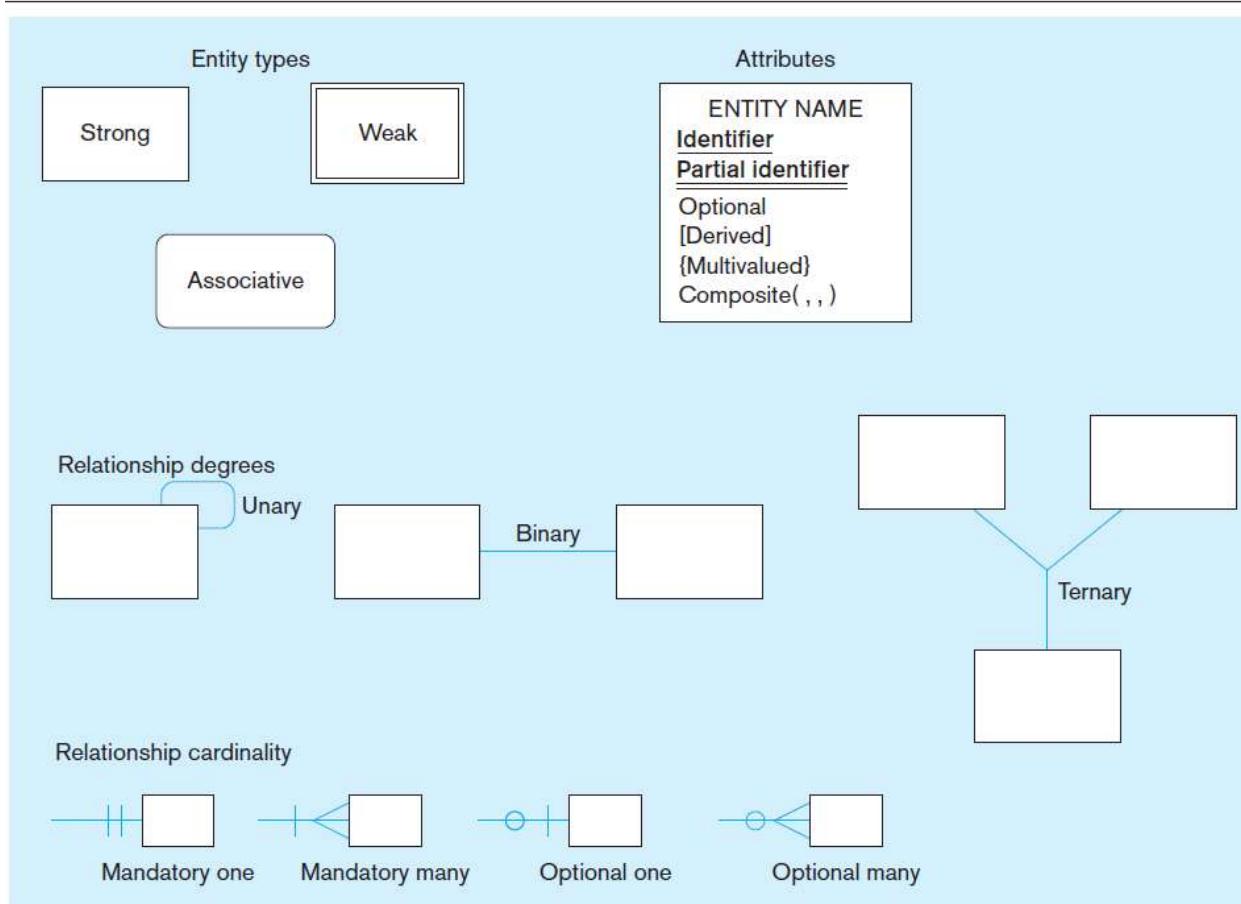
მაგალითად მე-5 წესი ასე გამოისახება:

**<CUSTOMER> <may> <Submit> <any number> <ORDER>**  
**<მომხმარებელი> <შეუძლია> <აკეთებს> <ნებისმიერო რაოდენობის> <შეკვეთა>**

ეს გრამატიკა იძლევა იმის სტანდარტულ გზას, რომ თითოეული ურთიერთობა ჩამოვაყალიბოთ ბიზნესის წესი ბუნებრივ ინგლისურენოვან<sup>1</sup> დებულებაში.

### აღნიშვნები E-R მოდელში

ნახ. 2-2-ზე ნაჩვენებია აღნიშვნები, რომლებიც გამოიყენება E-R დიაგრამებისათვის. სამწუხაროდ ერთიანი სტანდარტის აღნიშვნა არ არსებობს (უმეტესი გამოყენებადი აღნიშვნები განხილული იქნება მოგვიანებით, დამატების სახით). ნახ. 2-2-ში მოცემულია სხვადასხვა სასურველი თვისებების აღნიშვნები, რომლებიც დღეს ყველაზე ხშირად გამოიყენება E-R ნახაზის პროგრამულ ინსტრუმენტებში (ხელსაწყოებში) და ასევე საშუალებას გვაძლევს ზუსტად მოვახდინოთ სიტუაციის მოდელირება პრაქტიკაში.



ნახ.2-2 აღნიშვნები, რომლებიც გამოიყენება E-R დიაგრამებისათვის.

<sup>1</sup> როგორც წესი ICTs პროექტების მოდელირებისას გამოიყენება ინგლისური ენა. საჭიროების შემთავევაში სხვა ენაზე განმარტებები ამ პროექტების განუყოფელი ნაწილია.

ბევრ სიტუაციაში საკმარისია E-R-თვის უფრო მარტივი აღნიშვნების გამოყენება. ხატვის ინსტრუმენტების უმეტესობა, როგორიცაა Microsoft Visio ან SmartDraw ან CASE ინსტრუმენტებში გამოყენებული ხელსაწყოები, როგორიცაა Oracle Designer, CA ERwin, ან powerDesigner, ვერ ასახავენ ყველა არსის და ატრიბუტის ტიპებს რომლებსაც ვიყენებთ დაპროექტებისას. მნიშვნელოვანია აღინიშნოს, რომ ნებისმიერი აღნიშვნებისათვის საჭიროა სპეციალურ ანოტაციები (განმარტებები), რომლებიც ყოველთვის არ არის მოცემული დიაგრამის ინსტრუმენტში, რათა წარმოვადგინოთ ორგანიზაციული სიტუაციის ყველა ბიზნესის წესი.

## ორგანიზაციის (საქმიანი რეგლამენტის) წესების მოდელირება

მონაცემთა მმართველი ორგანიზაციის (ან/და საქმიანი გარემოს) მარეგულირებელი დოკუმენტირებული წესები და პოლიტიკა წარმოადგენენ მონაცემთა მოდელირების საფუძველს. ბიზნესის წესები და პოლიტიკა არეგულირებენ მონაცემთა დამუშავებისა და შენახვის სისტემაში მონაცემთა შექმნას, განახლებას და ამოღებას; ამრიგად, ისინი უნდა იყოს აღწერილი იმ მონაცემებთან ერთად, რომლებთანაც დაკავშირებულია ისინი. მაგალითად, პოლიტიკა „უნივერსიტეტში ყველა სტუდენტს უნდა ჰყავდეს ფაკულტეტის მრჩეველი“ აიმულებს მონაცემებს (მონაცემთა ბაზაში) თითოეული სტუდენტის შესახებ მონაცემები უკავშირდებოდეს რომელიმე მრჩეველის შესახებ მონაცემებს. ასევე, „სტუდენტი არის ნებისმიერი ადამიანი, ვინც ჩააბარა ერთიანი ეროვნული მისაღები გამოცდები“ არა მხოლოდ განსაზღვრავს „სტუდენტის“ ცნებას კონკრეტული უნივერსიტეტისთვის, არამედ ასახავს პოლიტიკას. ამ უნივერსიტეტის (მაგ., აშკარად, კურსდამთავრებულები სტუდენტები არიან, ხოლო საშუალო სკოლის მოსწავლე, რომელმაც გაიარა რეგისტრაცია მისაღებ გამოცდებზე, მაგრამ ვერ ჩააბარა, არ არის სტუდენტი).

ბიზნესის წესები და პოლიტიკა არ არის უნივერსალური; მაგალითად, სხვადასხვა უნივერსიტეტს შეიძლება ჰქონდეს განსხვავებული პოლიტიკა სტუდენტების ჩასარიცხად და შეიძლება მოიცავდეს სხვადასხვა ტიპის სტუდენტებს. ასევე, ორგანიზაციის წესები და პოლიტიკა შეიძლება შეიცვალოს (ჩვეულებრივ ნელა) დროთა განმავლობაში; უნივერსიტეტმა შეიძლება გადაწყვიტოს, რომ სტუდენტს არ უნდა დაენიშნოს ფაკულტეტის მრჩეველი, სანამ სტუდენტი არ აირჩევს სპეციალობას და ა.შ.

მონაცემთა ბაზის შესაქმნელად მონაცემთა ბაზის ანალიტიკოსმა უნდა უზრუნველყოს:

- იმ წესების იდენტიფიცირება და გაგება, რომლებიც მონაცემებს არეგულირებს;
- ამ წესები ისე, წარმოადგინა, რომ მათ ერთმნიშვნელოვნად გაგება შეძლონ ინფორმაციული სისტემის შექმნელებმაც და მომხმარებლებმაც;
- ამ წესების დანერგვა მონაცემთა ბაზის ტექნოლოგიაში.

მონაცემთა მოდელირება მნიშვნელოვანი ინსტრუმენტია ამ პროცესში. იმის გამო, რომ მონაცემთა მოდელირების მიზანია ბიზნესის შესახებ წესების დოკუმენტირება, შემოვიტანოთ მონაცემთა მოდელირების განხილვა და საწარმოსთან ურთიერთობის აღნიშვნები ბიზნესის წესების მიმოხილვით. მონაცემთა მოდელები ვერ შესძლებენ ბიზნესის ყველა წესის წარმოდგენას (და ამის საჭიროება არცაა, რადგან ბიზნესის ყველა წესი არ აწესრიგებს მონაცემებს); მონაცემთა მოდელები შესაბამის დოკუმენტაციასთან და სხვა ტიპის ინფორმაციული სისტემის მოდელებთან ერთად (მაგალითად, მოდელები, რომლებიც

მონაცემების დამუშავების დოკუმენტირებას ახდენს) წარმოადგენენ ბიზნესის ყველა წესს, რომელიც უნდა განხორციელდეს ინფორმაციული სისტემების საშუალებით.

## ბიზნეს წესების მიმოხილვა

ბიზნესის წესი არის ”განაცხადი, რომელიც განსაზღვრავს ან ზღუდავს ბიზნესის გარკვეულ ასპექტს. ის მიზნად ისახავს ბიზნესის სტრუქტურის დამტკიცებას ან ბიზნესის ქცევაზე კონტროლის ან გავლენის მოხდენას. . . წესები ხელს უშლის, განაპირობებს ან გვთავაზობს მოვლენების მოხდენას ”. მაგალითად, შემდეგი ორი განაცხადი არის ბიზნესის წესების საერთო გამოხატულება, რომელიც გავლენას ახდენს მონაცემთა დამუშავებასა და შენახვაზე:

- ”სტუდენტს შეუძლია დარეგისტრირდეს საგანზე მხოლოდ იმ შემთხვევაში, თუ მან წარმატებით შეასრულა ამ საგნის წინაპირობები.”
- ”პრივილიგირებული მომხმარებელი ისარგებლებს 10 პროცენტიანი ფასდაკლებით, თუ მას არ აქვს ვადაგადაცილებული ანგარიშის ბალანსი.”

დღესდღეობით ორგანიზაციების უმეტესობა (და მათი თანამშრომლები) ხელმძღვანელობენ ასეთი წესების ათასობით კომბინაციით. საერთო ჯამში, ეს წესები გავლენას ახდენს ქცევაზე და განსაზღვრავს, თუ როგორ რეაგირებს ორგანიზაცია მის გარემოზე. ბიზნეს წესების შეგროვება და დოკუმენტირება მნიშვნელოვანი, რთული ამოცანაა. ბიზნესის წესების სრულყოფილად შეგროვება და სტრუქტურირება, შემდეგ მონაცემთა ბაზის ტექნოლოგიების საშუალებით მათი დაცვა, იძლევა იმის გარანტიას, რომ ინფორმაციული სისტემას გამართულად მუშაობას და ინფორმაციის მომხმარებლები იგებენ იმას თუ რა შეჰყავთ და რას ხედავენ.

## ბიზნესის წესების პარადიგმა.

ბიზნეს წესების კონცეფცია ინფორმაციული სისტემებში გარკვეული დროის განმავლობაში გამოიყენება. არსებობს მრავალი პროგრამული პროდუქტი, რომლებიც ორგანიზაციებს ეხმარება თავიანთი ბიზნესის წესების მართვაში (მაგალითად, JRules ILOG— სგან, IBM კომპანია). მონაცემთა ბაზის სამყაროში უფრო ხშირად გვხვდება ტერმინი „მთლიანობის შეზღუდვა“ რომელიც დაკავშირებული ამ წესების გამოყენებასთან. ამ ტერმინის აზრი გარკვეულწილად უფრო შეზღუდულია მოცულობაში, რაც ჩვეულებრივ გულისხმობს მონაცემთა ბაზაში მონაცემთა სწორი მნიშვნელობებისა და კავშირების შენარჩუნებას. ბიზნეს წესებზე დაფუძნებული მიდგომა ემყარება შემდეგ პირობებს:

- ბიზნესის წესები საწარმოს მირითადი კონცეფციაა, რადგან ისინი წარმოადგენენ ბიზნესის პოლიტიკას და ხელმძღვანელობენ ინდივიდუალურ და საერთო ქცევას. კარგად სტრუქტურირებული ბიზნეს წესები შეიძლება განისაზღვროს საბოლოო მომხმარებლებისთვის ბუნებრივ ენაზე და სისტემების შემქმნელთა მონაცემთა მოდელში.
- ბიზნესის წესები შეიძლება გამოიხატოს საბოლოო მომხმარებლებისთვის ნაცნობი ტერმინებით. ამრიგად, მომხმარებლებს შეუძლიათ განსაზღვრონ და შემდეგ შეინარჩუნონ საკუთარი წესები.
- ბიზნესის წესების მხარდაჭერა ძალზე ადვილია. ისინი ინახება ცენტრალურ საცავში და თითოეული წესი გამოიხატება მხოლოდ ერთხელ, შემდეგ ნაწილდება ორგანიზაციაში. თითოეული წესის აღმოჩენა და დოკუმენტირება ხდება მხოლოდ ერთხელ, რათა გამოყენებულ იქნას სისტემის განვითარების ყველა პროექტში.

- ბიზნეს წესების შესრულების დაცვა შეიძლება ავტომატიზირდეს პროგრამული უზრუნველყოფის გამოყენებით, რომელსაც შეუძლია განმარტოს წესები და შეასრულოს ისინი მონაცემთა ბაზის მართვის სისტემის მთლიანობის მექანიზმების გამოყენებით.

## ბიზნესის წესების მოქმედების სფერო

მონაცემთა ბაზების დაპროექტებისათვის ჩვენ ვიხილავთ მხოლოს იმ ბიზნეს წესებს, რომლებიც გავლენას ახდენენ მხოლოდ ორგანიზაციის მონაცემთა ბაზაზე. ორგანიზაციების უმეტესობას აქვს უამრავი წესი და/ან პოლიტიკა, რომლებიც ამ განმარტების მიღმაა. მაგალითად, წესი "პარასკევი არის საქმიანი ჩატვირთვის დღე" შეიძლება იყოს მნიშვნელოვანი პოლიტიკის განცხადება, მაგრამ ის არავითარ გავლენას არ ახდენს მონაცემთა ბაზებზე. ამის საპირისპიროდ, წესი "სტუდენტს შეუძლია დარეგისტრირდეს საგანზე მხოლოდ იმ შემთხვევაში, თუ მან წარმატებით დაასრულა ამ საგნის წინაპირობები" არის ჩვენი მოქმედების სფერო, რადგან იგი ზღუდავს იმ ოპერაციებს, რომლებიც შეიძლება დამუშავდეს მონაცემთა ბაზაში. კერძოდ, ეს იწვევს წესის მიეროვანი ტრანზაქციის უარყოფას, რომლის მიხედვითაც ხდება სტუდენტის რეგისტრაცია, რომელსაც არ აქვს აუცილებელი წინაპირობები გავლილი. ბიზნესის ზოგიერთი წესი არ შეიძლება წარმოდგენილი იყოს მონაცემთა საერთო მოდელირების განმარტებში; ის წესები, რომელთა წარმოდგენა არ შეიძლება არსი-კავშირი დიაგრამის ვარიაციით, გადმოცემულია ბუნებრივ ენაზე, ზოგი კი წარმოდგენილია რელაციური მონაცემთა მოდელში.

## ბიზნესის (საქმიანობის) კარგი წესები

იმისდა მიუხედავად თუ რა სახითაა წარმოდგენილი ბიზნესის (საქმიანობის) წესები, ბუნებრივ ენაზე, სტრუქტურირებული მოდელის ან სხვა ინფორმაციული სისტემების დოკუმენტაციით, მათ უნდა ჰქონდეთ გარკვეული მახასიათებლები, რათა შესაბამისობაში იყვნენ ადრე აღწერილ წინაპირობებთან. ეს მახასიათებლები მოყვანილია ცხრილში 2-1. ამ მახასიათებლებს დაკმაყოფილების მეტი შანსი ექნებათ, თუ ბიზნესის წესი განისაზღვრება, მტკიცდება და განკუთვნება ბიზნესს, და არა ტექნიკური სფეროს ადამიანებს. ბიზნესმენები ხდებიან ბიზნესის წესების განმკარგულობრბი. მონაცემთა ბაზის ანალიტიკოსი ხელს უწყობს წესების გამოვლენას და არასწორად გამოხატული წესების გარდაქმნას წესებში, რომლებიც აკმაყოფილებენ სასურველ მახასიათებლებს.

## ბიზნეს წესების შეგროვება

ბიზნესის წესები ჩნდებიან (შესაძლოა არაცხადად) ბიზნესის ფუნქციების, მოვლენების, პოლიტიკის, ერთეულების, დაინტერესებული მხარეების და სხვა ობიექტების აღწერილობაში. ამ აღწერილობის ნახვა შესაძლებელია ინდივიდუალური და ჯგუფური საინფორმაციო სისტემების მოთხოვნების შეგროვებისას ინტერვიუების სესიების ჩანაწერებში, ორგანიზაციულ დოკუმენტებში (მაგალითად, პერსონალის სახელმძღვანელოები, წესები, კონტრაქტები, მარკეტინგის ბროშურები და ტექნიკური ინსტრუქციები) და სხვა წყაროებში. წესების განსაზღვრა ხდება ორგანიზაციაში „ვინ“, „რა“, „როდის“, „სად“, „რატომ“ და „როგორ“ კითხვებზე პასუხების შეგროვებისას. ჩვეულებრივ, მონაცემთა ანალიტიკოსმა ზედმიწევნით უნდა განმარტოს წესების პირველი დებულებები, რადგან თავდაპირველი დებულებები შეიძლება იყოს ბუნდოვანი ან არაზუსტი (რასაც ზოგიერთი ადამიანი „საქმიან ჭორებს“ უწოდებს). ამრიგად, ზუსტი წესები ფორმულირდება განმეორებითი გამოკითხვებით. ამ დროს უნდა დაისვას ისეთი კითხვები, როგორიცაა „ეს

ყოველთვის ასეა?“ ”არსებობს სპეციალური გარემოებები, როდესაც ალტერნატივა ხდება?” “არსებობს თუ არა ამ ადამიანის განსხვავებული ტიპები?” ”მხოლოდ ერთი არის თუ ბევრი?” და ”საჭიროა თუ არა მათი ისტორიის შენახვა, ან ამჟამინდელი მონაცემები მხოლოდ სასარგებლოა?” ასეთი კითხვები შეიძლება სასარგებლო გამოდგეს მონაცემთა მოდელის თითოეული ტიპის კონსტრუქციის წესების დასადგენად.

#### **ცხრილი 2-1 . ბიზნესის (საქმიანობის) კარგი წესები**

მახასიათებელი	განმარტება
დეკლარაციული	ბიზნესის წესი არის პოლიტიკაზე განაცხადი, და არა ის, თუ როგორ ხდება პოლიტიკის მიღება ან განხორციელება; წესი არ აღწერს პროცესს ან განხორციელებას, არამედ აღწერს იმას, თუ რას ამოწმებს პროცესი.
ზუსტი	შესაბამისი ორგანიზაციისათვის წესს მხოლოდ ერთი ინტერპრეტაცია უნდა ჰქონდეს ყველა დაინტერესებული პირისათვის და მისი მნიშვნელობა უნდა იყოს მკაფიო.
ატომური	ბიზნესის წესი აღნიშნავს ერთ განაცხადს და არა რამდენიმეს; წესის არცერთი ნაწილი არ შეიძლება იდგეს განყენებულ წესად (ე.ი. წესი განუყოფელი, მაგრამ საკმარისია).
თანმიმდევრული	ბიზნესის წესი შინაგანად უნდა იყოს არაწინააღდეგობრივი (ანუ არ უნდა შეიცავდეს ურთიერთსაწინააღმდეგო განცხადებებს) და უნდა შეესაბამებოდეს (და არ ეწინააღმდეგებოდეს) სხვა წესებს.
გამომხატველობითი	ბიზნესის წესი უნდა იყოს ჩამოყალიბებული ბუნებრივ ენაზე, მაგრამ ეს უნდა იყოს ასევე წარმოდგენილი სტრუქტურულ ბუნებრივ ენაზე ისე, რომ არ მოხდეს არასწორი ინტერპრეტაცია.
მკაფიო	ბიზნესის წესები არ არის ჭარბი, მაგრამ ბიზნესის წესი შეიძლება ეხებოდეს სხვა წესებს (განსაკუთრებით განმარტებებს).
ბიზნესზე ორიენტირებული	ბიზნესის წესი ფორმულირებულია ისეთი ტერმინებით, რომელიც ბიზნესმენებს ესმით, და რადგან ეს წარმოადგენს ბიზნესის პოლიტიკის დებულებას, მხოლოდ ბიზნესმენებს შეუძლიათ შეცვალონ ან გააუქმონ წესი; ამრიგად, ბიზნესის წესი ბიზნესს ეკუთვნის.

#### **მონაცემთა სახელები და განმარტებები**

მონაცემთა გაგებისა და მოდელირებისთვის ფუნდამენტურია მონაცემთა ობიექტების სახელის განსაზღვრა და მისი განმარტება. მონაცემთა ობიექტების სახელდება და მათი განმარტება, უნდა მოხდეს მანამ, სანამ ისინი ერთმნიშვნელოვნად გამოიყენება

ორგანიზაციის მონაცემების მოდელში. არსი-კავშირი განმარტებებში, არსებს, კავშირებს და ატრიბუტებს უნდა მიეცეს მკაფიო სახელები და განმარტებები.

## მონაცემთა სახელები

არსებობს სპეციალურ სახელმძღვანელო მითითებები არსების, კავშირების და ატრიბუტების სახელდებისათვის არსი-კავშირი მონაცემთა მოდელში, მაგრამ არსებობს ზოგადი სახელმძღვანელო მითითებებიც მონაცემთა ნებისმიერი ობიექტის სახელდების შესახებ. მონაცემთა სახელები უნდა აკმაყოფილებდნენ შემდეგ მოთხოვნებს:

- სახელი უნდა ეხებოდეს ბიზნესს და არა ტექნიკურ (ტექნიკურ ან პროგრამულ უზრუნველყოფას) მახასიათებლებს; ასე რომ, CUSTOMER (მომხმარებელი) კარგი სახელია, მაგრამ File10, Bit7 და Payroll Report Sort Key არ არის კარგი სახელები.
- სახელი უნდა იყოს შინაარსიანი, თითქმის თვითდოკუმენტირებული (ანუ, განმარტება ხვეწავს და განმარტავს სახელს, ობიექტის მნიშვნელობის არსის მითითების გარეშე); უნდა მოვერიდოთ ზოგადი სიტყვების გამოყენებას, როგორიცაა „has“ (აქვს), „is“ (არის), „person“ (პიროვნება) ან „it“ (ის).
- სახელი უნდა იყოს უნიკალური იმ სახელითან შედარებით, რომელიც გამოიყენება მონაცემთა ყველა სხვა ცალკეული ობიექტისთვის; სიტყვები უნდა შეიცავდეს მონაცემთა სახელს, თუ ისინი განასხვავებენ მონაცემთა ობიექტს სხვა მსგავსი მონაცემთა ობიექტებისგან (მაგალითად, სახლის მისამართი უნდა განსხვავდებოდეს კამპუსის მისამართისაგან).
- სახელი უნდა იყოს ადვილად წაკითხვადი ისე, რომ სახელი სტრუქტურირებული იყოს, როგორც ბუნებრივად იქნებოდა ნათქვამი (მაგალითად, Grade Point Average (საშუალო ქულა) კარგი სახელია, ხოლო Average Grade Relative To A (A საშუალო კლასის საშუალო ნიშანი), შესაძლოა ზუსტი იყოს, მაგრამ უხერხული სახელია.
- სახელი უნდა იყოს აღებული სიტყვების დამტკიცებული სიიდან; თითოეული ორგანიზაცია ხშირად ირჩევს ლექსიკას, საიდანაც უნდა აირჩიონ მონაცემთა სახელების მნიშვნელოვანი სიტყვები (მაგალითად, სასურველია მაქსიმუმი, არასდროს ზედა ზღვარი, ჭერი ან უმაღლესი); ასევე შეიძლება გამოყენებულ იქნას ალტერნატიული სახელები ან ფსევდონიმები, როგორც დამტკიცებული აბრევიატურა (მაგ., CUST მომხმარებლისათვის CUSTOMER) და შეიძლება აბრევიატურის გამოყენება რეკომენდირებული იყოს, რათა მოკლე იყოს მონაცემთა სახელები მონაცემთა ბაზის ტექნოლოგიის მაქსიმალური სიგრძის დასაკმაყოფილებლად.
- სახელი უნდა იყოს განმეორებადი, რაც იმას ნიშნავს, რომ სხვადასხვა ადამიანს ან ერთსა და იმავე ადამიანს სხვადასხვა დროს ზუსტად ან თითქმის ერთი და იგივე სახელი უნდა ჩამოუყალიბდეს; ეს ხშირად იმას ნიშნავს, რომ არსებობს სტანდარტული იერარქია ან ნიმუში სახელებისათვის (მაგალითად, სტუდენტის დაბადების თარიღი იქნება სტუდენტის დაბადების თარიღი და დასაქმებულის დაბადების თარიღი იქნება თანამშრომლის დაბადების თარიღი).
- სახელი უნდა იცავდეს სტანდარტულ სინტაქსს, რაც ნიშნავს, რომ სახელის ნაწილები უნდა შეესაბამებოდეს ორგანიზაციის მიერ მიღებულ სტანდარტულ შეთანხმებას.

მონაცემთა სახელდების პროცესში ხდება:

1. მონაცემთა განმარტების მომზადება;

2. უმნიშვნელო ან დაუშვებელი სიტყვების ამოღება (სიტყვები, რომლებიც არ არის დასახელებულთა ნუსხაში); გასათვალისწინებელია, რომ განმარტებაში AND და OR სიტყვების არსებობა შეიძლება გულისხმობდეს ორი ან მეტი მონაცემთა ობიექტის კომბინირებას, და შეიძლება სასურველი გახდეა ობიექტების გამოყოფა და სხვადასხვა სახელების მინიჭება;
3. სიტყვების აზრიანი, განმეორებადი გზით განლაგება;
4. სტანდარტული მნიშვნელობის მინიჭება თითოეული სიტყვისთვის;
5. იმის დადგენა, არსებობს თუ არა ეს სახელი და თუ არსებობს, სხვა საკვალიფიკაციო საშუალებების დამატება, რომ სახელი უნიკალური გახდეს.

### **მონაცემთა განმარტებები**

განმარტება (რომელსაც ზოგჯერ სტრუქტურულ მტკიცებას უწოდებენ) ბიზნესის წესის სახეობად ითვლება. განმარტება არის ტერმინის ან ფაქტის ახსნა. ტერმინი არის სიტყვა ან ფრაზა, რომელსაც კონკრეტული მნიშვნელობა აქვს ბიზნესისთვის. ტერმინების მაგალითებია კურსი, განყოფილება, მანქანის დაქირავება, რეისი, დაჯავშნა და მგზავრი. ტერმინები ხშირად წარმოადგენს საკვანძო სიტყვებს, რომლებიც გამოიყენება მონაცემთა სახელების შესაქმნელად. ტერმინები უნდა განისაზღვროს ფრთხილად და ლაკონურად. ამასთან, საჭირო არ არის ისეთი ზოგადი ტერმინების განსაზღვრა, როგორიცაა დღე, თვე, პიროვნება ან ტელევიზია, რადგან ამ ტერმინებს გაგება უმრავლესობისათვის არაორაზროვანია.

ფაქტი არის კავშირი ორ ან მეტ ტერმინს შორის. ფაქტი დასტურდება, როგორც მარტივი დეკლარაციული განცხადება, რომელიც დაკავშირებულია ტერმინებთან. ფაქტების მაგალითები, რომლებიც განმარტებებია, შემდეგია (განმარტებული ტერმინები მონიშნულია ხაზგასმით):

- კურსი არის სწავლების მოდული კონკრეტულ სასწავლო პროგრამაში. ეს განმარტება აერთიანებს ორ ტერმინს: „სწავლების მოდული“ და „სასწავლო პროგრამა“. ჩავთვალოთ, რომ ეს არის ჩვეულებრივი ტერმინები, რომელთა დამატებით განმარტება არაა საჭირო;
- „მომხმარებელს\_შეუძლია მოითხოვოს მანქანის მოდელი გაქირავების ფილიალში კონკრეტული თარიღისათვის.“ ეს ფაქტი, რომელიც წარმოადგენს დაქირავების მოთხოვნის მოდელის განმარტებას, აკავშირებს ოთხ ხაზგასმულ ტერმინს. ამ ტერმინებიდან სამი წარმოადგენს ბიზნესის სპეციფიკურ ტერმინებს, რომელთა განსაზღვრა ინდივიდუალურად უნდა მოხდეს (თარიღი ჩვეულებრივი ტერმინია).

ფაქტის განცხადი არანაირ შეზღუდვას არ ადებს ფაქტის ეგზემპლატს. მაგალითად, მეორე ფაქტის განცხადებაში უადგილო იქნება იმის დამატება, რომ მომხმარებელს არ შეუძლია მოითხოვოს ორი განსხვავებული მანქანის მოდელი ერთსა და იმავე დღეს. ასეთი შეზღუდვები წარმოადგენს ცალკეულ ბიზნეს წესებს.

## **მონაცემთა კარგი განმარტებები**

მოვახდინოთ კარგი განმარტებების ილუსტრირება არსების, კავშირების და ატრიბუტებისთვის. არსებობს ზოგადი სახელმძღვანელო მითითებები, რომელიც განსაზღვრულია საერთაშორისო სტანდარტის დონეზე (ISO / IEC, 2004):

- განმარტებები (და ყველა სხვა სახის ბიზნესის წესები) გროვდება იმავე წყაროებიდან, საიდანაც ინფორმაციული სისტემების ყველა მოთხოვნა. ამრიგად, სისტემური ანალიტიკოსები და მონაცემთა ანალიტიკოსები უნდა ეძებდნენ მონაცემთა ობიექტებს და მათ განმარტებებს, რადგან ხდება ინფორმაციული სისტემების მოთხოვნების წყაროების შეისწავლა მათ მიერ;
- განმარტებებს, როგორც წესი, თან ახლავს დიაგრამები, მაგალითად, არსი-კავშირი დიაგრამები. განმარტებას არ სჭირდება იმის გამეორება, რაც ნაჩვენებია დიაგრამაზე, არამედ უნდა დაემატოს დიაგრამას.
- განმარტებები წარმოდგება მხოლობით ფორმაში და განმარტავს რა არის მონაცემთა ელემენტი და არა რა არ არის. განმარტებაში გამოიყენება საყოველთაოდ გასაგები თავისი მნიშვნელობით განყენებული ტერმინები და აბრევიატურები და არ შეიცავს სხვა განმარტებებს. ის უნდა იყოს ლაკონური და მონაცემთა არსებით მნიშვნელობაზე კონცენტრირებული, მაგრამ მასში ასევე შეიძლება მითითებული იყოს მონაცემთა ობიექტის ისეთი მახასიათებლები როგორიცაა:
  - ✓ დახვეწილობები
  - ✓ განსაკუთრებული ან გამონაკლისი პირობები
  - ✓ მაგალითები
  - ✓ სად, როდის და როგორ ხდება მონაცემების შექმნა ან გამოთვლა ორგანიზაციაში
  - ✓ მონაცემები სტატიკურია თუ დროთა განმავლობაში იცვლება
  - ✓ მონაცემები სინგულარულია თუ მრავლობითი მათი ატომური ფორმით
  - ✓ ვინ განსაზღვრავს მონაცემთა ღირებულებას
  - ✓ ვინ ფლობს მონაცემებს (ანუ ვინ აკონტროლებს განსაზღვრას და გამოყენებას)
  - ✓ არის თუ არა მონაცემები არასავალდებულო, თუ დასაშვებია ცარიელი (რასაც ჩვენ ნულს ვუწოდებთ) მნიშვნელობები
  - ✓ შესაძლებელია თუ არა მონაცემების დანაწევრება უფრო ატომურ ნაწილებად ან ხშირი კომბინირება სხვა მონაცემებთან უფრო რთული ან აგრეგირიბეული ფორმით. თუ მონაცემთა განმარტებაში არ შედის, ამ მახასიათებლების დოკუმენტირება საჭიროა სხვაგან, სადაც სხვა მეტამონაცემები ინახება.
  - ✓ მონაცემთა ობიექტი არ უნდა დაემატოს მონაცემთა მოდელს, მაგალითად, არსი-კავშირ სქემას, სანამ არ იქნება ზედმიწევნით განმარტებული (და სახელდებული) და არ მოხდება ამ განმარტებაზე შეთანხმება. მაგრამ მოსალოდნელია, რომ მონაცემთა განმარტება შეიცვლება მას შემდეგ, რაც ობიექტს განათავსებენ დიაგრამაზე, რადგან მონაცემთა მოდელის შემუშავების პროცესი ამოწმებს მონაცემების მნიშვნელობის გაგებას. მონაცემების მოდელირება განმეორებადი პროცესია.

მონაცემთა კარგ განმარტებას დიდი მნიშვნელობა აქვს მონაცემთა წარმატებული მოდელის შექმნისათვის. კარგი განმარტებების შემუშავება მეტწილად დამოკიდებულია ორგანიზაციაზე, რისთვისაც იქმნება საინფორმაციო სისტემა. შეიძლება ჩანდეს, რომ ორგანიზაციაში თანხმობის მიღება სხვადასხვა ტერმინებისა და ფაქტების განმარტებების

შესახებ უნდა იყოს შედარებით მარტივი, მაგრამ ჩვეულებრივ ეს ასე არაა. სინამდვილეში, ეს ერთ – ერთი ყველაზე რთული გამოწვევა, რომელსაც ვხვდებით მონაცემთა მოდელირებაში. ორგანიზაციისთვის ჩვეულებრივ მოვლენად შეიძლება ჩაითვალოს მრავალი განმარტების (ალბათ ათეული ან მეტი) არსებობა მაგალითად ისეთი ზოგადი ტერმინებისათვის როგორიცაა მომხმარებელი ან შეკვეთა.

განმარტებების შემუშავებისას დამახასიათებელი პრობლემების საილუსტრაციოდ, განვიხილოთ მონაცემთა ობიექტი “სტუდენტი”, რომელიც გვხვდება ტიპურ უნივერსიტეტში. სტუდენტის განმარტების მაგალითია: ”პირი, რომელიმაც მიიღო ზოგადი განათლება და ჩააბარა ერთიანი ეროვნული გამოცდები.” ეს განმარტება, სავარაუდოდ, სადაცოა, რადგან ის ალბათ ძალიან ვიწროა. ადამიანი, რომელიც სტუდენტია, ჩვეულებრივ, რამდენიმე ეტაპს გადის, როგორიცაა შემდეგი:

1. პერსპექტივა - გარკვეული ფორმალური კონტაქტი, რომელიც მიუთითებს უნივერსიტეტისადმი ინტერესზე
2. აბიტურიებტი - რეგისტრირდება გამოცდებზე
3. ჩარიცხული აბიტურიენტი - ჩაირიცხა უნივერსიტეტში
4. ჩარიცხული სტუდენტი - გაიარა ადმინისტრაციული რეგისტრაცია უნივერსიტეტში
5. აქტიური სტუდენტი - გაიარა აკადემიური რეგისტრაცია უნივერსიტეტში
6. სტატუსშემერებული სტუდენტი - აიღო აკადემიური შევებულება
7. სტატუსშეწყვეტილი სტუდენტი - გაირიცხა უნივერსიტეტიდან და შეიძლება ხელახლა ჩააბაროს მისაღები გამოცდები
8. კურსდამთავრებული - წარმატებით დაასრულა სასწავლო პროგრამა.

წარმოიდგინეთ ამ სიტუაციაში ერთი განმარტების შესახებ კონსენსუსის მოპოვების სირთულე! როგორც ჩანს, შეიძლება განვიხილოთ სამი ალტერნატივა:

1. გამოვიყენოთ მრავალი განმარტება სხვადასხვა სიტუაციების დასაფარავად. ეს შეიძლება უაღრესად დამაბნეველი იყოს, თუ არსებობს მხოლოდ ერთი არსის ტიპი, ამიტომ ეს მიდგომა არ არის რეკომენდებული (მრავალი განმარტება არ არის კარგი განმარტება). შესაძლოა შესაძლებელი იყოს მრავალი არსის ტიპის შექმნა, სტუდენტის თითო სიტუაციისთვის. ამასთან, რადგან არსის ტიპებს შორის მნიშვნელოვანი მსგავსებაა, არისის ტიპებს შორის განსხვავება შეიძლება დამაბნეველი იყოს და მონაცემთა მოდელმა შეიძლება მრავალი კონსტრუქცია წარმოაჩინოს.
2. გამოვიყენოთ ძალიან ზოგადი განმარტება, რომელიც მოიცავს უმეტეს სიტუაციებს. ამ მიდგომას შეიძლება დასჭირდეს მონაცემების დამატება სტუდენტების შესახებ, მოცემული სტუდენტის რეალური სტატუსის დასაფიქსირებლად. მაგალითად, მონაცემები სტუდენტის სტატუსის შესახებ, აბიტურიენტი და ა.შ. შეიძლება იყოს საკმარისი. მეორეს მხრივ, თუ ერთსა და იმავე სტუდენტს შესაძლოა ჰქონდეს მრავალი სტატუსი (მაგალითად, ერთი პროგრამის დასრულების პერსპექტივა და ხელმეორედ დარეგისტრირება მისსაღებ გამოცდებზე) ამან შეიძლება არ იმუშაოს.
3. განვიხილოთ სტუდენტისთვის მრავალი, დაკავშირებული მონაცემთა ობიექტის გამოყენების შესაძლებლობა. მაგალითად, ჩვენ შეგვიძლია შევქმნათ სტუდენტის ზოგადი ტიპი და შემდეგ სხვა კონკრეტული ტიპები უნიკალური მახასიათებლების მქონე სტუდენტებისათვის (ამ შესაძლებლობაზე ვისაუბრებთ მოგვიანებით).

## არსების (სუბიექტების) და ატრიბუტების მოდელირება

E-R მოდელის ძირითადი კონსტრუქციებია არსები ((სუბიექტები), კავშირები და ატრიბუტები. როგორც ნაჩვენებია ნახაზზე 2-2, მოდელი მრავალი ვარიაციის შექმნის საშუალებას იძლევა თითოეული ამ კონსტრუქციისთვის. E-R მოდელის სიმდიდრე საშუალებას აძლევს დამპროეტებლებს ზუსტად და გამომსახველობით მოახდინონ რეალური სიტუაციების მოდელირება, რაც მოდელის პოპულარობას განმარტავს.

### არსები

არსი (სუბიექტი) არის პირი, ადგილი, ობიექტი, ღონისძიება (მოვლენა) ან კონცეფცია მომხმარებლის გარემოში, რომლის შესახებ ორგანიზაციას სურს შეინარჩუნოს/შეინახოს/დაამუშაოს მონაცემები. ამრიგად, არსი არის არსებითი სახელი.

ქვემოთ მოყვანილიაამ ტიპის არსების (სუბიექტების) მაგალითები:

**პირი:** თანამშრომელი, სტუდენტი, პაციენტი

**ადგილი:** მაღაზია, საწყობი, სახელმწიფო

**ობიექტი:** მანქანა, შენობა, ავტომობილი

**ღონისძიება:** რეალიზაცია, რეგისტრაცია, განახლება

**კონცეფცია:** ანგარიში, კურსი, სამუშაო ცენტრი

*Person: EMPLOYEE, STUDENT, PATIENT*

*Place: STORE, WAREHOUSE, STATE*

*Object: MACHINE, BUILDING, AUTOMOBILE*

*Event: SALE, REGISTRATION, RENEWAL*

*Concept: ACCOUNT, COURSE, WORK CENTER*

### არსის (სუბიექტის) ტიპი - არსის (სუბიექტის) ეგზემპლარი

არსებობს მნიშვნელოვანი განსხვავება არსის ტიპებსა და არსის ერთეულებს (ეგზემპლარს) შორის. არსის ტიპი არის არსების ერთობლიობა, რომლებსაც აქვთ საერთო მახასიათებლები ან თვისებები. E-R მოდელის თითოეულ ტიპის არსს ენიჭება სახელი. იმის გამო, რომ სახელი წარმოადგენს ნივთების კრებულს (ან ერთობლიობას), ის ყოველთვის მხოლობით ფორმაშია. საყოველთაოდ მიღებულია არსების ტიპების სახელების გამოსახვა ლათინური ანბანის (ინგლისური) დიდი ასოებით. E-R დიაგრამაში არსის სახელი მოთავსებულია უჯრის შიგნით, რომელიც წარმოადგენს არსის ტიპს (იხ. ნახ. 2-1). **არსის ეგზემპლარი** არის არსის ტიპის ერთი შემთხვევა. ნახ. 2-3 ასახავს განსხვავებას არსის ტიპსა და მის ორ ეგზემპლარს შორის. არსის ტიპი აღწერილია მონაცემთა ბაზაში მხოლოდ ერთხელ (მეტამონაცემების გამოყენებით), ხოლო ამ ტიპის ობიექტის მრავალი შემთხვევა შეიძლება წარმოდგენილი იყოს მონაცემთა ბაზაში შენახული მონაცემებით. მაგალითად, უმეტეს ორგანიზაციებში არის ერთი EMPLOYEE არსის ტიპი, მაგრამ მონაცემთა ბაზაში შეიძლება ინახებოდეს ასობით (ან თუნდაც ათასობით)

ამ არსის ტიპის შემთხვევა. ჩვენ ხშირად ვიყენებთ ტერმინს „არსი“, ვიდრე „არსის ეგზემპლარს“, როდესაც მნიშვნელობა ნათელია ჩვენი განხილვის კონტექსტიდან.

Entity type: EMPLOYEE			
Attributes	Attribute Data Type	Example Instance	Example Instance
Employee Number	CHAR (10)	642-17-8360	534-10-1971
Name	CHAR (25)	Michelle Brady	David Johnson
Address	CHAR (30)	100 Pacific Avenue	450 Redwood Drive
City	CHAR (20)	San Francisco	Redwood City
State	CHAR (2)	CA	CA
Zip Code	CHAR (9)	98173	97142
Date Hired	DATE	03-21-1992	08-16-1994
Birth Date	DATE	06-19-1968	09-04-1975

**FIGURE 2-3 Entity type EMPLOYEE with two instances**

### არსის ტიპი სისტემაში მონაცემების შეტანის, გამოტანის ან მომხმარებლის წარმოდგენის მიმართებაში

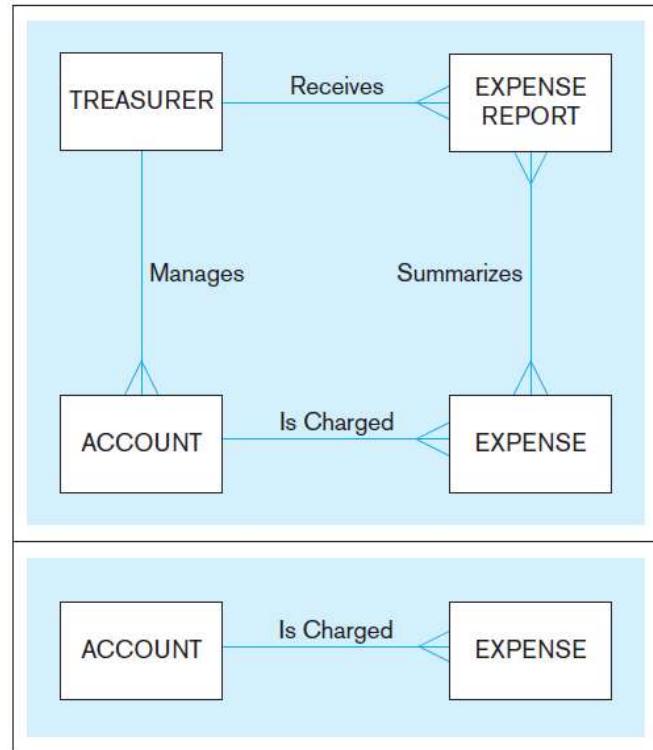
E-R დიაგრამების ხატვის პროცესში გავრცელებულ შეცდომს წარმოადგენს მონაცემთა არსების (სუბიექტების) აღრევა საერთო საინფორმაციო სისტემების მოდელის სხვა ელემენტებთან. ეს ხშირია მაშინ თუ ისინი უკვე იცნობენ მონაცემთა პროცესის მოდელირებას (მაგალითად, მონაცემთა ნაკადის დიაგრამა). ასეთი დაბნეულობის თავიდან ასაცილებლად მარტივი წესი არსებობს - რომ მონაცემთა ჭეშმარიტ არსს აქვს მრავალი შესაძლო ეგზემპლარი, რომელთაგან თითოეულს აქვს განმასხვავებელი მახასიათებელები, ასევე ერთი ან რამდენიმე სხვა აღწერითი მონაცემი.

განვიხილოთ ნახ. 2-4 ა, რომელიც შეიძლება შედგენილი იყოს რომელიდაც კლუბის ხარჯების აღრიცხვის სისტემისთვის საჭირო მონაცემთა ბაზაში. ამ სიტუაციაში, კლუბის ხაზინადარი მართავს ანგარიშებს, იღებს ხარჯების ანგარიშებს და აფიქსირებს ხარჯების ტრანზაქციებს თითოეული ანგარიშის შესაბამისად. ამასთან უნდა განვსაზღვროთ საჭიროა თუ არა თვალყური ვადევნოთ ხაზინადარის (TREASURER არსის ტიპი) მონაცემებს და მის ზედამხედველობას ანგარიშებზე (კავშირი Manages) და ანგარიშების მიღებას (კავშირი Receives). ხაზინადარი არის პირი, რომელიც შეიყვანს მონაცემებს ანგარიშებისა და ხარჯების შესახებ და იღებს ხარჯების ანგარიშებს. ეს არის მონაცემთა ბაზის მომხმარებელი. იმის გამო, რომ არსებობს მხოლოდ ერთი ხაზინადარი, TREASURER მონაცემების შენახვა საჭირო არ არის. გარდა ამისა უნდა განისაზღვროს საჭიროა თუ არა არსი EXPENSE REPORT (ხარჯების ანგარიში). იმის გამო, რომ ხარჯების ანგარიში გამოითვლება ხარჯების გარიგებებიდან და ანგარიშის ნაშთებიდან, ეს არის მონაცემთა ბაზაში მონაცემების მოპოვებისა და ხაზინადარის მიერ მიღებული ინფორმაციის შედეგი. მიუხედავად იმისა, რომ დროთა განმავლობაში ხაზინადარს გადაეცემა ხარჯების შესახებ მრავალი შემთხვევის აღწერი მონაცემები,

რომლებიც საჭიროა ანგარიშის შინაარსის გამოსათვლელად ყოველ ჯერზე, ეს ინფორმაცია უკვე წარმოდგენილია ACCOUNT და EXPENSE არსების ტიპებით.

**FIGURE 2-4** Example of inappropriate entities  
(a) System user (Treasurer) and output (Expense Report) shown as entities

(b) E-R diagram with only the necessary entities



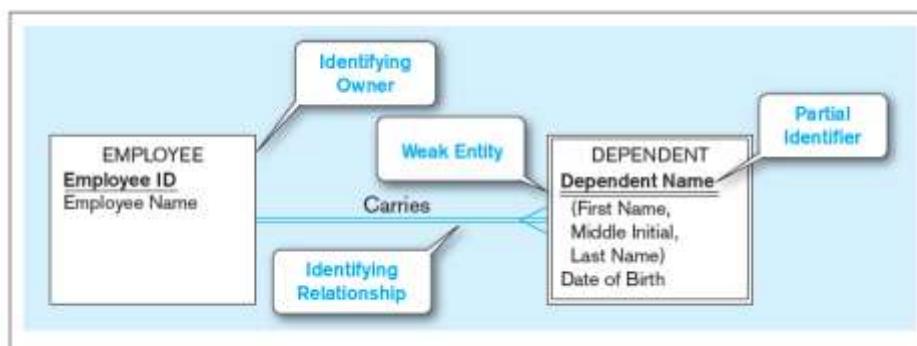
კიდევ ერთი განმარტება იმის გასაგებად, თუ რატომ შეიძლება იყოს შეცდომა ERD ნახაზზე 2-4 ა, არის კავშირების სახელების ხასიათი, „Receives“ (მიღება) და „Summarizes“ (შეჯამება). კავშირების ეს სახელები განეკუთვნება ბიზნეს საქმიანობას, რომელიც გადასცემს ან გარდაქმნის მონაცემებს, და არა უბრალოდ ერთი სახის მონაცემების სხვა სახის მონაცემებთან ასოცირებას. ნახ. 2-4 ბ-ში მოცემული მარტივი E-R დიაგრამა გვიჩვენებს არსებს და კავშირს, რომელიც საკმარისი იქნება კლუბის ხარჯების სისტემის მოსაგვარებლად, როგორც აღწერილია აქ.

### ძლიერი და სუსტი სახეობის არსის ტიპები

ორგანიზაციაში იდენტიფიცირებული ძირითადი არსების უმეტესობა კლასიფიცირდება როგორც ძლიერი სახეობი არსის ტეპები. ძლიერი არსი (სუბიექტი) არის ის არსი, რომელიც არსებობს სხვა ტიპის არსისაგან (სუბიექტისაგან) დამოუკიდებლად (მონაცემთა მოდელირების ზოგი პროგრამული უზრუნველყოფა, სინამდვილეში, იყენებს ტერმინს დამოუკიდებელი არსი/სუბიექტი). მაგალითებად გამოდგება STUDENT (სტუდენტი), EMPLOYEE (თანამშრომელი), AUTOMOBILE (ავტომობილი). ძლიერი ტიპის არსებს ყოველთვის აქვთ უნიკალური მახასიათებელი (ე.წ. იდენტიფიკატორი) - ეს არის ატრიბუტი ან ატრიბუტების კომბინაცია, რომელიც ცალსახად გამოყოფს ამ არსის თითოეულ მოვლენას (ეგზემპლარს).

ამის საპირისპიროდ, სუსტი სახეობის არსის ტიპი არის ისეთი არსის ტიპი, რომლის არსებობა დამოკიდებულია სხვა არსზე (სუბიექტზე). (მონაცემთა მოდელირების ზოგი პროგრამა, სინამდვილეში, იყენებს ტერმინს დამოკიდებული არსი/სუბიექტი.) E-R დიაგრამაში სუსტი ტიპის არსის (სუბიექტს) არ აქვს ბიზნესისთვის მნიშვნელობა იმ არსის (სუბიექტის გარეშე), რომელზეც ის დამოკიდებულია. არსს, რომელზეც დამოკიდებულია სუსტი არსი ეწოდება საიდენტიფიკაციო მფლობელი (ან მოკლედ უბრალოდ მფლობელი). სუსტი არსს, როგორც წესი, არ აქვს საკუთარი იდენტიფიკატორი. საერთოდ, E-R დიაგრამაზე, სუსტ არსს აქვს ატრიბუტი, რომელიც ნაწილობრივ იდენტიფიკატორს წარმოადგენს. შემდგომი დაპროექტების ეტაპზე სუსტი არსისთვის იდენტიფიკატორის ჩამოყალიბდება ნაწილობრივი იდენტიფიკატორისა და მისი მფლობელის იდენტიფიკატორის შერწყმით ან სუროგატი იდენტიფიკატორის ატრიბუტის შექმნით ხდება.

სუსტი არსის მაგალითი, საიდენტიფიკაციო კავშირით, ნაჩვენებია ნახ. 2-5. EMPLOYEE არის ძლიერი ტიპის არსი, იდენტიფიკატორით Employee ID (თანამშრომელი ID) (იდენტიფიკატორი აღნიშნულია ატრიბუტის ხაზგასმით). DEPENDENT არის სუსტი ტიპის არსი, რასაც ორმაგ ხაზოვანი მართვულებელი მიუთითებს. სუსტი არსის ტიპსა და მის მფლობელს შორის კავშირს საიდენტიფიკაციო კავშირი ეწოდება. ნახაზზე 2-5, Carries არის საიდენტიფიკაციო კავშირი (მითითებულია ორმაგი ხაზით). Aტრიბუტი Dependent Name ნაწილობრივ იდენტიფიკატორს წარმოადგენს. Dependent Name (დამოკიდებული სახელი) არის კომპოზიციური ატრიბუტი, რომელიც შეიძლება დაიყოს კომპონენტებად (ამ საკითხს მოგვიანებით განვიხილავთ). ნაწილობრივ იდენტიფიკატორის აღსანიშნად გამოიყენება ორმაგი ხაზგასმა. შემდგომი დაპროექტების ეტაპზე, Dependent Name (დამოკიდებული სახელი) შეერწყმება Employee ID (თანამშრომელი ID) (მესაკუთრის იდენტიფიკატორი) და შექმნის სრულ იდენტიფიკატორს DEPENDENT-ისთვის. ძლიერი და სუსტი არსების წყვილების რამდენიმე დამატებითი მაგალითია: BOOK – BOOK COPY, PRODUCT – SERIAL PRODUCT.



**FIGURE 2-5** Example of a weak entity and its identifying relationship

### არსების (სუბიექტების) ტიპების სახელდებაა და განმარტება

მონაცემთა ობიექტების სახელდებისა და განსაზღვრის ზოგადი სახელმძღვანელო მითითებების გარდა, არსებობს რამდენიმე სპეციალური სახელმძღვანელო მითითება არსების ტიპების სახელდებისათვის, რომლებიც შემდეგში მდგომარეობს:

- არსის (სუბიექტის) ტიპის სახელი არის მხოლობითი არსებითი სახელი (როგორიცაა CUSTOMER, STUDENT ან AUTOMOBILE); არსი არის პიროვნება, ადგილი, ობიექტი,

მოვლენა ან კონცეფცია, ხოლო სახელი არის არსის ტიპისთვის, რომელიც წარმოადგენს არსების ეგზემპლარების ერთობლიობას (მაგ., STUDENT წარმოადგენს სტუდენტებს ნანა ხაჩიძე, დავით დიდუბელი და ა.შ. ) ხშირია მრავლობითი ფორმის მითითება (შესაძლებელია E-R დიაგრამის თანმხლები CASE ხელსაწყოს საცავში), რადგან ზოგჯერ E-R დიაგრამა უკეთესად იკითხება მრავლობითის გამოყენებით. მაგალითად, ნახ. 2-1, ჩვენ ვიტყოდით, რომ მიმწოდებელმა შეიძლება უზრუნველყოს ITEMS (ნივთების) მოწოდება. იმის გამო, რომ მრავლობითი რიცხვი ყოველთვის არ იქმნება მხოლობით არსებითი სახელზე s –ის დამატებით, უმჯობესია ზუსტი მრავლობითი ფორმის დოკუმენტირება.

- არსის ტიპის სახელი უნდა იყოს სპეციფიკური ორგანიზაციისთვის. ამრიგად, ერთმა ორგანიზაციამ შეიძლება გამოიყენოს არსის ტიპის სახელი CUSTOMER, ხოლო მეორე ორგანიზაციამ შეიძლება გამოიყენოს არსის ტიპის სახელი CLIENT (ეს ერთ-ერთი ამოცანა, მაგალითად, შეძენილი მონაცემების მოდელის გამართვისას). სახელი უნდა იყოს აღწერითი ყველასათვის ორგანიზაციაში და განსხვავებული ორგანიზაციის ყველა სხვა ტიპის არსების სახელისგან. მაგალითად, PURCHASE ORDER (შეკვეთის შესყიდვა) შეკვეთებისთვის განსხვავდება CUSTOMER ORDER (მომზმარებლის შესყიდვა) შესყიდვების შეკვეთისაგან. არსის ორივე ამ ტიპს არ შეიძლება დაერქვას ORDER.
- არსის ტიპის სახელი უნდა იყოს ლაკონური და რაც შეიძლება ნაკლებსიტყვიანი. მაგალითად, უნივერსიტეტის მონაცემთა ბაზაში, არსის ტიპის სახელი REGISTRATION (რეგისტრაცია) საკმარისი იქნება, თუ ის აღწერს სტუდენტის დარეგისტრირებას საგანზე. ხოლო სახელი STUDENT REGISTRATION FOR CLASS (სტუდენტების რეგისტრაცია კლასში), ზუსტია, მაგრამ ძალიან მრავალსიტყვიანია.
- შესაძლებელია თითოეული ტიპის არსის სახელისთვის მიეთითოს აბრევიატურა, ან მოკლე სახელი, და ეს საკმარისია E-R დიაგრამაში მისი გამოყენებისათვის; აბრევიატურები უნდა ემორჩილებოდეს ყველა იმავე წესს, როგორც არსების სრული სახელები.
- ღონისძიების (მოვლენების) ტიპის არსებს (სუბიექტებს) უნდა დაერქვას ღონისძიების შედეგის და არა ღონისძიების აქტივობის ან პროცესის შესაბამისი სახელი. მაგალითად, პროექტის მენეჯერის მიერ თანამშრომლის პროექტზე სამუსაოდ დანიშვნის შემთხვევაში შედეგად მიიღება „დანიშვნა“ (ASSIGNMENT), ხოლო თუ სტუდენტი დაუკავშირდა თავის ფაკულტეტის მრჩეველს გარკვეული ინფორმაციის მისაღებად არის „კონტაქტი“ (CONTACT).
- ერთი და იგივე ტიპის არსებისათვის გამოყენებული სახელი უნდა იყოს იგივე ყველა E-R დიაგრამებზე, რომელზეც ჩანს ეს არსის ტიპი. ამრიგად, სახელი რომელიც გამოიყენება არსის ტიპისათვის ორგანიზაციისთვის უნდა იყოს სტანდარტი ყველასათვის. ამასთან, არსის ზოგიერთ ტიპს ექნება მეტსახელი, ან ალტერნატიული სახელი, რომელიც სინონიმებია, რომლებიც გამოიყენება ორგანიზაციის სხვადასხვა ნაწილში. მაგალითად, არსის ტიპის ITEM-ს შეიძლება ჰქონდეს მეტსახელი MATERIAL (წარმოებისთვის) და DRAWING (პროექტირებისათვის). მეტსახელები მითითებულია მონაცემთა ბაზის შესახებ დოკუმენტაციაში, როგორიცაა CASE ინსტრუმენტის საცავში.

ასევე არსებობს არსების (სუბიექტების) ტიპების განმარტების გარკვეული კონკრეტული სახელმძღვანელო მითითებები, რომლებიც შემდეგია:

- არსის (სუბიექტის) ტიპის განმარტება ჩვეულებრივ იწყება „X არის. . . ” ეს არის არსის ტიპის მნიშვნელობის ასახვის ყველაზე პირდაპირი და ცხადი გზა.

- არსის (სუბიექტის) ტიპის განმარტება უნდა შეიცავდეს დებულებას იმის შესახებ, თუ რა არის უნიკალური მახასიათებელი არსის (სუბიექტის) ტიპის თითოეული ეგზემპლარისათვის. ხშირ შემთხვევაში, იდენტიფიკატორის მითითება იურიდიული პირის ტიპისთვის ხელს უწყობს პირის იდენტიფიცირებას. ნაბ. 2-4 ბ-ის მაგალითა: „ხარჯი (expense) არის გადახდა ზოგიერთი საქონლის ან მომსახურების შეძენისთვის. ხარჯის დადგენა ხდება უურნალიში შემოსავლის ნომრით.“
- არსის (სუბიექტის) ტიპის განსაზღვრებაში ნათლად უნდა იყოს განმარტებული, თუ არსის რომელი ეგზემპლარები შედიან და არ შედიან ამ ტიპის არსში; ხშირად, საჭიროა ჩამოვთვალოთ იმ არსების ტიპები, რომლებიც გამორიცხულია. მაგალითად, „მომხმარებელი არის ფიზიკური პირი ან ორგანიზაცია, რომელმაც შეუკვეთა ჩვენთან პროდუქტი ან ის, რომელსაც ჩვენ დავუკავშირდით ჩვენი პროდუქტის რეკლამირებისთვის ან პოპულარიზაციისთვის. მომხმარებელი არ მოიცავს პირებს ან ორგანიზაციებს, რომლებიც ყიდულობენ ჩვენს პროდუქტებს მხოლოდ ჩვენი მომხმარებლების, დისტრიბუტორების ან აგენტების საშუალებით.“
- არსის (სუბიექტის) ტიპის განმარტება ხშირად მოიცავს აღწერილობას, თუ როდის იქმნება და იშლება კონკრეტული არსის ტიპის ეგზემპლარი. მაგალითად, წინა პუნქტში, მაგალითიდან შეგვიძლია დავასკვნათ მომხმარებლი აშკარად იქმნება, როდესაც პირი ან ორგანიზაცია აკეთებს თავის პირველ შეკვეთას; რადგან ამ განსაზღვრებაში სხვა რამ არ არის მითითებული, მომხმარებლის მონაცემები არასდროს არ წაიშალება, ან ის წაიშლება ზოგადი წესების საფუძველზე, რომლებიც მიეთითება მონაცემთა ბაზიდან მონაცემთა გასუფთავების შესახებ. განცხადებას იმაზე, თუ როდის უნდა წაიშალოს იურიდიული პირის ჩანაწერი (ეგზემპლარი) ზოგჯერ მოიხსენიება, როგორც არსის ტიპის შენარჩუნება. მომხმარებლის არსის ტიპის განმარტებისთვის შესაძლებელია წაშლის შესახებ განცხადების შექმნა: „მომხმარებელი არარა მომხმარებელი, თუ სამ წელზე მეტი წელი განმავლობაში არ გააკეთა შეკვეთა.“
- ზოგიერთი არსის ტიპისთვის, განმარტებაში უნდა იყოს მითითებული, როდის შეიძლება გარდაიქმნას ეგზემპლარი სხვა ტიპის არსის ეგზემპლარად. მაგალითად, განვიხილოთ სამშენებლო კომპანია, რომლისთვისაც პოტენციური მომხმარებლების მიერ მიღებული წინადადებები ხდება კონტრაქტი. ამ შემთხვევაში, სატენდერო წინადადება შეიძლება განისაზღვროს შემდეგნაირად: ”სატენდერო წინადადება არის ჩვენი ორგანიზაციის იურიდიული შეთავაზება კლიენტისთვის სამუშაოს შესრულების შესახებ. წინადადება იქმნება მაშინ, როდესაც ჩვენი კომპანიის ოფიცერი ხელს აწერს სატენდერო დოკუმენტს; სატენდერო წინადადება ხდება ხელშეკრულების ნიმუში, როდესაც მივიღებთ შეთავაზების ასლს, რომელსაც ხელს აწერს მომხმარებლის ოფიცერი.“
- ზოგიერთი ტიპის არსისათვის, განმარტებამ უნდა მიუთითოს, თუ რა ისტორიის დაცვა ხდება ამ ტიპის არსის ეგზემპლართან მიმართებაში. მაგალითად, ნახაზი 2-1-ში ITEM- ის მახასიათებლები შეიძლება შეიცვალოს დროთა განმავლობაში და შეიძლება დაგვჭირდეს ინდივიდუალური ღირებულებების სრული ისტორიის და მათი მოქმედების დროის შენახვა. ისტორიის შენახვის შესახებ ასეთ განცხადებებს შეიძლება ჰქონდეს გავლენა იმაზე, თუ როგორ წარმოვადგენთ არსის ტიპს E-R დიაგრამაზე და საბოლოოდ, თუ როგორ ვინახავთ მონაცემებს არსის ეგზემპლარებისათვის.

## ატრიბუტები

არსის (სუბიექტის) თითოეულ ტიპს აქვს მასთან დაკავშირებული ატრიბუტების ნაკრები. ატრიბუტი არის არსის ტიპის თვისება ან მახასიათებელი, რომელიც ორგანიზაციის ინტერესს წარმოადგენს. ასევე შესაძლებელია ატრიბუტი ჰქონდეს ზოგიერთ ტიპის კავშირებისაც. ამრიგად, ატრიბუტი არის არსებითი სახელი. ქვემოთ მოცემულია რამდენიმე ტიპური არსი და მათთან დაკავშირებული ატრიბუტები:

STUDENT (სტუდენტი)      Student ID, Student Name, Home Address, Phone Number  
(სტუდენტის ID, სტუდენტის სახელი, სახლის მისამართი, ტელეფონის ნომერი)

AUTOMOBILE (ავტომობილი)      Vehicle ID, Color, Weight, Horsepower (ავტომობილის ID, ფერი, წონა, ცხენის ძალა)

ატრიბუტების დასახელების აღნიშვნისას პირველ ასოდ იყენებენ დიდ ასოებს, რასაც მცირე ასოები მოყვება. თუ ატრიბუტის სახელი ერთზე მეტი სიტყვისგან შედგება, სიტყვებს შორის იყენებენ ინტერვალს და თითოეულ სიტყვას იწყება დიდი ასოთი, მაგალითად, Employee Name (თანამშრომლის სახელი) ან Student Home Address (სტუდენტის სახლის მისამართი). E-R დიაგრამებში (უმეტესში) ატრიბუტი წარმოდგება მისი სახელის განთავსებით მის მიერ აღწერილ არსში. ატრიბუტები ასევე შეიძლება დაკავშირებულნი იყვნენ იყოს კავშირებთანაც. უნდა გავითვალისწინოთ, რომ ატრიბუტი დაკავშირებულია ზუსტად ერთ არსთან ან კავშირთან.

ყურადღება მივაქციეთ ნახ. 2-5, იმ ფაქტს რომ DEPENDENT-ის ყველა ატრიბუტი მხოლოდ თანამშრომლის დამოკიდებულების მახასიათებლებია და არა უშუალოს თანამშრომლის. ტრადიციული E-R აღნიშვნით, არსის ტიპი (არა მხოლოდ სუსტი არსები, არამედ ნებისმიერი არსები) არ შეიცავს იმ არსების ატრიბუტებს, რომლებთანაც ისინი დაკავშირებულნი არიან (რასაც შეიძლება გარე ატრიბუტები ეწოდოს). მაგალითად, DEPENDENT არ შეიცავს რაიმე ატრიბუტს, რომელიც მიუთითებს, თუ რომელ თანამშრომელს უკავშირდება ეს დამოკიდებული თანამშრომელი. E-R მონაცემთა მოდელის ეს არასასურველი ფუნქცია შეესაბამება მონაცემთა ბაზის მონაცემთა გაზიარებულ თვისებას.

## სავალდებულო და არასავალდებულო ატრიბუტები

პოტენციურად თითოეულ არსს (ან არსის ტიპის ეგზემპლარს) გააჩნია მნიშვნელობა, რომელიც ასოცირდება ამ ტიპის არსის თითოეულ ატრიბუტთან. ატრიბუტს, რომელიც უნდა არსებობდეს თითოეული არსის ეგზემპლარისათვის ეწოდება სავალდებულო ატრიბუტი, ხოლო ატრიბუტს, რომელსაც შეიძლება არ ჰქონდეს მნიშვნელობა, არასავალდებულო ატრიბუტი ეწოდება. მაგალითად, ნახ. 2-6 გვიჩვენებს არსის STUDENT ეგზემპლარები მათი ატრიბუტის მნიშვნელობებით. ამ არსის ერთადერთი არასავალდებულო ატრიბუტი არის Major (ზოგიერთ სტუდენტს, ჯერ არ აურჩევიათ სპეციალობა); ამასთან, ორგანიზაციის წესებით, სტუდენტისათვის უნდა ჰქონდეს მნიშვნელობა ყველა დანარჩენ ატრიბუტს; ეს ნიშნავს იმას, რომ ჩვენ არ შეგვიძლია სტუდენტის შესახებ რაიმე მონაცემების შენახვა არსი STUDENT ეგზემპლარში, თუ არ არსებობს მნიშვნელობები ყველა სავალდებულო

ატრიბუტისთვის. E-R დიაგრამის სხვადასხვა აღნიშვნებში, თითოეული ატრიბუტის წინ შეიძლება გამოჩნდეს სიმბოლო, რომელიც მიუთითებს, სავალდებულოა ის (მაგ., \*) თუ არასავალდებულო (მაგ., O), ან სავალდებულო ატრიბუტები ჩაიწერება სქელი შრიფტით, ხოლო არასავალდებულო ატრიბუტები ჩვეულებრივი შრიფტით (ფორმატი, რომელსაც ამ ტექსტში ვიყენებთ); ხშირ შემთხვევაში, სავალდებულო ან არასავალდებულო მითითებულია დამატებით დოკუმენტაციაში. მნიშვნელობის არმქონე ატრიბუტს ნულოვან ატრიბუტს უწოდებენ. გარდა ამისა უნდა აღინიშნოს, რომ თითოეულ არს აუცილებლად აქვს საიდენტიფიკაციო ატრიბუტი, რომელსაც შემდეგ განვიხილავთ, და დამატებით კიდევ ერთი ან მეტი ატრიბუტი. თუ შევცდებით შევქმნათ არსი, რომელსაც აქვს მხოლოდ იდენტიფიკატორი, ეს არსი საგარაუდოდ არ იქნება ლეგიტიმური. მონაცემთა ასეთ სტრუქტურაში შეიძლება უბრალოდ ინახებოდეს იურიდიული მნიშვნელობების ჩამონათვალი ზოგიერთი ატრიბუტისთვის, რომელიც უკეთ ინახება მონაცემთა ბაზის გარეთ.

Entity type: STUDENT				
Attributes	Attribute Data Type	Required or Optional	Example Instance	Example Instance
Student ID	CHAR (10)	Required	876-24-8217	822-24-4456
Student Name	CHAR (40)	Required	Michael Grant	Melissa Kraft
Home Address	CHAR (30)	Required	314 Baker St.	1422 Heft Ave
Home City	CHAR (20)	Required	Centerville	Miami
Home State	CHAR (2)	Required	OH	FL
Home Zip Code	CHAR (9)	Required	45459	33321
Major	CHAR (3)	Optional	MIS	

**FIGURE 2-6** Entity type STUDENT with required and optional attributes

### მარტივი და კომპოზიტიური ატრიბუტები

ზოგიერთი ატრიბუტი შეიძლება დაიყოს მნიშვნელოვან შემადგენელ ნაწილებად (დეტალური ატრიბუტები). გავრცელებული მაგალითია Name (სახელი), რომელიც ნაჩვენებია 2-5 ნახ.-ზე; სხვა მაგალითია Address (მისამართი), რომელიც ჩვეულებრივ შეიძლება დაიყოს შემდეგ კომპოზიტურ ატრიბუტებად: Street Address, City, State, and Postal Code (ქუჩა, ქალაქი, სახელმწიფო და საფოსტო კოდი). კომპოზიტური ატრიბუტი არის ატრიბუტი, რომელიც სედგება სხვადასხვა კომპონენტებისაგან, რომლებიც უფრო დეტალური ატრიბუტებია. ნახ. 2-7 გვიჩვენებს აღნიშვნას, რომელსაც ვიყენებთ ამ მაგალითში მოცემული კომპოზიტური ატრიბუტებისთვის. ხატვის ინსტრუმენტების უმეტესობას არ აქვს კომპოზიტური ატრიბუტების აღნიშვნის საშუალება, ასე რომ უბრალოდ ხდება ყველა კომპონენტის ნაწილის ჩამოთვლა. კომპოზიტური ატრიბუტები უზრუნველყოფს მნიშვნელოვან მოქნილობას მომხმარებლებისთვის, რომლებსაც შეუძლიათ კომპოზიტური ატრიბუტის მითითება როგორც ერთიანი ერთეული, ანდა ამ ატრიბუტის ცალკეული კომპონენტების მითითება. მაგალითად, მომხმარებელს შეუძლია მისამართს ან მიმართოს მის რომელიმე კომპონენტს, მაგალითად ქუჩის მისამართს. გადაწყვეტილება ატრიბუტის მის შემადგენელ

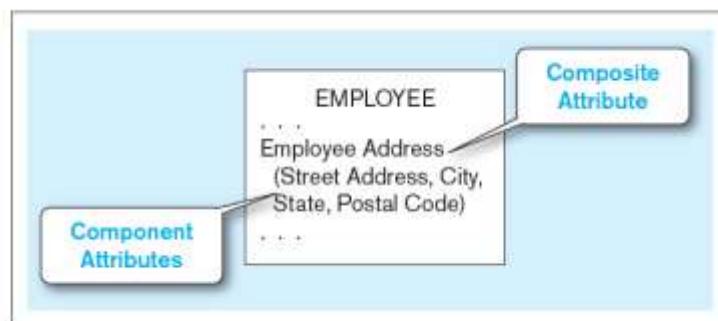
ნაწილებად დაყოფის შესახებ დამოკიდებულია იმაზე სურთ თუ არა მომხმარებლებს ამ ცალკეულ კომპონენტებზე მიმართვის საშუალება, ამიტომ მათ აქვთ ორგანიზაციული მნიშვნელობა. რა თქმა უნდა, დიზაინერი ყოველთვის უნდა შეეცადოს გაითვალისწინოს მონაცემთა ბაზის შესაძლო სამომავლო ვარიანტები.

**მარტივი** (ან ატრიბუტი) ატრიბუტი არის ატრიბუტი, რომლის დაყოფა შეუძლებელია ორგანიზაციისთვის მნიშვნელოვან მცირე კომპონენტებად. მაგალითად, AUTOMOBILE-სთან დაკავშირებული ყველა ატრიბუტი მარტივია: Vehicle ID, Color, Weight, Horsepower (ავტომობილის ID, ფერი, წონა და ცხენის ძალა).

### ერთმნიშვნელოვანი და მრავალმნიშვნელოვანი ატრიბუტები

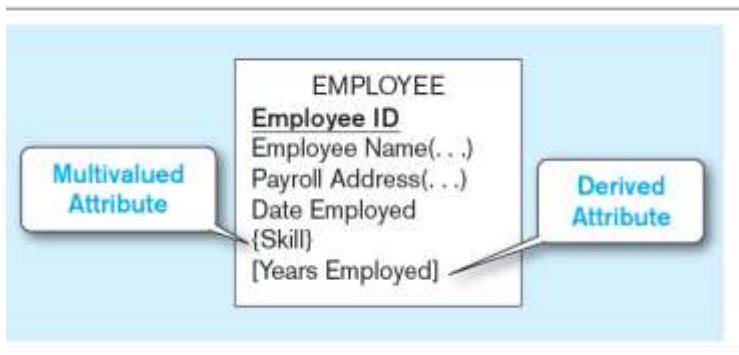
ნახ. 2-6 გვიჩვენებს არსის ორი ეგზემპლარს მათი ატრიბუტის მნიშვნელობებით. თითოეული არსის ეგზემპლარისათვის ფიგურაში თითოეულ ატრიბუტს აქვს ერთი მნიშვნელობა. ხშირად ხდება, რომ არსებობს ატრიბუტი, რომელსაც შეიძლება ჰქონდეს ერთზე მეტი მნიშვნელობა მოცემული ეგზემპლარისათვის. მაგალითად, 2-8-ე EMPLOYEE არსის ტიპს აქვს ატრიბუტი სახელად Skill (უნარი), რომლის მნიშვნელობებით აღწერილია თანამშრომელის უნარი (ან უნარები). რა თქმა უნდა, ზოგიერთ თანამშრომელს შეიძლება ჰქონდეს ერთზე მეტი უნარი, როგორიცაა PHP Programmer და C ++ Programmer. მრავალმნიშვნელოვანი ატრიბუტი არის ატრიბუტი, რომელსაც შეუძლია ერთზე მეტი მნიშვნელობა მიიღოს მოცემული ერთეულის (ან კავშირის) ეგზემპლარისათვის. ამ ტექსტში ჩვენ მიუთითებთ ატრიბუტის მრავალმნიშვნელოვნობაზე სახელწოდების ფიგურულ ფრჩხილებში ჩასმით, როგორც ეს ნაჩვენებია Skill ატრიბუტისათვის EMPLOYEE მაგალითზე 2-8 ნახ.-ზე. Microsoft Visio-ში, ასეთი ატრიბუტის არში განთავსების შემდეგ, შეიძლება შეიცვალოს ეს ატრიბუტი (სვეტი), არჩეულ იქნას კოლექციის ჩანართი და აირჩეს ერთ – ერთი ვარიანტი (როგორც წესი არჩევანი იქნება MultiSet, მაგრამ სხვა ვარიანტი შეიძლება უფრო შესაფერისი იყოს კონკრეტული სიტუაციისთვის) E-R დიაგრამის აგების სხვა საშუალებებში ატრიბუტის სახელის შემდეგ გამოიყენება ვარსკვლავი (\*), ან გამოყენება დამატებითი დოკუმენტაცია მრავალმნიშვნელოვანი ატრიბუტის მისათითებლად.

**FIGURE 2-7** A composite attribute



მრავალმნიშვნელოვანობა და კომპოზიტიურობა განსხვავებული ცნებებია, თუმცა დამწყებთა მიერ ხშირად ხდება ამ ტერმინებს არევა. ზემოთ მოყვანილ მაგალითში მრავალმნიშვნელოვანი ატრიბუტი Skill (უნარი), შეიძლება მრავალჯერ შეგვხვდეს თითოეული თანამშრომლისათვის; თანამშრომლის სახელი და სახელფასო მისამართი ორივე

სავარაუდო კომპოზიტური ატრიბუტია, თითოეული მათგანი ერთხელ გვხვდება თითოეული თანამშრომლისთვის, მაგრამ მათ აქვთ კომპონენტი, უფრო ატომური ატრიბუტები, რომლებიც სიმარტივისთვის ნაჩვენებია 2-8 ნახაზზე.



**FIGURE 2-8** Entity with multivalued attribute (Skill) and derived attribute (Years Employed)

### შენახული და ნაწარმოები ატრიბუტები

ზოგიერთი ატრიბუტის მნიშვნელობა, რომელიც მომხმარებელს აინტერესებს, შეიძლება გამოითვალის ან გამომდინარეობდეს სხვა მასთან დაკავშირებული ატრიბუტის მნიშვნელობიდან, რომელიც ინახება მონაცემთა ბაზაში. მაგალითად, დავუშვათ რომ ორგანიზაციისთვის, EMPLOYEE არსის ტიპს აქვს Date Employment ატრიბუტი. თუ მომხმარებლებმა უნდა იცოდნენ რამდენი წელია ადამიანია დასაქმებული, ეს მნიშვნელობა შეიძლება გამოითვალის დასაქმების თარიღისა და დღევანდელი თარიღის გამოყენებით. მიღებული ატრიბუტი არის ატრიბუტი, რომლის მნიშვნელობების გამოთვლა შესაძლებელია დაკავშირებული ატრიბუტის მნიშვნელობებით (გარდა ამისა, მონაცემები არ არის მომხმარებლის ბაზაში, მაგალითად, დღევანდელი თარიღი, მიმდინარე დრო ან სისტემის მომხმარებლის მიერ მოწოდებული უსაფრთხოების კოდი). ნაწარმოები ატრიბუტი E-R დიაგრამაში ატრიბუტის სახელის გარშემო კვადრატული ფრჩხილების გამოყენებით აღინიშნება, როგორც ეს ნაჩვენებია ნახ. 2-8 – ში Years Employment ატრიბუტისთვის. ზოგიერთი E-R დიაგრამის ინსტრუმენტი იყენებს ატრიბუტის სახელის წინ დახრილი ხაზის (/) აღნიშვნას, რომ მიუთითოს, რომ იგი ნაწარმოებია.

ზოგიერთ სიტუაციაში, ატრიბუტის მნიშვნელობა შეიძლება გამომდინარეობდეს დაკავშირებული არსის ატრიბუტებიდან. მაგალითად, გაითვალისწინეთ Pine Valley Furniture Company-ში თითოეული მომხმარებლისთვის შექმნილი ინვოისი. შეკვეთა Total იქნება INVOICE არსის ატრიბუტი, რომელიც მიუთითებს დოლარის მთლიანი თანხით, რომელიც დარიცხულია მომხმარებლისთვის. შეკვეთის ჯამური ღირებულების გამოთვლა შესაძლებელია გაფართოებული ფასის მნიშვნელობების (ერთეულის ფასი გამრავლებული გაყიდული რაოდენობის) ჯამით სხვადასხვა ხაზის ერთეულებისათვის, რომლებიც ასახულია ინვოისზე. ისეთი მნიშვნელობების გამოთვლის ფორმულები, როგორიცაა ბიზნესის ერთ – ერთი წესი.

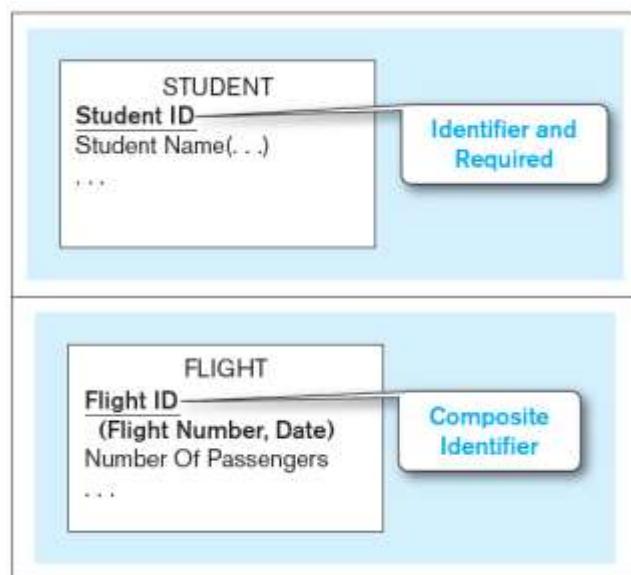
## იდენტიფიკატორი ატრიბუტი

იდენტიფიკატორი არის ატრიბუტი (ან ატრიბუტების კომბინაცია), რომლის მნიშვნელობა განასხვავებს არსის ტიპის ინდივიდუალურ შემთხვევებს. არსის ტიპის ერთიდაგივე არცერთ ორ ეგზემპლარს არ შეიძლება ჰქონდეს ერთიდაგივე მნიშვნელობის იდენტიფიკატორი ატრიბუტი. ადრე მოყვანილ მაგალითსი STUDENT (სტუდენტი) ტიპის არსის იდენტიფიკატორი არის Student ID, ხოლო AUTOMOBILE- ის იდენტიფიკატორი Vehicle ID. გავითვალისწინოთ, რომ ატრიბუტი, როგორიცაა Student Name, არ არის იდენტიფიკატორი, რადგან ბევრ სტუდენტს შეიძლება ჰქონდეს იგივე სახელი და სტუდენტებს, ისევე როგორც ყველა ადამიანს, შეუძლიათ შეცვალონ თავიანთი სახელები. არსებობს კანდიდატი ინედტიფიკატორის ცნებაც. ატრიბუტი კანდიდატი იდენტიფიკატორი რომ იყოს, არსის თითოეული მნიშვნელობისათის უნდა ჰქონდეს ერთი მნიშვნელობა. E-R დიაგრამაზე იდენტიფიკატორის სახელი აღინიშნება ხაზის გასმით, როგორც ეს ნაჩვენებია STUDENT არსის ტიპის მაგალითზე, ნახაზი 2-9 ა. ატრიბუტი იდენტიფიკატორი რომ იყოს, ატრიბუტი უნდა იყოს აუცილებელი (ასე რომ განმასხვავებელი მნიშვნელობა უნდა არსებობდეს), ამიტომ იდენტიფიკატორი ასევე არის სქელი შრიფტით გამოსახული. E-R ნახაზის ზოგიერთი პროგრამა იდენტიფიკატორის წინ განათავსებს სიმბოლოს, რომელსაც ეწოდება სტერეოტიპი (მაგ., <>ID> ან <>PK>>).

ზოგიერთი არსის ტიპისთვის არ არსებობს ერთი (ან ატომური) ატრიბუტი, რომელიც იდენტიფიკატორის ფუნქციას შეასრულებს (ან უზრუნველყოფს უნიკალურობას). ამასთან, ორი (ან მეტი) ატრიბუტი, რომელიც გამოიყენება კომბინირებულად, შეიძლება გამოდგეს როგორც იდენტიფიკატორი. კომპოზიტური იდენტიფიკატორი არის იდენტიფიკატორი, რომელიც შედგება კომპოზიტური ატრიბუტისგან. დიაგრამა 2-9 ბ აჩვენებს ობიექტს FLIGHT კომპოზიტური იდენტიფიკატორით Flight ID. ფრენის ID-ს თავის მხრივ აქვს კომპონენტის ატრიბუტები ფრენის ნომერი და თარიღი. ეს კომბინაცია საჭიროა FLIGHT ცალსახად დასადგენად.

**FIGURE 2-9** Simple and composite identifier attributes  
(a) Simple identifier attribute

(b) Composite identifier attribute



ზოგიერთ არსს შეიძლება ჰქონდეს ერთზე მეტი კანდიდატი იდენტიფიკატორი. თუ არსებობს ერთზე მეტი კანდიდატი იდენტიფიკატორი, დიზაინერმა იდენტიფიკატორად უნდა აირჩიოს ერთი მათგანი. არსებობს კრიტერიუმები, რომელიც გამოიყენება იდენტიფიკატორების შერჩევისას:

1. იდენტიფიკატორად უნდა შეირჩეს ისეთი ატრიბუტი, რომელიც არ შეცვლის მნიშვნელობას არსის ტიპის თითოეული ეგზემპლარის სასიცოცხლო ციკლის განმავლობაში. მაგალითად, თანამშრომლის სახელისა და ხელფასის კომბინაცია (თუნდაც უნიკალური) ცუდი არჩევანი იქნება, როგორც EMPLOYEE-ს იდენტიფიკატორი, რადგან თანამშრომლის სახელისა და ხელფასის მნიშვნელობები შეიძლება ადვილად შეიცვალოს თანამშრომლის დასაქმების პერიოდში.
2. იდენტიფიკატორი უნდა აირჩეს ისე, რომ არსის თითოეული ეგზემპლარისათვის ამ ატრიბუტს გარანტირებული ჰქონდეს სწორი მნიშვნელობის მინიჭება და არ იყოს ნულოვანი (ან უცნობი). თუ იდენტიფიკატორი კომპოზიტური ატრიბუტია, მაგალითად, ფრენის ID 2-9 ბსურათზე, დარწმუნებულები უნდა ვიყოთ, რომ იდენტიფიკატორის ყველა ნაწილს ექნება სწორი მნიშვნელობები.
3. უნდა მოვერიდოთ ე.წ. ინტელექტუალური იდენტიფიკატორის (ან გასაღების) გამოყენებას, რომელთა სტრუქტურა მიუთითებს კლასიფიკაციას, ადგილმდებარეობას და ა.შ. მაგალითად, იდენტიფიკატორის მნიშვნელობის პირველი ორი ციფრი შეიძლება მიუთითებდეს საწყობის მდებარეობას. ასეთი კოდები ხშირად იცვლება პირობების შეცვლისთანავე, რაც იდენტიფიკატორის მნიშვნელობებს აბათილებს.
4. გასათვალისწინებელია ერთი ატრიბუტიანი სუროგატი იდენტიფიკატორების ჩანაცვლების შესაძლებლობა დიდი კომპოზიტური იდენტიფიკატორით. მაგალითად, ატრიბუტი სახელწოდებით Game Number („თამაშის ნომერი“) შეიძლება გამოყენებულ იქნას არსის ტიპის GAME-სთვის, ნაცვლად მასპინძელი გუნდისა და სტუმრების გუნდის კომბინაციისა.

## ატრიბუტების სახელდება და განძარტება

მონაცემთა არსების სახელდების ზოგადი სახელმძღვანელო მითითებების გარდა, არსებობს ატრიბუტების სახელდების რამდენიმე სპეციალური სახელმძღვანელო, რომლებიც შემდეგნაირად გამოიყურება:

- ატრიბუტის სახელი არის მხოლობითი არსებითი სახელი ან არსებითი ფრაზა (მაგალითად, Customer ID, Age, Product Minimum Price, or Major), რომლებიც წარმოიქმნება როგორც მონაცემთა მნიშვნელობები, და წარმოადგენს არსების ცნებებს ან ფიზიკურ მახასიათებლებს. ცნებები და ფიზიკური მახასიათებლები აღწერილია სახელებით.
- ატრიბუტის სახელი უნდა იყოს უნიკალური. ერთი და იმავე არსის ტიპის ორ ატრიბუტს არ შეიძლება ჰქონდეს ერთი და იგივე სახელი და სასურველია, სიცხადისათვის არსის ყველა ტიპისათვის არ არსებობდეს ორი ატრიბუტი ერთი და იგივე სახელით.
- იმისათვის, რომ ატრიბუტის სახელი გახდეს უნიკალური და უფრო მკაფიო, თითოეული ატრიბუტის სახელი უნდა შეესაბამებოდეს სტანდარტულ ფორმატს. მაგალითად, თქვენს უნივერსიტეტს შეუძლია შექმნას სტუდენტური GPA, სტუდენტის GPA-სგან განსხვავებით, როგორც ატრიბუტების დასახელების სტანდარტული ფორმატის მაგალითი. გამოსაყენებელ ფორმატს დაადგენს თითოეული ორგანიზაცია. ფორმატის წარმოდგენის გავრცელებული სახეა [არსის ტიპის სახელი [[კვალიფიკატორი]]] კლასი, სადაც [...] არასავალდებულო პუნქტია და {...} მიუთითებს იმაზე, რომ წინადადება შეიძლება განმეორდეს. არსის ტიპის სახელი არის იმ არსის (სუბიექტის) სახელი, რომელთანაც დაკავშირებულია ატრიბუტი. არსის ტიპის სახელი შეიძლება გამოყენებულ იქნას ატრიბუტის სახელის განსამარტად. ის თითქმის ყოველთვის გამოიყენება თითოეული არსის იდენტიფიკატორი ატრიბუტისთვის (მაგალითად, Customer ID - მომხმარებლის ID). კლასი არის ფრაზა ორგანიზაციის მიერ განსაზღვრული ფრაზების სიიდან, რომლებიც წარმოადგენს არსის დასაშვებ მახასიათებლებს ან თვისებებს (ან ამ მახასიათებლების აბრევიატურა). მაგალითად, კლასისთვის დასაშვები მნიშვნელობები (და მასთან დაკავშირებული დამტკიცებული აბრევიატურა) შეიძლება იყოს სახელი (Nm), იდენტიფიკატორი (ID), თარიღი (Dt) ან ოდენობა (Amt). ცხადია კლასის არსენობა აუცილებელია. კვალიფიკატორი არის ფრაზა ორგანიზაციის მიერ განსაზღვრული ფრაზების სიიდან, რომლებიც გამოიყენება კლასებზე შეზღუდვების დასადებად. არსის ტიპის თითოეული ატრიბუტის უნიკალურობის უზრუნველსაყოფად შეიძლება საჭირო გახდეს ერთი ან მეტი კვალიფიკატორი. მაგალითად, კლასიფიკატორი შეიძლება იყოს Maximum (Max), Hourly (Hrly), ან State (St). ასევე შეიძლება საჭირო არ იყოს კვალიფიკატორი: Employee Age (თანამშრომელთა ასაკი) და Student Major (სტუდენტის სპეციალობა) ორივე სრულად გამოკვეთილი ატრიბუტის სახელებია. ზოგჯერ შეიძლება აუცილებელი იყოს კვალიფიკატორი. მაგალითად, Employee Birth Date (თანამშრომლის დაბადების თარიღი) და Employee Hire Date (თანამშრომლის დაქირავების თარიღი) არის თანამშრომლის ორი ატრიბუტი, რომლებიც საჭიროებენ ერთ კვალიფიკატორს. ზოგიერთ შემთხვევაში კი ერთზე მეტი კვალიფიკატოსი შეიძლება იყოს საჭირო. მაგალითად, Employee Residence City Name (ან Emp Res Cty Nm) არის თანამშრომლის საცხოვრებელი ქალაქის სახელი, ხოლო Employee Tax City სახელი (ან Emp Tax Cty Nm) არის ქალაქის სახელი, რომელშიც თანამშრომელი რეგისტრირებულია.

- სხვადასხვა არსის ტიპების მსგავსი ატრიბუტები უნდა იყენებდეს ერთნაირ კვალიფიკატორს და კლასებს, მხოლოდ იმ შემთხვევაში, თუ ეს არის ორგანიზაციაში გამოყენებული სახელები. მაგალითად, „ფაკულტეტებისა და სტუდენტების საცხოვრებელი ქალაქი უნდა იყოს, შესაბამისად, ფაკულტეტის რეზიდენციის ქალაქის სახელი და სტუდენტური საცხოვრებელი ქალაქის სახელი“. მსგავსი სახელების გამოყენება მომხმარებლებს უადვილებს იმის გაგებას, რომ ამ ატრიბუტების მნიშვნელობები მოდის იმავე მნიშვნელობთაგან, რასაც დომენებს ვეძახით. მომხმარებლებს შეიძლება სურთ ისარგებლონ მოთხოვნების საერთო დომენებით (მაგ., იპოვონ სტუდენტები, რომლებიც ცხოვრობენ იმავე ქალაქში, როგორც მათი მრჩეველი) და მომხმარებლებისთვის უფრო ადვილი იქნება იმის გაცნობიერება, რომ ასეთი შესატყვისი შეიძლება იყოს, თუ გამოყენებული იქნება იგივე კვალიფიკატორი და კლასის ფრაზები.

გარდა ატრიბუტების სახელდების მითითებებისა ასევე არსებობს ატრიბუტების განმატრების გარკვეული კონკრეტული სახელმძღვანელო მითითებები, რომლებიც შემდეგნაირად გამოიყურება:

- ატრიბუტის განმარტებაში მიეთითება თუ რა ატრიბუტია ეს და რატომ არის მნიშვნელოვანი. განმარტება ხშირად ატრიბუტის სახელის თანმდევია. მაგალითად, სტუდენტის რეზიდენციის ქალაქის სახელი შეიძლება განისაზღვროს, როგორც ”ქალაქის სახელი, რომელიც სტუდენტის მუდმივი საცხოვრებელი ადგილია“.
- ატრიბუტის განმარტებამ მკვეთრად უნდა განისაზღვროს თუ რა არის შეტანილი და რა არ არის შეტანილი მის მნიშვნელობაში; მაგალითად, ”თანამშრომლის ყოველთვიური წელფასის თანხა არის ყოველთვიურად გადახდილი თანხის ოდენობა დასაქმებულის საცხოვრებელი ქვეყნის ვალუტაში, ყოველგვარი შეღავათების, პრემიებისა, ანაზღაურების ან სპეციალური გადასახადების გარდა.“
- ატრიბუტის ნებისმიერი მეტსახელი, ან ალტერნატიული სახელი შეიძლება მითითებული იყოს განმარტებაში ან შეიძლება სხვა ადგილას ატრიბუტის შესახებ დოკუმენტაციაში, რომელიც შესაძლოა ინახებოდეს CASE ინსტრუმენტის საცავში, რომელიც გამოიყენება მონაცემთა განმარტებების შესანარჩუნებლად.
- ასევე შეიძლება სასურველი იყოს განმარტებაში ატრიბუტის მნიშვნელობების წყაროს მითითება. წყაროს მითითებამ შეიძლება უფრო ნათელი გახადოს მონაცემთა მნიშვნელობა. მაგალითად, ”მომხმარებლის სტანდარტული ინდუსტრიული კოდი“ არის მომხმარებლის ბიზნესის ტიპის მითითება.
- ატრიბუტის განმარტებამ (ან სხვა დაზუსტებამ CASE ხელსაწყოს საცავში) ასევე უნდა მიუთითოს, სავალდებულოა ატრიბუტის მნიშვნელობა თუ არა სავალდებულო. ეს ბიზნესი წესი ატრიბუტის შესახებ მნიშვნელოვანია მონაცემთა მთლიანობის შესანარჩუნებლად. განმარტებით, არსის ტიპის იდენტიფიკატორის შესაბამისი ატრიბუტი სავალდებულოა. თუ ატრიბუტის მნიშვნელობა სავალდებულოა, მაშინ ობიექტის ტიპის ეგზემპლარის შესაქმნელად, უნდა იყოს მითითებული ამ ატრიბუტის მნიშვნელობა. აუცილებელი ნიშნავს, რომ არსის ეგზემპლარს ყოველთვის უნდა ჰქონდეს მნიშვნელობა ამ ატრიბუტისთვის, არა მხოლოდ ეგზემპლარის შექმნის დროს. არასავალდებულო ნიშნავს, რომ ამ ატრიბუტის მნიშვნელობა შეიძლება არ არსებობდეს არსის ტიპის ეგზემპლარის შესანახად. ატრიბუტის არასავალდებოლოებისას შესაძლებელია შემდგომში მიეთითოს, უნდა არსებობდეს თუ არა ყოველთვის მონაცემების შეყვანის მერე. მაგალითად, ”თანამშრომელთა დეპარტამენტის იდენტიფიკატორი არის განყოფილების

იდენტიფიკატორი, რომელსზეც მიმაგრებულია თანამშრომელი. თანამშრომელი შეიძლება არ იქნეს მიმაგრებული განყოფილებაზე დაქირავებისას (ამიტომ ეს ატრიბუტი თავდაპირველად არასავალდებულო), მაგრამ მას შემდეგ, რაც თანამშრომელი მიმაგრდება განყოფილებაზე, თანამშრომელი ყოველთვის უნდა დაინიშნოს რომელიმე განყოფილებაში.”

- ატრიბუტის განმარტებამ (ან სხვა დაზუსტებამ CASE ხელსაწყოს საცავში) შეიძლება ასევე მიუთითოს, შეიძლება თუ არ შეიცვალოს ატრიბუტის მნიშვნელობა მას შემდეგ, რაც მიენიჭება მნიშვნელობა და სანამ წაიშლება არსის ეგზემპლარი. ეს ბიზნესის წესი ასევე აკონტროლებს მონაცემთა მთლიანობას. არაიტელექტუალურმა იდენტიფიკატორებმა შეიძლება დროთა განმავლობაში არ შეცვალონ მნიშვნელობები. არსის ეგზემპლარისათვის ახალი არაიტელექტუალური იდენტიფიკატორის მინიჭების მიზნით, ეს ეგზემპლარი ჯერ უნდა წაიშალოს და შემდეგ ხელახლა შეიქმნას.
- მრავალმნიშვნელოვანი ატრიბუტისთვის, ატრიბუტის განსაზღვრებაში უნდა მიეთითოს ატრიბუტის მნიშვნელობის მაქსიმალური და მინიმალური რაოდენობა არსის ეგზემპლარისათვის მაგალითად, ”Employee Skill Name” არის უნარი, რომელსაც თანამშრომელი ფლობს. თითოეული თანამშრომელი უნდა ფლობდეს მინიმუმ ერთ უნარს, ხოლო თანამშრომელს შეუძლია ჩამოთვლოს ყველაზე მეტი 10 უნარი”. ატრიბუტის მრავალმნიშვნელოვანობის მიზეზი შეიძლება იყოს ატრიბუტის „ისტორია“ შენახვის აუცილებლობა. მაგალითად, ”თანამშრომლის ყოველწლიური გაცდენის დღეების რიცხვი არის კალენდარული წლის დღეების რაოდენობა, როდესაც თანამშრომელი არ იმყოფებოდა სამსახურში“. თანამშრომელი განიხილება გამცდენად იმ შემთხვევაში, თუ ის მუშაობს დღის დაგეგმილი საათების 50 პროცენტზე ნაკლებს. ამ ატრიბუტის მნიშვნელობა უნდა იყოს დაცული ყოველი წლის განმავლობაში, როდესაც თანამშრომელი მუშაობს კომპანიაში.”
- ატრიბუტის განმატრებაში შეიძლება მიეთითოს წებისმიერი დამოკიდებულება რომელიც ამ ატრიბუტს აქვს სხვა ატრიბუტებთან. მაგალითად, ”თანამშრომელთა შვებულების დღეების რაოდენობა“ არის თანამშრომლის ანაზღაურებადი შვებულების დღეების რაოდენობა. თუ თანამშრომელის არტიბუტს „თანამშრომლის ტიპი“, აქვს მნიშვნელობა - ”გათავისუფლებული“, მაშინ თანამშრომლის შვებულების დღეების მაქსიმალური მნიშვნელობა განისაზღვრება ფორმულის მიხედვით, რომელიც მოიცავს თანამშრომლის წლების მუშაობას».

## კავშირების (დამოკიდებულებების) მოდელირება

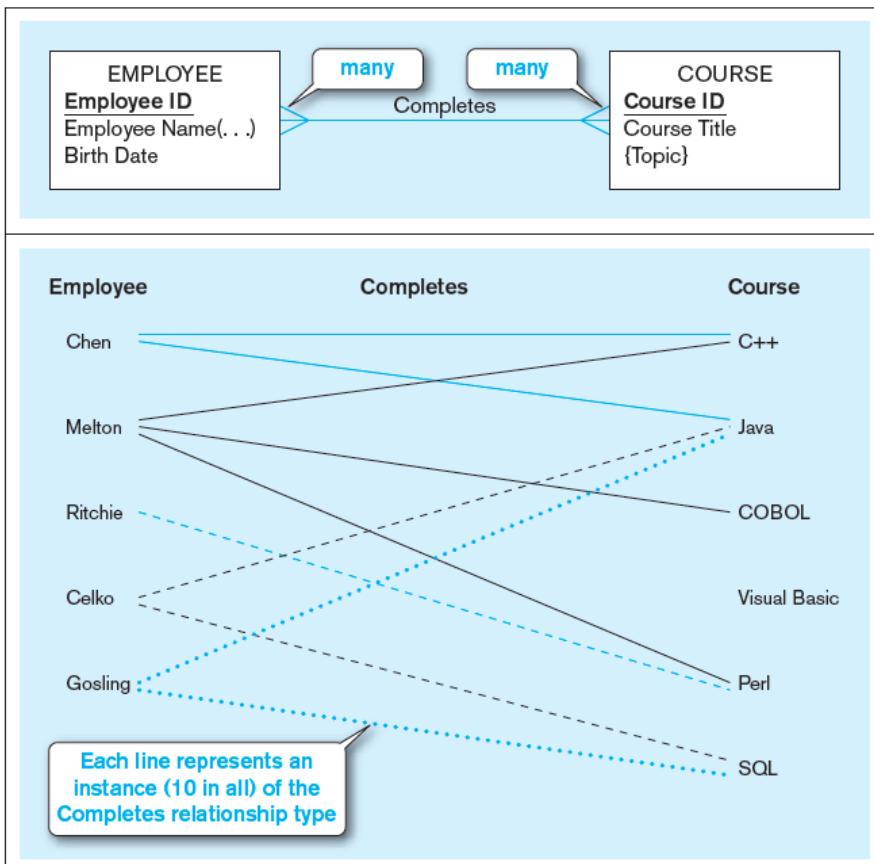
კავშირი (დამკიდებულება) არის წებო, რომელიც აერთიანებს E-R მოდელის სხვადასხვა კომპონენტს. ინტუიციურად, კავშირი არის ურთიერთქმედება ორგანიზაციისთვის საინტერესო ერთი ან მეტი ტიპის არსებს შორის. ამრიგად, კავშირს აქვს ზმნური ფრაზის სახელი. კავშირები და მათი მახასიათებლები (ხარისხი და კარდინალობა) წარმოადგენს ბიზნესის წესებს და, როგორც წესი, კავშირები წარმოადგენს ყველაზე რთულ ბიზნეს წესებს, რომლებიც ასახულია ERD-ში. სხვა სიტყვებით რომ ვთქვათ, სწორედ აյ ხდება მონაცემთა მოდელირება ძალიან საინტერესო და სახალისო, და ასევე გადამწყვეტი მნიშვნელობა აქვს მონაცემთა ბაზის მთლიანობის კონტროლისთვის. კავშირები აუცილებელია მონაცემთა ბაზის თითქმის ყველა მნიშვნელოვანი გამოყენებისათვის; მაგალითად, კავშირები საშუალებას აძლევს iTimes-ს იპოვოს თქვენ მიერ შეძენილი მუსიკა, თქვენს მობილური ტელეფონის კომპანიას იპოვნოს ყველა ტექსტური შეტყობინება თქვენს ერთიან ნაკრებში და ა.შ.

კავშირების უფრო ნათლად გასაგებად, უნდა განვასხვაოთ კავშირის ტიპები და კავშირის ეგზემპლარები. საილუსტრაციოდ, განვიხილოთ არსის ტიპები EMPLOYEE და COURSE, სადაც კურსი წარმოადგენს სასწავლო კურსებს, რომლებიც შეიძლება გაიარონ თანამშრომლებმა. კონკრეტული თანამშრომლების მიერ დასრულებული კურსების დასადგენად, ჩვენ განვისაზღვრავთ კავშირს სახელწოდებით „ასრულებს“, ორი ტიპის არსებს შორის (იხ. ნახ. 2-10 ა). ეს არის კავშირი ბევრი-ბევრთან, რადგან თითოეულ თანამშრომელს შეუძლია დაასრულოს ნებისმიერი რაოდენობის კურსები (არცერთი, ერთი ან მრავალი კურსი), ხოლო მოცემული კურსი შეიძლება დასრულდეს ნებისმიერი რაოდენობის თანამშრომლების მიერ (არავინ არ დაასრულოს, ერთმა თანამშრომელმა, ბევრმა თანამშრომელმა). მაგალითად, ნახ. 2-10 ბ-ში, თანამშრომელმა მელტონმა დაასრულა სამი კურსი (C++, COBOL და Perl). SQL კურსი დაასრულა ორმა თანამშრომელმა (ცელკო და გოსლინგი), ხოლო Visual Basic კურსი არავის დაუმთავრებია.

ამ მაგალითში არსებობს ორი არსის ტიპი EMPLOYEE და COURSE (თანამშრომელი და კურსი), რომლებიც მონაწილეობენ კავშირში „ასრულებს“. ზოგადად, კავშირში შეიძლება მონაწილეობდეს ნებისმიერი რაოდენობა არსის ტიპი (ერთიან მრავალი).

**FIGURE 2-10** Relationship type and instances

(a) Relationship type (Complete)



(b) Relationship instances

კავშირების აღსანიშნად უმეტესწილად გამოიყენება ზმნური ფორმები. იმის გამო, რომ კავშირები ხშირად წარმოიქმნება ორგანიზაციაში მიმდინარე მოვლენის გამო, არსის ეგზემპლარები და კავშირებული არიან ქმედების განხორციელების გამო; ამრიგად, ზმნური ფრაზის შესაბამება ამ კავშირის აღწერის კარგი საშუალებაა. ეს ზმნური ფრაზა უნდა იყოს აწყობი დროში და აღწერითი. კავშირის წარმოსადგენად მრავალი გზა არსებობს. მონაცემთა მოდელის ზოგიერთ შემუშავებელს ურჩევნია კავშირის სახელის ორ ფორმატიანი წარმოდგენა გამიყენოს, თითო სახელი კავშირის თითოეული მიმართულებით. გამოყენებული იქნება ერთი თუ ორი ზმნის ფრაზა, სტრუქტურული მნიშვნელობა იგივეა, ასე რომ შესაძლებელია ორივე ფორმატის გამოყენება, თუ ცხადია კავშირის მნიშვნელობა თითოეული მიმართულებით.

### კავშირის საბაზისო ცნებები და განმარტებები

კავშირის ტიპი არის არსებითი კავშირი არსების ტიპებს შორის. ფრაზა „არსებითი“ გულისხმობს, რომ კავშირი საშუალებას გვაძლევს ვუპასუხოთ შეკითხვებს, რომელთა პასუხიც ვერ მიიღება მხოლოდ არსების ტიპებით. კავშირის ტიპი აღინიშნება ხაზით, რომელზეც დატანილის კავშირის სახელი, როგორც მაგალითზე ნაჩვენებია ნახ. 2-10a, ან ორი სახელით, როგორც ნახ. 2-1. კავშირების სახელდებისათვის უმჯობესია მოკლე, მომხმარებლისთვის მნიშვნელოვანი აღწერითი ზმნური ფრაზის გამოყენება.

კავშირის ეგზემპლარი არის ურთიერთდაამოკიდებულება არსის ეგზემპლარებს შორის, სადაც კავშირის თითოეული ეგზემპლარი ასოცირდება მასში მონაწილე თითოეული ტიპის არსიდან ზუსტად ერთი არსის ეგზემპლართან. მაგალითად, ნახ. 2-10b-ზე მოცემული 10 სტრიქონიდან თითოეული წარმოადგენს კავშირის ეგზემპლარს ერთ თანამშრომელს და ერთ კურსს შორის, რაც მიუთითებს, რომ თანამშრომელმა დაასრულა ეს კურსი. მაგალითად, თანამშრომელი რიჩისა და კურსის Perl შორის ერთი კავშირია.

**TABLE 2-2 Instances Showing Date Completed**

Employee Name	Course Title	Date Completed
Chen	C++	06/2014
Chen	Java	09/2014
Melton	C++	06/2014
Melton	COBOL	02/2015
Melton	SQL	03/2014
Ritchie	Perl	11/2014
Celko	Java	03/2014
Celko	SQL	03/2015
Gosling	Java	09/2014
Gosling	Perl	06/2014

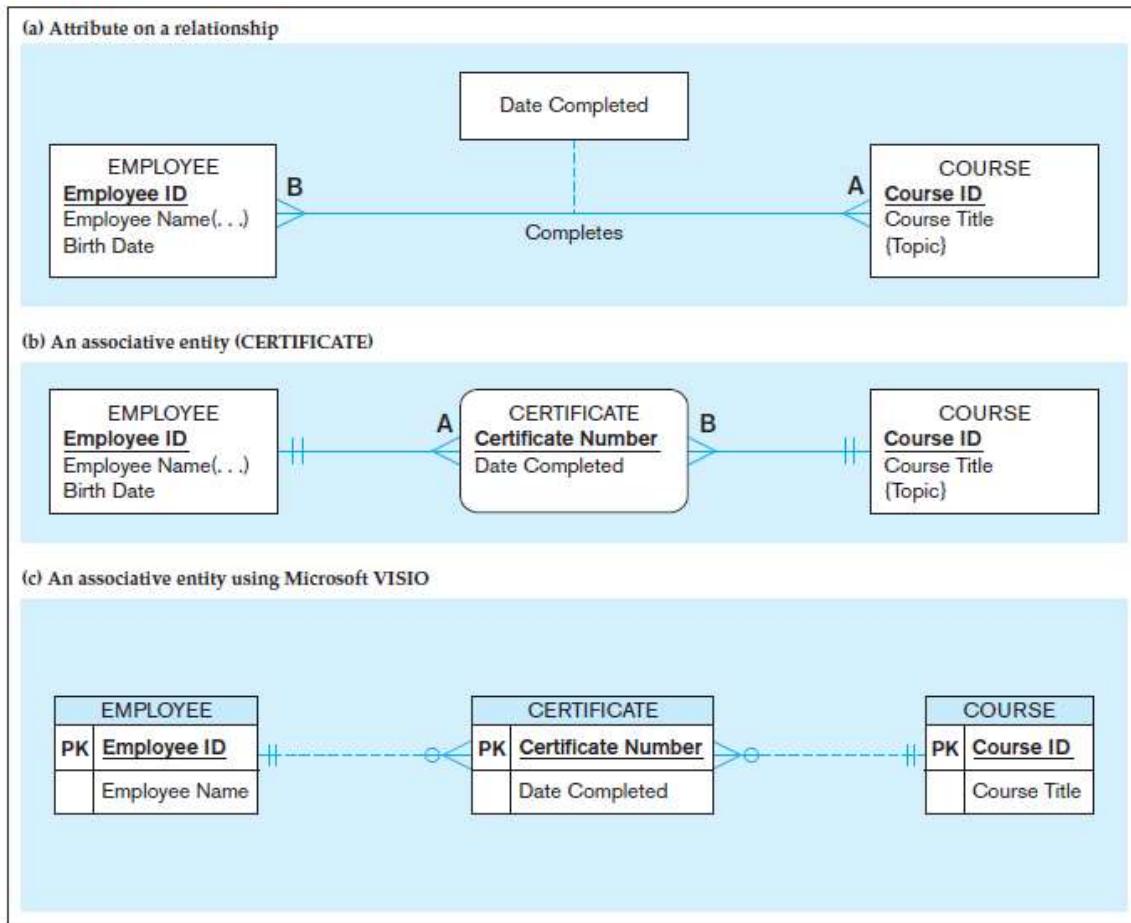
### კავშირის ატრიბუტები

მოყვანილი მაგალითიდან აშკარაა, რომ არსებს აქვთ ატრიბუტები, ატრიბუტები შეიძლება ასოცირდებოდეს „ბევრთან-ბევრთან“ (ან „ერთი ერთანაც“) კავშირებთანაც. მაგალითად, დავუშვათ, რომ ორგანიზაციას სურს ჩაწეროს თარიღი (თვე და წელი), როდესაც თანამშრომელი დაასრულებს თითოეულ კურსს. ამ ატრიბუტს ეწოდა დასრულების თარიღი. მაგალითისთვის იხილეთ ცხრილი 2-2.

სად უნდა განთავსდეს ატრიბუტი დასრულებული თარიღი E-R დიაგრამაზე? ნახ 2-10 ა-დან ჩანს, რომ დასრულების თარიღი არ ასოცირდება არც დასაქმებულთან და არც კურსთან. ეს იმიტომ, რომ დასრულების თარიღი უფრო კავშირის საკუთრებაა, ვიდრე რომელიმე არსის. სხვა სიტყვებით რომ ვთქვათ, ასრულებს კავშირის თითოეული ეგზემპლარისათვის არსებობს მნიშვნელობა დასრულების თარიღისთვის. ერთ-ერთი ასეთი შემთხვევა (მაგალითად) აჩვენებს, რომ თანამშრომელმა სახელად მელტონმა დაასრულა კურსი სახელწოდებით C ++ 06/2014 წელს.

ERD-ის დიაგრამაზე (ნახ. 2-11 ა) ატრიბუტი Date Completed არის მართვულთხედში ჩასმული, რომელიც დაკავშირებულია Completes კავშირთან ხაზთან. საჭიროების შემთხვევაში, ამ კავშირს შეიძლება დაემატოს სხვა ატრიბუტები, როგორიცაა კურსის შეფასება, ინსტრუქტორი და ოთახის ადგილმდებარეობა.

**FIGURE 2-11** An associative entity



### ასოციაციური არსი

კავშირში ერთი ან მეტი ატრიბუტის არსებობა დამპროექტებელს მიანიშნებს, რომ კავშირი უნდა წარმოადგენდეს, როგორც არსის ტიპი. ამ მომენტის ხაზგასასმელად, E-R დიაგრამის შექმნის ინსტრუმენტების უმეტესობაში მოითხივება, რომ ასეთი ატრიბუტები განთავსდეს არსის ტიპში. **ასოციაციური არსი** არსის არსის ტიპის სახეობა, რომელიც ასოცირდება ერთი ან მეტი არსის ტიპის ეგზემპლარებთან და შეიცავს ატრიბუტებს, რომლებიც დამახასიათებელია ამ ერთეულთა ეგზემპლარებს შორის კავშირისათვის. **ასოციაციური არსი** CERTIFICATE წარმოდგენილია მომრგვალებული კუთხეებიანი მართვულებით, როგორც ეს ნაჩვენებია ნახ.

2-11 ბ-ზე. E-R დიაგრამის შექმნის ინსტრუმენტების უმეტესობას არ გააჩნია სპეციალური სიმბოლო ასოციაციური არსებისათვის. ასოციაციურ არსებს ზოგჯერ გერუნდებითაც (ფორმა, რომელიც ზმნისგან არის მიღებული, მაგრამ მოქმედებს როგორც არსებითი სახელი, ინგლისურად მთავრდება -ing – ით) აღინიშნება, რადგან კავშირის სახელი (ზმნა) ჩვეულებრივ გარდაიქმნება არსის სახელად, რომელიც არსებითი სახელია. მივაქციოთ ყურადღება, რომ ნახ. 2-11 ბ-ში, რომ ასოციაციურ არს და ძლიერ არს შორის კავშირებს სახელები არ გააჩნიათ. ეს იმიტომ, რომ ასოციაციური არსი წარმოადგენს კავშირს. ნახ. 2-11 გ გვიჩვენებს, თუ როგორ გამოისახება ასოციაციური არსები Microsoft Visio- ს გამოყენებით. ასოციაციური არსის მსგავსი გამოსახვა ხდება E-R დიაგრამის შექმნის ინსტრუმენტების უმეტესობით. Visio- ში

კავშირის ხაზები წყვეტილია, რადგან CERTIFICATE იდენტიფიკატორში არ შეიცავს დაკავშირებული არსების იდენტიფიკატორებს (სერთიფიკატის ნომერი საკმარისია ).

როგორ მიუთითოთ იმაზე, რომ კავშირის გარდაიქმნას ასოციაციური არსის ტიპად?

ამისათვის უნდა არსებობდეს ოთხი პირობა:

1. კავშირში მონაწილე ყველა არსის ტიპების კავშირი უნდა იყოს "მრავალი".
2. შედეგად მიღებული ასოციაციური ტიპის არსს საბოლოო მომხმარებლებისათვის აქვს დამოუკიდებელი მნიშვნელობა და, სასურველია, მისი იდენტიფიკაცია მოხდეს ერთი ატრიბუტიანი იდენტიფიკატორით.
3. ასოციაციურ არსს იდენტიფიკატორის გარდა აქვს ერთი ან მეტი ატრიბუტი.
4. ასოციაციური არსი მონაწილეობს ერთ ან რამდენიმე კავშირში, დამოუკიდებლად კავშირში მონაწილე არსებისაგან.

ნახ. 2-11 ბ გვიჩვენებს კავშირს Completes („ასრულებს“), რომელიც ასოციაციური არსის ტიპად არის გადაკეთებული. ამ შემთხვევაში კომპანიის სასწავლო დეპარტამენტმა გადაწყვიტა გადასცეს სერთიფიკატი თითოეულ თანამშრომელს, რომელიც კურსს გაივლის. ამრიგად, სუბიექტს CERTIFICATE ეწოდება, რომელსაც, რა თქმა უნდა, დამოუკიდებელი მნიშვნელობა აქვს საბოლოო მომხმარებლებისთვის. ასევე, თითოეულ სერთიფიკატს აქვს ნომერი (Certificate Number - სერთიფიკატის ნომერი), რომელიც იდენტიფიკატორის ფუნქციას ასრულებს. მასში ასევე შედის ატრიბუტი Date Completed (დასრულების თარიღი). მივაჟციოთ ყურადღება ნახ. 2-11 ბ და ნახ. 2-11 გ Visio ვერსიას, სადაც ორივე არსის EMPLOYEE (თანამშრომელი) და COURSE (კურსი) მონაწილეობა სავალდებულოა CERTIFICATE-სთან ორივე კავშირში. ეს არის ზუსტად ის, რაც ხდება მაშინ, როდესაც უნდა წარმოვადგინოთ ბევრი-ბევრთან კავშირი (ნახ.ზე 2-11 ა), როგორც ორი ერთი-ბევრთან კავშირი (პირობა, რომელიც დაკავშირებულია CERTIFICATE- თან ნახაზებში 2-11b და 2-11c).

უნდა გავითვალისწინოთ, რომ კავშირის ასოციაციურ არსად გადაკეთებამ გამოიწვია კავშირის სახელის შეცვლა. ანუ კარდინალურობა "მრავალი" ახლა მთავრდება ასოციაციურ არსზე და არა თითოეულ მონაწილე არსის ტიპზე. ნახაზზე 2-11, ეს გვიჩვენებს, რომ თანამშრომელს, რომელსაც შეუძლია დაასრულოს ერთი ან მეტი კურსი (A აღნიშვნა ნახაზზე 2-11 ა), შეიძლება მიენიჭოს ერთზე მეტი სერთიფიკატი (აღნიშვნა A ნახაზზე 2-11 ბ); და რომ კურს, რომელიც შეიძლება ერთმა ან მეტმა თანამშრომელმა დაასრულოს (აღნიშვნა B დიაგრამა 2-11 ა), შეიძლება ჰქონდეს მრავალი სერთიფიკატს მინიჭებული (აღნიშვნა B, სურათი 2-11 ბ).

## კავშირის ხარისხი

კავშირის ხარისხი არის არსების ტიპების რაოდენობა, რომლებიც მონაწილეობენ ამ კავშირში ამრიგად, ნახ. 2-11-ში კავშირი Completes (ასრულებს) არის მე -2 ხარისხის, რადგან არსებობს ორი არსის ტიპი: EMPLOYEE და COURSE. სამი ყველაზე გავრცელებული კავშირის ხარისხი E-R მოდელებში არის უნიარული (ხარისხი 1), ბინარული (ხარისხი 2) და სამეული (ხარისხი 3). უფრო მაღალი ხარისხის კავშირები შესაძლებელია, მაგრამ ისინი პრაქტიკაში იშვიათად გვხვდება, ამიტომ შემოვიფარგლოთ ამ სამი შემთხვევით. უნიარული, ბინარული და სამეული კავშირების მაგალითები მოცემულია ნახაზზე 2-12. (ზოგიერთ ფიგურაში ატრიბუტები არ ჩანს სიმარტივისათვის).

2-12 ნახ.-დან შეგვიძლია დავასკვნათ, რომ ნებისმიერი კონკრეტული მონაცემთა მოდელი წარმოადგენს კონკრეტულ სიტუაციას და არა განზოგადებას. მაგალითად, განვიხილოთ ურთიერთობა Manages (მართვა) ნახაზზე 2-12 ა. ზოგიერთ ორგანიზაციაში შესაძლებელია ერთმა თანამშრომელმა მართოს მრავალი სხვა თანამშრომელი (მაგ., მატრიცული ორგანიზაციით). E-R მოდელის შემუშავებისას მნიშვნელოვანია, რომ გვესმოდეს კონკრეტული ორგანიზაციის ბიზნესის წესები, რომელის მოდელირებასაც ვახდენთ.

### უნარული კავშირი

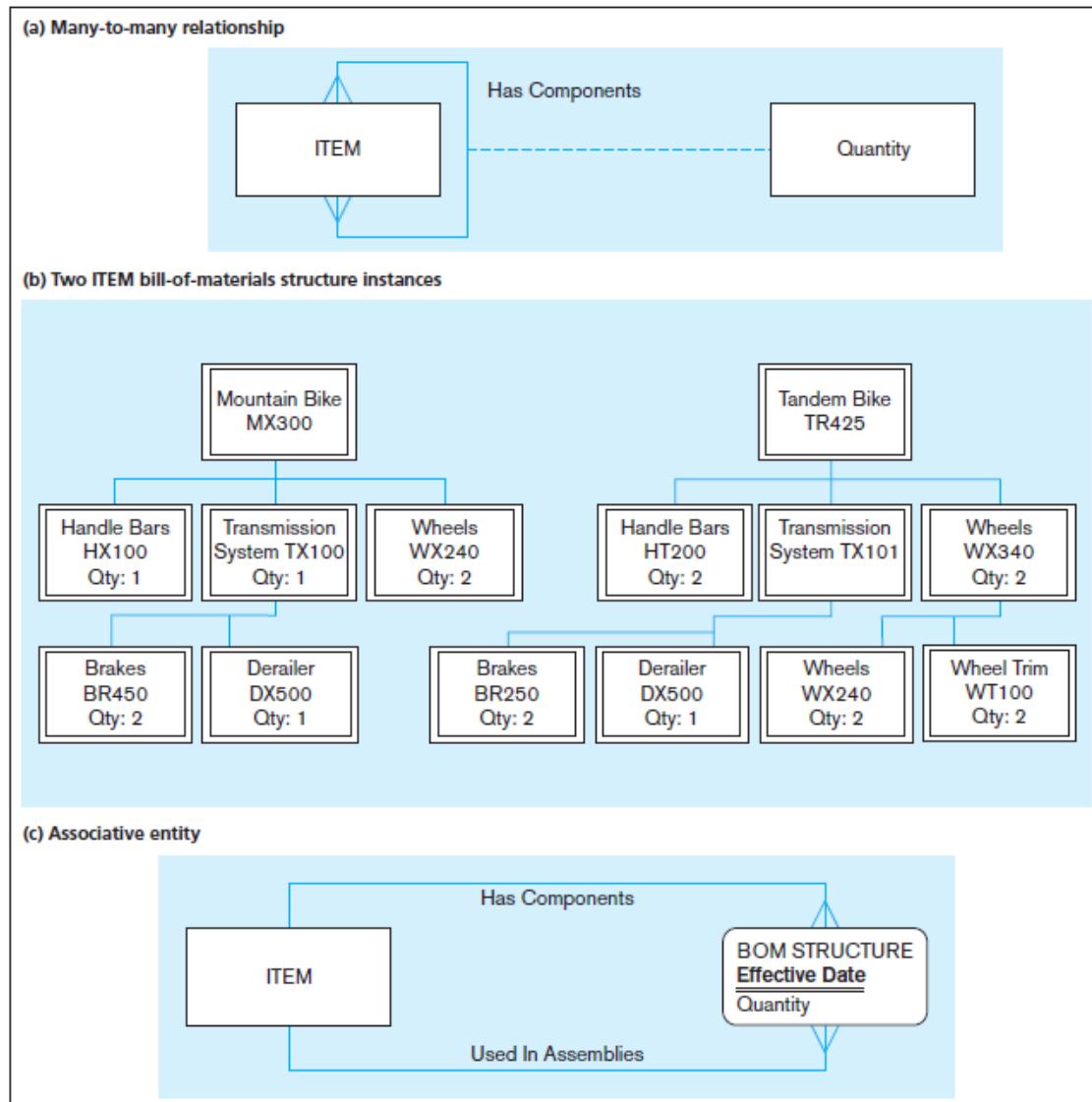
უნიარული კავშირი არის კავშირი ერთი არსის ტიპის ეგზემპლარებ შორის (უნარულ კავშირებს რეკურსიულ კავშირებს უწოდებენ.) ნახაზზე 2-12 ა. სამი მაგალითია ნაჩვენები - პირველ მაგალითში, Is Married To ნაჩვენებია როგორც ინდივიდუალური ურთიერთობა PERSON არსის ტიპის კავშირებს შორის. იმის გამო, რომ ეს არის ერთი-ერთთან კავშირი, აღნიშვნაში მითითებულია, რომ საჭიროა მხოლოდ ახლანდელი ქორწინების არსებობა, თუ ის არსებობს. რა შეიცვლებოდა, თუ თითოეული ადამიანისთვის ქორწინების ისტორიის შენახვა დაგვჭირდებოდა? მეორე მაგალითში, Manages (მართვა) ნაჩვენებია როგორც ერთ – ერთან კავშირი EMPLOYEE არსის ტიპის ეგზემპლარებს შორის. ამ კავშირის გამოყენებით ჩვენ შეგვიძლია დავადგინოთ, მაგალითად, თანამშრომლები, რომლებიც ექვემდებარებიან კონკრეტულ მენეჯერს. მესამე მაგალითია უნარული ურთიერთობის გამოყენების ერთი შემთხვევა, რომელიც წარმოადგენს თანმიმდევრობას, ციკლს ან პრიორიტეტულ სიას. ამ მაგალითში, სპორტული გუნდები დაკავშირებულია მათ პოზიციასთან ლიგაში (კავშირი Stands After) (შენიშვნა: ამ მაგალითებში უგულველყოფილია არის თუ არა სავალდებულო ეს კავშირები ან შეიძლება თუ არა იგივე არსის ეგზემპლარების გამეორება იმავე კავშირის ეგზემპლარებში).

ნახ. 2-13 გვიჩვენებს სხვა უნიარული კავშირების მაგალითს, რომელსაც უწოდებენ *bill-of materials structure* (მასალების უწყისების სტრუქტურა). ბევრი წარმოებული პროდუქტი მზადდება ნაკრებისაგან, რომლებიც, თავის მხრივ, შედგება ქვენაკრებებისა და ნაწილებისგან და ა.შ. როგორც ნაჩვენებია ნახაზზე 2-13 ა, ჩვენ შეგვიძლია წარმოვადგინოთ ეს სტრუქტურა, როგორც მრავალჯერადი უნარული კავშირი. ამ ნახატში არსის ტიპი ITEM გამოიყენება ყველა ტიპის კომპონენტის წარმოსაჩენად, ხოლო კავშირის ტიპის დასახელებისათვის ვიყენებთ Has Components- ს, რომელიც უფრო დაბალი დონის საგნებს უკავშირებს ქვედა დონის საგნებს.

ამ სტრუქტურის მასალების ორი შემთხვევა ნაჩვენებია ნახაზზე 2-13 ბ. თითოეული ეს დიაგრამა გვიჩვენებს თითოეული ნივთის უშუალო კომპონენტებს, აგრეთვე ამ კომპონენტის რაოდენობებს. მაგალითად, TX100 ელემენტი შედგება BR450 პუნქტისგან (რაოდენობა 2) და DX500 ელემენტისგან (რაოდენობა 1). მარტივი შესამოწმებელია, რომ კავშირები სინამდვილეში „ბევრი-ბევრთანაა“. ზოგირთ ერთეულს აქვს ერთზე მეტი ტიპის კომპონენტი (მაგალითად, ელემენტს MX300 აქვს სამი უშუალო ტიპს კომპონენტი: HX100, TX100 და WX240). ასევე, ზოგიერთი კომპონენტი გამოიყენება რამდენიმე უფრო მარალი დონის ნაკრებებში. მაგალითად, WX240 ელემენტი გამოიყენება როგორც MX300, ასევე WX340 ელემენტებში.

ატრიბუტის Quantity (რაოდენობის) არსებობა კავშირში მიაწინებს იმაზე, რომ ანალიტიკოსი განიხილავს კავშირის კომპონენტების ასოციაციურ არსად გადაკეთებას. დიაგრამა 2-13 გ ნაჩვენებია არსის ტიპი BOM STRUCTURE, რომელიც ქმნის კავშირს ITEM არსის ტიპის ეფექტუარებს შორის. BOM STRUCTURE-ს დაემატა მეორე ატრიბუტი (სახელწოდებით Effective Date), თარიღის ჩასაწერად, როდესაც ეს კომპონენტი პირველად იქნა გამოყენებული შესაბამის ნაკრებში. ეს თარიღი ხშირად საჭიროა, როდესაც საჭიროა ძალაში შესვლის ისტორიის დაფიქსირება. მონაცემთა მოდელის სხვა სტრუქტურები შეიძლება გამოყენებულ იქნას უნარული კავშირებისთვის, ასეთ იერარქიების ჩათვლით.

**FIGURE 2-13 Representing a bill-of-materials structure**



## **ბინარული კავშირები**

ბინარული კავშირი არის კავშირი ორი არსის ტიპების ეგზემპლარებს შორის და არის კავშირის ყველაზე გავრცელებული ტიპი, რომელიც გვხვდება მონაცემთა მოდელირებაში. სურათი 2-12 ბ აჩვენებს სამ მაგალითს. პირველი (ერთი-ერთთან) მიუთითებს იმაზე, რომ თანამშრომელს ეძლევა ერთი პარკირების ადგილი, და რომ თითოეულ პარკინგის ადგილი ეკუთვნის ერთ თანამშრომელს. მეორე (ერთი - ბევრთან) მიუთითებს იმაზე, რომ პროდუქტის ხაზი შეიძლება შეიცავდეს რამდენიმე პროდუქტს და რომ თითოეული პროდუქტი მხოლოდ ერთ პროდუქტის ხაზს ეკუთვნის. მესამე (ბევრი-ბევრთან) გვიჩვენებს, რომ სტუდენტს შეუძლია დარეგისტრირდეს ერთზე მეტ საგანზე და თითოეულ საგანზე შეიძლება დარეგისტრირდეს მრავალი სტუდენტი.

## **სამეული კავშირი**

სამეული კავშირი არის ერთდროული კავშირი სამი არსის ტიპის ეგზემპლარებს შორის. ტიპიური ბიზნეს სიტუაცია, რომელიც იწვევს სამეულ ურთიერთობას, ნაჩვენებია ნახაზზე 2-12 გ. ამ მაგალითში გამყიდველებს შეუძლიათ სხვადასხვა ნაწილების მიწოდება საწყობებში. კავშირი Supplies (მიწოდება) გამოიყენება კონკრეტული ნაწილების ჩასაწერად, რომლებიც მოცემულმა გამყიდველმა მიაწოდა კონკრეტულ საწყობას. ამრიგად, არსებობს სამი არსის ტიპი: VENDOR, PART და WAREHOUSE. კავშირში Supplies ორი ატრიბუტია: Shipping Mode (ტრანსპორტირების რეჟიმი) და Unit Cost (ერთეულის ღირებულება). მაგალითად, Supplies ერთმა შემთხვევამ შეიძლება დააფიქსიროს ის ფაქტი, რომ გამყიდველი X-ს შეუძლია გაგზავნოს C ნაწილი Y საწყობში, რომ გადაზიდვის რეჟიმი არის მეორე დღის საპარო გადაზიდვა და ღირებულებაა 5 \$ ერთეულზე.

აუცილებლად უნდა განვასხვავოთ სამეული კავშირი სამი ბინარული კავშირისაგან მაგალითად, Unit Cost (ერთეულის ღირებულება) არის Supplies (მომარაგება) კავშირის ატრიბუტი ნახაზზე 2-12 გ. Unit Cost (ერთეულის ღირებულება) ვერ იქნება სათანადოდ აღქმული რომელიმე შესაძლო ბინარული კავშირებიდან რომელიმეში სამი არსის ტიპებს შორის, მაგალითად, PART (ნაწილი) და WAREHOUSE (საწყობი) შორის. ამრიგად, მაგალითად, თუ გვეტყოდნენ, რომ X გამყიდველს შეუძლია C ნაწილის გაგზავნა ერთეულის ღირებულებით 8 დოლარი, ეს მონაცემები არასრული იქნება, რადგან ისინი არ მიუთითებენ, თუ რომელ საწყობში გაიგზავნება ნაწილები.

როგორც ყოველთვის, ატრიბუტის არსებობა 2-12 გ ნახაზზე მოცემულ კავშირში Supplies მიუთითებს კავშირის ასოცირების არსის ტიპის შექმნაზე. ნახ. 2-14 გვიჩვენებს სამეული კავშირის ალტერნატიულ (და სასურველია) სურათს 2-12 გ. დიაგრამა 2-14-ში გამოიყენება (ასოციაციური) არსის ტიპით SUPPLY SCHEDULE, შეცვლაზე Supplies კავშირის ნაცვლად ნახაზი 2-12c- დან. ცხადია, რომ არსის ტიპის SUPPLY SCHEDULE დამოუკიდებელ ინტერესს წარმოადგენს მომხმარებლისთვის. ამასთან გასათვალისწინებელია, რომ იდენტიფიკატორი ჯერ კიდევ არ არის მინიჭებული SUPPLY SCHEDULE-ში. ეს მისაღებია. თუ E-R მოდელირების დროს ასოცირებულ არსს არ მიენიჭა იდენტიფიკატორი, იდენტიფიკატორი მიენიჭება (ან გასაღები) ლოგიკური მოდელირებისას. ეს იქნება კომპოზიტური იდენტიფიკატორი, რომლის კომპონენტები შედგება იდენტიფიკატორისგან თითოეული არსის ტიპისთვის (ამ მაგალითში, PART, VENDOR და WAREHOUSE).

როგორც ადრე აღვნიშნეთ, ჩვენ არ ვახდენთ სტრიქონების მარკირებას SUPPLY SCHEDULE- ის სამ არსზე. ეს იმიტომ ხდება, რომ ეს ხაზები არ წარმოადგენ ბინარულ კავშირს. იმისთვის,

რომ შევინარჩუნოთ იგივე მნიშვნელობა, რაც ფიგურა 2-12 გ-ის სამეული კავშირისას, ჩვენ არ შეგვიძლია დავყოთ Supplies (მიწოდება) კავშირი სამ ბინარულ კავშირად.

არსებობს გარკვეული სახელმძღვანელო მითითება სამეული (და უფრო მარალი ხარისხის) კავშირებისთვის: ყველა სამეულის (ან უფრო მაღალი) კავშირის გადაყვანა ასოციაციურ არსებად (როგორც ამ მაგალითშია). ბევრი ავტორი მიიჩნევს, რომ მონაწილეობის შეზღუდვები (აღწერილი კარდინალურობის შეზღუდვების თვალსაზრისით) არ შეიძლება ზუსტად წარმოდგენილ იქნას სამეული კავშირებისთვის, ატრიბუტებით კავშირის ხაზზე ნიშნების გათვალისწინებით. ამასთან, ასოციაციურ არსად გადაქცევით შეიძლება შეზღუდვების ზუსტად წარმოდგენა. ასევე, E-R დიაგრამის ხატვის მრავალი ინსტრუმენტი, მათ შორის CASE ინსტრუმენტების უმეტესობა, ვერ ახდენენ სამეული კავშირების წარმოადგენას. ასე რომ, მართალია ეს ინსტრუმენტები სემანტიკურად ზუსტი არ არიან, უნდა გამოვიყენოთ ეს ინსტრუმენტები ასოციაციურ ერთეულთან სამეული ან უფრო მაღალი რიგის კავშირების წარმოსადგენად.

### კარდინალურობის შეზღუდვა

მონაცემთა მოდელირებაში არსებობს კიდევ ერთი მნიშვნელოვანი აღნიშვნა საერთო და მნიშვნელოვან ბიზნეს წესების წარმოსადგენად. დავუშვათ, არსებობს ორი არსის ტიპი, A და B, რომლებიც შეერთებულია კავშირით. კარდინალურობის შეზღუდვა განსაზღვრავს B არსის იმ ეგზემპლართა რაოდენობას, რომლებიც შეიძლება (ან უნდა) უკავშირდებოდნენ A არსის თითოეულ ეგზემპლარს. მაგალითად, განვიხილოთ ვიდეო მაღაზია, რომელიც აქირავებს ფილმების DVD-ს. იმის გამო, რომ მაღაზიაში შეიძლება იყოს ერთზე მეტი DVD თითოეული ფილმისთვის, ეს ინტუიციურად არის კავშირი „ერთი-ბევრთან“ როგორც ეს ნაჩვენებია ნახაზზე 2-16a. ასევე შესაძლებელია, რომ მაღაზიაში არ იყოს მოცემული დროს მოცემული ფილმის რაიმე DVD დისკი (მაგ., ყველა ასლი გაქირავებულია). ჩვენ გვჭირდება უფრო ზუსტი აღნიშვნა, რომ მიუთითოს კარდინალურობის დიაპაზონი კავშირისათვის.

### მინიმალური კარდინალურობა

კავშირის მინიმალური კარდინალურობა - ეს არის B არსის ეგზემპლარების მინიმალური რაოდენობა, რომელიც შეიძლება უკავშირდებოდეს A არსის თითოეულ ეგზემპლარს. ჩვენს DVD მაგალითში ფილმის DVD-ების მინიმალური რაოდენობა ნულოვანია. როდესაც მონაწილეთა მინიმალური რაოდენობა ნულოვანია, ჩვენ ვამბობთ, რომ არსის ტიპი B არჩევითად მონაწილეობს კავშირში. ამ მაგალითში DVD (სუსტი არსის ტიპი) არის Is Stocked As კავშირი სურვილისამებრ. ეს ფაქტი მითითებულია ნულოვანი სიმბოლოთი DVD არსის მახლობლად ხაზის გასწროვ, ნახაზზე 2-16b.

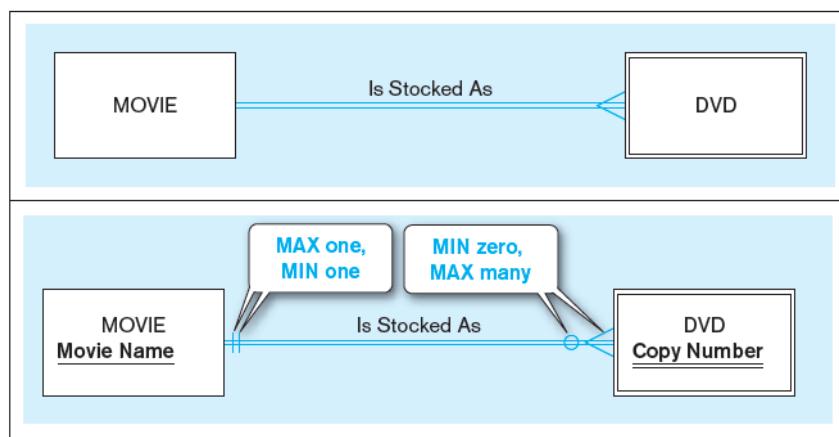
### მაქსიმალური კარდინალურობა

კავშირის მაქსიმალური კარდინალურობა - ეს არის B არსის ეგზემპლარების მაქსიმალური რაოდენობა, რომელიც შეიძლება უკავშირდებოდეს A არსის თითოეულ ეგზემპლარს. ვიდეოს მაგალითში, DVD არსის ტიპის მაქსიმალური კარდინალურობა არის "მრავალი" - ეს არის ერთზე მეტი დაუზუსტებელი რიცხვი. ამაზე მიუთითებს "ყორნის ფეხის" სიმბოლო DVD არსის სიმბოლოს გვერდით ხაზზე 2-16 ბ.

კავშირი რა თქმა უნდა, ორმხრივია, ამიტომ MOVIE (ფილმი) გვერდითაც არის კარდინალურობის აღნიშვნა. მივაქციოთ ყურადრება, რომ მინიმუმი და მაქსიმუმი ორივე ერთის ტოლია (იხ. სურათი 2-16 ბ). ამას კარდინალურობის სავალდებულობის ერთეულს

უწოდებენ. სხვა სიტყვებით რომ ვთქვათ, ფილმის თითოეული DVD-ზე უნდა იყოს ზუსტად ერთი ფილმის ასლი. ზოგადად, კავშირში მონაწილეობა შეიძლება იყოს არჩევითი ან სავალდებულო მონაწილე არსებისათვის. თუ მინიმალური კარდინალობა ნულოვანია, მონაწილეობა არჩევითია; თუ მინიმალური კარდინალურობა ერთია, მონაწილეობა სავალდებულოა.

**FIGURE 2-16** Introducing cardinality constraints  
(a) Basic relationship



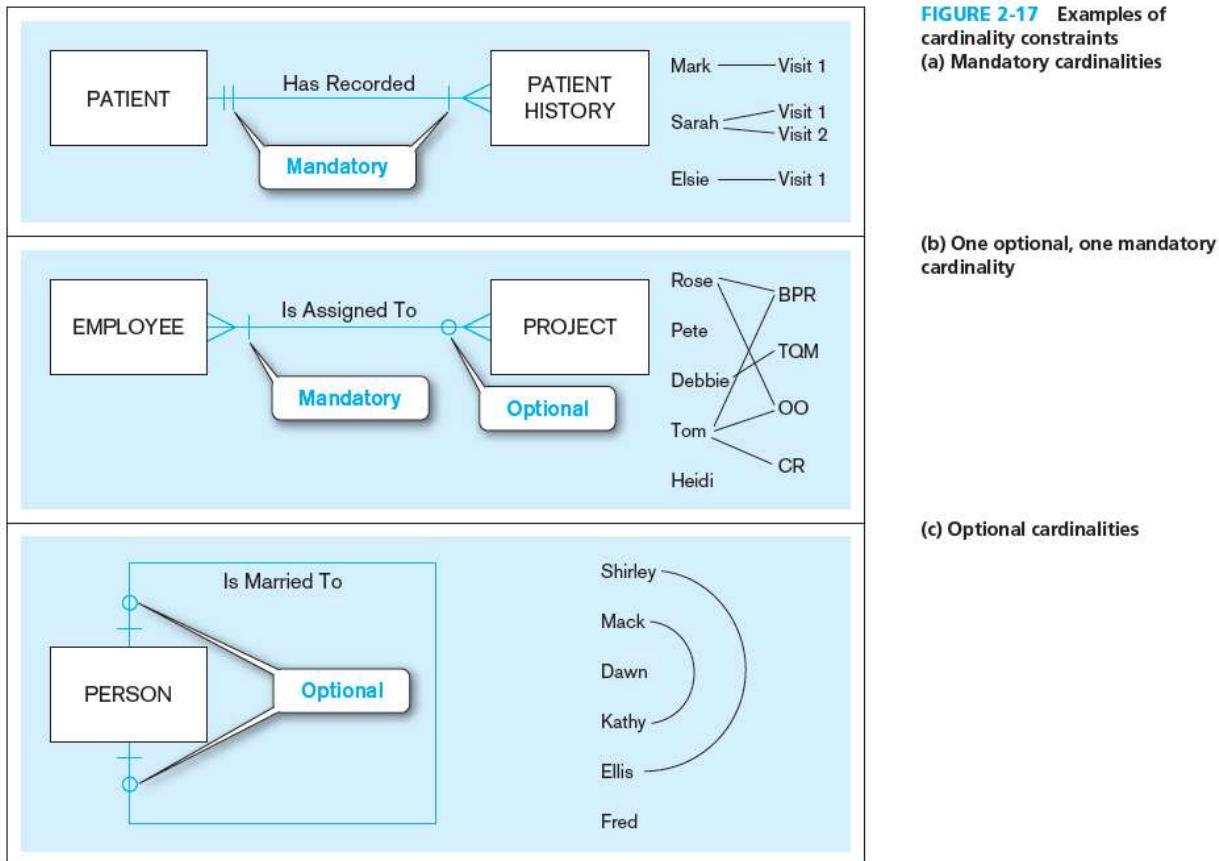
დიაგრამა 2-16 ბ-ზე არსის თითოეულ ტიპს დაემატა ზოგიერთი ატრიბუტი. გავითვალისწინოთ, რომ DVD წარმოდგენილია როგორც სუსტი არსი. იმიტომ, რომ DVD არ შეიძლება არსებობდეს, თუ ფილმი არ არსებობს. MOVIE-ს იდენტიფიკატორი არის ფილმის სახელი. DVD-ს არ აქვს უნიკალური იდენტიფიკატორი. ამასთან, ასლის ნომერი არის ნაწილობრივი იდენტიფიკატორი, რომელიც ფილმის სახელთან ერთად ცალსახად განსაზღვრავს DVD-ს ეგზემპლარს.

მოვიყვანოთ რამდენიმე მაგალითი კარდინალურობის უფრო ნათლად წარმოსადგენად.

ნახაზზე 2-17 მოცემული სამი კავშირის მაგალითი, აჩვენებს მინიმალური და მაქსიმალური კარდინალურობის ყველა შესაძლო კომბინაციას. თითოეულ მაგალითში აღნიშნულია ბიზნესის წესი თითოეული კარდინალურობის შეზღუდვისთვის და ნაჩვენებია E-R აღნიშვნა. თითოეულ მაგალითში ასევე ნაჩვენებია კავშირის ზოგიერთი მაგალითი კავშირის ხასიათის გასარკვევად. ქვემოთ მოცემულია ბიზნესის წესები ნახაზზე 2-17 ნაჩვენები თითოეული მაგალითისთვის:

- პაციენტი (PATIENT) ჩაიწერა (Has Recorded) პაციენტის ისტორია (PATIENT HISTORY) (სურათი 2-17 ა) თითოეულ პაციენტს აქვს ერთი ან მეტი პაციენტის ისტორია (პაციენტის საწყისი ვიზიტი ყოველთვის აღირიცხება როგორც PATIENT HISTORY ეგზემპლარი). PATIENT HISTORY თითოეული ეგზემპლარი "ეკუთვნის" ზუსტად ერთ PATIENT (პაციენტი).
- თანამშრომელი (EMPLOYEE) ინიშნება (Is Assigned To) პროექტზე (PROJECT) (სურათი 2-17 ბ) თითოეულ პროექტზე ინიშნება მინიმუმ ერთი თანამშრომელი მაინც (ზოგიერთ პროექტზე ერთზე მეტი). თითოეული თანამშრომელი შეიძლება ან (სურვილისამებრ) არ დაინიშნოს რომელიმე არსებულ პროექტზე (მაგალითად, თანამშრომელი პიტი) ან შეიძლება დაინიშნოს ერთ ან მეტ პროექტზე.

3. პიროვნება (PERSON) დაქორწინებულია (Is Married To) პიროვნებაზე (PERSON) (სურათი 2-17 გ) ეს არასავალდებულო კარდინალურობაა ორივე მიმართულებით - ნული ან ერთი, რადგან ადამიანი შეიძლება მოცემულ მომენტში იყოს დაქორწინებული ან არ იყოს.



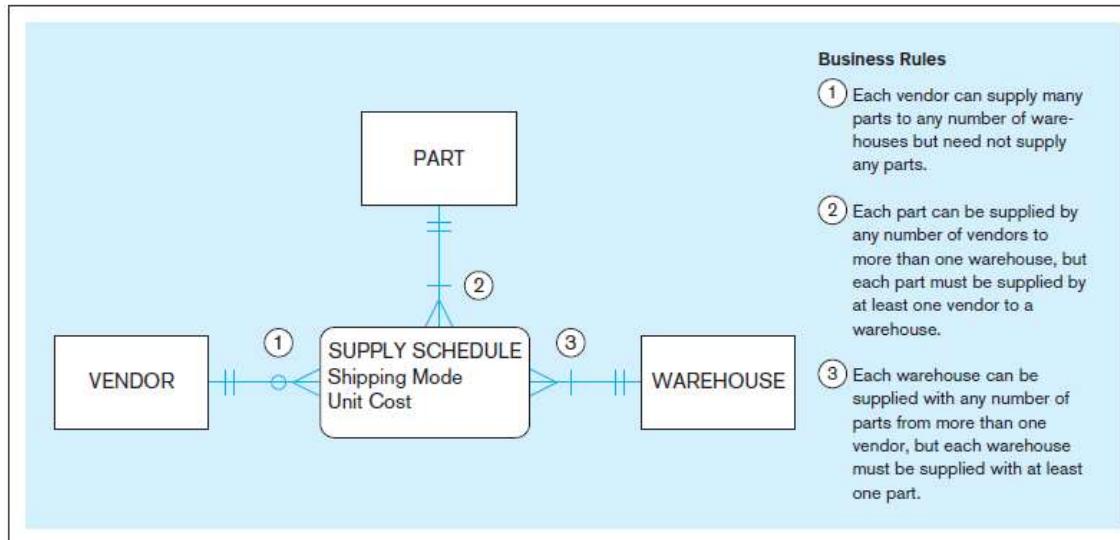
მაქსიმალური კარდინალურობა შესაძლებელია ფიქსირებული იყოს რიცხვით და არა ნებისმიერი მნიშვნელობით "მრავალი". მაგალითად, დავუშვათ, რომ კორპორატიული პოლიტიკა ადგენს, რომ თანამშრომელს შეუძლია ერთდღოულად მაქსიმუმ ხუთ პროექტზე იმუშაოს. ჩვენ შეგვიძლია ამ ბიზნესის წესის ჩვენება, 5-ის განთავსებით ზემოთ ან ქვემოთ „ყვავის თათის“ ძირში PROJECT არსის გვერდით, სურათი 2-17 ბ.

### სამეული კავშირი

სურათი 2-14-ზე ჩვენ ვაჩვენეთ სამეული კავშირი ასოციაციური არსის ტიპითან SUPPLY SCHEDULE (მიწოდების გრაფიკთი). ახლა ამ დიაგრამას დავამატოთ კარდინალურობის შეზღუდვები, ამ სიტუაციის ბიზნესის წესების საფუძველზე. E-R სქემა, შესაბამისი ბიზნესის წესებით, ნაჩვენებია ნახაზზე 2-18. გავითვალისწინოთ, რომ PART (ნაწილი) და WAREHOUSE (საწყობი) უნდა განეკუთვნებოდეს SUPPLY SCHEDULE (მიწოდების განრიგის) რომელიმე ეგზემპლარს, და VENDOR (მომწოდებელი) სურვილისამებრ შეიძლება არ იღებდეს მონაწილეობას. თითოეული მონაწილე სუბიექტის კარდინალურობა სავალდებულოა, რადგან

SUPPLY SCHEDULE თითოეული ეგზემპლარი დაკავშირებული უნდა იყოს თითოეული ამ არსის ტიპების ზუსტად ერთ ეგზემპლართან (მიწოდების გრაფიკი არის ასოციაციური არსი).

როგორც ადრე აღვნიშნეთ, სამეული კავშირი არ არის სამი ორობითი კავშირის ექვივალენტური. სამწუხაროდ, სამეული კავშირების გამოსახვა CASE-ის მრავალი ინსტრუმენტით შეუძლებელია; ამის ნაცვლად, იძულებული ვართ წარმოადგინოთ სამეული კავშირები, როგორც სამი ორობითი (ანუ ასოციაციური ერთეული, რომელსაც აქვს ორი ორობითი კავშირი) თუ იძულებულნი ვართ დავხაზოთ სამი ორობითი კავშირი, მაშინ არ უნდა დავხაზოთ ორობით ურთიერთობები სახელებით და უნდა დავრჩმუნდეთ, რომ კარდინალობა სამი ძლიერი არსის გვერდით არის სავალდებულო.



**FIGURE 2-18** Cardinality constraints in a ternary relationship

### დროზე დამოკიდებული მონაცემების მოდელირება

მონაცემთა ბაზის შინაარსი დროთა განმავლობაში იცვლება. მას შემდეგ განახლდა ინტერესი მიკვლევადობის და ორგანიზაციის ისტორიის სურათის რეკონსტრუქციისადმი სხვადასხვა მარეგულირებელი მოთხოვნებისთვის უზრუნველსაყოფად. მაგალითად, მონაცემთა ბაზაში, რომელიც შეიცავს პროდუქტის ინფორმაციას, თითოეული პროდუქტის ერთეული ფასი შეიძლება შეიცვალოს, მატერიალური და შრომითი ხარჯების და საბაზრო პირობების ცვალებადობის გამო. თუ საჭიროა მხოლოდ მიმდინარე ფასის დაფიქსირება, ფასის მოდელირება შესაძლებელია როგორც ერთმნიშვნელოვანი ატრიბუტი. ამასთან, საბუღალტრო აღრიცხვის, ბილინგის, ფინანსური ანგარიშგების და სხვა მიზნებისათვის, სავარაუდოდ, საჭიროა ფასების ისტორიის და თითოეული პერიოდის მოქმედების ვადის შნახვა. როგორც სურათი 2-19 გვიჩვენებს, შესსაძლებელია ამ მოთხოვნის კონცეპტუალიზაცია, როგორც ფასების სერია და თითოეული ფასის მოქმედების თარიღი. ამის შედეგად წარმოიქმნება (კომპოზიტიური) მრავალმნიშვნელოვანი ატრიბუტი, სახელწოდებით Price History, კომპონენტებით Price (ფასი) და Effective Date (მოქმედების თარიღი). ასეთი კომპოზიციური,

მრავალმნიშვნელოვანი ატრიბუტის მნიშვნელოვანი მახასიათებელია ის, რომ კომპონენტის ატრიბუტები ერთად განიხილებიან. ამრიგად, ნახაზზე 2-19, თითოეული ფასი დაწყვილებულია ძალაში შესვლის (Effective Date) შესაბამის თარიღთან.

დიაგრამა 2-19-ში, ატრიბუტის Price (ფასი) თითოეულ მნიშვნელობას აქვს „დროითი ჭდე“ (**time stamp**) რომელიც აღნიშნავს მისი მოქმედების თარიღს (Effective Date). დროის ჭდე უბრალოდ დროის მნიშვნელობაა, მაგალითად თარიღი და დრო, რომელიც დაკავშირებულია მონაცემთა მნიშვნელობასთან. დროის ჭდე შეიძლება უკავშირდებოდეს მონაცემთა ნებისმიერ მნიშვნელობას, რომელიც დროთა განმავლობაში იცვლება, როდესაც ამ მონაცემთა მნიშვნელობების ისტორიის შენახვა/შენარჩუნება საჭიროა. დროის მარკები შეიძლება ჩაიწეროს მონაცემის მნიშვნელობის შეტანის დროს (ტრანზაქციის დრო); მონაცემის მნიშვნელობის ძალაში შესვლის ან მოქმედების შეწყვეტის დროს; ან კრიტიკული მოქმედებების შესრულების დრო, როგორიცაა განახლებები, შესწორებები ან აუდიტები. ეს სიტუაცია მსგავსია თანამშრომელთა უნარების დიაგრამების ნახ. 2-15 ბ-ზე; ამრიგად, ალტერნატივა, რომელიც არ არის ნაჩვენები ნახ. 2-19-ზე, არის ფასების ისტორიის ცალკე არსის ტიპის გამოყოფა, როგორც ხდება Microsoft Visio-ს გამოყენებით.

მარტივი დროის ჭედის გამოყენება (როგორც წინა მაგალითში) ხშირად საკმარისია დროზე დამოკიდებული მონაცემების მოდელირებისათვის. ამასთან, დრომ შეიძლება შემოიტანოს ფაქტი სირთულეები მონაცემთა მოდელირებაში. მაგალითად, განვიხილოთ სურათი 2-17 გ. მოცემული დროის მომენტისთვის და არა ისტორიის საჩვენებლად. მეორეს მხრივ, თუ ჩვენ დაგვჭირდებოდა ინდივიდების ქორწინების სრული ისტორიის აღრიცხვა, ურთიერთობა Is Married To იქნებოდა არასავალდებულო ურთიერთობა „ბევრი-ბევრთან“, გარდა ამისა, შეიძლება დაგვჭირდეს თითოეული ქორწინების დასაწყისი და დასრულების თარიღის (სურვილისამებრ) დაფიქსირება; ეს თარიღები, ნახაზი 2-13 გ-ში მოცემული „მასალების სტრუქტურის“ (bill-of-materials) მსგავსი იქნება კავშირის ატრიბუტის ან ასოციაციური არსის ატრიბუტებით წარმოდგენილი.

### არსების ტიპებს შორის მრავლობითი კავშირების მოდელირება

მოცემულ ორგანიზაციაში შეიძლება არსებობდეს ერთზე მეტი კავშირი ერთი და იგივე არსების ტიპებს შორის. ნახაზზე 2-21 ნაჩვენებია ორი მაგალითი. დიაგრამა 2-21 ა გამოსახავს ორი კავშირს არსის ტიპებს შორის EMPLOYEE და DEPARTMENT. ამ ფიგურაში, გამოყენებულია სახელწოდებები თითოეული მიმართულების კავშირისათვის; ამ აღნიშვნით აშკარად ჩანს, თუ რა კარდინალურობაა კავშირის თითოეული მიმართულებისთვის (რაც მნიშვნელოვანი ხდება EMPLOYEE -ზე უნარული კავშირის მნიშვნელობის გასარკვევად). ერთი კავშირი EMPLOYEE უკავშირებს DEPARTMENT, სადაც ისინი მუშაობენ. ეს კავშირი არის „ერთი-ბენტან“, რომელსაც აქვს მიმართულება Has Workers და სავალდებულოა ორივე მიმართულებით. ეს ნიშნავს, რომ დეპარტამენტს უნდა ჰყავდეს მინიმუმ ერთი თანამშრომელი, რომელიც იქ მუშაობს (შესაძლოა დეპარტამენტის მენეჯერი) და თითოეული თანამშრომელი უნდა მუშაობდეს ზუსტად ერთ განყოფილებაში (შენიშვნა: ეს კონკრეტული ბიზნეს წესებია, რომლებიც საილუსტრაციოდ გამოვიყენეთ. მნიშვნელოვანია, რომ შემუშავდეს E-R დიაგრამა კონკრეტული სიტუაციისთვის იმ ბიზნესის წესების სრული გაგებით, რომლებიც გამოიყენება ამ გარემოში. მაგალითად, თუ EMPLOYEE შეიძლება

მოიცავდეს პენსიონერებს, მაშინ თითოეული თანამშრომელი შეიძლება მუშაობდეს ზუსტად ერთ განყოფილებაში; გარდა ამისა E-R მოდელი 2-2 ა – ზე მოცემულია ის, რომ ორგანიზაცია უნდა ინახავდეს ინფორმაციას ამჟამად რომელ დეპარტამენტი (DEPARTMENT) მუშაობს თითოეული თანამშრომელი (EMPLOYEE), და არა დეპარტამენტის დასაქმების ისტორიას. აუცილებლად უნდა გვახსოვდეს - მონაცემთა მოდელის სტრუქტურა ასახავს ინფორმაციას, რომლის დამახსოვრებასაც საჭიროებს ორგანიზაცია.

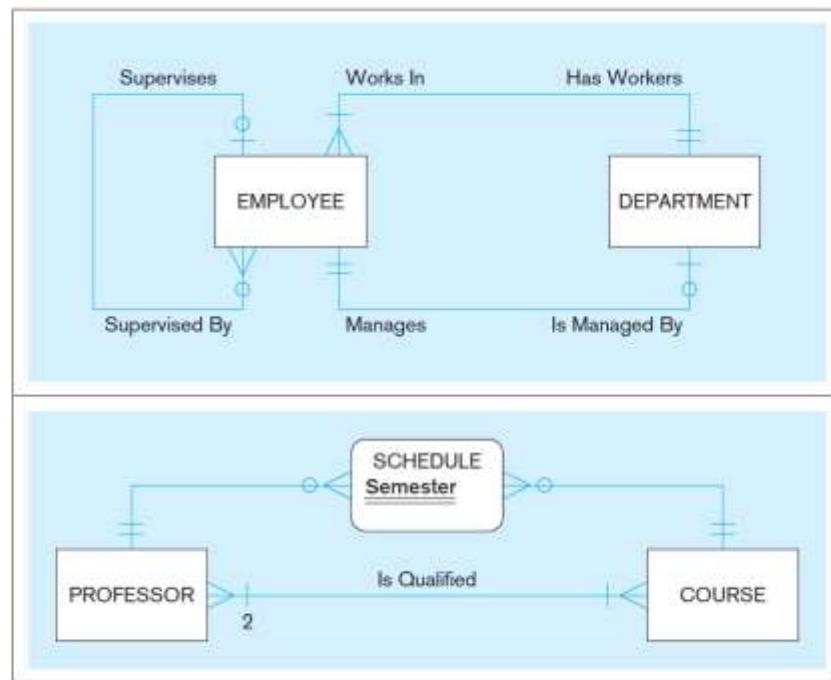
მეორე კავშირი თანამშრომელსა (EMPLOYEE) და დეპარტამენტს (DEPARTMENT) შორის ასახავს თითოეულ დეპარტამენტთის დამოვიდებულებას იმ თანამშრომელთან, რომელიც ხელმძღვანელობს ამ განყოფილებას. კავშირი DEPARTMENT -> EMPLOYEE, რომელსაც ამ მიმართულებას - Is Managed By - ასახავს სავალდებულოა, მიუთითებს, რომ დეპარტამენტს უნდა ჰყავდეს ზუსტად ერთი მენეჯერი. EMPLOYEE -> DEPARTMENT მიმართულების კავშირი Manages (ხელმძღვანელობს) არასავალდებულოა, რადგან მოცემული თანამშრომელი ან არის ან არ არის განყოფილების მენეჯერი.

დიაგრამა 2-21 ა ასევე აჩვენებს უნიარულ კავშირს, რომელითაც თითოეულ თანამშრომელს უკავშირებს მის ხელმძღვანელს და პირიქით. ეს კავშირი აღრიცხავს ბიზნესის წესს, რომ თითოეულ თანამშრომელს შეიძლება ჰყავდეს ზუსტად ერთი ხელმძღვანელი (Supervised By). და პირიქით, თითოეულ თანამშრომელს შეუძლია ზედამხედველობა გაუწიოს თანამშრომელთა ნებისმიერ რაოდენობას ან შეიძლება არ იყოს ზედამხედველი.

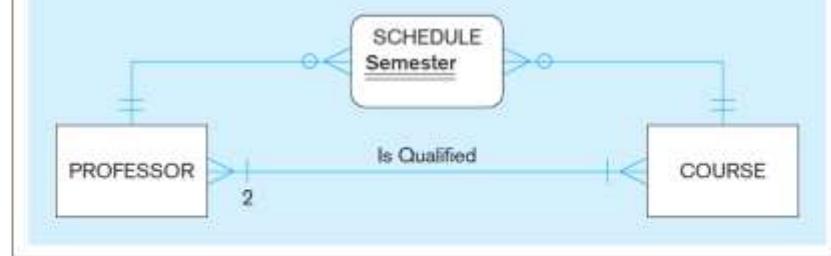
სურათი 2-21 ბ-ში მოცემულია ორი კავშირი არსის ტიპების პროფესორსა (PROFESSOR) და კურსს (COURSE) შორის. კავშირი Is Qualified უთანადებს პროფესორებს იმ კურსებთან, რომელთა სწავლისთვისაც გააჩნიათ კვალიფიკაცია. მოცემულ კურსს უნდა ჰყავდეს მინიმუმ ორი კვალიფიციური ინსტრუქტორი (მაგალითი იმისა, თუ როგორ გამოიყენება ფიქსირებული მნიშვნელობა მინიმალური ან მაქსიმალური კარდინალურისთვის). ეს აღნიშნავს, რომ კურსი არასოდეს იქნება ერთი ინსტრუქტორის "საკუთრება". პირიქით, თითოეულ ინსტრუქტორს უნდა ჰქონდეს კვალიფიკაცია, რომ ასწავლოს მინიმუმ ერთი კურსი მაინც (გონივრული მოლოდინი).

კვლავ განვიხილოთ ნახ. 2-21 ბ. გავარკვიოთ რა არის SCHEDULE ასოციაციური არსის იდენტიფიკატორი. გასათვალისწინებელია, რომ Semester (სემესტრი) ნაწილობრივი იდენტიფიკატორია; ამრიგად, სრული იდენტიფიკატორი იქნება PROFESSOR-ის იდენტიფიკატორი, დამატებული როგორც კურსის (COURSE) იდენტიფიკატორი, ასევე Semester. იმის გამო, რომ ასოციაციური არსისთვის ასეთი სრული იდენტიფიკატორი შეიძლება გახდეს გრძელი და რთული, ხშირად ხდება სუროგატი იდენტიფიკატორების შექმნა თითოეული ასოციაციური არსისთვის; ამგვარად Schedule ID (განრიგის ID) შეიქმნება როგორც SCHEDULE იდენტიფიკატორი, ხოლო Semester იქნება ატრიბუტი. ამ შემთხვევაში დაიკარგა მკაფიო ბიზნეს წესი, რომლის მიხედვით PROFESSOR და COURSE იდენტიფიკატორების და Semester კომბინაცია უნდა იყოს უნიკალური თითოეული SCHEDULE ეგზემპლარისათვის (რადგან ეს კომბინაცია არის განრიგის იდენტიფიკატორი). რა თქმა უნდა, ამას შეიძლება დაემატოს კიდევ ერთი ბიზნესის წესი.

**FIGURE 2-21** Examples of multiple relationships  
(a) Employees and departments



(b) Professors and courses (fixed lower limit constraint)



### კავშირების სახელდება და განმარტება

მონაცემთა ობიექტების სახელდების ზოგადი სახელმძღვანელო მითითებების გარდა, კავშირების სახელდების რამდენიმე სპეციალური სახელმძღვანელო მითითება არსებობს, რომლებიც შემდეგში მდგომარეობს:

- კავშირის სახელი არის ზმნური ფრაზა (მაგალითად, Assigned To - დაინიშნა , Supplies ესაჭიროება, Teaches - ასწავლის). კავშირები წარმოადგენენ მოქმედებებს, რომლებიც ჩვეულებრივ აწყობენ დროშია, ამიტომ გარდამავალი ზმნები (მოქმედება რაღაცაზე) ყველაზე შესაფერისია. კავშირის სახელი ასახავს განხორციელებულ მოქმედებას და არა მოქმედების შედეგს (მაგალითად, გამოიყენეთ Assigned To და არა Assignment). სახელი ასახავს მონაწილე არსის ტიპებს შორის ურთიერთქმედების შინაარსს და არა პროცესს (მაგალითად, გამოიყენეთ EMPLOYEE is Assigned To a PROJECT, და არა EMPLOYEE is Assigning a PROJECT)
- თავიდან უნდა აიცილოთ ბუნდოვანი სახელები, როგორიცაა Has-„აქვს“ ან Is Related To - „არის დაკავშირებული“. გამოიყენეთ აღწერითი, ძლიერი ზმნური ფრაზები, რომლებიც ხშირად აღებულია მოქმედების ზმნებიდან, რომლებიც გვხვდება კავშირის განსაზღვრებაში.

ასევე არსებობს კავშირების განმარტების გარკვეული კონკრეტული სახელმძღვანელო მითითებები, რომლებიც შემდეგში მდგომარეობს:

- კავშირის განმარტება განსაზღვრავს რა მოქმედება ხორციელდება და რატომ არის ეს მნიშვნელოვანი. შეიძლება მნიშვნელოვანი იყოს იმის მითითება, ვინ ან რა ახორციელებს მოქმედებას, მაგრამ არ არის მნიშვნელოვანი ახსნა, თუ როგორ ხდება მოქმედება. კავშირში ჩართული ბიზნეს ობიექტების მითითება ბუნებრივია, მაგრამ იმის გამო, რომ E-R

დიაგრამა გვიჩვენებს, თუ რა ტიპის არსები მონაწილეობენ კავშირებში და სხვა აღწერები განმარტავს არსების ტიპებს, არაა საჭირო ბიზნესის ობიექტების აღწერა.

- შესაძლოა ასევე მნიშვნელოვანი იყოს მოქმედების სიცხადისათვის მაგალითები. მაგალითად, Registered For სტუდენტსა და კურსს შორის კავშირისათვის შეიძლება სასარგებლო იყოს იმის ახსნა, რომ ეს მოიცავს როგორც ადგილზე რეგისტრაციას, ასევე ონლაინ რეგისტრაციას და მოიცავს რეგისტრაციებს, რომლებიც ხორციელდება აკადემიური რეგისტრაციის პერიოდში.
- განმარტებამ უნდა განსაზღვროს ნებისმიერი არასავალდებულო მონაწილეობა. უნდა აიხსნას თუ რა პირობებში ხდება ნულოვანი კავშირის შემთხვევები, შეიძლება ეს მოხდეს მხოლოდ მაშინ, როდესაც არსის ეგზემპლარი პირველად შეიქმნება, ან შეიძლება ეს მოხდეს ნებისმიერ დროს. მაგალითად, „Registered For კურსს უკავშირებს სტუდენტს, რომლებიც დარეგისტრირდნენ, და კურსები, რომლებზეც სტუდენტი დარეგისტრირდა. რეგისტრაციის პერიოდის დაწყებამდე კურსზე არ იქნება სტუდენტი დარეგისტრირებული და შეიძლება არასოდეს ჰყავდეს რეგისტრირებული სტუდენტი. სტუდენტი არ დარეგისტრირდება არცერთ კურსზე რეგისტრაციის პერიოდის დაწყებამდე და შეიძლება არ დარეგისტრირდეს არც ერთ კურსზე (ან შეიძლება დარეგისტრირდეს კურსზე, შემდეგ კი უარი თქვას რომელიმე ან ყველა კურსზე).
- კავშირის განმარტებამ ასევე მკაფიოდ გამოხატოს მაქსიმალური კარდინალურობის მიზეზი, გარდა შემთხვევისა „მრავალი“. მაგალითად, Assigned To კავშირი თანამშრომლის დასაკავშირებლად იმ პროექტებთან, რომლებშიც ეს თანამშრომელია ჩართული და პროექტთან დაკავშირებული თანამშრომლების აღსარიცხად. ორგანიზაციის შინაგანაწესის გამო, თანამშრომელი არ შეიძლება მოცემულ დროის განმავლობაში ოთხზე მეტ პროექტში ღებულობდეს მონაწილეობას. ეს მაგალითი, ასახავს ბევრ ბიზნეს წესს, რომელიც განსაზღვრავს მაქსიმალურ კარდინალურობას, რომელიც ასევე არ იყოს მუდმივი. ამ მაგალითში, ახალმა შინაგანაწესმა შეიძლება გაზარდოს ან შეამციროს ეს ზღვარი. ამრიგად, მაქსიმალური კარდინალურობის განსაზღვრაში უნდა განხორციელდეს ცვლილებები.
- კავშირების განმარტებამ უნდა ახსნას ურთიერთგამომრიცხავი კავშირები. ურთიერთგამომრიცხავი ურთიერთობები არის ურთიერთობები, რომელთა არსების ეგზემპლარს შეუძლია მონაწილეობა მიიღოს რამდენიმე ალტერნატიული კავშირიდან მხოლოდ ერთში. განვიხილოთ შემდეგი მაგალითი: "Playys On აკავშირებს უნივერსიტეტთაშორის სპორტულ გუნდს თავის სტუდენტ-მონაწილეებთან და მიუთითებს, თუ რომელ გუნდებში თამაშობს სტუდენტი. სტუდენტებს, რომლებიც თამაშობენ უნივერსიტეტთაშორის სპორტულ გუნდებში, არ შეუძლიათ იმუშაონ უნივერსიტეტის სამსახურში. ურთიერთგამომრიცხავი შეზღუდვის კიდევ ერთი მაგალითია, როდესაც თანამშრომელი არ შეიძლება ერთსა და იმავე თანამშრომელის მენეჯერიც იყოს და იყოს მასზე დაქორწინებული.
- კავშირის განმარტებამ უნდა ახსნას მასში მონაწილეობის ნებისმიერი შეზღუდვა. ურთიერთგამომრიცხველობა ერთ-ერთი შეზღუდვაა, მაგრამ შეიძლება იყოს სხვაც. მაგალითად, კავშირი Supervised By უკავშირებს თანამშრომელს სხვა თანამშრომლებთან, რომლებსაც იგი ზედამხედველობს და თანამშრომელს უკავშირებს სხვა თანამშრომელს,

რომელიც მას ზედამხედველობს. თანამშრომელს არ შეუძლია მასზე ზედამხედველობა, ხოლო თანამშრომელს არ შეუძლია სხვა თანამშრომლებზე ზედამხედველობა, თუ მისი სამუშაო კლასიფიკაციის დონე 4-ზე დაბალია.

- კავშირის განმარტებამ უნდა ახსნას ისტორიის მასშტაბები, რომლებიც ინახება კავშირში. მაგალითად, ” ხდება საავადმყოფოს საწოლის დაკავშირება (მიმაგრება) პაციენტთან. ინახება მხოლოდ მიმდინარე საწოლის დაკავება (მიმაგრება). როდესაც პაციენტის მიიღება არ ხდება, ეს პაციენტი არ არის მიმაგრებული საწოლზე, ხოლო საწოლი შეიძლება ვაკანტური იყოს დროის ნებისმიერ მონაკვეთში”. კავშირის ისტორიის აღწერის კიდევ ერთი მაგალითია: „ადგილი“ აკავშირებს მომხმარებელს იმ შეკვეთებთან, რომლებიც მან დადო ჩვენს კომპანიასთან და უკავშირებს შეკვეთას დაკავშირებულ მომხმარებელთან. მხოლოდ ორი წლის შეკვეთები ინახება მონაცემთა ბაზაში, ამიტომ ყველა შეკვეთა ვერ მიიღებს მონაწილეობას ამ ურთიერთობაში“.
- კავშირის განმარტებამ უნდა ახსნას, შეუძლია თუ არა კავშირის ეგზემპლარში მონაწილე არსის ეგზემპლარს გადასცეს მონაწილეობბა კავშირის სხვა ეგზემპლარს. მაგალითად, „ადგილი აკავშირებს მომხმარებელს იმ შეკვეთებთან, რომლებიც მან დადო ჩვენს კომპანიასთან და უკავშირებს შეკვეთას დაკავშირებულ მომხმარებელთან. შეკვეთა სხვა მომხმარებელზე არ გადაეცემა“.

## არსი-კავშირი გაუმჯობესებული მოდელი (EER მოდელი)

ჩვენს მიერ აღწერილი E-R ძირითადი მოდელი პირველად 1970 – იანი წლების შუა პერიოდში შემოიღეს. იგი შეეფერებოდა ყველაზე გავრცელებული ბიზნეს პრობლემების მოდელირებას და ფართო გამოყენება ჰქონდა. თუმცა ბიზნესის გარემო მკვეთრად შეიცვალა. საქმიანი კავშირები უფრო გართულდა და შედეგად, ბიზნესის მონაცემებიც გაცილებით რთულია. თანამედროვე ბიზნეს სამყარო ორგანიზაციულ მონაცემთა ბაზებს ბევრად მეტ მოთხოვნებს უყენებს. ამ ცვლილებების უკეთ დასაძლევად, მკვლევარებმა და კონსულტანტებმა განაგრძეს E-R მოდელის გაუმჯობესება, რათა უფრო ზუსტად მოხდეს რთული მონაცემების წარმოდგენა, რომლებიც გვხვდება დღევანდელ ბიზნეს გარემოში. ტერმინი „გაუმჯობესებული არსი-კავშირი (EER) მოდელი“ გამოიყენება იმ მოდელის აღსანიშნად, რომელიც წარმოიშვა ორიგინალური E-R მოდელის ახალი სამოდელო კონსტრუქციებით გაფართოების შედეგად. ეს გაფართოებები EER მოდელს სემანტიკურად ობიექტზე ორიენტირებული მონაცემების მოდელის მსგავსს ხდის.

EER მოდელში ჩასმული ყველაზე მნიშვნელოვანი სამოდელო კონსტრუქცია არის სუპერტიპის/ქვეტიპის კავშირები. ეს კონსტრუქცია საშუალებას გვაძლევს, მოვახდინოთ ზოგადი არსის ტიპის მოდელირება (ე.წ. სუპერტიპი) და შემდეგ მისი დაყოფა რამდენიმე სპეციალიზებული არსის ტიპებად (ე.წ. ქვეტიპები). მაგალითად, არსის ტიპი CAR შეიძლება მოდელირებული იყოს როგორც სუპერტიპი ქვეტიპებით SEDAN, SPORTS CAR, COUPE და A.შ. თითოეული ქვეტიპი ატრიბუტებს მერკვიდრეობით იღებს თავისი სუპერტიპიდან და დამატებით შეიძლება ჰქონდეს სხვა ატრიბუტები და მონაწილეობა მიღოს საკუთარ, მხოლოდ ამ ქვეტიპისათვის განკუთვნილ კავშირებში. სუპერტიპი/ქვეტიპის კავშირების მოდელირებისთვის ახალი აღნიშვნების დამატებამ მნიშვნელოვნად გაუმჯობესა ძირითადი E-R მოდელის მოქნილობა.

E-R და განსაკუთრებით EER დიაგრამები, საწყისი ბიზნეს ამოცანიდან გამომდინარე, შეიძლება დიდი და რთული გახდეს, რის გამოსახვისათვისაც დაგვჭირდეს მრავალი გვერდის (ან ძალიან მცირე შრიფტის) გამოყენება. ზოგიერთი მონაცემთა ბაზა მოიცავს ასობით არსს. ბევრ მომხმარებელს და მენეჯერს, რომლებიც განსაზღვრავენ მონაცემთა ბაზის მოთხოვნებს ან იყენებენ მას, არ სჭირდებათ ყველა არსის, კავშირის და ატრიბუტის დანახვა მონაცემთა ბაზის იმ ნაწილის გასაგებად, რომლითაც ისინი ყველაზე მეტად არიან დაინტერესებული. არსების კლასტერირება წარმოადგენს არსი-კავშირი მონაცემთა მოდელის ნაწილის იმავე მონაცემების უფრო მაკრო-დონის ხედვად გადაქცევის საშუალებას. არსის კლასტერირება არის იერარქიული დეკომპოზიციის ტექნიკა (ჩალაგების პროცესი, რომლის დროსაც სისტემა იყოფა შემდგომ დამატებით და კიდევ შემდგომ დამატებით ნაწილებად), რამაც შეიძლება გაამარტივოს E-R დიაგრამების წაკითხვა და მონაცემთა ბაზების შემუშავება. არსების და კავშირების დაჯგუფებით, შეგვიძლია ჩამოაყალიბოთ E-R დიაგრამა ისე, რომ ყურადღება მიექცეს მოდელის იმ დეტალებს, რომლებიც ყველაზე მნიშვნელოვანია მოცემული მონაცემების მოდელირების ამოცანაში.

როგორც ადრე იქნა ნაჩვენები, უნივერსალური და ინდუსტრიის სპეციფიკური მონაცემების განზოგადებადი მოდელები, რომლებიც ფართოდ იყენებდნენ EER შესაძლებლობებს, ძალიან მნიშვნელოვანი გახდა თანამედროვე მონაცემების შემდგენებისათვის. ამ შეფუთულმა (დაფასოებულმა) მონაცემების მოდელებმა და მონაცემთა მოდელის შაბლონებმა მონაცემთა შემმუშავებლები უფრო ეფექტური გახადეს და შექმნეს მონაცემთა უფრო მაღალი ხარისხის

მოდელები. სუპერტიპები/ქვეტიპების EER მახასიათებლები მნიშვნელოვანია მონაცემთა განზოგადებული მოდელების შესაქმნელად; ასევე გამოიყენება დამატებითი განზოგადებული კონსტრუქციები, როგორიცაა მატიპიზირებელი არსები და კავშირები. მონაცემთა მოდელის შემმუშავებლელთათვის ძალზე მნიშვნელოვანი გახდა იმის ცოდნა, თუ როგორ მოირგონ მონაცემთა მოდელის შაბლონი ან მონაცემთა მოდელი ძირითადი პროგრამული პაკეტისთვის (მაგ., საწარმოს რესურსების დაგეგმვა ან მომხმარებლებთან კავშირის მენეჯმენტი), ისევე როგორც ეს გახდა ჩვეულებრივი რამ ინფორმაციული სისტემის შემქმნელებისთვის მზა პროგრამული პაკეტების და პროგრამული კომპონენტების მორგება.

### ზეტიპებისა და ქვეტიპების წარმოდგენა

გავიხსენოთ განმარტება - არსის ტიპი არის ეგზემპლართა ერთობლიობა, რომლებიც საერთო თვისებებსა ან მახასიათებლებს იზიარებენ. მიუხედავად იმისა, რომ არსების ეგზემპლარები, რომლებსაც მოიცავს არასის ტიპი, მსგავსია, არაა აუცილებელი მათ ზუსტად ერთნაირი ატრიბუტები ჰქონდეთ. მაგალითად, გავიხსენოთ სავალდებულო და არასავალდებულო ატრიბუტები. მონაცემთა მოდელირების ერთ-ერთი მთავარი გამოწვევაა ამოვიცნოთ და მკაფიოდ წარმოადგინოთ არსები, რომლებიც თითქმის ერთნაირია, ანუ არსების ტიპები, რომლებსაც აქვთ საერთო თვისებები, მაგრამ ასევე აქვთ ერთი ან მეტი განსხვავებული თვისება, რომლებიც ორგანიზაციისთვის. მაგალითად, STUDENT არის არსის ტიპი უნივერსიტეტში. სტუდენტის ორი ქვეტიპია GRADUATE STUDENT (მაგისტრატურის სტუდენტი) და UNDERGRADUATE STUDENT (ბაკალავრიატის სტუდენტი). ამ მაგალითში, STUDENT- ს სუპერტიპად ვგულისხმობთ. სუპერტიპი არის ზოგადი არსის ტიპი, რომელსაც აქვს კავშირი ერთ ან მეტ ქვეტიპთან.

### ძირეული ცნებები და აღნიშვნები

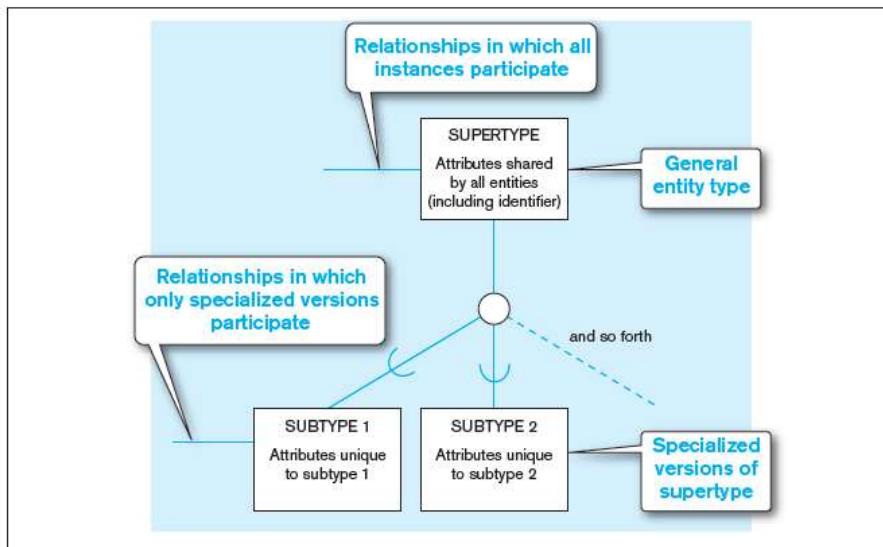
აღნიშვნა, რომელიც გამოიყენება სუპერტიპი/ქვეტიპი კავშირებისათვის ამ ტექსტში, ნაჩვენებია ნახაზზე 3-1a. სუპერტიპი უკავშირდება ხაზს წრესთან, რაც თავის მხრივ უკავშირდება ხაზს თითოეულ ქვეტიპს, რომელიც განსაზღვრულია. U- ფორმის სიმბოლო ქვეჯგუფს წრესთან დამაკავშირებელ თითოეულ სტრიქონზე ხაზს უსვამს, რომ ქვეტიპი არის სუპერტიპის ქვეჯგუფი. იგი ასევე მიუთითებს ქვეტიპი/სუპერტიპის კავშირის მიმართულებაზე (U არასავალდებულოა, რადგან სუპერტიპი/ქვეტიპის კავშირის მნიშვნელობა და მიმართულება, როგორც წესი, აშკარაა). ნახაზი 3-1 ბ აჩვენებს Microsoft Visio-ს მიერ გამოყენებული EER აღნიშვნის ტიპს (რომელიც ამ ტექსტში გამოყენებულის მსგავსია) და ნახაზი 3-1 გ გვიჩვენებს EER აღნიშვნის ტიპს, რომელიც გამოიყენება ზოგიერთი CASE ხელსაწყოს მიერ (მაგ., Oracle Designer); ნახაზი 3-1c ასევე წარმოადგენს ფორმას, რომელიც ხშირად გამოიყენება უნივერსალური და ინდუსტრიის სპეციფიკური მონაცემების მოდელებისათვის.

ატრიბუტები, რომლებიც ყველა არსისთვის საზიაროა (იდენტიფიკატორის ჩათვლით), დაკავშირებულია სუპერტიპთან. ატრიბუტები, რომლებიც უნიკალურია კონკრეტული ქვეტიპისთვის, დაკავშირებულია ამ ქვეტიპთან. იგივეა კავშირებშიც. ამ აღნიშვნას კიდევ ემატება სხვა კომპონენტები, რომლებიც დამატებით მნიშვნელობას იძლევან სუპერტიპი/ქვეტიპი კავშირებში.

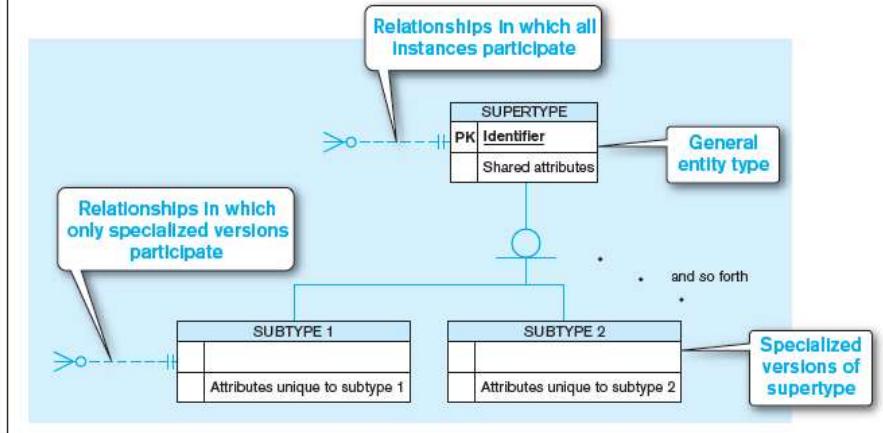
## ქვეტიპის და ზეტიპის მაგალითი

სუპერტიპი/ქვეტიპი კავშირის თვალსაჩინოებისათვის განვიხილოთ მარტივი, მაგრამ ზოგადი მაგალითი. დავუშვათ, რომ ორგანიზაციას ჰყავს სამი ძირითადი ტიპის თანამშრომელი: საათობრივი, ხელფასიანი და კონტრაქტორების კონსულტანტები. ქვემოთ მოცემულია რამდენიმე მნიშვნელოვანი ატრიბუტი ამ ტიპის თანამშრომლებისთვის:

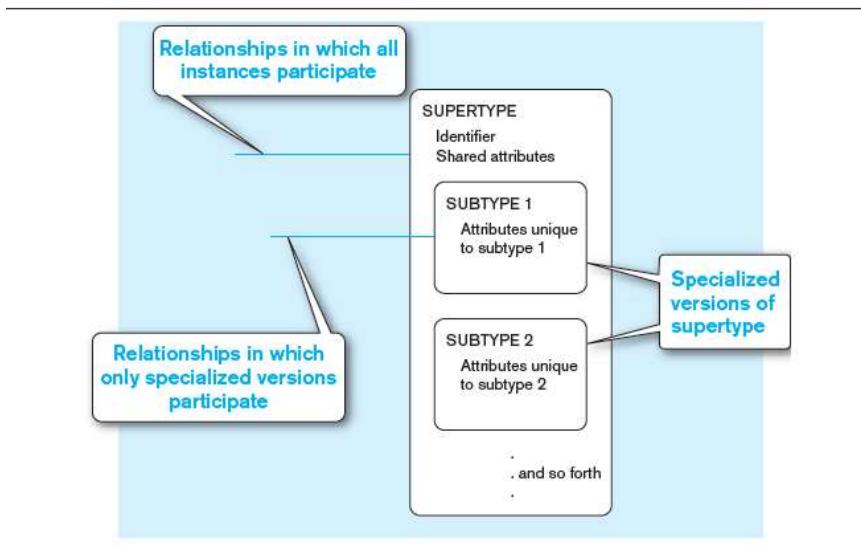
**FIGURE 3-1** Basic notation for supertype/subtype relationships  
(a) EER notation



(b) Microsoft Visio notation



**FIGURE 3-1 (continued)**  
(c) Subtypes inside supertypes notation



- **Hourly employees** - Employee Number, Employee Name, Address, Date Hired, Hourly Rate (საათობრივი თანამშრომელი - თანამშრომლის ნომერი, თანამშრომლის სახელი, მისამართი, დაქირავების თარიღი, საათობრივი ანაზღაურება)
- **Salaried employees** - Employee Number, Employee Name, Address, Date Hired, Annual Salary, Stock Option (ხელფასიანი თანამშრომლები - თანამშრომლის ნომერი, თანამშრომლის სახელი, მისამართი, დაქირავების თარიღი, წლიური ანაზღაურება, საფონდო აქციები)
- **Contract consultants** - Employee Number, Employee Name, Address, Date Hired, Contract Number, Billing Rate (კონტრაქტორის კონსულტანტები - თანამშრომლის ნომერი, თანამშრომლის სახელი, მისამართი, დაქირავების თარიღი, ხელშეკრულების ნომერი, ბილინგის კურსი)

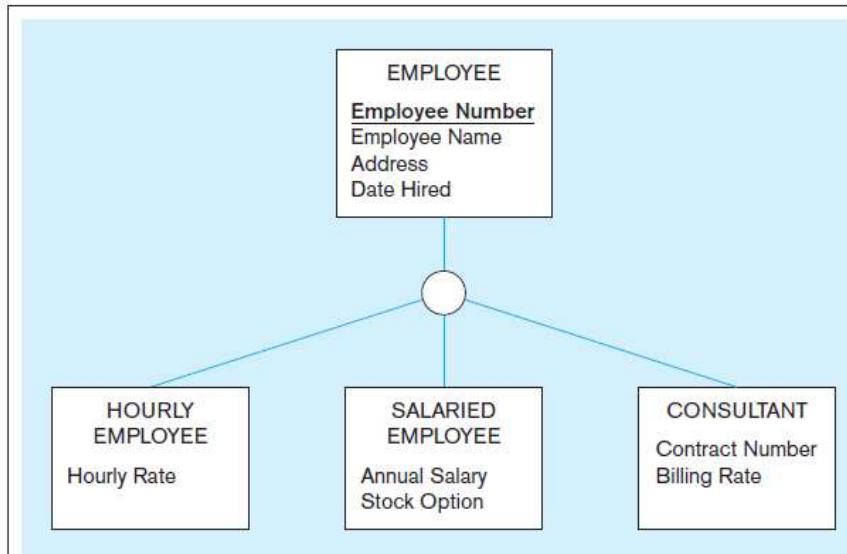
გავითვალისწინოთ, რომ თანამშრომლის ყველა ტიპს აქვს რამდენიმე საერთო ატრიბუტი: თანამშრომლის ნომერი, თანამშრომლის სახელი, მისამართი და დაქირავების თარიღი. გარდა ამისა, თითოეულ ტიპს აქვს ერთი ან მეტი ატრიბუტი, რომელიც განსხვავდება სხვა ტიპის ატრიბუტებისგან (მაგალითად, საათობრივი კურსი მხოლოდ საათობრივი თანამშრომლებისთვისაა დამახასიათებელი). თუ ამ ვითარებაში შევიმუშავებდით მონაცემთა კონცეპტუალურ მოდელს, შეიძლება განგვეხილა სამი არჩევანი:

1. განვსაზღვროთ ერთი არსის ტიპი, სახელწოდებით EMPLOYEE. მიუხედავად იმისა, რომ კონცეპტუალურად მარტივია, ამ მიდგომას აქვს ის მინუსი, რომ EMPLOYEE უნდა ჰქონდეს სამივე ტიპის თანამშრომლის ყველა ატრიბუტი. საათობრივი მოსამსახურის ეგზემპლარისათის (მაგალითად), ატრიბუტები, როგორიცაა წლიური ანაზღაურება და ხელშეკრულების ნომერი, არ იქნება გამოიყენებული (არასავალდებულო ატრიბუტები) და ექნება მნიშვნელობა NULL ან არ გამოიყენება. პროგრამის შემუშავების გარემოში გადაყვანისას, ამ ტიპის არსის გამოყენება საკმაოდ რთულია - უნდა გაუმკლავდთ მრავალ ვარიაციას.
2. განისაზღვროს არსის ცალკეული ტიპი სამივე არსისთვის. ეს მიდგომა არ იძლევა თანამშრომლების საერთო თვისებების გამოყოფის საშუალებას და მომხმარებლები სიფრთხილით უნდა მოეკიდონ სისტემის გამოყენებისას სწორი ტიპის არსის არჩევას.

3. განსაზღვრეთ სუპერტიპი სახელწოდებით EMPLOYEE ქვეტიპებით HOURLY EMPLOYEE, SALARIED EMPLOYEE და CONSULTANT. ეს მიდგომა იყენებს ყველა თანამშრომლის საერთო თვისებებს, თუმცა ამასთან იგი ანსხვავებს თითოეული ტიპის თვისებებს.

ნახაზი 3-2 გვიჩვენებს EMPLOYEE სუპერტიპის წარმოდგენას მისი სამი ქვეტიპით, გაუმჯობესებული E-R აღნიშვნის გამოყენებით. ყველა თანამშრომლის მიერ გაზიარებული ატრიბუტები ასოცირდება EMPLOYEE არსის ტიპთან. ატრიბუტები, რომლებიც თავისებურია თითოეული ქვეტიპისთვის, შედის მხოლოდ ამ ქვეტიპში.

**FIGURE 3-2** Employee supertype with three subtypes



### ატრიბუტის მემკვიდრეობითობა

ქვეტიპი თავისთავად არის არსის ტიპი. ქვეტიპის არსის ეგზემპლარი წარმოადგენს სუპერტიპის იგივე არსის ეგზემპლარს. მაგალითად, თუ "ტერეზ ჯონსი" არის CONSULTANT ქვეტიპის ეგზემპლარი, მაშინ ეს იგივე პიროვნება აუცილებლად EMPLOYEE სუპერტიპის ეგზემპლარიცაა. შედეგად, ქვეტიპის არსი უნდა ფლობდეს არა მხოლოდ საკუთარი ატრიბუტების მნიშვნელობებს, არამედ ატრიბუტების მნიშვნელობებს, როგორც სუპერტიპის წევრი, იდენტიფიკატორის ჩათვლით.

ატრიბუტის მემკვიდრეობა არის თვისება, რომლითაც ქვეტიპის არსი მემკვიდრეობით იღებენ ყველა ატრიბუტის მნიშვნელობებს და სუპერტიპის ყველა არსის ეგზემპლარს. ეს მნიშვნელოვანი თვისება ზედმეტს ხდის სუპერტიპის ატრიბუტების ან კავშირების ქვეტიპებთან შეტანას. მაგალითად, Employee Name არის EMPLOYEE-ის ატრიბუტი (სურათი 3-2), მაგრამ არა EMPLOYEE-ის ქვეტიპები. ამრიგად, თანამშრომლის სახელი "ტერეზ ჯონსი" მემკვიდრეობით მიიღება EMPLOYEE სუპერტიპიდან. ამასთან, ბილინგის კოეფიციენტი იგივე თანამშრომლისთვის ქვეტიპის CONSULTANT ატრიბუტია.

ცხადია, რომ ქვეტიპის წევრი უნდა იყოს სუპერტიპის წევრი. მაგრამ არაა აუცილებელი სუპერტიპის წევრი იყოს ერთი (ან მეტი) ქვეტიპის წევრი. ეს კონკრეტული ბიზნესის მდგომარეობიდან (ბიზნეს წესებიდან) გამომდინარეობს.

## სუპერტიპი/ქვეტიპი კავშირის გამოყენების პირობები

სუპერტიპი/ქვეტიპი კავშირის გამოყენებისათვის არსებობს ორი პირობა, რომელთაგან შეიძლება გავითვალისწინოთ ერთერთი ან ორივე ერთდღოულად:

1. არსებობენ ატრიბუტები, რომლებიც გამოიყენება არსის ტიპის ზოგიერთ (მაგრამ არა ყველა) ეგზემპლარისასთვის. მაგალითად, EMPLOYEE არსის ტიპი ნახაზზე 3-2.
2. ქვეტიპის ეგზემპლარები მონაწილეობენ ამ ქვეტიპისთვის დამახასიათებელ უნიკალურ კავშირებში.

ნახაზი 3-3 არის ქვეტიპის კვშირის გამოყენების მაგალითი, რომელიც ასახავს ორივე ამ სიტუაციას. საავადმყოფოს არსის ტიპის PATIENT (პაციენტს) აქვს ორი ქვეტიპი: OUTPATIENT (ამბულატორიული) და RESIDENT PATIENT (რეზიდენტი) პაციენტი (იდენტიფიკატორი არის პაციენტის ID). ყველა პაციენტს აქვს Admit Date (მიღების თარიღი) ატრიბუტი, ისევე როგორც Patient Name (პაციენტის სახელი). ასევე, ყველა პაციენტზე ზრუნავს RESPONSIBLE PHYSICIAN (პასუხისმგებელი ექიმი), რომელიც შეიმუშავებს პაციენტის მკურნალობის გეგმას.

თითოეულ ქვეტიპს აქვს ატრიბუტი, რომელიც უნიკალურია ამ ქვეტიპისთვის. OUTPATIENT აქვთ Checkback Date (შემოწმების თარიღი), ხოლო RESIDENT PATIENT აქვთ Date Discharged (გამოწერის თარიღი). ასევე, რეზიდენტ პაციენტებს აქვთ უნიკალური კავშირი, რომელიც თითოეულ პაციენტს საწილზე ათავსებს (გავითვალისწინოთ, რომ ეს სავალდებულო კავშირია, ეს არასავალდებულო იქნებოდა, თუ იგი პაციენტთან იქნებოდა დამაგრებული). თითოეული საწოლი შეიძლება მიემაგროს ან არ მიემაგროს პაციენტს.

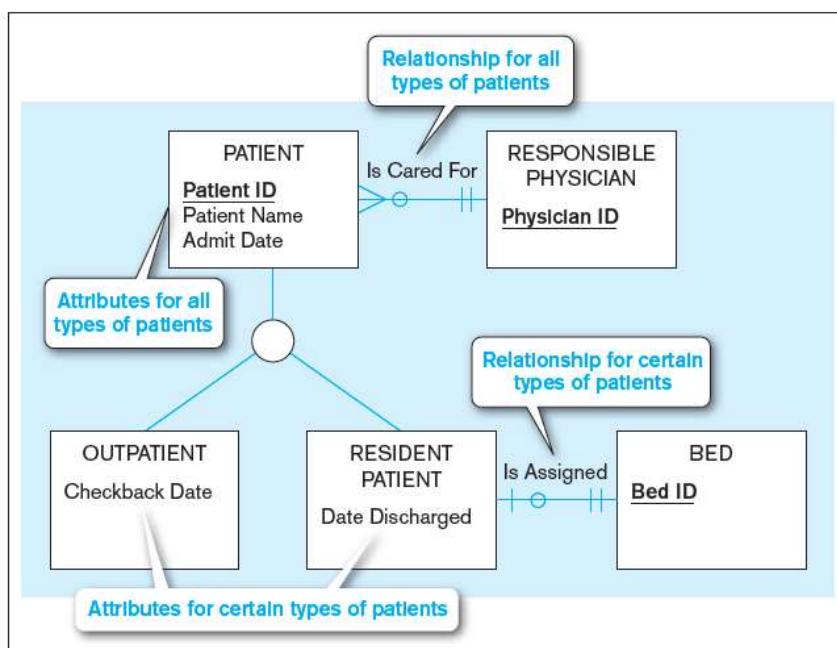


FIGURE 3-3 Supertype/subtype relationships in a hospital

## განკურძოებისა და განზოგადების წარმოდგენა

ჩვენ აღვწერეთ და წარმოვაჩინეთ სუპერტიპი / ქვეტიპი კავშირის ძირითადი პრინციპები, მათ შორის "კარგი" ქვეტიპის მახასიათებლები. მაგრამ აუცილებელია შეგვეძლოს რეალური მონაცემების მოდელების შემუშავებისას ამოვიცნოთ ასეთი კავშირების გამოყენების შესაძლებლობები. არსებობს ორი პროცესი - განზოგადება და განკურძოება - რომლებიც ემსახურებიან მენტალურ მოდელებს სუპერტიპი/ქვეტიპის კავშირების ჩამოყალიბებაში.

### განზოგადება

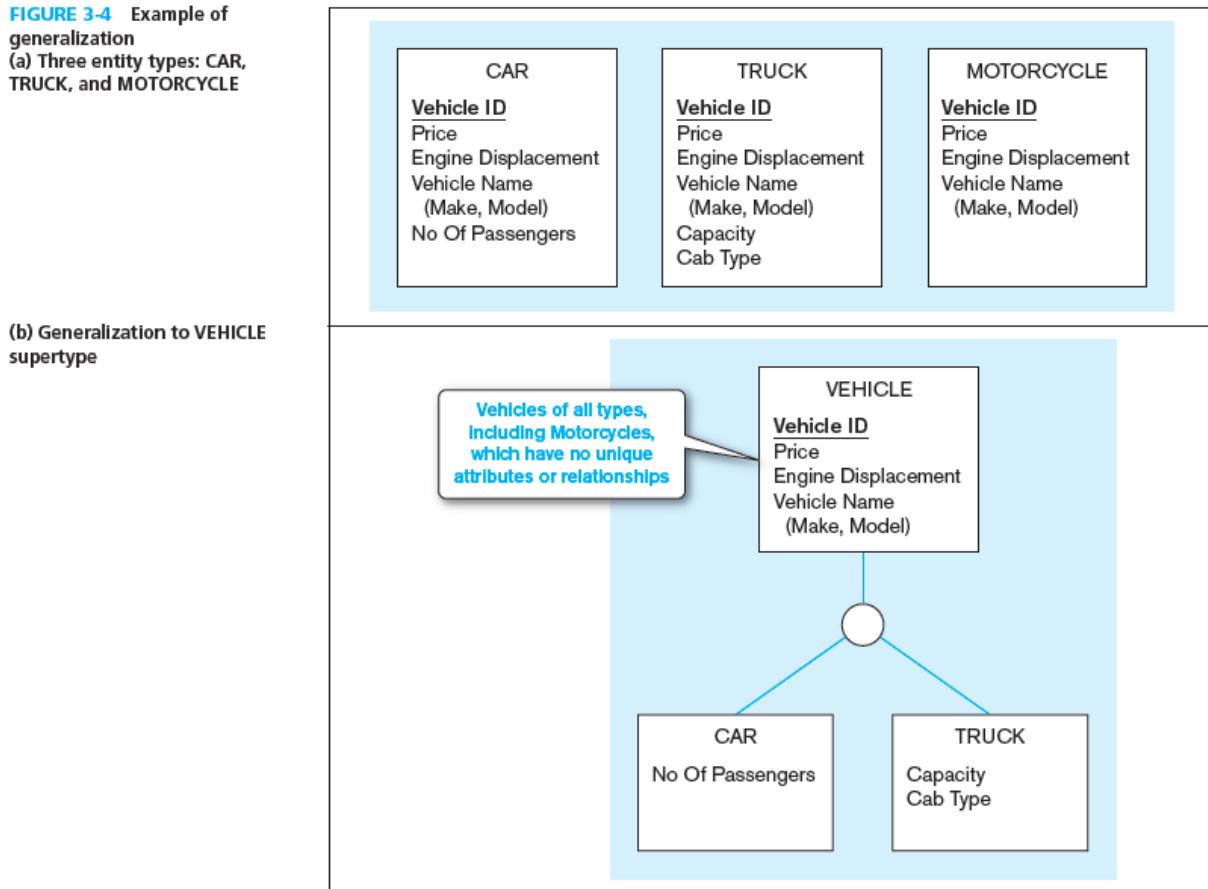
ადამიანის ინტელექტის უნიკალური ასპექტია უნარი და მიდრეკილება მოახდინოს საგნების და გამოცდილების კლასიფიცირება და მათი თვისებების განზოგადება. მონაცემთა მოდელირებისას, განზოგადება არის უფრო ზოგადი არსის ტიპის განსაზღვრის პროცესი უფრო სპეციალიზებული არსის ტიპებიდან. ამრიგად, განზოგადება არის ქვემოდან ზემოთ პროცესი. განზოგადების მაგალითი ნაჩვენებია ნახაზზე 3-4. დიაგრამა 3-4 ა-ში განისაზღვრა არსის სამი ტიპი: CAR, TRUCK და MOTORCYCLE. ამ ეტაპზე, მონაცემთა მოდელის შემუშავებელი აპირებს წარმოადგინოს ისინი ცალ-ცალკე E-R დიაგრამაზე. ამასთან, უფრო დაწვრილებითი გამოკვლევისას, ვხედავთ, რომ ამ სამი სახის არსს აქვს მრავალი საერთო ატრიბუტი: Vehicle ID, Vehicle Name (Make, Model), Price, Engine Displacement (სატრანსპორტო საშუალების ID (იდენტიფიკატორი), ავტომობილის სახელი კომპონენტებით: Make, Model), ფასი, ძრავის მოცულობა) ეს ფაქტი (განმტკიცებულია საერთო იდენტიფიკატორის არსებობით) იმაზე მეტყველებს, რომ სამი არსის ტიპიდან თითოეული უფრო ზოგადი არსის ტიპის ვერსიაა.

ეს უფრო ზოგადი არსის ტიპი (სახელწოდებით VEHICLE) წარმოქმნილი სუპერტიპი / ქვეტიპი კავშირებთან ერთად ნაჩვენებია ნახაზზე 3-4 ბ. არსს CAR აქვს კონკრეტული ატრიბუტი No Of Passenger, ხოლო TRUCK- ს აქვს ორი კონკრეტული ატრიბუტი: Capacity (ტევადობა) და Cab Type (კაბინის ტიპი). ამრიგად, განზოგადებამ საშუალება მოგვცა დავაჯგუფოთ არსთა ტიპები მათ საერთო ატრიბუტებთან ერთად და ამავე დროს შევინარჩუნოთ კონკრეტული ატრიბუტები, რომლებიც განსაკუთრებულია თითოეული ქვეტიპისთვის.

გაითვალისწინეთ, რომ არსის ტიპი MOTORCYCLE არ არის შეტანილი კავშირიში. იგი შეგნებულად არ არის შეტანილი, რადგან არ აკმაყოფილებს ადრე განხილული ქვეტიპის პირობებს. ნახაზის 3-4 a და b ნაწილების შედარებისას შევამჩნევთ, რომ MOTORCYCLE- ის ერთადერთი ატრიბუტი არის ის, რაც საერთოა ყველა მანქანისთვის; აქ არ არის მოტოციკლების სპეციფიკური ატრიბუტები. გარდა ამისა, MOTORCYCLE- ს არ აქვს კავშირი სხვა არსის ტიპთან. ამრიგად, მოტოციკლის ქვეტიპის შექმნა არ არის საჭირო.

ის ფაქტი, რომ არ არსებობს ქვეტიპი MOTOCYCLE (მოტოციკლი), იმაზე მეტყველებს, რომ შესაძლებელი იქნება აღმოჩნდეთ VEHICLE-ის (სატრანსპორტო საშუალება) ისეთი შემთხვევა, რომელიც არ იქნება მისი რომელიმე ქვეტიპის წევრი. ამ ტიპის შეზღუდვების შესახებ განვიხილავთ შეზღუდვების დაზუსტებაზე მსჯელობისასა მოგვიანებით.

**FIGURE 3-4** Example of generalization  
 (a) Three entity types: CAR, TRUCK, and MOTORCYCLE



### განკურძოება

როგორც ვნახეთ, განზოგადება არის ქვემოდან ზემოთ პროცესი. განკურძოება არის ზემოდან ქვემოთ პროცესი, განზოგადების საპირისპირო პროცესი. დავუშვათ, რომ ჩვენ განვსაზღვრეთ არის ტიპი მისი ატრიბუტებით. განკურძოება არის სუპერტიპის ერთი ან მეტი ქვეტიპის განსაზღვრისა და სუპერტიპი/ქვეტიპი კავშირების ჩამოყალიბების პროცესი. თითოეული ქვეტიპი ყალიბდება ზოგიერთი განმასხვავებელი მახასიათებლის საფუძველზე, მაგალითად, ქვეტიპისთვის დამახასიათებელი ატრიბუტების ან კავშირების საფუძველზე.

განკურძოების მაგალითი ნაჩვენებია ნახაზში 3-5. დიაგრამა 3-5 ა აჩვენებს არსის ტიპს, სახელწოდებით PART, რამდენიმე მის ატრიბუტთან ერთად. იდენტიფიკატორი არის Part No (ნაწილის N) და სხვა ატრიბუტებია: Description (აღწერა), Unit Price (ერთეულის ფასი), Location (ადგილმდებარეობა), Qty On Hand (არსებული რაოდენობა), Routing Number (მარშრუტის ნომერი), Supplier (მიმწოდებელი) (ბოლო ატრიბუტი მრავალმნიშვნელოვანი და კომპოზიტურია, რადგან შეიძლება არსებობდეს ერთზე მეტი მომწოდებელი, რომელსაც დეტალის ერთეულის თავისი ფასი აქვს).

მომხმარებლებთან დისკუსიის დროს, ჩვენ აღმოვაჩენთ, რომ ნაწილების ორი წყარო არსებობს: ზოგი იწარმოება კომპანიაში, ზოგი კი შეძენილია მომწოდებლებისგან. გარდა ამისა, ჩვენ აღმოვაჩენთ, რომ ზოგიერთი ნაწილი ორივე წყაროდანაა მიღებული. ამ შემთხვევაში, არჩევანი

დამოკიდებულია ისეთ ფაქტორებზე, როგორიცაა წარმოების მოცულობა, ნაწილების ერთეულის ფასი და ა.შ.

დიაგრამა 3-5 ა-ის შესაბამისად ზოგიერთი ატრიბუტი ვრცელდება ყველა ნაწილზე, განურჩევლად წყაროსი. ამასთან, ზოგი დამოკიდებულია წყაროზე. ამრიგად, ატრიბუტი Routing Number (მარშრუტის ნომერი) ვრცელდება მხოლოდ წარმოებულ ნაწილებზე, ხოლო ატრიბუტები Supplier ID (მიმწოდებლის ID) და Unit Price (ერთეულის ფასი) ვრცელდება მხოლოდ შეძენილ ნაწილებზე. ეს ფაქტორები მიგვანიშნებს, რომ PART (ნაწილი) უნდა იყოს განკერძოებული ქვეტიპების MANUFACTURED PART (წარმოებული ნაწილის) და PURCHASED PART (შეძენილი ნაწილი) განსაზღვრით (სურათი 3-5 ბ).

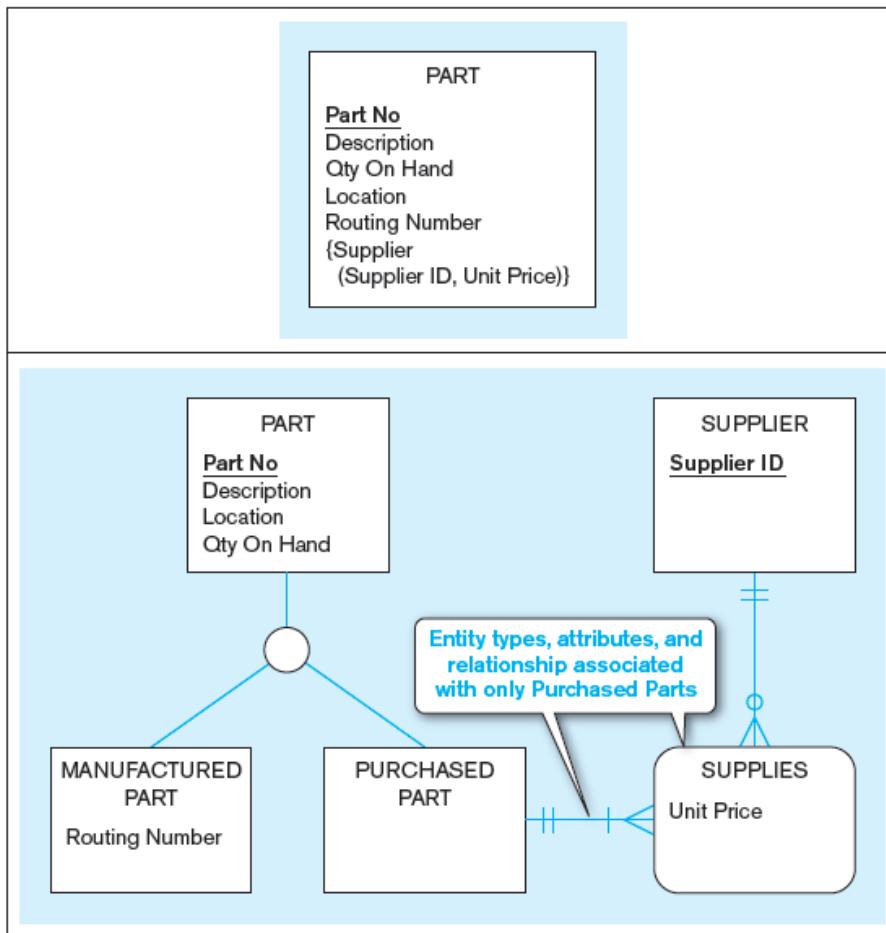
ნახაზზე 3-5 ბ, ატრიბუტი Routing Number (მარშრუტის ნომერი) დაკავშირებულია ქვეტიპთან MANUFACTURED PART (წარმოებულ ნაწილთან). მონაცემთა შემმუშავებელი თავდაპირველად აპირებდა Supplier ID (მიმწოდებლის ID)-ს და and Unit Price (ერთეულის ფასი) დაკავშირებას ქვეარსთან PURCHASED PART (შესყიდული ნაწილი). ამასთან, მომხმარებლებთან შემდგომი დისკუსიების დროს, მონაცემთა შემმუშავებელმა შესთავაზა, რომ შექმნას SUPPLIER არსის ტიპი და ასოციაციური არსი, რომელიც დააკავშირებს MANUFACTURED PART-ს SUPPLIER-ს თან. ეს ასოციაციური არსი (სახელად SUPPLIES ნახაზი 3-5 ბ) საშუალებას აძლევს მომხმარებლებს უფრო ადვილად დააკავშირონ შეძენილი ნაწილები თავიანთ მომწოდებლებთან. გავითვალისწინოთ, რომ ატრიბუტი Unit Price (ერთეულის ფასი) ახლა ასოციაციება ასოციაციურ არსთან, ასე რომ ერთეულის ფასი ნაწილისთვის შეიძლება განსხვავდებოდეს ერთი მომწოდებლისა მეორესგან. ამ მაგალითში, განკერძოებამ საშუალება მოგვცა მოგვეხდინა პრობლემის დომენის სასურველი წარმოდგენა.

#### განზოგადების და განკერძოების კომბინირება

განკერძოება და განზოგადება ორივე მაღალი დონის ტექნიკას სუპერტიპ /ქვეტიპი კავშირების შემუშავებისსათვის. ტექნიკა, რომელსაც ვიყენებთ კონკრეტულ დროს, დამოკიდებულია რამდენიმე ფაქტორზე, მაგალითად, პრობლემის საგნობრივი არის ხასიათზე, წინა მოდელირების მცდელობებზე და პირად უპირატესობაზე. ჩვენ მზად უნდა ვიყოთ, გამოვიყენოთ ორივე მიდგომა და შევცვალოთ ერთი მიმართულებიდან მეორე მიმართულებაზე, როგორც ამას ზემოთ განხილული ფაქტორები განსაზღვრავენ.

#### შეზღუდვების მინიშნება სუპერტიპი/ქვეტიპი კავშირებში

ჩვენ განვიხილეთ სუპერტიპი /ქვეტიპი კავშირების ძირითადი კონცეფციები და შემოვიტანეთ რამდენიმე ძირითადი აღნიშვნა ამ კონცეფციების წარმოსადგენად. ასევე აღვწერეთ განზოგადებისა და განკერძოების პროცესები, რაც მონაცემთა მოდელის შემმუშავებელს ეხმარება ამოიცნოს ამ კავშირების გამოყენების შესაძლებლობები. შემოვიტანოთ დამატებითი აღნიშვნები სუპერტიპი /ქვეტიპის კავშირების შეზღუდვების გამოსახატავად. ეს შეზღუდვები საშუალებას გვაძლევს დავაფიქსიროთ რამდენიმე მნიშვნელოვანი ბიზნეს წესი, რომელიც მოქმედებს ამ კავშირებზე. შეზღუდვების ორი ყველაზე მნიშვნელოვანი ტიპი, რომლებიც ხშირად გვხვდება, არის სისრულისა და თანაუკვეთობის შეზღუდვები



**FIGURE 3-5** Example of specialization  
(a) Entity type PART

(b) Specialization to MANUFACTURED PART and PURCHASED PART

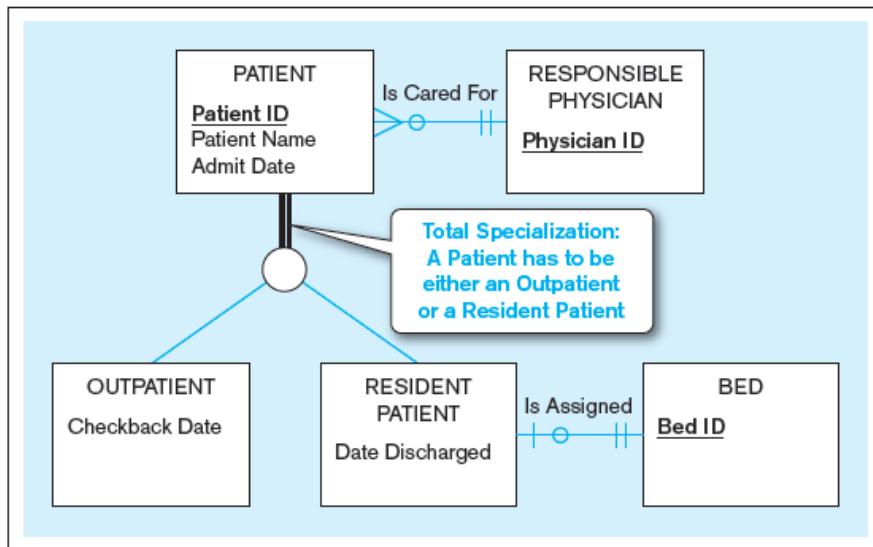
### სისრულის შეზღუდვების მინიშნება

სისრულის შეზღუდვა პასუხს სცემს კითხვას, უნდა იყოს თუ არა სუპერტიპის ეგზემპლარი თუნდაც მინიმუმ ერთი ქვეტიპის წევრი. სისრულის შეზღუდვას აქვს ორი შესაძლო წესი: სრული განკერძოება და ნაწილობრივი განკერძოება. სრული განკერძოების წესი განსაზღვრავს, რომ სუპერტიპი არსის თითოეული ეგზემპლარი უნდა იყოს კავშირის რომელიმე ქვეტიპის წევრი. ნაწილობრივი განკერძოების წესი განსაზღვრავს, რომ სუპერტიპის არსის ეგზემპლარისთვის ნებადართულია, რომ არ მიეკუთვნოს რომელიმე ქვეტიპს.

### სრული განკერძოების წესი

ნახაზი 3-6 ა იმეორებს PATIENT (პაციენტი) მაგალითს (სურათი 3-3) და შემოაქვს აღნიშვნა სრული განკერძოებისათის ამ მაგალითში ბიზნესის წესი შემდეგია: პაციენტი უნდა იყოს ამბულატორიული ან რეზიდენტი პაციენტი (ამ საავადმყოფოში პაციენტის სხვა ტიპები არ არსებობს). სრული განკერძოება აღინიშნება ორმაგი ხაზით, რომელიც ვრცელდება PATIENT (პაციენტი) არსის ტიპიდან წრემდე. Microsoft Visio აღნიშვნაში სრული განკერძოება ეწოდება "კატეგორია დასრულებულია" და ნაჩვენებია აგრეთვე ორმაგი ხაზით კატეგორიის წრის ქვეშ სუპერტიპსა და მასთან დაკავშირებულ ქვეტიპებს შორის.

**FIGURE 3-6** Examples of completeness constraints  
(a) Total specialization rule



ამ მაგალითში, ყოველ ჯერზე, როდესაც PATIENT-ის ახალი ეგზემპლარი შედის სუპერტიპში, შესაბამისი ეგზემპლარი ჩაირთვება ან OUTPATIENT (ამბულატორიულ) ან RESIDENT PATIENT (რეზიდენტ) პაციენტში. თუ ეგზემპლარი ჩასმულია RESIDENT PATIENT-ში, იქმნება კავშირის Is Assigned ეგზემპლარი პაციენტის საავადმყოფოს საწოლზე მისამაგრებლად.

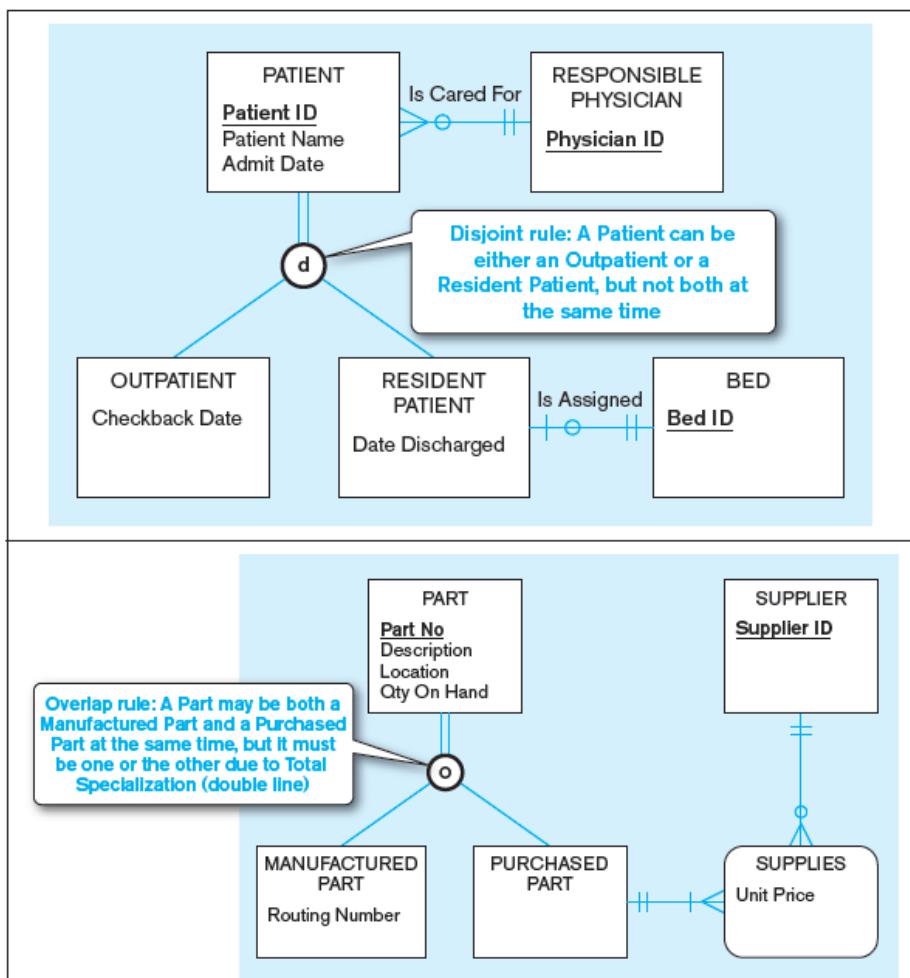
#### ნაწილობრივი განკურძოების წესი

ნახაზი 3-6 ბ იმეორებს VEHICLE-ის მაგალითს და მისი ქვეტიპებს CAR და TRUCK ნახაზი 3-4-დან. გავიხსენოთ, რომ ამ მაგალითში მოტოციკლი არის სატრანსპორტო საშუალების ტიპი, მაგრამ ის მოცემულ მოდელში არ არის წარმოდგენილი ქვეტიპის სახით. ამრიგად, თუ სატრანსპორტო საშუალება არის მანქანა, ის უნდა წარმოიშვას CAR-ის ეგზემპლარად, ხოლო თუ იგი არის სატვირთო მანქანა, ის უნდა გამოჩნდეს, როგორც TRUCK (სატვირთო ავტომობილი) ეგზემპლარი. ამასთან, თუ სატრანსპორტო საშუალება არის მოტოციკლი, ის ვერ გამოჩნდება, როგორც რომელიმე ქვეტიპის ეგზემპლარი. ეს არის ნაწილობრივი განკურძოების მაგალითი და იგი აღინიშნება ერთი ხაზით VEHICLE სუპერტიპიდან წრისკენ.

#### თანაუკვეთობის შეზღუდვების მინიშნება

თანაუკვეთობის შეზღუდვა განსაზღვრავს, შეუძლია თუ არა სუპერტიპის ეგზემპლარს ერთდროულად იყოს ორი (ან მეტი) ქვეტიპის წევრი. თანაუკვეთობის შეზღუდვას აქვს ორი შესაძლო წესი: თანაუკვეთობის წესი და გადაფარვის წესი. თანაუკვეთობის წესი განსაზღვრავს, რომ თუ არსის ეგზემპლარი (სუპერტიპის) არის ერთი ქვეტიპის წევრი, იგი ამავდროულად არ შეიძლება იყოს რომელიმე სხვა ქვეტიპის წევრი. თანაკვეთის წესი განსაზღვრავს, რომ არსის ეგზემპლარი ერთდროულად შეიძლება იყოს ორი (ან მეტი) ქვეტიპის წევრი. თითოეული ამ წესის მაგალითი ნაჩვენებია ნახაზზე 3-7.

**FIGURE 3-7** Examples of disjointness constraints  
(a) Disjoint rule



### თანაუკულეთობის წესი

დიაგრამა 3-7 ა გვიჩვენებს PATIENT (პაციენტი)-ის მაგალითს ნახაზი 3-6 ა-დან. ბიზნესის წესი ამ შემთხვევაში შემდეგია: ნებისმიერ დროს, პაციენტი უნდა იყოს ამბულატორიული ან რეზიდენტი პაციენტი, მაგრამ არ შეიძლება იყოს ორივე ერთდროულად. ეს არის თანაუკულეთობის წესი, რომელიც ეს მითითებულია სუპერტიპთან და მის ქვეტიპებთან შერთებულ წრეში d ასოით. გავითვალისწინოთ, რომ ამ ფიგურაში, PATIENT-ის ქვეკლასი შეიძლება შეიცვალოს დროთა განმავლობაში, მაგრამ მოცემულ დროს, PATIENT მხოლოდ ერთი ტიპისაა.

### გადაფარვის წესი

დიაგრამა 3-7 ბ აჩვენებს არსის ტიპს PART თავისი ორი ქვეტიპით, MANUFACTURED PART (წარმოებული ნაწილი) და PURCHASED PART (ნაყიდი ნაწილი) (ნახაზი 3-5 ბ). გავიხსენოთ მაგალითის განხილვიდან, რომ ზოგიერთი ნაწილი დამზადებულია ცაა და შეძენილიც. ამ მტკიცებას გარკვეული განმარტება ესაჭიროება. ამ მაგალითში, PART- ის ეგზემპლარი არის ცალკეული ნაწილის რაოდენობა (ანუ ნაწილის ტიპი) და არა ცალკეული ნაწილი (მითითებულია იდენტიფიკატორით, რომელიც არის ნაწილი No). მაგალითად, განვიხილოთ

ნაწილის ნომერი 4000. მოცემულ დროს, ამ ნაწილის ხელთ არსებული რაოდენობა შეიძლება იყოს 250, საიდანაც 100 დამზადებულია და დანარჩენი 150 შეძენილი ნაწილებია. ამ შემთხვევაში არ არის მნიშვნელოვანი ცალკეული ნაწილების თვალყურისდევნება. როდესაც ცალკეული ნაწილების თვალყურისდევნება მნიშვნელოვანია, თითოეულ ნაწილს ენიჭება სერიული ნომრის იდენტიფიკატორი, ხოლო რაოდენობა არის ერთი ან ნულოვანი, დამოკიდებულია იმაზე, არსებობს თუ არა ეს ცალკეული ნაწილი.

გადაფარვის წესი მიეთითება წრეში ასო 0-ს განთავსებით, როგორც ეს ნაჩვენებია ნახაზზე 3-7 ბ. გავითვალისწინოთ, რომ ამ ნახაზში ასევე მითითებულია სრული განკერძოების წესი, რასაც ორმაგი ხაზი მიუთითებს. ამრიგად, ნებისმიერი ნაწილი უნდა იყოს შეძენილი ან წარმოებული ნაწილი, ან შეიძლება ერთდროულად იყოს ორივე.

## ქვეტიპის დისკრიმინატორების განსაზღვრა

სუპერტიპი/ქვეტიპი კავშირის გათვალისწინებით, განვიხილოთ სუპერტიპის ახალი ეგზემპლარის ჩასმის პრობლემა. რომელ ქვეტიპში (ასეთის არსებობის შემთხვევაში) უნდა იყოს ჩასმული ეს ეგზემპლარი? ჩვენ უკვე განვიხილეთ სხვადასხვა შესაძლო წესები, რომლებიც მოქმედებს ამ სიტუაციაში. საჭიროა მარტივი მექანიზმი ამ წესების განსახორციელებლად, თუ ეს ხელმისაწვდომია. ხშირად ეს შეიძლება განხორციელდეს ქვეტიპის დისკრიმინატორის გამოყენებით. ქვეტიპის დისკრიმინატორი არის სუპერტიპის ატრიბუტი, რომლის მნიშვნელობები განსაზღვრავს სამიზნე ქვეტიპს ან ქვეტიპებს.

### თანაუკუთხი ქვეტიპები

ქვეტიპის დისკრიმინატორის გამოყენების მაგალითი ნაჩვენებია სურათი 3-8. ეს მაგალითია EMPLOYEE სუპერტიპისა და მისი ქვეტიპებისათვის, რომლებიც მოცემულია ნახაზზე 3-2. გავითვალისწინოთ, რომ ამ მაჩვენებელს დაემატა შემდეგი შეზღუდვები: სრული განკერძოება და თანაუკუთხი ქვეტიპები. ამრიგად, თითოეული თანამშრომელი უნდა იყოს საათობრივი, ხელფასიანი ან კონსულტანტი.

სუპერტიპის დაემატა ახალი ატრიბუტი Employee Type (თანამშრომლის ტიპი), რომელიც ემსახურება ქვეტიპის დისკრიმინატორს. როდესაც სუპერტიპს დაემატება ახალი თანამშრომელი, ეს ატრიბუტი იშიფრება სამი მნიშვნელობიდან ერთში, შემდეგნაირად: "H" (საათობრივი), "S" (ხელფასიანი) ან "C" (კონსულტანტისთვის). ამ კოდის მიხედვით, შემდეგ ეგზემპლარი ენიჭება შესაბამის ქვეტიპს.

აღნიშვნა, რომელსაც ვიყენებთ ქვეტიპის დისკრიმინატორის დასაზუსტებლად, ასევე ნაჩვენებია ნახაზზე 3-8. გამოხატვა Employee Type = (რომელიც წარმოადგენს პირობით ოპერატორის მარცხენა მხარს) განთავსებულია სუპერტიპიდან წრისკენ მიმავალ ხაზთან. ატრიბუტის მნიშვნელობა, რომელიც აირჩევს შესაბამის ქვეტიპს (ამ მაგალითში, ან "H", "S" ან "C") მოთავსებულია ამ ქვეტიპისკენ მიმავალი ხაზის მიმდებარედ. მაგალითად, პირობა Employee Type = "S" გამოიწვევს ეგზემპლარის ჩასმას SALARIED EMPLOYEE ქვეტიპში.

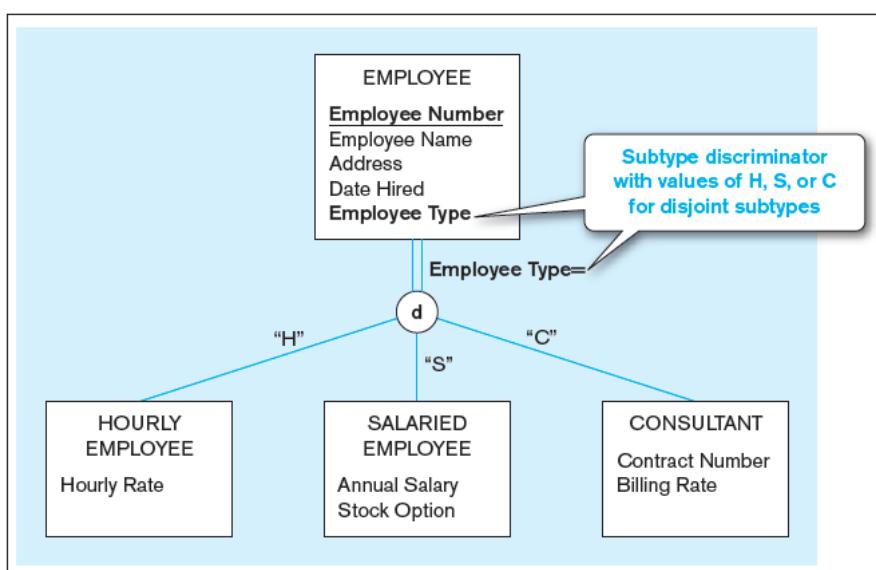


FIGURE 3-8 Introducing a subtype discriminator (disjoint rule)

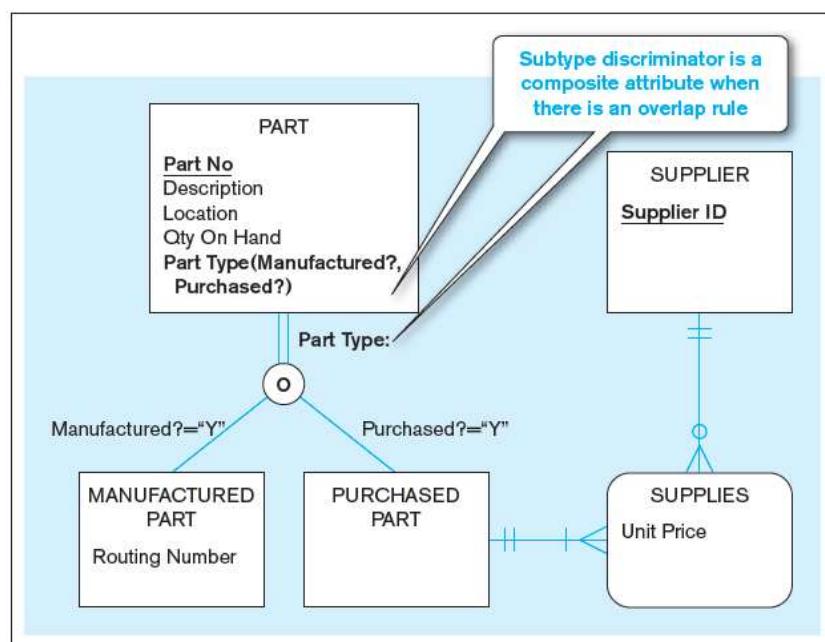
## გადამფარი ქვეტიპები

ქვეტიპების გადაფარვისას ქვეტიპის დისკრიმინატორისთვის ოდნავ შეცვლილი მიღებული უნდა იქნას გამოყენებული. მიზეზი ისაა, რომ სუპერტიპის მოცემულმა ეგზემპლარმა შეიძლება მოითხოვოს ეგზემპლარის შექმნა ერთზე მეტ ქვეტიპში. ამ სიტუაციის მაგალითი ნაჩვენებია ნახაზზე 3-9 ნაწილში PART და მისი გადაფარვითი ქვეტიპები. PART -ს დაემატა ახალი ატრიბუტი სახელწოდებით Part Type. ნაწილის ტიპი არის კომპოზიტური ატრიბუტი კომპონენტებით Manufactured? (დამზადებულია?) და Purchased? (შეძენილია?). თითოეული ეს ატრიბუტი არის ლოგიკური ცვლადი (ანუ იღებს მხოლოდ მნიშვნელობებს „დიახ“ - "Y" და „არა“ - "N"). როდესაც ნაწილს დაემატება ახალი ეგზემპლარი ეს კომპონენტები კოდირდება შემდეგნაირად:

Type of Part	Manufactured?	Purchased?
Manufactured only	"Y"	"N"
Purchased only	"N"	"Y"
Purchased and manufactured	"Y"	"Y"

ამ მაგალითისთვის ქვეტიპის დისკრიმინატორის მითითების მეთოდი ნაჩვენებია ნახაზზე 3-9. გავითვალისწინოთ, რომ ეს მიღებული შეიძლება გამოყენებულ იქნას ნებისმიერი რაოდენობის გადაფარვითი ქვეტიპისთვის.

**FIGURE 3-9** Subtype discriminator (overlap rule)



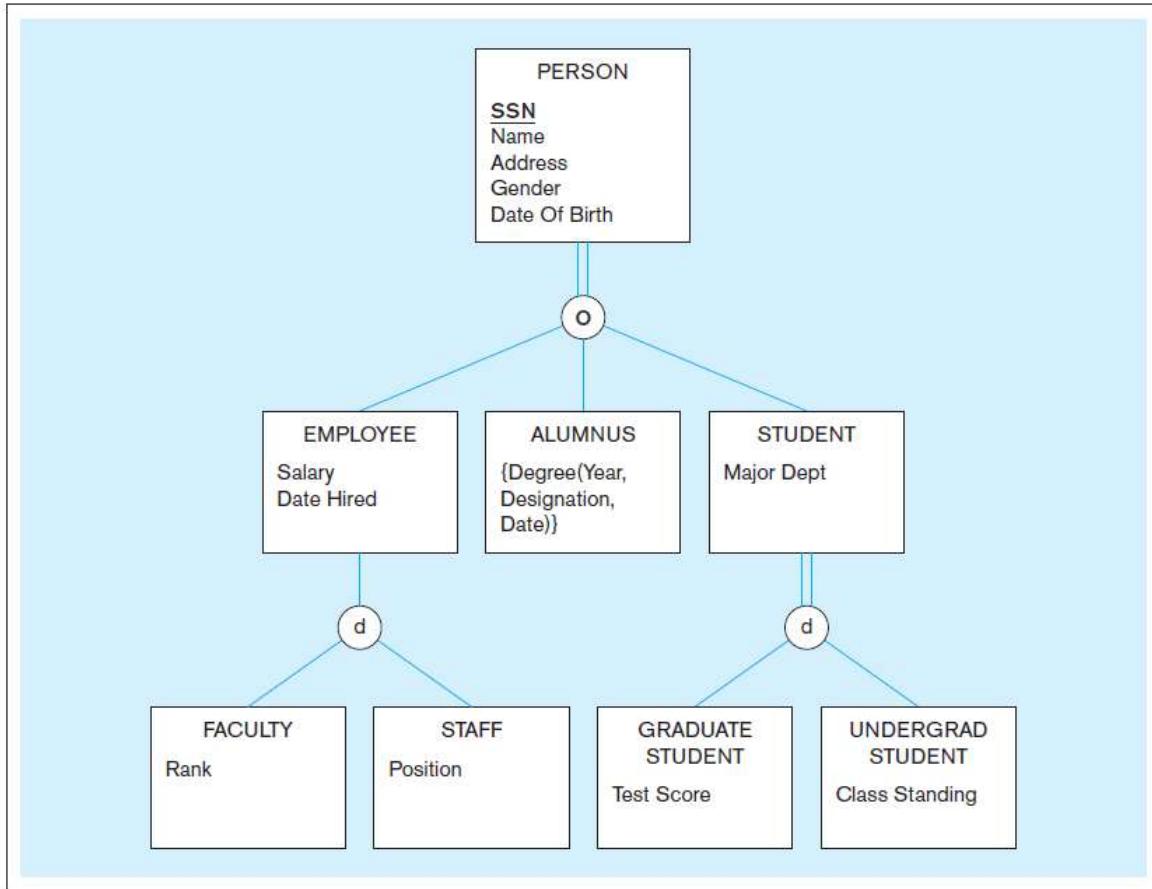
## სუპერტიპი / ქვეტიპი იერარქიების განსაზღვრა

ჩვენ განვიხილეთ სუპერტიპი/ქვეტიპი კავშირების არაერთი მაგალითი. ამ მაგალითებში ნებისმიერი ქვეტიპისთვის შესაძლებელია განსაზღვრული იყოს სხვა ქვეტიპი (ამ შემთხვევაში, ქვეტიპი ხდება სუპერტიპი ახლად განსაზღვრული ქვეტიპისთვის).

სუპერტიპი/ქვეტიპი იერარქია არის სუპერტიპისა და ქვეტიპის იერარქიული მოწყობა, სადაც თითოეულ ქვეტიპს აქვს მხოლოდ ერთი სუპერტიპი.

წარმოგიდგნენთ სუპერტიპი/ქვეტიპი იერარქიის მაგალითი ნახაზით 3-10.

**FIGURE 3-10 Example of supertype/subtype hierarchy**



### სუპერტიპი/ქვეტიპი იერარქიის მაგალითი

დავუშვათ, რომ მოგვეთხოვება ადამიანური რესურსების მოდელირება უნივერსიტეტში. განკერძოების გამოყენებით (მიღომა ზემოდან ქვემოთ) შეგვიძლია შემდეგნაირად მოვიქცეთ: დავიწყოთ იერარქიის წვეროთი, ჯერ შევქმნათ ყველაზე ზოგადი არსის ტიპის მოდელი. ამ შემთხვევაში, არსის ყველაზე ზოგადი ტიპია PERSON. ჩამოვთვალოთ და დავაკავშიროთ PERSON-ის ყველა ატრიბუტი. 3-10 ნახაზზე ნაჩვენები ატრიბუტებია SSN (identifier), Name, Address, Gender, and Date Of Birth. The (SSN (იდენტიფიკატორი), სახელი, მისამართი, სქესი და დაბადების თარიღი). იერარქიის ზედა ნაწილში არსებულ ტიპს ზოგჯერ ფესვს უწოდებენ.

შემდეგ განვსაზღვროთ ფესვის ყველა ძირითადი ქვეტიპი. ამ მაგალითში, PERSON-ის სამი ქვეტიპი არსებობს: EMPLOYEE (თანამშრომელი) - უნივერსიტეტში მომუშავე პირები, STUDENT (სტუდენტი) - ვინც მეცადინეობებს ესწრება, და ALUMNUS (კურსდამთავრებულები) - დამთავრებული. თუ ჩავთვლით, რომ უნივერსიტეტისთვის სხვა სახის პირები არ არსებობენ, მოქმედებს სრული განვერძოების წესი, როგორც ეს ნაჩვენებია ნახატზე. PERSON შეიძლება მიეკუთვნებოდეს ერთზე მეტ ქვეტიპს (მაგალითად, ALUMNUS და EMPLOYEE), ამიტომ გამოიყენება გადაფარვის წესი. გავითვალისწინოთ, რომ გადაფარვა ნებისმიერი გადაფარვის საშუალებას იძლევა

(PERSON შეიძლება ერთდროულად იყოს ნებისმიერ წყვილში ან სამივე ქვეტიპში). თუ გარკვეული კომბინაციები არ არის დაშვებული, აკრძალული კომბინაციების აღმოსაფხვრელად უნდა შეიქმნას უფრო დახვეწილი სუპერტიპი/ქვეტიპი იერარქია.

ატრიბუტები, რომლებიც კონკრეტული ქვეტიპისთვის გამოიყენება, ნაჩვენებია ნახაზზე. ამრიგად, EMPLOYEE-ს თითოეულ ეგზემპლარს აქვს მნიშვნელობა Date Hired (დაქირავების თარიღი) და ხელფასი (Salary). Major Dept (მირითადი დეპარტამენტი) STUDENT- ის ატრიბუტია, ხოლო Degree (ხარისხი) (კომპონენტებით Year, Designation და Date წელი, დანიშნულება და თარიღი) არის ALUMNUS- ის მრავალმნიშვნელოვანი კომპოზიტიური ატრიბუტი.

შემდეგი ნაბიჯით უნდა შეფასდეს გამოდგება თუ არა უკვე განსაზღვრული რომელიმე ქვეტიპი შემდგომი განკურძოებისათვის. ამ მაგალითში, EMPLOYEE იყოფა ორ ქვეტიპად: FACULTY (აკადემიური პერსონალი) და STAFF (ადმინისტრაციული პერსონალი). FACULTY-ს აქვს სპეციფიკური ატრიბუტის Rank (რანგი), ხოლო STAFF-ს აქვს სპეციფიკური ატრიბუტი Position (პოზიცია). გავითვალისწინოთ, რომ ამ მაგალითში EMPLOYEE ქვეტიპი ხდება სუპერტიპი FACULTY-ისა და STAFF-თვის. იმის გამო, რომ შეიძლება არსებობდეს თანამშრომლების ტიპები, გარდა FACULTY და STAFF (მაგალითად, სტუდენტის თანაშემწები), მიეთითება ნაწილობრივი განკურძოების წესი. ამასთან, თანამშრომელი ერთდროულად არ შეიძლება იყოს როგორც FACULTY ასევე STAFF. ამიტომ წრეში მითითებულია თანაუკვეთობის წესი.

ასევე განისაზღვრება ორი ქვეტიპი სტუდენტისთვის: UNDERGRAD STUDENT (მაგისტრანტი) და GRADUATE STUDENT (ბაკალავრიატის სტუდენტი). UNDERGRAD STUDENT- ს აქვს ატრიბუტი Class Standing, ხოლო GRADUATE STUDENT- ს აქვს ატრიბუტი Test Score. გავითვალისწინოთ, რომ მითითებულია სრული განკურძოება და თანაუკვეთობის წესი; ჩვენ უნდა შეგვეძლოთ ამ შეზღუდვებისთვის ბიზნეს წესების მითითება.

### [სუპერტიპი/ქვეტიპი იერარქიის შეჯამება](#)

ჩვენ აღვნიშნავთ იერარქიაში მოცემულ ატრიბუტების ორი მახასიათებლის შესახებ, რომლებიც ნაჩვენებია ნახაზზე 3-10:

1. ატრიბუტები მიეთითება იერარქიაშია რაც შესაძლებელია უმაღლეს ლოგიკურ დონეზე (საფეხურზე ზემოდან). მაგალითად, იმის გამო, რომ SSN (ანუ სოციალური დაცვის ნომერი) ვრცელდება ყველა პირზე, იგი ენიჭება ფესვს. ამის საპირისპიროდ, Date Hired ეხება მხოლოდ თანამშრომლებს, ამიტომ იგი ენიჭება EMPLOYEE-ს. ეს მიდგომა უზრუნველყოფს ატრიბუტების გაზიარებას რაც შეიძლება მეტი ქვეტიპისათვის.
2. ქვეტიპები, რომლებიც უფრო დაბალია იერარქიაში, ატრიბუტებს მემკვიდრეობით მიიღებენ არა მხოლოდ მათი უშუალო სუპერტიპიდან, არამედ იერარქიის ყველა სუპერტიპიდან, ფესვამდე. მაგალითად, FACULTY-ის ეგზემპლარს აქვს მნიშვნელობები ყველა შემდეგი ატრიბუტისთვის: SSN, სახელი, მისამართი, სქესი და დაბადების თარიღი (PERSON -ისგან); დაქირავების თარიღი და ხელფასი (EMPLOYEE იდან); და რანგი (FACULTY იდან).

### [არსის კლასტერი](#)

საწარმოს მასშტაბით არსებულ ზოგიერთ ინფორმაციულ სისტემას აქვს 1000-ზე მეტი არსის ტიპი და კავშირი. ორგანიზაციული მონაცემების ასეთი მოცულობის წარმოდგენისათვის არსებობს ერთერთი მიდგომა - ფუნქციონალური დაშლის

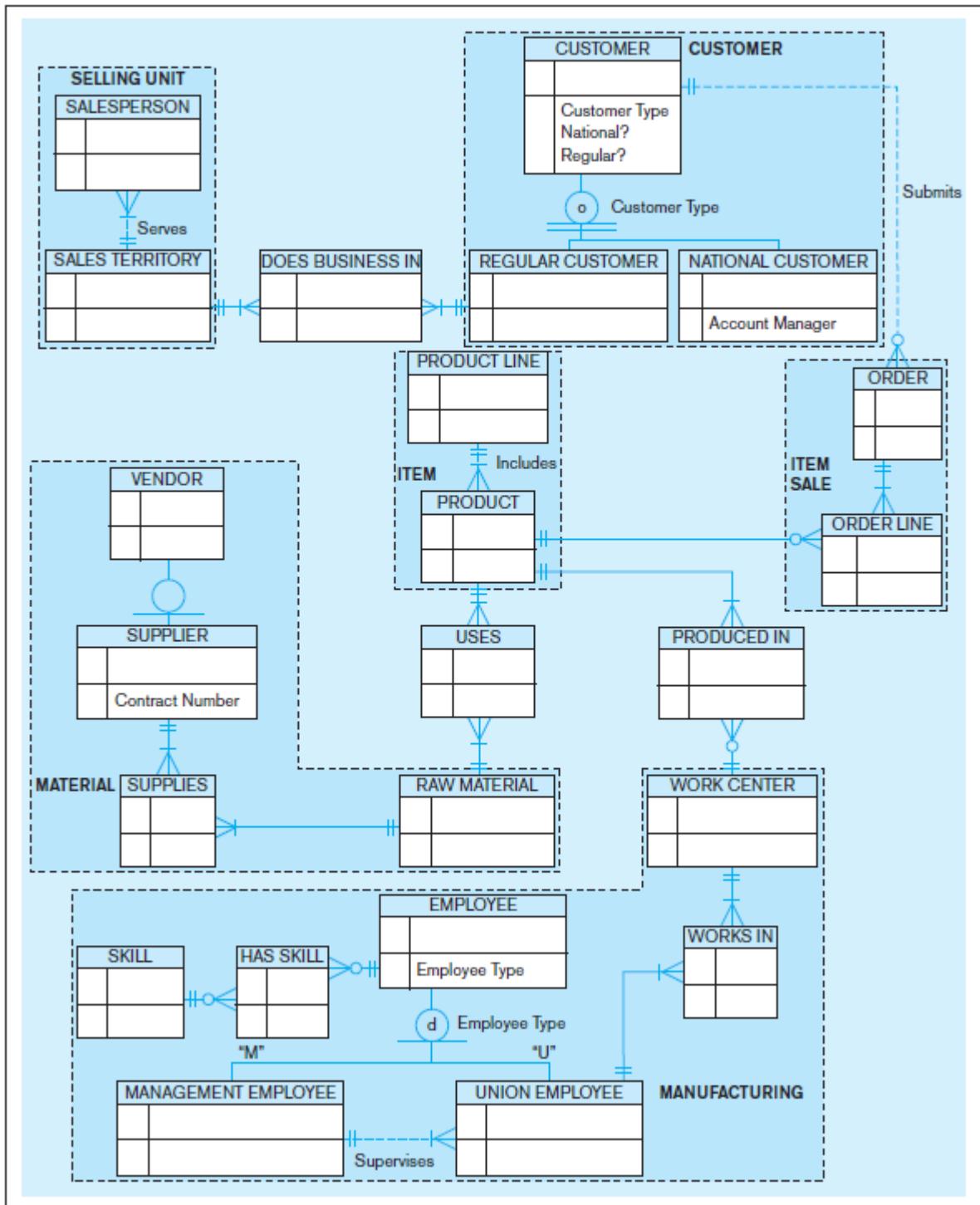
(დეკომპოზიციის) კონცეფცია. ფუნქციონალური დეკომპოზიცია არის განმეორებადი მიღებოდა სისტემის დაყოფაზე შესაბამის კომპონენტებად, ისე რომ თითოეული კომპონენტის შექმნა შესაძლებელი იყოს თავისთავად, სხვა კომპონენტებთან კავშირის განადგურების გარეშე. ფუნქციური დეკომპოზიცია ეფექტურია, რადგან ის ამარტივებს დიზაინის შეცვლას და საშუალებას აძლევს ადამიანებს ყურადღება მიაქციონ სისტემის იმ ნაწილზე, რაშიც ისინი დაინტერესებული არიან. მონაცემთა მოდელირებისას მსგავსი მრავალი მიღებოდა დაკავშირებული ER დიაგრამების შექმნასთან, თითოეულში მოცემულია მონაცემთა მოდელის სხვადასხვა (შესაძლოა გადაფარული) სეგმენტების ან ქვეჯუფების დეტალები (მაგ., სხვადასხვა სეგმენტები, რომლებიც ვრცელდება სხვადასხვა დეპარტამენტებზე, ინფორმაციული სისტემის პროგრამებში, ბიზნეს პროცესებში ან კორპორატიული განყოფილებები).

არსის კლასტერირება მოსახერხებელი მეთოდია დიდი და რთული ორგანიზაციის მონაცემების მოდელების წარმოსადგენად. არსების კლასტერი არის ერთ ან მეტი არსების ტიპიებისა და მასთან დაკავშირებული კავშირების ერთობლიობა, დაჯგუფებული ერთ აბსტრაქტული ტიპების არსში. იმის გამო, რომ არსის კლასტერი იქცევა, როგორც არსის ტიპი, არსის კლასტერები და არსის ტიპები შეიძლება შემდგომი დაჯგუფდეს უფრო მაღალი დონის არსების კლასტერად. არსის კლასტერიზაცია არის მონაცემთა მოდელის მარო დონის წარმოდგენის იერარქიული დაშლა უფრო დახვეწილ წარმოდგენებად, რის შედეგადაც მონაცემთა სრული, დეტალური მოდელი მიიღება.

დიაგრამა 3-13 ასახავს Pine Valley ავეჯის კომპანიის მოდელის არსების კლასტერიზაციის ერთ შესაძლო შედეგს. დიაგრამა 3-13a აჩვენებს მონაცემთა სრულ მოდელს არსების შესაძლო კლასტერების შემოფარგლული არეებით; დიაგრამა 3-13 ბ გვიჩვენებს დეტალური EER დიაგრამის გარდაქმნის საბოლოო შედეგს, რომელიც წარმოდგენილია მხოლოდ არსების კლასტერებისა და კავშირების შემცველ EER დიაგრამად (EER დიაგრამა შეიძლება მოიცავდეს როგორც არსების კლასტერებს, ასევე არსების ტიპებს, მაგრამ ეს დიაგრამა მოიცავს მხოლოდ არსების კლასტერებს).

- SELLING UNIT (გაყიდვის ნაწილი) არსი წარმოადგენს SALESPERSON და SALES TERRITORY არსის ტიპების და Serves (ემსახურება) კავშირის;
- CUSTOMER (მომხმარებელი) წარმოადგენს CUSTOMER (მომხმარებელთა) სუპერტიპს, მის ქვეტიპებს და სუპერტიპსა და ქვეტიპებს შორის კავშირის;
- ITEM SALE (ნივთების გაყიდვა) წარმოადგენს ORDER არსის ტიპს და ORDER LINE ასოციაციურ ერთეულს, ასევე მათ შორის კავშირის;
- ITEM წარმოადგენს PRODUCT LINE და PRODUCT არსის ტიპებს და ასევე კავშირებს Includes (მოიცავს);
- MANUFACTURING (წარმოება) წარმოადგენს WORK CENTER-ის (სამუშაო ცენტრი) და EMPLOYEE (თანამშრომელთა) სუპერტიპის არსს და მის ქვეტიპებს, აგრეთვე Works in ასოციაციურ არსს და აკონტროლებს კავშირებს და სუპერტიპსა და მის ქვეტიპებს შორის კავშირის (სურათი 3-14 ნაჩვენებია MANUFACTURING არსის დაყოფა კომპონენტებად).
- MATERIAL (მასალა) წარმოადგენს RAW MATERIAL და VENDOR არსის ტიპებს, SUPPLIER ქვეტიპს, Supplies ასოციაციურ არსს და სუპერტიპი/ქვეტიპის კავშირის VENDOR და SUPPLIER შორის.

**FIGURE 3-13** Entity clustering for Pine Valley Furniture Company  
 (a) Possible entity clusters (using Microsoft Visio)



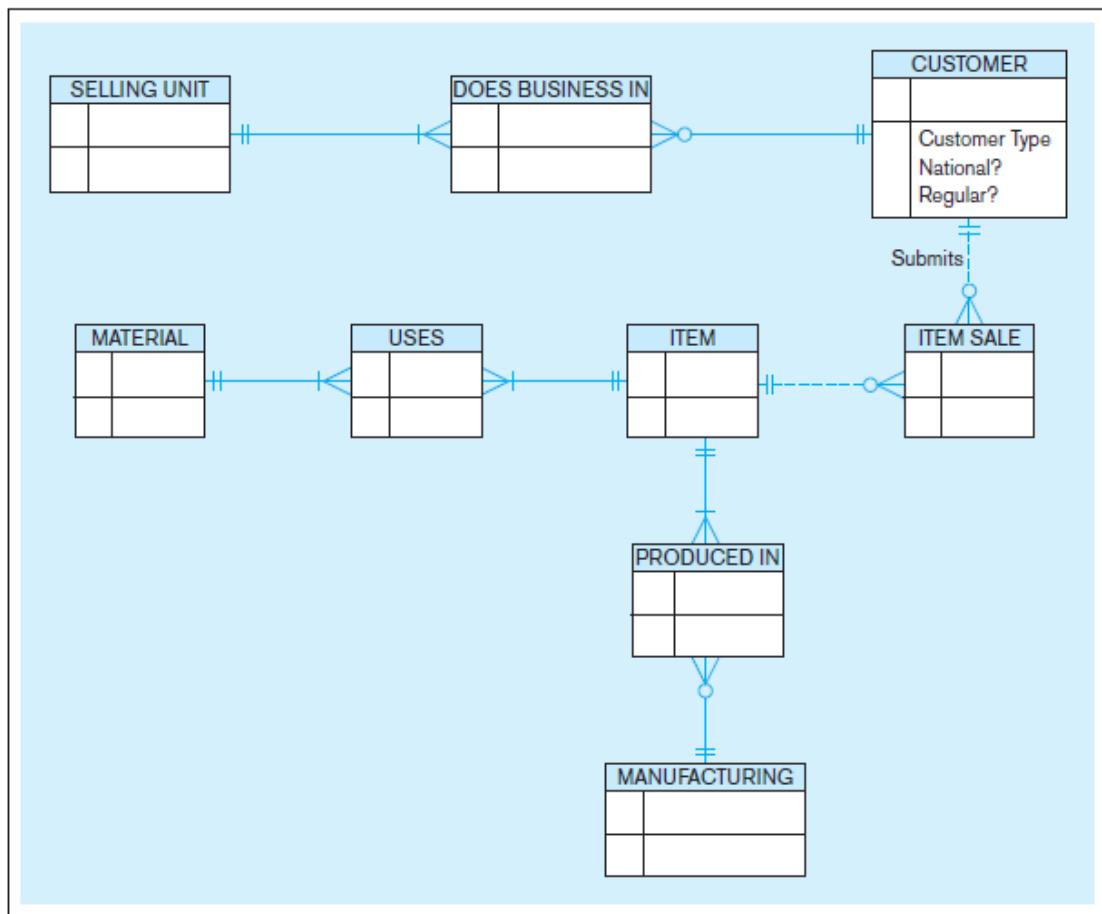
დიაგრამა 3-13-ში არსების კლასტერები ჩამოყალიბდა (1) სუპერტიპისა და მისი ქვეტიპის აბსტრაქციით (იხ. CUSTOMER არსის კლასტერი) და (2) პირდაპირ დაკავშირებული არსის ტიპებისა და მათი კავშირების კომბინირებით ( SELLING UNIT, ITEM, MATERIAL, და

MANUFACTURING არსების კლასტერები). არსის კლასტერი ასევე შეიძლება ჩამოყალიბდეს ძლიერი არსის და მასთან დაკავშირებული სუსტი არსის ტიპების

შერწყმით. იმის გამო, რომ არსების კლასტერიზაცია იერარქიულია, თუკი სასურველი იქნებოდა, შეგვეძლო შეგვედგინა კიდევ ერთი EER დიაგრამა, რომელშიც გავაერთიანებთ SELLING UNIT და CUSTOMER არსების კლასტერებს DOES BUSINESS IN ასოციაციურ არსში ერთი არსების კლასტერად, რადგან ის უშუალოდ დაკავშირებულ არსის კლასტერს წარმოადგენს.

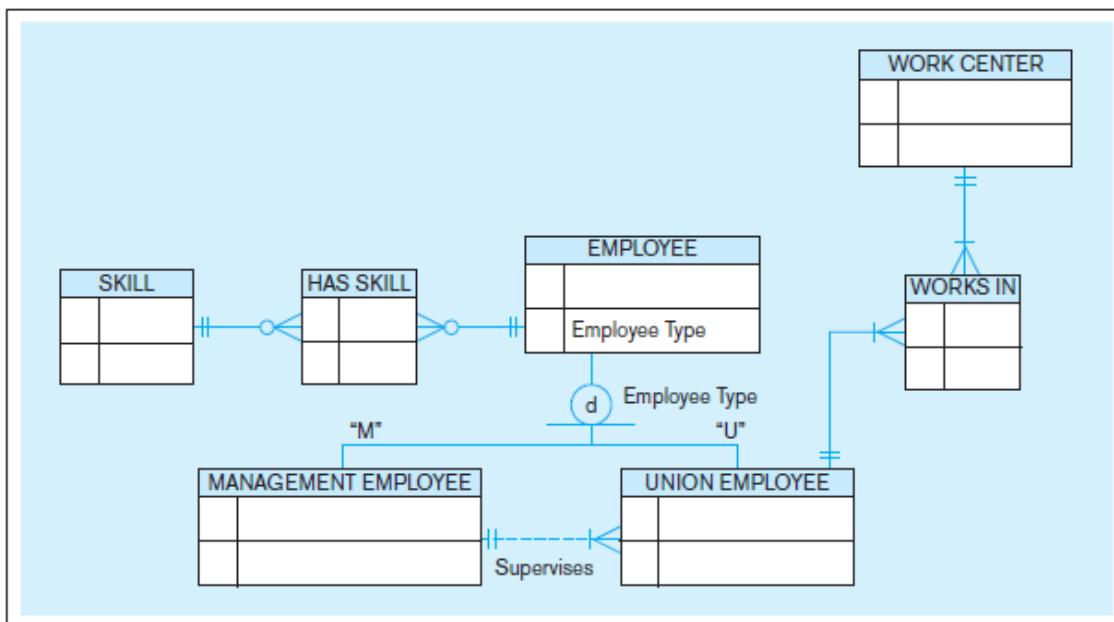
**FIGURE 3-13 (continued)**

(b) EER diagram for entity clusters (using Microsoft Visio)



არსის კლასტერი ფოკუსირებული უნდა იყოს მომხმარებლების, შემმუშავებლის ან მენეჯერების ინტერესის სფეროზე. არსის ტიპების და კავშირების დაჯგუფება, არსის კალასტერის შესაქმნელად, დამოკიდებულია ჩვენს მიზანზე. მაგალითად, ORDER არსის ტიპი შეიძლება დაჯგუფდეს CUSTOMER არსის კლასტერთან და ORDER LINE არსის ტიპი შეიძლება დაჯგუფდეს ITEM არსის კლასტერთან, Pine Valley ავეჯის მონაცემების მოდელის არსების კლასტერის მაგალითზე. ეს დაჯგუფება გამოაგდება ITEM SALE კლასტერს, რომელიც შესაძლოა არ იყოს საინტერესო მომხმარებელთა არცერთი ჯგუფისთვის. ასევე, შეგვიძლია მონაცემთა სრულ მოდელში გააკეთოთ არსების რამდენიმე განსხვავებული კლასტერიზაცია, რომელთაგან თითოეულს განსხვავებული ფოკუსირება ექნება.

FIGURE 3-14 MANUFACTURING entity cluster



### შეფუთული (დაფასოებული) მონაცემების მოდელები

”შეფუთულიც (დაფასოებული) მონაცემების მოდელი“ (მისი შექმნა) ერთ-ერთი მთავარი სტრატეგიული შენაძენია ორგანიზაციებისათვის სწრაფი შედეგებისა და გრძელვადიანი წარმატების მისაღწევად ბიზნესში. შეფუთული მონაცემების მოდელები ცვლის „თამაშის წესებს“ მონაცემთა მოდელირებაში.

მონაცემთა მოდელირების პროექტის დასაწყებად სულ უფრო პოპულარული ხდება მიდგომა, რომელიც გულისხმობს პაკეტური ან წინასწარ განსაზღვრული მონაცემების მოდელის ან ე.წ. უნივერსალური მოდელის ან ინდუსტრიის სპეციფიკური მოდელის მიღებას (ზოგიერთი პროვადერი ამას ლოგიკურ მონაცემთა მოდელს უწოდებს [LDM], მაგრამ ეს სინამდვილეში არის EER დიაგრამები. მონაცემთა მოდელი შეიძლება ასევე იყოს შეძენილი პროგრამული პაკეტის ნაწილი, როგორიცაა საწარმოს რესურსების დაგეგმვა ან მომხმარებელთან ურთიერთობის მართვის სისტემა). შეფუთული მონაცემების ეს მოდელები არაა ფიქსირებული; უფრო მეტიც, მონაცემთა მოდელის შემმუშავებლის მიერ ხდება წინასწარ განსაზღვრული მოდელის მორგება ორგანიზაციის ბიზნეს წესებზე, ინდუსტრიის (მაგ. ტრანსპორტირება ან კომუნიკაციები) ან შერჩეული ფუნქციონალური სფეროს (მაგ., ფინანსები ან წარმოება) მონაცემების მოდელების საუკეთესო პრაქტიკის საფუძველზე. მონაცემების მოდელირების ამ მიდგომის მთავარი დაშვებაა, ის რომ იგივე ინდუსტრიის ან ფუნქციონალური სფეროს საწარმოების ძირითადი სტრუქტურები ან მოდელები მსგავსია. შეფუთული მონაცემების მოდელები ხელმისაწვდომია სხვადასხვა კონსულტანტებისა და მონაცემთა ბაზის ტექნოლოგიის მომწოდებლებისგან. მიუხედავად იმისა, რომ შეფუთული მონაცემთა მოდელები იაფი არ არის, ბევრი თვლის, რომ საერთო ღირებულება დაბალია და მონაცემთა მოდელირების ხარისხი უკეთესია ასეთი რესურსების გამოყენებისას.

მონაცემთა უნივერსალური მოდელი არის მონაცემთა ზოგადი ან შაბლონური მოდელი, რომელიც შეიძლება გამოყენებულ იქნეს, როგორც მონაცემთა მოდელირების პროექტის ათვლის წერტილი. ზოგიერთი ადამიანი ამ მონაცემების მოდელს შაბლონებს უწოდებს, მსგავსად დაპროგრამების კოდის მრავალჯერადი გამოყენების შაბლონების ცნებისა. მონაცემთა უნივერსალური მოდელი არ არის ”სწორი“ მონაცემთა მოდელი, მაგრამ ის

წარმატებული ათვლის წერტილია ორგანიზაციისთვის მონაცემთა შესანიშნავი მოდელის შესაქმნელად.

მონაცემთა მოდელირების პროექტის დაწყება მონაცემების უნივერსალური მოდელით ბოლო პერიოდში პოპულარული გახდა. ყველაზე დამაჯერებელი მიზეზი იმისა, თუ რატომ იყენებენ ამ მიდგომას პროფესიონალური მონაცემთა მოდელების შემმუშავებლები შეიძლება ასე განმარტოთ:

- მონაცემთა მოდელების შემუშავება შესაძლებელია დაგროვილი გამოცდილების საფუძველზე შექმნილი და შემოწმებული კომპონენტების გამოყენებით. ამ მონაცემთა მოდელების განახლება ხდება მომწოდებლის მიერ, ინდუსტრიაში მონაცემთა ახალი სახეობების აღიარების შესაბამისად (მაგალითად, RFID);
- პროექტები ნაკლებ დროს და ხარჯებს საჭიროებენ, რადგან ძირითადი კომპონენტები და სტრუქტურები უკვე განსაზღვრულია და საჭიროა მხოლოდ სწრაფად მორგება კონკრეტულ სიტუაციაში.
- მონაცემთა მოდელებში მნიშვნელოვანი კომპონენტების გამოტოვების ან მოდელური შეცდომების დაშვების ნაკლები აღბათობაა, რომელიც შეიძლება გამოიწვიოს საერთო შესაძლებლობების გაუცნობიერებამ.
- მოცემული მონაცემების მოდელი კონკრეტული საწარმოსთვის უფრო ადვილად ვითარდება, რადგან მოცემული სიტუაციისთვის დადგენილია მონაცემთა დამატებითი მოთხოვნები. შეძენილი მოდელი ამცირებს სამომავლო დამუშავებას, რადგან პაკეტი ასწორებს მას და ითვალისწინებს სამომავლო საჭიროებებს.
- არსებული მონაცემთა ბაზის მონაცემთა მოდელები ადვილად იკითხება მონაცემთა შემმუშავებლებისა და მონაცემთა მართვის სხვა პროფესიონალების მიერ, რადგან ისინი ეფუძნება მსგავს სიტუაციებში დანახულ საერთო კომპონენტებს.
- სუპერტიპი/ქვეტიპი იერარქიებისა და სხვა სტრუქტურების ფართო გამოყენება უნივერსალური მონაცემების მოდელებში ხელს უწყობს მონაცემთა ხელმეორე გამოყენებას და მონაცემების ერთიანობას, და არა მონაცემთა ვიწრო წარმოდგენას ორგანიზაციაში.
- „ბევრი-ბევრთან“ კავშირისა და ასოციაციური არსის ფართო გამოყენება, მაშინაც კი, როდესაც მონაცემთა მოდელის მიერ შესაძლებელია „ერთი-ბევრთან“ კავშირის დამყარება, მონაცემთა მოდელს უფრო მეტი მოქნილობას აძლევს ნებისმიერ სიტუაციაში და, ბუნებრივია, ამუშავებს დროით ჭდეებს და ინახავს კავშირების მნიშვნელოვან ისტორიას, რაც შეიძლება მნიშვნელოვანი იყოს სხვადასხვა ბიზნეს წესებისათვის.
- თუ იმავე ინდუსტრიის ბევრი კომპანია იყენებს მონაცემთა უნივერსალურ მოდელს, როგორც ორგანიზაციული მონაცემთა ბაზის საფუძველი, შეიძლება გამარტივდეს ორგანიზაციული სისტემებისთვის მონაცემთა გაზიარება (მაგალითად, დაჯავშნა მანქანების გამქირავებელ ფირმებსა და ავიაკომპანიებს შორის).

განახლებული მონაცემთა მოდელირების პროცესი შეფუთული მონაცემების მოდელებით

მონაცემთა მოდელირება შეფუთული მონაცემების მოდელით მოითხოვს არანაკლებ უნარს, ვიდრე მონაცემების მოდელირება ნულიდან. შეფუთული მონაცემების მოდელები მონაცემთა მოდელის შემმუშავებლების საჭიროებას არ ამცირებს - პირიქით

სინამდვილეში პაკეტთან სამუსაოდ საჭიროა მარალი დონის უნარ-ჩვევები. შეფუთული მონაცემების მოდელები საკმაოდ რთულია, რადგან ისინი შემუშავებულია ყველა შესაძლო გარემოების დასაფარავად. მონაცემთა შემქმნელმა კარგად უნდა იცოდეს როგორც ორგანიზაცია, ასევე როგორც პროგრამული პაკეტი, რომ მოარგოს ამ ორგანიზაციის სპეციფიკურ წესებს.

მონაცემთა მოდელის შეძენისას ვყიდულობთ მეტამონაცემებს. ჩვეულებრივ, ვიღებთ მონაცემთა მოდელის სრულ აღწერას, რომელიც ჩვეულებრივ მითითებულია სტრუქტურირებული მონაცემების მოდელირების ინსტრუმენტში, როგორიცაა ERwin Computer Associated- დან ან Oracle Designer- ის Oracle Corporation- დან. მონაცემთა მოდელის მიმწოდებელმა შეადგინა EER დიაგრამა; ახდენს მონაცემთა მოდელის ყველა ელემენტის სახელდებას და განსაზღვრას; აღწერს მონაცემთა ატრიბუტის ყველა მახასიათებლის ტიპს (სიმბოლო, რიცხვითი, სურათი), სიგრძე, ფორმატი და ა.შ. შეგვიძლია დავგეჭდოთ მონაცემთა მოდელი და სხვადასხვა რეპორტები მისი შინაარსის შესახებ, პერსონალიზაციის პროცესის მხარდასაჭერად. მას შემდეგ, რაც მოვირგებთ მოდელს, შეგვიძლია გამოვიყენოთ მონაცემთა მოდელირების ინსტრუმენტი, რათა ავტომატურად შეიქმნას SQL ბრძანებები მონაცემთა ბაზის განსაზღვრის სხვადასხვა მონაცემთა მართვის სისტემისთვის.

მონაცემთა მოდელირების პროცესი შეძენილი მოდელის დანერგვის პროცესისაგან განსხვავებულია შემდეგი ქმედებებით:

- იმის გამო, რომ შეძენილი მონაცემთა მოდელი ფართოა, ვიწყებთ მონაცემთა მოდელის იმ ნაწილების იდენტიფიკაციით, რომლებიც ხეხბა ჩვენს მონაცემების მოდელირების სიტუაციას. პირველ რიგში კონცენტრირება ხდებს ამ ნაწილებზე და დეტალებზე. როგორც მონაცემთა მოდელირების უმეტეს შემთხვევაში ჯერ ვიწყებთ არსებით, შემდეგ ატრიბუტები და ბოლოს კავშირები. უნდა გავითვალისწინოთ, როგორ იმუშავებს ორგანიზაცია მომავალში და არა მხოლოდ დღეს.
- შემდეგ ხდება მონაცემების იდენტიფიცირებული ელემენტების სახელების გადარქმევა ორგანიზაციის ადგილობრივი ტერმინების შესაბამისად (თუ პაკეტში გამოყენებულია ზოგადი სახელები).
- რადგან ხშირ შემთხვევაში, შეფუთული მონაცემების მოდელი გამოყენება ახალ ინფორმაციული სისტემების შესაქმნელად, რომლებიც ანაცვლებენ არსებულ მონაცემთა ბაზებს, პაკეტიდან გამოსაყენებელი მონაცემები უნდა შეუთანადეს მონაცემთა ბაზაში არსებულ მონაცემებს.

ამ შეთანადების გამოყენების ერთ – ერთი გზაა მიგრაციის გეგმების შემუშავება არსებული მონაცემთა ბაზის ახალ სტრუქტურებად გადასაქცევად. გასათვალისწინებელია რამდენიმე ძირითადი პუნქტი ამ შეთანადების პროცესის უზრუნველსაყოფად:

- პაკეტიში იქნება მონაცემთა ისეთი ელემენტები, რომლებიც არ არის ამჟამინდელ სისტემებში, და მონაცემთა ბაზებში იქნება გარკვეული მონაცემთა ელემენტები, რომლებიც არ არის პაკეტში. ამრიგად, ზოგიერთი ელემენტი არ იქნება ასახული შეთანადებისას ახალ და ძველ გარემოში.
- ზოგადად, შეძენილი მონაცემების მოდელში ჩასმული ბიზნეს წესები მოიცავს ყველა შესაძლო გარემოებას (მაგალითად, მომხმარებელთა მაქსიმალურ რაოდენობას, რომლებიც დაკავშირებულია მომხმარებლის შეკვეთასთან). შეძენილი მონაცემების მოდელი იძლევა დიდ მოქნილობას, მაგრამ ზოგადი

დანიშნულების ბიზნეს წესი შეიძლება ძალიან სუსტი იყოს თქვენი კონკრეტული ორგანიზაციისათვის.

- იმის გამო, რომ შეძენილი მონაცემების მოდელი ყოვლისმომცველია, ვერანაირად ვერ შევძლებთ შევქმნათ და დავასრულოთ მონაცემთა სრული ბაზა ან თუნდაც მონაცემთა მთლიანი მოდელის პერსონალიზაცია ერთ პროექტში.

# მონაცემთა ბაზის ლოგიკური დიზაინი და რელაციური მოდელი

მონაცემთა ბაზის შემუშავებისას მონაცემთა ბაზის ანალიზის ფაზის ბოლოს, სისტემებისა და მონაცემთა ბაზების ანალიტიკოსებს საკმაოდ მკაფიოდ აქვთ გააზრებული მონაცემთა შენახვისა და წვდომის მოთხოვნები. ამასთან, ანალიზის დროს შემუშავებული მონაცემთა მოდელი არაა დაკავშირებული მონაცემთა ბაზის პროგრამული რეალიზაციის რომელიმე კონკრეტულ ტექნოლოგიასთან. მონაცემთა ბაზის პროგრამულ განხორციელებამდე, კონცეპტუალური მონაცემების მოდელი უნდა აისახოს მონაცემთა მოდელში, რომელიც თავსებადი იქნება ამ მიზნისთვის გამოყენებული მონაცემთა ბაზის მართვის სისტემასთან.

მონაცემთა ბაზის დაპროექტებისას განხორციელებული ქმედებები მოითხოვს მონაცემთა ბაზის ანალიზის დროს შემუშავებული მონაცემთა შენახვის მოთხოვნების გარდაქმნას სახელმძღვანელო სპეციფიკაციებად მონაცემთა ბაზის დანერგვის პროცესში. სპეციფიკაციების ორი ფორმა არსებობს:

1. ლოგიკური სპეციფიკაციები, რომლებიც ასახავს კონცეპტუალურ მოთხოვნებს მონაცემთა მოდელში, რომელიც დაკავშირებულია მონაცემთა ბაზის მართვის კონკრეტულ სისტემასთან.
2. ფიზიკური სპეციფიკაციები, სადაც მითითებულია მონაცემთა შენახვის ყველა პარამეტრი, რომლებიც შემდეგ გამოიყენება მონაცემთა ბაზის დანერგვისთვის. ამ ფაზის განმავლობაში მონაცემთა ბაზა რეალურად განისაზღვრება მონაცემთა განსაზღვრის ენის გამოყენებით.

გავიხილოთ მონაცემთა ბაზის ლოგიკური დიზაინი, კერძოდ - რელაციური მონაცემების მოდელი. მონაცემთა ბაზის ლოგიკური დიზაინი წარმოადგენს მონაცემების კონცეპტუალური მოდელის ლოგიკურ მონაცემთა მოდლად გარდაქმნის პროცესს, რომელიც შეესაბამება და თავსებადია მონაცემთა ბაზის სპეციფიკურ ტექნოლოგიასთან. ხშირად მონაცემთა ბაზის გამოცდილი შემმუშავებელი მონაცემთა ბაზის ლოგიკურ დიზაინს ქმნის მონაცემთა კონცეპტუალური მოდელირების პარალელურად, თუ საწყის ეტაპზევე ცნობილია მონაცემთა ბაზის შესაქმნელი ტექნოლოგიის ტიპი. ამასთან, მნიშვნელოვანია მოვახდინოთ მათი, როგორც ცალკეული ნაბიჯების განხილვა, რათა კონცენტრირება იყოს მონაცემთა ბაზის განვითარების თითოეულ მნიშვნელოვან ნაწილზე. კონცეპტუალური მონაცემების მოდელირება გულისხმობს ორგანიზაციის გაგებას - მოთხოვნების სწორად წარმოჩენას. მონაცემთა ბაზის ლოგიკური დიზაინი წარმოადგენს მონაცემთა ბაზის სტაბილური სტრუქტურების შექმნას - ტექნიკურ ენაზე მოთხოვნების სწორად გამოხატვას. ორივე მნიშვნელოვანი ნაბიჯია, რომელიც ფრთხილად უნდა შესრულდეს.

მიუხედავად იმისა, რომ არსებობს მონაცემების სხვა ლოგიკური მოდელები, ჩვენ განვიხილავთ მონაცემთა რელაციურ მოდელს ორი მიზეზის გამო: პირველი, მონაცემების რელაციური მოდელი ყველაზე ხშირად გამოიყენება მონაცემთა ბაზის თანამედროვე პროგრამებში; მეორე, მონაცემთა ბაზის ლოგიკური დაპროექტების ზოგიერთი პრინციპები რელაციური მოდელისთვის სხვა ლოგიკურ მოდელებზეც ვრცელდება.

ადრე განხილული მაგალითების საშუალებით ჩვენ არაფორმალურად განვიხილეთ მონაცემების რელაციური მოდელი. ამასთან, მნიშვნელოვანია აღინიშნოს, რომ რელაციური მონაცემების მოდელი ლოგიკური მონაცემების მოდელის ფორმაა და, როგორც ასეთი, იგი განსხვავდება მონაცემთა კონცეპტუალური მოდელებისგან. ამრიგად, E-R მონაცემთა მოდელი არ არის რელაციური მონაცემების მოდელი და E-R მოდელი შეიძლება არ დაემორჩილოს კარგად სტრუქტურირებული რელაციური მონაცემების მოდელის წესებს (რომელსაც ნორმალიზაცია ეწოდება), რადგან E-R მოდელი შეიქმნა სხვა მიზნებისთვის - მონაცემთა მოთხოვნების და ბიზნესის წესების გაგება-გაზიარებისათვის - და არა მონაცემთა სტრუქტურირებისათვის.

მონაცემთა ბაზის ლოგიკური დაპროექტების მიზანია კონცეპტუალური მოდელის (რომელიც წარმოადგენს ორგანიზაციის მოთხოვნებს მონაცემთა მიმართ) თარგმნას მონაცემთა ბაზის ლოგიკურ პროექტში, რომლის განხორციელება შესაძლებელია მონაცემთა ბაზის მართვის სისტემის მეშვეობით. შედეგად მიღებული მონაცემთა ბაზები უნდა აკმაყოფილებდეს მომხმარებლის საჭიროებებს მონაცემთა გაზიარების, მოქნილობისა და ხელმისაწვდომობის სიმარტივის უზრუნველყოფით.

## რელაციური მოდელი

მონაცემების რელაციური მოდელი პირველად 1970 წელს წარმოადგინა E. F. Codd- მა, ხოლო შემდეგ IBM-მა. გაშვებული იყო ორი ადრეული კვლევითი პროექტი რელაციონალური მოდელის განხორციელებადობის და პროტოტიპი სისტემების შემუშავების შესაძლებლობის დასამტკიცებლად. პირველი System R (რელაციური DBMS-ის პროტოტიპი [RDBMS]) შემუშავებით დასრულდა IBM-ის სან ხოსეს სამეცნიერო ლაბორატორიაში 1970 – იანი წლების ბოლოს. მეორე, ბერკლის კალიფორნიის უნივერსიტეტში, აკადემიურად ორიენტირებული RDBMS სისტემა Ingres-ის შემუშავებით დაგვირგვინდა. კომერციული RDBMS პროდუქციის გავრცელება დაიწყო დაახლოებით 1980 წელს. დღეს RDBMS-ები გახდა მონაცემთა ბაზის მართვის დომინანტური ტექნოლოგია და კომპიუტერებისთვის ასობით RDBMS პროდუქტი არსებობს, სმარტფონებიდან და პერსონალური კომპიუტერიდან მაგისტრალური ჩარჩოებით დასრულებული.

## ძირითადი განმარტებები

მონაცემების რელაციური მოდელი მონაცემებს წარმოადგენს ცხრილების სახით. რელაციური მოდელი ემყარება მათემატიკურ თეორიას და, შესაბამისად, მას აქვს მყარი თეორიული საფუძველი. ამასთან, ჩვენ გვჭირდება მხოლოდ რამდენიმე მარტივი ცნება რელაციონალური მოდელის აღსაწერად. ამიტომ, მისი ადვილად გააზრება და გამოყენება შეიძლება მათთვისაც, ვინც არ იცნობს ძირითად თეორიას. მონაცემების რელაციური მოდელი შედგება შემდეგი სამი კომპონენტისგან:

1. მონაცემთა სტრუქტურა (*Data structure*) - მონაცემები ორგანიზებულია ცხრილების სახით, სტრიქონებითა და სვეტებით;

2. **მონაცემთა მანიპულირება (Data manipulation)** - რელაციაში შენახული მონაცემების მანიპულირებისთვის გამოიყენება მძლავრი ოპერაციები (ჩვეულებრივ ხორციელდება SQL ენის გამოყენებით);
3. **მონაცემთა მთლიანობა (Data integrity)** - მოდელი მოიცავს მექანიზმებს ბიზნესის წესების დასაზუსტებლად, რომელებიც ინარჩუნებს მონაცემთა მთლიანობას მათი მანიპულირებისას.

EMPLOYEE1			
EmplID	Name	DeptName	Salary
100	Margaret Simpson	Marketing	48,000
140	Allen Beeton	Accounting	52,000
110	Chris Lucero	Info Systems	43,000
190	Lorenzo Davis	Finance	55,000
150	Susan Martin	Marketing	42,000

**FIGURE 4-1 EMPLOYEE1 relation with sample data**

### მონაცემთა რელაციური სტრუქტურა

**რელაცია** - მონაცემთა ორგანზომილებიანი სახელდებული ცხრილია. თითოეული რელაცია (ან ცხრილი) შედგება სახელდებული სვეტებისა და განუზღვრელი რაოდენობის უსახელო სტრიქონებისსაგან. ატრიბუტი, რომელიც შეესაბამება მის განმარტებას E-R მოდელის განმარტებით, არის რელაციის სახელდებული სვეტი. რელაციის თითოეული სტრიქონი შეესაბამება ჩანაწერს, რომელიც შეიცავს მონაცემთა (ატრიბუტის) მნიშვნელობებს ერთი არსისათვის. დიაგრამა 4-1 გვიჩვენებს რელაციის მაგალითს, სახელწოდებით EMPLOYEE1. ეს რელაცია შეიცავს შემდეგ ატრიბუტებს, რომელებიც აღწერს თანამშრომლებს: EmpID, Name, DeptName, Salary. ცხრილის ხუთი სტრიქონი შეესაბამება ხუთ თანამშრომელს. მნიშვნელოვანია გვესმოდეს, რომ ნახაზი 4-1-ში მოცემული მონაცემების მიზანია EMPLOYEE1 რელაციის სტრუქტურის ილუსტრაცია; ისინი თავად რელაციის ნაწილები არ არიან. მაშინაც კი, თუ ფიგურას მონაცემების სხვა სტრიქონებს დაუმატებთ ან არსებულ სტრიქონებში რაიმე მონაცემს შევცვლით, ეს მაინც იგივეა EMPLOYEE1 რელაციაა. არც სტრიქონის წაშლა ცვლის რელაციას. სინამდვილეში, ჩვენ შეგვიძლია წავშალოთ ნახაზი 4-1-ში ნაჩვენები ყველა სტრიქონი და EMPLOYEE1 რელაცია მაინც იარსებებს. სხვა სიტყვებით რომ ვთქვათ, სურათი 4-1 EMPLOYEE1 რელაციის მაგალითია.

ჩვენ შეგვიძლია გამოვხატოთ რელაციის სტრუქტურა მოვლე ჩანაწერების გამოყენებით, რომელშიც რელაციის სახელს მოჰყვება (ფრჩხილებში) ატრიბუტების სახელები ამ რელაციაში. EMPLOYEE1 – სთვის გვექნებოდა:

---

**EMPLOYEE1(EmpID, Name, DeptName, Salary)**

---

## რელაციის გასაღებები

ჩვენ უნდა შეგვეძლოს მონაცემთა სტრიქონის შენახვა და გამოთხოვა რელაციაში, ამ სტრიქონში შენახული მონაცემთა მნიშვნელობების საფუძველზე. ამ მიზნის მისაღწევად, ყველა რელაციას უნდა ჰქონდეს **პირველადი გასაღები**. პირველადი გასაღები არის ატრიბუტი ან ატრიბუტების კომბინაცია, რომელიც ცალსახად განსაზღვრავს თითოეულ სტრიქონს რელაციაში. პირველადი გასაღები ატრიბუტის სახელის (ების) აღსანიშნად გამოიყენება ხაზგასმა. მაგალითად, EMPLOYEE1 რელაციის პირველადი გასაღები არის EmpID (ატრიბუტი ხაზგასმულია ნახაზზე 4-1.) მოკლე აღნიშვნით, ამ რელაციას შემდეგნაირად გამოვხატავთ:

---

**EMPLOYEE1(EmpID, Name, DeptName, Salary)**

---

პირველადი გასაღების ცნება უკავშირდება ადრე განსაზღვრულ იდენტიფიკატორის ტერმინს. ატრიბუტი ან ატრიბუტების კრებული, რომელიც მითითებულია როგორც არსის იდენტიფიკატორი E-R დიაგრამაზე, შეიძლება იყოს იგივე ატრიბუტები, რომლებიც წარმოადგენს ამ არსის წარმოადგენის რელაციის პირველად გასაღებს. არსებობს გამონაკლისები: მაგალითად, ასოციაციურ არსებს არ უნდა ჰქონდეთ იდენტიფიკატორი, ხოლო სუსტი არსის (ნაწილობრივი) იდენტიფიკატორი მხოლოდ შესაბამისი რელაციის ძირითადი გასაღების მხოლოდ ნაწილს წარმოადგენს. გარდა ამისა, შეიძლება არსებობდეს არსის რამდენიმე ატრიბუტი, რომელიც შეიძლება გახდეს ასოცირებული რელაცია პირველადი გასაღები. ყველა ამ სიტუაციას განვიხილავთ მოგვიანებით.

კომპოზიტური გასაღები არის პირველადი გასაღები, რომელიც შედგება ერთზე მეტი ატრიბუტისგან. მაგალითად, DEPENDENT რელაციის პირველადი გასაღები, სავარაუდოდ, შედგებოდა EmpID და DependentName კომბინაციით.

ხშირად გვჭირდება წარმოვადგინოთ დამოკიდებულება (კავშირი) ორ ცხრილსა თუ რელაციას შორის. ეს მიიღწევა გარე გასაღებების გამოყენებით. გარე გასაღები არის ატრიბუტი (შესაძლოა კომპოზიტური) რელაციაში, რომელიც სხვა რელაციის პირველადი გასაღებია. მაგალითად, განვიხილოთ დამოკიდებულება (კავშირი) EMPLOYEE1 და DEPARTMENT:

---

**EMPLOYEE1(EmpID, Name, DeptName, Salary)  
DEPARTMENT(DeptName, Location, Fax)**

---

ატრიბუტი DeptName არის გარე გასაღები EMPLOYEE1- ში. ის მომხმარებელს საშუალებას აძლევს დააკავშიროს ნებისმიერი თანამშრომელი იმ განყოფილებასთან, რომელშიც ის დასაქმებულია. ზოგიერთი იმ ფაქტს, რომ ატრიბუტი გარე გასაღებია მიუთითებს წყვეტილი ხაზის გამოყენებით, როგორიცაა:

## EMPLOYEE1(EmplID, Name, DeptName, Salary)

### რელაციის თვისებები

ჩვენ განვსაზღვრეთ რელაციები, როგორც მონაცემთა ორგანზომილებიანი ცხრილი. ამასთან, ყველა ცხრილი არ არის რელაცია. რელაციას აქვთ რამდენიმე თვისება, რაც მათ განასხვავებს არარაელაციური ცხრილებისგან. ეს თვისებებია;

1. მონაცემთა ბაზაში თითოეულ რელაციას (ან ცხრილს) აქვს უნიკალური სახელი.
2. თითოეული სტრიქონისა და სვეტის გადაკვეთაზე მნიშვნელობა არის ატომური (ან ერთჯერადი). ცხრილის კონკრეტულ სტრიქონზე თითოეულ ატრიბუტთან ასოცირდება მხოლოდ ერთი მნიშვნელობა; დაუშვებელია მრავალმნიშვნელოვანი ატრიბუტების არსებობს;
3. თითოეული სტრიქონი უნიკალურია; რელაციაში არც ერთი სტრიქონი არ შეიძლება იყოს იდენტური;
4. ცხრილის თითოეულ ატრიბუტს (ან სვეტს) აქვს უნიკალური სახელი;
5. სვეტების თანმიმდევრობა (მარცხნიდან მარჯვნივ) უმნიშვნელოა. რელაციაში სვეტების თანმიმდევრობა შეიძლება შეიცვალოს რელაციის მნიშვნელობის ცვლილების ან გამოყენების გარეშე.
6. სტრიქონების თანმიმდევრობა (ზემოდან ქვედა) უმნიშვნელოა. როგორც სვეტების შემთხვევაში, რელაციის სტრიქონების რიგი შეიძლება შეიცვალოს ან შეინახოს ნებისმიერი თანმიმდევრობით.

### მრავალმნიშვნელოვანი ატრიბუტის გადატანა ცხრილიდან

რელაციის ჩამოთვლილი თვისებებიდან მეორე აცხადებს, რომ დაუშვებელია რელაციაში მრავალმნიშვნელოვანი ატრიბუტის არსებობა. ამრიგად, ცხრილი, რომელიც შეიცავს ერთ ან რამდენიმე მრავალმნიშვნელოვან ატრიბუტს, არ არის რელაცია. მაგალითად, დიაგრამა 4-2a აჩვენებს თანამშრომლის მონაცემებს EMPLOYEE1 რელაციიდან, რომელიც მოიცავს იმ კურსებს, რომლებიც ამ თანამშრომლებმა გაიარეს. იმის გამო, რომ მოცემულმა თანამშრომელმა შეიძლება ერთზე მეტი კურსი გაიარა, CourseTitle და DateCompleted მრავალმნიშვნელოვანი ატრიბუტებია. მაგალითად, EmpID 100-ის შესაბამისმა თანამშრომელმა გაიარა ორი კურსი, შესაბამისად, არსებობს ორი კურსი: CourseTitle (SPSS და Surveys) და DateCompleted (6/9/2015 და 10/7/2015), რომელიც ასოცირდება EmpID-ის ერთ მნიშვნელობასთან (100). თუ თანამშრომელს არ აქვს გავლილი რაიმე კურსები, CourseTitle და DateCompleted ატრიბუტის მნიშვნელობები ნულოვანია (მაგალითისთვის იხილეთ EmpID 190 თანამშრომელი).

ვაჩვენოთ, თუ როგორ უნდა აღმოიფხვრას მრავალმნიშვნელოვანი ატრიბუტები ნახაზზე 4-2 b, მოცემული მონაცემების შესაბამისი მნიშვნელობების შევსებით ნახაზი 4-2a – ში მოცემული მონაცემების შესაბამისი მნიშვნელობებით. შედეგად, 4-2 b სურათზე მოცემულ ცხრილს აქვს მხოლოდ ერთმნიშვნელოვანი ატრიბუტები და ახლა აკმაყოფილებს რელაციის

ატომურობის თვისებას. ამ რელაციას მივანიჭოთ სახელი EMPLOYEE2, რათა განვასხვაოთ იგი EMPLOYEE1- ისგან. ამასთან, როგორც ვხედავთ, ამ ახალ რელაციას აქვს არასასურველი თვისებები. ზოგიერთ მათგანს მოგვიანებით განვიხილავთ, მაგრამ ერთ-ერთი მათგანია ის, რომ პირველადი გასაღების სვეტი EmpID აღარ წარმოადგენს ცალსახად თითოეულ სტრიქონს.

**FIGURE 4-2** Eliminating multivalued attributes

(a) Table with repeating groups					
EmplID	Name	DeptName	Salary	CourseTitle	DateCompleted
100	Margaret Simpson	Marketing	48,000	SPSS Surveys	6/19/2015 10/7/2015
140	Alan Beeton	Accounting	52,000	Tax Acc	12/8/2015
110	Chris Lucero	Info Systems	43,000	Visual Basic C++	1/12/2015 4/22/2015
190	Lorenzo Davis	Finance	55,000		
150	Susan Martin	Marketing	42,000	SPSS Java	6/16/2015 8/12/2015

(b) EMPLOYEE2 relation					
EMPLOYEE2					
EmplID	Name	DeptName	Salary	CourseTitle	DateCompleted
100	Margaret Simpson	Marketing	48,000	SPSS	6/19/2015
100	Margaret Simpson	Marketing	48,000	Surveys	10/7/2015
140	Alan Beeton	Accounting	52,000	Tax Acc	12/8/2015
110	Chris Lucero	Info Systems	43,000	Visual Basic	1/12/2015
110	Chris Lucero	Info Systems	43,000	C++	4/22/2015
190	Lorenzo Davis	Finance	55,000		
150	Susan Martin	Marketing	42,000	SPSS	6/19/2015
150	Susan Martin	Marketing	42,000	Java	8/12/2015

## მონაცემთა ბაზის ნიმუში

რელაციური მონაცემთა ბაზა შეიძლება შედგებოდეს ნებისმიერი რაოდენობის რელაციებისაგან. მონაცემთა ბაზის სტრუქტურა აღიწერება სქემის გამოყენებით, რომელიც წარმოადგენს მონაცემთა ბაზის საერთო ლოგიკური სტრუქტურის აღწერას. სქემის გამოხატვის ორი საერთო მეთოდი არსებობს:

1. მოკლე ტექსტური განაცხადები, რომელშიც თითოეული რელაციაა დასახელებული და მისი ატრიბუტების სახელები ფრჩხილებში მოჰყვება;
2. გრაფიკული წარმოდგენა, რომელშიც თითოეული რელაცია წარმოდგენილია მართვულებით, რომელიც შეიცავს რელაციის ატრიბუტებს.

ტექსტურ განაცხადებს უპირატესობას ანიჭებენ სიმარტივის გამო. ამასთან, გრაფიკული წარმოდგენა გვთავაზობს დამოწმებითი მთლიანობის შეზღუდვის უკეთ გამოხატვას.

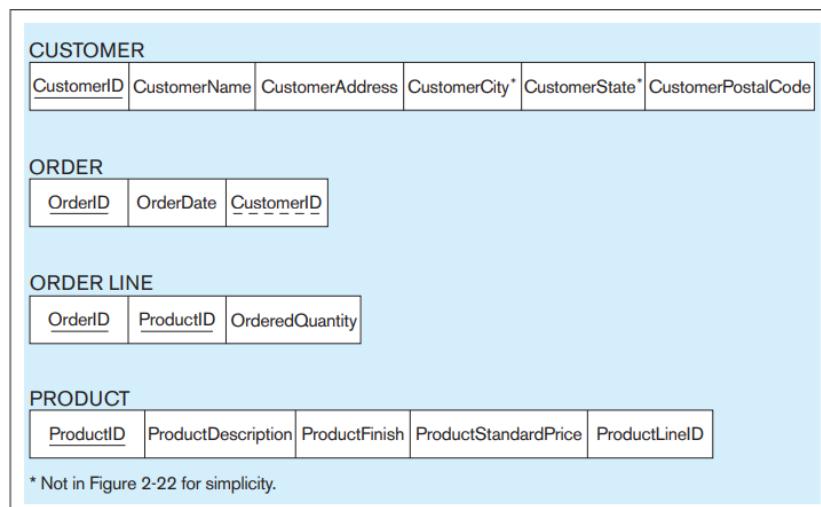
განვიხილოთ Pine Valley ავეჯის კომპანიის მაგალითი. ნახაზზე 4-3 ნაჩვენებია ოთხი რელაციისაგან შემდგარი სქემა. ეს რელაციებია CLIENT, ORDER, ORDER LINE და PRODUCT. ამ რელაციების გასაღები ატრიბუტები ხაზგასმულია და თითოეულ რელაციაში შედის სხვა მნიშვნელოვანი ატრიბუტები. ქვემოთ მოცემულია ამ რელაციების ტექსტური აღწერა, რათა შევძლოთ მათი შედარება:

---

**CUSTOMER**(CustomerID, CustomerName, CustomerAddress,  
CustomerCity, CustomerState, CustomerPostalCode)  
**ORDER**(OrderID, OrderDate, CustomerID)  
**ORDER LINE**(OrderID, ProductID, OrderedQuantity)  
**PRODUCT**(ProductID, ProductDescription, ProductFinish,  
ProductStandardPrice, ProductLineID)

---

**FIGURE 4-3** Schema for four relations (Pine Valley Furniture Company)



გაითვალისწინეთ, რომ ORDER LINE- ის პირველადი გასაღები არის კომპოზიტური გასაღები, რომელიც შედგება OrderID და ProductID ატრიბუტებისგან. ასევე, CustomerID არის გარე გასაღები ORDER რელაციაში. ეს საშუალებას აძლევს მომხმარებელს შეუკვეთოს შეკვეთა მომხმარებელთან, რომელმაც შეკვეთა წარადგინა. ORDER LINE- ს ორი გარე გასაღები აქვს: OrderID და ProductID. ეს გასაღებები საშუალებას აძლევს მომხმარებელს დაუკავშიროს შეკვეთის თითოეული სტრიქონი შესაბამის შეკვეთასთან და პროდუქტთან. იმ შემთხვევებში, როდესაც გარე გასაღები ასევე არის კომპოზიციური გასაღების ნაწილი (მაგალითად, ეს), ხშირად ატრიბუტის მხოლოდ ძირითად საკვანძო როლს აღნიშნავენ უწყვეტი ხაზგასმით.

ამ მონაცემთა ბაზის ეგზემპლარი ნაჩვენებია დიაგრამა 4-4-ზე. ამ ნახატზე მოცემულია ოთხი ცხრილი, მონაცემთა ნიმუშით. დავაკვირდეთ, როგორ გვაძლევს გარე გასაღები სხვადასხვა ცხრილების კავშირს. რელაციური სქემის მაგალითის შექმნა მონაცემთა ნიმუშთან ერთად სასარგებლობა ოთხი მიზეზის გამო:

1. მონაცემთა ნიმუში საშუალებას გვაძლევს შევამოწმოთ ჩვენი დაშვებები დიზაინთან დაკავშირებით;
2. მონაცემთა ნიმუში გვთავაზობს ჩვენი დიზაინის სიზუსტის შემოწმების მოსახერხებელ გზას;
3. მონაცემთა ნიმუში ხელს უწყობს მომხმარებლებთან კომუნიკაციის გაუმჯობესებას ჩვენი დიზაინის განხილვისას;
4. ნიმუშის მონაცემები შეიძლება გამოყენებულ იქნას პროტოტიპის პროგრამების შესაქმნელად და მოთხოვნების შესამოწმებლად.

## მთლიანობის შეზღუდვები

მონაცემთა რელაციური მოდელი მოიცავს რამდენიმე სახის შეზღუდვას, ან მისაღები მნიშვნელობებისა და მოქმედებების შეზღუდვების წესებს, რომელთა მიზანია მონაცემთა ბაზაში მონაცემთა სიზუსტისა და მთლიანობის შენარჩუნება. მთლიანობის შეზღუდვის ძირითადი ტიპები არის დომენის შეზღუდვები, არსის მთლიანობა და დამოწმებითი მთლიანობა.

**TABLE 4-1 Domain Definitions for INVOICE Attributes**

Attribute	Domain Name	Description	Domain
CustomerID	Customer IDs	Set of all possible customer IDs	character: size 5
CustomerName	Customer Names	Set of all possible customer names	character: size 25
CustomerAddress	Customer Addresses	Set of all possible customer addresses	character: size 30
CustomerCity	Cities	Set of all possible cities	character: size 20
CustomerState	States	Set of all possible states	character: size 2
CustomerPostalCode	Postal Codes	Set of all possible postal zip codes	character: size 10
OrderID	Order IDs	Set of all possible order IDs	character: size 5
OrderDate	Order Dates	Set of all possible order dates	date: format mm/dd/yy
ProductID	Product IDs	Set of all possible product IDs	character: size 5
ProductDescription	Product Descriptions	Set of all possible product descriptions	character: size 25
ProductFinish	Product Finishes	Set of all possible product finishes	character: size 15
ProductStandardPrice	Unit Prices	Set of all possible unit prices	monetary: 6 digits
ProductLineID	Product Line IDs	Set of all possible product line IDs	integer: 3 digits
OrderedQuantity	Quantities	Set of all possible ordered quantities	integer: 3 digits

## დომენის შეზღუდვები

ყველა მნიშვნელობა, რომელიც გამოჩენდება რელაციის სვეტში, უნდა იყოს ერთი და იგივე დომენიდან. დომენი არის მნიშვნელობების ერთობლიობა, რომელიც შეიძლება მიენიჭოს ატრიბუტს. დომენის განმარტება ჩვეულებრივ შედგება შედეგი კომპონენტებისგან: დომენის სახელი, მნიშვნელობა, მონაცემთა ტიპი, ზომა (ან სიგრძე) და დასაშვები მნიშვნელობები ან დასაშვები დიაპაზონი (ასეთის არსებობის შემთხვევაში). ცხრილი 4-1 გვიჩვენებს დომენის განმარტებებს იმ დომენებისთვის, რომლებიც ასოცირდება ატრიბუტებთან 4-3 და 4-4 ნახაზებში.

## არსის მთლიანობა

არსის მთლიანობის წესი შექმნილია იმის უზრუნველსაყოფად, რომ ყველა რელაციას ჰქონდეს პირველადი გასაღები და რომ ამ პირველადი გასაღების ყველა მონაცემების მნიშვნელობები იყოს ქმედითი. კერძოდ, იძლევა გარანტიას, რომ პირველადი გასაღების ყველა ატრიბუტი არ არის ნულოვანი.

ზოგიერთ შემთხვევაში, კონკრეტულ ატრიბუტს არ შეიძლება მიენიჭოს მონაცემთა მნიშვნელობა. არსებობს ორი სიტუაცია, რომლის დროსაც ეს შეიძლება მოხდეს: ან არ არსებობს მონაცემთა შესაბამისი მნიშვნელობა, ან მნიშვნელობების მინიჭებისას მნიშვნელობები არ არის ცნობილი. დავუშვათ, მაგალითად, რომ ვავსებთ დასაქმების ფორმას, რომელსაც აქვს ფაქსის ნომერი. თუ ფაქსის ნომერი არ გაქვთ, ამ ადგილს ცარიელს დავტოვებთ, რადგან ის არ გვეხება. ან ვთქვათ მოგვეთხოვება შეავსოთ წინა დამსაქმებლის ტელეფონის ნომერი. თუ ნომერი არ გვახსოვს, შეიძლება ცარიელი დავტოვოთ, რადგან ეს ინფორმაცია არ არის ცნობილი.

რელაციური მონაცემების მოდელი საშუალებას გვაძლევს ატრიბუტს მივუთითოთ ნულოვანი მნიშვნელობა უბრალოდ აღწერილ სიტუაციებში. Null არის მნიშვნელობა, რომელიც შეიძლება მიენიჭოს ატრიბუტს, როდესაც სხვა მნიშვნელობა არ გამოიყენება ან მოქმედი მნიშვნელობა უცნობია. სინამდვილეში, null არ არის მნიშვნელობა, არამედ ის მიუთითებს მნიშვნელობის არარსებობაზე. მაგალითად, ეს არ არის იგივე, რაც რიცხვითი ნულოვანი ან პრობელების სტრიქონი. ნულოვნების ჩართვა რელაციურ მოდელში გარკვეულწილად სადაცოა, რადგან მას ზოგჯერ ანომალურ შედეგებამდე მიყვავართ. ამასთან, რელაციური მოდელის გამომგონებელი კოდი მხარს უჭერს null-ების გამოყენებას გამოტოვებული მნიშვნელობებისთვის.

ყველა თანახმაა, რომ პირველადი გასაღების მნიშვნელობების გადახრა დაუშვებელია. ამრიგად, არსის მთლიანობის წესი აცხადებს შემდეგს: პირველადი გასაღების არცერთი ატრიბუტი (ან პირველადი გასაღების ატრიბუტის კომპონენტი) არ შეიძლება იყოს ნულოვანი.

## დამოწმებითი მთლიანობა

რელაციური მონაცემების მოდელში ცხრილებს შორის კავშირი განისაზღვრება გარე გასაღებების გამოყენებით. მაგალითად, დიაგრამა 4-4-ში, CUSTOMER და ORDER ცხრილებს შორის კავშირი განისაზღვრება CustomerID ატრიბუტის, როგორც გარე გასაღების ORDER-ში ჩასმით. ეს გულისხმობს იმას, რომ სანამ ORDER ცხრილში არ ჩავსამთ ახალ სტრიქონს, ამ შევეთის მომხმარებელი უკვე უნდა არსებობდეს CUSTOMER ცხრილში. თუ შვისწავლით ORDER ცხრილის სტრიქონებს 4-4 ნახაზზე, ვნახავთ, რომ მომხმარებლის ყველა ნომერი შევეთისთვის უკვე ჩანს CUSTOMER ცხრილში.

დამოწმებითი მთლიანობის შეზღუდვა არის წესი, რომელიც ინარჩუნებს შეთანადება ორი რელაციის სტრიქონებში. წესში ნათქვამია, რომ თუ ერთ რელაციაში არის გარე გასაღები, ან თითოეული კარე გასაღების მნიშვნელობა უნდა ემთხვეოდეს პირველადი გასაღების

მნიშვნელობას სხვა რელაციაში, ან გარე გასაღების მნიშვნელობა უნდა იყოს ნულოვანი (მაგ. ნაბ. 4-4-ზე მოცემული ცხრილები).

რელაციური სქემის გრაფიკული ვერსია გთავაზობთ მარტივ ტექნიკას კავშირების იდენტიფიკაციისთვის, სადაც დამოწმებითი მთლიანობა უნდა განხორციელდეს. დიაგრამა 4-5 გვიჩვენებს სქემა 4-3-ში მოცემული რელაციების. ისარი დახატულია ყოველი გარე გასაღებიდან შესაბამის პირველად გასაღებისკენ. სქემაში მოცემული თითოეული ისრისთვის უნდა განისაზღვროს დამოწმებითი მთლიანობის შეზღუდვა. გვახსოვდეს, რომ OrderID და ProductID ORDER LINE-ში არის როგორც გარე გასაღებები, ასევე კომპოზიტური ძირითადი გასაღების კომპონენტები.

**FIGURE 4-4** Instance of a relational schema (Pine Valley Furniture Company)

The screenshot shows four data grids representing tables in a relational database:

- Customer\_T**: Contains 15 records. Data (approximate):
 

CustomerID	CustomerName	CustomerAddress	CustomerCity	CustomerState	CustomerPostalCode
1	Contemporary Casuals	1355 S Hines Blvd	Gainesville	FL	32601-2871
2	Value Furniture	15145 S.W. 17th St.	Plano	TX	75094-7743
3	Home Furnishings	1900 Allard Ave.	Albany	NY	12209-1125
4	Eastern Furniture	1925 Beltline Rd.	Carteret	NJ	07008-3188
5	Impressions	5585 Westcott Ct.	Sacramento	CA	94206-4056
6	Furniture Gallery	325 Flatiron Dr.	Boulder	CO	80514-4432
7	Period Furniture	394 Rainbow Dr.	Seattle	WA	97954-5589
8	California Classics	816 Peach Rd.	Santa Clara	CA	96915-7754
9	M and H Casual Furniture				4620-2314
10	Seminole Interiors				4646-4423
11	American Euro Lifestyles				7508-5621
12	Battle Creek Furniture				9015-3401
13	Heritage Furnishings				7013-8834
14	Kaneohe Homes				6744-2537
15	Mountain Scenes				4403-4432
- OrderLine\_T**: Contains 18 records. Data (approximate):
 

OrderID	ProductID	OrderedQuantity
1001	1	2
1001	2	2
1001	4	1
1002	3	5
1003	3	3
1004	6	2
1004	8	2
1005	4	4
1006	4	1
1006	5	2
1006	7	2
1007	1	3
1007	2	2
1008	3	3
1008	8	3
1009	4	2
1009	7	3
1010	8	10
- Order\_T**: Contains 10 records. Data (approximate):
 

OrderID	OrderDate	CustomerID
1001	10/21/2015	1
1002	10/21/2015	8
1003	10/22/2015	15
1004	10/22/2015	5
1005	10/24/2015	3
1006	10/24/2015	2
1007	10/27/2015	11
1008	10/30/2015	12
1009	11/5/2015	4
1010	11/5/2015	1
- Product\_T**: Contains 8 records. Data (approximate):
 

ProductID	ProductDescription	ProductFinish	ProductStandardPrice	ProductLineID
1	End Table	Cherry	\$175.00	1
2	Coffee Table	Natural Ash	\$200.00	2
3	Computer Desk	Natural Ash	\$375.00	2
4	Entertainment Center	Natural Maple	\$650.00	3
5	Writers Desk	Cherry	\$325.00	1
6	8-Drawer Desk	White Ash	\$750.00	2
7	Dining Table	Natural Ash	\$800.00	2
8	Computer Desk	Walnut	\$250.00	3

დამოწმებითი მთლიანობის უზრუნველყოფისასა უნდა შეგვეძლოს იმის გარკვევა ნებადართულია თუ არა გარე გასაღების მნიშვნელობა იყოს NULL . თუ თითოეულ შეკვეთას უნდა ჰყავდეს მომხმარებელი (სავალდებულო ურთიერთობა), მაშინ გარე გასაღები CustomerID არ შეიძლება იყოს ნულოვანი ORDER რელაციაში. თუ კავშირი არასავალდებულოა, მაშინ გარე გასაღები შეიძლება იყოს ნულოვანი. შესაძლებელია თუ არა

გარე გასაღების null მითითება, უნდა მოხდეს მონაცემთა ბაზის განსაზღვრისას, გარე გასაღების ატრიბუტის თვისების მითითებისას.

სინამდვილეში, შეიძლება თუ არა გარე გასაღების ნულოვანი მნიშვნელობა, უფრო რთულია E-R დიაგრამაზე მოდელირებისას და განსაზღვრისას, ვიდრე აქამდე ვაჩვენეთ. მაგალითად, რა ემართება შეკვეთის მონაცემებს, თუ ავირჩევთ მომხმარებელს, რომელმაც შეკვეთა უკვე წარადგინა? შეიძლება გვინდოდეს გაყიდვების ნახვა, მაშინაც კი, თუ მომხმარებელზე აღარ ვიზრუნებთ. შესაძლებელია სამი არჩევანი:

1. წავშალოთ მასთან დაკავშირებული შეკვეთები (ე.წ. კასკადური წაშლა), ამ შემთხვევაში ჩვენ ვკარგავთ არა მხოლოდ მომხმარებელს, არამედ გაყიდვების მთლიან ისტორიას;
2. ავკრძალოთ მომხმარებლის წაშლა მანამ, სანამ ყველა დაკავშირებული შეკვეთა არ წაიშლება (უსაფრთხოების შემოწმება).
3. გარე გასაღებში მოვათავსოთ null მნიშვნელობა (გამონაკლისი, რათა შეკვეთის შექმნისას შეკვეთას ჰქონდეს CustomerID მნიშვნელობა, დაკავშირებული მომხმარებლის წაშლის შემთხვევაში, CustomerID შეიძლება გახდეს null).

## რელაციური ცხრილების შექმნა

განვმარტოთ რელაციური ცხრილის შექმნა SQL მონაცემთა განსაზღვრის ენის **CREATE TABLE** განაცხადის გამოყენებით. პრაქტიკაში, ცხრილის ეს განმარტებები იქმნება მონაცემთა ბაზის შემუშავების პროცესში მოგვიანებით. თუმცა ჩვენ ამ პროცედურებს აქ ვაჩვენებთ პროცესის უწყვეტობისთვის და განსაკუთრებით იმის გასაცნობად, თუ როგორ ხდება SQL-ში მთლიანობის შეზღუდვების რეალიზაცია.

SQL ცხრილის განმარტებები ნაჩვენებია 4-6 ნახაზზე. რელაციურ სქემაში ნაჩვენები ოთხი რელაციიდან თითოეული ცხრილი იქმნება ცალ-ცალკე (სურათი 4-5). შემდეგ განისაზღვრება ცხრილის თითოეული ატრიბუტი. გავითვალისწინოთ, რომ მონაცემთა ტიპი და სიგრძე თითოეული ატრიბუტისთვის აღებულია დომენის განმარტებებიდან (ცხრილი 4-1). მაგ: ატრიბუტი CustomerName Customer\_T ცხრილში განისაზღვრება, როგორც VARCHAR (ცვლადი სიმბოლო) მონაცემთა ტიპი, სიგრძით 25. NOT NULL მითითებით, თითოეული ატრიბუტი შეიძლება შეიზღუდოს ნულოვანი მნიშვნელობის მინიჭებით.

პირველადი გასაღები მითითებულია თითოეული ცხრილისთვის PRIMARY KEY პირობის გამოყენებით, თითოეული ცხრილის განმარტების ბოლოს. OrderLine\_T ცხრილი ასახავს, თუ როგორ უნდა მიუთითოთ პირველადი გასაღები, როდესაც ეს გასაღები არის ატრიბუტი. ამ მაგალითში OrderLine\_T- ის პირველადი გასაღები არის OrderID და ProductID კომბინაცია. ოთხი ცხრილის თითოეული პირველადი გასაღები ატრიბუტი იზღუდება NOT NULL-ით. ეს ახორციელებს ადრე აღწერილ არსის მთლიანობის შეზღუდვას. გავითვალისწინოთ, რომ NOT NULL შეზღუდვა ასევე შეიძლება გამოყენებულ იქნეს პირველადი გასაღებისაგან განსხვავებული ატრიბუტებსთვისაც.

დამოწმებითი მთლიანობის შეზღუდვები მარტივად განისაზღვრება მე-4-5 ნახაზზე ნაჩვენები გრაფიკული სქემის საფუძველზე. ისარი გამოდის თითოეული გარე გასაღებიდან და

მიუთითებს კავშირში არსებული რელაციის პირველად გასაღებზე. SQL ცხრილის განსაზღვრებაში, FOREIGN KEY REFERENCES განაცხადი შესაბამება თითოეულ ამ ისარს. ამრიგად, ცხრილისთვის Order\_T, გარე გასაღები CustomerID მიუთითებს Customer\_T- ის პირველად გასაღებაზე, რომელსაც ასევე უწოდებენ CustomerID. მიუხედავად იმისა, რომ ამ შემთხვევაში გარე გასაღები და პირველადი გასაღები ერთი და იგივე სახელისაა, ეს არა სავალდებულო. მაგალითად, გარე გასაღების ატრიბუტს CustomerID- ის ნაცვლად შეიძლება დაერქვას CustNo. ამასთან, გარე და პირველადი გასაღებები უნდა იყოს ერთი და იგივე დომენისგან (ანუ მათ უნდა ჰქონდეთ მონაცემთა ერთი და იგივე ტიპი).

OrderLine\_T ცხრილი გვთავაზობს ცხრილის მაგალითს, რომელსაც ორი გარე გასაღები აქვს. ამ ცხრილის გარე გასაღებები მითითებულია Order\_T და Product\_T ცხრილებში, შესაბამისად. გვითვალისწინოთ, რომ ეს ორი გარე გასაღები ასევე OrderLine\_T პირველადი გასაღების კომპონენტებია. ამ ტიპის სტრუქტურა ძალზე გავრცელებულია, „ბევრი-ბევრთან“ კავშირის განხორციელებისას.

**FIGURE 4-6** SQL table definitions

```

CREATE TABLE Customer_T
  (CustomerID          NUMBER(11,0)    NOT NULL,
   CustomerName        VARCHAR2(25)   NOT NULL,
   CustomerAddress     VARCHAR2(30),
   CustomerCity        VARCHAR2(20),
   CustomerState       CHAR(2),
   CustomerPostalCode  VARCHAR2(9),
  CONSTRAINT Customer_PK PRIMARY KEY (CustomerID);

CREATE TABLE Order_T
  (OrderId              NUMBER(11,0)    NOT NULL,
   OrderDate            DATE DEFAULT SYSDATE,
   CustomerID          NUMBER(11,0),
  CONSTRAINT Order_PK PRIMARY KEY (OrderId),
  CONSTRAINT Order_FK FOREIGN KEY (CustomerID) REFERENCES Customer_T (CustomerID);

CREATE TABLE Product_T
  (ProductID            NUMBER(11,0)    NOT NULL,
   ProductDescription   VARCHAR2(50),
   ProductFinish        VARCHAR2(20),
   ProductStandardPrice DECIMAL(6,2),
   ProductLineID        NUMBER(11,0),
  CONSTRAINT Product_PK PRIMARY KEY (ProductID);

CREATE TABLE OrderLine_T
  (OrderId              NUMBER(11,0)    NOT NULL,
   ProductID           NUMBER(11,0)    NOT NULL,
   OrderedQuantity      NUMBER(11,0),
  CONSTRAINT OrderLine_PK PRIMARY KEY (OrderId, ProductID),
  CONSTRAINT OrderLine_FK1 FOREIGN KEY (OrderId) REFERENCES Order_T (OrderId),
  CONSTRAINT OrderLine_FK2 FOREIGN KEY (ProductID) REFERENCES Product_T (ProductID));

```

## კარგად სტრუქტურირებული რელაციები

ნორმალიზაციის შესახებ დისკუსიისთვის მოსამზადებლად საჭიროა პასუხი გაეცეს შემდეგ კითხვას: რას წარმოადგენს კარგად სტრუქტურირებული რელაცია? ინტუიციურად, კარგად სტრუქტურირებული რელაცია შეიცავს მინიმალურ სიჭარბეს და მომხმარებლებს საშუალებას აძლევს შეცდომისა და შეუსაბამობის გარეშე ჩასვან, შეცვალონ და წაშალონ ცხრილის სტრიქონები. EMPLOYEE1 (სურათი 4-1) ასეთი რელაციაა. ცხრილის თითოეული სტრიქონი შეიცავს მონაცემებს, რომლებიც აღწერს ერთ თანამშრომელს, ხოლო თანამშრომლის მონაცემებში ნებისმიერი ცვლილება (მაგალითად, ხელფასის ცვლილება) შემოიფარგლება ცხრილის ერთ რიგში განსახორციელები ცვლილებით. ამის საპირისპიროდ, EMPLOYEE2 (სურათი 4-2b) არ არის კარგად სტრუქტურირებული რელაცია. თუ შევისწავლით ცხრილში მოცემულ მონაცემთა ნიმუშს, შევამჩნევთ მნიშვნელოვან სიჭარბეს. მაგალითად, EmpID- ის, Name- ს, DeptName- ს და Salary- ს მნიშვნელობები გამოჩნდება ორ სხვადასხვა სტრიქონში 100, 110 და 150 თანამშრომლებისთვის. შესაბამისად, თუ EmpID- 100-ის შესაბამისი თანამშრომლის ხელფასი შეიცვალა, ეს ფაქტი ორ სტრიქონში უნდა ჩავწეროთ.

ცხრილში სიჭარბემ შეიძლება გამოიწვიოს შეცდომები ან შეუსაბამობები (ე.წ. ანომალიები), როდესაც მომხმარებელი ცდილობს განაახლოს ცხრილში მოცემული მონაცემები. ჩვენ, როგორც წესი, გვაწუხებს სამი სახის ანომალია:

1. ჩასმის ანომალია - დავუშვათ, რომ EMPLOYEE2- ს უნდა დავამატოთ ახალი თანამშრომელი. ამ რელაციის პირველადი გასაღები არის EmpID და CourseTitle კომბინაცია (როგორც ადრე აღვნიშნეთ). ამიტომ, ახალი სტრიქონის ჩასასმელად, მომხმარებელმა უნდა მიუთითოს როგორც EmpID-ის, ასევე CourseTitle-ის მნიშვნელობები, რადგან პირველადი გასარების მნიშვნელობები არ შეიძლება იყოს ნულოვანი ან არარსებული. ეს ანომალიაა, რადგან მომხმარებელს უნდა შეეძლოს შეიტანოს თანამშრომლის მონაცემები, კურსებზე მონაცემების მიწოდების გარეშე.
2. წაშლის ანომალია - დავუშვათ, რომ ცხრილიდან წაიშლება EmpID- 140-ის შესაბამისი პერსონალის მონაცემები. ეს გამოიწვევს ინფორმაციის დაკარგვას იმის შესახებ, რომ ამ თანამშრომელმა დაასრულა კურსი (Tax Acc) 12/8/2015. ფაქტობრივად, ეს კარგავს ინფორმაციას, რომ ამ კურსს ჰქონდა შეთავაზება, რომელიც დასრულდა ამ დღეს.
3. მოდიფიკაციის ანომალია - დავუშვათ, რომ EmpID- 100-ის შესაბამის თანამშრომელს გაეზარდა ხელფასი. ჩვენ უნდა დავაფიქსიროთ ეს ზრდა თითოეული ამ თანამშრომლის შესაბამის ყოველ სტრიქონში (ორი შემთხვევა ნახაზზე 4-2); წინააღმდეგ შემთხვევაში, მონაცემები არათანმიმდევრული იქნება.

ეს ანომალიები მიუთითებს, რომ EMPLOYEE2 არ არის კარგად სტრუქტურირებული რელაცია. ამ რელაციის პრობლემა ისაა, რომ იგი შეიცავს მონაცემებს ორი არსის შესახებ: EMPLOYEE და COURSE. ჩვენ გამოვიყენებთ ნორმალიზაციის თეორიას EMPLOYEE2-ის ორ რელაციად დაყოფისთვის. შედეგად მიღებული ერთ-ერთი რელაციაა EMPLOYEE1 (სურათი 4-1). მეორეს EMP COURSE- ს დავარქმევთ, რომელიც მოცემულია მონაცემების ნიმუშით 4-7 ნახაზზე. ამ რელაციის პირველადი გასაღები არის EmpID-ისა და CourseTitle-ის კომბინაცია და ამ ატრიბუტების სახელებს ხაზს ვუსვამთ დიაგრამა 4-7-ზე, ამ ფაქტის გასაზრდელად. დიაგრამა

4-7-ზე გამოსახული EMP COURSE არ შეიცავს ადრე აღწერილი ანომალიების ტიპებს და, შესაბამისად, კარგად არის სტრუქტურირებული.

EmplID	CourseTitle	DateCompleted
100	SPSS	6/19/2015
100	Surveys	10/7/2015
140	Tax Acc	12/8/2015
110	Visual Basic	1/12/2015
110	C++	4/22/2015
150	SPSS	6/19/2015
150	Java	8/12/2015

**FIGURE 4-7 EMP COURSE**

#### EER დიაგრამის გარდაქმნა რელაციურ სტრუქტურად

ლოგიკური დიზაინის დროს ჩვენ გარვდაქმნით E-R (და EER) დიაგრამებს, რომლებიც შემუშავდა კონცეპტუალური დიზაინის დროს, რელაციური მონაცემთა ბაზის სქემებად. ამ პროცესის საწყის წყაროს წარმოადგენს „არსი-კავშირი“ (და გაუმჯობესებული E-R) დიაგრამები, ხოლო შედეგები წარმოადგენს რელაციურ სქემებს.

EER დიაგრამების რელაციად გარდაქმნა (ან ასახვა) შედარებით მარტივი პროცესია, კარგად განსაზღვრული წესებით. სინამდვილეში, მრავალ CASE ინსტრუმენტს შეუძლია ავტომატურად შეასრულოს გარდაქმნის მრავალი ბიჯი. ამასთან, მნიშვნელოვანია ამ პროცესში ბიჯების არსის გაზრება ოთხი მიზეზის გამო:

1. CASE ინსტრუმენტებს ხშირად არ შეუძლიათ მონაცემთა უფრო რთული კავშირების (ურთდამოკირებულებების) მოდელირება, როგორიცაა სამეული კავშირები და სუპერტიპი/ქვეტიპი კავშირები, ამ სიტუაციებში შეიძლება დაგვჭირდს ბიჯების ხელით შესრულება;
2. ზოგჯერ არსებობს ლეგიტიმური ალტერნატივა, რომლისთვისაც უნდა ავირჩიოთ კონკრეტული გადაწყვეტა;
3. მზად უნდა ვიყოთ CASE ინსტრუმენტის საშუალებით მიღებული შედეგების ხარისხის შესამოწმებლად;
4. ტრანსფორმაციის პროცესის გაგება დაგვეხმარებათ იმის გაგებაში, თუ რატომ განსხვავდება კონცეპტუალური მონაცემების მოდელირება (რეალურ დომენზე მოდელირება) ლოგიკური მონაცემების მოდელირებისგან (ანუ წარმოადგენს მონაცემთა ერთეულებს დომენში ისე, რომ ეს შეიძლება განხორციელდეს DBMS-ით).

ჩვენ განვიხილეთ სამი სახის არსი:

1. რეგულარული არსი (**Regular entities**) - არსები, რომლებიც დამოუკიდებელად არსებობენ და ზოგადად წარმოადგენენ რეალურ ობიექტებს, როგორიცაა პირები და პროდუქტები. არსების რეგულარული ტიპები წარმოდგენილია ერთმაგი ხაზით შემოფარგლული მართვულთხედებით.
2. სუსტი არსები (**Weak entities**) - არსები, რომელთა არსებობა შეუძლებელია, მფლობელი (რეგულარული) არსის ტიპთან საიდენტიფიკაციო ურთიერთობის გარეშე. სუსტი არსის აღინიშნება ორმაგი ხაზით შემოფარგლული მართვულთხედით.
3. ასოციაციური არსები **Associative entities** (ასევე უწოდებენ გერუნდებს) - წარმოიქმნება სხვა არსების ტიპებს შორის „მრავალი-მრავალთან“ კავშირისას. ასოციაციური არსები წარმოდგენილია მომრგვალებული კუთხეებიანი მართვულთხედით.

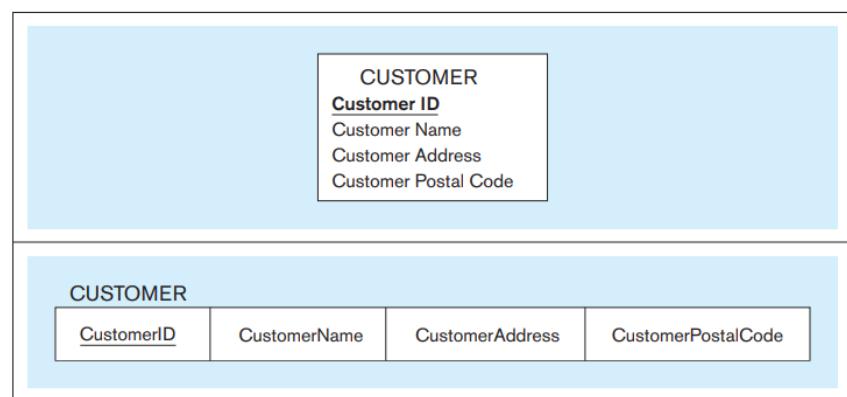
#### ბიჯი 1: რეგულარული არსების ასახვა

თითოეული რეგულარული არსის ტიპი E-R სქემაში გარდაიქმნება რელაციად (ორგანზომილებიან ცხრილად, რომელსაც ხშირად „დამოკიდებულებასაც“ უწოდებენ). რელაციის სახელი, ზოგადად, იგივეა, რაც არსის ტიპის სახელი. არსის ტიპის თითოეული მარტივი ატრიბუტი ხდება რელაციის ატრიბუტი. არსის ტიპის იდენტიფიკატორი ხდება შესაბამისი რელაციის პირველადი გასაღები. აუცილებელია შეამოწიდეს ასეთი სახით წარმოქმნილი პირველადი გასაღები, რათა ის აკმაყოფილებდეს იდენტიფიკატორების თვისებებს.

დიაგრამა 4-8 ა გვიჩვენებს CUSTOMER არსის ტიპს Pine Valley ავეჯის კომპანიისთვის (იხ. სურათი 2-22) შესაბამისი CUSTOMER რელაცია გრაფიკული ფორმით არის ნაჩვენები ნახაზზე 4-8 ბ. ამ ფიგურაში და ამ მონაკვეთში მოცემულია მხოლოდ რამდენიმე ძირითადი ატრიბუტი თითოეული რელაციისათვის, რათა გავამარტივოთ ფიგურები.

**FIGURE 4-8** Example of mapping a regular entity  
(a) CUSTOMER entity type

(b) CUSTOMER relation



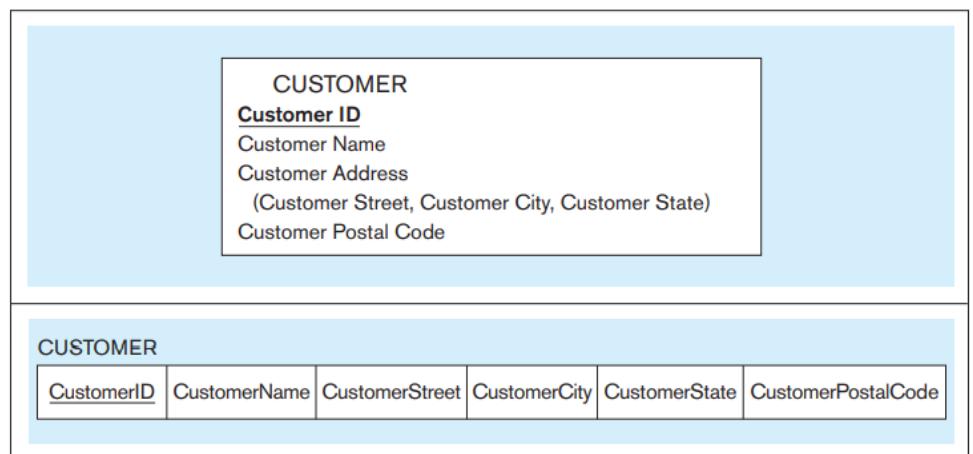
#### კომპოზიტიური ატრიბუტები

როდესაც ჩვეულებრივი არსის ტიპს აქვს კომპოზიტური ატრიბუტი, კომპოზიტური ატრიბუტის მხოლოდ მარტივი კომპონენტები შედის ახალ რელაციაში, როგორც მისი

ატრიბუტები. დიაგრამა 4-9 გვიჩვენებს 4-8 ნახაზზე მოყვანილი მაგალითის ვარიანტს, სადაც მომხმარებლის მისამართი წარმოდგენილია კომპოზიციური ატრიბუტი კომპონენტებით **Street, City** და **State** (იხ. სურათი 4-9a). ეს ობიექტი მონიშნულია CUSTOMER რელაციაში, რომელიც შეიცავს მარტივი მისამართის ატრიბუტებს, როგორც ეს ნაჩვენებია ნახაზზე 4-9b. მიუხედავად იმისა, რომ მომხმარებლის სახელი არის გამოსახული, როგორც მარტივი ატრიბუტი დიაგრამა 4-9 ა-ში, იგი შეიძლებოდა ყოფილიყო მოდელირებული (და, პრაქტიკულად, იქნებოდა) როგორც კომპოზიციური ატრიბუტი კომპონენტებით: **Last Name**, **First Name** და **Middle Initial** (გვარი, სახელი და შუა ასო) CUSTOMER რელაციის დაპროექტებისას (სურათი 4-9 b), შეგვიძლიათ ავირჩიოთ ამ მარტივი ატრიბუტების გამოყენება CustomerName-ის ნაცვლად. კომპოზიციურ ატრიბუტებთან შედარებით, მარტივი ატრიბუტები აუმჯობესებს მონაცემთა ხელმისაწვდომობას და ხელს უწყობს მონაცემთა ხარისხის შენარჩუნებას. მაგალითად, მონაცემთა საანგარიშო და სხვა შედეგობრივი მიზნებისათვის მრავალად უფრო ადვილია, თუ CustomerName წარმოდგენილი იქნება მისი კომპონენტებით (გვარი, სახელი და შუა ასო ცალკე). ამ გზით, მონაცემების შესახებ ინფორმაციის წარმოდგენის ნებისმიერ პროცესს შეუძლია შექმნას მისთვის საჭირო ფორმატი.

**FIGURE 4-9 Example of mapping a composite attribute**  
**(a) CUSTOMER entity type with composite attribute**

**(b) CUSTOMER relation with address detail**



### მრავალმნიშვნელოვანი ატრიბუტი

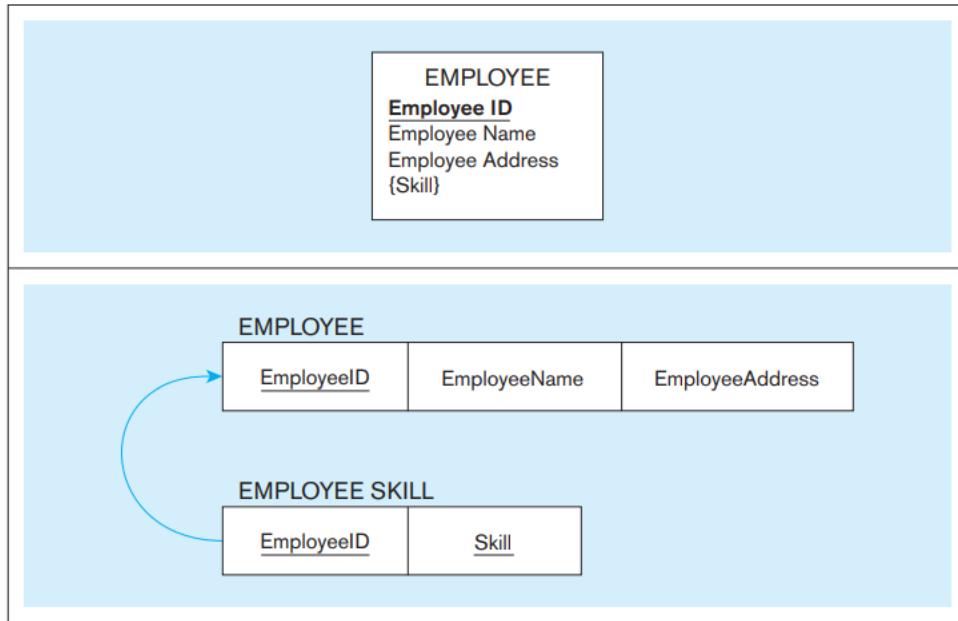
როდესაც ჩვეულებრივი არისი ტიპი შეიცავს მრავალმნიშვნელოვან ატრიბუტს, იქმნება ორი ახალი რელაცია (ნაცვლად ერთის). პირველი რელაცია შეიცავს არსის ტიპის ყველა ატრიბუტს, გარდა მრავალმნიშვნელოვანი ატრიბუტისა. მეორე რელაცია შეიცავს ორ ატრიბუტს, რომლებიც ქმნიან მეორე რელაციის პირველად გასაღებს. ამ ატრიბუტებიდან პირველი არის პირველადი გასაღები პირველი რელაციიდან, რომელიც ხდება მეორე რელაციის გარე გასაღები. მეორე არის მრავალმნიშვნელოვანაი ატრიბუტი. მეორე რელაციის სახელი უნდა ასახავდეს მრავალმნიშვნელოვანი ატრიბუტის მნიშვნელობას.

რელაცია EMPLOYEE SKILL არ შეიცავს არაგასაღებ ატრიბუტებს (ასევე უწოდებენ დესკრიპტორებს). თითოეულ სტრიქონში უბრალოდ ფიქსირდება ის ფაქტი, რომ კონკრეტული თანამშრომელი ფლობს კონკრეტულ უნარს. ეს საშუალებას გვაძლევთ

მომხმარებლებს შევთავაზოთ, ამ რელაციაზე ახალი ატრიბუტების დამატების შესაძლებლობა. მაგალითად, ატრიბუტები YearsExperience და/ან CertificationDate შეიძლება იყოს შესაბამისი ახალი მნიშვნელობები, ამ რელაციაზე დამატებისთვისნ. თუ თვითონ SKILL საჭიროებს დამატებით ატრიბუტებს, შეგვიძლია შევქმნათ ცალკე SKILL კავშირი. ამ შემთხვევაში, EMPLOYEE SKILL ხდება ასოცირებული არსი EMPLOYEE-სა და SKILL-ს შორის.

მრავალმნიშვნელოვანი ატრიბუტის ასახვის მაგალითი მოყვანილია ნახ.4-10-ზე.

თუ არსის ტიპი შეიცავს ბევრ მრავალმნიშვნელოვან ატრიბუტს, თითოეული მათგანი გადაიქცევა ცალკეულ რელაციად.



**FIGURE 4-10** Example of mapping an entity with a multivalued attribute  
(a) EMPLOYEE entity type with multivalued attribute

## ბიჯი 2: სუსტი არსის ასახვა

როგორც აღვნიშნეთ, სუსტი არსის ტიპს არ შეუძლია დამოუკიდებელი არსებობა, არსებობს მხოლოდ იდენტიფიცირების გზით სხვა არსის ტიპთან, რომელსაც ეწოდება მფლობელი. სუსტი არსის ტიპს არ აქვს სრული იდენტიფიკატორი, მაგრამ მას უნდა ჰქონდეს ატრიბუტი, სახელწოდებით ნაწილობრივი იდენტიფიკატორი, რომელიც საშუალებას იძლევა განასხვაოს სუსტი არსის სხვადასხვა შემთხვევები თითოეული მფლობელი არსის ეგზემპლარისთვის.

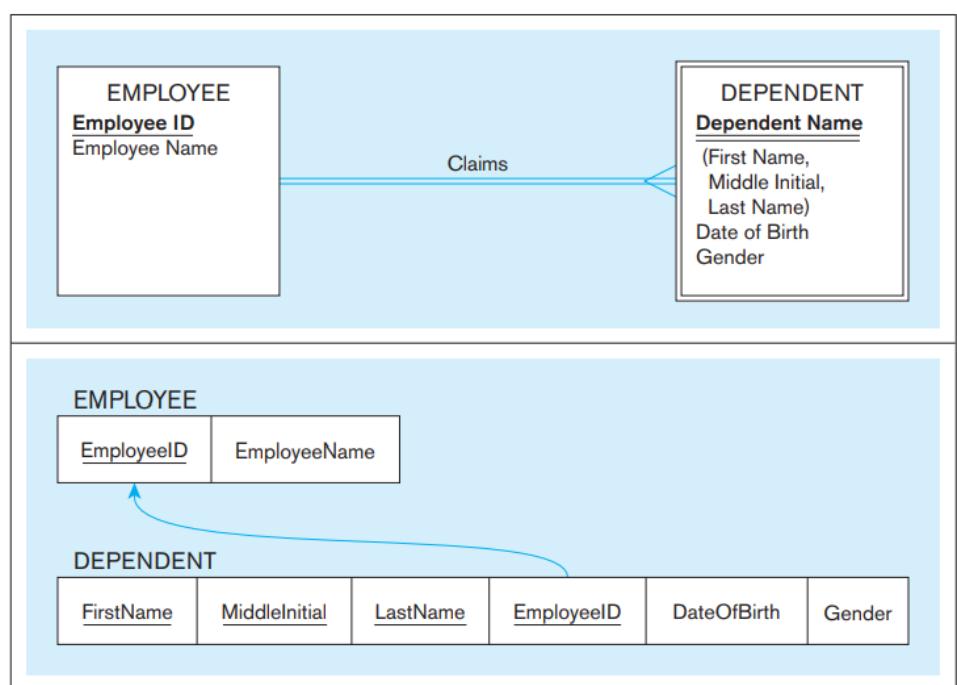
სუსტი არსის ასახვისათვის პირველ რიგში უნდა განხორციელდეს იგივე პროცედურები რაც არწერილი იყო პირველ ბიჯში - თითოეული სუსტი არსის ტიპისთვის შევქმნათ ახალი რელაცია და შეიტანეთ ყველა მარტივი ატრიბუტი (ან კომპოზიციური ატრიბუტების მარტივი

კომპონენტები), როგორც ამ რელაციის ატრიბუტები. შემდეგ ამ ახალ რელაციაში შევიტანოთ მაიდენტიფიცირებადი რელაციის პირველადი გასაღები, როგორც გარე გასაღები ატრიბუტი. ახალი რელაციის პირველადი გასაღები არის მაიდენტიფიცირებადი რელაციის პირველადი გასაღებისა და სუსტი არსის ტიპის ნაწილობრივი იდენტიფიკატორის კომბინაცია.

ამ პროცესის მაგალითი ნაჩვენებია ნახატზე 4-11. დიაგრამა 4-11a აჩვენებს სუსტი არსის ტიპის DEPENDENT და მისი მაიდენტიფიცირებელი სუბიექტის არსის EMPLOYEE შეერთებას მაიდენტიფიცირებელი კავშირით Claims (იხილეთ სურათი 2-5). გავითვალისწინოთ, რომ ატრიბუტი Dependent Name, რომელიც ამ რელაციის ნაწილობრივი იდენტიფიკატორია, არის კომპოზიციური ატრიბუტი კომპონენტებით: First Name, Middle Initial, და Last Name. ამრიგად, ვთვლით, რომ მოცემული თანამშრომლისთვის ეს ელემენტები ცალსახად განსაზღვრავს დამოკიდებულ პირს.

დიაგრამა 4-11 ბ აჩვენებს ორ რელაციას, რომლებიც წარმოიქმნება ამ E-R დიაგრამის სეგმენტიდან. DEPENDENT- ის პირველადი გასაღები ოთხი ატრიბუტისგან შედგება: EmployeeID, FirstName, MiddleInitial და LastName. DateOfBirth და Gender არის არაგასარები ატრიბუტები. გარე გასაღების კავშირი მის პირვანდელ გასაღებთან ნახატზე მითითებულია ისრით.

**FIGURE 4-11 Example of mapping a weak entity  
(a) Weak entity DEPENDENT**



პრაქტიკში, ხშირად გამოიყენება ალტერნატიული მიდგომა DEPENDENT პირველადი გასაღების გასამარტივებლად: შევქმნათ ახალი ატრიბუტი (ე.წ. DependentID), რომელიც გამოყენებული იქნება სუროგატი პირველადი გასაღებისთვის, დიაგრამა 4-11 ბ. ამ მიდგომით რელაცია DEPENDENT აქვს შემდეგი ატრიბუტები:

**DEPENDENT(DependentID, EmployeeID, FirstName, MiddleInitial,  
LastName, DateOfBirth, Gender)**

DependentID არის უბრალოდ რიგითი ნომერი, რომელიც ენიჭება თითოეულ დასაქმებულზე დამოკიდებულ პირს. გავითვალისწინოთ, რომ ეს გამოსავალი უზრუნველყოფს უნიკალურ იდენტიფიკაციას თითოეული დამოკიდებული ადამიანისთვის.

**როდის იქმნება სუროგატული გასაღები**

სუროგატული გასაღები ჩვეულებრივ იქმნება გასაღების სტრუქტურების გამარტივების მიზნით. სუროგატი გასაღები უნდა შეიქმნას, როდესაც რომელიმე შემდეგი პირობაა დაცული:

- არსებობს კომპოზიტიური პირველადი გასაღები (ისევე როგორც DEPENDENT რელაციის შემთხვევაში, რომელიც ნაჩვენებია ადრე ოთხკომპონენტიანი პირველადი გასაღებით).
- ბუნებრივი პირველადი გასაღები (ანუ გასაღები, რომელიც გამოიყენება ორგანიზაციაში და აღიარებულია კონცეპტუალური მონაცემების მოდელირებაში, როგორც იდენტიფიკატორი) არაეფექტურია. მაგალითად, შეიძლება იყოს ძალიან გრძელი და, შესაბამისად, „მვირადლირებული“ მონაცემთა ბაზის პროგრამული უზრუნველყოფის დამუშავებისას, თუ იგი გამოიყენებული იქნება როგორც გარე გასაღები, სხვა ცხრილებზე წვდომისას.
- ბუნებრივი პირველად გასაღები ხელმეორედ გამოიყენება (ანუ, გასაღები ხელახლა გამოიყენება ან მეორდება პერიოდულად, ისე რომ, ის შეიძლება რეალურად არ იყოს უნიკალური დროთა განმავლობაში); ამ პირობის ზოგადი განაცხადია - როდესაც ბუნებრივი პირველადი გასაღები, ფაქტობრივად, ვერ იქნება გარანტირებული, რომ დროთა განმავლობაში შეინარჩუნებს უნიკალურობას (მაგალითად, შეიძლება არსებობდეს დუბლიკატები, მაგალითად სახელები ან სათაურები).

სუროგატი გასაღების შექმნისას, ბუნებრივი გასაღები ყოველთვის ინახება როგორც არაგასაღები მონაცემები იმავე რელაციაში, რადგან ბუნებრივ გასაღებს აქვს ორგანიზაციული მნიშვნელობა, რომელიც უნდა იყოს დაფიქსირებული მონაცემთა ბაზაში. სინამდვილეში, სუროგატი გასაღები მომხმარებლებისთვის არაფერს ნიშნავს, ამიტომ ისინი მომხმარებელს, როგორც წესი, არასოდეს უჩანს. ამის სანაცვლოდ, ბუნებრივი გასაღებები იდენტიფიკატორად გამოიყენება ძიებებში.

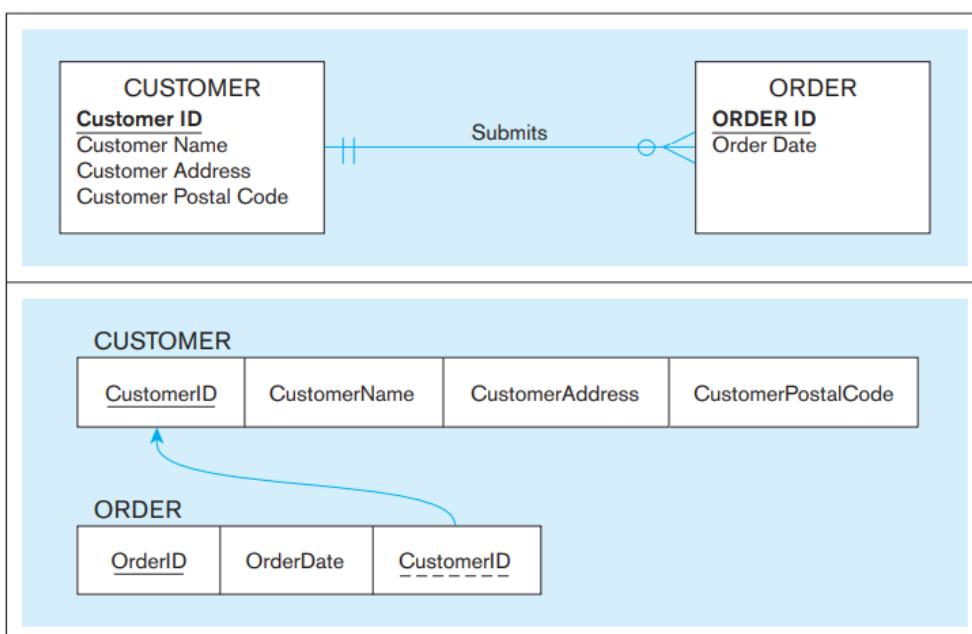
**ბიჯი 3: ორობითი (ბინარული) კავშირის ასახვა**

კავშირის ასახვი პროცედურა დამოკიდებულია როგორც კავშირის ხარისხზე (უნიარული, ორობითი ან სამეული) ასევე კავშირის კარდინალურობაზე.

## ერთი-მრავალთან ბინარული კავშირის ასახვა

თითოეული ორობითი (ბინარული) 1: M კავშირისათვის, პირველ რიგში, იქმნება რელაცია კავშირში მონაწილე ორი არსის თითოეული ტიპისთვის, ბიჯი 1-ში აღწერილი პროცედურის გამოყენებით. შემდეგ, არსის კავშირის ერთი მხარის პირველადი გასაღები ატრიბუტი (ან ატრიბუტები) შეგვაქვს, როგორც გარე გასაღები რელაციაში, რომელიც კავშირის „მრავალის“ მხარეს წარმოადგენს (ამ წესის დამახსოვრებისათვის შეგვიძლია გამოვიყენოთ მნენონიკა: პირველადი გასაღები მიემართება „მრავალის“ მხარეს).

ამ მარტივი პროცესის საილუსტრაციოდ, გამოვიყენოთ კავშირი მომხმარებლებსა და შეკვეთებს შორის Pine Valley ავეჯის კომპანიისთვის (იხ. სურათი 2-22). ეს 1:M კავშირი ილუსტრირებულია ნახაზზე 4-12a. დიაგრამა 4-12b გვიჩვენებს ამ წესის გამოყენების შედეგს არსის ტიპების წარმოსაჩენად 1: M კავშირში. CUSTOMER – ის პირველადი გასაღები CustomerID (ერთი მხარე) შედის ORDER-ში, როგორც გარე გასაღებები. გარე გასაღების კავშირი მითითებულია ისრით. გავითვალისწინოთ, რომ არ არის საჭირო გარე გასაღების ატრიბუტის CustomerID დასახელება იყოს იდენტური კავშირში მონაწილე არსის ტიპის პირველადი გასარებისა. ამასთან, არსებითია, რომ მას ჰქონდეს იგივე დომენი, რაც პირველად გასაღებს, რომელსაც ის მიმართავს.



**FIGURE 4-12** Example of mapping a 1:M relationship  
(a) Relationship between CUSTOMER and ORDER entities

(b) CUSTOMER and ORDER relations with a foreign key in ORDER

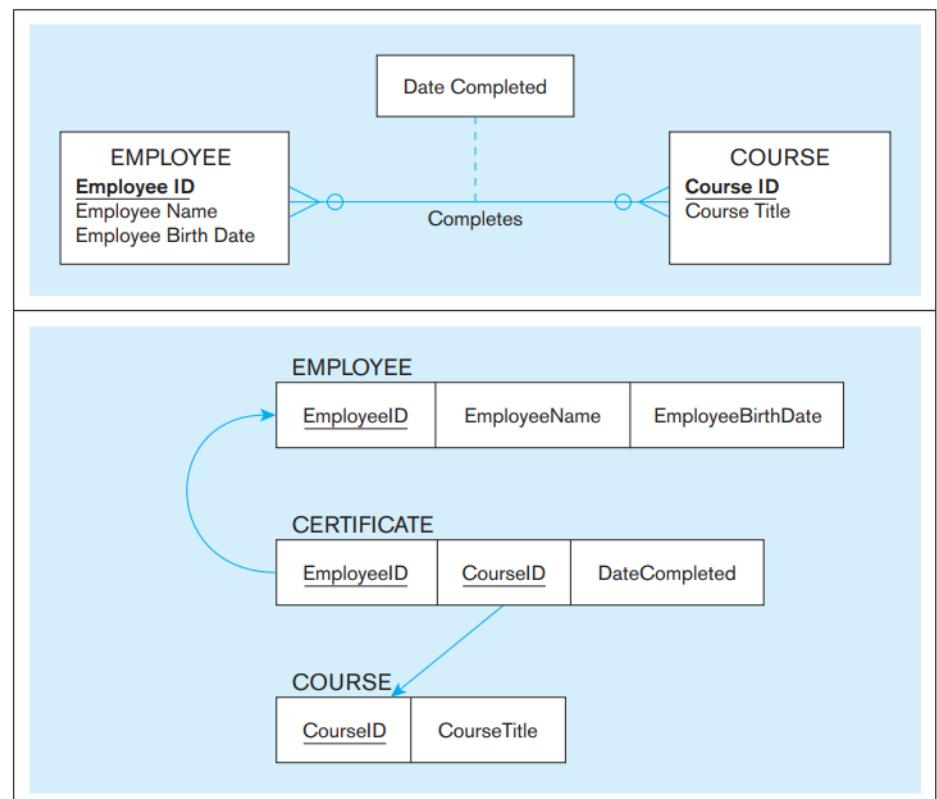
## მრავალი-მრავალთან ბინარული კავშირის ასახვა

დავუშვათ, რომ არსებობს ბინარული მრავალი-მრავალთან (M:N) კავშირი ორ არსის ტიპებს შორის, A და B. ასეთი კავშირისთვის იქმნება ახალი რელაცია C. შევიტანოთ, როგორც გარე გასაღების ატრიბუტები C-ში თითოეული ორი არსის ტიპის პირველადი გასაღები, როგორც

გარე გასაღები. ეს ატრიბუტები ერთად გახდება C-ს პირველადი გასაღები. ნებისმიერი არაგასაღები ატრიბუტი, რომლებიც ასოცირდება M:N კავშირში, შედის C რელაციაში.

დიაგრამა 4-13 გვიჩვენებს ამ წესის გამოყენების მაგალითს. დიაგრამა 4-13a გვიჩვენებს კავშირი Completes („დასრულებულია“) არსის ტიპებს EMPLOYEE და COURSE-ს შორის, სურათი 2-11a-დან. დიაგრამა 4-13 ბ აჩვენებს სამ რელაციას (EMPLOYEE, COURSE და CERTIFICATE), რომლებიც წარმოიქმნება არსის ტიპებისა და Complete კავშირისაგან. თუ Completes წარმოდგენილი იქნებოდა როგორც ასოციაციური არსი, როგორც ეს გაკეთებულია ნახაზზე 2-11b, მსგავსი შედეგი მოხდებოდა, (ამ საკითხს განვიხილავთ მოგვიანებით). M:N კავშირის შემთხვევაში, პირველ რიგში იქმნება რელაცია თითოეული რეგულარული არსის ტიპიდან EMPLOYEE და COURSE. შემდეგ იქმნება ახალი რელაცია (სახელად CERTIFICATE ნახაზზე 4-13b) Completes კავშირისათვის. CERTIFICATE-ის პირველადი გასაღები არის EmployeeID-ისა და CourseID-ის კომბინაცია, რომლებიც EMPLOYEE-ისა და COURSE-ის შესაბამისი პირველადი გასაღებებია. როგორც დიამაგრზე მითითებული, ეს ატრიბუტები გარე გასაღებებია, რომლებიც შესაბამის წერტილზე გასასვლელად "მიუთითებენ". არაგასაღები ატრიბუტი DateCompleted ასევე ჩნდება CERTIFICATE რელაციაში. მართალია აქ ნაჩვენები არ არის, მაგრამ სასურველია შეიქმნას სუროგატული პირველადი გასაღები CERTIFICATE რელაციისათვის.

**FIGURE 4-13 Example of mapping a M:N relationship  
(a) Completes relationship (M:N)**



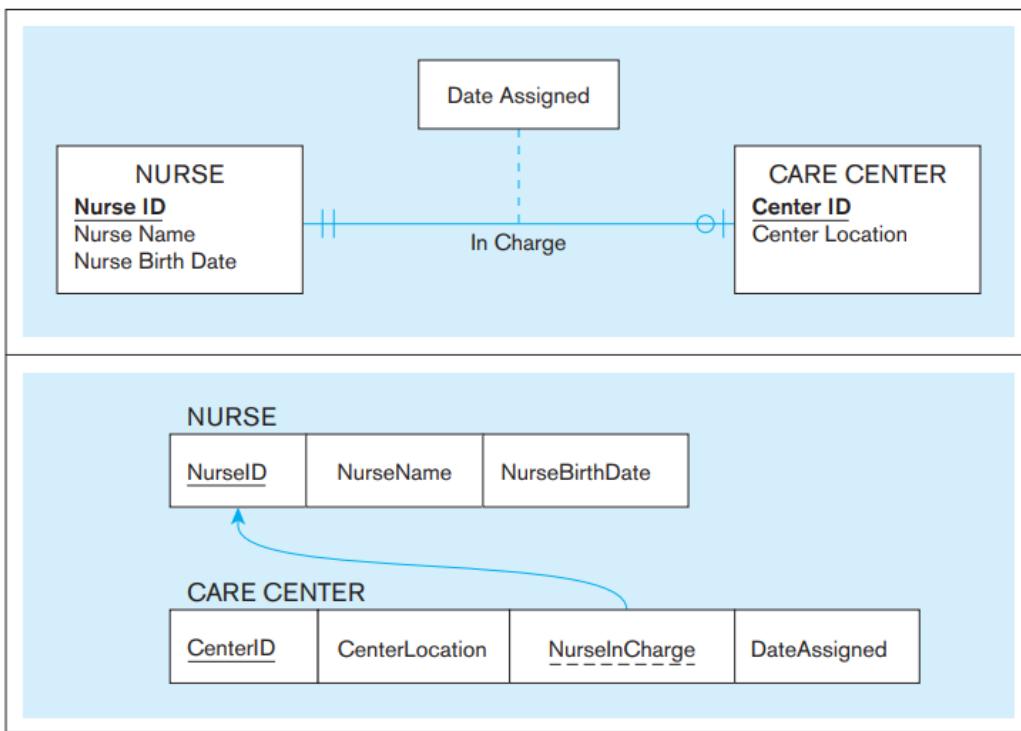
## ერთი-ერთთან ბინარული კავშირის ასახვა

ბინარული კავშირი ერთი-ერთთან შეიძლება განიხილებოდეს, როგორც ერთი-მრავალთან კავშირის კერძო შემთხვევა. ამგვარი კავშირის რელაციად ასახვის პროცესი მოითხოვს ორ ბიჯს. პირველი, იქმნება ორი რელაცია, თითო მონაწილე არსის თითოეული ტიპებისთვის. მეორე, ერთი რელაციის პირველადი გასაღები ჩაისმის გარე გასაღებად, მეორე რელაციაში.

1:1 კავშირშირისას ერთი-ერთი მიმართულებით კავშირი თითქმის ყოველთვის არასავალდებულოა, ხოლო მეორე მიმართულებით სავალდებულო (ამ ტერმინების აღნიშვნის გადახედვა შეგიძლიათ დიაგრამაზე 2-1). კავშირის არასავალდებულო მხარეს არსებულ რელაციაში უნდა მიუთითოთ გარე გასაღები, რომელიც მიუთითებს არსის ტიპის პირველად გასაღებზე, რომელსაც აქვს სავალდებულო მონაწილეობა 1:1 კავშირში. ეს მიდგომა ხელს შეუშლის გარე გასაღების ატრიბუტში null მნიშვნელობების შენახვის აუცილებლობას. თავად კავშირთან დაკავშირებული ნებისმიერი ატრიბუტი ასევე შედის იმავე რელაციაში, სადაც გარე გასაღები (აქვე აღვნიშნოთ, რომ ნოზმალიზაციის თვალსაზრისით, რასაც მოგვიანებით განვიხილავთ, თუ კავშირის შეიცავს ატრიბუტებს, უმჯობესია კავშირი წარმოავდგინოთ ასოციაციური არსის სახით).

ამ პროცედურის გამოყენების მაგალითი ნაჩვენებია ნახაზზე 4-14. დიაგრამა 4-14a აჩვენებს ორობითი 1:1 კავშირს არსის ტიპების NURSE (მედდა) და CARE CENTER (მზრუნველობის ცენტრი) შორის. თითოეულ მზრუნველობის ცენტრს უნდა ჰყავდეს ექთანი, რომელიც პასუხისმგებელია ამ ცენტრში. ამრიგად, CARE CENTER-ის კავშირი NURSE-თან სავალდებულოა, ხოლო NURSE-ს კავშირი CARE CENTER-თან არასავალდებულოა (ვინაიდან ნებისმიერი ექთანი შეიძლება იყოს ან არ იყოს პასუხისმგებელი ზრუნვის ცენტრში). ატრიბუტი Date Assigned (დანიშვნის თარიღი) თან ერთვის In Charge კავშირს.

ამ კავშირების ასახვის შედეგი რელაციების კავშირებით ნაჩვენებია ნახაზზე 4-14 ბ. ორი რელაცია NURSE და CARE CENTER იქმნება ორი არსის ტიპებისგან. იმის გამო, რომ CARE CENTER არასავალდებულო მონაწილეა, ამ რელაციაში განთავსებულია გარე გასაღები. ამ შემთხვევაში, გარე გასაღებია NurseInCharge. მას აქვს იგივე დომენი, როგორც NurseID და პირველადი გასაღების მიმართება ნაჩვენებია ნახაზზე. ატრიბუტი DateAssigned ასევე მდებარეობს CARE CENTER-ში და ნებადართული არ იქნება მისი ნულოვანობა.



#### ბიჯი 4: ასოციაციური არსების ასახვა

როგორც ადრე აღვნიშნეთ, როდესაც მონაცემთა მოდელის შექმნისას „მრავალი-მრავალთან“ კავშირთან შეჯახებისას შესაძლებელია კავშირის მოდელირებისათვის ავირჩიოთ ასოციაციური არსის გამოყენება E-R დიაგრამაზე. ეს მიდგომა ყველაზე შესაფერისა მაშინ, როდესაც საბოლოო მომხმარებელს შეუძლია უკეთესად წარმოაჩინოს კავშირის, როგორც არსის ტიპი და არა როგორც M:N კავშირი. ასოციაციური არსის ასახვა მოიცავს იგივე ნაბიჯებს, როგორიცაა M: N კავშირის ასახვა, როგორც ეს აღწერილია მე -3 ბიჯზე.

პირველი ბიჯი არის სამი რელაციის შექმნა: თითო თითო ორი მონაწილე არსის ტიპისთვის და მესამე ასოციაციური არსისთვის. ასოციაციური არსისაგან წარმოქმნილ რელაციას ასოციაციურ რელაციად ვგულისხმობთ. მეორე ბიჯი დამოკიდებულია იმაზე, იყო თუ არა E-R დიაგრამაზე ასოციაციური არსისთვის დანიშნული იდენტიფიკატორი.

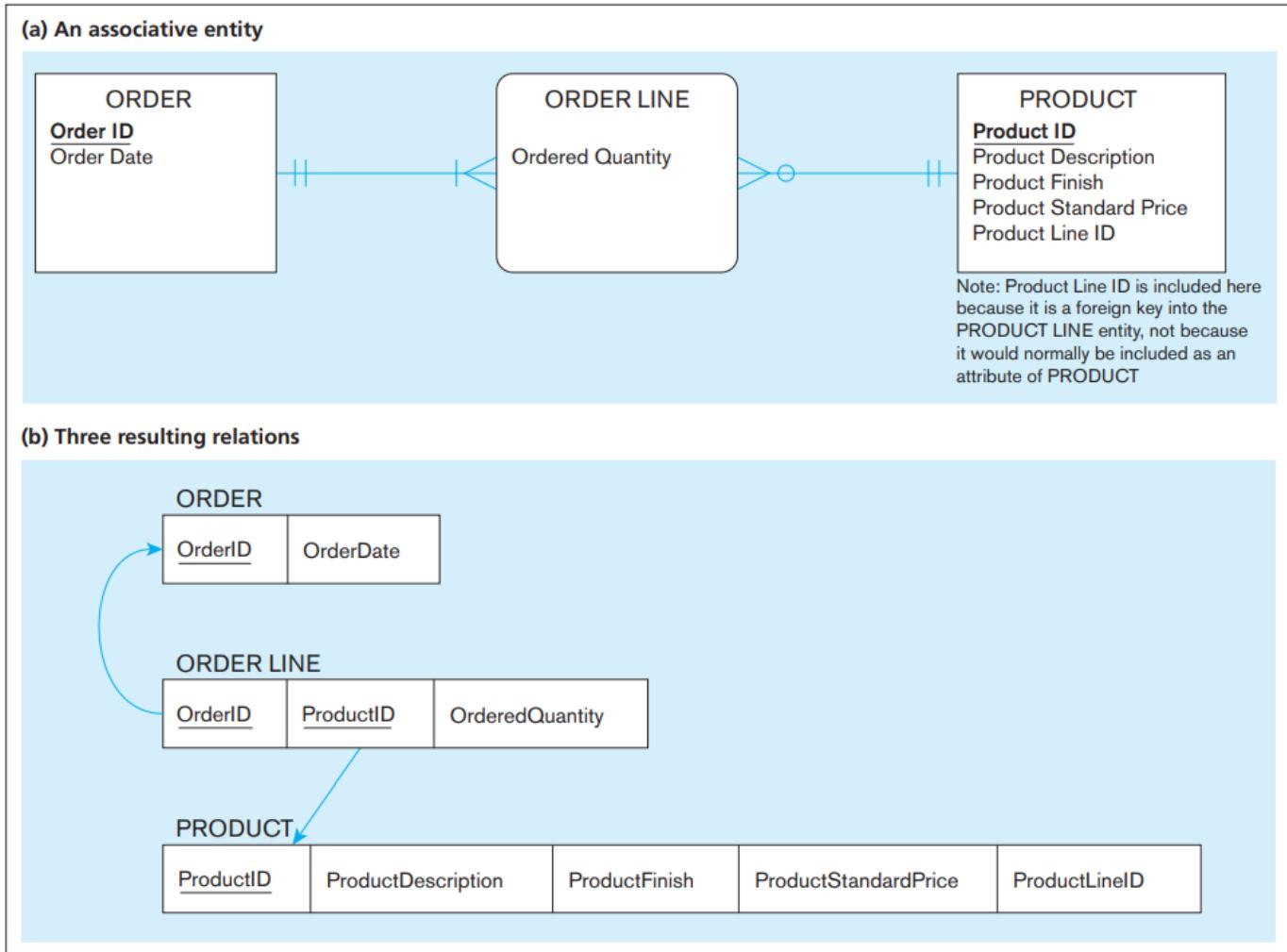
#### იდენტიფიკატორი არ არის დანიშნული

თუ იდენტიფიკატორი არ იყო დანიშნული, ასოციაციური რელაციის ნაგულისხმევი პირველადი გასაღები არის კომპოზიტური გასაღები, რომელიც შედგება ორი ძირითადი გასაღები ატრიბუტისგან დანარჩენი ორი რელაციიდან. ეს ატრიბუტები შემდეგ გარე გასაღებებს წარმოადგენს, რომლებიც სხვა დანარჩენ რელაციებს უკავშირდება.

ამ საქმის მაგალითი ნაჩვენებია ნახაზზე 4-15. დიაგრამა 4-15a აჩვენებს ასოციაციურ არსს ORDER LINE, რომელიც აკავშირებს ORDER და PRODUCT არსის ტიპებს Pine Valley

Furniture Company- ში (იხ. სურათი 2-22). დიაგრამა 4-15 ბ აჩვენებს სამ რელაციას, რომელიც ამ სქემიდან გამომდინარებს. გავითვალისწინოთ ამ მაგალითის მსგავსება M:N კავშირთან, რომელიც ნაჩვენებია ნახაზზე 4-13.

**FIGURE 4-15 Example of mapping an associative entity**



### იდენტიფიკატორი დანიშნულია

ზოგჯერ მონაცემთა მოდელის შემმუშავებლები ერთ-ერთ ატრიბუტს ანიჭებენ იდენტიფიკატორის ფუნქციას ასოციაციური არსის ტიპისთვის E-R დიაგრამაზე. ეს ფაქტი ორმა მიზეზმა შეიძლება განაპირობოს მონაცემთა კონცეპტუალური მოდელის შემუშავების ეტაპზე:

1. ასოციაციური არსის ტიპს გააჩნია ბუნებრივი ერთი ატრიბუტი იდენტიფიკატორი, რომელიც ნაცნობია საბოლოო მომხმარებლებისთვის.

2. ნაგულისხმევ იდენტიფიკატორს (რომელიც შედგება იდენტიფიკატორებისგან თითოეული არსის თითოეული ტიპებისთვის) არ შეუძლია ცალსახად განსაზღვროს ასოციაციური არსის ეგზემპლარები.

ეს მოტივები ავსებს სუროგატი გასაღების შექმნის ძირითადი მიზეზებს. ასოცირებული არსის ჩანაწერის პროცესი ამ შემთხვევაში იცვლება: იქმნება ახალი (ასოციაციური) რელაცია ასოციაციური არსის წარმოსადგენად; ამასთან, ამ რელაციის ძირითადი გასაღები არის E-R დიაგრამაზე მინიჭებული იდენტიფიკატორი (და არა ნაგულისხმევი გასაღები). ორი ძირითადი არსის ძირითადი გასაღები ასოცირებულ რელაციაში შედის როგორც გარე გასაღები.

ამ პროცესის მაგალითი ნაჩვენებია ნახაზზე 4-16. დიაგრამა 4-16a გვიჩვენებს ასოციაციური არსის ტიპს SHIPMENT, რომელიც აკავშირებს CUSTOMER და VENDOR არსის ტიპებს.

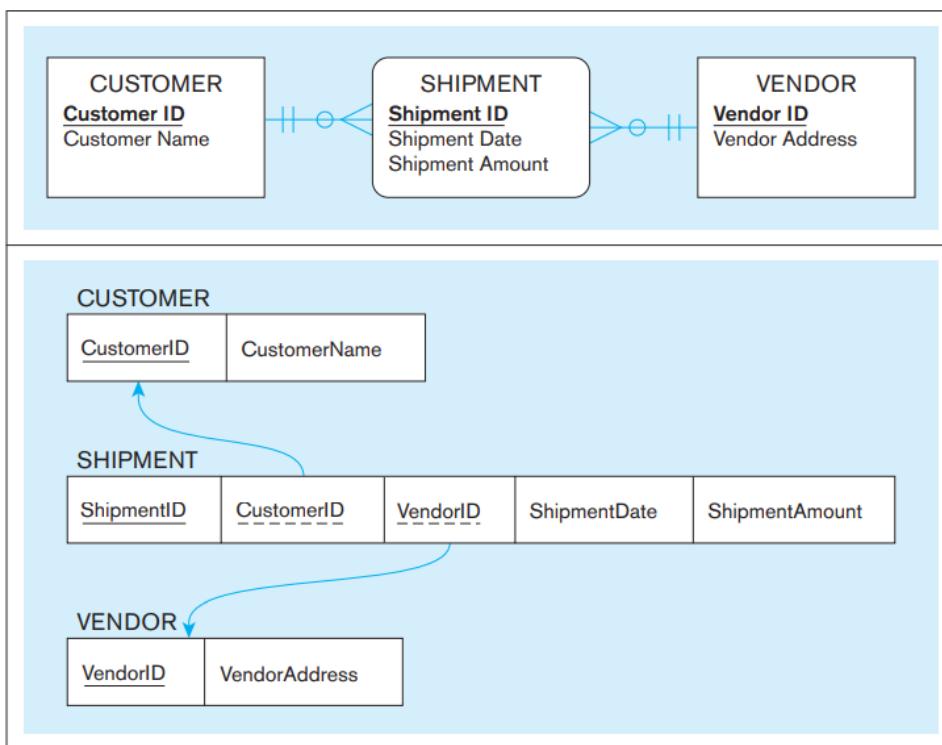
Shipment ID შეირჩა SHIPMENT- ის იდენტიფიკატორად ორი მიზეზის გამო:

1. Shipment ID არის ბუნებრივი იდენტიფიკატორი ამ არსისთვის, რომელიც ძალიან ნაცნობია საბოლოო მომხმარებლებისთვის.
2. ნაგულისხმევი იდენტიფიკატორი, რომელიც შედგება CustomerID და VendorID კომბინაციისგან, ცალსახად არ განსაზღვრავს გადაზიდვის შემთხვევებს. სინამდვილეში, მოცემული გამყიდველი, როგორც წესი, ბევრ გადაზიდვას ახორციელებს მოცემულ მომხმარებელზე. ატრიბუტის თარიღის დამატებაც არ იძლევა გარანტიას უნიკალურობის, რადგან მოცემული თარიღისთვის კონკრეტული გამყიდველის მიერ შეიძლება იყოს ერთზე მეტი გადაზიდვა განხორციელებული. სუროგატი გასაღებს ShorageID ცალსახად განსაზღვრავს თითოეულ გადაზიდვას.

SHIPMENT ასოციაციურ არსთან ასოცირებული ორი არაგასაღები ატრიბუტია Shipment Date და Shipment Amount.

ამ არსების რელაციაბთან შეთანადების ასახვის შედეგი ნაჩვენებია დიაგრამა 4-16 ბ. ახალ ასოციაციურ რელაციას SHIPMENT დაარქვეს. პირველადი გასაღებია ShorageID, CustomerID და VendorID მოცემულია როგორ გარე გასაღებები ამ რელაციაში და ShriageDate და ShriageAmount არიან არა გასაღები ატრიბუტები. ასევე შესაძლებელია, დიზაინერმა გადაწყვიტოს, როგორც ლოგიკური მოდელირების პროცესის ნაწილი, დაამატოს სუროგატი გასაღები იმ რელაციაში, რომელსაც ადრე არ ჰქონდა. ამ შემთხვევებში რეკომენდებულია კონცეპტუალური მოდელის განახლება, მისი მუდმივი შენარჩუნების მიზნით.

**FIGURE 4-16** Example of mapping an associative entity with an identifier  
 (a) SHIPMENT associative entity



### ბიჯი 5: უნიარული კავშირების ასახვა

ადრე ჩვენ განვსაზღვრეთ უნიარული კავშირი, როგორც ურთიერთობა ერთი არსის ტიპის ეგზემპლარებს შორის. უნიარულ კავშირებს რეკურსიულ კავშირებსაც უწოდებენ. უნიარული კავშირების ორი უმნიშვნელოვანესი შემთხვევა განიხილება - კავშირი „ერთი-მრავალთან“ და „მრავალი-მრავალთან“. ამ ორ შემთხვევას განვიხილავთ ცალკე, რადგან ასახვის მიდგომა ორი ტიპისთვის გარკვეულწილად განსხვავებულია.

### უნიარული კავშირი „ერთი-მრავალთან“

არსის ტიპი უნიარულ კავშირში აისახება რელაციაში პროცედურით, რომელიც აღწერილია ბიჯში 1. შემდეგ, იმავე რელაციას ემატება გარე გასაღები ატრიბუტი; ეს ატრიბუტი ეფუძნება პირველადი გასაღების მნიშვნელობებს იმავე რელაციაში (ამ გარე გასაღებას იგივე დომენი უნდა ჰქონდეს, როგორც პირველად გასაღებს). გარე გასაღების ამ ტიპს რეკურსიულ გარე გასაღებს უწოდებენ.

დიაგრამა 4-17a გვიჩვენებს უნიარულ კავშირს, სახელწოდებით Manages, რომელიც ორგანიზაციის თითოეულ თანამშრომელს უკავშირებს სხვა თანამშრომელს, რომელიც მისი

მენეჯერია. თითოეულ თანამშრომელს შეიძლება ჰყავდეს ერთი მენეჯერი; მოცემულმა თანამშრომელმა შეიძლება მართოს ნულიდან მრავალი თანამშრომელი.

EMPLOYEE რელაცია, რომელიც წარმოიქმნება ამ არსის და კავშირის ასახვიდან გამომდინარე, ნაჩვენებია ნახაზზე 4-17b. (რეკურსიული) გარე გასაღებს რელაციაში დაერქვა ManagerID. ამ ატრიბუტს აქვს იგივე დომენი, როგორც პირველად გასაღებს EmployeeID. ამ რელაციის თითოეული სტრიქონი ინახავს შემდეგ მონაცემებს მოცემული თანამშრომლისთვის: EmployeeID, EmployeeName, EmployeeDateOfBirth და ManagerID (ანუ EmployeeID ამ თანამშრომლის მენეჯერისთვის). გავითვალისწინოთ, რომ რადგან ეს გარე გასაღებია, ManagerID მიუთითებს EmployeeID.

#### უნარული კავშირი „მრავალი-მრავალთან“

ამ ტიპის კავშირით იქმნება ორი რელაცია: ერთი წარმოადგენს არსის ტიპს და მეორე - ასოციაციურ არსს M:N კავშირის წარმოსადგენად. ასოციაციური რელაციის პირველადი გასაღები შედგება ორი ატრიბუტისგან. ამ ატრიბუტებს (რომელთაც არ უნდა ჰქონდეთ ერთიდაიგვე სახელი) ორივე იღებს თავის მნიშვნელობებს პირველი რელაციის პირველადი გასაღებიდან. კავშირის ნებისმიერი არაგასაღები ატრიბუტი შედის ასოციაციურ რელაციაში.

უნიარული M: N კავშირის ასახვის მაგალითი ნაჩვენებია ნახაზზე 4-18. დიაგრამა 4-18 ა აჩვენებს bill-of-materials კავშირს (კავშრი არწერს ურთიერთდამოკიდებულებას იმ ნივთებს შორის, რომლებიც სხვა ნივთებისაგან ან კომპონენტებისგან არის აწყობილი). კავშირი (ე.წ.- Contains შეიცავს) არის M:N, რადგან მოცემული ნივთი შეიძლება შეიცავდეს მრავალ კომპონენტს (ნივთს) და, პირიქით, ნივთი (კომპონენტი) გამოყენება მრავალ სხვა ნივთში (კომპონენტში).

რელაციები, რომლებიც წარმოიქმნება ამ არსის და მისი კავშირის სქემიდან ნაჩვენებია ნახაზზე 4-18 ბ. ITEM რელაცია პირდაპირ იმავე ტიპის არსისდან არის ასახული. კომპონენტი არის ასოციაციური რელაცია, რომლის პირველადი გასაღები შედგება ორი ატრიბუტისგან, რომლებსაც უწოდეს ItemNo და ComponentNo. ატრიბუტი Quantity არის ამ რელაციის არაგასაღები ატრიბუტი, რომელიც მოცემული ნივთისთვის აღრიცხავს ამ ერთეულში გამოყენებული კონკრეტული კომპონენტის ერთეულის რაოდენობას. გავითვალისწინოთ, რომ ItemNo და ComponentNo მიუთითებენ ITEM რელაციის პირველად გასაღებზე (ItemNo). ბუნებრივია ამ რელაციისთვის სუროგატი გასაღების მიცემა, რომ თავიდან იქნას აცილებული კომპოზიტურ გასაღებთან დაკავშირებული რაიმე პრაქტიკული სირთულე.

ამ რელაციის გამოთხოვნა ადვილად შეგვიძლია, მაგალითად, რათა განვსაზღვროთ, მოცემული ნივთის კომპონენტები. შემდეგი SQL მოთხოვნა ჩამოთვლის ყველა კომპონენტებს (და მათი რაოდენობას) პუნქტის 100 ნომრისთვის:

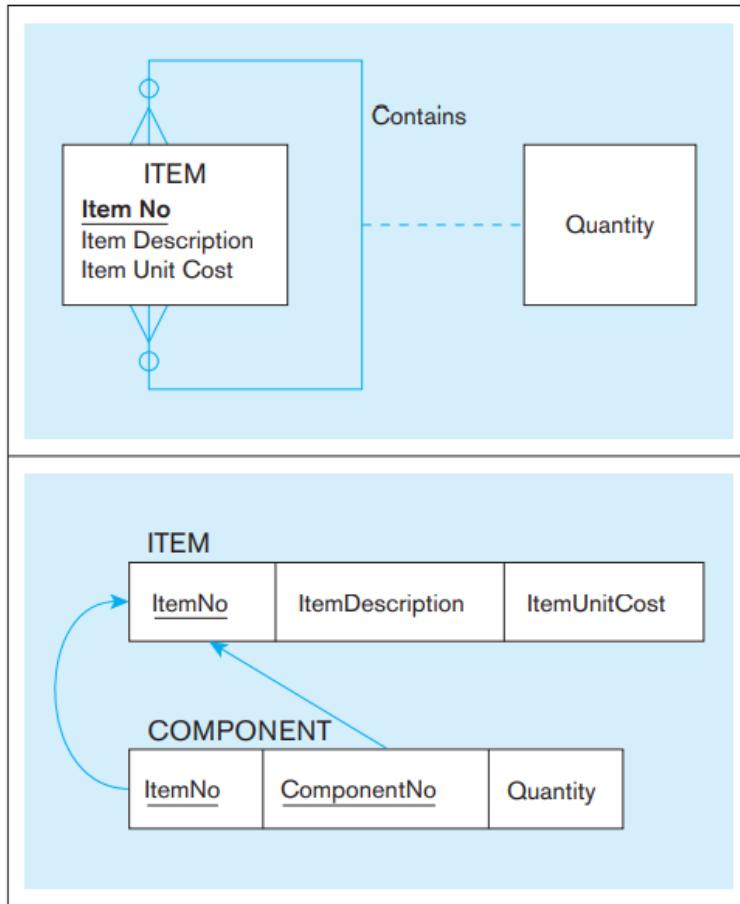
---

SELECT ComponentNo, Quantity

```

FROM Component_T
WHERE ItemNo = 100;

```



**FIGURE 4-18** Example of mapping a unary  $M:N$  relationship  
**(a) Bill-of-materials relationship**  
**Contains ( $M:N$ )**

#### ბიჯი 6: სამეული (და n-ური) სავშირების ასახვა

გავიხსენოთ, რომ სამეული კავშირი არის ურთიერთობა სამ არსის ტიპებს შორის. საზოგადოდ მიღებულია სამეული კავშირის ასოციაციურ არსად გარდაქმნა, რათა უფრო ზუსტად მოხდეს მონაწილეობის შეზღუდვების წარმოდგენა.

ასოციაციური არსის ტიპის ასაგებად, რომელიც აკავშირებს სამი რეგულარული არსის ტიპს, ჩვენ ვქმნით ახალ ასოციაციურ რელაციას. ამ რელაციის ნაგულისხმევი პირველადი გასაღები შედგება სამი მონაწილე არსის ტიპების პირველადი გასაღები ატრიბუტებისაგან (ზოგიერთ შემთხვევაში საჭიროა დამატებითი ატრიბუტები უნიკალური პირველად გასაღების შესაქმნელად). ეს ატრიბუტები შემდეგ მოქმედებს გარე გასაღებების როლში, რომლებიც მიუთითებენ მონაწილე არსების ტიპების ინდივიდუალურ პირველად გასაღებებზე. ასოციაციური არსის ტიპის ნებისმიერი ატრიბუტი ხდება ახალი რელაციის ატრიბუტი.

სამეული კავშირის ასახვის მაგალითი (წარმოდგენილია როგორც ასოციაციური არსის ტიპი) ნაჩვენებია ნახაზზე 4-19. დიაგრამა 4-19a არის E-R სეგმენტი (ან ხედი), რომელიც წარმოადგენს პაციენტს, რომელსაც მკურნალობს ექიმი. ასოციაციური არსის ტიპს PATIENT TREATMENT (პაციენტის მკურნალობა) აქვს ატრიბუტები PTreatment Date, PTreatment time, და PTreatment results; მნიშვნელობები ჩაიწერება ამ ატრიბუტებისთვის თითოეული ეგზემპლარისათვის PATIENT TREATMENT-ში.

ამ წარმოდგენის ასახვის შედეგი ნაჩვენებია ნახაზზე 4-19b. პირველადი გასაღების ატრიბუტები PatientID, PhysicianID და TreatmentCode ხდებიან გასაღებები PATIENT TREATMENT-ში. TREATMENT-ის გარე გასაღებს ერქმევა PTreatmentCode PATIENT LREATMENT-ში. ამ სვეტის სახელს ვიყენებთ იმის საილუსტრაციოდ, რომ გარე გასაღების სახელი არა სავალდებულო იყოს იგივე რაც პირველადი გასაღების სახელი, რომელსაც ის უკავშირდება, რადგან მნიშვნელობები მოდის ერთი და იგივე დომენიდან. ეს სამი ატრიბუტი არის პაციენტის მკურნალობის პირველადი გასაღების კომპონენტები. ამასთან, ისინი ცალსახად არ განსაზღვრავენ მოცემულ მკურნალობას, რადგან პაციენტმა შეიძლება იგივე მკურნალობა მიიღოს იმავე ექიმისგან ერთზე მეტჯერ. შესაძლოა ატრიბუტის Date (თარიღის), როგორც პირველადი გასაღების ნაწილად (სხვა დანარჩენ სამ ატრიბუტთან ერთად) ჩართვამ წარმოქმნის პირველადი გასაღები. ეს ასე მოხდება, თუ მოცემული პაციენტი კონკრეტული ექიმისგან მხოლოდ ერთ მკურნალობას მიიღებს მოცემულ თარიღში. ამასთან, სავარაუდოდ, ეს ასე არ არის. მაგალითად, პაციენტს შეუძლია მიიღოს მკურნალობა დილით, შემდეგ ისევ იგივე მკურნალობა დღის მეორე ნახევარში. ამ საკითხის მოსაგვარებლად, ჩვენ პირველადი გასაღების ნაწილად მოვიყვანთ PTreatmentDate და PTreatmentTime. ამიტომ, პაციენტის მკურნალობის პირველადი გასაღები შედგება ხუთი ატრიბუტისგან, რომლებიც ნაჩვენებია ნახაზზე 4-19 ბ: PatientID, PhysicianID, TreatmentCode, PTreatmentDate და PTreatmentTime. ერთადერთი არაფრისმომცემი ატრიბუტი არის PTreatmentResults.

მიუხედავად იმისა, რომ ეს პირველადი გასაღები ტექნიკურად სწორია, ის რთულია, რთულია მისი მართვა და მიღდრევილია შეცდომებისკენ. უკეთესი მიდგომაა სუროგატი გასაღების შემოღება, მაგალითად, PTreatmentID, ანუ სერიული ნომერი, რომელიც ცალსახად განსაზღვრავს თითოეულ მკურნალობას. ამ შემთხვევაში, თითოეული ყოფილი პირველადი გასაღების ატრიბუტი გარდა PTreatment Date და PTreatment Time ხდება გარე გასაღები PATIENT TREATMENT რელაციაში. სხვა მსგავსი მიდგომაა ე.წ. საწარმოს გასაღების გამოყენება, რომელსაც მოგვიანებით აღვწერთ.

#### ბიჯი 7: სუპერტიპი/ქვეტიპი კავშირის ასახვა

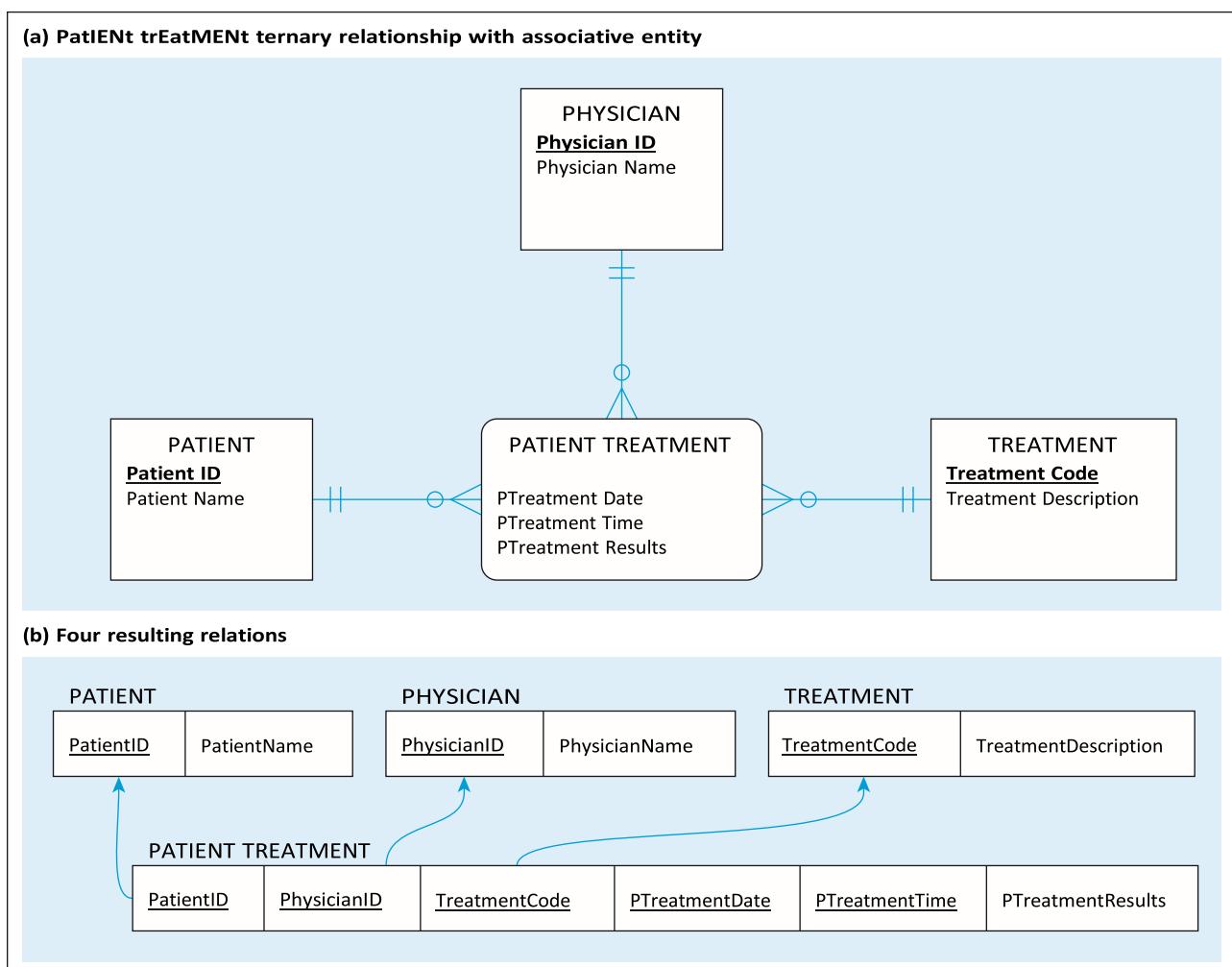
რელაციური მონაცემების მოდელი პირდაპირ არ ახდენს სუპერტიპი/ქვეტიპი კავშირის მხარდაჭერას. საბედნიეროდ, არსებობს სხვადასხვა სტრატეგია, რომელთა საშუალებით

მონაცემთა ბაზის დამპროექტებლებს შეუძლიათ გამოიყენონ ეს კავშირები რელაციური მონაცემების მოდელთან. ყველაზე ხშირად გამოიყენება შემდეგი სტრატეგია:

1. შევქმნათ ცალკე რელაციები სუპერტიპისთვის და მისი თითოეული ქვეტიპისთვის;
2. სუპერტიპისთვის შექმნილ რელაციას მივანიჭოთ ატრიბუტები, რომლებიც საერთოა სუპერტიპის ყველა წევრისთვის, პირველადი გასაღების ჩათვლით;
3. თითოეული ქვეტიპისთვის რელაციას მივანიჭოთ სუპერტიპის პირველადი გასაღები და მხოლოდ ის ატრიბუტები, რომლებიც მხოლოდ ამ ქვეტიპისთვისაა დამახასიათებელი;
4. სუპერტიპის ერთი (ან მეტი) ატრიბუტის მივანიჭოთ ქვეტიპის დისკრიმინატორის ფუნქცია (ქვეტიპის დისკრიმინატორის როლი განხილული გვქონდა ადრე).

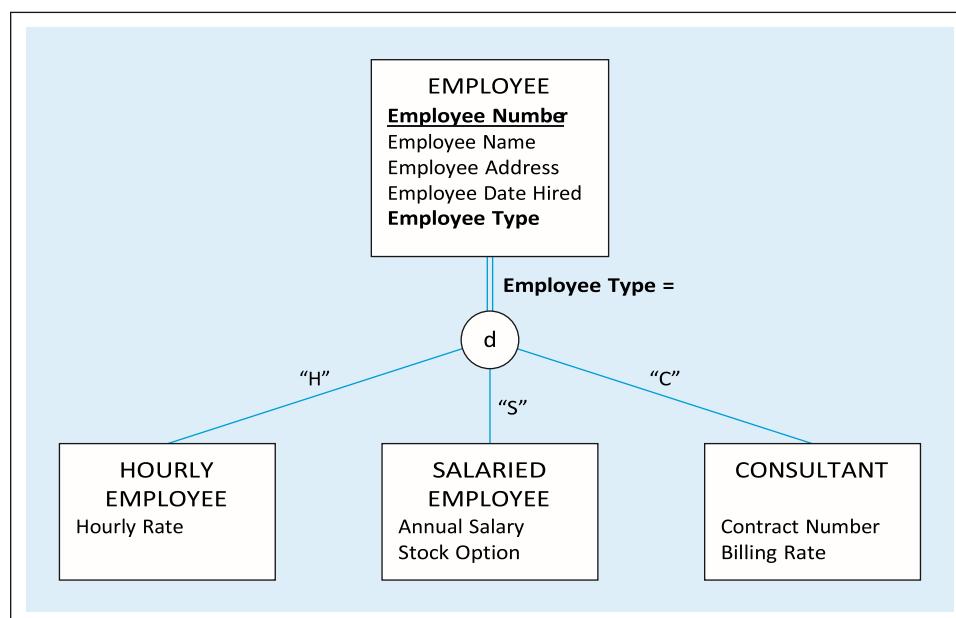
Дру

**figure 4-19** Example of mapping a ternary relationship



ამ პროცედურის გამოყენების მაგალითი ნაჩვენებია 4-20 და 4-21 ნახაზებზე. დიაგრამა 4-20 გვიჩვენებს სუპერტიპს EMPLOYEE ქვეტიპებით HOURLY EMPLOYEE, SALARIED EMPLOYEE და CONSULTANT (ეს მაგალითი აღწერილი იყო ადრე, ხოლო დიაგრამა 4-20 არის ფიგურა 3-8-ის გამეორება). EMPLOYEE-ის პირველადი გასაღებია EmployeeNumber, ხოლო ატრიბუტი Employee Type ქვეტიპის დისკრიმინატორია.

ამ დიაგრამის ამ წესების გამოყენებით რელაციურ სტრუტურად ასახვის შედეგი ნაჩვენებია 4-21 ნახაზზე. გვაქვს სუპერტიპისთვის (EMPLOYEE) ერთი და სამივე ქვეტიპისთვის თითოთითო რელაციის პირველადი გასაღებია EmployeeNumber. პრეფიქსი გამოიყენება თითოეული ქვეტიპის პირველადი გასაღების სახელის გასარჩევად. მაგალითად, SEmployeeNumber არის SALARIED EMPLOYEE რელაციის პირველადი გასაღების სახელი. თითოეული ეს ატრიბუტი არის გარე გასაღები, რომელიც მიულითებს სუპერტიპის პირველად გასაღებაზე, როგორც ეს მითითებულია დიაგრამაზე ისრებით. თითოეული ქვეტიპის რელაცია შეიცავს მხოლოდ იმ ატრიბუტებს, რომლებიც უნიკალურია ქვეტიპისთვის.



**figure 4-20 Supertype/  
subtype relationships**

თითოეული ქვეტიპისთვის შეიძლება შეიქმნას რელაცია, რომელიც შეიცავს ამ ქვეტიპის ყველა ატრიბუტს (როგორც სპეციფიკური, ასევე მემკვიდრეობით მიღებული) SQL

ბრძანების გამოყენებით, რომელიც უერთდება ქვეტიპს თავისი სუპერტიპით. მაგალითად, დავუშვათ, რომ გვინდა ვაჩვენოთ ცხრილი, რომელიც შეიცავს ყველა ატრიბუტს SALARIED EMPLOYEE- სთვის. გამოიყენება შემდეგი ბრძანება:

```
SELECT *
FROM Employee_T, SalariedEmployee_T
WHERE EmployeeNumber = SEmployeeNumber;
```

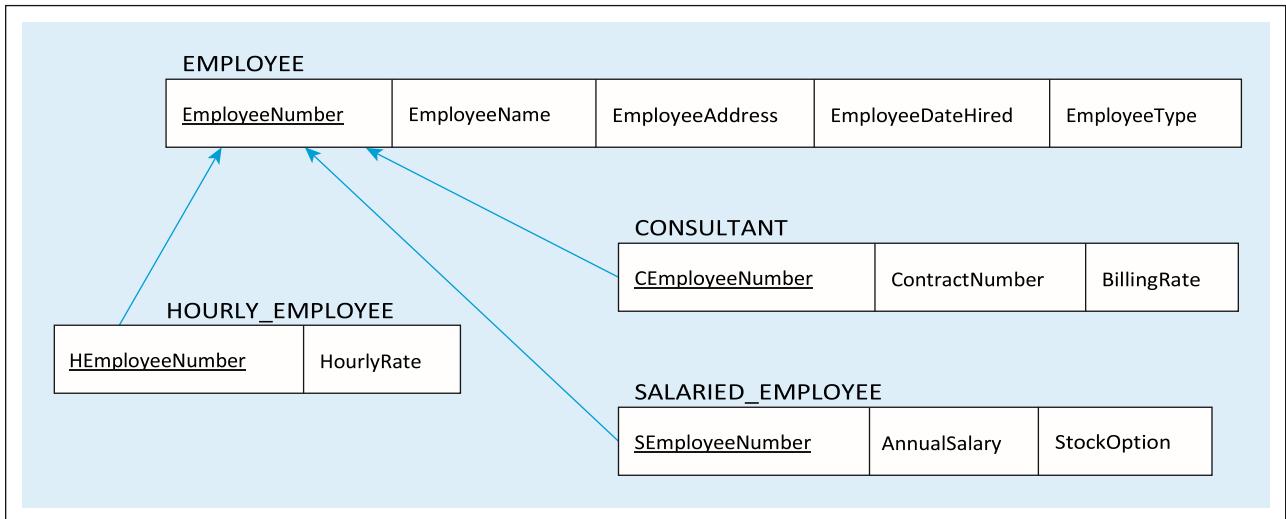


figure 4-21 Mapping supertype/subtype relationships to relations

TABLE 4-2 Summary of EEr-to-relational transformations

EEr Structure	relational representation (Sample Figure)
Regular entity	Create a relation with primary key and nonkey attributes (Figure 4-8)
Composite attribute	Each component of a composite attribute becomes a separate attribute in the target relation (Figure 4-9)
Multivalued attribute	Create a separate relation for multivalued attribute with composite primary key, including the primary key of the entity (Figure 4-10)
Weak entity	Create a relation with a composite primary key (which includes the primary key of the entity on which this entity depends) and nonkey attributes (Figure 4-11)
Binary or unary 1:M relationship	Place the primary key of the entity on the one side of the relationship as a foreign key in the relation for the entity on the many side (Figure 4-12; Figure 4-17 for unary relationship)

Binary or unary $M:N$ relationship or associative entity without its own key	Create a relation with a composite primary key using the primary keys of the related entities plus any nonkey attributes of the relationship or associative entity (Figure 4-13, Figure 4-15 for associative entity, Figure 4-18 for unary relationships)
Binary or unary 1:1 relationship	Place the primary key of either entity in the relation for the other entity; if one side of the relationship is optional, place the foreign key of the entity on the mandatory side in the relation for the entity on the optional side (Figure 4-14)
Binary or unary $M:N$ relationship or associative entity with its own key	Create a relation with the primary key associated with the associative entity plus any nonkey attributes of the associative entity and the primary keys of the related entities as foreign keys (Figure 4-16)
Ternary and $n$ -ary relationships	Same as binary $M:N$ relationships above; without its own key, include as part of primary key of relation for the relationship or associative entity the primary keys from all related entities; with its own surrogate key, the primary keys of the associated entities are included as foreign keys in the relation for the relationship or associative entity (Figure 4-19)
Supertype/subtype relationship	Create a relation for the superclass, which contains the primary and all nonkey attributes in common with all subclasses, plus create a separate relation for each subclass with the same primary key (with the same or local name) but with only the nonkey attributes related to that subclass (Figure 4-20 and 4-21)

EER დიაგრამის რელაციურ მონაცემთა მოდელად გარდაქმნების მოკლე არწერა ბიჯებით მოცემულია ცხრილში 4-2.

## რელაციური მოდელის ნორმალიზაცია

### კარგად სტრუქტურირებული რელაციები

რას წარმოადგენს კარგად სტრუქტურირებული რელაცია? ინტუიციურად, კარგად სტრუქტურირებული რელაცია შეიცავს მინიმალურ სიჭარბეს და მომხმარებლებს საშუალებას აძლევს შეცდომისა და შეუსაბამობის გარეშე ჩასვან, შეცვალონ და წაშალონ ცხრილის სტრიქონები. EMPLOYEE1 (სურათი 4-1) ასეთი რელაციაა. ცხრილის თითოეული სტრიქონი შეიცავს მონაცემებს, რომლებიც აღწერს ერთ თანამშრომელს, ხოლო თანამშრომლის მონაცემებში ნებისმიერი ცვლილება (მაგალითად, ხელფასის ცვლილება) შემოიფარგლება ცხრილის ერთ რიგში განსახორციელები ცვლილებით. ამის საპირისპიროდ, EMPLOYEE2 (სურათი 4-2b) არ არის კარგად სტრუქტურირებული რელაცია. თუ შევისწავლით ცხრილში მოცემულ მონაცემთა ნიმუშს, შევამჩნევთ მნიშვნელოვან სიჭარბეს. მაგალითად, EmpID-ის, Name-ს, DeptName-ს და Salary-ს მნიშვნელობები გამოჩნდება ორ სხვადასხვა სტრიქონში 100, 110 და 150 თანამშრომლებისთვის. შესაბამისად, თუ EmpID- 100-ის შესაბამისი თანამშრომლის ხელფასი შეიცვალა, ეს ფაქტი ორ სტრიქონში უნდა ჩავწეროთ.

EMPLOYEE1			
EmplID	Name	DeptName	Salary
100	Margaret Simpson	Marketing	48,000
140	Allen Beeton	Accounting	52,000
110	Chris Lucero	Info Systems	43,000
190	Lorenzo Davis	Finance	55,000
150	Susan Martin	Marketing	42,000

**FIGURE 4-1 EMPLOYEE1 relation with sample data**

ცხრილში სიჭარბემ შეიძლება გამოიწვიოს შეცდომები ან შეუსაბამობები (ე.წ. ანომალიები), როდესაც მომხმარებელი ცდილობს განახლოს ცხრილში მოცემული მონაცემები. ჩვენ, როგორც წესი, გვაწუხებს სამი სახის ანომალია:

- ჩასმის ანომალია - დავუშვათ, რომ EMPLOYEE2-ს უნდა დავამატოთ ახალი თანამშრომელი. ამ რელაციის პირველადი გასაღები არის EmpID და CourseTitle კომბინაცია. ამიტომ, ახალი სტრიქონის ჩასასმელად, მომხმარებელმა უნდა მიუთითოს როგორც EmpID-ის, ასევე CourseTitle-ის მნიშვნელობები, რადგან პირველადი გასაღების მნიშვნელობები არ შეიძლება იყოს ნულოვანი ან არარსებული. ეს ანომალიაა, რადგან

მომხმარებელს უნდა შეეძლოს შეიტანოს თანამშრომლის მონაცემები, კურსებზე მონაცემების მიწოდების გარეშე.

5. ჩამოყალიბდეთ რომ ცხრილიდან წაიშლება EmpID- 140-ის შესაბამისი პერსონალის მონაცემები. ეს გამოიწვევს ინფორმაციის დაკარგვას იმის შესახებ, რომ ამ თანამშრომელმა დაასრულა კურსი (Tax Acc) 12/8/2015. ფაქტობრივად, ეს კარგავს ინფორმაციას, რომ ამ კურსს ჰქონდა შეთავაზება, რომელიც დასრულდა ამ დღეს.
  
  
  
  
  
6. მოდიფიკირეთ ანომალია - დავუშვათ, რომ EmpID- 100-ის შესაბამის თანამშრომელს გაეზარდა ხელფასი. ჩვენ უნდა დავაფიქსიროთ ეს ზრდა თითოეული ამ თანამშრომლის შესაბამის ყოველ სტრიქონში (ორი შემთხვევა ნახაზზე 4-2); წინააღმდეგ შემთხვევაში, მონაცემები არათანმიმდევრული იქნება.

**(b) EMPLOYEE2 relation**

EMPLOYEE2

EmplID	Name	DeptName	Salary	CourseTitle	DateCompleted
100	Margaret Simpson	Marketing	48,000	SPSS	6/19/2015
100	Margaret Simpson	Marketing	48,000	Surveys	10/7/2015
140	Alan Beeton	Accounting	52,000	Tax Acc	12/8/2015
110	Chris Lucero	Info Systems	43,000	Visual Basic	1/12/2015
110	Chris Lucero	Info Systems	43,000	C++	4/22/2015
190	Lorenzo Davis	Finance	55,000		
150	Susan Martin	Marketing	42,000	SPSS	6/19/2015
150	Susan Martin	Marketing	42,000	Java	8/12/2015

ეს ანომალიები მიუთითებს, რომ EMPLOYEE2 არ არის კარგად სტრუქტურირებული რელაცია. ამ რელაცის პრობლემა ისაა, რომ იგი შეიცავს მონაცემებს ორი არსის შესახებ: EMPLOYEE და COURSE. ჩვენ გამოვიყენეთ ნორმალიზაციის თეორიას EMPLOYEE2-ის თუ რელაციად დაყოფისთვის. შედეგად მიღებული ერთ-ერთი რელაციაა EMPLOYEE1 (სურათი 4-1). მეორეს EMP COURSE- ს დავარქმევთ, რომელიც მოცემულია მონაცემების ნიმუშით 4-7 ნახაზზე. ამ რელაციის პირველადი გასაღები არის EmplID-ისა და CourseTitle-ის კომბინაცია და ამ ატრიბუტების სახელებს ხაზს ვუსვამთ დიაგრამა 4-7-ზე, ამ ფაქტის გასაზრდელად. დიაგრამა 4-7-ზე გამოსახული EMP COURSE არ შეიცავს წინ აღწერილი ანომალიების ტიპებს და, შესაბამისად, კარგად არის სტრუქტურირებული.

**FIGURE 4-7 EMP COURSE**

EmplD	CourseTitle	DateCompleted
100	SPSS	6/19/2015
100	Surveys	10/7/2015
140	Tax Acc	12/8/2015
110	Visual Basic	1/12/2015
110	C++	4/22/2015
150	SPSS	6/19/2015
150	Java	8/12/2015

[https://us02web.zoom.us/rec/share/4fz9CCtvgmMYBm7w0\\_vR7-MX9iYVgYN4qe8U4nmMJliPv4tibbCWSAxduuNPiPJ.\\_r6deEeVdrkCrCl5?startTime=1617699864000](https://us02web.zoom.us/rec/share/4fz9CCtvgmMYBm7w0_vR7-MX9iYVgYN4qe8U4nmMJliPv4tibbCWSAxduuNPiPJ._r6deEeVdrkCrCl5?startTime=1617699864000)

### ნორმალიზაცია

EER დიაგრამების რელაციურ მოდელად ადრე აღწერილი ბიჯების შესაბამისად გარდაქმნას კარგად სტრუქტურირებულ რელაციებთან მივყავართ. ამასთან, არ არსებობს გარანტია, რომ ყველა ანომალია მოიხსნება ამ ბიჯების დაცვით. ნორმალიზება არის ფორმალური პროცესი იმის დასადგენად, რომელი ატრიბუტები უნდა დაჯგუფდეს რელაციებში ისე, რომ ყველა ანომალია მოიხსნას. მაგალითად, ჩვენ გამოვიყენეთ ნორმალიზაციის პრინციპები EMPLOYEE2 ცხრილის (მისი ზედმეტობის) EMPLOYEE1 (ნახაზი 4-1) და EMP COURSE (ნახაზი 4-7) გადასაკეთებლად. მონაცემთა ბაზის შემუნავების მთელი პროცესის დროს ძირითადი ორი შემთხვევა არსებობს, როდესაც ჩვეულებრივ შესაძლებელია სარგებლის მიღება ნორმალიზაციის გამოყენებით:

1. მონაცემთა ბაზის ლოგიკური დაპროექტებისას - უნდა გამოვიყენოთ ნორმალიზაციის ცნებები იმ კავშირების ხარისხის შესამოწმებლად, რომლებიც მიღებულია E-R დიაგრამებიდან;
2. ძველი სისტემების უკუდაპროექტებისას - უკუდაპროექტებისას ძველი სისტემების მრავალი ცხრილი და მომხმარებლის ხედები ჭარბია და მოიცავენ ანომალიებს.

ჯერჯერობით ჩვენ წარმოვადგინეთ კარგად სტრუქტურირებული რელაციების ინტუიციური განხილვა; ამასთან, ჩვენ გვჭირდება ასეთი რელაციების ფორმალური განმარტებები, მათი შემუშავების პროცესთან ერთად. ნორმალიზაცია არის ანომალიების შემცველი რელაციების თანმიმდევრული შემცირების პროცესი, უფრო მცირე ზომის, კარგად სტრუქტურირებული რელაციების წარმოსაქმნელად. ნორმალიზაციის რამდენიმე ძირითადი მიზანია:

1. მინიმუმადე შემცირდეს მონაცემთა სიჭარბე, რითაც თავიდან იქნას აცილებული ანომალიები და შენარჩუნდეს შენახვის ადგილი.

2. რეფერენტული მთლიანობის (დამოწმებითი ერთიანობა) შეზღუდვების გამოყენების გამარტივება.
3. გამარტივდეს მონაცემთა მხარდაჭერა (ჩასმა, განახლება და წაშლა).
4. მოხდეს უკეთესი დიზაინის მხარდაჭერა, რომელიც წარმოადგენს რეალური სამყაროს უკეთეს ასახვას და უფრო მყარ საფუძველს სამომავლო გაფართოებისათვის.

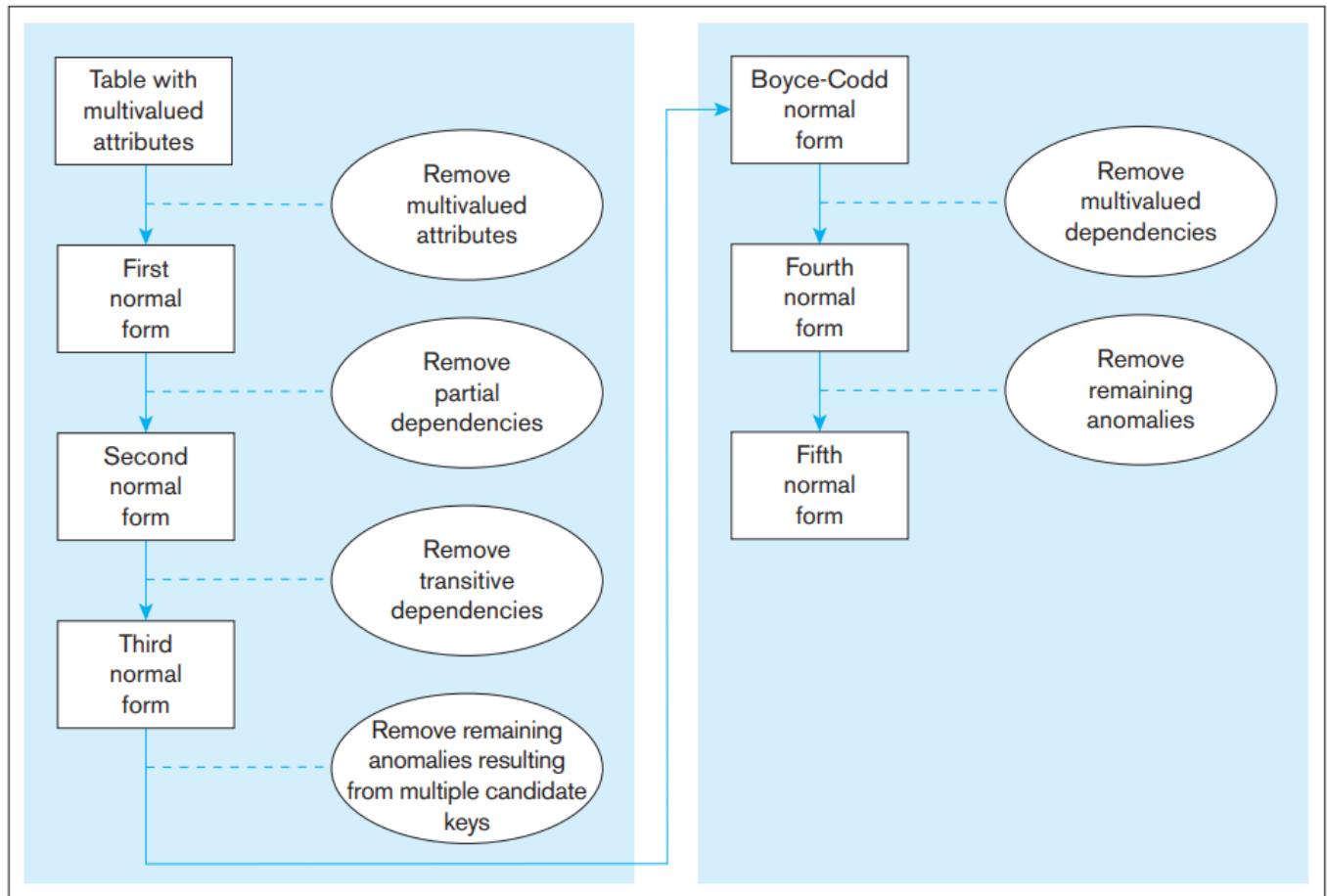
ნორმალიზება არ იძლევა ვარაუდს იმის შესახებ, თუ როგორ იქნება გამოყენებული მონაცემები დისპლეიზე, მოთხოვნებსა თუ ანგარიშებში. ნორმალიზაცია, იმის საფუძველზე, რასაც ჩვენ ნორმალურ ფორმებს და ფუნქციურ დამოკიდებულებებს დავარქმევთ, განსაზღვრავს ბიზნესის წესებს და არა მონაცემთა გამოყენებას. გარდა ამისა, მონაცემთა ნორმალიზება ხდება მონაცემთა ბაზის ლოგიკური დაპროექტების დასრულებისთანავე. ამრიგად, ნორმალიზება არანაირ შეზღუდვას არ ადებს იმაზე, თუ ფიზიკურად როგორ შეიძლება ან უნდა იყოს შენახული მონაცემები ან, შესაბამისად, ეფექტურად დამუშავებული. ნორმალიზება არის ლოგიკური მონაცემების მოდელირება, რომელიც გამოიყენება იმის უზრუნველსაყოფად, რომ მონაცემები კარგად იყოს სტრუქტურირებული ორგანიზაციის მასშტაბით.

### ბიჯები ნორმალიზაციაში

ნორმალიზაციის განხორციელება შესაძლებელია ეტაპობრივად, რომელთაგან თითოეული შეესაბამება ნორმალურ ფორმას (იხ. სურათი 4-22). ნორმალური ფორმა არის რელაციების მდგომარეობა, რომელიც მოითხოვს ატრიბუტებს (ან ფუნქციურ დამოკიდებულებებს) შორის ურთიერთდამოკიდებულებასთან დაკავშირებით გარკვეული წესების დაცვას. ეს ნორმალური ფორმებია:

1. პირველი ნორმალური ფორმა - ამოღებულია ყველა მრავალმნიშვნელოვანი ატრიბუტი (რომელსაც განმეორებად ჯგუფებს უწოდებენ), ამიტომ ცხრილის თითოეული სტრიქონისა და სვეტის გადაკვეთაზე არის ერთი მნიშვნელობა (შესაძლოა null) (როგორც სურათი 4-2 ბ).
2. მეორე ნორმალური ფორმა - ამოღებულია ნებისმიერი ნაწილობრივი ფუნქციური დამოკიდებულება (ანუ, არაგასაღები ატრიბუტების იდენტიფიცირება ხდება მთლიანი პირველად გასაღებით).
3. მესამე ნორმალური ფორმა - ყველა ტრანზიტული დამოკიდებულება ამოღებულია (მაგ., არაგასაღები ატრიბუტების იდენტიფიცირება ხდება მხოლოდ პირველადი გასაღებით).
4. მოის-კოდის ნორმალური ფორმა - ყველა დარჩენილი ანომალია, რომელიც ფუნქციონალური დამოკიდებულების შედეგად წარმოიშვა, ამოღებულია (რადგან ერთიდამავე არაგასაღები ატრიბუტისთვის არსებობდა ერთზე მეტი შესაძლო პირველადი გასაღები).
5. მეოთხე ნორმალური ფორმა - ნებისმიერი მრავალჯერადი დამოკიდებულება ამოღებულია.
6. მეხუთე ნორმალური ფორმა - ამოღებულია დარჩენილი ანომალიები.

**FIGURE 4-22** Steps in normalization



## ფუნქციური დამოკიდებულებები და გასაღებები

ბოის-კოდის ნორმალურ ფორმამდე ნორმალიზება ემყარება ფუნქციური დამოკიდებულებების ანალიზს. ფუნქციონალური დამოკიდებულება არის შეზღუდვა ორ ატრიბუტს ან ატრიბუტის ორ ნაკრებს შორის. ნებისმიერი R დამოკიდებულებისათვის (რელაციისსათვის), ატრიბუტი B ფუნქციურად არის დამოკიდებული A ატრიბუტზე, თუ A-ს ყველა მოქმედი ეგზემპლარისათვის, A-ს მნიშვნელობა ცალსახად განსაზღვრავს B-ს მნიშვნელობას. B-ს ფუნქციონალური დამოკიდებულება A-ზე გამოსახულია ისრით, შემდეგშნაირად:  $A \rightarrow B$ . ფუნქციონალური დამოკიდებულება არ არის მათემატიკური დამოკიდებულება: B არ შეიძლება გამოითვალოს A-დან, უფრო მეტიც, თუ A-ს მნიშვნელობა ცნობილია, შეიძლება არსებობდეს მხოლოდ ერთი მნიშვნელობა B. ასევე ატრიბუტი შეიძლება ფუნქციურად იყოს დამოკიდებული ორი (ან მეტი) ატრიბუტის კომბინაციაზე. მაგალითად, განვიხილოთ დამოკიდებულება EMP COURSE (EmpID, CourseTitle, DateCompleted) ნაჩვენებია დიაგრამა 4-7. ამ რელაციაში ფუნქციურ დამოკიდებულებას წარმოვადგენთ შემდეგნაირად:

მძიმით EmpID- სა და CourseTitle- ს აღინიშნება ლოგიკური „და“ ოპერატორი, რადგან DateCompleted ფუნქციურად დამოკიდებულია EmpID-ზე და CourseTitle-ზე.

ამ განაცხადში ფუნქციონალური დამოკიდებულება გულისხმობს, რომ კურსის დასრულების თარიღი განისაზღვრება დასაქმებულის ვინაობით და კურსის სახელწოდებით. ფუნქციური დამოკიდებულების ტიპიური მაგალითებია შემდეგი:

1. **SSN → Name, Address, Birthdate** (SSN → სახელი, მისამართი, დაბადების თარიღი) - პირის სახელი, მისამართი და დაბადების თარიღი ფუნქციურად არის დამოკიდებული ამ პირის პირად ნომერზე (სხვა სიტყვებით რომ ვთქვათ, თითოეული SSN- სთვის შეიძლება იყოს მხოლოდ ერთი სახელი, ერთი მისამართი და ერთი დაბადების დღე).
2. **VIN → Make, Model, Color** (VIN → მარკა, მოდელი, ფერი) - ავტომობილის მარკა, მოდელი და ორიგინალი ფერი ფუნქციურად დამოკიდებულია ავტომობილის საიდენტიფიკაციო ნომერზე (როგორც ზემოთ, თითოეულ VIN – სთან შეიძლება იყოს მხოლოდ ერთი მნიშვნელობის მარკა, მოდელი და ფერი).
3. **ISBN → Title, FirstAuthorName, Publisher** (ISBN → სათაური, პირველი ავტორის სახელი, გამომცემელი) - წიგნის სათაური, პირველი ავტორის სახელი და გამომცემელი ფუნქციურად დამოკიდებულია წიგნის საერთაშორისო სტანდარტული წიგნის ნომერზე (ISBN).

### დეტერმინანტი

ატრიბუტს ისრის მარცხენა მხარეს ფუნქციურ დამოკიდებულებაში განმსაზღვრელს ანუ დეტერმინატორს უწოდებენ. წინა სამ მაგალითში SSN, VIN და ISBN დეტერმინატორია. EMP COURSE რელაციაში (სურათი 4-7), დეტერმინატორია EmpID და CourseTitle.

### კანდიდატი გასაღები

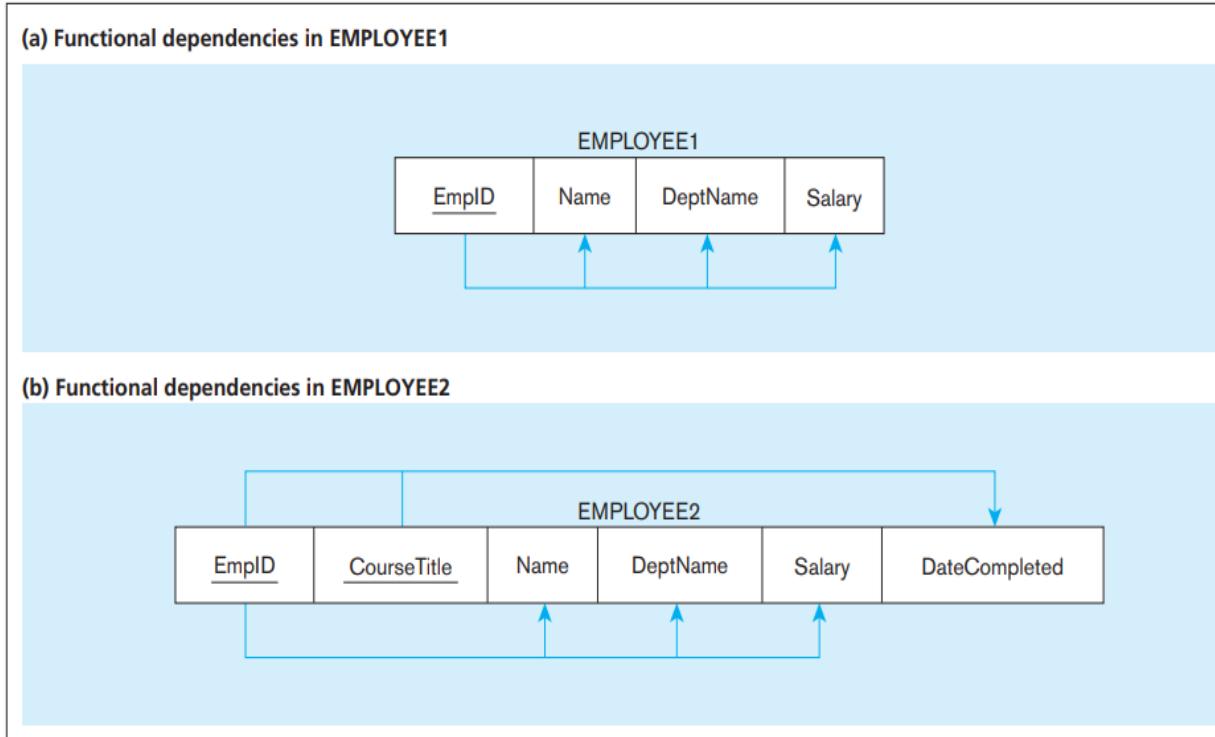
კანდიდატი გასაღები არის ატრიბუტი, ან ატრიბუტების კომბინაცია, რომელიც ცალსახად განსაზღვრავს სტრიქონს რელაციაში. კანდიდატი გასაღები უნდა აკმაყოფილებდეს შემდეგ თვისებებსს, რომლებიც ადრე განხილული ექვსი თვისების ქვესიმრავლეს წარმოადგენს:

1. უნიკალური იდენტიფიკაცია - თითოეული სტრიქონისათვის გასაღების მნიშვნელობამ ცალსახად უნდა განსაზღვროს ეს მწკრივი. ეს თვისება გულისხმობს, რომ თითოეული არაგასარები ატრიბუტი ფუნქციურად დამოკიდებულია ამ გასაღებაზე.
2. არასიჭარბე - გასაღების არცერთი ატრიბუტის წაშლა არაა შესაძლებელი უნიკალური იდენტიფიკაციის თვისების დარღვევის გარეშე.

გამოვიყენოთ წინა განმარტება წინ აღწერილ ორ რელაციაში კანდიდატი გასაღებების დასადგენად. EMPLOYEE1 რელაციას (სურათი 4-1) აქვს შემდეგი სქემა: EMPLOYEE1(EmpID, Name, DeptName, Salary). EmpID ერთადერთი დეტერმინატორია ამ მიმართებაში. ყველა სხვა ატრიბუტი ფუნქციურად დამოკიდებულია EmpID-ზე. ამიტომ, EmpID არის კანდიდატი

გასაღები და (რადგან სხვა კანდიდატი გასაღებები არ არსებობს) ასევე არის ძირითადი გასაღები.

წარმოვადგინოთ რელაციის ფუნქციური დამოკიდებულებები 4-23 ნახაზზე ნაჩვენები აღნიშვნების გამოყენებით. დიაგრამა 4-23a გვიჩვენებს EMPLOYEE1-ის ასახვას. ფიგურაში ჰამოდის პირველადი გასაღებიდან (EmpID) და უერთდება ამ ხაზს. ვერტიკალური ისრები შემდეგში მიუთითებს თითოეულ არაგასაღებ ატრიბუტზე, რომლებიც ფუნქციურად დამოკიდებულია პირველად გასაღებზე.



**FIGURE 4-23** Representing functional dependencies

რელაციისათვის EMPLOYEE2 (სურათი 4-2b), გასათვალისწინებელია, რომ (განსხვავებით EMPLOYEE1) EmpID ცალსახად არ განსაზღვრავს სტრიქონს ამ რელაციაში. მაგალითად, EmpID-ის ნომერზე ცხრილში არის ორი სტრიქონი. ამ რელაციაში არსებობს ორი ტიპის ფუნქციური დამოკიდებულება:

1. EmpID → Name, DeptName, Salary
2. EmpID, CourseTitle → DateCompleted

ფუნქციონალური დამოკიდებულებები მიუთითებს, რომ EmpID და CourseTitle კომბინაცია არის ერთადერთი კანდიდატი გასარები (და, შესაბამისად, ძირითადი გასაღები) EMPLOYEE2-ისთვის. სხვა სიტყვებით რომ ვთქვათ, EMPLOYEE2- ის პირველადი გასაღები არის კომპოზიტიური გასაღები. არც EmpID და არც CourseTitle არ განსაზღვრავს სტრიქონს ამ რელაციაში და, შესაბამისად, თავისთავად არ შეიძლება იყოს კანდიდატი გასაღები. თუ გადავხედავთ მონაცემებს ნახაზზე 4-2 ბ, დავრწმუნდებით, რომ EmpID- ისა და CourseTitle- ის კომბინაცია ცალსახად განსაზღვრავს EMPLOYEE2- ის თითოეულ სტრიქონს. ჩვენ

წარმოვადგენთ ამ რელაციაში ფუნქციონალურ დამოკიდებულებებს ნახაზზე 4-23b. გაითვალისწინეთ, რომ DateCompleted ერთადერთი ატრიბუტია, რომელიც ფუნქციურად დამოკიდებულია სრულ პირველად გასაღებზე, რომელიც შედგება ატრიბუტებისაგან EmpID და CourseTitle.

დეტერმინანტებსა და კანდიდატ გასაღებებს შორის ურთიერთდამოკიდებულების შეჯამება შეგვიძლია შემდეგნაირად: კანდიდატი გასაღები ყოველთვის არის დეტერმინანტი, ხოლო დეტერმინანტი შეიძლება იყოს ან არ იყოს კანდიდატი გასაღები. მაგალითად, EMPLOYEE2-ში, EmpID არის დეტერმინანტი, მაგრამ არა კანდიდატი გასაღები. კანდიდატი გასაღები არის დეტერმინანტი, რომელიც ცალსახად განსაზღვრავს დანარჩენ (არაგასაღებ) ატრიბუტებს რელაციაში. დეტერმინანტი შეიძლება იყოს კანდიდატი გასაღები (მაგალითად, EmpID in EMPLOYEE1), კომპოზიტიური კანდიდატი გასაღების ნაწილი (მაგალითად, EmpID in EMPLOYEE2), ან არაგასაღები ატრიბუტი.

### ნორმალიზაციის მაგალითი

ფუნქციური დამოკიდებულებების და გასაღებების შესწავლის შემდეგ, შესაძლებელია აღვწეროთ და ავსახოთ ნორმალიზაციის ბიჯები. თუ EER მონაცემთა მოდელის ტრანსფორმაცია მოხდა მონაცემთა ბაზის რელაციების ყოვლისმომცველ ნაკრებად, საჭიროა თითოეული მათგანის ნორმალიზება. სხვა შემთხვევებში, როდესაც ლოგიკური მონაცემების მოდელი იქმნება მომხმარებლის ინტერფეისებიდან, როგორიცაა ეკრანული ასახვები, ფორმები და ანგარიშები, შესაძლოა საჭირო გახდეს რელაციური სტრუქტურის შექმნა თითოეული სამომხმარებლო ინტერფეისისათვის და შემდეგომში მოხდეს ამ რელაციების ნორმალიზება.

ილუსტრაციისთვის ვიყენებთ მომხმარებელთა ზედნადებს Pine Valley ავეჯის კომპანიისგან (იხ. სურათი 4-24.)

### ბიჯი 0: წარმოადგინეთ ხედი ცხრილის სახით

პირველი ბიჯი (ნორმალიზაციის წინმსწრები) წარმოადგენს მომხმარებლის ხედვის (ამ შემთხვევაში, ინვოისის) წარმოდგენას ერთი ცხრილის ან ატრიბუტების (სვეტის სათაურების სახით) შემცველი რელაციის სახით. ნიმუშის მონაცემები უნდა ჩაიწეროს ცხრილის სტრიქონებში, მათ შორის ნებისმიერი განმეორებითი ჯგუფი, რომლებიც მოცემულია მონაცემებში. ინვოისის ცხრილი ნაჩვენებია ნახაზზე 4-25. გავითვალისწინოთ, რომ მეორე შეკვეთის მონაცემები (OrderID 1007) მოცემულია ნახაზზე 4-25 ამ მონაცემების სტრუქტურის უფრო მეტად განსამარტად.

**FIGURE 4-24** Invoice (Pine Valley Furniture Company)

PVFC Customer Invoice					
Customer ID	2	Order ID	1006		
Customer Name	Value Furniture	Order Date	10/24/2015 <th data-cs="2" data-kind="parent"></th> <th data-kind="ghost"></th>		
Address	15145 S.W. 17th St. Plano TX 75022				
Product ID	Product Description	Finish	Quantity	Unit Price	Extended Price
7	Dining Table	Natural Ash	2	\$800.00	\$1,600.00
5	Writer's Desk	Cherry	2	\$325.00	\$650.00
4	Entertainment Center	Natural Maple	1	\$650.00	\$650.00
				Total	\$2,900.00

#### ბიჯი 1: პირველ ნორმალურ ფორმად გარდაქმნა

რელაცია პირველ ნორმალურ ფორმაშია (1nF), თუ გამოიყენება ორივე შემდეგი შეზღუდვა:

1. რელაციაში არ არის განმეორებითი ჯგუფები (ამრიგად, ცხრილი თითოეული სტრიქონისა და სვეტის გადაკვეთაზე არის ერთი ფაქტი).
2. განსაზღვრულია პირველადი გასაღები, რომელიც ცალსახად განსაზღვრავს თითოეულ სტრიქონს რელაციაში.

#### განმეორებადი ჯგუფების ამოღება

როგორც ვხედავთ, ინვოისის მონაცემები 4-25 ნახაზზე შეიცავს განმეორებად ჯგუფს თითოეული პროდუქტისთვის, რომელიც მოცემულია კონკრეტული შეკვეთის მიხედვით. ამრიგად, OrderID 1006 შეიცავს სამ განმეორებით ჯგუფს, რომლებიც შეესაბამება ამ შეკვეთის სამ პროდუქტს.

ზემოთ ვაჩვენეთ, თუ როგორ უნდა ამოვიღოთ განმეორებითი ჯგუფები ცხრილიდან, მონაცემთა შესაბამისი მნიშვნელობების შევსებით ცხრილის ადრე არსებულ ცარიელ უჯრედებში (იხ. ნახაზები 4-2 ა და 4-2 ბ). ინვოისის ცხრილისთვის ამ პროცედურის გამოყენება იძლევა ახალ რელაციას (სახელად INVOICE), რომელიც ნაჩვენებია 4-26 ნახაზზე.

#### პირველადი გასაღების შერჩევა

INVOICE -ში ოთხი დეტერმინატორია და მათი ფუნქციური დამოკიდებულება შემდეგია:

OrderID → OrderDate, CustomerID, CustomerName, CustomerAddress

CustomerID → CustomerName, CustomerAddress

ProductID → ProductDescription, ProductFinish,

ProductStandardPrice      OrderID,      ProductID      →

OrderedQuantity

ამ შემთხვევაში ინფორმაცია ფუნქციური დამოკიდებულებების შესახებ ბიზნესის წესების შესაბამისად ორგანიზაციიდან მოდის (Pine Valley ავეჯის კომპანიის ბიზნესის ბუნების შესწავლის შედეგად). ასევე შეგვიძლია ვნახოთ, რომ ნახაზზე 4-26 მოცემული მონაცემები არ არღვეს რომელიმე ამ ფუნქციურ დამოკიდებულებას. მაგრამ, რადგან ამ ცხრილის ყველა შესაძლო სტრიქონს ვერ ვხედავთ, ვერ ვიქნებით დარწმუნებული, რომ არ იქნებოდა ისეთი ინვოისი, რომელიც დაარღვევდა ერთ-ერთ ამ ფუნქციურ დამოკიდებულებას. ამრიგად, დამოკიდებული უნდა ვიყოთ ორგანიზაციის წესების ჩვენს გაეხაზე.

**FIGURE 4-25 INVOICE data (Pine Valley Furniture Company)**

OrderID	Order Date	Customer ID	Customer Name	Customer Address	ProductID	Product Description	Product Finish	Product StandardPrice	Ordered Quantity
1006	10/24/2015	2	Value Furniture	Plano, TX	7	Dining Table	Natural Ash	800.00	2
					5	Writer's Desk	Cherry	325.00	2
					4	Entertainment Center	Natural Maple	650.00	1
1007	10/25/2015	6	Furniture Gallery	Boulder, CO	11	4-Dr Dresser	Oak	500.00	4
					4	Entertainment Center	Natural Maple	650.00	3

როგორც ხედავთ, INVOICE- ის ერთადერთი კანდიდატი გასაღები არის კომპოზიტური გასაღები, რომელიც შედგება ატრიბუტების OrderID და ProductID (რადგან ამ ატრიბუტების მნიშვნელობების ნებისმიერი კომბინაციის ცხრილში მხოლოდ ერთი სტრიქონია). შესაბამისად, OrderID და ProductID ხაზგასმულია ნახაზზე 4-26, რაც მიუთითებს იმაზე, რომ ისინი ქმნიან პირველად გასაღებს.

პირველადი გასაღების ფორმირებისას ფრთხილად უნდა ვიყოთ, რომ არ შევიტანოთ ზედმეტი (და, შესაბამისად, არასაჭირო) ატრიბუტები. ამრიგად, მიუხედავად იმისა, რომ CustomerID არის INVOICE-ისიდენტიფიკატორი, იგი არ შედის პირველადი გასაღების ნაწილად, რადგან გასაღების ყველა ატრიბუტი იდენტიფიცირებულია OrderID და ProductID კომბინაციით.

დიაგრამა, რომელიც აჩვენებს ამ ფუნქციურ დამოკიდებულებებს INVOICE რელაციისთვის ნაჩვენებია ნახაზზე 4-27. ეს დიაგრამა წარმოადგენს INVOICE- ის ყველა ატრიბუტის პორიზონტალურ ჩამონათვალს, პირველადი გასაღები ატრიბუტების ხაზგასმით (OrderID და ProductID). გავითვალისწინოთ, რომ ერთადერთი ატრიბუტი, რომელიც დამოკიდებულია სრულ გასაღებზე, არის OrderedQuantity. ყველა სხვა ფუნქციური დამოკიდებულება ან ნაწილობრივია, ან ტრანზიტული დამოკიდებულება (ორივე განისაზღვრება შემდეგში).

### ანომალიები 1nf-ში

მიუხედავად იმისა, რომ განმეორებითი ჯგუფები ამოღებულია, 4-26 ნახაზის მონაცემები კვლავ შეიცავს მნიშვნელოვან სიჭარბეს. მაგალითად, CustomerID, CustomerName და CustomerAddress Value ავეჯისთვის ჩაიწერება ცხრილში სამ სტრიქონში (მინიმუმ). ამ სიჭარბის შედეგად, ცხრილში მოცემული მონაცემებით მანიპულირებამ შეიძლება გამოიწვიოს ისეთი ანომალიები, როგორიცაა შემდეგი:

- ჩასმის ანომალია -** ამ ცხრილის სტრუქტურით კომპანიას არ შეუძლია წარუდგინოს ახალი პროდუქტი (ვთქვათ, საუზმის მაგიდა ProductID 8-ით) და დაამატოს იგი მონაცემთა ბაზაში, სანამ არ იქნება პირველი შეკვეთა: ცხრილში ჩანაწერების დამატება არ შეიძლება ProductID-ისა და OrderID-ის გარეშე. კიდევ ერთი მაგალითი, თუ მომხმარებელი დარეკავს და ითხოვს მის OrderID 1007-ს სხვა პროდუქტის დამატებას, უნდა ჩაისვას ახალი სტრიქონი, რომელშიც უნდა განმეორდეს შეკვეთის თარიღი და მომხმარებლის ყველა ინფორმაცია. ეს იწვევს მონაცემთა ტირაჟირებას და მონაცემთა შეყვანის პოტენციურ შეცდომებს (მაგალითად, მომხმარებლის სახელი შეიძლება შეიტანონ შეცდომით).
- წაშლის ანომალია -** თუ მომხმარებელი დარეკავს და ითხოვს სასადილო მაგიდის წაშლას მისი OrderID 1006-დან, ეს სტრიქონი უნდა წაიშალოს რელაციიდან და ჩვენ ვკარგავთ ინფორმაციას ამ ნივთის მოპირკეთების (Natural Ash) და ფასის (800.00 აშშ დოლარი) შესახებ.
- განახლების ანომალია -** თუ Pine Valley Furniture (როგორც ფასის კორექტირების ნაწილი) ზრდის ერთერთი პროდუქტის ფასს (ProductID 4) ფასს 750,00 დოლარამდე, ეს ცვლილება უნდა ჩაიწეროს ამ ნივთის შემცველ ყველა სტრიქონში. (სურათი 4-26-ში ორი ასეთი რიგია.)

**FIGURE 4-26 INVOICE relation (1NF) (Pine Valley Furniture Company)**

OrderID	Order Date	Customer ID	Customer Name	Customer Address	ProductID	Product Description	Product Finish	Product StandardPrice	Ordered Quantity
1006	10/24/2015	2	Value Furniture	Plano, TX	7	Dining Table	Natural Ash	800.00	2
1006	10/24/2015	2	Value Furniture	Plano, TX	5	Writer's Desk	Cherry	325.00	2
1006	10/24/2015	2	Value Furniture	Plano, TX	4	Entertainment Center	Natural Maple	650.00	1
1007	10/25/2015	6	Furniture Gallery	Boulder, CO	11	4-Dr Dresser	Oak	500.00	4
1007	10/25/2015	6	Furniture Gallery	Boulder, CO	4	Entertainment Center	Natural Maple	650.00	3

## ბიჯი 2: მეორე ნორმალურ ფორმაში გარდაქმნა

INVOICE რელაციაში ზედმეტი სიჭარბის (და შედეგად მიღებული ანომალიები) მოშორება შესაძლებელია მეორე ნორმალურ ფორმაში (2nF), თუ ის პირველ ნორმალურ ფორმაშია და არ შეიცავს ნაწილობრივ ფუნქციურ დამოკიდებულებებს. ნაწილობრივი ფუნქციონალური დამოკიდებულება არსებობს, როდესაც არაგასაღები ატრიბუტი ფუნქციურად დამოკიდებულია პირველადი გასაღების ნაწილზე (მაგრამ არა მთლიანზე). როგორც ვხედავთ, ნახაზზე 4-27-ზე არსებობს შემდეგი ნაწილობრივი დამოკიდებულებები:

$\text{OrderID} \rightarrow \text{OrderDate, CustomerID, CustomerName, CustomerAddress}$

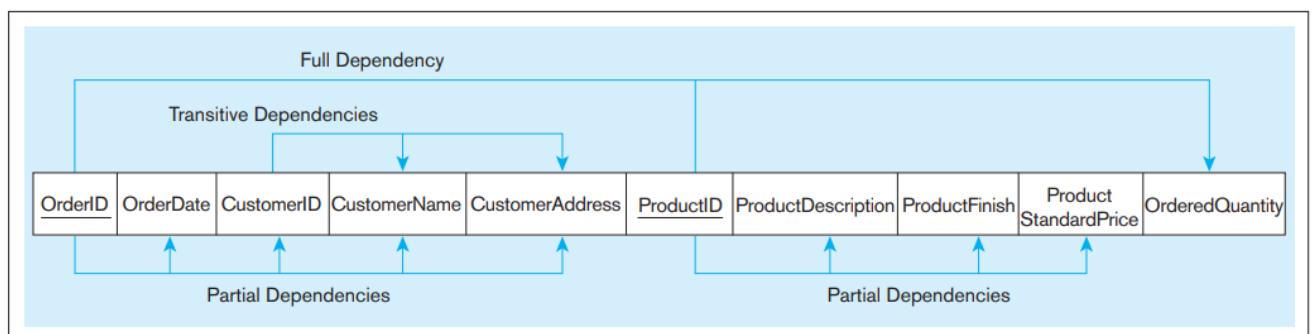
$\text{ProductID} \rightarrow \text{ProductDescription, ProductFinish, ProductStandardPrice}$

ამ ნაწილობრივი დამოკიდებულებების პირველ ნაწილში (მაგალითად) ნათქვამია, რომ შეკვეთის თარიღი ცალსახად განისაზღვრება შეკვეთის ნომრით და მას არანაირი კავშირი არ აქვს  $\text{ProductID}$ - სთან.

ნაწილობრივი დამოკიდებულებების მქონე რელაციის მეორე ნორმალურ ფორმაში გადასაყვანად საჭიროა შემდეგი ბიჯები:

1. შეიქმნას ახალი რელაციები პირველადი გასაღების თითოეული ატრიბუტისთვის (ან ატრიბუტების კომბინაციისთვის), რომელიც განსაზღვრავს ნაწილობრივ დამოკიდებულებას. ეს ატრიბუტი იქნებიან პირველადი გასაღები ახალ რელაციაში.
2. გადაადგილდეს არაგასაღები ატრიბუტები, რომლებიც მხოლოდ ამ პირველად ატრიბუტზეა (ატრიბუტებზე) დამოკიდებული ძველი რელაციიდან ახალ რელაციაში.

ამ ბიჯების შესრულების შედეგები ინვიცი რელაცია ნაჩვენებია სურათი 4-28. ნაწილობრივი დამოკიდებულების მოხსნა იწვევს ორი ახალი რელაციის შექმნას: PRODUCT და CUSTOMER ORDER. INVOICE რელაციაში ახლა დარჩა მხოლოდ პირველადი გასაღებების ატრიბუტები ( $\text{OrderID}$  და  $\text{ProductID}$ ) და  $\text{OrteredQuantity}$ , რაც ფუნქციურად დამოკიდებულია მთლიან გასაღებზე. ამ ურთიერთობას დავარქვათ ORDER LINE, რადგან ამ ცხრილის თითოეული სტრიქონი წარმოადგენს შეკვეთის ერთ სტრიქონს.

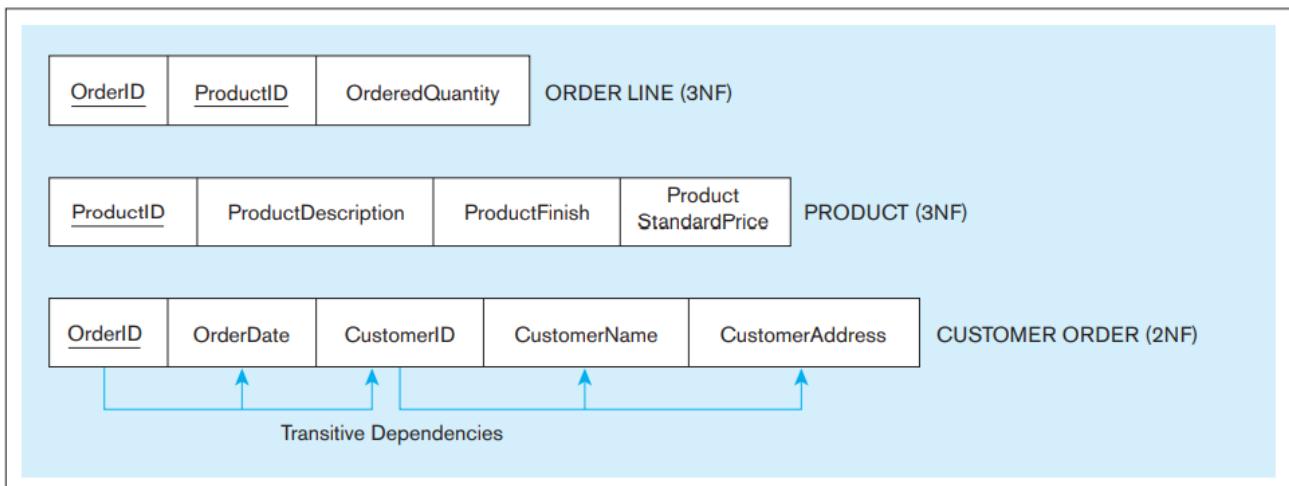


**FIGURE 4-27** Functional dependency diagram for INVOICE

როგორც 4-28 ნახაზზეა მითითებული, რელაცია ORDER LINE და PRODUCT მესამე ნორმალური ფორმისაა. ამასთან, CUSTOMER ORDER შეიცავს ტრანზიტულ დამოკიდებულებებს და, შესაბამისად, (თუმცა მეორე ნორმალური ფორმა) ჯერ არ არის მესამე ნორმალურ ფორმაში.

პირველ ნორმალურ ფორმაში მყოფი დამოკიდებულება მეორე ნორმალურ ფორმაში იქნება, თუ რომელიმე შემდეგი პირობებიდან ერთი მოქმედებს:

1. პირველადი გასაღები შედგება მხოლოდ ერთი ატრიბუტისგან (მაგალითად, ატრიბუტი ProductID რელაციაში PRODUCT, ნახაზი 4-28). განმარტების თანახმად, ასეთ რელაციაში ნაწილობრივი დამოკიდებულება არ შეიძლება იყოს.
2. რლლაციაში არ არსებობს არაგასაღები ატრიბუტები (ამრიგად, რელაციაში არსებული ყველა ატრიბუტი პირველადი გასაღების კომპონენტებია). ასეთ დამოკიდებულებაში ფუნქციური დამოკიდებულება არ არსებობს
3. ყველა არაგასაღების ატრიბუტი ფუნქციურად დამოკიდებულია პირველად გასაღებში შემავალი ატრიბუტების სრულ ნაკრებზე (მაგ., ატრიბუტი OrderedQuantity ORDER LINE მიმართებაში, სურათი 4-28).



**FIGURE 4-28** Removing partial dependencies

### ბიჯი 3: მესამე ნორმალურ ფორმაში გარდაქმნა

რელაცია მესამე ნორმალურ ფორმაშია (3nF), თუ ის მეორე ნორმალურ ფორმაშია და ტრანზიტული დამოკიდებულებები არ არსებობს. რელაციაში ტრანზიტული დამოკიდებულება არის ფუნქციური დამოკიდებულება პირველადი გასაღებისა და ერთი ან მეტი არაგასაღებ ატრიბუტებს შორის, რომლებიც დამოკიდებულია პირველად გასაღებზე სხვა არაგასაღები ატრიბუტის საშუალებით. მაგალითად, CUSTOMER ORDER რელაციაში ორი ტრანზიტული დამოკიდებულებაა, ნაჩვენებია ნახაზზე 4-28:

---

OrderID → CustomerID → CustomerName

OrderID → CustomerID → CustomerAddress

---

სხვა სიტყვებით რომ ვთქვათ, მომხმარებლის სახელი და მისამართი ცალსახადა იდენტიფიცირებული CustomerID- ის მიერ, მაგრამ CustomerID არ არის პირველადი გასაღების ნაწილი (როგორც ადრე აღვნიშნეთ).

ტრანზიტული დამოკიდებულება ქმნის არასაჭირო სიჭარბეს, რამაც შეიძლება გამოიწვიოს ადრე განხილული ანომალიების ტიპი. მაგალითად, CUSTOMER ORDER ტრანზიტული დამოკიდებულება (სურათი 4-28) მოითხოვს, რომ მომხმარებლის სახელი და მისამართი ხელახლა შეიტანოს ყოველ ჯერზე, როდესაც მომხმარებელი წარუდგენს ახალ შეკვეთას, მიუხედავად იმისა, რამდენჯერ არის შეყვანილი მანამდე.

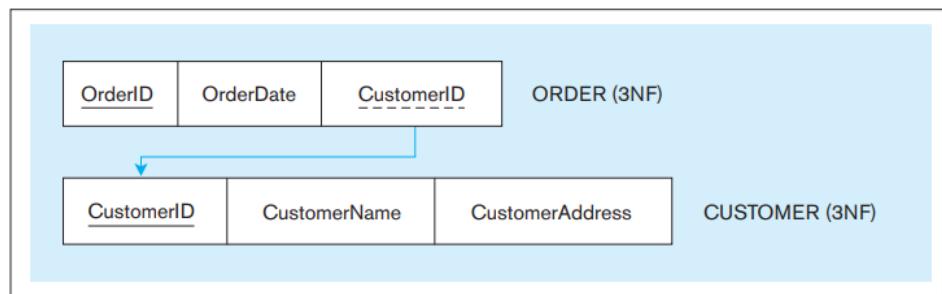
#### ტრანზიტული დამოკიდებულების ამოღება

რელაციიდან ტრანზიტული დამოკიდებულების ამოღება მარტივადაა შესაძლებელი სამსაფეხურიანი პროცედურის საშუალებით:

1. თითოეული არაგასაღები ატრიბუტისთვის (ან ატრიბუტების სიმრავლისთვის), რომელიც დეტერმინანტია რელაციაში, შეიქმნას ახალი რელაცია. ეს ატრიბუტი (ან ატრიბუტების ნაკრები) ხდება ახალი რელაციის პირველადი გასაღები.
2. მოხდეს ყველა ატრიბუტის, რომლებიც ფუნქციურად არის დამოკიდებული მხოლოდ ახალი რელაციის პირველად გასაღებზე, გადატანა ძველიდან ახალ რელაციაში.
3. ატრიბუტი, რომელიც წარმოადგენს პირველად გასაღებს ახალ რელაციაში, დარჩეს როგორც გარე გასაღები, რომელიც საშუალებას მოგვცენს დავაკავშროთ ორი რელაცია.

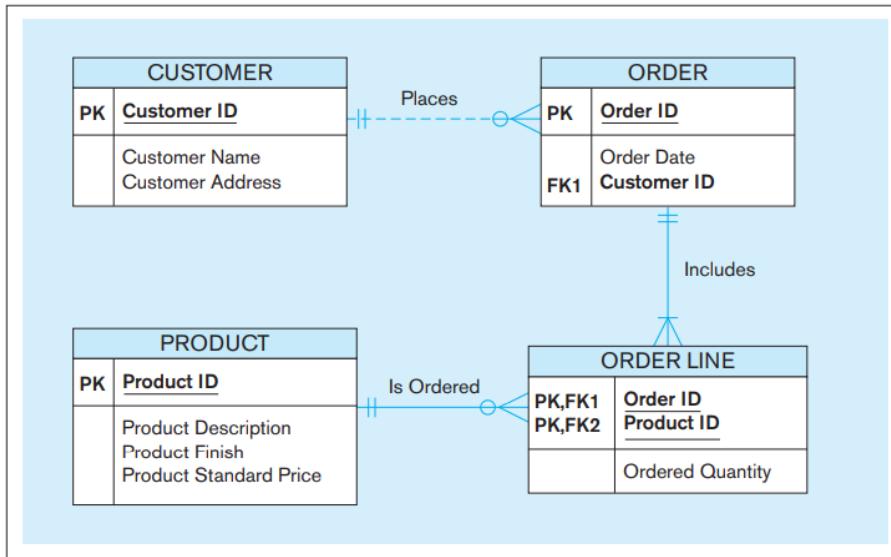
ამ ნაბიჯების გამოყენების შედეგები CUSTOMER ORDER რელაციასთან დაკავშირებით ნაჩვენებია დიაგრამა 4-29. შეიქმნა ახალი რელაცია სახელწოდებით CUSTOMER, ტრანზიტული დამოკიდებულების კომპონენტების მისაღებად. დეტერმინანტი CustomerID ხდება ამ რელაციის პირველადი გასაღები და ატრიბუტები CustomerName და CustomerAddress გადადის რელაციაში CUSTOMER ORDER ეწოდა ORDER და ატრიბუტი CustomerID რჩება როგორც უცხოური გასაღები ამ მიმართებაში. ეს საშუალებას გვაძლევს შევასრულოთ შეკვეთა მომხმარებელთან, რომელმაც შეკვეთა წარადგინა. როგორც 4-29 გრაფიკზეა მითითებული, ეს რელაციები ახლა მესამე ნორმალურ ფორმაშია.

**FIGURE 4-29** Removing transitive dependencies



INVOICE ხედვის მონაცემების ნორმალიზებამ შექმნა ოთხი რელაცია მესამე ნორმალურ ფორმაში: CUSTOMER, PRODUCT, ORDER და ORDER LINE. ამ ოთხი რელაციის და მათი ასოციაციების ამსახველი რელაციური სქემა (რომელიც შექმნილია Microsoft Visio- ს გამოყენებით) ნაჩვენებია დიაგრამა 4-30. გავითვალისწინოთ, რომ CustomerID არის გარე

გასაღები ORDER-ში, ხოლო OrderID და ProductID არის გარე გასაღებები ORDER LINE-ში (გარე გასაღებები ნაჩვენებია Visio-ში ლოგიკური, მაგრამ არა კონცეპტუალური მონაცემების მოდელისთვის). ასევე გავითვალისწინოთ, რომ მინიმალური კარდინალურობები ნაჩვენებია კავშირებზე, მიუხედავად იმისა, რომ ნორმალიზებული შედეგები არ აჩვენებს იმას, თუ რა უნდა იყოს მინიმალური კარდინალურობები.



**FIGURE 4-30** Relational schema for INVOICE data (Microsoft Visio notation)

#### დეტერმინანტები და ნორმალიზაცია

ჩვენ ვაჩვენეთ ნორმალიზაცია 3NF-ის მეშვეობით ეტაპობრივად. თუმცა არსებობს მარტივი მალსახმობიც. თუ გადახედავთ ოთხი დეტერმინანტის საწყის კომპლექტს და მასთან დაკავშირებული ფუნქციურ დამოკიდებულებებს ინვოისის მომხმარებლის ხედიდან, თითოეული მათგანი შეესაბამება ერთ-ერთ რელაციას დიაგრამა 4-30. თითოეული დეტერმინანტი არის რელაციის პირველადი გასაღები, ხოლო თითოეული რელაციის არაგასაღები არის ის ატრიბუტები, რომლებიც ფუნქციურად დამოკიდებულია თითოეულ დეტერმინანტზე. არსებობს დახვეწილი, მაგრამ მნიშვნელოვანი განსხვავება: იმის გამო, რომ OrderID განსაზღვრავს CustomerID, CustomerName და CustomerAddress და CustomerID განსაზღვრავს მის დამოკიდებულ ატრიბუტებს, CustomerID ხდება გარე გასაღები ORDER რელაციაში, სადაც წარმოდგენილია CustomerName და CustomerAddress. თუ შესაძლებელია დეტერმინანტების განსაზღვრა, რომლებსაც არ აქვთ გადამფარი დამოკიდებული ატრიბუტები, შესაძლებელია კავშირის განსაზღვრაც. ამრიგად, შესაძლებელია ნორმალიზაციის ეტაპობრივად განხორციელება, როგორც ილუსტრირებულია მოყვანილ მაგალითში „ინვოისზე“, ან შესაძლებელია შეიქმნას რელაციები პირდაპირ 3NF-ში დეტერმინანტების ფუნქციური დამოკიდებულებიდან.

#### ბიჯი 4: შემდგომი ნორმალიზაცია

0-დან 3-ის ჩათვლთ ბიჯების დასრულების შემდეგ, ყველა არაგასაღები ატრიბუტი დამოკიდებული იქნება პირველადი გასაღების, მთლიანად პირველად გასაღებზე და სხვა

არაფერზე, გარდა ძირითადი გასაღებისა. სინამდვილეში, ნორმალური ფორმები არის ფუნქციური დამოკიდებულების წესები და, ამრიგად, შედეგია დეტერმინანტებისა და მათთან დაკავშირებული არაგასაღებების მოძებნა. ზემოთ აღწერილი ბიჯები ხელს უწყობს თითოეული დეტერმინანტისა და მასთან დაკავშირებული არაგასაღები ატრიბუტების შემცველი რელაციების შექმნას.

ნორმალიზაციის შესახებ განხილვის დასაწყისიდანვე გავიხსენოთ, რომ 3NF-ს გარდა დამატებით განმარტებულია სხვა ნორმალური ფორმებიც.

### რელაციების შერწყმა

ჩვენ აღვწერეთ, თუ როგორ უნდა გარდაქიმნას EER დიაგრამები რელაციებად. ეს ტრანსფორმაცია ხდება მაშინ, როდესაც ჩვენ ზემოდან ვიღებთ მონაცემთა მოთხოვნების დაწვრილებითი ანალიზის შედეგებს და ვიწყებთ მათი სტრუქტურირებას მონაცემთა ბაზაში განსახორციელებლად. შემდეგ აღვწერეთ, თუ როგორ უნდა გადავამოწმოთ მიღებული რელაციები ისე, რომ აკმაყოფილებდნენ მესამე (ან უფრო მაღალი) ნორმალური ფორმას და საჭიროების შემთხვევაში შევასრულოთ ნორმალიზაციის ბიჯები.

ლოგიკური დიზაინის ჰარგლებში, ნორმალიზებული რელაციები შეიძლება შეიქმნას რიგი ცალკეული EER დიაგრამებიდან და (შესაძლოა) მომხმარებლის სხვა ხედვებისგან (მაგ., შეიძლება არსებობდეს მონაცემთა ბაზის ქვედა ან პარალელური განვითარების ქმედება ორგანიზაციის სხვადასხვა სფეროებისთვისაც) როგორც ზემოდან ქვემოთ). მაგალითად, ნორმალიზაციის საილუსტრაციოდ წინა განყოფილებაში გამოყენებული ინვოისის გარდა, შეიძლება არსებობდეს შეკვეთის ფორმა, ანგარიშის ბალანსის ანგარიში, წარმოების მარმრუტიზაცია და მომხმარებლის სხვა ხედვები, რომელთაგან თითოეული ნორმალიზებულია ცალკე. ადრე განხილული მონაცემთა ბაზის სამსქემანი არქიტექტურა ხელს უწყობს მონაცემთა ბაზის განვითარების პროცესების ერთდროულად გამოყენებას. სინამდვილეში, საშუალო და მსხვილი ორგანიზაციების უმეტესობა ახდენს მრავალი დამოუკიდებელი სისტემების განვითარებას, რასაც გარკვეულ მომენტში შეიძლება დასჭირდეს გაერთიანება მონაცემთა ბაზის შესაქმნელად. შედეგიად ამ სხვადასხვა პროცესებით წარმოქმნილი ზოგიერთი რელაცია შეიძლება ზედმეტი იყოს; ანუ ისინი შეიძლება ისინი ეხებოდეს ერთიდამავე ობიექტებს. ასეთ შემთხვევებში, უნდა მოხდეს მათი გაერთიანება სიჭარბის აღმოსაფხვრელად. აღვწეროთ ეს პროცესი - რელაციების შერწყმა (ასევე უწოდებენ ხედვის ინტეგრაციას). ეს მნიშვნელოვანია სამი მიზეზის გამო:

1. მსხვილ პროექტებზე მუშაობა ლოგიკური დიზაინის დროს რამდენიმე ქვეჯგუფს აერთიანებს, ამიტომ ხშირად ხდება რელაციების შერწყმის საჭიროება.
2. არსებული მონაცემთა ბაზის ახალი ინფორმაციის მოთხოვნებთან ინტეგრირება ხშირად იწვევს სხვადასხვა ხედვების ინტეგრირების აუცილებლობას.
3. სასიცოცხლო ციკლის განმავლობაში შეიძლება წარმოიშვას მონაცემთა ახალი მოთხოვნები, ამიტომ საჭიროა ნებისმიერი ახალი რელაციის შერწყმა უკვე შემუშავებულთან.

### მაგალითი

დავუშვათ, რომ მომხმარებლის ხედვის მოდელირება წარმოქმნის შემდეგ 3NF რელაციას:

---

EMPLOYEE1(EmployeeID, Name, Address, Phone)

მეორე მომხმარებლის ხედვის მოდელობამ შეიძლება მოგვცეს შემდეგი რელაცია:

EMPLOYEE2(EmployeeID, Name, Address, Jobcode, NoYears)

იმის გამო, რომ ამ ორ რელაციას აქვს ერთი და იგივე პირველადი გასაღები (EmployeeID), ისინი სავარაუდოდ აღწერს ერთსა და იმავე ობიექტს და შეიძლება გაერთიანდეს ერთ რელაციაში. რელაციების შერწყმის შედეგია შემდეგი რელაცია:

EMPLOYEE(EmployeeID, Name, Address, Phone, Jobcode, NoYears)

გავითვალისწინოთ, რომ ატრიბუტი, რომელიც ორივე რელაციაში ჩანს (მაგ., სახელი ამ მაგალითში) შერწყმულ რელაციაში მხოლოდ ერთხელ გამოჩნდება.

#### ინტეგრაციის პრობლემების მიმოხილვა

რელაციების ინტეგრირებისას, როგორც წინა მაგალითში, მონაცემთა ბაზის ანალიტიკოსმა უნდა გააცნობიეროს მონაცემთა მნიშვნელობა და მომზადებული უნდა იყოს ნებისმიერი პრობლემის გადასაჭრელად, რომელიც შეიძლება წარმოიშვას ამ პროცესში. ამ ნაწილში ჩვენ აღწერს და მოკლედ ასახავს ოთხ პრობლემას, რომლებიც წარმოიქმნება ინტეგრაციის თვალსაზრისით: სინონიმები, ჰომონიმები, ტრანზიტული დამოკიდებულებები და სუპერტიპის / ქვეტიპის კავშირები.

#### სინონიმები

ზოგიერთ სიტუაციაში, ორ (ან მეტ) ატრიბუტს შეიძლება ჰქონდეს განსხვავებული სახელები, მაგრამ იგივე მნიშვნელობა (მაგალითად, როდესაც ისინი აღწერენ არსის ერთსა და იმავე მახასიათებელს). ასეთ ატრიბუტებს სინონიმებს უწოდებენ. მაგალითად, EmployeeID და EmployeeNo შეიძლება იყოს სინონიმები. სინონიმების შემცველი რელაციების შერწყმისას, მომხმარებლებისგან უნდა მიიღოთ შეთანხმება (თუ შესაძლებელია) ატრიბუტის ერთ, სტანდარტიზებულ სახელზე და ამოღებულ იქნას სხვა სინონიმები (სხვა ალტერნატივაა მესამე სახელის არჩევა სინონიმების შესაცვლელად). მაგალითად, განვიხილოთ შემდეგი რელაციები:

STUDENT1(StudentID, Name)

STUDENT2(MatriculationNo, Name, Address)

ამ შემთხვევაში, ანალიტიკოსი ადასტურებს, რომ StudentID და MatriculationNo არის სტუდენტის პირადობის ნომრის სინონიმები და იდენტური ატრიბუტებია (კიდევ ერთი

შესაძლებლობა ის არის, რომ ეს ორივე კანდიდატი გასაღებია და მხოლოდ ერთი მათგანი უნდა იყოს არჩეული როგორც პირველადი გასაღები). ერთი შესაძლო გადაწყვეტა იქნება ორი ატრიბუტის სახელიდან ერთის სტანდარტიზაცია, მაგალითად StudentID. კიდევ ერთი ვარიანტია გამოვიყენოთ ახალი ატრიბუტის სახელი, მაგალითად StudentNo, ორივე სინონიმის შესაცვლელად. ამ უკანასკნელი მიდგომის ვარაუდით, ორი რელაციია შერწყმა შემდეგ შედეგს გამოიღებს:

---

**STUDENT(StudentNo, Name, Address)**

---

ხშირად, როდესაც არსებობს სინონიმები, საჭიროა მონაცემთა ბაზის ზოგიერთ მომხმარებელს მისცეს საშუალება მოახდინოს ერთიდაიმავე მონაცემებზე მითითება სხვადასხვა სახელით. მომხმარებლისთვის შეიძლება საჭირო გახდეს ნაცნობი სახელების გამოყენება, რომლებიც შეესაბამება ორგანიზაციის მათ ნაწილში არსებულ ტერმინოლოგიას. მეტსახელი (ფსევდონიმი) - არის ატრიბუტისთვის გამოყენებული ალტერნატიული სახელი. მონაცემთა ბაზის მართვის მრავალი სისტემა სამუალებას იძლევა განისაზღვროს მეტსახელი, რომელიც შეიძლება გამოყენებულ იქნას პირველადი ატრიბუტის ჭდის ნაცვლად.

**ობონიმები**

ატრიბუტის სახელს, რომელსაც შეიძლება ჰქონდეს ერთზე მეტი მნიშვნელობა, ჰომონიმს უწოდებენ. მაგალითად, ტერმინი account (ანგარიში) შეიძლება ეხებოდეს ბანკის მიმდინარე ანგარიშს, შემნახველ ანგარიშს, სესხის ანგარიშს ან სხვა სახის ანგარიშს (და შესაბამისად, ანგარიში გულისხმობს სხვადასხვა მონაცემს, მისი გამოყენების შესაბამისად)

რელაციების შერწყმისას უნდა მოხდეს ჰომონიმების მოძიება. განვიხილოთ შემდეგი მაგალითი:

---

**STUDENT1(StudentID, Name, Address)  
STUDENT2(StudentID, Name, PhoneNo, Address)**

---

მომხმარებლებთან დისკუსიების დროს, ანალიტიკოსმა შეიძლება აღმოაჩინოს, რომ ატრიბუტი მისამართი STUDENT1- ში ეხება სტუდენტის უნივერსიტეტის მისამართს, ხოლო STUDENT2- ში იგივე ატრიბუტი ეხება სტუდენტის მუდმივ (ან სახლის) მისამართს. ამ კონფლიქტის მოსაგვარებლად, აღბათ დაგვჭირდება ახალი ატრიბუტების სახელების შექმნა, რომ შერწყმული რელაცია გახდეს:

---

**STUDENT(StudentID, Name, PhoneNo, CampusAddress, PermanentAddress)**

---

### ტრანზიტული დამოკიდებულება

როდესაც ორი 3NF რელაცია შეირწყმება და ქმნის ერთ რელაცია, ამან შეიძლება გამოიწვიოს ტრანზიტული დამოკიდებულების წარმოქმნა. მაგალითად, განვიხილოთ შემდეგი ორი რელაცია:

---

STUDENT1(StudentID, MajorName)

STUDENT2(StudentID, Advisor)

---

იმის გამო, რომ STUDENT1 და STUDENT2 ერთნაირი პირველადი გასაღები აქვთ, ორი რელაციის შერწყმა შეიძლება:

---

STUDENT(StudentID, MajorName, Advisor)

---

ამასთან, ჩავთვალოთ, რომ თითოეულ MajorName (ძირითად მიმართულებას) ჰყავს ზუსტად ერთი მრჩეველი. ამ შემთხვევაში, მრჩეველი ფუნქციურად დამოკიდებულია MajorName-ზე:

---

MajorName → Advisor

---

თუ წინა ფუნქციური დამოკიდებულება არსებობს, მაშინ სტუდენტი არის 2NF, მაგრამ არა 3NF, რადგან ის შეიცავს ტრანზიტულ დამოკიდებულებას. ანალიტიკოსს შეუძლია შექმნას 3NF რელაციები ტრანზიტული დამოკიდებულების მოხსნის გზით. MajorName ხდება გარე გასაღები STUDENT-ში:

---

STUDENT(StudentID, MajorName)

MAJOR (MajorName, Advisor)

---

### სუპერტიპი/ქვეტიპი კავშირები

ეს კავშირები შეიძლება დამალული იყოს მომხმარებლის ხედვებში ან კავშირებში. დავუშვათ, რომ საავადმყოფოთვის შემდეგი ორი რელაცია გვაქვს:

---

PATIENT1(PatientID, Name, Address)

PATIENT2(PatientID, RoomNo)

---

თავდაპირველად, როგორც ჩანს, ეს ორი რელაცია შეიძლება გაერთიანდეს ერთ რელაციაში PATIENT. ამასთან, ანალიტიკოსი სწორად უჭვობს, რომ არსებობს ორი განსხვავებული ტიპის პაციენტი: რეზიდენტი პაციენტები და ამბულატორიები. PATIENT1 შეიცავს ატრიბუტებს, რომლებიც საერთოა ყველა პაციენტისათვის. PATIENT2 შეიცავს ატრიბუტს (RoomNo), რომელიც დამახასიათებელია მხოლოდ რეზიდენტი პაციენტებისათვის. ამ სიტუაციაში ანალიტიკოსმა უნდა შექმნას სუპერტიპი/ ქვეტიპი კავშირი ამ არსისთვის;

---

PATIENT(PatientID, Name, Address)  
RESIDENTPATIENT(PatientID, RoomNo)  
OUTPATIENT(PatientID, DateTreated)

---

ჩვენ შევქმნით OUTPATIENT რელაცია იმის საჩვენებლად, თუ როგორ შეიძლება გამოიყენოდეს საჭიროების შემთხვევაში, მაგრამ ეს არ არის აუცილებელი, მხოლოდ PATIENT1 და PATIENT2 მომხმარებლის ხედების გათვალისწინებით.

#### დამასრულებელი ბიჯი რელაციური გასაღებების განსასაზღვრად

ჩვენ ადრე განვიხილეთ იდენტიფიკატორების შერჩევის რამდენიმე კრიტერიუმი: ისინი არ იცვლიან მნიშვნელობებს დროთა განმავლობაში და უნდა იყოვნენ უნიკალური და ცნობილი, არაინტელექტუალური და იყენებდნენ ერთ სუროგატს კომპოზიტური ატრიბუტის იდენტიფიკატორისთვის. სინამდვილეში, ამ კრიტერიუმებიდან არც ერთი არ უნდა გამოიყენოდეს მონაცემთა ბაზის დანერგვამდე (ანუ, როდესაც იდენტიფიკატორი გახდება პირველადი გასაღები და განისაზღვრება, როგორც ველი ფიზიკურ მონაცემთა ბაზაში). სანამ რელაციები ცხრილებად გარდაიქმნება, საჭიროების შემთხვევაში, რელაციების ძირითადი გასაღებები უნდა შეიცვალოს ამ კრიტერიუმების შესაბამისად.

მონაცემთა ბაზის ექსპერტებმა გაამღიერეს კრიტერიუმები პირველადი გასაღების სპეციფიკისათვის. ასევე სასურველია, რომ პირველადი გასაღები იყოს უნიკალური მთელ მონაცემთა ბაზაში (ე.წ. საწარმოს გასაღები) და არა მხოლოდ უნიკალური იმ ფარდობით ცხრილში, რომელზეც ის გამოიყენება. პირველადი გასაღების ეს კრიტერიუმი უფრო ჰგავს იმას, რასაც ობიექტზე ორიენტირებულ მონაცემთა ბაზაში ობიექტის იდენტიფიკატორი ეწოდება. ამ რეკომენდაციით, რელაციის პირველადი გასაღები ხდება მონაცემთა ბაზის სისტემის შიდა მნიშვნელობა და არ აქვს ბიზნესის მნიშვნელობა.

კანდიდატი პირველადი გასაღები, როგორცაა EmpID ფიგურაში 4-1 EMPLOYEE1 რელაციაში ან CustomerID რელაციაში CUSTOMER (სურათი 4-29), თუ ოდესმე გამოიყენებოდა ორგანიზაციაში, ბიზნეს გასაღებს ან ბუნებრივ გასაღებს უწოდებენ და შევა რელაციაში როგორც არაგასაღები ატრიბუტი. EMPLOYEE1 და CUSTOMER რელაციებს (და მონაცემთა ბაზაში არსებულ ყველა სხვა რელაციას) შემდგომ აქვთ ახალი საწარმოს გასაღები ატრიბუტი (ე.წ. ვთქვათ, ObjectID), რომელსაც არ აქვს ბიზნესის მნიშვნელობა.

საწარმოს გასაღების გამოყენების ერთ-ერთი მთავარი მოტივაცია მონაცემთა ბაზის განვითარების შესაძლებლობა - მონაცემთა ბაზის შექმნის შემდეგ მონაცემთა ბაზაში ახალი რელაციების შერწყმა. მაგალითად, განვიხილოთ შემდეგი ორი რელაცია:

**EMPLOYEE(EmpID, EmpName, DeptName, Salary)**  
**CUSTOMER(CustID, CustName, Address)**

---

ამ მაგალითში, საწარმოს გასაღების გარეშე, EmpID- ს და CustID- ს შეიძლება ჰქონდეთ ან არ ჰქონდეთ იგივე ფორმატი, სიგრძე და მონაცემთა ტიპი, იმის მიუხედავად იქნება ის ინტელექტუალური თუ არაინტელექტუალური. დავუშვათ, ორგანიზაცია ავითარებს ინფორმაციის დამუშავების საჭიროებებს და აცნობიერებს, რომ თანამშრომლებს ასევე შეუძლიათ იყვნენ მომხმარებლები, ამიტომ თანამშრომელი და მომხმარებელი უბრალოდ ერთი და იგივე PERSON სუპერტიპის ორი ქვეტიპია. ამრიგად, ორგანიზაციას სურს ჰქონდეს სამი რელაცია:

**PERSON(PersonID, PersonName)**  
**EMPLOYEE(PersonID, DeptName, Salary)**  
**CUSTOMER(PersonID, Address)**

---

ამ შემთხვევაში, იგულისხმება, რომ ერთი ადამიანისათვის PersonID უნდა იყოს ერთი და იგივე ყველა როლის შემთხვევაში. თუ EmpID და CustID მნიშვნელობები შეირჩა PERSON რელაციის შექმნამდე, EmpID და CustID მნიშვნელობები აღბათ არ დაემთხვევა. უფრო მეტიც, თუ EmpID-ის და CustID-ის მნიშვნელობებს შევცვლით ახალი PersonID-ის შესაბამისად, როგორ უნდა დავრწმუნდეთ, რომ ყველა EmpID და CustIDs უნიკალურია, თუ სხვა თანამშრომელს ან მომხმარებელს უკვე აქვს მინიჭებული PersonID მნიშვნელობა? კიდევ უფრო უარესი, თუ არსებობს სხვა ცხრილები, რომლებიც ეხება, ვთქვათ, EMPLOYEE- ს, მაშინ ამ სხვა ცხრილების გარე გასაღებები უნდა შეიცვალოს, რაც ქმნის გარე გასაღებების ცვლილებების ტალღურ ეფექტს. ერთადერთი გარანტირებული გზა, რომ რელაციების თითოეული პირველადი გასაღები უნიკალურია მონაცემთა ბაზაში, არის თავიდანვე საწარმოს გასაღების შექმნა, ასე რომ პირველადი გასაღებები არასდროს არ უნდა შეიცვალოს.

ჩვენს მაგალითში თავდაპირველი მონაცემთა ბაზა (PERSON-ის გარეშე) საწარმოს გასაღებით ნაჩვენებია ნახატებში 4-31 ა (რელაციები) და 4-31 ბ (მონაცემების ნიმუში). ამ ფიგურაში, EmpID და CustID ახლა ბიზნესის გასაღებია და OBJECT ყველა სხვა რელაციების სუპერტიპია. OBJECT-ს შეიძლება ჰქონდეს ატრიბუტები, როგორიცაა ობიექტის ტიპის სახელი (ამ მაგალითში შედის როგორც ატრიბუტი ObjectType), შექმნის თარიღი, ბოლოს ცვლილების თარიღი ან ნებისმიერი სხვა შიდა სისტემის ატრიბუტი არსის ეგზემპლარისათვის. შემდგომ, როდესაც საჭიროა PERSON, მონაცემთა ბაზა ვითარდება დაპროექტების თვალსაზრისით, როგორც ნაჩვენებია 4-31c ნახაზებზე (რელაციები) და 4-31d (მონაცემთა ნიმუში). მონაცემთა ბაზაში PERSON- ის განვითარება კვლავ მოითხოვს არსებულ ცხრილებში გარკვეულ ცვლილებებს, მაგრამ არა პირველადი გასაღებების მნიშვნელობების შეცვლას. ატრიბუტი name გადადის PERSON-ში, რადგან ის საერთოა ორივე ქვეტიპისთვის და EMPLOYEE- ს და CUSTOMER- ს ემატება გარე გასაღები, რათა მოხდეს „საერთო“ პირის ეგზემპლარზე მითითება. ცხრილიში ადვილია არაგასაღები ატრიბუტების შესაბამისი სვეტების, თუნდაც გარე გასაღებების, დამატება და წაშლა. ამის საპირისპიროდ, რელაციის პირველადი გასაღების შეცვლა დაუშვებელია მონაცემთა ბაზის მართვის სისტემების უმრავლესობის მიერ, გარე გასაღების ტალღური ეფექტების დიდი გავლენის (ბაზაზე ზემოქმედების) გამო.

**FIGURE 4-31** Enterprise key

(a) Relations with enterprise key

OBJECT (OID, ObjectType)
EMPLOYEE (OID, EmplD, EmpName, DeptName, Salary)
CUSTOMER (OID, CustID, CustName, Address)

(b) Sample data with enterprise key

OBJECT	
OID	ObjectType
1	EMPLOYEE
2	CUSTOMER
3	CUSTOMER
4	EMPLOYEE
5	EMPLOYEE
6	CUSTOMER
7	CUSTOMER

EMPLOYEE

OID	EmplD	EmpName	DeptName	Salary
1	100	Jennings, Fred	Marketing	50000
4	101	Hopkins, Dan	Purchasing	45000
5	102	Huber, Ike	Accounting	45000

CUSTOMER

OID	CustID	CustName	Address
2	100	Fred's Warehouse	Greensboro, NC
3	101	Bargain Bonanza	Moscow, ID
6	102	Jasper's	Tallahassee, FL
7	103	Desks 'R Us	Kettering, OH

(c) Relations after adding PERSON relation

OBJECT (OID, ObjectType)
EMPLOYEE (OID, EmplD, DeptName, Salary, PersonID)
CUSTOMER (OID, CustID, Address, PersonID)
PERSON (OID, Name)

**FIGURE 4-31** (continued)

OBJECT		PERSON		EMPLOYEE		CUSTOMER		
<u>OID</u>	ObjectType	<u>OID</u>	Name	<u>OID</u>	EmplID	DeptName	Salary	<u>PersonID</u>
1	EMPLOYEE	8	Jennings, Fred	1	100	Marketing	50000	8
2	CUSTOMER	9	Fred's Warehouse	4	101	Purchasing	45000	11
3	CUSTOMER	10	Bargain Bonanza	5	102	Accounting	45000	12
4	EMPLOYEE	11	Hopkins, Dan					
5	EMPLOYEE	12	Huber, Ike					
6	CUSTOMER	13	Jasper's					
7	CUSTOMER	14	Desks 'R Us					
8	PERSON							
9	PERSON							
10	PERSON							
11	PERSON							
12	PERSON							
13	PERSON							
14	PERSON							

## ლაბორატორია 2

### მონაცემთა ბაზის შექმნა SQL Server-ში

SQL Server-ში მონაცემთა ბაზა შედგება სხვადასხვა ტიპის ობიექტებისაგან, როგორიცაა ცხრილები, ფუნქციები, შენახული(Stored) პროცედურები, View და ა.შ. SQL Server-ის თითოეულ ინსტანციას შეიძლება ჰქონდეს ერთი ან მეტი მონაცემთა ბაზა. SQL Server მონაცემთა ბაზები ინახება ფაილურ სისტემაში ფაილების სახით.

### მონაცემთა ბაზის ტიპები SQL Server-ში

SQL Server-ში გვხვდება ორი ტიპის მონაცემთა ბაზა: სისტემური მონაცემთა ბაზა და მომხმარებლის მონაცემთა ბაზა

სისტემური მონაცემთა ბაზები ავტომატურად იქმნება SQL Server-ის დაყენებისას. მათ იყენებს როგორც SSMS ასევე სხვა SQL Server API-ები და ხელსაწყოები, ამიტომ არ არის რეკომენდებული სისტემური მონაცემთა ბაზების ხელით შეცვლა:

- master: მონაცემთა ბაზა ინახავს სისტემის დონის ყველა ინფორმაციას SQL Server-ის ინსტანსზე. იგი მოიცავს მეტამონაცემებს, როგორიცაა შესვლის (Logins) ანგარიშები, ბოლო წერტილები (endpoints), დაკავშირებული სერვერები და სისტემის კონფიგურაციის პარამეტრები.
- model: სამოდელო მონაცემთა ბაზა გამოიყენება როგორც შაბლონი SQL Server-ის ინსტანციაზე შექმნილი ყველა მონაცემთა ბაზისთვის.
- msdb: ამ მონაცემთა ბაზას იყენებს SQL Server Agent-ი შეტყობინებების (alerts) და სამუშაოების (jobs) დაგეგმვისთვის და ასევე მას იყენებენ SQL Server Management Studio, Service Broker და Database Mail.
- tempdb: მონაცემთა ბაზა გამოიყენება დროებითი ობიექტების, შუალედური შედეგების და შიდა ობიექტების შესანახად, რომელიც მონაცემთა ბაზის dmv-ების ქმნის.

მომხმარებლის მიერ განსაზღვრული მონაცემთა ბაზები იქმნება მონაცემთა ბაზის მომხმარებლის მიერ T-SQL ან SSMS გამოყენებით აპლიკაციის ან/და სხვა მონაცემების შესანახად. შესაძლებელია მაქსიმუმ 32767 მონაცემთა ბაზა შექმნას SQL Server ინსტანციაში.

SQL Server-ში ახალი მონაცემთა ბაზის შექმნის ორი გზა არსებობს:

1. T-SQL გამოყენებით
2. SQL Server Management Studio-ს გამოყენებით

### შექმნით მონაცემთა ბაზა T-SQL სკრიპტის გამოყენებით

თქვენ შეგიძლიათ შეასრულოთ SQL სკრიპტი Query რედაქტორში Masterმონაცემთა ბაზის გამოყენებით. შეგახსენებთ რომ Query რედაქტორის გამოძახება შეგიძლიათ ინსტრუმენტების პანელზე არსებული New Query ღილაკის დაწყაპუნებით, ამ კლავიშთა კომბინაციით Ctrl+N, ავტოფორმით სკრიპტი:

**მზა სკრიპტის გაშვება შესაძლებელია Execute ღილაკით ან F5 კლავიშით.**

სინტაქსი:

```
USE [master]
GO
CREATE DATABASE <name>
```

შემდეგი სკრიპტი ქმნის "Cinema" მონაცემთა ბაზას.

```
USE [master]
GO
CREATE DATABASE [Cinema]
```

თუმცა ამ შემთხვევაში ბაზის ყველა პარამეტრი არჩეულია გაჩუმების პრინციპით. თუ გვსურს დავალაგოთ ბაზის ფიზიკური სტრუქტურა, შევცვალოთ ფაილების რაოდენობა, ლოკაცია ან ზომები საჭიროა თავად დავწეროთ და მივუთითოთ ბაზის პარამეტრები.

Ms SQL საშუალებას გვაძლევს ბაზის შექმნის დროს წინასწარ დავალაგოთ მისი ფაილების სრუქტურა, შევქმნათ ფაილური აგუფები და მათში დავანაწილოთ ფაილები და შესაბამისად ცხრილებიც. აღნიშნული სტრუქტურის სწორად დალაგება იქნება შემდგომ ბაზის ფიზიკური ფაილების ადვილად მართვის გარანტი, როგორც მათი მიგრაციისას ასევე სარუზერვო ასლების შექმნისას. მიუხედავად ამ ყველაფრისა მაინც ძირითად გამოწვევად Ms SQL ბაზის შექმნის დროს მიიჩნევა მისი რელაციური სტრუქტურის სწორად დალაგება, და ვფიქრობ რომ ნაკლები ყურადღება ექცევა მის ფაილურ და ფიზიკურ ნაწილს... მეტიც, ძალიან ხშირად პროგრამისტები მიიჩნევენ რომ ოპტიმალურიცაა ბაზისთვის გაჩუმების პრინციპით გამოყოლი ორი ფაილი: ერთი ძირითადი ფაილი მონაცემებისათვის-.mdf და მეორე ტრანზაქციის ლოგი .ldf.

თავიდან რომ დავიწყოთ, თითოეული ბაზის შექმნის დროს იქმნება ერთი ფაილური აგუფი Primary, და მასში ავტომატურად განთავსდება ერთი ძირითადი ფაილი mdf (Main Database File) და ასევე იქმნება ერთი ტრანზაქციის ლოგის ფაილი ldf (Log Database File) რომელიც ინახავს მთლიანი ბაზის ქმედებების ისტორიას როგორც სრული ასევე ნაწილობრივი ტრანზაქციების დროს, იგი არ მიეკუთვნება არცერთ ფაილურ აგუფს.

**CREATE DATABASE** Cinema შემოკლებული T-SQL ბრძანების შესრულების დროს გაჩუმების პრინციპით იქმნება მონაცემთა ბაზა Cinema ხოლო სერვერის დისკზე ასევე გაჩუმების პრინციპით წინასწარ არჩეულ ლოკაციაზე იქმნება ზემოთ ნახსენები ორი ფაილი, რომელთა დასახელებებიც ასევე არჩეულია წინასწარ და ძირითად ფაილს აქვს იგივე დასახელება რაც ბაზას, ჩვენს შემთხვევაში Cinema.mdf ხოლო ლოგს კი Cinema\_Log.ldf.

```
CREATE DATABASE <database name>
[ON [PRIMARY]
 ([NAME = <'logical file name'>,]
  FILENAME = <'file name'>
  [, SIZE = <size in kilobytes, megabytes, gigabytes, or terrabytes>]
  [, MAXSIZE = size in kilobytes, megabytes, gigabytes, or terrabytes>]
  [, FILEGROWTH = <kilobytes, megabytes, gigabytes, or terrabytes|percentage>] )
[LOG ON
 ([NAME = <'logical file name'>,]
  FILENAME = <'file name'>
  [, SIZE = <size in kilobytes, megabytes, gigabytes, or terrabytes>]
  [, MAXSIZE = size in kilobytes, megabytes, gigabytes, or terrabytes>]
  [, FILEGROWTH = <kilobytes, megabytes, gigabytes, or terrabytes|percentage>] )]
```

- სტრუქტურაში ჩანს ყველა SQL Server მონაცემთა ბაზისთვის სავალდებულო ოპერაციული სისტემის ფაილი ორი: **მონაცემთა ფაილი და Log (ჟურნალის) ფაილი**,
- მონაცემთა ფაილები შეიცავს მონაცემებს და ობიექტებს, როგორიცაა ცხრილები, ხედები (View), შენახული პროცედურები, ინდექსები და ა.შ.
  - ჟურნალის ფაილები შეიცავს ინფორმაციას, რომელიც საჭიროა მონაცემთა ბაზაში ყველა ტრანზაქციის აღსადგენად. უნდა არსებობდეს მინიმუმ ერთი ჟურნალის ფაილი თითოეული მონაცემთა ბაზისთვის.

## გავიაზროთ ფაილის შექმნის ბრძანების თითოეული პარამეტრის დანიშულება:

- **ON** პირობის შემდეგ ხდება მითითება თუ რომელ ფაილურ ჯგუფში ხვდება ფაილი, სტანდარტულ შემთხვევაში გვაქვს ორი სავალდებულო ფაილი, ესენია: **მონაცემთა ფაილი და Log (ჟურნალის)** ფაილი, აქედან პირველი ხვდება ძირითად (Primary) ფაილურ ჯგუფში, რისი მითითებაც ხდება On პირობით, ჩვენს შემთხვევაში **ON PRIMARY** წინადადებით, ხოლო მეორე სავალდებულო ფაილი არის ტრანზაქციის ჟურნალი რომელიც არცერთ ჯგუფში არ ხვდება და მისთვის მიეთითება **LOG ON** პირობა. გარდა სავალდებულოდ არსებული ძირითად (Primary) ფაილური ჯგუფისა თქვენ შეგიძლიათ დამატებით შექმნათ ახალი ფაილური ჯგუფები და განათავსოთ ბაზის ფიზიკური ფაილები მათში. შეგასწენებით რომ ON პირობის მერე იწერება თუ რომელ ფაილურ ჯგუფში ფაილი.
- **NAME** პარამეტრში მიეთითება ფაილის ლოგიკური სახელი, რომლითაც ჩვენ მას საჭიროების შემთხვევაში მივმართავთ სკრიპტში.
- **FILENAME** განსაზღვრავს ფიზიკური ფაილის სახელს დისკზე. როგორც უკვე ვთქვით სავალდებულოა ერთი მონაცემთა ფაილის შექმნა ძირითად (Primary) ფაილურ ჯგუფში და მისი გაფართოვება იქნება .mdf და ერთი ტრანზაქციის ჟურნალის ფაილის შექმნა გაფართოვებით .ldf თუმცა დამატებით საჭიროების შემთხვევაში შეიძლება შექმნას დამატებითი მონაცემთა ფაილები .ndf გაფართოებით და დამატებითი ჟურნალის ფაილები კვლავ .ldf გაფართოვებით.
- **SIZE** განსაზღვრავს ფაილის საწყის ზომას, რომელიც გაიზრდება თუკი ფაილი ბოლომდე გამოყენებს რეზერვირებულ ზომას. იგი სტანდატულად ეთითება მეგაბაიტებში, თუმცა შესაძლებელია სხვა ერთეულების გამოყენებაც.
- **MAXSIZE** წარმოადგენს ფაილის ზომის „ზედა“ ზღვარს. მარტივად რომ ვთქვათ სადამდე უნდა გაიზარდოს ზომა ფაილის გავსების შემთხვევაში. თუკი მისი მნიშვნელობაა „UNLIMITED“ ე.ი. ფაილი გააგრძელებს ზრდას სანამ ფიზიკურ დისკზე იქნება ადგილი /ზრდის საშუალება. ეს პარამეტრიც სტანდატულად ეთითება მეგაბაიტებში, თუმცა შესაძლებელია სხვა ერთეულების გამოყენებაც.
- **FILEGROWTH** როგორც უკვე ავღნიშნეთ ბაზის ფაილი გადავსებისას იზრდება, თუმცა რამდენად გაიზრდება ყოველ ჯერზე კონტროლდება აღნიშული პარამეტრით, იგი ეთითება მეგაბაიტებში ან პროცენტულად, თუმცა შესაძლებელია სხვა ერთეულების გამოყენებაც.

ზემოთ ჩამოვთვალეთ ხუთი პარამეტრი რომელიც აქვს თითოეულ ბაზის ფაილს, თუმცა თითოეულ მათგანს აქვს „მნიშვნელობა გაჩუმების პრინციპით“ (default) რაც ნიშნავს რომ თუ ზომას ან მაქსიმალურ ზომას გამოვტოვებთ ისინი ფაილს მიენიჭება სტანდარტების მიხედვით.

### პირველი ბაზის სრული T-SQL სკრიპტი

ყოველივე ზემოთ აღნიშული მერე შესაძლებელია დავწეროთ ჩვენი პირველი ბაზის სრული T-SQL სკრიპტი თრივე სავალდებულო ფაილის აღწერით:

```
CREATE DATABASE [Cinema]
    ON PRIMARY
    (
        NAME = N'Cinema',
        FILENAME = N'D:\MyFile\Cinema.mdf' ,
        SIZE = 8MB ,
        MAXSIZE = UNLIMITED,
        FILEGROWTH = 64MB )

        LOG ON
        (
            NAME = N'Cinema_log',
            FILENAME = N'D:\MyFile\Cinema_log.ldf' ,
            SIZE = 8MB ,
            MAXSIZE = UNLIMITED,
            FILEGROWTH = 64MB )
GO
```

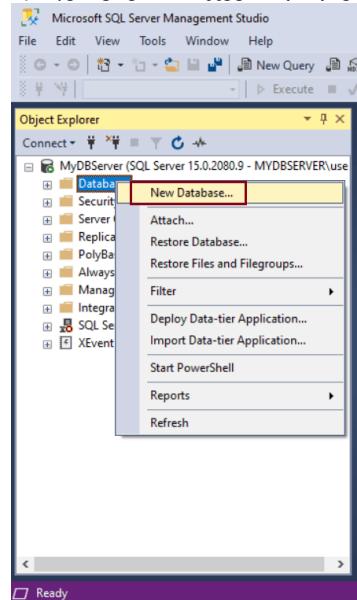
გაითვალისწინეთ საქალალდე, რომელ ლოკაციაზეც აპირებთ შექმნათ ბაზის ფაილები უნდა არსებობდეს წინასწარ.

ახლა გახსენით SSMS და განაახლეთ მონაცემთა ბაზების საქალალდე და დაინახავთ, რომ გამოჩნდა „Cinema“ მონაცემთა ბაზა.

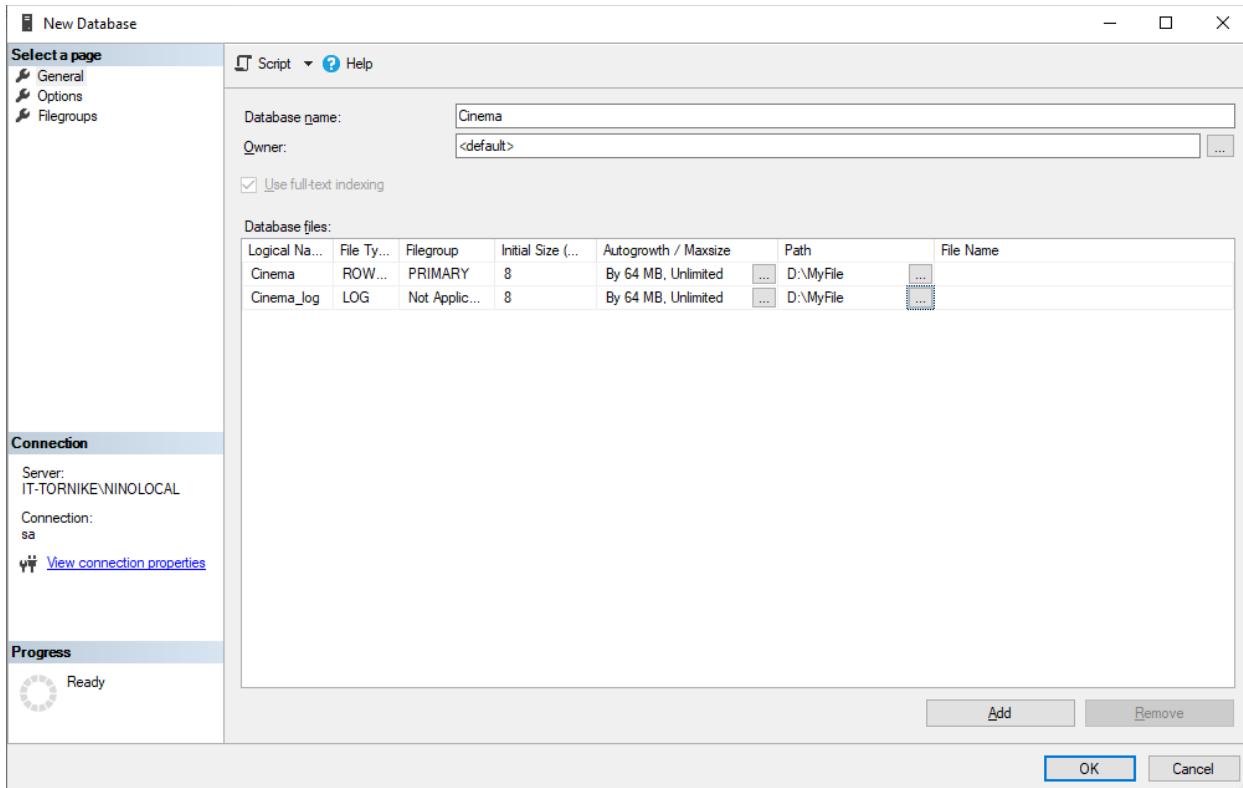
უკვე შექმნილი ბაზის Script ის ნახვა შესაძლებელია ბაზაზე მარჯვენა ღილაკით გამოტანილი კონტექსტური მენიუდან Script database As/Create to/New Query Editor Window ბრძანებით.

ახლა შევასრულოთ იგივე მოქმედება მენეჯმენტ სტუდიის დახმარებით: (გაითვალისწინეთ რომ სერვერზე დაუშვებელია ერთი და იმავე დასახელებით ორი ბაზის შექმნა)

დააწყაპუნეთ მარჯვენა ღილაკით Databases საქალალდეზე და დააწყაპუნეთ New Database.. მენიუს ბრძანებაზე:



ახალი მონაცემთა ბაზის ფანჯარაში შეიყვანეთ ახალი მონაცემთა ბაზის სახელი, როგორც ეს ნაჩვენებია ქვემოთ.

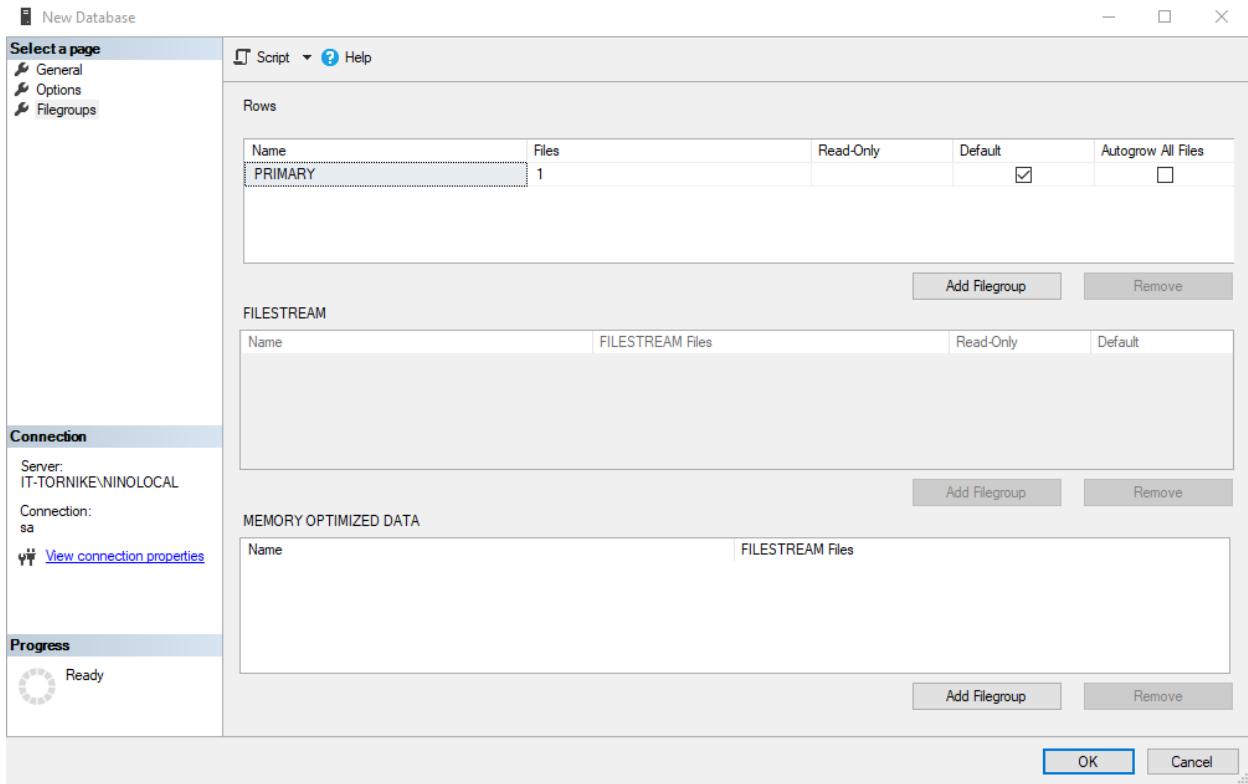


მონაცემთა ბაზის მფლობელი შეიძლება დარჩეს ნაგულისხმევად ან მფლობელის შესაცვლელად დააწყაპუნეთ ღილაკზე [...].

მონაცემთა ბაზის ფაილების ბადის ქვეშ ჩანს რომ ყველა SQL Server მონაცემთა ბაზას აქვს ჩვენთვის უკვე ნაცნობი ორი ოპერაციული სისტემის ფაილი: **მონაცემთა ფაილი და Log (ჟურნალის) ფაილი.** რომელთა დასახელებებიც ავტომატურად ივსება ბაზის დასახელების შესაბამისად, ისევე როგორც ზომები თუმცა თქვენ შეგიძლიათ შეცვალოთ ნაგულისხმევი მნიშვნელობები აღნიშნული ფაილებისთვის.

მონაცემთა ბაზის ოფციების (Options) გვერდზე თქვენ შეგიძლიათ შეცვალოთ Collation (სიმბოლოთა შაბლონები), Recovery (აღდგენის მოდელი) ასევე თავსებადობის დონე და შეკავების ტიპი, აღნიშნულ საკითხებს ამ ლექციაზე არ განვიხილავთ.

ახლა აირჩიეთ Filegroups ჩანართი. ფაილური ჯგუფები არის ფიზიკური ფაილები თქვენს დისკზე, სადაც ინახება SQL სერვერის მონაცემები. ნაგულისხმევად, ძირითადი (Primary) მონაცემთა ფაილი იქმნება ახალი მონაცემთა ბაზის შექმნისას, ხოლო ტრანზაქციის ლოგის ფაილი არ მიეკუთვნება არცერთ ფაილურ ჯგუფს. შესაბამისად გვექნება მხოლოდ ერთი ფაილური ჯგუფი:



Ok დილაკზე დაწყაპუნების შემდეგ შეიქმნება მონაცემთა ბაზა. რომლის დანახვაც შესაძლებელი იქნება Object Explorer ფანჯრის შიგნით, სერვერის ბაზების ჩამონათვალში.

თუმცა თუ ბაზის სტრუქტურიდან გამომდინარე წინასწარი გათვლით (იდეალურ შემთხვევაში) ვიცით რომ მოსალოდნელია ბაზის ზომის მკეთრად ზრდა, შესაბამისად დახარისხებული გვაქვს მონაცემები სწორად, რათა არ დავდგეთ დამუშავების დროის გაზრდის, გეგმიური სარეზერვო კოპირებების (back up) აღების ან სულაც მიმდინარე ფიზიკურ დისკზე არასაკმარისი სივრცის ქონის პრობლემასთან „სწორი“ გადაწყვეტილება იქნება, ბაზის დაშლა ფაილურ ჯეგუებად, და ერთის მაგიერ მონაცემთა რამოდენიმე ფიზიკური ფაილის განთავსება. ასევე შემდგომ მათში ცხრილების სწორად გადანაწილება.

მარტივად რომ ვთქვათ სხვადასხვა ‘მძიმე’ ცხრილები შეიძლება მოვათავსოთ სხვადასხვა ფიზიკურ ფაილებში, სხვადასხვა ლოკაციებზე, და ცალცალკე ვაკეთოთ მათი სარეზერვო კოპირება. მაგრამ ცხრილის ჩასმა ხდება ფაილურ ჯმუფში და არა უშუალოდ ფაილში. იგივე შეიძლება ითქვას სარეზერვო კოირებაზე. მაგალითისთვის შევქმათ იგივე ბაზა რამოდენიმე ფაილური ჯმუფითა და ფაილით:

```
CREATE DATABASE [Cinema]
ON PRIMARY
( NAME = N'Cinema',
FILENAME = N'D:\MyFile\Cinema.mdf' ,
SIZE = 8192KB ,
MAXSIZE = UNLIMITED,
FILEGROWTH = 65536KB ),

( NAME = N'Cinema',
FILENAME = N'D:\MyFile\Cinema.ndf' ,
SIZE = 1024KB ,
MAXSIZE = UNLIMITED,
FILEGROWTH = 65536KB ),
```

```

FILEGROUP [SECONDARY]
( NAME = N'Cinema3',
  FILENAME = N'D:\MyFile\Cinema3.ndf' ,
  SIZE = 1024KB , MAXSIZE = UNLIMITED,
  FILEGROWTH = 65536KB
)

LOG ON
( NAME = N'Cinema_log' , FILENAME = N'D:\MyFile\Cinema_log.ldf' , SIZE = 1024KB
, MAXSIZE = 2048GB , FILEGROWTH = 65536KB ),
( NAME = N'Cinema_log2' , FILENAME = N'D:\MyFile\Cinema_log2.ldf' , SIZE =
1024KB , MAXSIZE = 2048GB , FILEGROWTH = 65536KB )

```

თუ დავაკვირდებით შევამჩნევთ რომ მონაცემთა მთავარი ფაილი .mdf მხოლოდ ერთია ხოლო მომდევნო მონაცემთა ფაილები განურჩევლათ ფაილური ჯგუფისა არის .ndf გაფართოების და წარმოადგენს მონაცემთა მეორად ფაილებს.

აღნიშნულ ეტაპზე საქაღალდე გამოიყურება D:\MyFile შემდეგნაირად:

 Cinema.mdf	8,192 KB
 Cinema_log.ldf	1,024 KB
 Cinema_log2.ldf	1,024 KB
 Cinema2.ndf	1,024 KB
 Cinema3.ndf	1,024 KB

ხოლო შესაბამისი ბაზის მდგომარეობა სერვერზე შეგიძლიათ ნახოთ

EXEC sp\_helpdb [Cinema] ბრძანებით.

უკვე შექმნილი ფაილების მოდიფიკაცია შესაძლებელია **MODIFY** ბრძანებით, თუმცა მანამდე საჭიროა გავხსნათ ბაზის რედაქტირების რეჟიმი **ALTER** ბრძანებით. ამიერიდან თუ გვენდომება რაიმე ახალი ობიექტის შექმნა გამოვიყენებთ ჩვენთვის უკვე ნაცნობ ბრძანება **CREATE**-ს. ხოლო წასაშლელად ბრძანება **DROP**-ს.

შევცვალოთ ძირითადი ფაილის პარამეტრები:

```

ALTER DATABASE [Cinema]
MODIFY FILE
( NAME = N'Cinema',
  FILENAME = N'D:\MyFile\Cinema.mdf' ,
  SIZE = 20 MB ,
  MAXSIZE = 20 GB,
  FILEGROWTH = 8 MB )

```

GO

დავამატოთ ბაზას ერთი მეორადი ჯგუფის ფაილი, გავხსნათ ბაზის რედაქტირების რეჟიმი **ALTER** ბრძანებით და ჩავამატოთ შიგ ფაილი **ADD** ოპერატორის გამოყენებით:

```

ALTER DATABASE [Cinema]
ADD FILE
( NAME = N'Cinema4',
  FILENAME = N'D:\MyFile\Cinema4.ndf' ,
  SIZE = 30 MB ,
  MAXSIZE = 30 GB,
  FILEGROWTH = 10 MB )

```

წავშალოთ ბოლოს დამატებული ფაილი იგივე საფეხურების გამოორებით, გაითვალისწინეთ რომ წაშლისა და რედაქტირების დროს ფაილის ლოგიკური სახელი გამოიყენება მასზე მიმთითებლად.

```
ALTER DATABASE [Cinema]
    REMOVE FILE Cinema4
```

დავამატოთ ახალი ფაილური ჯგუფი

```
ALTER DATABASE [Cinema]
    ADD FILEGROUP NEWGROUP;
    დავამატოთ ბაზას კიდევ ერთი მონაცემთა ფაილი ამჯერად ახალ ფაილურ ჯგუფში :
```

```
ALTER DATABASE [Cinema]
    ADD FILE
        ( NAME = N'Cinema4',
        FILENAME = N'D:\MyFile\Cinema4.ndf'
    ) TO FILEGROUP NEWGROUP
```

დავამატოთ ტრანზაქციის ჟურნალი

```
ALTER DATABASE [Cinema]
    ADD LOG FILE
    (
        NAME = Cinema_log3,
        FILENAME = N'D:\MyFile\Cinema_lo3.ldf',
        SIZE = 5MB,
        MAXSIZE = 100MB,
        FILEGROWTH = 5MB
    )
```

როგორც უკვე ვიცით ბაზის შექმნისთანავე იქმნება ძირითადი ფაილური ჯგუფი რომელიც ასევე წარმოადგენს ჯგუფს გაჩუმების მეთოდით, რაც ნიშნავს რომ ფაილის დამატებისას თუ არ დავაკონკრეტებთ რომელ ჯგუფში უნდა ჩაჯდეს იგი ავტომატურად ჩაჯდება ძირითად (PRIMARY) ფაილურ ჯგუფში. Default ფაილური ჯგუფის ცვლილება შესაძლებელია შემდეგი სკრიპტით:

```
ALTER DATABASE [Cinema]
    MODIFY FILEGROUP NEWGROUP DEFAULT
```

ახლა კვლავ ძირითადი ჯგუფი დავაბრუნოთ Default-ად და NEWGROUP წავშალოთ. (გაითვალისწინეთ რომ წაშლისას ჯგუფში არ უნდა იყოს ფაილი. წაშალეთ ჯერ ფაილები შემდეგ აგრძელეთ:

```
ALTER DATABASE [Cinema]
    MODIFY FILEGROUP [PRIMARY] DEFAULT
```

```
ALTER DATABASE [Cinema]
    REMOVE FILE Cinema4
```

```
ALTER DATABASE [Cinema]
    REMOVE FILEGROUP NEWGROUP
```

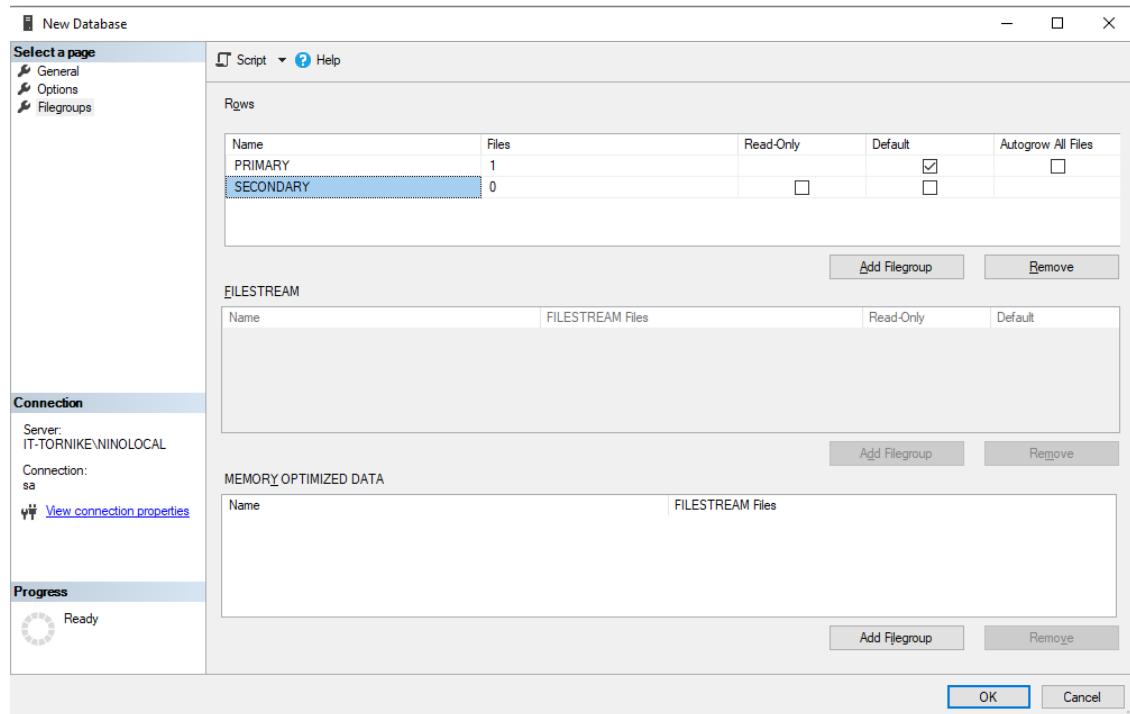
წავშალოთ ბაზა:

```
DROP DATABASE [Cinema]
```

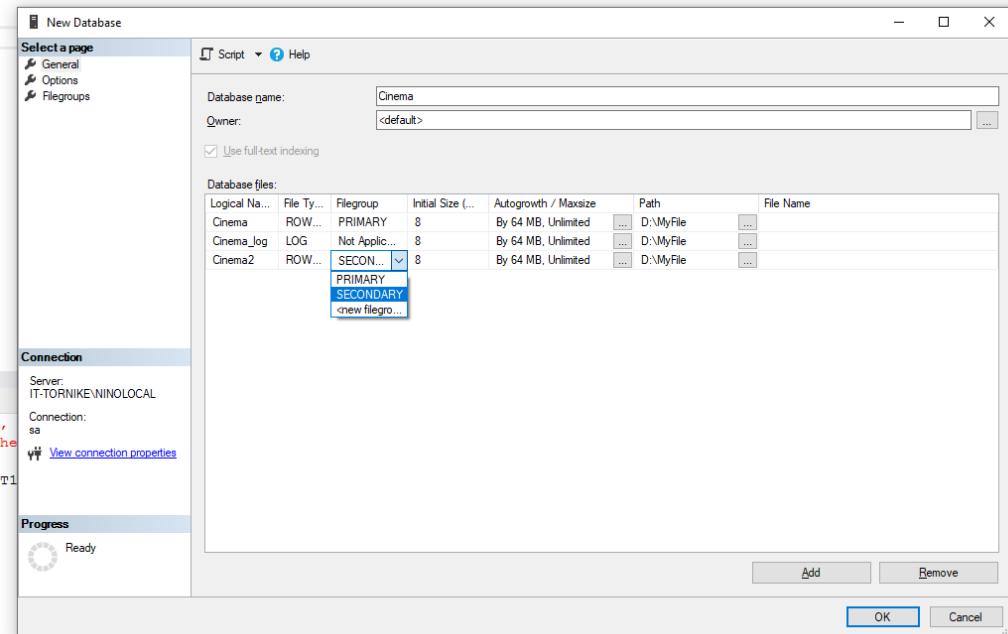
ვინაიდან აღნიშნულ თავში რამოდენიმე DDL (Data Definition Language)-ის ბაზის სტრუქტურირების ძირითადი ბრძანებები გავიარეთ, მივცეთ ამას ნორმალიზებული ფორმა მეხსიერებაში ☺

<b>Create</b>	ობიექტის შექმნა
<b>Drop</b>	ობიექტის წაშლა
<b>Alter</b>	ობიექტის ცვლილება ამ შემთხვევაში ხდება შიგთავსის ცვლილება და შესაძლებელია გამოყენებული იყოს ქვებრძანებები ძირითადთან ერთად: Add ქვეობიექტის ჩამატება Modify ქვეობიექტის რედაქტირება Remove ქვეობიექტის წაშლა

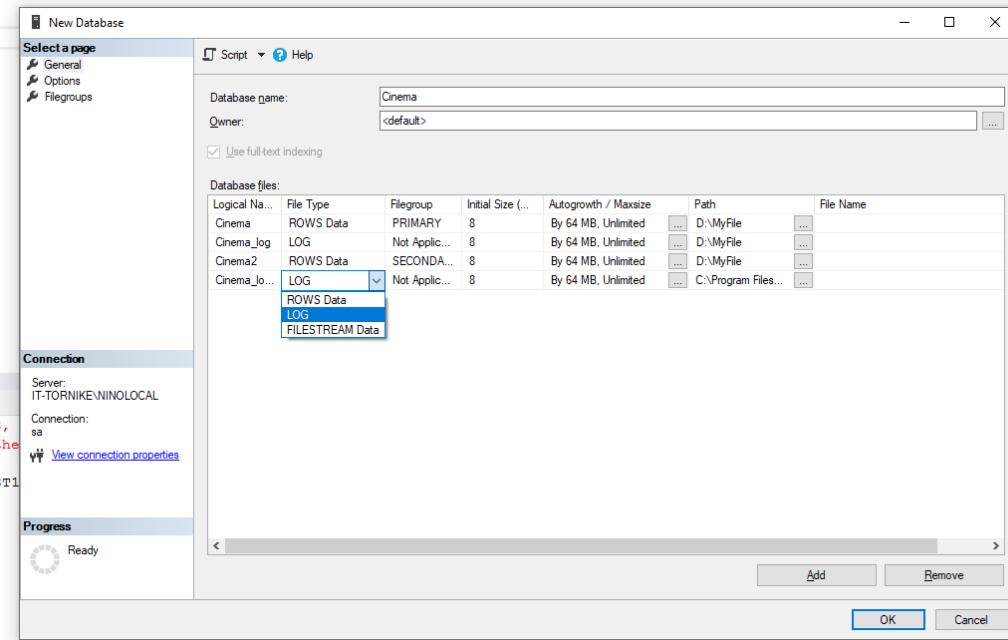
თუ დამატებითი ფაილებისა და ფაილური ჯგუფების დამატება გვსურს დიზაინის რეჟიმში, ბაზის შექმნის დროს ზემოთ ნაჩვენებ New Database ფანარში უნდა გადავიდეთ Filegroups ჩანართზე და Add FileGroup ღილაკით დავამატოთ ახალი ფაილური ჯგუფი, გავუწეროთ დასახელება:



შემდეგ დავბრუნდეთ General ჩანართში Add ღილაკის გამოყენებით დავამატოთ ფაილი, გავუწეროთ სასურველი სახელი და Filegroup გრაფაში ჩამოსაშლელ სიაში ავურჩიოთ ახლად შექმნილი ჯგუფი:



ანალოგიური მეთოდით შეიქმნება ტრანზაქციის ჟურნალები, მხოლოდ ფაილის ტიპში ROWS Data (მონაცემთა ფაილის) მაგიერ ავარჩევთ LOG(ჟურნალის) ტიპს



ასევე ავარჩევთ ფაილის სასურველ ლოკაციას. საჭიროების შემთხვევაში კი აქვე გვაქვს Remove ღილაკი არასაჭირო ფაილების წასაშლელად.

## SQL სერვერის მონაცემთა ტიპები

SQL Server-ში “data type” განსაზღვრავს მონაცემთა ტიპს, რომელიც შეიძლება ჩავწეროთ ცხრილის ამა თუ იმ სვეტში, მაგალითად ერთ სვეტში შეიძლება ვინახავდეთ მთელ რიცხვებს, მეორეში-სტრიქონულ მონაცემებს, მესამეში თარიღსა და დროს, ორობით სტრიქონებს და ა.შ.

მონაცემთა ტიპების სწორად განსაზღვრას გადამწყეტი მნიშვნელობა აქვს ბაზის დაპროექტებისას, ვინაიდან იგი გავლენას ახდენს ბაზის და შესაბამისად აპლიკაციის სწრაფქმედებასა და ეფექტურობაზე.

SQL Server-ში ჩაშენებულია ყველა სახის მონაცემთა ტიპები, თუმცა საჭიროების შემთხვევაში თქვენ თავადაც შეგიძლიათ შექმნათ საკუთარი ტიპი.

### მონაცემთა ტიპის კატეგორიები:

კატეგორია	მონაცემთა ტიპი
ზუსტი რიცხვები	bit, tinyint, smallint, int, bigint, decimal, numeric, money, smallmoney
ნამდვილი რიცხვები	Real, Float
თარიღი და დრო	date, smalldatetime, datetime, datetime2, datetimeoffset, time
სტრიქონი/სიმბოლო	char, varchar, text
უნიკოდის სიმბოლოები	nchar, nvarchar, ntext
სხვა	cursor, hierarchyid, sql_variant, spatial Geometry types, spatial Geography types, rowversion, uniqueidentifier, xml, table

### ზუსტი რიცხვები

მონაცემთა ტიპი	მნიშვნელობების დიაპაზონი	აღწერა/ზომა
bit	0,1 ან NULL	მონაცემთა ყველაზე პატარა ტიპი 1 ბაიტი ზომით.
tinyint	0255-მდე	1 ბაიტი
smallint	-32,768-დან 32,767-მდე	2 ბაიტი
int	-2,147, 483,648	4 ბაიტი
bigint	-9,223,372, 036,854,775,808	8 ბაიტი
decimal	-10^38+1-დან 10^38-1-მდე	რიცხვითი მონაცემთა ტიპი, რომელსაც აქვს ფიქსირებული სიზუსტე და მასშტაბი.
smallmoney	-214,748.3648-დან 214,748.3647-მდე	4 ბაიტი
money	-922,337,203,685,477,5808-დან 922,337,203,685,477.5807-მდე	8 ბაიტი

### ნამდვილი რიცხვები

მონაცემთა ტიპი	მნიშვნელობების დიაპაზონი	აღწერა/ზომა
float(n)	- 1.79E+308 -დან -2.23E-308-მდე, 0	ზომა დამოკიდებულია n-ის მნიშვნელობაზე
real	- 3.40E + 38 -დან -1.18E - 38-მდე, 0 და 1.18E - 38 -დან 3.40E + 38-მდე	4 ბაიტი

## თარიღი და დრო

მონაცემთა ტიპი	მნიშვნელობების დიაპაზონი	აღწერა/ზომა
date	0001-01-01 -დან 9999-12-31-მდე	3 ბაიტი
datetime	1753-01-01 -დან 9999-12-31-მდე	8 ბაიტი
datetime2	0001-01-01 -დან 9999-12-31-მდე	სიზუსტე < 3 : 6 ბაიტი
smalldatetime	1900-01-01 -დან 2079-06-06-მდე	4 ბაიტი
datetimeoffset	0001-01-01 -დან	10 ბაიტი
time	00:00:00.0000000 -დან 23:59:59.9999999-მდე	5 ბაიტი

## სტრიქონი/სიმბოლო

მონაცემთა ტიპი	მნიშვნელობების დიაპაზონი	აღწერა/ზომა
char[(n)]	1 -დან 8000 სიმბოლომდე	n ბაიტი
varchar[(n)]	1 -დან 8000	n ბაიტი + 2 ბაიტი
varchar(max)	1 -დან 2^31-1	n ბაიტი + 4 ბაიტი
text	0 -დან 2,147,483,647	n ბაიტი + 4 ბაიტი

## უნიკოდის სიმბოლოები

მონაცემთა ტიპი	მნიშვნელობების დიაპაზონი	აღწერა/ზომა
nchar[(n)]	1 -დან 4000 სიმბოლომდე	2n ბაიტი
nvarchar[(n   max)]	1 -დან 4000	2n ბაიტი
ntext	0 -დან 1,073,741,823	2-ჯერ მეტი სტრიქონის სიგრძეზე

## ორობითი სტრიქონები

მონაცემთა ტიპი	მნიშვნელობების დიაპაზონი	აღწერა/ზომა
binary[(n)]	1 -დან 8000 ბაიტამდე	n ბაიტი
varbinary[(n   max)]	1 -დან 8000	სტრიქონის სიგრძეს + 2 ბაიტი
Image	0 -დან 2,147,483,647	ცვლადი სიგრძის ორობითი მონაცემები

## სხვა

მონაცემთა ტიპი	აღწერა/ზომა
cursor	მონაცემთა ტიპი ცვლადებისთვის ან შენახული პროცედურების OUTPUT პარამეტრები, რომლებიც შეიცავს მითითებას კურსორზე.
rowversion	აბრუნებს ავტომატურად გენერირებულ უნიკალურ ორობით რიცხვებს მონაცემთა ბაზაში.
hierarchyid	ცვლადი სიგრძის სისტემის მონაცემთა ტიპი

მონაცემთა ტიპი	აღწერა/ზომა
uniqueidentifier	16 ბაიტიანი GUID (უნიკალური კოდი)
sql_variant	ინახავს სხვადასხვა SQL სერვერის მხარდაჭერილ მონაცემთა ტიპების მნიშვნელობებს.
xml	ინახავს xml მონაცემებს
Spatial Geometry type	გამოიყენება ბრტყელ კოორდინატულ სისტემაში (ევკლიდური) მონაცემების წარმოსაჩენად.
table	ეს არის სპეციალური მონაცემთა ტიპი, რომელიც გამოიყენება შედეგთა ნაკრების (result set) დროებით შესანახად მოგვიანებით დასამუშავებლად.

## ლაბორატორია 3

### Contents

ცხრილის შექმნა .....	2
ცხრილის შექმნა T-SQL სკრიპტის გამოყენებით .....	2
მონაცემთა ტიპის კატეგორიები: .....	3
ზუსტი რიცხვები .....	3
ნამდვილი რიცხვები .....	3
თარიღი და დრო .....	4
სტრიქონი/სიმბოლო .....	4
უნიკოდის სიმბოლოები .....	4
ორობითი სტრიქონები .....	4
სხვა .....	4
შეზღუდვები: .....	5
NOT NULL .....	5
IDENTITY .....	5
CHECK .....	5
UNIQUE .....	5
DEFAULT .....	6
PRIMARY KEY .....	6
FOREIGN KEY .....	6
შევქმნათ ცხრილი SSMS -ის გამოყენებით .....	8
ALTER TABLE ADD Columns სვეტის დამატება ცხრილში .....	11
სვეტის ან ცხრილის სახელის გადარქმევა .....	13
ცხრილისა და სვეტების სახელის გადარქმევა SSMS-ის გამოყენებით: .....	13
ცხრილის სვეტების/სვეტის წაშლა .....	14
წაშალეთ სვეტები SSMS-ის გამოყენებით .....	14
ცხრილის სვეტების მოდიფიკაცია .....	16
მოქმედებები შეზღუდვებზე .....	16
დამატებით ცხრილების შექმნა .....	17

## ცხრილის შექმნა

ცხრილები არის მონაცემთა ბაზის ობიექტები, რომლებიც შეიცავს მონაცემთა ბაზაში არსებულ ყველა მონაცემს. ცხრილში მონაცემები ლოგიკურად არის ორგანიზებული რიგებად და სვეტებად. თითოეული სვეტი წარმოადგენს ველს და თითოეული სტრიქონი კი წარმოადგენს უნიკალურ ჩანაწერს. ზემოთხსენებულ ყველა სვეტს აქვს მასთან დაკავშირებული მონაცემთა ტიპი. ის წარმოადგენს ამ სვეტის მონაცემთა ტიპს. თითოეული ცხრილის სახელი უნიკალურია მონაცემთა ბაზის დონეზე.

მონაცემთა ბაზაში ცხრილების რაოდენობა შეზღუდულია მხოლოდ მონაცემთა ბაზაში დაშვებული ობიექტების რაოდენობით (2,147,483,647). მომხმარებლის მიერ განსაზღვრულ ცხრილს კი შეიძლება ჰქონდეს 1024 სვეტამდე.

SQL Server-ში ახალი ცხრილის შესაქმნელად ორი გზა არსებობს:

- T-SQL სკრიპტის გამოყენება
- Table Designer-ის გამოყენება SQL Server Management Studio-ში

## ცხრილის შექმნა T-SQL სკრიპტის გამოყენებით

ამისათვის საჭიროა დავწეროთ CREATE TABLE ბრძანება SSMS-ის Query:

აღნიშნული ბრძანების სინტაქსი შემდეგია

```
CREATE TABLE [database_name.] [schema_name.] table_name (
    pk_column_name data_type PRIMARY KEY,
    column_name2 data_type [NULL | NOT NULL],
    column_name3 data_type [NULL | NOT NULL],
    . . . ,
    [table_constraints]
);
```

CREATE TABLE საბრძანებო სიტყვის შემდეგ მიეთითება ცხრილის სახელი (რა სახელითაც გვსურს მისი შექმნა), შესაძლებელია მივუთითოთ ცხრილის სრული ან შემოკლებული(მხოლოდ ცხრილის სახელი) მარტივად რომ ვთქვათ სრული სახელის მითითებისას იწერება ჯერ ბაზის სახელი, შემდეგ სქემის სახელი და ბოლოს თავად ცხრილის სახელი [database\_name.] [schema\_name.] table\_name სტილში. ხოლო მხოლოდ ცხრილის სახელის მითითებისას დანარჩენი პარამეტრები იგულისხმება გაჩუმების პრინციპით.

ცხრილის სახელის მითითების შემდეგ ვხსნით მრგვალ ფრჩხილებს-() და ვწერთ სვეტების ჩამონათვალს, რომლებიც გვინდა რომ გვქონდეს ცხრილში.

სვეტს გააჩნია ჩემდეგი სამი პარამეტრი

column\_name data\_type constraints

- დასახელება
- მონაცემთა ტიპი
- შეზღუდვა

აქედან მხოლოდ პირველი ორია სავალდებულო, ხოლო შეზღუდვები შეიძლება ან საერთოდ არ ჰქონდეს სვეტს, ან რამოდენიმე ერთად ედოს. სვეტები ერთმანეთისგან გამოყოფილია მძიმით. ასევე მნიშვნელოვანია რომ შეზღუდვები შეიძლება დაიწეროს როგორც სვეტის გვერდით ასევე სვეტების ჩამონათვლის ბოლოს.

### მონაცემთა ტიპის კატეგორიები:

ცხრილის სვეტებისთვის შეიძლება გამოყენებული იყოს მონაცემთა სხვადასხვა ტიპები:

კატეგორია	მონაცემთა ტიპი
ზუსტი რიცხვები	bit, tinyint, smallint, int, bigint, decimal, numeric, money, smallmoney
ნამდვილი რიცხვები	Real, Float
თარიღი და დრო	date, smalldatetime, datetime, datetime2, datetimeoffset, time
სტრიქონი/სიმბოლო	char, varchar, text
უნიკოდის სიმბოლოები	nchar, nvarchar, ntext
სხვა	cursor, hierarchyid, sql_variant, spatial Geometry types, spatial Geography types, rowversion, uniqueidentifier, xml, table

### ზუსტი რიცხვები

მონაცემთა ტიპი	მნიშვნელობების დიაპაზონი	აღწერა/ზომა
bit	0,1 ან NULL	მონაცემთა ყველაზე პატარა ტიპი 1 ბაიტი ზომით.
tinyint	0255-მდე	1 ბაიტი
smallint	-32,768-დან 32,767-მდე	2 ბაიტი
int	-2,147, 483,648	4 ბაიტი
bigint	-9,223,372, 036,854,775,808	8 ბაიტი
decimal	-10^38+1-დან 10^38-1-მდე	რიცხვითი მონაცემთა ტიპი, რომელსაც აქვს ფიქსირებული სიზუსტე და მასშტაბი.
smallmoney	-214,748.3648-დან 214,748.3647-მდე	4 ბაიტი
money	-922,337,203,685,477,5808-დან 922,337,203,685,477.5807-მდე	8 ბაიტი

### ნამდვილი რიცხვები

მონაცემთა ტიპი	მნიშვნელობების დიაპაზონი	აღწერა/ზომა
float(n)	- 1.79E+308 -დან -2.23E-308-მდე, 0	ზომა დამოკიდებულია n-ის მნიშვნელობაზე
real	- 3.40E + 38 -დან -1.18E - 38-მდე, 0 და 1.18E - 38 -დან 3.40E + 38-მდე	4 ბაიტი

## თარიღი და დრო

მონაცემთა ტიპი	მნიშვნელობების დიაპაზონი	აღწერა/ზომა
date	0001-01-01 -დან 9999-12-31-მდე	3 ბაიტი
datetime	1753-01-01 -დან 9999-12-31-მდე	8 ბაიტი
datetime2	0001-01-01 -დან 9999-12-31-მდე	სიზუსტე < 3 : 6 ბაიტი
smalldatetime	1900-01-01 -დან 2079-06-06-მდე	4 ბაიტი
datetimeoffset	0001-01-01 -დან	10 ბაიტი
time	00:00:00.0000000 -დან 23:59:59.9999999-მდე	5 ბაიტი

## სტრიქონი/სიმბოლო

მონაცემთა ტიპი	მნიშვნელობების დიაპაზონი	აღწერა/ზომა
char[(n)]	1 -დან 8000 სიმბოლომდე	n ბაიტი
varchar[(n)]	1 -დან 8000	n ბაიტი + 2 ბაიტი
varchar(max)	1 -დან 2^31-1	n ბაიტი + 4 ბაიტი
text	0 -დან 2,147,483,647	n ბაიტი + 4 ბაიტი

## უნიკოდის სიმბოლოები

მონაცემთა ტიპი	მნიშვნელობების დიაპაზონი	აღწერა/ზომა
nchar[(n)]	1 -დან 4000 სიმბოლომდე	2n ბაიტი
nvarchar[(n max)]	1 -დან 4000	2n ბაიტი
ntext	0 -დან 1,073,741,823	2-ჯერ მეტი სტრიქონის სიგრძეზე

## ორობითი სტრიქონები

მონაცემთა ტიპი	მნიშვნელობების დიაპაზონი	აღწერა/ზომა
binary[(n)]	1 -დან 8000 ბაიტამდე	n ბაიტი
varbinary[(n max)]	1 -დან 8000	სტრიქონის სიგრძეს + 2 ბაიტი
Image	0 -დან 2,147,483,647	ცვლადი სიგრძის ორობითი მონაცემები

## სხვა

მონაცემთა ტიპი	აღწერა/ზომა
cursor	მონაცემთა ტიპი ცვლადებისთვის ან შენახული პროცედურების OUTPUT პარამეტრები, რომლებიც შეიცავს მითითებას კურსორზე.
rowversion	აბრუნებს ავტომატურად გენერირებულ უნიკალურ ორობით რიცხვებს მონაცემთა ბაზაში.
hierarchyid	ცვლადი სიგრძის სისტემის მონაცემთა ტიპი
uniqueidentifier	16 ბაიტიანი GUID (უნიკალური კოდი)

მონაცემთა ტიპი	აღწერა/ზომა
sql_variant	ინახავს სხვადასხვა SQL სერვერის მხარდაჭერილ მონაცემთა ტიპების მნიშვნელობებს.
xml	ინახავს xml მონაცემებს
Spatial Geometry type	გამოიყენება ბრტყელ კოორდინატულ სისტემაში (ეკლიდური) მონაცემების წარმოსაჩენად.
table	ეს არის სპეციალური მონაცემთა ტიპი, რომელიც გამოიყენება შედეგთა ნაკრების (result set) დროებით შესანახად მოგვიანებით დასამუშავებლად.

## შეზღუდვები:

მოკლედ განვიხილოთ რა შეზღუდვები შეიძლება ჰქონდეს სვეტს/სვეტებს:

### NOT NULL

DirectionId INT NOT NULL

არაცარიელი სვეტი. თუ სვეტში არ შეგვაქვს ჩანაწერი მაშინ მას აქვს NULL მნიშვნელობა, შესაბამისად ცარიელია და წარმოადგენს არასავალდებულო ველს. არასავალდებულო ველებისგან განსხვავებით, სავალდებულო ველებში აუცილებელია მნიშვნელობის შეტანა. თუკი სვეტს მინიჭებული აქვს NOT NULL შეზღუდვა ე.ი ის წარმოადგენს სავალდებულო ველს და ჩანაწერების შეტანისას მისი გამოტოვება შეუძლებელია.

### IDENTITY

SubjectId INT IDENTITY (1,1)

ავტომატური გადანომრვა, ბაზაში მონაცემების განლაგებისა და შემგომ მათი სწრაფი ძებნის მეთოდებისთვის საჭიროა რომ ყოველ ჩანაწერს ქონდეს თავისი უნიკალური ნომერი (გამონაკლისების გარდა). თუმცა ამ უნიკალური ნომრის ყოველთვის ხელით მინიჭება მოუხერხებული და ზოგჯერ შეუძლებელიცაა, მაგალითად ბაზაში ემატება სტუდენტი და მას ენიჭება უნიკალური ნომერი, და რა ნომერი უნდა იყოს ეს? სავარაუდოდ ბოლოს+1, გამოდის უნდა მოვიძიოთ ბოლოს დამატებული სტუდენტი, გამოვთვალით აზალი სტუდენტის უნიკალური ნომერი და მივანიჭოთ მას? რა თქმა უნდა არა! სვეტის შეზღუდვა IDENTITY ავტომატურად ნომრავს სტრიქონებს, განსაღვრული ფუძით და ბიჯით. გაჩუმების პრინციპით ფუძეც და ბიჯიც 1-ის ტოლია IDENTITY(1,1) და ვიღებთ სვეტების ნუმერაციას 1,2,3,4..... თუმცადა თუ გვინდა 2000 წლიდან ყოველი 5 წლის ბიჯით წლების მინიჭება გვექნება IDENTITY(5,2000)შეზღუდვის სინტაქსი ფუძით 2000 და ბიჯით 5. ასევე საყურადღებოა რომ IDENTITY შეზღუდვის მქონე სვეტები არ ივსება მომხმარებლის მიერ და არ მონაწილეობს შევსების ოპერაციებში.

### CHECK

Email VARCHAR (30) CHECK (Email LIKE '%@%')

„შემოწმება“ -ეს არის შეზღუდვა რომელიც ამოწმებს სვეტს ნებისმიერი ლოგიკური პირობის ჭეშმარიტობაზე, მაგალითად რომ მნიშვნელობა დადებითია, ან რაიმე დიაპაზონშია მოქცეული. რომ თარიღი უკვე გასული არაა, პირადი ნომრის სიგრძე ზუსტად 11 სიმბოლოა, ან მეილის სვეტი აუცილებლად შეიცავს '@'-ს სიმბოლოს

### UNIQUE

PersonalId VARCHAR(11) UNIQUE

უნიკალურობის შეზღუდვა, როდესაც საჭიროა რომ სვეტში არ მეორდებოდეს მონაცემები, და თუ პირადი ნომრის სვეტში ერთ მომხმარებელს უკვე მინიჭებული აქვს 01001001001 პირადი ნომერი, მეორე მომხმარებელთან იგივე პირადი ნომერის ჩაწერა ვერ მოხერხდეს.

## DEFAULT

City NVARCHAR(50) DEFAULT (N'თბილისი')

მნიშვნელობა გაჩუმების პრინციპით, თუკი სვეტში მნიშვნელობა არასავალდებულოა მაგრამ მაინც ვანიჭებთ მას გაჩუმების პრინციპით, მაშინ ჩვენ გვჭირდება შეზღუდვა DEFAULT. მაგალითად ვთქვათ საცხოვრებელი ქალაქი არ არის სავალდებულო ველი და მასში წინასწარ ამორჩეულია მნიშვნელობა 'თბილისი'. რაც ნიშნავს რომ შესაძლებელია მივუთითოთ სასურველი ქალაქი, მაგრამ თუ არ მივუთითებთ სისტემა ავტომატურად ჩაწერს მნიშვნელობას 'თბილისი'.

## PRIMARY KEY

ცხრილის გასაღები ველი, რომელიც უზრუნველყოფს ცხრილში უნიკალური არანულოვანი ინდექსის არსებობას, რითაც იძლევა ცალსახად ძებნის და ცხრილთან კავშირის შესაძლებლობას.

## FOREIGN KEY

ძირითად და გარე გასაღებებზე დეტალურად ვისაუბრებთ მომდევნო თემაში

შემდეგი CREATE TABLE ბრძანება ქმნის ცხრილს Directions

CREATE TABLE Directions

```
(  
DirectionId INT NOT NULL PRIMARY KEY IDENTITY,  
DirectoryName NVARCHAR(150) NOT NULL UNIQUE ,  
DirectionHead INT NULL  
)
```

GO

ზემოთ მოყვანილი SQL სკრიპტი შექმნის ახალ Directions ცხრილს ნაგულისხმევ სქემაში dbo და მონაცემთა ბაზაში, რომელიც მითითებულია შეკითხვის (Query) რედაქტორის მიერ SSMS-ში. როგორც უკვე აღვნიშნეთ შესაძლებელია მიუთითოთ ცხრილის სრული სახელი ფორმატში DatabaseName.SchemaName.TableName. თუ dbo სქემაში და Faculty ბაზაში შევქმნით იმავე ცხრილს, მივიღებთ სკრიპტს:

CREATE TABLE Faculty .dbo.Directions

```
(  
DirectionId INT NOT NULL PRIMARY KEY IDENTITY,  
DirectoryName NVARCHAR(150) NOT NULL UNIQUE ,  
DirectionHead INT NULL  
)
```

- ცხრილის სახელი შეიძლება იყოს მაქსიმუმ 128 სიმბოლო.
- სვეტების სახელები მითითებულია [ColumnName DataType Constraint] მძიმით გამოყოფილი ფორმატით. ჩვენ ვქმნით DirectionId, DirectionName და DirectionHead სვეტებს.

- სვეტის სახელის შემდეგ ეთითება მონაცემთა რომელ ტიპს შეინახავს სვეტი, ჩვენს შემთვევაში გამოყენებული იქნება **INT**, **NVARCHAR** და **INT** ტიპები.
- NOT NULLმიუთითებს, რომ სვეტი არ შეიძლება იყოს ცარიელი. თუ აღნიშნული შეზღუდვა არ იქნება მითითებული ნაგულისხმევად, სვეტი დაუშვებს null მნიშვნელობას.
- IDENTITY განსაზღვრავს მთელი რიცხვების სვეტს ავტომატურად გენერირებული ნუმერაციით. გაჩუმების პრინციპით ნაგულისმებია რომ მნიშვნელობა დაიწყება 1-დან და გაიზრდება 1-ით ყოველი ახალი სტრიქონისთვის.
- PRIMARY KEY

შემდეგი სტკიპრტი წაშლის Directions ცხრილს, თუ უკვე არსებობს და შემდეგ ხელახლა ქმნის მას.

```
USE Faculty
```

```
GO
```

```
DROP TABLE IF EXISTS Faculty.dbo.Directions; --drop table if already exists
```

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

ასევე შეზღუდვის შექმნა შესაძლებელია დამხმარე სიტყვა “CONSTRAINT” გამოყენებით. თუ შეზღუდვას დავწერთ სრული ფორმატით მაშინ დაგვჭირდება სიტყვა CONSTRAINT და შეზღუდვის სახელი. მაგალითად პირველ სვეტზე არსებული PRIMARY KEY შეზღუდვის დადება შესაძლებელია ისე რომ ჩვენ თავად დავარქვათ შეზღუდვას სახელი და არ მოხდეს მისი დაგენერირება სისტემის მიერ:

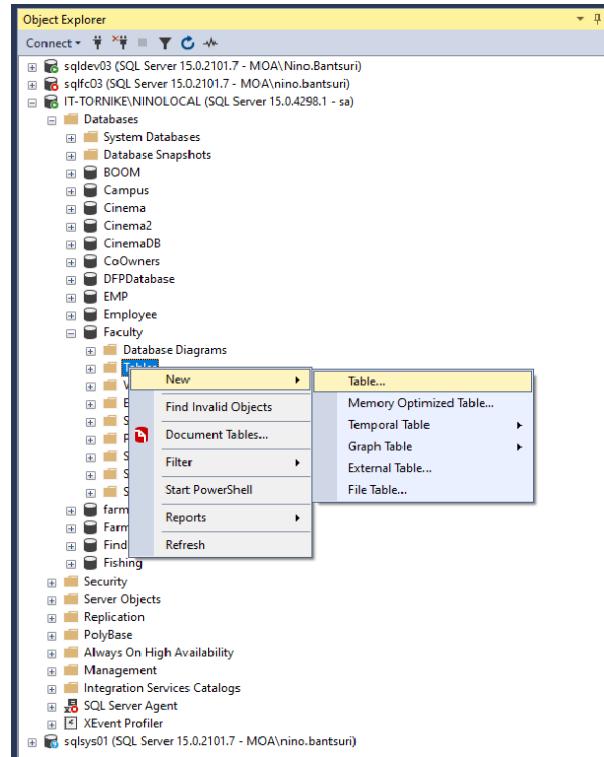
```
DirectionId INT NOT NULL CONSTRAINT PK_Directions PRIMARY KEY
```

ამ შემთვევაში ჩვენ გვეცოდინება რომ გასაღებ შეზღუდვას ჰქვია PK\_Directions, და შედარებით გავიმარტივდება მასზე ოპერაციები საქიროების შემთხვევაში. და ბოლოს სრული ფორმით შეზღუდვის დაწერა შეიძლება სვეტების ჩამონათვლის ბოლოს, უბრალოდ გასათვალისწინებელია, რომ შეზღუდვის სვეტისგან დამოკიდებლად დაწერისას უნდა მივუთითოთ რომელ სვეტს/სვეტებს ედება იგი:

```
CREATE TABLE Directions
(DirectionId INT NOT NULL IDENTITY,
DirectoryName NVARCHAR(150) NOT NULL UNIQUE ,
DirectionHead int NULL,
CONSTRAINT PK_Directions PRIMARY KEY (DirectionId)
)
```

## შევქმნათ ცხრილი SSMS -ის გამოყენებით

- ახალი ცხრილის შესაქმნელად, გახსენით SSMS და დაუკავშირდით თქვენს sql სერვერს.
- Object Explorer-ში ჩამოშალეთ Faculty მონაცემთა ბაზა (ან ჯერ შექმნით Faculty ბაზა)
- ახლა, დააწყვეტით tables საქაღალდეზე მარჯვენა ღილაკით და აირჩიეთ New Table, როგორც ეს ნაჩვენებია ქვემოთ:



შედეგად გაიხსნება ცხრილის დიალის რეჟიმი, სადაც შეგიძლიათ შეიყვანოთ სვეტის სახელი, მისი მონაცემთა ტიპი და ასევე მიუთითოთ იძლევა თუ არა სვეტი null-ებით შევსების უფლებას.

The screenshot shows the SQL Server Object Explorer with a table named 'dbo.Table\_1\*' selected. In the center, there's a grid for defining columns. The first column is named 'DirectionId'. The 'Data Type' dropdown menu is open, showing options such as 'int', 'money', 'nchar(10)', 'ntext', 'numeric(18, 0)', 'nvarchar(50)', 'nvarchar(MAX)', and 'real'. The 'Allow Nulls' checkbox is also visible for this column.

- Column Name: ჩაწერეთ სვეტის უნიკალური სახელი.
- Data Type: აირჩიეთ მონაცემთა ტიპი სვეტისთვის ჩამოსაშლელი სიიდან. აირჩიეთ შესაბამისი სიგრძე სტრიქონის მონაცემთა ტიპებისთვის.
- Allow Nulls: აირჩიეთ, დაუშვათ თუ არა Nulls თითოეული სვეტისთვის.

შეიყვანეთ სვეტების ინფორმაცია ყველა იმ სვეტისთვის, რომელიც გსურთ გქოდეთ თქვენს ცხრილში. ქვემოთ მოცემულია Directions ცხრილის სვეტები.

სვეტის დამატებითი პარამეტრებისა და შეზღუდვების დაყენება, როგორიცაა გამოთვლადი სვეტის პარამეტრები და აიდენტიტეტი შეზღდვა შესაძლებელია Column Properties ფანჯრიდან. ამისათვის აირჩიეთ სვეტი და გადაგისტ Column Properties პანელში.

პირველადი გასაღების შეზღუდვის მოსანიშნად დაწყაპუნეთ მაუსის მარჯვენა ღილაკით სვეტზე და აირჩიეთ Set Primary Key:

თქვენ შეგიძლიათ დაკონფიგურიროთ პირველადი გასაღები, ავტომატურად გადანომვრად (IDENTITY) სვეტად Column Properties ფანჯრიდან.

IT-TORNIKE\NINOLO..y - dbo.Directions\* SQLQuery83.sql - IT...AL.Faculty (sa (53))\*

Column Name	Data Type	Allow Nulls
DirectionId	int	<input type="checkbox"/>
DirectoryName	nvarchar(150)	<input type="checkbox"/>
DirectionHead	int	<input checked="" type="checkbox"/>
		<input type="checkbox"/>

Column Properties

DTS-published No  
Full-text Specification No  
Has Non-SQL Server Subscriber No  
Identity Specification Yes  
  (Is Identity) Yes  
    Identity Increment 1  
    Identity Seed 1  
Indexable Yes  
Is Columnset No  
Is Sparse No  
Merge-published No

(Is Identity)

ნაგულისხმევად, ცხრილი იქმნება dbo სქემაში. ცხრილისთვის განსხვავებული სქემის  
მისანიშებლად დააწყაპუნეთ მარჯვენა ღილაკით Table-Designer პანელზე და აირჩიეთ Properties  
ჩანართი, სქემის (Schema) ჩამოსაშლელი სიიდან აირჩიეთ შესაბამისი სქემა.

IT-TORNIKE\NINOLO..y - dbo.Directions\* SQLQuery83.sql - IT...AL.Faculty (sa (53))\*

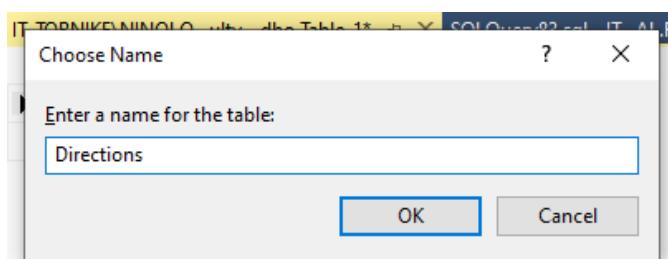
Column Name	Data Type	Allow Nulls
DirectionId	int	<input type="checkbox"/>
DirectoryName	nvarchar(150)	<input type="checkbox"/>
DirectionHead	int	<input checked="" type="checkbox"/>
		<input type="checkbox"/>

Properties

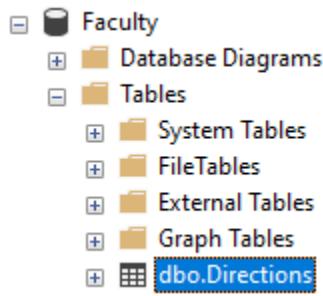
[Tbl] dbo.Directions

(Identity)  
  (Name) Directions  
  Database Name Faculty  
  Description  
  Schema dbo  
  Server Name  
  Table Designer  
    Identity Column  
    Indexable  
    Lock Escalation  
  Regular Data Space Specification  
    Replicated  
    Row GUID Column  
    Text/Image Filegroup

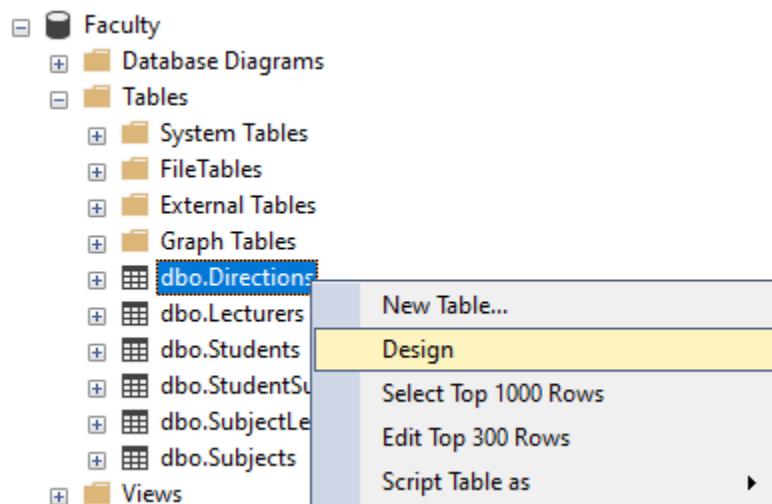
ახლა ფაილის მენიუდან აირჩიეთ შენახვა ამ ცხრილის შესაქმნელად, ან დააჭირეთ (Ctrl+S)  
კომბინაციას. შეიყვანეთ Directions როგორც ცხრილის სახელი და დააწყაპუნეთ OK



ახალი ცხრილის სანახავად განაახლეთ Tables საქაღალდე Object Explorer-ში. ცხრილი Directions ახლა ხელმისაწვდომია ცხრილების ჩანართში.



ამრიგად, თქვენ შექმნათ ახალი ცხრილი ცხრილის დიზაინ რეკიმის გამოყენებით SSMS-ში, აღნიშნული რეკიმის გახსნა უკვე არსებული ცხრილისთვის შესაძლებელია თუ ცხრილზე დავაჭროთ მარჯვენა ღილაკს და კონტრილური მენიუდან ამოვარჩევთ Design ჩანართს



რის შემდეგაც კვლავ მოვხვდებით ნაცნობ ფანჯარაში სადაც შევძლებთ საჭირო შესწორებების შეტანას და შენახვას ცხრილში. მაგალითად განვიხილოთ კიდევ ერთი სვეტის ჩამატება არსებულ სვეტში, ჯერ აღნიშული მოქმედება შევასრულოთ სკრიპტის საშუალებით შემდგომ დავუბრუნდეთ დიზაინს.

## ALTER TABLE ADD Columns სვეტის დამატება ცხრილში

თქვენ შეგიძლიათ დაამატოთ სვეტები არსებულ ცხრილს ALTER TABLE ბრძანების გამოყენებით.

ALTER TABLE ბრძება ასევე შეიძლება გამოყენებულ იქნას არსებული ცხრილის სვეტების სახელის გადასარქმევად ან წასაშლელად.

გამოყენეთ ALTER TABLE ADD სკრიპტი არსებული ცხრილის ერთი ან მეტი სვეტის დასამატებლად:

```
ALTER TABLE [schema_name.]table_name
ADD column_name1 data_type constraint,
column_name2 data_type constraint
...
column_nameN data_type constraint;
```

შემდეგი ამატებს ცხრილს Address სვეტს ტიპით varchar და ზომით 500.

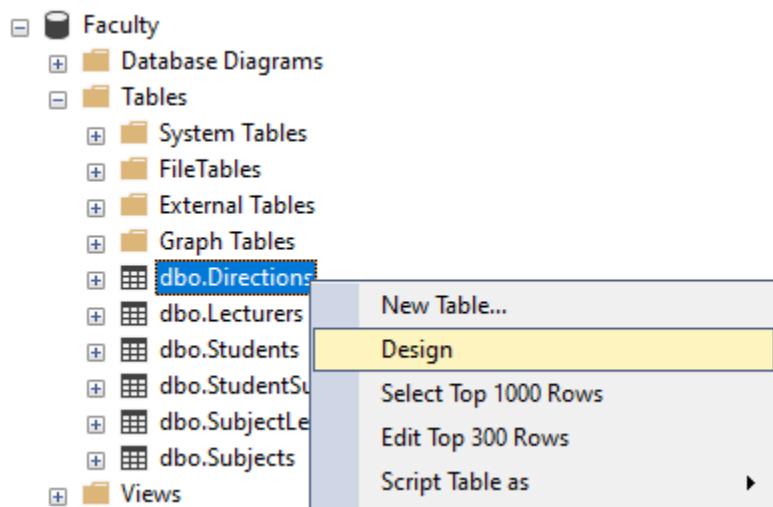
GO

```
ALTER TABLE [dbo].[Directions]
ADD Address VARCHAR(500) NOT NULL
```

მძიმით გამოყოფის შემთხვევაში შესაძლებელია რამოდენიმე სვეტის დამატება ერთი Alter ბრძანებით

```
ALTER TABLE [dbo].[Directions]
ADD Phone VARCHAR(30) NULL,
      Mail VARCHAR(30) NULL
```

აღნიშნული მოქმედების დიზაინში გასამორჩდად ცხრილზე დავაჭიროთ მარჯვენა ღილაკს და კონტექსტური მენიუდან ამოვარჩევთ Design ჩანართი:



გაიხსნება ცხრილის რედაქტირები რეჟიმი:

დააწყაპუნეთ პირველ ცარიელ უჯრედზე ბოლო სვეტის სახელის სვეტის ქვეშ და შეიყვანეთ სვეტის სახელი, როგორც ეს ნაჩვენებია ქვემოთ

Column Name	Data Type	Allow Nulls
DirectionId	int	<input type="checkbox"/>
DirectoryName	nvarchar(150)	<input type="checkbox"/>
DirectionHead	int	<input checked="" type="checkbox"/>
Address		<input type="checkbox"/>

შემდეგ სვეტში აირჩიეთ მონაცემთა ტიპი ჩამოსაშლელიდან და სიგრძე, საჭიროების შემთხვევაში. სავალდებულოდ შესავსებია თუ არა სვეტი (NULL/NOT NULL) და შეინახეთ File->Save Directions ან Ctrl+s კომბინაციით.

## სვეტის ან ცხრილის სახელის გადარქმევა

როგორც აქამდეც აღვნიშნეთ ობიექტების სახელები წარმოადგენენ მითითებლებს მატზე, შესაბამისად შეუძლებელია მათი სახელის ცვლილება alter ბრძანებით. ამისათვის უნდა გამოვიყენოთ სისტემის შენახული პროცედურა sp\_rename.

```
EXEC sp_rename 'old_name', 'new_name' [, 'object_type'];
```

ცხრილის სახელის გადარქმევა: ცხრილის სახელის გადარქმევისთვის, 'old\_name' უნდა იყოს მოცემული ცხრილის არსებული სახელი ან schema.table ფორმაში

სვეტის სახელის გადარქმევა: ცხრილში სვეტის სახელის გადარქმევის მიზნით, 'old\_name' უნდა იყოს მოცემული table.column ან schema.table.column ფორმაში.

ინდექსის გადარქმევა: ინდექსის სახელის გადარქმევის მიზნით, 'old\_name' უნდა იყოს მოცემული table.index ან schema.table.index ფორმაში.

შეზღუდვების გადარქმევა: შეზღუდვის სახელის გადარქმევისთვის, 'old\_name' უნდა იყოს მოცემული schema.constraint ფორმაში.

შემდეგი გადაარქმევს Directions ცხრილს ცხრილად TempDirections.

```
EXEC sp_rename 'Directions', 'TempDirections'
```

და პირიქით:

```
EXEC sp_rename 'TempDirections', 'Directions'
```

შემდეგი სკრიპტი გადაარქმევს 'Directions' ცხრილის Address სვეტს TempAddress.

```
EXEC sp_rename 'Directions.Address', 'TempAddress';
```

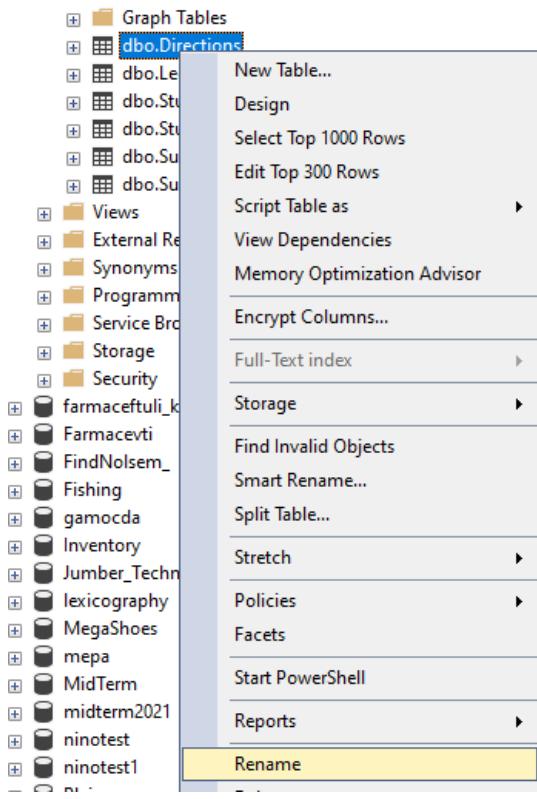
შევასრულოთ იგივე მოქმედება, დავაბრუნოთ საწყისი სახელი ამჯერად დავაკონკრეტოთ ობიექტის ტიპი:

```
EXEC sp_rename 'Directions.TempAddress', 'Address', 'COLUMN';
```

## ცხრილისა და სვეტების სახელის გადარქმევა SSMS-ის გამოყენებით:

გახსენით SSMS და ჩამოშალეთ მონაცემთა ბაზის საქაღალდე.

აირჩიეთ და დააწყაპუნეთ მარჯვენა ღილაკით ცხრილზე ან სვეტზე, რომლის გადარქმევაც გსურთ და დააწყაპუნეთ სახელის გადარქმევა(Rename). შეიყვანეთ ახალი სახელი არსებულ სახელზე გადაწერით და დააჭირეთ ENTER ღილაკს.



## ცხრილის სვეტების/სვეტის წაშლა

გამოიყენეთ ALTER TABLE DROP COLUMN ბრძანება T-SQL-ის გამოყენებით ცხრილის ერთი ან მეტი სვეტის წასაშლელად.

```
ALTER TABLE [schema_name.]table_name
DROP column column_name1, column_name2,... column_nameN;
```

შემდეგი სკრიპტი წაშლის Directions ცხრილის სვეტს Address.

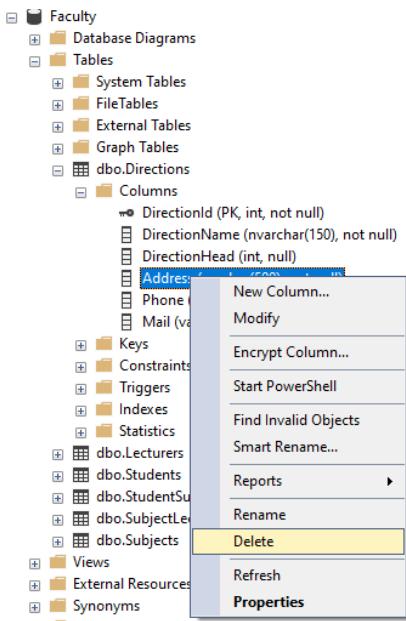
```
ALTER TABLE dbo.[Directions]
DROP COLUMN Address;
```

შემდეგი სკრიპტი წაშლის Directions ცხრილის რამოდენიმე სვეტს.

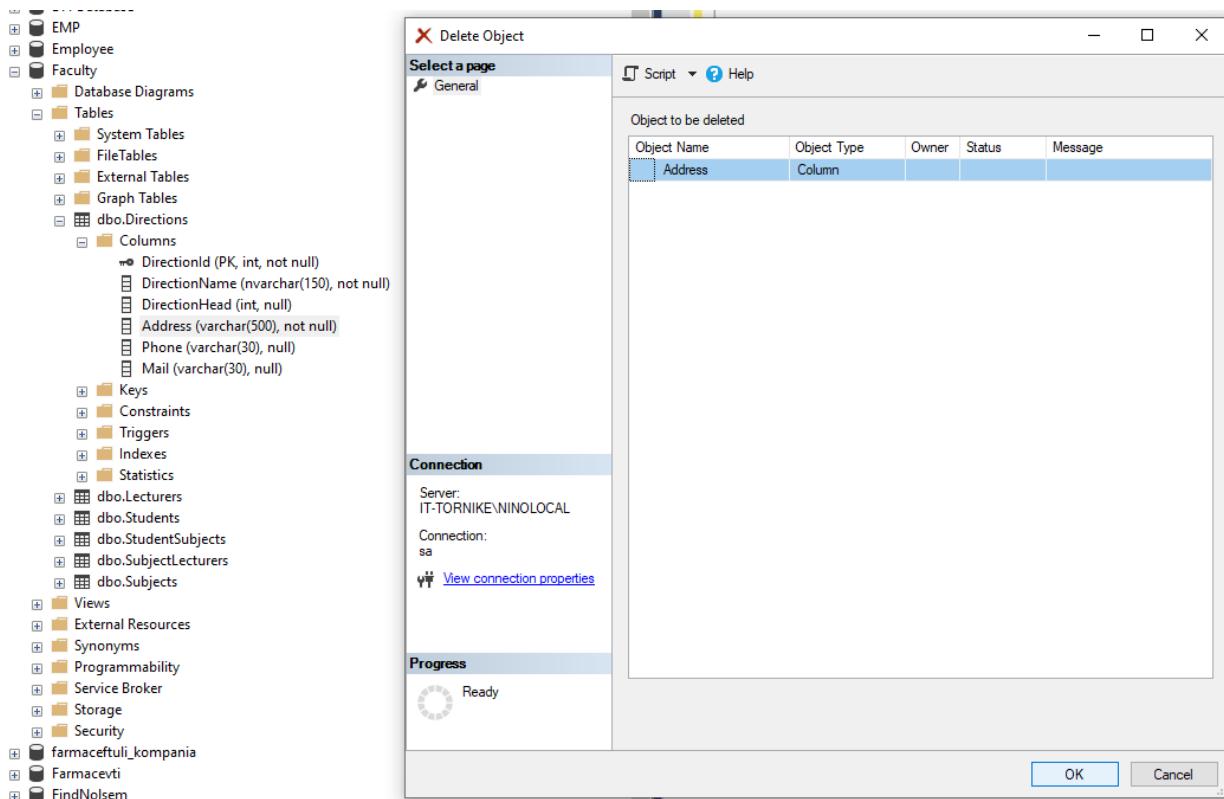
```
ALTER TABLE dbo.[Directions]
DROP COLUMN [Phone],[Mail];
```

## წაშლეთ სვეტები SSMS-ის გამოყენებით

დააწყაპუნეთ მაუსის მარჯვენა ღილაკით სვეტის სახელზე, რომლის წაშლაც გსურთ და კონტექსტური მენიუდან აარჩიეთ Delete ბრძანება:



გაიხსნება „Delete Object“ ფანჯარა, დავეთანხმოთ Okლილაკზე დაჭერით:



## ცხრილის სვეტების მოდიფიკაცია

გამოიყენეთ ALTER TABLE ALTER COLUMN ბრძანება T-SQL-ის გამოყენებით ცხრილის ერთი ან მეტი სვეტის მოდიფიკაციისათვის.

სკრიპტის საშუალებით DirectionName სვეტის ზომა გაიზრდება 250 სიმბოლომდე:

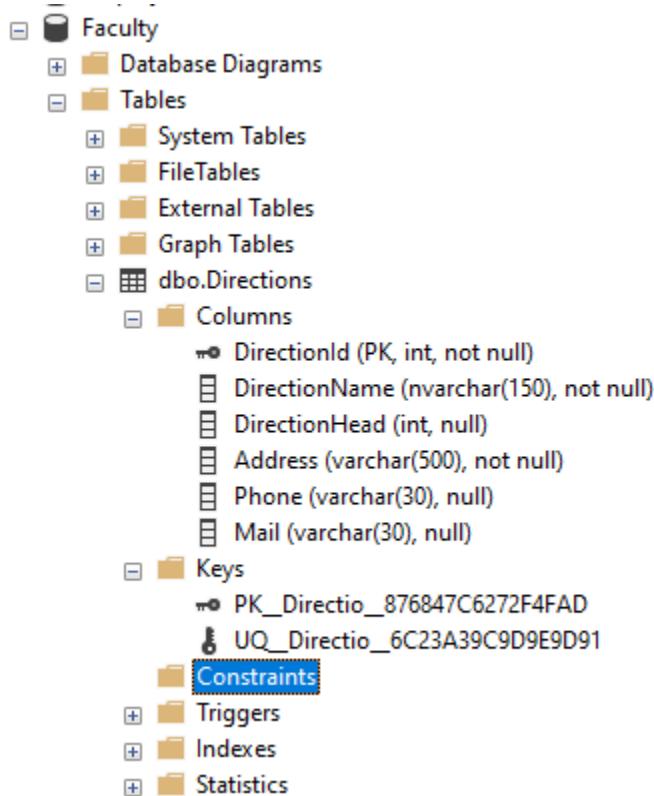
```
ALTER TABLE dbo.Directions  
    ALTER COLUMN DirectionName VARCHAR (250)
```

აღნიშნული ცვლილებების შეტანა შესაძლებელია ჩვენთვის უკვე ნაცნობი ცხრილის დიზაინის რეჟიმიდან. თუ ცხრილზე დავაჭროთ მარჯვენა ღილაკს და კონტექსტური მენიუდან ამოვარჩევთ Design ჩანართს.

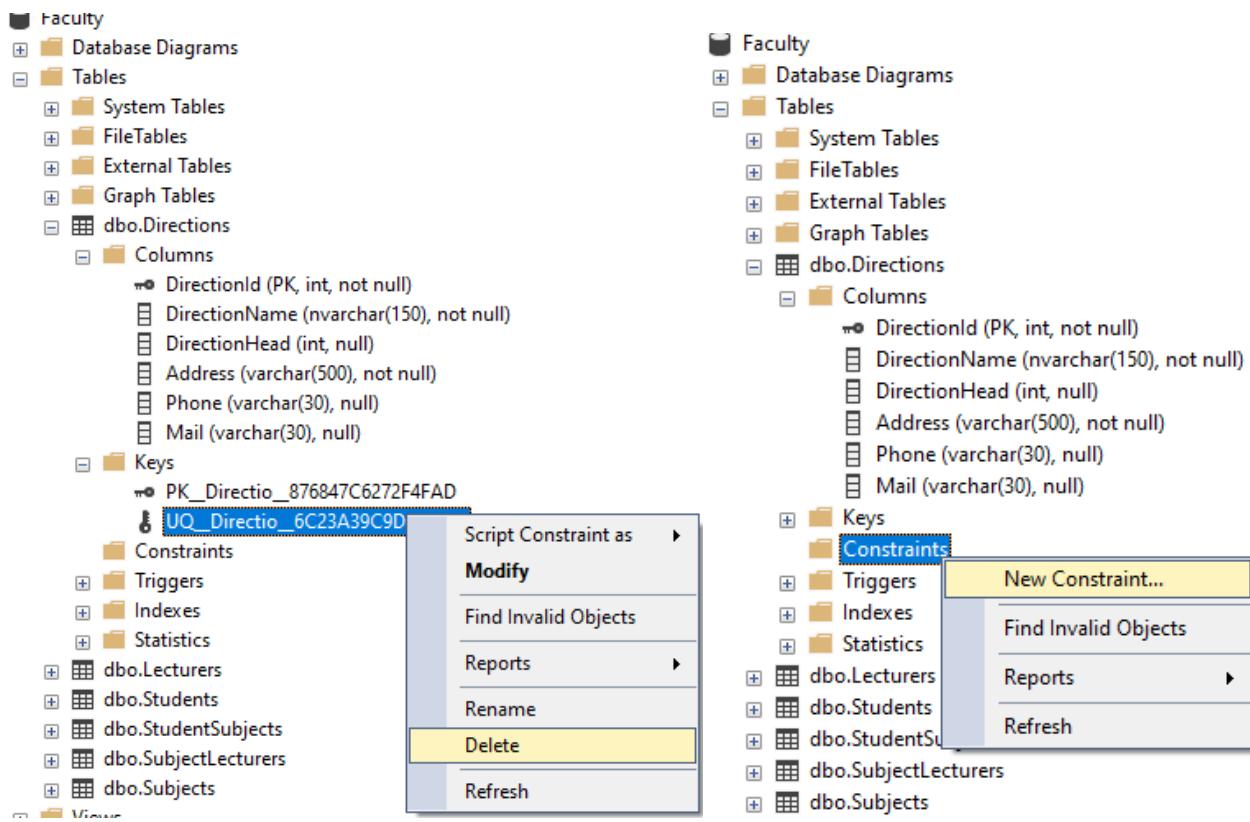
## მოქმედებები შეზღუდვებზე

შეზღუდვების დამატება შესაძლებელია CREATE ან ALTER TABLE T-SQL ბრძანებების საშუალებით. CREATE TABLE მაგალითები უკვე ვნახეთ აღნიშნული თემის დასაწყისში. Alter ბრძანებით შეზღუდვების დამატება/წაშლა შესაძლებელია შემდეგი სკრიპტების გამოყენებით. აღსანიშნავია რომ მნიშვნელოვანია წაშლის დროს ვიცოდეთ შეზღუდვის სახელი, თუ თავად არ დაგვითირებეთ შეზღუდვის სახელი უნდა მოვიძიოთ ისინი დიზაინ რეჟიმში:

ცხრილის Keys ან Constraint ჩანართში:



რის შემდეგაც შეგვიძლია დავაკოპიროთ დასახელება და წავშალოთ შეზღუდვა სკრიპტის გამოყენებით. თუმცა შეზღუდვის წაშლა/დამატება ასევე შესაძლებელია დიზაინ რეჟიმიდან:



დავამატოთ ახალი შეზღუდვა რომ მიმართულების ხელმძღვანელის ნომერი DirectionHead იყოს უნიკალური, ანუ ერთ ადამიანს მხოლოდ ერთი მიმართულების ხელმძღვანელობა შეეძლოს და ერთი და იგივე პიროვნება არ დაინიშნოს რამოდენიმე მიმართულების ხელმძღვანელად:

```
ALTER TABLE [dbo].[Directions]
ADD CONSTRAINT UQ_DirectionHead UNIQUE([DirectionHead])
```

მას მერე რაც ჩვენს მიერ შერჩეული სახელით დავამატეთ შეზღუდვა UQ\_DirectionHead. შესაძლებელია მისი წაშლა სწორედ ამ სახელის გამოყენებით:

```
ALTER TABLE [dbo].[Directions]
DROP CONSTRAINT UQ_DirectionHead
GO
```

## დამატებით ცხრილების შექმნა

მოგროვილი ცოდნის საშუალებით დავამატოთ რამოდენიმე ცხრილი შესაბამისი შეზღუდვებით:

```
CREATE TABLE Subjects
(
SubjectId INT IDENTITY (1,1) ,
SubjectName NVARCHAR(150) NOT NULL,
```

```

DirectionId INT NOT NULL,
Credit INT,
CONSTRAINT PK_Subjects PRIMARY KEY (SubjectId)
)

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 '%@%')
)

CREATE TABLE Students
(StudentId INT IDENTITY (1,1000) CONSTRAINT PK_Students PRIMARY KEY,
StudentName nvarchar(30) NOT NULL,
StudentLastName nvarchar(30) NOT NULL,
PersonalId VARCHAR(11) UNIQUE,
DateOfBirth DATE,
Phone VARCHAR (20),
Email VARCHAR (30),
City NVARCHAR(50) DEFAULT (N'თბილისი'),
DirectionId INT NOT NULL,
CONSTRAINT CK_Email CHECK (Email LIKE '%@%')
)

CREATE TABLE SubjectLecturers
(SubjectLecturerId INT IDENTITY PRIMARY KEY,
SubjectId INT NOT NULL,
LecturerId INT NOT NULL
CONSTRAINT UQ_SubjectLecturers UNIQUE(SubjectId,LecturerId)
)

CREATE TABLE StudentSubjects
(StudentSubjectId INT IDENTITY PRIMARY KEY,
StudentId INT NOT NULL,
SubjectId INT NOT NULL,
RegisterDate DATETIME NOT NULL DEFAULT (GETDATE()),
Midterm INT NULL,
Exam INT NULL,
Quiz INT NULL,
Project INT NULL,
IsPassed BIT NOT NULL DEFAULT(0)
)

```



## ლაბორატორია 4

### ცხრილების კავშირები SQL სერვერზე:

რელაციური მონაცემთა ბაზის სწორად დასაპროქტებლად 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 პირველადი გასაღები აერთიანებს Unique და Not null შეზღუდვებს, თუმცა საბოლოოდ მას ზევრად 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
- მეოლი
- 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 იჯდეს 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 (Foreign 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?

- დამოკიდებულ ცხრილში ვერ დაემატება ისეთი ჩანაწერი ( ჩვენს შემთხვევაში StudentId) რომელიც არ არსებობს ძირითად ცხრილში (ვერც დამატების და ვერ განახლების დროს)
- 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 ერთი-ერთან
- 1:N ერთი-ბევრთან

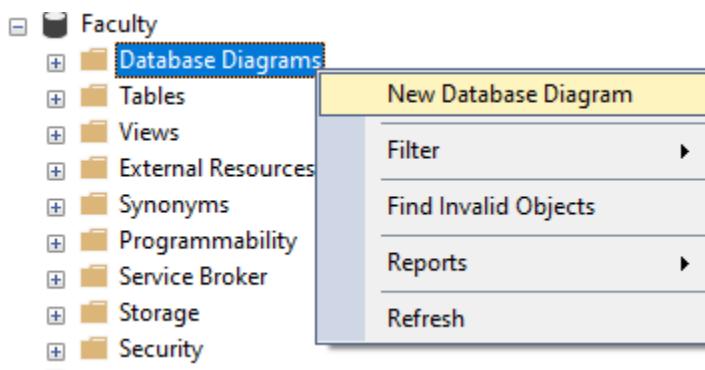
### 3. N:M ბევრი-ბევრთან

#### ერთი-ერთთან კავშირი

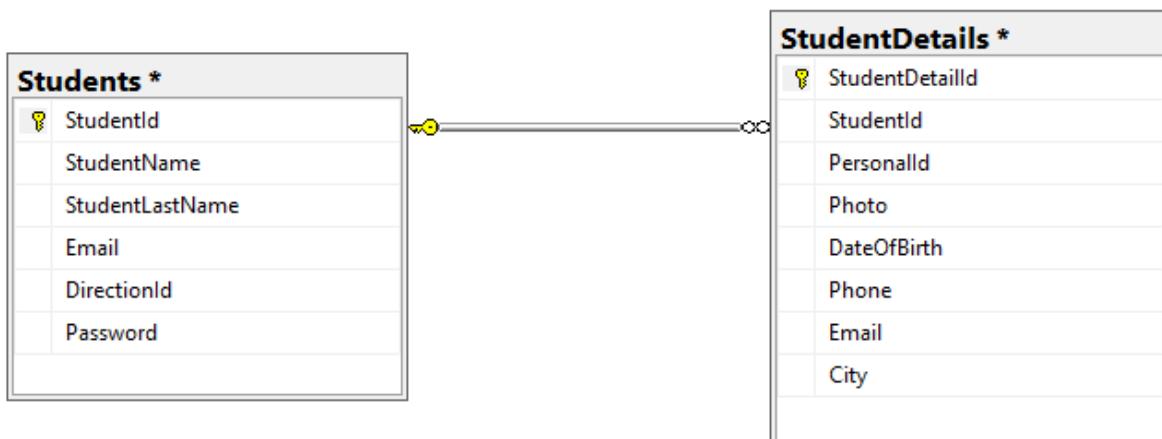
ერთი-ერთთან კავშირის დროს ერთი ცხრილის ერთი ჩანაწერი იქნება დაკავშირებული სხვა ცხრილის ერთ ან არცერთ ჩანაწერთან. მაგალითად, Students ცხრილის თითოეულ ჩანაწერს შეიძლება ჰქონდეს StudentDetails ცხრილში შესაბამისი სტრიქონი, რომელიც ინახავს ამ კონკრეტული სტუდენტის შესახებ დამატებით ინფორმაციას ასე რომ, თითოეულ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

\*კავშირების ვიზუალურად უკეთ აღსაქმნელად შეგიძლიათ გამოიყენოთ ბაზის დიაგრამა:

ჩამოშალეთ Faculty ბაზა, ჩანართ Database diagrams დაჭირეთ 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:

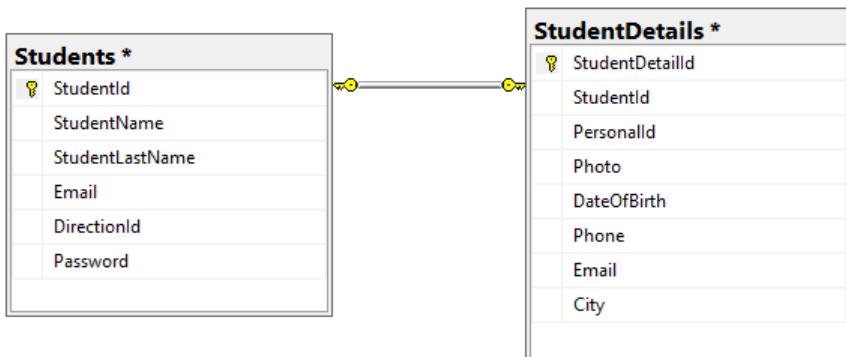


ზემოთ, StudentId სვეტი არის პირველადი (Primary) გასაღები ცხრილში Students, ხოლო StudentDetails ცხრილში წარმოადგენს გარე (Foreign) გასაღების სვეტს. მისი საშუალებით StudentDetails ცხრილი დაკავშირებულია Students ცხრილის შესაბამის ჩანაწერთან.

თუმცადა 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 შესაბამებოდეს მაქსიმუმ 1 ჩანაწერი კავშირის მეორე ცხრილიდან, საჭიროა სტუდენტი 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 მიმმული 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 ძალისხმეულის 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.

თუმცადა 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:

- 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.
- რომელია დამოკიდებული ცხრილი (რომელშიც იქნება ძირითად ცხრილზე  
დამოკიდებული მონაცემი)
  - სტუდენტის ცხრილი. ვინაიდან სტუდენტამდე უნდა არსებობდეს  
მიმართულება.
- გვაქვს თუ არა ძირითად ცხრილში პირველადი გასაღები( ვართ თუ არა მზად  
კავშირისთვის)

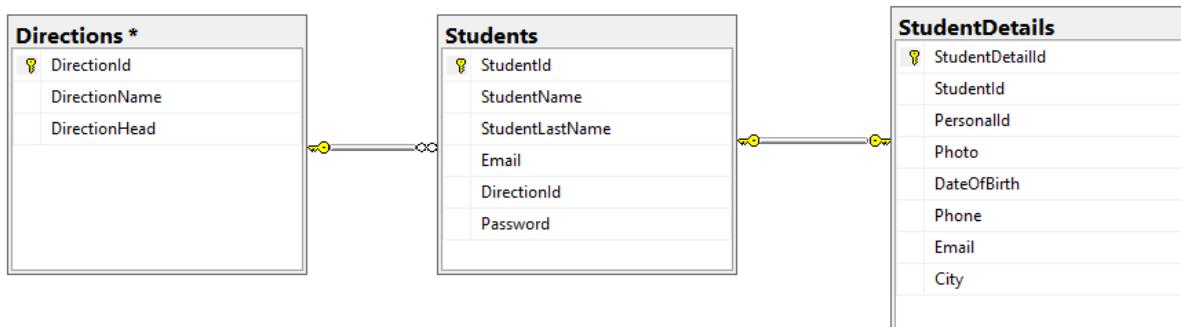
- დიახ, ძირითად, მიმართულების ცხრილს აქვს `DirectionId INT PRIMARY KEY`
4. გვაქვს თუ არა შესაბამისი კავშირის ველი. წინააღმდეგ შემთვევაში უნდა შეიქმნას დამოკიდებულ ცხრილში „ველი კავშირისთვის“ რომელზეც დაედება გარე გასაღები
- სტუდენტის ცხრილში ასევე გვაქვს კავშირის ველი `DirectionId INT` რომელზეც მოხდება 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 მიმართულების ცხრილიდან, ხოლო ერთი მიმართულება შესაძლებელია არჩეული ჰქონდეს **ბევრ** სტუდენტს.

შედეგად მივიღეთ ერთი ბევრთან კავშირი, რომელიც ასევე დავამატეთ დიაგრამაზე:



დავამატოთ საგნების ცხრილი:

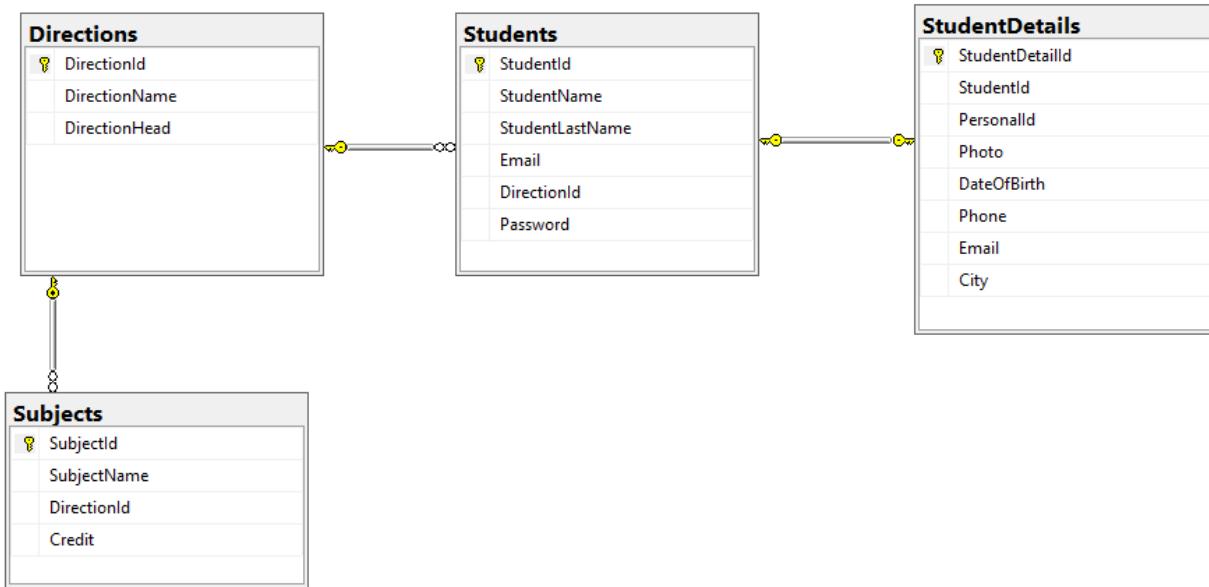
საჭირო ველები იქნება:

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 განვსაზღვროთ 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

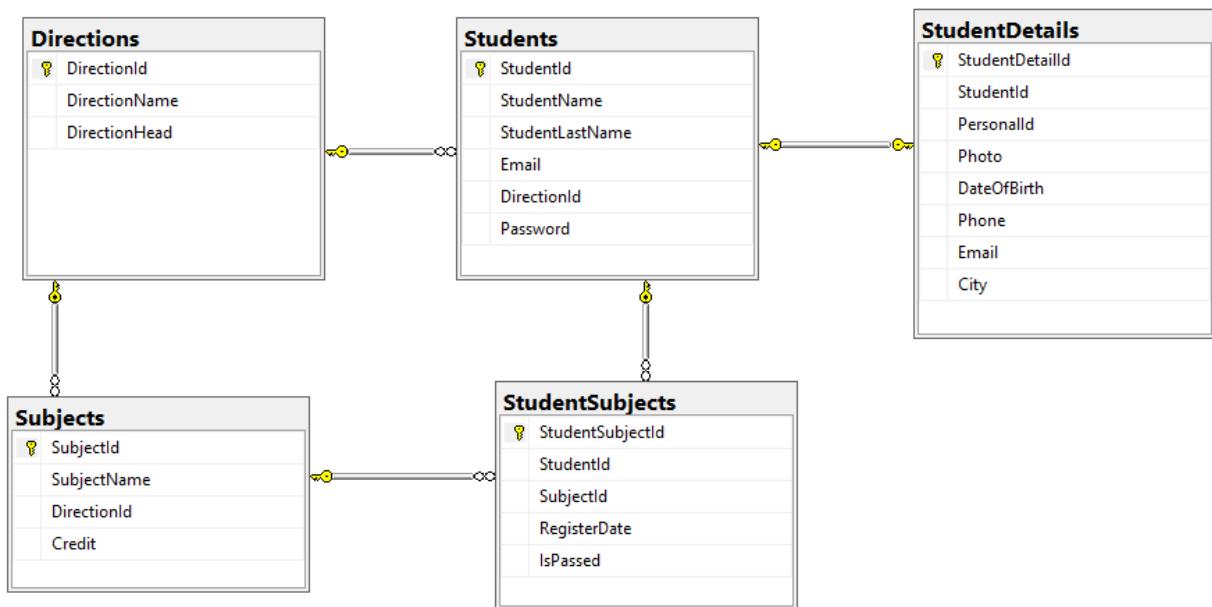
მარტივად რომ ვთქვათ ცხრილი რომელიც შეინახავს სტუდენტებისა და საგნების წყვილების ნომრებს:

სჭირო ველები იქნება:

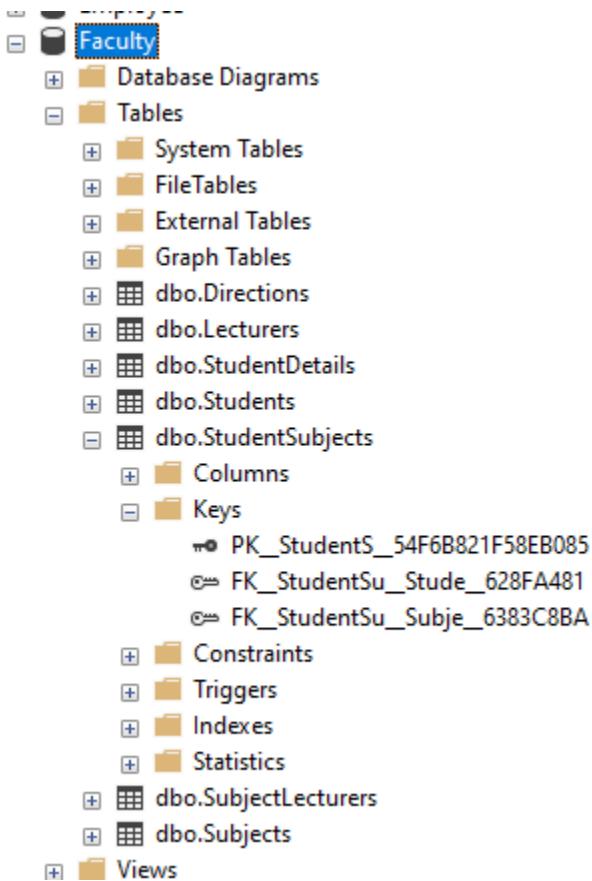
- საკუთარი უნიკალური საძიებო გასაღები ველი(სასურველია)
- სტუდენტის აიდი, რომელიც დაკავშირებულია სტუდენტის ცხრილთან, რათა არ მოხდეს არარსებული სტუდენტის ნომრის ჩაწერა.
- საგნის აიდი, რომელიც დაკავშირებულია საგნის ცხრილთან, რათა არ მოხდეს არარსებული საგნის ნომრის ჩაწერა.
- თუ კი კავშირს აქვს ატრიბუტები, 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 განვისაზღვრავს აღნიშნული კავშირების სახელები, აუცილებლად დაგვჭირდება სისტემის მიერ დაგნერირებული 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** როგორც მოიქცეს დამოკიდეული ცხრილი თუ ძირითად ცხრილში ველის ნომერი ახლდება?
- **ON DELETE** როგორც 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 პარამეტრები 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.

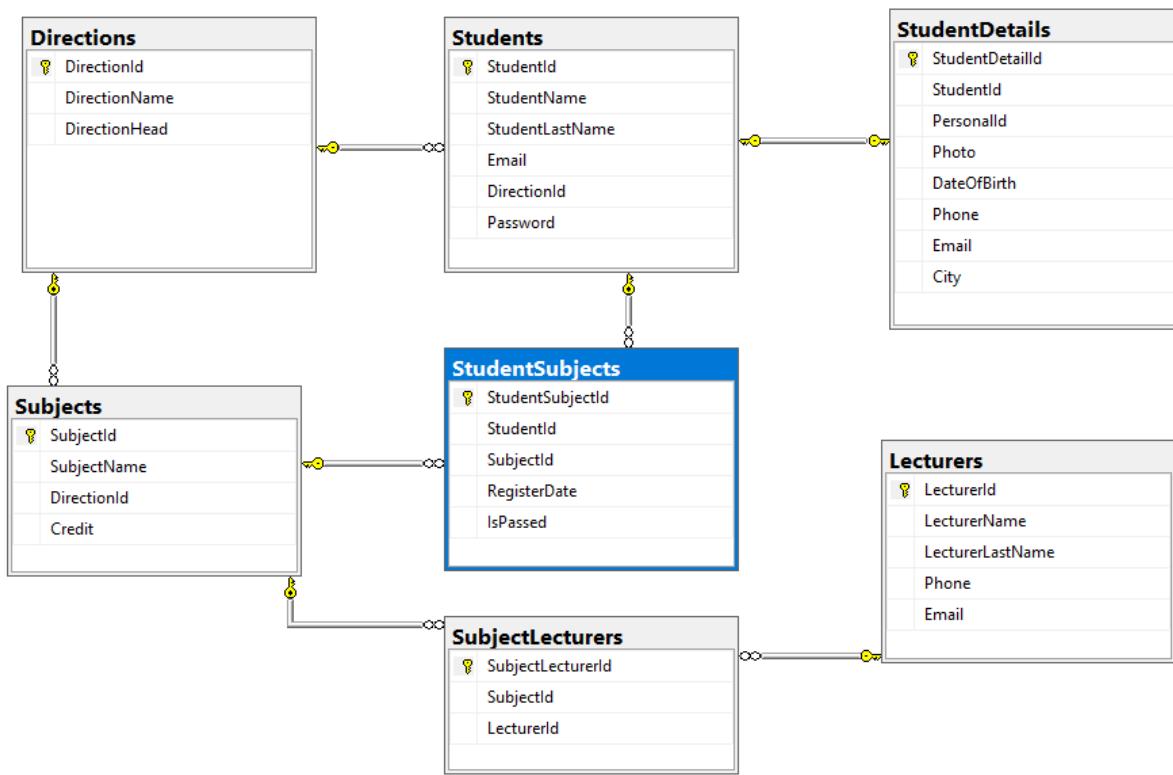
დავამატოთ ლექტორის ცხრილი:

```
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:





## ლაბორატორია 5

მონაცემების შეტანა ცხრილებში INSERT ბრძანების გამოყენებით  
ბრძანება INSERT INTO გამოიყენება SQL Server-ის მონაცემთა ბაზის ცხრილში ერთი  
ან რამდენიმე სტრიქონის ჩასაწერად.  
სინტაქსი:

```
INSERT INTO table_name (column_name1, column_name2...)
VALUES (column1_value, column2_value...);
```

მოდი ჩავსვათ მონაცემები ცხრილში Directions, რომელიც შევქმნით წინა  
ლაბორატორიაზე.

```
□ dbo.Directions
  □ Columns
    • DirectionId (PK, int, not null)
    □ DirectionName (nvarchar(150), not null)
    □ DirectionHead (int, null)
```

(საჭიროების შემთხვევაში გამოვიყენოთ ჩვენს მიერ უკვე შემქნილი ბაზის სკრიპტი,  
რომელიც მოცემულია თავის ბოლოს)

შემდეგი INSERT INTO ბრძანება ჩასვამს ერთ რიგს ზემოთ მოყვანილი ცხრილის  
ყოველ სვეტში:

```
INSERT INTO [dbo].[Directions]
(
    [DirectionName], [DirectionHead]
)
VALUES
(
    N'კომპიუტერული მეცნიერება', NULL
)
```

ჩასმის ბრძანების ტანი გამოიყურება შემდეგნაირად:

1. **INSERT INTO** საბრძანებო სიტყვა რომესაც მოსდევს ცხრილის დასახელება  
რომელშიც შედის მონაცემი. ჩვენს შემთხვევაში **[dbo].[Directions]**
2. მრგვალ ფრჩხილებში მოთავსებული იმ სვეტების ჩამონათვალი რომლებიც  
უნდა შეისვლო შესაბამისი მიმდევრობით. ჩვენს შემთხვევაში DirectionId სვეტი  
არის აიდენტითი ავტოგადანომვრადი სვეტი, რომლის მონაცემიც  
გენერირდება ავტომატურად და შესაბამისად იგი არ მონაწილეობს **INSERT**

ოპერაციაში. ივსება მხოლოდ DirectionName და DirectionHead სვეტები, თუმცა ვინდაინდან DirectionHead-ის ნომერი ამ ეტაპზე რჩება ცარიელი და უნდა მივანიჭოთ მოგვიანებით, შესძლებელია მისი გამოტოვებაც.

3. **VALUES** საბრძანებო სიტყვა და კვლავ მრგვალ ფრჩხილებში მოთავსებული მნიშვნელობები ზემოთ ჩამოთვლილი სტრიქონებისთვის, ჩვენს შემთხვევაში როგორც უკვე ვთქვით ივსება DirectionName და DirectionHead სვეტები და მრგვალ ფრჩხილებში უნდა გადავცეთ მნიშვნელობები იმავე თანმიმდევრობით. (**N'კომპიუტერული მეცნიერება',NULL**)

მნიშვნელოვანია რომ სვეტების რაოდენობა და მნიშვნელობების რაოდენობა ასევე ტიპი და თანიმდევრობა იყოს იდენტური, თუ ვავსებთ ორ სვეტს და პირველი არის სტრიქონი ხოლო მეორე მთელი რიცხვი საჭიროა პარამეტრიც გადავცეთ ორი პირველი სტრიქონი და მეორე მთელი რიცხვი.

შეუძლებელია შევსების დროს გამოვტოვოთ სავალდებულოდ შესავსები სვეტები თუ ისინი არ ივსება ავტოგადანომვრით ან გაჩუმების პრინციპით.

ჩასმული მონაცემების სანახავად შეასრულეთ Select \* from Directions; მოთხოვნა შეკითხვის რედაქტორში, როგორც ეს ნაჩვენებია ქვემოთ.

	DirectionId	DirectionName	DirectionHead
1	1	კომპიუტერული მეცნიერება	NULL

როგორც ხედავთ მოხდა Directions ცხრილში პირველი სტრიქონის შეტანა.

შესაძლებელი იყო ზემოთ მოცემულ კოდში გამოგვეტოვებინა სვეტი რომელიც არ იყო სავალდებულოდ შესავსები, მასში ავტომატურად ჩაიწერებოდა NULL ცარიელი მნიშვნელობა:

```
INSERT INTO [dbo].[Directions]
(
    [DirectoryName]
)
VALUES
(
    N'კომპიუტერული მეცნიერება'
```

)  
თუ **VALUES** საბრძანებო სიტყვის შემდგომ მრგვალი მნიშვნელობების შემცველი მრგვალი ფრჩხილების რამოდენიმე ბლოკს ჩავსვავთ და ერთმანეთისგან მძიმით გამოვყოფთ მოხდება ერთი ბრძანებით რამოდენიმე ჩანაწერის შეტანა:

```
INSERT INTO [dbo].[Directions]
(
    [DirectionName]
)
VALUES
(N'ინფორმაციული სისტემები'),
(N'ინფორმაციული ტექნოლოგიები')
```

თუ ცხრილის ყველა სვეტი ივსება, (გარდა IDENTITY-სა) შესაძლებელია **INSERT INTO** სიტყვის შემდეგ არ მივუთითოთ შესავსები სვეტების ჩამონათვალი. ასეთ შემთხვევაში ვალდებულები ვართ შევსების დროს დავიცვათ მონაცემთა ცხრილში არსებული მიმდევრობა:

```
INSERT INTO [dbo].[Directions]
VALUES
(
N'კომპიუტერული მეცნიერება', NULL
)
```

შეგვიძლია კვლავ დავათვალიეროთ ცხრილი Select \* from Directions ბრძანებით.

The screenshot shows the SQL Server Management Studio interface. In the top bar, there is a button with a yellow exclamation mark icon. Below the top bar, the text "Select \* from Directions" is typed into the command line. To the left of the command line, there is a zoom control set to "137 %". Below the command line, there are two tabs: "Results" and "Messages". The "Results" tab is selected and displays a table with four rows of data. The table has four columns: "DirectionId", "DirectionName", "DirectionHead", and "DirectionHead" (repeated). The data is as follows:

DirectionId	DirectionName	DirectionHead	DirectionHead
1	კომპიუტერული მეცნიერება	NULL	NULL
2	ინფორმაციული სისტემები	NULL	NULL
3	ინფორმაციული ტექნოლოგიები	NULL	NULL
4	კომპიუტერული მეცნიერება	NULL	NULL

ბაზაში დაგვემატა მიმართულებები, თავისი შესაბამისი რიგითი ნომრით (აიდი ველით) რომელიც შეგვიძლია გამოვიყენოთ და შევავსოთ საგნების ცხრილი:

The screenshot shows the 'dbo.Subjects' table structure. It has four columns: SubjectId (PK, int, not null), SubjectName (nvarchar(150), not null), DirectionId (FK, int, not null), and Credit (int, null). The 'SubjectId' column is highlighted.

dbo.Subjects
Columns
SubjectId (PK, int, not null)
SubjectName (nvarchar(150), not null)
DirectionId (FK, int, not null)
Credit (int, null)
..

ვინაიდან ცხრილის სრულად შევსება ხდება და აიდი ველის გარდა არცერთი სვეტის გამოტოვება არ ხდება შესაძლებელია მრგვალ ფრჩხილებში არ მივუთითოთ სვეტების ჩამონათვალი, მაგრამ დავიცვათ საგნების ცხრილში არსებული სტვეტების მიმდევრობა და ჯერ შევავსოთ დასახელება, შემდეგ მიმართულების ნომერი (რომელსაც ვიღებთ წინა ცხრილიდან)და ბოლოს კრედიტების რაოდენობა:

```
insert into Subjects
values
(N'პროექტი',1, 10)
,(N'ალგორითმები',2,5)
,(N'ქსელები',3,5)
,(N'NLP',1,5)
,(N'დაპროგრამება',2,6)
,(N'მანქანური სწავლება',1,3)
,(N'ნეირონული ქსელები',3, 5)
```

შეგვიძლია კვლავ დავათვალიეროთ ცხრილი Select \* from Subjects ბრძანებით.

The screenshot shows the execution of the query 'select \* from Subjects'. The results are displayed in a table with columns: SubjectId, SubjectName, DirectionId, and Credit. The data is as follows:

	SubjectId	SubjectName	DirectionId	Credit
1	3	პროექტი	1	10
2	4	ალგორითმები	2	5
3	5	ქსელები	3	5
4	6	NLP	1	5
5	7	დაპროგრამება	2	6
6	8	მანქანური სწავლება	1	3
7	9	ნეირონული ქსელები	3	5

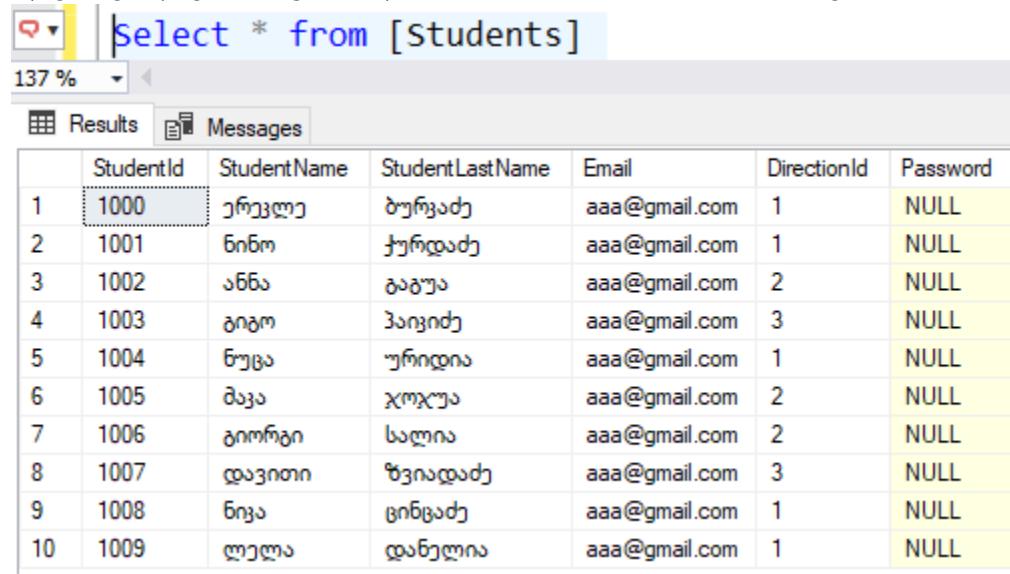
ახლა შევავსოთ სტუდენტების ცხრილი:

dbo	Students
Columns	
PK	StudentId (PK, int, not null)
NVARCHAR(30)	StudentName (nvarchar(30), not null)
NVARCHAR(30)	StudentLastName (nvarchar(30), not null)
VARCHAR(30)	Email (varchar(30), null)
FK	DirectionId (FK, int, not null)
VARBINARY(250)	Password (varbinary(250), null)

გამოვტოვოთ პაროლის ველი ამიტომ სავალდებულოა მრგვალ ფრჩხილებში  
მივუთითოთ შესავსები ველების ჩამონათვალი:

```
INSERT INTO [dbo].[Students]
(
    [StudentName],
    [StudentLastName],
    [Email],
    [DirectionId]
)
VALUES
(N'ერეკლე', N'ბურვაძე', 'aaa@gmail.com', 1),
(N'ნინო', N'ქურდაძე', 'aaa@gmail.com', 1),
(N'ანნა', N'გაგუა', 'aaa@gmail.com', 2),
(N'გიგო', N'პაივიძე', 'aaa@gmail.com', 3),
(N'ნუცა', N'ურიდია', 'aaa@gmail.com', 1),
(N'მაკა', N'ჯოჯუა', 'aaa@gmail.com', 2),
(N'გიორგი', N'სალია', 'aaa@gmail.com', 2),
(N'დავითი', N'ზვიადაძე', 'aaa@gmail.com', 3),
(N'ნიკა', N'ცინცაძე', 'aaa@gmail.com', 1),
(N'ლელა', N'დანელია', 'aaa@gmail.com', 1);
```

დავათვალიეროთ ცხრილი Select \* from [Students]ბრძანებით.



The screenshot shows the SQL Server Management Studio interface with a query window containing the following code:

```
Select * from [students]
```

The results pane displays the data from the Students table:

	StudentId	StudentName	StudentLastName	Email	DirectionId	Password
1	1000	ერეკლე	ბურვაძე	aaa@gmail.com	1	NULL
2	1001	ნინო	ქურდაძე	aaa@gmail.com	1	NULL
3	1002	ანნა	გაგუა	aaa@gmail.com	2	NULL
4	1003	გიგო	პაივიძე	aaa@gmail.com	3	NULL
5	1004	ნუცა	ურიდია	aaa@gmail.com	1	NULL
6	1005	მაკა	ჯოჯუა	aaa@gmail.com	2	NULL
7	1006	გიორგი	სალია	aaa@gmail.com	2	NULL
8	1007	დავითი	ზვიადაძე	aaa@gmail.com	3	NULL
9	1008	ნიკა	ცინცაძე	aaa@gmail.com	1	NULL
10	1009	ლელა	დანელია	aaa@gmail.com	1	NULL

ახლა კი ზოგიერთი სტუდენტისთვის შევავსოთ შესაბამისი დეტალურის ცხრილი:

```
INSERT INTO [dbo].[StudentDetails]
(
[StudentId],
[DateOfBirth],
[PersonalId],
[Phone],
[City]
)
VALUES

(1001, '3/4/2000', '02020104340', '577324543', N'ბათუმი'),
(1000, '3/4/2001', '01212010434', '578464543', N'თელავი'),
(1002, '1/5/1998', '01110104311', 'N'578324543', N'ქუთაისი'),
(1003, '5/7/2002', '02020111240', 'N'57731243', N'თელავი'),
(1005, '7/8/2001', '01320111233', 'N'577324543', N'ბათუმი'),
(1007, '10/2/2003', '60000000000', 'N'571224543', DEFAULT),
(1009, '1/3/1997', '70000000000', 'N'599324543', DEFAULT)
```

დავათვალიეროთ ცხრილი Select \* from [StudentDetails] ბრძანებით.

The screenshot shows the SQL Server Management Studio interface with a query window containing the following code:

```
Select * from [dbo].[StudentDetails]
```

The results tab displays the following data:

	StudentDetailId	StudentId	PersonalId	Photo	DateOfBirth	Phone	Email	City
1	3	1001	02020104340	NULL	2000-03-04	577324543	NULL	ბათუმი
2	4	1000	01212010434	NULL	2001-03-04	578464543	NULL	თელავი
3	5	1002	01110104311	NULL	1998-01-05	578324543	NULL	ქუთაისი
4	6	1003	02020111240	NULL	2002-05-07	57731243	NULL	თელავი
5	7	1005	01320111233	NULL	2001-07-08	577324543	NULL	ბათუმი
6	8	1007	60000000000	NULL	2003-10-02	571224543	NULL	თბილისი
7	9	1009	70000000000	NULL	1997-01-03	599324543	NULL	თბილისი

თუ კი სკრიპტში რომელიმე სტრიქონში გვინდა რომ სვეტი შეივსოს გაჩუმების პრინციპით მინიჭებული მნიშვნელობით უნდა შესაბამისი მნიშვნელობის ადგილას მივუთითოთ სიტყვა **DEFAULT**.

მონაცემების განახლება ცხრილებში Update ბრძანების გამოყენებით ცხრილში ჩანაწერების განახლებისთვის გამოიყენება ბრძანება UPDATE სინტაქსი:

```
UPDATE table_name
SET column_name1 = new_value,
column_name2 = new_value,
...
[WHERE Condition];
```

იმისათვის რომ განვაახლოთ ერთი ან რამოდენიმე სტრიქონი Students ცხრილში საჭიროა მივუთითოთ

1. **UPDATE** საბრძანებო სიტყვა, შემდეგ კი ცხრილის სახელი რომელშიც ხდება ცვლილება, ჩვენს შემთხვევაში **UPDATE [dbo].[Students]**
2. **SET** საბრძანებო სიტყვა, შესაცვლელი სვეტებისა და მათი ახალი მნიშვნელობით, რამოდენიმე შესაცვლელი სვეტის არსებობის შემთხვევაში თითოეული გამოვყოთ მძიმით. ჩვენს შემთხვევაში მეილში ჩავწეროთ ახალი მნიშვნელობა **SET [Email]='Nino123@gmail.com'**
3. **WHERE** პირობა რომელშიც მივუთითებთ რომელ სტრიქონზე ხდება ცვლილება, შეიძლება განახლების ოპერაციას საერთოდ არ ქონდეს ლოგიკური პირობა, მაგრამ უნდა გავითვალისწინოთ რომ ამ შემთხვევაში განახლება მთელი სვეტი. ჩვენს შემთხვევაში მეილი განვაახლოთ სტუდენტის ნომერ 1001 სთვის. **WHERE [StudentId]=1001**

მივიღებთ კოდს:

```
UPDATE [dbo].[Students]
SET [Email]='Nino123@gmail.com'
WHERE [StudentId]=1001
```

დავათვალიეროთ ცხრილი **Select \* from [Students]** ბრძანებით.

The screenshot shows the SQL Server Management Studio interface with a query window containing the following code:

```
Select * from [Students]
```

The results pane displays the data from the Students table:

	StudentId	StudentName	StudentLastName	Email	DirectionId	Password
1	1000	ერევლი	ბურჯაძე	aaa@gmail.com	1	NULL
2	1001	ნინო	ჭურდაძე	Nino123@gmail.com	1	NULL
3	1002	ანნა	ბაბუა	aaa@gmail.com	2	NULL
4	1003	გიგო	ვაკიძე	aaa@gmail.com	3	NULL
5	1004	ნუცა	ურიკიძე	aaa@gmail.com	1	NULL
6	1005	მაკა	ჯოჯია	aaa@gmail.com	2	NULL
7	1006	გიორგი	სალია	aaa@gmail.com	2	NULL
8	1007	დავითი	ზვიადაძე	aaa@gmail.com	3	NULL
9	1008	ნიკა	ცინცაძე	aaa@gmail.com	1	NULL
10	1009	ლელა	დანელია	aaa@gmail.com	1	NULL

სტუდენტის ნომერ 1001-ს განუახლდა მეილის ველი.

ახლა განვაახლოთ რამოდენიმე სვეტი ერთად, მაგალითად სტუდენტის ნომერ 1005-ს დეტალურ ინფორმაციაში შევუცვალოთ ქალაქი და დაბადების თარიღი:

```

UPDATE [dbo].[StudentDetails]
SET [City]=N'რუსთავი',
[DateOfBirth]='2000-03-07'
WHERE [StudentId]=1005

```

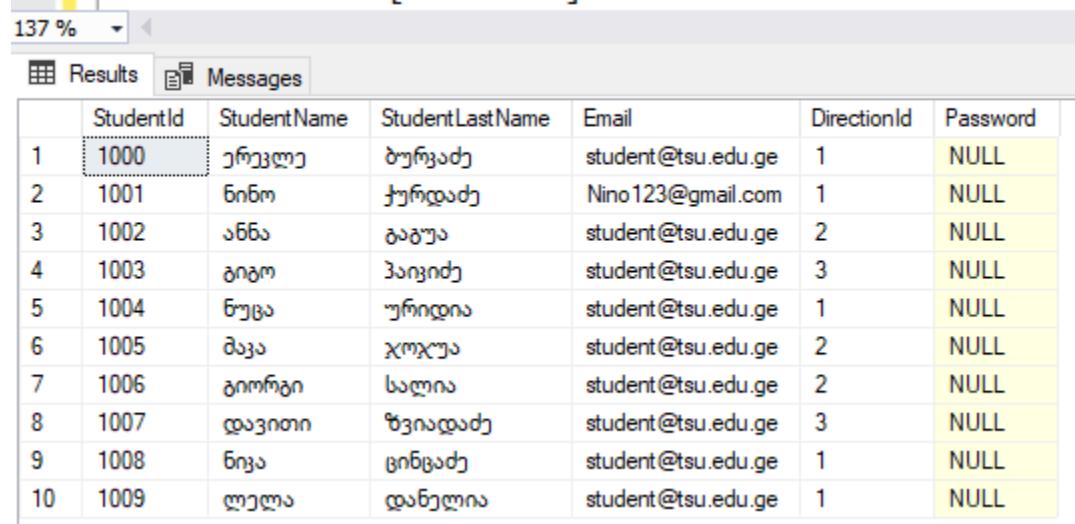
ასევე თუ ლოგიკურ (WHERE) პირობას ერთდროულად რამოდენიმე ჩანაწერი აკმაყოფილებს, შესაძლებელია რამოდენიმე სტრიქონის ერთდროულად განახლება. მაგალითად იმ სტუდენტებს ვისაც აქამდე მეილში ეწერათ 'aaa@gmail.com' ჩავუწეროთ 'student@tsu.edu.ge'

```

UPDATE [dbo].[Students]
SET [Email]='student@tsu.edu.ge'
WHERE [Email]='aaa@gmail.com'

```

კვლავ დავათვალიეროთ ცხრილი `Select * from [Students]` ბრძანებით.



The screenshot shows the SQL Server Management Studio interface with the following details:

- Query Editor:** Contains the SQL command: `Select * from [Students]`.
- Status Bar:** Shows "137 %".
- Results Tab:** Active tab, showing the query results.
- Messages Tab:** Tab next to Results.
- Table Data:** The Students table has 10 rows of data.

	StudentId	StudentName	StudentLastName	Email	DirectionId	Password
1	1000	ერეკო	ბურჯაძე	student@tsu.edu.ge	1	NULL
2	1001	ნინო	ჭურდაძე	Nino123@gmail.com	1	NULL
3	1002	ანნა	ბაბუა	student@tsu.edu.ge	2	NULL
4	1003	გიორგი	პავლიძე	student@tsu.edu.ge	3	NULL
5	1004	ნუცა	ურიდია	student@tsu.edu.ge	1	NULL
6	1005	მაკა	ჯოჯუა	student@tsu.edu.ge	2	NULL
7	1006	გიორგი	სალია	student@tsu.edu.ge	2	NULL
8	1007	დავითი	ზვიადაძე	student@tsu.edu.ge	3	NULL
9	1008	ნიკა	ცინცაძე	student@tsu.edu.ge	1	NULL
10	1009	ლელა	დანელია	student@tsu.edu.ge	1	NULL

განახლება სხვა ცხრილიდან: \*(ბონუსი)

შესაძლებელია ჩანაწერების განახლება სხვა ცხრილიდან, მაგალითად `StudentDetails` ცხრილში გვაქვს ცარიელი იმეილის ველი, თუმცა აღნიშნული ველი შევსებულია ცხრილ `Students` -ში. შესაძლებელია დაიწეროს პირობა რომელიც შეავსებს შესაბამისი ნომრის მქონე სტუდენტის მეილს დეტალურის ცხრილში საწყისი `Students` ცხრილის გამოყენებით:

```

UPDATE [dbo].[StudentDetails]
SET [Email]=[s].email
FROM [dbo].[Students] AS [s]
WHERE s.[StudentId]=[StudentDetails].[StudentId]

```

დავათვალიეროთ ცხრილი Select \* from [StudentDetails] ბრძანებით.

The screenshot shows the SQL Server Management Studio interface with a query window containing the command "Select \* from [StudentDetails]". Below the query window is a results grid titled "Results". The results show 7 rows of student data:

	StudentDetailId	StudentId	PersonalId	Photo	DateOfBirth	Phone	Email	City
1	3	1001	02020104340	NULL	2000-03-04	577324543	Nino123@gmail.com	ბათუმი
2	4	1000	01212010434	NULL	2001-03-04	578464543	student@tsu.edu.ge	თელავი
3	5	1002	01110104311	NULL	1998-01-05	578324543	student@tsu.edu.ge	ქუთაისი
4	6	1003	02020111240	NULL	2002-05-07	57731243	student@tsu.edu.ge	თელავი
5	7	1005	01320111233	NULL	2000-03-07	577324543	student@tsu.edu.ge	რიგისტავი
6	8	1007	60000000000	NULL	2003-10-02	571224543	student@tsu.edu.ge	თბილისი
7	9	1009	70000000000	NULL	1997-01-03	599324543	student@tsu.edu.ge	თბილისი

მონაცემების წაშლა ცხრილებში Delete ბრძანების გამოყენებით  
ცხრილში ჩანაწერების წაშლისთვის გამოიყენება ბრძანება Delete  
სინტაქსი:

```
DELETE FROM table_name  
[WHERE Condition];
```

იმისათვის რომ წავშალოთ ერთი ან რამოდენიმე სტრიქონი Students ცხრილში  
საჭიროა მივუთითოთ

1. **DELETE FROM** საბრძანებო სიტყვა, შემდეგ კი ცხრილის სახელი რომელშიც  
ხდება ცვლილება, ჩვენს შემთხვევაში **DELETE FROM [dbo].[Students]**
2. **WHERE** პირობა რომელშიც მივუთითებთ რომელი სტრიქონის წაშლა ხდება.  
შეიძლება წაშლის ოპერაციას საერთოდ არ ქონდეს ლოგიკური პირობა, მაგრამ  
უნდა გავითვალისწინოთ რომ ამ შემთხვევაში წაშლება მთელი ცხრილის  
ჩანაწერები. ჩვენს შემთხვევაში წავშალოთ სტუდენტი ნომერი 1006. **WHERE  
[StudentId]=1006**

მივიღებთ კოდს:

```
DELETE FROM [dbo].[Students]  
WHERE [StudentId]=1006
```

ჩანაწერი **DELETE FROM [dbo].[Students]** წაშლის ყველა ჩანაწერს სტუდენტების  
ცხრილიდან.

აგრეთვე მნიშვნელოვანია რომ უკვე არსებული გარე გასაღებების გათვალისწინებით  
შეუძლებელია ისეთი ველების წაშლა რომელსაც აქვს კავშირი. მაგალითად

Directions ცხრილიდან შეუძლებელია წავშალოთ მიმართულება ნომერი 1 ვინაიდან მასზე დაკავშირებულია საგნები და სტუდენტები, მაგრამ შეგვიძლია წავშალოთ მიმართულება ნომერი 4 რომელსაც არ აქვს კავშრები:

```
DELETE FROM [dbo].[Directions] WHERE [DirectionId]=1 -მივიღებთ შეცდომას.  
DELETE FROM [dbo].[Directions] WHERE [DirectionId]=4 - არ მივიღებთ  
შეცდომას.
```

თუ გვინდა რომ სტუდენტს წავუშალოთ დეტალურიდან მეილი, ამ შემთხვევაში არ უნდა გამოვიყეონთ **DELETE** ბრძანება, უნდა გადავაწეროთ ცარიელი ჩანაწერი **UPDATE** ბრანგებით:

```
UPDATE [dbo].[StudentDetails]  
SET [Email]=NULL  
WHERE [StudentDetailId]=3
```

### SELECT მოთხოვნა

SQL Server-ში SELECT ბრძანება გამოიყენება სტრიქონების/სვეტების მონაცემების მისაღებად ერთი ან რამოდენიმე არსებული ცხრილიდან. იგი იცავს SQL (Structured Query Language) სტანდარტებს.

სინტაქსი:

```
SELECT column1, column2,...columnN  
FROM table_name
```

### ყველა სვეტის არჩევა:

ოპერატორი \* წარმოადგენს ცხრილის ყველა სვეტს. თუ გვსურს ცხრილის ყველა სვეტის ნახვა, თქვენ არ გჭირდებათ თითოეული სვეტის სახელის მითითება SELECT მოთხოვნაში:

```
Select * from Students
```

მოთხოვნა აბრუნებს ცხრილიდან ყველა მწკრივისა და სვეტის მონაცემებს Students, როგორც ეს ნაჩვენებია ქვემოთ:

Select \* from [Students]

137 %

Results Messages

	StudentId	StudentName	StudentLastName	Email	DirectionId	Password
1	1000	ერებლი	ბურჯაძე	aaa@gmail.com	1	NULL
2	1001	ნინო	ქურდაძე	Nino123@gmail.com	1	NULL
3	1002	ანნა	ბაბუა	aaa@gmail.com	2	NULL
4	1003	გიგო	ვაკეთიძე	aaa@gmail.com	3	NULL
5	1004	ნუცა	ურიდისა	aaa@gmail.com	1	NULL
6	1005	მავა	ჯოჯუა	aaa@gmail.com	2	NULL
7	1006	გიორგი	სალია	aaa@gmail.com	2	NULL
8	1007	დავითი	ზვიადაძე	aaa@gmail.com	3	NULL
9	1008	ნიკა	ცინცაძე	aaa@gmail.com	1	NULL
10	1009	ლელა	დაწელია	aaa@gmail.com	1	NULL

კონკრეტული სვეტების მონაცემების მოთხოვნა:

იმისათვის რომ მივიღოთ მონაცემები მხოლოდ კონკრეტული სვეტებიდან, საჭურია მიუთითოთ სვეტების სახელები SELECT განცხადებაში:

```
SELECT StudentId
      ,StudentName
      ,StudentLastName
  FROM Students
```

მივიღებთ შედეგს:

137 %

Results Messages

	StudentId	StudentName	StudentLastName
1	1000	ერებლი	ბურჯაძე
2	1001	ნინო	ქურდაძე
3	1002	ანნა	ბაბუა
4	1003	გიგო	ვაკეთიძე
5	1004	ნუცა	ურიდისა
6	1005	მავა	ჯოჯუა
7	1007	დავითი	ზვიადაძე
8	1008	ნიკა	ცინცაძე
9	1009	ლელა	დაწელია

მეტსახელი სვეტებისა და ცხრილებისთვის

სურვილისამებრ შესაძლებელია მივუთითოთ მეტსახელი (Alias) ერთი ან მეტი სვეტისთვის SELECT მოთხოვნაში. მეტსახელი არის მოთხოვნის ცხრილის ან სვეტის დროებითი სახელი.

Alias-ის უპირატესობა:

- Alias ხდის სვეტს უკეთესად კითხვადს შედეგებში.
- ალიასი გამოიყენება შეკითხვისას ცხრილებისთვის მცირე, შემოკლებული და მნიშვნელოვანი სახელების მისანიჭებლად, რათა ადვილი იყოს ცხრილების მითითება მრავალი ცხრილის შეერთებისას.
- ალიასი გვეხმარება ამოვიცნოთ, რომელი სვეტი რომელ ცხრილს ეკუთვნის, მრავალი ცხრილიდან მონაცემების მიღების შემთხვევაში.

შემდეგი მოთხოვნა ქმნის მეტსახელებს წინა მთხოვნაში არსებული სვეტებისთვის. მიუთითეთ მეტსახელი ბრჭყალებში ან ოთხკუთხა ფრჩხილები, თუ გსურთ მასში ჰარის ან სხვა სიმბოლოს გამოყენება, ასევე შესაძლებელია ალიასის წინ AS სიტყვის გამოყენება :

```
SELECT StudentId AS 'Student Id'  
      ,StudentName  Name  
      ,StudentLastName [სტუდენტის გვარი]  
  FROM Students
```

The screenshot shows the SQL Server Management Studio interface with the 'Results' tab selected. The query window contains the same code as above. Below it, the results grid displays the following data:

Student Id	Name	სტუდენტის გვარი
1000	ერეკლე	ბურჯაძე
1001	ნინო	ჭურდაძე
1002	ანნა	გაგუა
1003	გიგო	პავლიძე
1004	ნურა	ურიდია
1005	მავა	ჯოჯუა
1007	დავითი	ზვიადაძე
1008	ნიკა	ცინცაძე
1009	ლელა	დანელია

+ ოპერატორი SELECT განცხადებაში

ოპერატორი +MS SQL Server-ში აერთიანებს სტრიქონულ მნიშვნელობებს ან კრებს ციფრულ მნიშვნელობებს. შემდეგი კოდი აერთიანებს ორ varchar სვეტს შედეგში.

```
SELECT StudentId AS 'Student Id'  
      ,StudentName+' '+StudentLastName  
  FROM Students  
|  
|

```
SELECT StudentId AS 'Student Id'  
      ,StudentName+' '+StudentLastName  
  FROM Students
```


```

Student Id	(No column name)
1000	ერეკლე ბურჯაძე
1001	ნინო ქურდაძე
1002	ანნა ბაბუა
1003	გიგო პაივიძე
1004	ნუცა ურიციძა
1005	მავა ჯოჯუა
1007	დავითი ზვისაძაძე
1008	ნიკა ცინცაძე
1009	ლელა დანელია

თუმცა ასეთი ტიპის შედგენილი სვეტები კარგავენ სახელს და აღნიშნულ შემთხვევებში აუცილებელია ალიასის გამოყენება:

```
|  
|

```
SELECT StudentId AS 'Student Id'  
      ,StudentName+' '+StudentLastName AS FullName  
  FROM Students
```


```

Student Id	FullName
1000	ერეკლე ბურჯაძე
1001	ნინო ქურდაძე
1002	ანნა ბაბუა
1003	გიგო პაივიძე
1004	ნუცა ურიციძა
1005	მავა ჯოჯუა
1007	დავითი ზვისაძაძე
1008	ნიკა ცინცაძე
1009	ლელა დანელია

შემდეგი სკრიპტი უბრალოდ აჯამებს ნომრებს შერჩეულ მოთხოვნაში.

```
SELECT 10 + 15; --აბრუნებს 25  
SELECT 10.5 + 15; --აბრუნებს 25.5
```

## FROM ოპერატორი

**SELECT** განცხადებას რომელიც კითხულობს მონაცემებს ცხრილებიდან აუცილებად უნდა ჰქონდეს **FROM** საბრძანებო სიტყვა. იგი გამოიყენება ცხრილების სახელების ჩამოსაწერად, საიდანაც გვინდა გამოვიტაონოთ მონაცემები და დავაკონკრეტოთ შეერთება ამ ცხრილებს შორის.

**FROM** პუნქტში შესაძლებელია მივუთითოთ რამოდენიმე ცხრილი. თუმცა, თუ ცხრილებს აქვთ იდენტური დასახელების მქონე სვეტები, მაშინ უნდა დავაკონკრეტოთ სვეტების სრული სახელები ცხრილის სახელის მითითებით, **table\_name.column\_name** სტილში:

შემდეგი სკრიპტი ირჩევს სვეტებს ორი ცხრილიდან:

```
SELECT * FROM Students, Directions;
```

```
SELECT Students.*, Directions.* FROM Students, Directions;
```

```
SELECT S.*, D.* FROM Students S, Directions D;
```

```
SELECT [S].[StudentName], [D].[DirectionName]
```

```
FROM Students [S], Directions [D];
```

**FROM** პუნქტში რამოდენიმე ცხრილის არსებობა **WHERE** პირობისა და **JOIN**-ის გარეშე დააბრუნებს დუბლირებულ მონაცემებს თითოეული ცხრილიდან. მაგალითად, თუ **Students** ცხრილს აქვს ორი სტრიქონი და **Directions** ცხრილს აქვს ორი მწკრივი, მაშინ ზემოაღნიშნული მოთხოვნა დააბრუნებს  $2 \times 2$ , ოთხ რიგს, სადაც ერთი ცხრილის სვეტები მეორდება მეორე ცხრილის სვეტებისთვის. ამის შესახებ მეტს შეიტყობთ **JOIN** გაკვეთილიდან.

## WHERE პირობა

SQL Server-ში **SELECT** მოთხოვნას შეიძლება საჭიროებისამებრ ჰქონდეს **WHERE** პუნქტი მონაცემების გასაფილტრად. **WHERE** პუნქტი შეიძლება შეიცავდეს ერთ ან მეტ ლოგიკურ პირობას ცხრილების მონაცემების გასაფილტრად. **WHERE** პირობა ყოველთვის მოდის **FROM** პუნქტის შემდეგ და **GROUP BY**, **HAVING** და **ORDER BY** ოპერატორების წინ.

სინტაქსი:

```
SELECT column1, column2,...columnN  
FROM table_name  
WHERE boolean_expression;
```

**WHERE** პუნქტი შეიძლება შეიცავდეს ერთ ან მეტ პირობას, რომლებსაც პირობითი ოპერატორების გომოყენებით ფილტრავს მონაცემებს. განიხილეთ შემდეგი მოთხოვნა:

```
SELECT * FROM Subjects
    WHERE Credit>5
```

ზემოხსენებულ მოთხოვნაში, პირობა Credit>5 აბრუნებს სტრიქონებს, სადაც Credit სვეტის მნიშვნელობა 5-ზე მეტია.

```
SELECT * FROM Subjects
    WHERE Credit>5
```

SubjectId	SubjectName	DirectionId	Credit
3	3როკეტი	1	10
7	დაპროგრამული კურსები	2	6

შემდეგი მოთხოვნა იყენებს BETWEEN ოპერატორს **WHERE** პირობაში:

```
SELECT * FROM StudentDetails
    WHERE [DateOfBirth] BETWEEN '2000-03-04' AND '2002-05-07'
```

პირობა [DateOfBirth] BETWEEN '2000-03-04' AND '2002-05-07' აბრუნებს რიგებს, სადაც სვეტის მნიშვნელობა DateOfBirth არის '2000-03-04' -დან '2002-05-07'-მდე (ორივე მნიშვნელობის ჩათვლით):

```
SELECT * FROM StudentDetails
    WHERE [DateOfBirth] BETWEEN '2000-03-04' AND '2002-05-07'
```

StudentDetailId	StudentId	PersonalId	Photo	DateOfBirth	Phone	Email	City
3	1001	02020104340	NULL	2000-03-04	577324543	Nino123@gmail.com	ბათუმი
4	1000	01212010434	NULL	2001-03-04	578464543	student@tsu.edu.ge	თელავი
6	1003	02020111240	NULL	2002-05-07	57731243	student@tsu.edu.ge	თელავი
7	1005	01320111233	NULL	2000-03-07	577324543	student@tsu.edu.ge	რუსთავი

პირობა WHERE მრავალი პირობით

**WHERE** პირობა შეიძლება შეიცავდეს მრავალ პირობას და აერთიანებდეს მათ AND და OR ოპერატორების გამოყენებით. შემდეგი მოთხოვნა იყენებს ლოგიკურ ოპერატორს AND-ს მონაცემების გასაფილტრად ორი პირობის საშუალებით:

```
SELECT * FROM Subjects
    WHERE Credit<=5 AND [DirectionId]=1
```

ზემოთ მოყვანილ მოთხოვნაში **WHERE** პირობა `Credit<=5 AND [DirectionId]=1` განსაზღვრავს ორ პირობას, რომლებიც გაერთიანებულია **AND** ოპერატორით. შედეგად მივიღებთ რიგებს ცხრილიდან **Subjects**, სადაც მნიშვნელობა **DirectionId** არის 1 და **Credit** ნაკლებია ან ტოლია 5-ის.



```
SELECT * FROM Subjects
WHERE Credit<=5 AND [DirectionId]=1
```

**Results** **Messages**

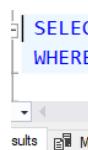
	SubjectId	SubjectName	DirectionId	Credit
1	6	NLP	1	5
2	8	მანქანური სწავლება	1	3

ქვემოთხოვნა **WHERE** პირობაში

**WHERE** პირობას ასევე შეუძლია გამოიყენოს სხვა მოთხოვნის შედეგად მიღებული მნიშვნელობა:

```
SELECT * FROM Students
WHERE DirectionId =(SELECT DirectionId FROM Directions WHERE
DirectoryName=N'კომპიუტერული მეცნიერება')
```

ზემოთ მოყვანილ სკრიპტში **WHERE** პირობა გამოიყურება შემდეგნაირად: **WHERE** **DirectionId** =(**SELECT** **DirectionId** **FROM** **Directions** **WHERE** **DirectoryName**=N'**კომპიუტერული მეცნიერება**'.). ამიტომ, პირველ რიგში შესრულდება ქვემოთხოვნა **SELECT DirectionId FROM Directions WHERE** **DirectoryName**=N'**კომპიუტერული მეცნიერება**' და მიღებული **DirectionId** მნიშვნელობა (ამ შემტხვევაში-1) გამოყენებული იქნება რიგების გასაფილტრად. და აღუშნული პირობა იდენტური იქნება **WHERE** **DirectionId** =1 პირობის.



```
SELECT * FROM Students
WHERE DirectionId =(SELECT DirectionId FROM Directions WHERE DirectionName=N'კომპიუტერული მეცნიერება')
```

**Results** **Messages**

StudentId	StudentName	StudentLastName	Email	DirectionId	Password
1000	ერეკ	ბურჯაძე	student@tsu.edu.ge	1	NULL
1001	ნინო	ჭურიაძე	Nino123@gmail.com	1	NULL
1004	ნურა	ურიდია	student@tsu.edu.ge	1	NULL
1008	ნინა	ცონგაძე	student@tsu.edu.ge	1	NULL
1009	ლელა	დაწელია	student@tsu.edu.ge	1	NULL

პირობითი ოპერატორები

შემდეგი ოპერატორების გამოყენება შესაძლებელია **WHERE** პირობაში:

ოპერატორი	აღწერა
=	ტოლია
>	მეტია
<	ნალებია
>=	მეტი ან ტოლი
<=	ნაკლები ან ტოლი
<> ან !=	არ უდრის. ზოგიერთ მონაცემთა ბაზაში != გამოიყენება არა ტოლი მნიშვნელობების შესადარებლად.
BETWEEN	დიაპაზონს შორის
LIKE	ნიმუშის მსგავსი: პროცენტის სიმბოლო (%): გამოსახავს სიმბოლოთა წებისმიერ რაოდენობას. ქვედატირე (_): ანაცვლებს ერთ სიმბოლოს. [სიმბოლოების თანმიმდევრობა]: ემთხვეოდეს ამ სიმრავლიდან წებისმიერ სიმბოლოს. [სიმბოლო-სიმბოლო]: წებისმიერი სიმბოლო მოცემული დიაპაზონიდან [^]: წებისმიერი სიმბოლო, რომელიც არ ეკუთვნის მოცემულ დიაპაზონს.
IN	სიის ელემენტებთან გატოლება

## GROUP BY ოპერატორი

SQL Server-ში GROUP BY ოპერატორი გამოიყენება შემაჯამებული მონაცემების მისაღებად ერთი ან მეტი დაჯუფების პირობის საფუძველზე. ჯუფები შეიძლება ჩამოყალიბდეს ერთ ან მეტ სვეტზე. მაგალითად, მოთხოვნა GROUP BY გამოყენებული იქნება თითოეულ მიმართულებაზე სტუდენტების რაოდენობის დასათვლელად, ან სტუდენტთა ჯამური რაოდენობის მისაღებად. ამისათვის ჩვენ უნდა გამოვიყენოთ აგრეგატული ფუნქციები, როგორიცაა COUNT(), MAX(), და ა.შ. SELECT მოთხოვნაში GROUP BY პირობის შედეგი აბრუნებს ერთ მწვრივს GROUP BY სვეტის თითოეული მნიშვნელობისთვის.

სინტაქსი:

```
SELECT column1, column2,...columnN FROM table_name
[WHERE]
[GROUP BY column1, column2...columnN]
[HAVING]
[ORDER BY]
```

SELECT მოთხოვნა შეიძლება შეიცავდეს მხოლოდ იმ სვეტებს, რომლებიც გამოიყენება GROUP BY პირობაში. SELECT მოთხოვნაში სხვა სვეტების ჩასართავად

უნდა გამოვიყენოთ აგრეგატული ფუნქციები, როგორიცაა **COUNT()**, **MAX()**, **MIN()**, **SUM()**, **AVG()**.

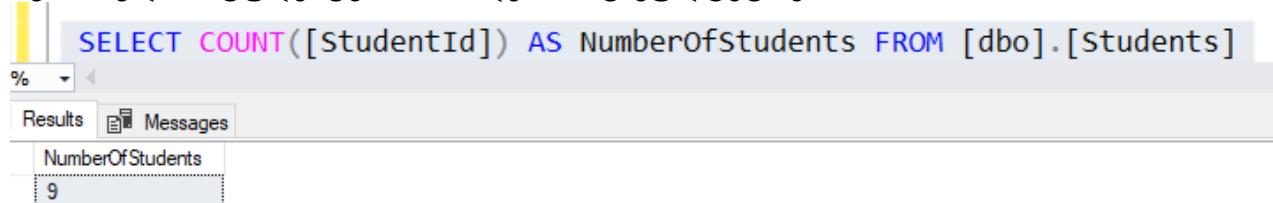
GROUP BY მახასიათებლები:

- GROUP BY პუნქტი გამოიყენება ჩანაწერების ჯგუფების შესაქმნელად.
- GROUP BY პუნქტი უნდა იჯდეს WHERE პუნქტის შემდეგ, მისი არსებობის შემთხვევაში და HAVING პუნქტამდე.
- GROUP BY პუნქტი შეიძლება შეიცავდეს ერთ ან მეტ სვეტს ამ სვეტების საფუძველზე ერთი ან მეტი ჯგუფის შესაქმნელად.
- მხოლოდ GROUP BY სვეტები შეიძლება შევიდეს SELECT პუნქტში. SELECT პუნქტში სხვა სვეტების გამოსაყენებლად გამოიყენეთ მათთან ერთად აგრეგატული ფუნქციები.

განვიხილოთ მოთხოვნა:

```
SELECT COUNT([StudentId]) AS NumberOfStudents FROM [dbo].[Students]
```

იგი ითვლის სტუდენტების რაოდენობა ფაკულტეტზე



The screenshot shows the SQL Server Management Studio interface. A query window is open with the following T-SQL code:

```
SELECT COUNT([StudentId]) AS NumberOfStudents FROM [dbo].[Students]
```

The results pane shows a single row with the column name "NumberOfStudents" and the value "9".

თუ გვსურს მიმართულებების მიხედვით სტუდენტების რაოდენობის დათვლა საჭიროა ცხრილი დავაჯგუფოთ მიმართულებების მიხედვით და შემდგომ დავთვალოთ რაოდენობა თითოეულ ჯგუფში:

```
SELECT COUNT([StudentId]) AS NumberOfStudents FROM [dbo].[Students]
GROUP BY [DirectionId]
```

```

CREATE DATABASE [Faculty];
GO

USE [Faculty1];

CREATE TABLE [Directions]
(
    [DirectionId] INT NOT NULL PRIMARY KEY IDENTITY,
    [DirectoryName] NVARCHAR(150) NOT NULL
        UNIQUE,
    [DirectionHead] INT
        NULL
);
;

CREATE TABLE [Students]
(
    [StudentId] INT IDENTITY(1000,1)
        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
        FOREIGN KEY REFERENCES [dbo].[Directions] ([DirectionId]),
    [Password] VARBINARY(250)
);
;

CREATE TABLE [StudentDetails]
(
    [StudentDetailId] INT IDENTITY PRIMARY KEY,
    [StudentId] INT
        UNIQUE
        FOREIGN KEY REFERENCES [dbo].[Students] ([StudentId]),
    [PersonalId] VARCHAR(11)
        UNIQUE,
    [Photo] VARBINARY(MAX),
    [DateOfBirth] DATE,
    [Phone] VARCHAR(20),
    [City] NVARCHAR(50)
        DEFAULT (N'თბილისი'),
);
;

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
);
;

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)
);

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 '%@%')
);

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]
    )
);

```

## ლაბორატორია 6

(ლაბორატორიის ბოლოს მოცემულია ბაზა, რომელიც დაგჭირდებათ  
მოთხოვნების შესასრულებლად)

### GROUP BY ოპერატორი

SQL Server-ში GROUP BY ოპერატორი გამოიყენება შემაჯამებული მონაცემების  
მისაღებად ერთი ან მეტი დაჯგუფების პირობის საფუძველზე. ჯგუფები შეიძლება  
ჩამოყალიბდეს ერთ ან მეტ სვეტზე. მაგალითად, მოთხოვნა GROUP BY  
გამოიყენებული იქნება თითოეულ მიმართულებაზე სტუდენტების რაოდენობის  
დასათვლელად, ან სტუდენტთა ჯამური რაოდენობის მისაღებად.  
ამისათვის ჩვენ უნდა გამოვიყენოთ აგრეგატული ფუნქციები,  
როგორიცაა **COUNT()**, **MAX()**, და ა.შ. SELECT მოთხოვნაში GROUP BY პირობის  
შედეგი აბრუნებს ერთ მწკრივს GROUP BY სვეტის თითოეული  
მნიშვნელობისთვის.

სინტაქსი:

```
SELECT column1, column2,...columnN FROM table_name
[WHERE]
[GROUP BY column1, column2...columnN]
[HAVING]
[ORDER BY]
```

SELECT მოთხოვნა შეიძლება შეიცავდეს მხოლოდ იმ სვეტებს, რომლებიც  
გამოიყენება GROUP BY პირობაში. SELECT მოთხოვნაში სხვა სვეტების ჩასართავად  
უნდა გამოვიყენოთ აგრეგატული ფუნქციები, როგორიცაა **COUNT()**, **MAX()**, **MIN()**,  
**SUM()**, **AVG()**.

GROUP BY მახასიათებლები:

- GROUP BY პუნქტი გამოიყენება ჩანაწერების ჯგუფების შესაქმნელად.
- GROUP BY პუნქტი უნდა იჯდეს WHERE პუნქტის შემდეგ, მისი არსებობის  
შემთხვევაში და HAVING პუნქტამდე.
- GROUP BY პუნქტი შეიძლება შეიცავდეს ერთ ან მეტ სვეტს ამ სვეტების  
საფუძველზე ერთი ან მეტი ჯგუფის შესაქმნელად.
- მხოლოდ GROUP BY სვეტები შეიძლება შევიდეს SELECT პუნქტში. SELECT  
პუნქტში სხვა სვეტების გამოსაყენებლად გამოიყენეთ მათთან ერთად  
აგრეგატული ფუნქციები.

განვიხილოთ მოთხოვნა:

```
SELECT COUNT([StudentId]) AS NumberOfStudents
FROM [dbo].[Students]
```

იგი ითვლის სტუდენტების რაოდენობა ფაკულტეტზე

The screenshot shows a SQL query being run in SSMS. The query is:

```
SELECT COUNT([StudentId]) AS NumberOfStudents FROM [dbo].[Students]
```

The results pane shows a single row with the value 9.

NumberOfStudents
9

მნიშვნელოვანია დავიმახსოვროთ, რომ თუ მთხოვნა შეიცავს აგრეგატულ ფუნქციებს (`COUNT()`, `MAX()`, `MIN()`, `SUM()`, `AVG()`...) მოთხოვნის პირობაში ზემოთაღნიშნულის გარდა სხვა ველის დამატება ხდება შეუძლებელი. მხოლოდ დაჯგუფების პირობის გამოყენება გვაძლევს საშუალებას დავამატოთ მოთხოვნაში ის ველი რომელიც გამოყენებულია `GROUP BY` -ს ტანში. მარტივად რომ ვთქვათ მოთხოვნა

```
SELECT  
COUNT([StudentId]) AS NumberOfStudents,  
[DirectionId]  
FROM [dbo].[Students]
```

დააბრუნებს შეცდომას: „`Column 'dbo.Students.DirectionId' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause.`“

„სვეტი „`dbo.Students.DirectionId`“ არასწორია მოთხოვნის სიაში, რადგან ის არ შეადგენს არც აგრეგატული ფუნქციის პარამეტრს და არც `GROUP BY` ოპერატორის შიგთავსს“ მარტივი მსახვედრია რომ თუ კი გვინდა სტუდენტთა რაოდენობა დავთვალოთ მიმართულებების მიხედვით ჯერ ცხრილი უნდა დაჯგუფდეს მიმართულებების შესაბამისად და შემდგომ უკვე დაჯგუფებულ დროებით ქვეცხრილებში მოხდეს სტუდენტთა რაოდენობების დათვლა.

```
SELECT COUNT([StudentId]) AS NumberOfStudents  
FROM [dbo].[Students]  
GROUP BY [DirectionId]
```

```

SELECT COUNT([StudentId]) AS NumberOfStudents
FROM [dbo].[Students]
GROUP BY [DirectionId]

```

137 %

	NumberOfStudents
1	5
2	3
3	2

თუმცა აღნიშნული მოთხოვნის შედეგში გაურკვეველია რომელ ფაკულტეტზე რამდენია სტუდენტების რაოდენობა, შესაბამისად საჭიროა რომ გამოვიტანოთ ინფორმაცია ფაკულტეტის შესახებ. ზემოაღნიშნული მოთხოვნა მოიცავს **GROUP BY DirectionId** დირექტივას, ვინდაიდან **DirectionId** გამოყენებულია როგორც დაჯგუფების ბრძანების პარამეტრი მისი გამოყენება შესაძლებელი ხდება **SELECT** ოპერატორში:

```

SELECT
[DirectionId],
COUNT([StudentId]) AS NumberOfStudents
FROM [dbo].[Students]
GROUP BY [DirectionId]

```

შედეგად შესაძლებელი ხდება გარკვევა თუ რომელ ნომერ მიმართულებაზე რამდენი სტუდენტია დარეგისტრირებული:

```

SELECT
[DirectionId],
COUNT([StudentId]) AS NumberOfStudents
FROM [dbo].[Students]
GROUP BY [DirectionId]

```

137 %

	DirectionId	NumberOfStudents
1	1	5
2	2	3
3	3	2

იმისთვის რომ ნომრის მაგიერ მოთხოვნაში მივიღოთ მიმართულების სახელი საჭიროა შევასრულოთ მოთხოვნა:

```

SELECT
[D].[DirectoryName],
COUNT([S].[StudentId]) AS NumberOfStudents
FROM [dbo].[Students]AS [S],[dbo].[Directions] AS [D]
WHERE [S].[DirectionId]=[D].[DirectionId]
GROUP BY [D].[DirectoryName]

```

	DirectoryName	NumberOfStudents
1	ინფორმაციული სისტემები	3
2	ინფორმაციული ტექნოლოგები	2
3	კომპიუტერული მეცნიერება	5

### HAVING პირობა

როგორც უკვე ვნახეთ დაჯგუფების GROUP BY პირობა სრულდება WHERE-ის შემდგომ, შესაბამისად ჯერ ხდება where პირობის მიხედვით საწყისი ცხრილის გაფილტვრა, შემდეგ დაჯგუფება კონკრეტული სვეტის/ სვეტების მიხედვით და შემდგომ მათზე აგრეგატული ფუნქციების შესრულება. გამოდის რომ აგრეგატული ფუნქციის შედეგზე ჩვენ აღარ გვაქვს ფილტრი. რომ გვინდოდეს მხოლოდ იმ მიმართულებების გამოტანა სადაც 3-ზე მეტი სტუდენტია დარეგისტრირებული, ზემოთ მოცემული ოპერატორების საშუალებით ნამდვილად ვერ შევძლებთ. ამისათვის შემოდის დაჯგუფების ლოგიკური პირობა HAVING.

### სინტაქსი:

```

SELECT column1, column2,...columnN
FROM table_name
[WHERE]
[GROUP BY column1, column2...columnN]
[HAVING conditions]
[ORDER BY]

```

- HAVING პუნქტი გამოიყენება დაჯგუფებული ჩანაწერების გასაფილტრად.
- HAVING პირობა უნდა მოდიოდეს GROUP BY ოპერატორის შემდეგ და ORDER BY პუნქტის წინ.

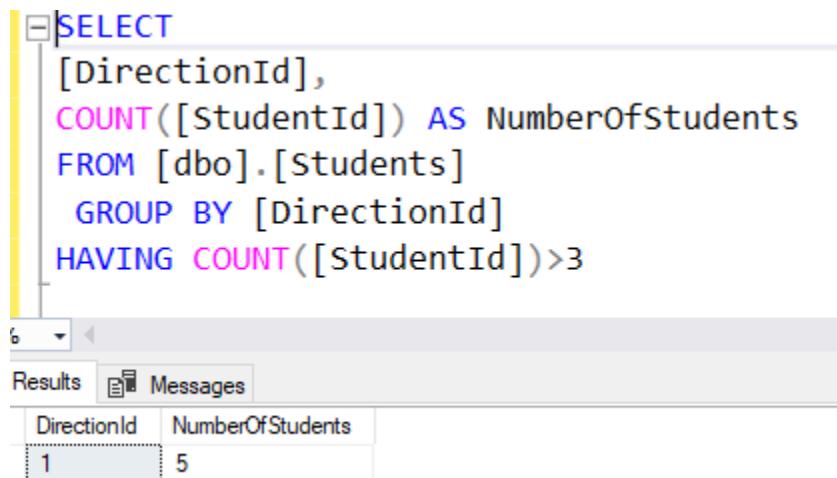
- **HAVING** ბრძანება შეიძლება შეიცავდეს ერთ ან მეტ ლოგიკურ პირობას.
- **HAVING** პირობა შეიძლება შეიცავდეს მხოლოდ სვეტებს, რომლებიც გამოიყენება **GROUP BY** ბრძანებასთან ერთად სელექტის მოთხოვნაში. მატივად რომ ვთქვათ **HAVING** პირობა ედება აგრეგატული ფუნქციის შედეგს და შესაბამისად მის ტანში ვიყენებთ ზემოთაღნიშნულ აგრეგატულ ფუნქციას:

**SELECT**

```
[DirectionId],
COUNT([StudentId]) AS NumberOfStudents
FROM [dbo].[Students]
GROUP BY [DirectionId]
HAVING COUNT([StudentId])>3
```

ზემოთ განხილულ კოდს დავამატოთ ფილტრი დაჯგუფების შედეგზე, რომ მხოლოდ ის მიმართულებები გამოვიტანოთ სადაც 3-ზე მეტი სტუდენტია დარეგისტრირებული.

დავაკვირდეთ მოთხოვნის ტანს, აგრეგატული ფუნქცია ჩადის **HAVING** პირობაში, ხოლო ყველა სვეტი რაც აგრეგატულის გარდა გვაქვს მოთხოვნის ჩამონათვალში უნდა ჩავიტანოთ **GROUP BY** ოპერატორში:



```
SELECT
[DirectionId],
COUNT([StudentId]) AS NumberOfStudents
FROM [dbo].[Students]
GROUP BY [DirectionId]
HAVING COUNT([StudentId])>3
```

DirectionId	NumberOfStudents
1	5

**ORDER BY** პირობა

SQL Server-ში, **ORDER BY** პუნქტი გამოიყენება **SELECT** მოთხოვნაში, რათა დაალაგოს შედეგი ერთი ან მეტი სვეტის ზრდადობის ან კლებადობის მიხედვით. სინტაქსი:

```
SELECT column1, column2, ...columnN
FROM table_name
```

```
[WHERE]
[GROUP BY]
[HAVING]
[ORDER BY column(s) [ASC|DESC]]
```

- **ORDER BY** პირობა გამოიყენება მოთხოვნის შედეგის ერთ ან რამდენიმე სვეტის მნიშვნელობების მიხედვით დასაგებლად. მაგაგალითად სტუდენტების ცხრილი დალაგდეს სტუდენტების გვარების მიხედვით, ხოლო იმ შემთხვევაში თუ გვარი განმეორდება გამოვიყენოთ მრავალდონიანი სორტირება და იდენტური ვერის მქონე სტუდენტები დავალაგოთ სახელის მიხედვით.
- **ORDER BY** პუნქტი უნდა მოდიოდეს **WHERE, GROUP BY** და **HAVING** პუნქტის შემდეგ, მოთხოვნაში მათი არსებობის შემთხვევაში.
- გამოიყენეთ **ASC** ან **DESC** სვეტის სახელის შემდეგ დალაგების თანმიმდევრობის შესაბამისად ზრდადობით ან კლებადობით განსასაზღვრად. ნაგულისხმევად, **ORDER BY** პუნქტი ახარისხებს ჩანაწერებს ზრდადობის მიხედვით.

```
SELECT [S].[StudentId],
       [S].[StudentName],
       [S].[StudentLastName],
       [S].[Email],
       [S].[DirectionId]
  FROM [dbo].[Students] AS [S]
 ORDER BY [S].[StudentLastName]
```

დაალაგებს სტუდენტების ცხრილს, სტუდენტთა გვარის სვეტის მიხედვით, ზრდადობით. ხოლო

```
SELECT [S].[StudentId],
       [S].[StudentName],
       [S].[StudentLastName],
       [S].[Email],
       [S].[DirectionId]
  FROM [dbo].[Students] AS [S]
 ORDER BY [S].[StudentLastName] desc
```

მოახდენს ცხრილის დალაგებას სტუდენტთა გვარის სვეტის მიხედვით კლებადობით.

დალაგება შეიძლება მოხდეს რამოდენიმე სვეტის მიხედვით, დავალაგოთ ჯერ გვარის ხოლო შემდეგ კი სახელის სვეტის მიხედვით (მრავალდონიანი სორტირება)

```
SELECT [S].[StudentId],
       [S].[StudentName],
```

```

[S].[StudentLastName],
[S].[Email],
[S].[DirectionId]
FROM [dbo].[Students] AS [S]
ORDER BY [S].[StudentLastName] ASC, [S].[StudentName] ASC

```

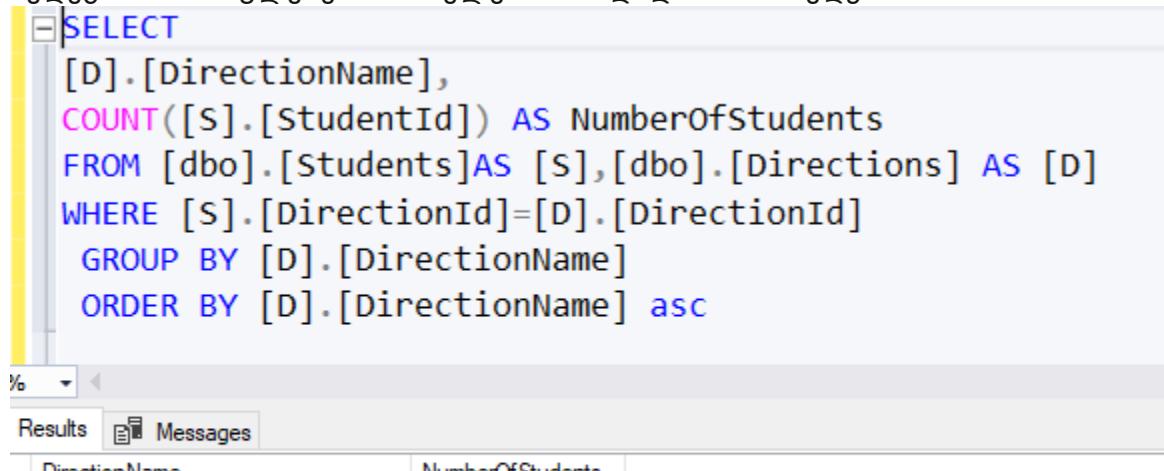
ასევე დალაგების ფუნქცია შეიძლება გამოყენებული იყოს დაჯგუფებულ  
მონაცემებზე:

```

SELECT
[D].[DirectoryName],
COUNT([S].[StudentId]) AS NumberOfStudents
FROM [dbo].[Students]AS [S],[dbo].[Directions] AS [D]
WHERE [S].[DirectionId]=[D].[DirectionId]
GROUP BY [D].[DirectoryName]
ORDER BY [D].[DirectoryName] asc

```

აღნიშნულ შემთხვევაშიც დალაგების პარამეტრად შეიძლება გამოვიყენოთ მხოლოდ  
GROUP BY ოპერატორის პარამეტრად მდგომი სვეტი/სვეტები. და დავალაგოთ  
შედეგი მიმართულებების სახელების ზრდადობის მიხედვით:



```

SELECT
[D].[DirectoryName],
COUNT([S].[StudentId]) AS NumberOfStudents
FROM [dbo].[Students]AS [S],[dbo].[Directions] AS [D]
WHERE [S].[DirectionId]=[D].[DirectionId]
GROUP BY [D].[DirectoryName]
ORDER BY [D].[DirectoryName] asc

```

DirectoryName	NumberOfStudents
ინფორმაციული სისტემები	3
ინფორმაციული ტექნოლოგიები	2
კომპიუტერული მეცნიერება	5

## INNER JOIN მოთხოვნა

**INNER JOIN** მოთხოვნა გამოიყენება ორი ან მეტი ცხრილიდან შესატყვისი  
ჩანაწერების მოსამიებლად, მითითებული პირობის საფუძველზე.

სინტაქსი:

```

SELECT table1.column_name(s), table2.column_name(s)
FROM table1
INNER JOIN table2

```

```
ON table1.column_name = table2.column_name;
```

განვიხილოთ კვლავ სტუდენტისა და მიმართულების ცხრილები:

	StudentId	StudentName	StudentLastName	Email	DirectionId	Password
1	1000	ერეკლე	ბურჯაძე	aaa@gmail.com	1	NULL
2	1001	ნინო	ჭურდაძე	aaa@gmail.com	1	NULL
3	1002	ანნა	ბაბუა	aaa@gmail.com	2	NULL
4	1003	გიგო	ვაკენძე	aaa@gmail.com	3	NULL
5	1004	ნუცა	ურიფია	aaa@gmail.com	1	NULL
6	1005	მავა	ჯოჯუა	aaa@gmail.com	2	NULL
7	1006	გიორგი	სალია	aaa@gmail.com	2	NULL
8	1007	დავითი	ზვიადაძე	aaa@gmail.com	3	NULL
9	1008	ნიკა	ცინფაძე	aaa@gmail.com	1	NULL
10	1009	ლელა	დანელია	aaa@gmail.com	1	NULL

	DirectionId	DirectoryName	DirectionHead
1	1	კომპიუტერული მეცნიერება	NULL
2	2	ინფორმაციული სისტემები	NULL
3	3	ინფორმაციული ტექნოლოგიები	NULL

თითოეულ სტუდენს აქვს მინიჭებული ვალიდური მიმართულების ნომერი. დავწეროთ მოთხოვნა, რომელიც აიღებს ჩანაწერებს ორივე ცხრილიდან, თითოეული სტუდენტისთვის ნახავს რა მიმართულების ნომერია მინიჭებული, შემდეგ მიმართულების ნომრის მიხედვით გადავა და მიმართულებების ცხრილში მოიძიებს შესაბამის მიმართულებას, აიღებს მის დასახელებას და მიუწერს სტუდენტს. ფაქტია რომ მოჭიდების, ანუ გადაბმის ველი ამ ორ ცხრილს შორის იქნებს მიმართულების ნომერი. მივიღებთ მოთხოვნას:

```
SELECT [S].[StudentId],
       [S].[StudentName],
       [S].[StudentLastName],
       [S].[Email],
       [S].[DirectionId],
       [D].[DirectoryName]
  FROM [dbo].[Students] AS [S]
    INNER JOIN [dbo].[Directions] AS [D]
      ON [D].[DirectionId] = [S].[DirectionId];
```

ორი ცხრილი შეუერთდა ერთმანეთს [D].[DirectionId] = [S].[DirectionId] პირობით, და ამოიკრიბა შედეგში ჩანაწერები რომლებიც აკმაყოფილებდა აღნიშნულ პირობას. ხოლო სტრიქონები სადაც DirectionId იყო NULL ან არ ჰქონდა შესაბამისი ჩანაწერი ორივე ცხრილში, უბრალოდ გამოიტოვება შედეგში.

```

SELECT [S].[StudentId],
       [S].[StudentName],
       [S].[StudentLastName],
       [S].[Email],
       [S].[DirectionId],
       [D].[DirectoryName]
  FROM [dbo].[Students] AS [S]
 INNER JOIN [dbo].[Directions] AS [D]
    ON [D].[DirectionId] = [S].[DirectionId];

```

Results Messages

StudentId	StudentName	StudentLastName	Email	DirectionId	DirectoryName
1000	ერეკლე	ბურვაძე	aaa@gmail.com	1	კომპიუტერული მეცნიერება
1001	ნინო	ქურდაძე	aaa@gmail.com	1	კომპიუტერული მეცნიერება
1002	ანნა	გაგუა	aaa@gmail.com	2	ინფორმაციული სისტემები
1003	გიმო	პავლიძე	aaa@gmail.com	3	ინფორმაციული ტექნოლოგიები
1004	ნურა	ურიდია	aaa@gmail.com	1	კომპიუტერული მეცნიერება
1005	მავა	ჯოაზუა	aaa@gmail.com	2	ინფორმაციული სისტემები
1006	გიორგი	სალია	aaa@gmail.com	2	ინფორმაციული სისტემები
1007	დავითი	ზვიადაძე	aaa@gmail.com	3	ინფორმაციული ტექნოლოგიები
1008	ნიკა	ცინაძე	aaa@gmail.com	1	კომპიუტერული მეცნიერება
1009	ლელა	დაწელია	aaa@gmail.com	1	კომპიუტერული მეცნიერება

შიდა INNER შეერთების დროს არ აქვს მნიშვნელობა რომელი ცხრილი იქნება პირველი და რომელი მეორე, შედეგი იქნება უცვლელი ვინაიდან მარტივად რომ ვთქვათ ცხრილებიდან ვიღებთ ჩანაწერების „თანაკვეთას“. ასევე თუ დავწერთ მხოლოდ JOIN სიტყვას მაინც გაჩინდების პრინციპით იგულისხმება INNER JOIN პირობა.

```

SELECT [S].[StudentId],
       [S].[StudentName],
       [S].[StudentLastName],
       [S].[Email],
       [S].[DirectionId],
       [D].[DirectoryName]
  FROM [dbo].[Directions] AS [D]
    JOIN [dbo].[Students] AS [S]
      ON [D].[DirectionId] = [S].[DirectionId]

```

და მეტიც მსგავსი შიდა შეერთების მაგალითები ჩვენ უკვე დავწერეთ WHERE პირობის გამოყენებით:

```

SELECT [S].[StudentId],
       [S].[StudentName],
       [S].[StudentLastName],

```

```

[S].[Email],
[S].[DirectionId],
[D].[DirectoryName]
FROM [dbo].[Directions] AS [D],
[dbo].[Students] AS [S]
WHERE [D].[DirectionId] = [S].[DirectionId];

```

დავწეროთ მოთხოვნა სამი ცხრილის შეერთებით, დამატებით გაოვიტანოთ სტუდენტის სახლის მიმდევარებელი ქალაქი და ტელეფონი:

```

SELECT [S].[StudentId],
[S].[StudentName],
[S].[StudentLastName],
[S].[Email],
[SD].[Phone],
[SD].[City],
[S].[DirectionId],
[D].[DirectoryName]
FROM [dbo].[Directions] AS [D]
INNER JOIN [dbo].[Students] AS [S]
ON [D].[DirectionId] = [S].[DirectionId]
INNER JOIN [dbo].[StudentDetails] AS [SD]
ON [SD].[StudentId] = [S].[StudentId];

```

---

```

SELECT [S].[StudentId],
[S].[StudentName],
[S].[StudentLastName],
[S].[Email],
[SD].[Phone],
[SD].[City],
[S].[DirectionId],
[D].[DirectoryName]
FROM [dbo].[Directions] AS [D]
INNER JOIN [dbo].[Students] AS [S]
ON [D].[DirectionId] = [S].[DirectionId]
INNER JOIN [dbo].[StudentDetails] AS [SD]
ON [SD].[StudentId] = [S].[StudentId]

```

% ▾

	Results	Messages					
StudentId	StudentName	StudentLastName	Email	Phone	City	DirectionId	DirectoryName
1001	ნინო	ქურდაძე	aaa@gmail.com	577324543	ბათუმი	1	კომპიუტერული მეცნიერება
1000	ერეკლე	ბურჯაძე	aaa@gmail.com	578464543	თელავი	1	კომპიუტერული მეცნიერება
1002	ანნა	გაბუა	aaa@gmail.com	578324543	ქუთაისი	2	ინფორმაციული სისტემები
1003	გიგო	ნაინძე	aaa@gmail.com	57731243	თელავი	3	ინფორმაციული ტექნოლოგიები
1005	მაკა	ჯოგაუა	aaa@gmail.com	577324543	ბათუმი	2	ინფორმაციული სისტემები
1007	დავითი	ზოსადაძე	aaa@gmail.com	571224543	თბილისი	3	ინფორმაციული ტექნოლოგიები
1009	ლელა	დანელია	aaa@gmail.com	599324543	თბილისი	1	კომპიუტერული მეცნიერება

თუ დავაკვირდებით მოთხოვნას ვნახავთ რომ შემცირდა სტუდენტთა რაოდენობა, და შედეგში აღარ დაბრუნდა 1004, 1006 და სხვა სტუდენტები.

რა შეიძლება იყოს აღნიშნულის გამომწვევი?

როგორც ზემოთ უკვე ვთქვით, შიდა შეერთებების დროს ჩვენ ვიღებთ ცხრილების თანაკვეთას, ანუ ჩანაწერებს რომლებიც საერთოა ორივე ცხრილისთვის.

[dbo].[StudentDetails] ცხრილის გადაბმა მოხდა ON [SD].[StudentId] = [S].[StudentId] პირობით რაც გულისხმობდა რომ ყოველი სტუდენტის ნომრისთვის Students ცხრილიდან, ჩვენ უნდა გვქონოდა შესაბამისი ჩანაწერი StudentDetails ცხრილში. წინააღმდეგ შემთვევაში მოხდებოდა ჩანაწერის გამოტოვება მთხოვნიდან. თუ დავაკვირდებით ამ ორი ცხრილის ჩანაწერებს აღმოვაჩენთ რომ გამოიტოვა სტუდენტების ზუსტად ის ჩანაწერები სადაც არ გვქონდა შესაბამისობა დეტალურ ცხრილში:

The screenshot shows the SQL Server Management Studio interface with two queries and their results.

Query 1:

```
SELECT * FROM [dbo].[Students] AS [S]
```

Query 2:

```
SELECT * FROM [dbo].[StudentDetails] AS [SD]
```

Results:

StudentId	StudentName	StudentLastName	Email	DirectionId	Password
1000	ერეკლე	ბურჯაძე	aaa@gmail.com	1	NULL
1001	ნინო	ქურდაძე	aaa@gmail.com	1	NULL
1002	ანა	ბაბუა	aaa@gmail.com	2	NULL
1003	გიგო	პავლიძე	aaa@gmail.com	3	NULL
1004	ნუცა	ურიფია	aaa@gmail.com	1	NULL
1005	მავა	ჯოჯავა	aaa@gmail.com	2	NULL
1006	გიორგი	სალია	aaa@gmail.com	2	NULL
1007	დავითი	ზვიადაძე	aaa@gmail.com	3	NULL
1008	ნიკა	ცინცაძე	aaa@gmail.com	1	NULL
1009	ლელა	დანელია	aaa@gmail.com	1	NULL

StudentDetailId	StudentId	PersonalId	Photo	DateOfBirth	Phone	City
1	1001	02020104340	NULL	2000-03-04	577324543	ბათუმი
2	1000	01212010434	NULL	2001-03-04	578464543	თელავი
3	1002	01110104311	NULL	1998-01-05	578324543	ქუთაისი
4	1003	02020111240	NULL	2002-05-07	57731243	თელავი
5	1005	01320111233	NULL	2001-07-08	577324543	ბათუმი
6	1007	60000000000	NULL	2003-10-02	571224543	თბილისი
7	1009	70000000000	NULL	1997-01-03	599324543	თბილისი

გამოდის „თანაკვეთის“ პირობის არსებობის გამო მოთხოვნიდან დავკარგე სტუდენტები რომლებსაც არ ქონდათ შევსებული დეტალური ინფორმაცია. შესაბამისად მსგავსი შეერთება არ გამოგვადგება, თუკი არ გვინდა გამოვტოვოთ აღნიშნული სტუდენტები.

## LEFT JOIN მოთხოვნა

LEFT JOIN არის შეერთების ტიპი, სადაც მნიშვნელოვანია რომელი ცხრილი წერია JOIN სიტყვის მარცხნივ და რომელი მარჯვივ, ვინდაიდან იგი აბრუნებს ყველა ჩანაწერს მარცხენა ცხრილიდან და შესატყვის ჩანაწერებს მარჯვენა ცხრილიდან. აქ, მარცხენა ცხრილი ნიშნავს ცხრილს, რომელიც დგას მარცხენა მხარეს ან მოთხოვნაში "LEFT JOIN" ფრაზის წინ, ხოლო მარჯვენა ცხრილი მიუთითებს ცხრილს, რომელიც დგას მარჯვენა მხარეს "LEFT JOIN" ფრაზის შემდეგ. აღიშნულ შემთხვევაში მარცხენა ცხრილი არის პრიორიტეტული, იგი მოთხოვნაში მოდის სრულად, ხოლო მარჯვენა ცხრილიდან ივესება შესაბამისობები აღნიშნული ცხრილისთვის. ხოლო შეუსაბამო ჩანაწერისთვის მარჯვენა ცხრილიდან აბრუნებს NULL-ს. ჩვენს შემთხვევაში ზემოთ არსებული პრობლემის საპასუხოდ, თუ შიდა შეერტებას (INNER JOIN) ჩავანაცვლებთ (LEFT JOIN) მარცხენა შეერთებით, გამოვა რომ მარცხენა, სტუდენტების ცხრილი წამოვა შედეგში სრულად, ხოლო მარჯვენა დეტალურის ცხრილიდან შეივსება მხოლოდ შესაბამისი ჩანაწერები. რომელ სტუდენტსაც არ ექნება დეტაურ ცხრილში შესაბამისი ინფორმაცია, საჭირო ველები შეევსება NULL-ებით.

```
SELECT [S].[StudentId],  
       [S].[StudentName],  
       [S].[StudentLastName],  
       [S].[Email],  
       [SD].[Phone],  
       [SD].[City],  
       [S].[DirectionId]  
FROM [dbo].[Students] AS [S]  
      LEFT JOIN [dbo].[StudentDetails] AS [SD]  
        ON [SD].[StudentId] = [S].[StudentId];
```

კოდის შესრულების შედეგად, მივიღებთ შედეგს:

```

SELECT [S].[StudentId],
       [S].[StudentName],
       [S].[StudentLastName],
       [S].[Email],
       [SD].[Phone],
       [SD].[City],
       [S].[DirectionId]
  FROM [dbo].[Students] AS [S]
 LEFT JOIN [dbo].[StudentDetails] AS [SD]
    ON [SD].[StudentId] = [S].[StudentId];

```

Results	Messages					
StudentId	StudentName	StudentLastName	Email	Phone	City	DirectionId
1000	ერეკლე	ბურჯაძე	aaa@gmail.com	578464543	თელავი	1
1001	ნინო	ჭერიაძე	aaa@gmail.com	577324543	ბათუმი	1
1002	ანა	ბაბუა	aaa@gmail.com	578324543	ქუთაისი	2
1003	გიგო	პილიძე	aaa@gmail.com	57731243	თელავი	3
1004	ნურა	ურიფია	aaa@gmail.com	NULL	NULL	1
1005	მავა	ჯოგაძე	aaa@gmail.com	577324543	ბათუმი	2
1006	გორგი	სალია	aaa@gmail.com	NULL	NULL	2
1007	დავითი	ზეისადაძე	aaa@gmail.com	571224543	თბილისი	3
1008	ნიკა	ცინცაძე	aaa@gmail.com	NULL	NULL	1
1009	ლელა	დაწელია	aaa@gmail.com	599324543	თბილისი	1

როგორც ხედავთ მივიღეთ სტუდენტთა სრული სია, ხოლო ტელეფონისა და ქალაქის ველები იმ სტუდენტებისთვის რომლებსაც არ აქვთ დეტალური შევსებულია NULL მნიშვნელობით.

ზოგიერთ მონაცემთა ბაზაში მარჯვენა შეერთებას უწოდებენ LEFT OUTER JOIN.

დავუკვირდეთ მოთხოვნას, StudentId გადაბმის ველი გვხვდება ორივე ცხრილში, თუმცა ჩვენ მოთხოვნაში მას ვიყენებთ მარცხენა, სტუდენტების ცხრილიდან.

ჩავამატოთ იგივე ველი დეტალურის ცხრილიდან და უფრო თვალსაჩინო იქნება რა მომენტში მოხდა ამ ორი ცხრილის გადაბმა და რა მომენტში ვერა:

```

SELECT [S].[StudentId],
       [SD].[StudentId],
       [S].[StudentName],
       [S].[StudentLastName],
       [S].[Email],
       [SD].[Phone],
       [SD].[City],
       [S].[DirectionId]
  FROM [dbo].[Students] AS [S]
 LEFT JOIN [dbo].[StudentDetails] AS [SD]
    ON [SD].[StudentId] = [S].[StudentId];

```

შედეგად მივიღებთ:

```

SELECT [S].[StudentId],
       [SD].[StudentId],
       [S].[StudentName],
       [S].[StudentLastName],
       [S].[Email],
       [SD].[Phone],
       [SD].[City],
       [S].[DirectionId]
  FROM [dbo].[Students] AS [S]
    LEFT JOIN [dbo].[StudentDetails] AS [SD]
      ON [SD].[StudentId] = [S].[StudentId];

```

% ▾

	Results	Messages					
StudentId	StudentId	StudentName	StudentLastName	Email	Phone	City	DirectionId
1000	1000	ერეკო	ბერიძე	aaa@gmail.com	578464543	თელავი	1
1001	1001	ნინო	ჭერიძე	aaa@gmail.com	577324543	ბათუმი	1
1002	1002	ანა	ბატუა	aaa@gmail.com	578324543	ჭეთაისი	2
1003	1003	გაბო	ვაკედე	aaa@gmail.com	57731243	თელავი	3
1004	NULL	ნუცა	ურიკიძე	aaa@gmail.com	NULL	NULL	1
1005	1005	მავა	ჯოლუა	aaa@gmail.com	577324543	ბათუმი	2
1006	NULL	გიორგი	სალია	aaa@gmail.com	NULL	NULL	2
1007	1007	დავითი	ზვანაძე	aaa@gmail.com	571224543	თბილისი	3
1008	NULL	ნუა	ცინცაძე	aaa@gmail.com	NULL	NULL	1
1009	1009	ლეთა	დამელია	aaa@gmail.com	599324543	თბილისი	1

### RIGHT JOIN მოთხოვნა

RIGHT JOIN წინამორბედის მსგავსად არის შეერთების ტიპი, სადაც მნიშვნელოვანია რომელი ცხრილი წერია JOIN სიტყვის მარცხნივ და რომელი მარჯვნივ, ვინდაიდან იგი აბრუნებს ყველა ჩანაწერს მარჯვენა ცხრილიდან და შესატყვის ჩანაწერებს მარცხენადან. აღიმუშავ შემთხვევაში უკვე მარჯვენა ცხრილი არის პრიორიტეტული, იგი მოთხოვნაში მოდის სრულად, ხოლო მარცხენა ცხრილიდან ივესბა შესაბამისი ველები აღნიშნული ცხრილისთვის. ხოლო შეუსაბამო ჩანაწერისთვის მარცხენა ცხრილიდან აბრუნებს NULL-ს.

დავწეროთ მოთხოვნა რომელიც დააბრუნებს ყველა სტუდენტს და საგნებს რომლებზეც რეგისტრირებული არიან ისინი:

```

SELECT [S].[StudentId],
       [S].[StudentName],
       [S].[StudentLastName],
       [SS].[StudentId],
       [SS].[SubjectId],
       [SS].[RegisterDate],
       [SJ].[SubjectId],
       [SJ].[SubjectName]

```

```

FROM [dbo].[Students] AS [S]
    LEFT JOIN [dbo].[StudentSubjects] AS [SS] ON [SS].[StudentId] =
[S].[StudentId]
        RIGHT JOIN [dbo].[Subjects] AS [SJ] ON [SJ].[SubjectId] =
[SS].[SubjectId]

```

დავიწყეთ სტუდენტების ცხრილიდან და გადავედით სტუდენტების და საგნების შემართებელ ცხრილზე მარცხენა შეერთებით, რათა აგვეღო ყველა სტუდენტის რეგისტრაციის მონაცემი, როგორც ხედავთ სრულადაა შევსებული ყველა სტუდენტზე საგნები, რაც ნიშნავს რომ ყველა სტუდენტი არის რეგისტრირებული ერთ საგანზე მაინც, შემდეგ კი შეალედური ცხრილიდან მარჯვენა შეერთებით გადავედით საგნების ცხრილზე, რათა პრიორიტეტი მიგვენიშებინა მარჯვენა ცხრილითვის და წამოგვეღო ყველა საგანი იმისდა მიუხედავად იყო თუ არა მასზე ვინმე რეგისტრირებული. ამ შემთხვევაში აღმოჩნდა რომ ქსელებზე არავინ არაა რეგისტრირებული.

```

SELECT [S].[StudentId],
       [S].[StudentName],
       [S].[StudentLastName],
       [SS].[StudentId],
       [SS].[SubjectId],
       [SS].[RegisterDate],
       [SJ].[SubjectId],
       [SJ].[SubjectName]
  FROM [dbo].[Students] AS [S]
    LEFT JOIN [dbo].[StudentSubjects] AS [SS] ON [SS].[StudentId] = [S].[StudentId]
        RIGHT JOIN [dbo].[Subjects] AS [SJ] ON [SJ].[SubjectId] = [SS].[SubjectId]

```

Results							
StudentId	StudentName	StudentLastName	StudentId	SubjectId	RegisterDate	SubjectId	SubjectName
1000	ერეკ	ბურჯაძე	1000	1	2023-04-10 11:08:09.497	1	ნეირონებული ქსელები
1001	ნინო	ჭურაძე	1001	1	2023-04-10 11:08:09.497	1	ნეირონებული ქსელები
1007	დავით	ზუსადაძე	1007	1	2023-04-10 11:08:09.497	1	ნეირონებული ქსელები
1001	ნინო	ჭურაძე	1001	2	2023-04-10 11:08:09.497	2	პროექტი
1002	ანა	გაბრი	1002	2	2023-04-10 11:08:09.497	2	პროექტი
1003	გიორგი	პაფოძე	1003	2	2023-04-10 11:08:09.497	2	პროექტი
1007	დავით	ზუსადაძე	1007	2	2023-04-10 11:08:09.497	2	პროექტი
1009	ლილი	დაწერია	1009	2	2023-04-10 11:08:09.497	2	პროექტი
1000	ერეკ	ბურჯაძე	1000	3	2023-04-10 11:08:09.497	3	ალორითიმები
1001	ნინო	ჭურაძე	1001	3	2023-04-10 11:08:09.497	3	ალორითიმები
1005	მარა	აზოვა	1005	3	2023-04-10 11:08:09.497	3	ალორითიმები
NULL	NULL	NULL	NULL	NULL	NULL	4	ქსელები
1000	ერეკ	ბურჯაძე	1000	5	2023-04-10 11:08:09.497	5	NLP
1003	გიორგი	პაფოძე	1003	5	2023-04-10 11:08:09.497	5	NLP
1007	დავით	ზუსადაძე	1007	5	2023-04-10 11:08:09.497	5	NLP
1001	ნინო	ჭურაძე	1001	6	2023-04-10 11:08:09.497	6	დაპროგრამება
1005	მარა	აზოვა	1005	6	2023-04-10 11:08:09.497	6	დაპროგრამება
1002	ანა	გაბრი	1002	7	2023-04-10 11:08:09.497	7	მატებური სწავლება
1003	გიორგი	პაფოძე	1003	7	2023-04-10 11:08:09.497	7	მატებური სწავლება

### FULL JOIN მოთხოვთ

FULL JOIN აბრუნებს ყველა ჩანაწერს ყველა მითითებული ცხრილიდან. ნებისმიერი შეუსაბამო ჩანაწერისთვის მარჯვენა ცხრილიდან ინება იგი თუ მარცხენა ცარიელი ველები ივსება NULL-ით.

ზოგიერთ მონაცემთა ბაზაში FULL JOIN ეწოდება FULL OUTER JOIN. მას შეუძლია დააბრუნოს ძალიან დიდი შედეგი, ვინაიდან იგი აბრუნებს ყველა სტრიქონს ყველა ცხრილიდან.

წავუშალოთ საგან კავშირი მიმართულებასთან (FOREIGN KEY) აღნიშნული საშუალებას მოგვცემს საგანი დავარეგისტრიროთ არარსებულ მიმართულებაზე. ამის შემდეგ დავამატოთ ერთი ახალი მიმართულება, რომელზეც არ გვექნება საგანი, და ორიც ახალი საგანი არარსებულ მიმართულებაზე.

```
INSERT INTO [dbo].[Directions]
([DirectionName])
VALUES
(N'ფიზიკა');
```

```
insert into Subjects
values
```

```
(N'ლიტერატურა',6, 10)
,(N'ფერწერა',6,5)
```

დავწეროთ სრული შეერთების მოთხოვნა:

```
SELECT [S].[SubjectId],
[S].[SubjectName],
[S].[DirectionId],
[D].[DirectionId],
[D].[DirectoryName]
FROM [dbo].[Subjects] AS [S]
FULL JOIN [dbo].[Directions] AS [D]
ON [D].[DirectionId] = [S].[DirectionId]
```

შედეგად მივიღებთ ყველა ველს, ორივე ცხრილიდან, მარცხენა ცხრილი სრულად, თუ კი რომელიმე ჩანაწერს არ ჰქონდა შესაბამისობა მარჯვენა ცხრილიდან შესაბამისი ველები შეივსო NULL-ით. ანალოგიურად სრულად ვიღებთ მარჯვენა ცხრილსაც და თუ კი რომელიმე ჩანაწერს არ ჰქონდა შესაბამისობა მარცხენა ცხრილიდან შესაბამისი ველები შეივსო NULL-ით.

```

SELECT [S].[SubjectId],
       [S].[SubjectName],
       [S].[DirectionId],
       [D].[DirectionId],
       [D].[DirectoryName]
  FROM [dbo].[Subjects] AS [S]
 FULL JOIN [dbo].[Directions] AS [D]
    ON [D].[DirectionId] = [S].[DirectionId]

```

The screenshot shows a SQL query being run in a database management system. The query performs a full join between the Subjects and Directions tables. The results are displayed in a grid:

SubjectId	SubjectName	DirectionId	DirectionId	DirectoryName
1	ნეირონული ქსელები	3	3	ინფორმაციული ტექნოლოგიები
2	პროექტი	1	1	კომპიუტერული მეცნიერება
3	ალგორითმები	2	2	ინფორმაციული სისტემები
4	ქსელები	3	3	ინფორმაციული ტექნოლოგიები
5	NLP	1	1	კომპიუტერული მეცნიერება
6	დაპროგრამება	2	2	ინფორმაციული სისტემები
7	მანქანური სწავლება	1	1	კომპიუტერული მეცნიერება
8	ლიტერატურა	6	NULL	NULL
9	ფერწერა	6	NULL	NULL
NULL	NULL	NULL	4	ფიზიკა

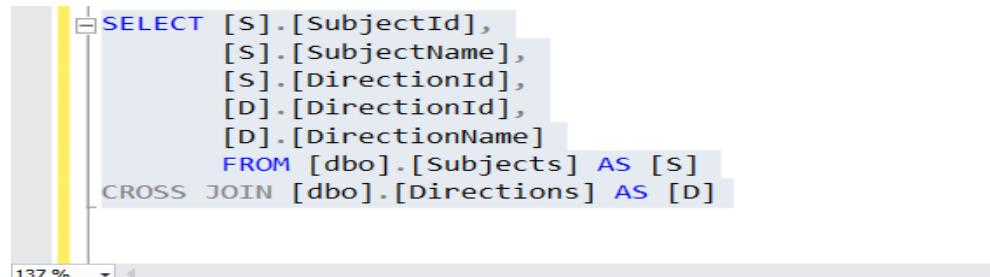
სურათზე ვხედავთ რომ საგნებს (ლიტერატურა და ფერწერა) რომლებსაც არ ქონდათ შესაბამისი მიმართულება და ფიზიკის მიმართულებას რომელსაც არ ჰქონდა საგნები შესაბამისი ველები შეევსო NULL-ით.

## CROSS JOIN მოთხოვნა

ჯვარედინი შეერთების ტიპი, როდესაც ორ ცხრილს არ აქვს საერთო ველი და თითეულ სვეტს ერთი ცხრილიდან შეესაბამება ყველა სვეტი მეორედან, ვინაიდან არ არსებობს გადაბმის On პირობა:

```
SELECT [S].[SubjectId],
       [S].[SubjectName],
       [S].[DirectionId],
       [D].[DirectionId],
       [D].[DirectoryName]
  FROM [dbo].[Subjects] AS [S]
CROSS JOIN [dbo].[Directions] AS [D]
```

თუკი გვაქვს 4 მიმართულება და 9 საგანი შედეგად ვიღებთ  $4 \times 9 = 36$  სტრიქონს.



The screenshot shows the SQL query being run in SSMS. The query is:

```
SELECT [S].[SubjectId],
       [S].[SubjectName],
       [S].[DirectionId],
       [D].[DirectionId],
       [D].[DirectoryName]
  FROM [dbo].[Subjects] AS [S]
CROSS JOIN [dbo].[Directions] AS [D]
```

The results grid displays 36 rows of data, representing all combinations of subjects and directions. The columns are:

	SubjectId	SubjectName	DirectionId	DirectionId	DirectoryName
1	1	ნეირონული ქსელები	3	1	კომპიუტერული მეცნიერება
2	2	პროექტი	1	1	კომპიუტერული მეცნიერება
3	3	ალგორითმები	2	1	კომპიუტერული მეცნიერება
4	4	ქსელები	3	1	კომპიუტერული მეცნიერება
5	5	NLP	1	1	კომპიუტერული მეცნიერება
6	6	დაპროგრამება	2	1	კომპიუტერული მეცნიერება
7	7	მანქანური სწავლება	1	1	კომპიუტერული მეცნიერება
8	8	ლიტერატურა	6	1	კომპიუტერული მეცნიერება
9	9	ფერწერა	6	1	კომპიუტერული მეცნიერება
10	1	ნეირონული ქსელები	3	2	ინფორმაციული სისტემები
11	2	პროექტი	1	2	ინფორმაციული სისტემები
12	3	ალგორითმები	2	2	ინფორმაციული სისტემები
13	4	ქსელები	3	2	ინფორმაციული სისტემები
14	5	NLP	1	2	ინფორმაციული სისტემები
15	6	დაპროგრამება	2	2	ინფორმაციული სისტემები
16	7	მანქანური სწავლება	1	2	ინფორმაციული სისტემები
17	8	ლიტერატურა	6	2	ინფორმაციული სისტემები
18	9	ფერწერა	6	2	ინფორმაციული სისტემები
19	1	ნეირონული ქსელები	3	3	ინფორმაციული ტექნოლოგიები
20	2	პროექტი	1	3	ინფორმაციული ტექნოლოგიები
21	3	ალგორითმები	2	3	ინფორმაციული ტექნოლოგიები
22	4	ქსელები	3	3	ინფორმაციული ტექნოლოგიები
23	5	NLP	1	3	ინფორმაციული ტექნოლოგიები
24	6	დაპროგრამება	2	3	ინფორმაციული ტექნოლოგიები
25	7	მანქანური სწავლება	1	3	ინფორმაციული ტექნოლოგიები
26	8	ლიტერატურა	6	3	ინფორმაციული ტექნოლოგიები
27	9	ფერწერა	6	3	ინფორმაციული ტექნოლოგიები
28	1	ნეირონული ქსელები	3	4	ფიზიკა
29	2	პროექტი	1	4	ფიზიკა
30	3	ალგორითმები	2	4	ფიზიკა
31	4	ქსელები	3	4	ფიზიკა
32	5	NLP	1	4	ფიზიკა
33	6	დაპროგრამება	2	4	ფიზიკა
34	7	მანქანური სწავლება	1	4	ფიზიკა
35	8	ლიტერატურა	6	4	ფიზიკა
36	9	ფერწერა	6	4	ფიზიკა

აღნიშნული შედეგი მიიღებოდა პირდაპირ ცხრილების მძიმით გამოყოფისას From პირობაში Where -ის გარეშე.

```
SELECT [S].[SubjectId],  
       [S].[SubjectName],  
       [S].[DirectionId],  
       [D].[DirectionId],  
       [D].[DirectoryName]  
  FROM [dbo].[Subjects] AS [S],[dbo].[Directions] AS [D]
```

0

```
CREATE DATABASE [Faculty];
GO

USE [Faculty];

CREATE TABLE [Directions]
(
    [DirectionId] INT NOT NULL PRIMARY KEY IDENTITY,
    [DirectoryName] NVARCHAR(150) NOT NULL
        UNIQUE,
    [DirectionHead] INT
        NULL
);

CREATE TABLE [Students]
(
    [StudentId] INT IDENTITY(1000,1)
        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
        FOREIGN KEY REFERENCES [dbo].[Directions] ([DirectionId]),
    [Password] VARBINARY(250)
);

CREATE TABLE [StudentDetails]
(
    [StudentDetailId] INT IDENTITY PRIMARY KEY,
    [StudentId] INT
        UNIQUE
        FOREIGN KEY REFERENCES [dbo].[Students] ([StudentId]),
    [PersonalId] VARCHAR(11)
        UNIQUE,
    [Photo] VARBINARY(MAX),
    [DateOfBirth] DATE,
    [Phone] VARCHAR(20),
    [City] NVARCHAR(50)
        DEFAULT (N'თბილისი'),
);

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
);

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)
);

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 '%@%')
);

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]
    )
);

INSERT INTO [dbo].[Directions]
([DirectoryName])
VALUES
(N'კომპიუტერული მეცნიერება'),
(N'ინფორმაციული სისტემები'),
(N'ინფორმაციული ტექნოლოგიები');

insert into Subjects
values
(N'პროექტი', 1, 10)
,(N'ალგორითმები', 2, 5)
,(N'ქსელები', 3, 5)
,(N'NLP', 1, 5)
,(N'დაპროგრამება', 2, 6)
,(N'მანქანური სწავლება', 1, 3)
,(N'ნეირონული ქსელები', 3, 5);

INSERT INTO [dbo].[Students]

```

```

(
    [StudentName],
    [StudentLastName],
    [Email],
    [DirectionId]
)
VALUES
(N'ერეკლე', N'ბურგაძე', 'aaa@gmail.com', 1),
(N'ნინო', N'ქურდაძე', 'aaa@gmail.com', 1),
(N'ანნა', N'გაგუა', 'aaa@gmail.com', 2),
(N'გიგო', N'პაივიძე', 'aaa@gmail.com', 3),
(N'ნუცა', N'ურიდია', 'aaa@gmail.com', 1),
(N'მაკა', N'ჯოჯუა', 'aaa@gmail.com', 2),
(N'გიორგი', N'სალია', 'aaa@gmail.com', 2),
(N'დავითი', N'ზვიადაძე', 'aaa@gmail.com', 3),
(N'ნიკა', N'ცინცაძე', 'aaa@gmail.com', 1),
(N'ლელა', N'დანელია', 'aaa@gmail.com', 1);

INSERT INTO [dbo].[StudentDetails]
(
[StudentId],
[DateOfBirth],
[PersonalId],
[Phone],
[City]
)
VALUES

(1001, '3/4/2000', '02020104340', '577324543', N'ბათუმი'),
(1000, '3/4/2001', '01212010434', '578464543', N'თელავი'),
(1002, '1/5/1998', '01110104311', 'N'578324543', N'ქუთაისი'),
(1003, '5/7/2002', '02020111240', 'N'57731243', N'თელავი'),
(1005, '7/8/2001', '01320111233', 'N'577324543', N'ბათუმი'),
(1007, '10/2/2003', '60000000000', 'N'571224543', DEFAULT),
(1009, '1/3/1997', '70000000000', 'N'599324543', DEFAULT);

INSERT INTO [dbo].[Lecturers]
(
[LecturerName],
[LecturerLastName],
[Phone],
[Email]
)
VALUES
( N'თამარ', N'იარაული', '577238432', 't.ijarauli@gmail.com' ),
( N'გიორგი', N'სამხარაძე', '598723432', 'g.samkharadze@gmail.com' ),
( N'ლევან', N'ცაბაძე', '585237433', 'l.tsabadze@gmail.com' ),
( N'ია', N'კილასონია', '511986098', 'i.kilasonia@gmail.com' ),
( N'თამთა', N'ბერიძე', '5772301293', 't.beridze@gmail.com' ),
( N'დავით', N'ბურდული', '555128712', 'd.burduli@gmail.com' )

```

```
INSERT INTO [dbo].[SubjectLecturers]
(
    [SubjectId],
    [LecturerId]
)
VALUES
(
    1, 1
),
(
    1, 3
),
(
    2, 2
),
(
    4, 6
),
(
    5, 3
),
(
    6, 4
),
(
    7, 4
),
(
    7, 5
)

INSERT INTO [dbo].[StudentSubjects]
(
    [StudentId],
    [SubjectId],
    [RegisterDate],
    [IsPassed]
)
VALUES
(
    1000, 1, GETDATE(), 0),
(
    1000, 3, GETDATE(), 0),
(
    1000, 5, GETDATE(), 0),
(
    1001, 2, GETDATE(), 0),
(
    1001, 3, GETDATE(), 0),
(
    1001, 6, GETDATE(), 0),
(
    1001, 1, GETDATE(), 0),
(
    1002, 2, GETDATE(), 0),
(
    1002, 7, GETDATE(), 0),
(
    1003, 7, GETDATE(), 0),
(
    1003, 5, GETDATE(), 0),
(
    1003, 2, GETDATE(), 0),
(
    1005, 3, GETDATE(), 0),
(
    1005, 6, GETDATE(), 0),
(
    1007, 1, GETDATE(), 0),
(
    1007, 2, GETDATE(), 0),
(
    1007, 5, GETDATE(), 0),
(
    1009, 2, GETDATE(), 0)
```

## ლაბორატორია 8

### პროცედურები

SQL Server-ში შენახული პროცედურა (Stored Procedure) არის T-SQL ბრძანებების ნაკრები, რომელიც ინახება მონაცემთა ბაზაში. შენახული პროცედურა იღებს შემავალ და გამომავალ პარამეტრებს, ასრულებს შესაბმის SQL ოპერატორებს და აბრუნებს შედეგების კომპლექტს, ასეთის არსებობის შემთხვევაში.

ნაგულისხმევად, პირველი შესრულების შემდეგ ითვლება რომ შენახული პროცედურა შედგა. თუმცა იგი ინახება ბაზაში და წარმოადგენს შესრულების გეგმას, რომელიც გამოიყენება იგივე მოქმედების განმეორებით უფრო სწრაფი შესრულებისთვის.

შენახული პროცედურები ორი ტიპისაა:

**მომხმარებლის მიერ განსაზღვრული პროცედურები (User-defined Procedures):** მომხმარებლის მიერ განსაზღვრული შენახული პროცედურა იქმნება მომხმარებლის მიერ განსაზღვრულ მონაცემთა ბაზაში ან ნებისმიერი სისტემურ მონაცემთა ბაზაში, გარდა რესურსების მონაცემთა ბაზისა.

**სისტემის პროცედურები (System procedures):** სისტემის პროცედურები წარმოადგენ SQL Server-ის ნაწილს. ისინი ფიზიკურად ინახება რესურსების შიდა, ფარულ მონაცემთა ბაზაში და ლოგიკურად გამოტანილია ყველა მონაცემთა ბაზის `sys` სქემაში. სისტემაში შენახული პროცედურები იწყება პრეფიქსით `sp_`.

შენახული პროცედურის შექმნა

გამოიყენეთ CREATE განცხადება შენახული პროცედურის შესაქმნელად.

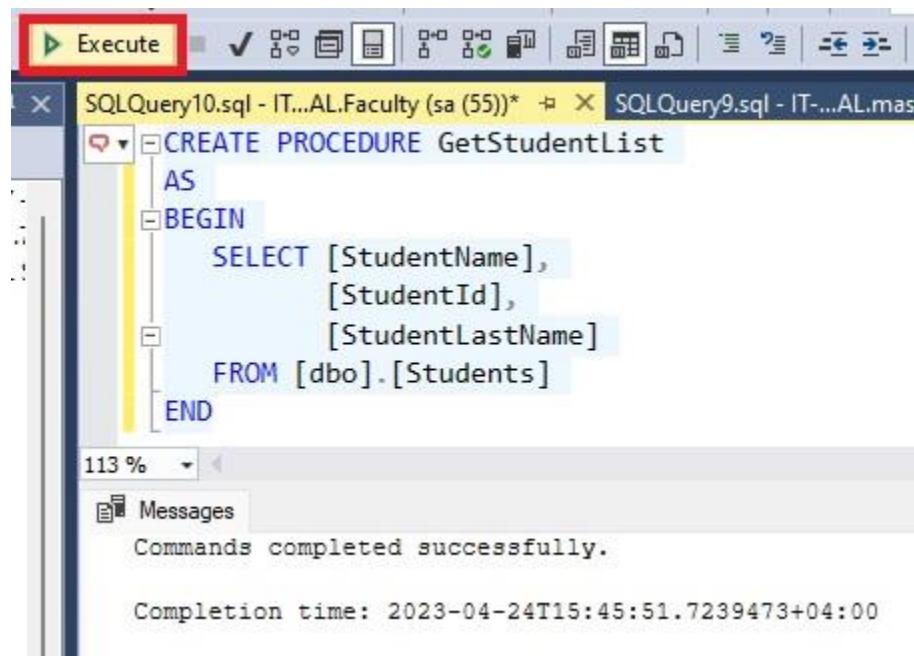
სინტაქსი:

```
CREATE [OR ALTER] {PROC | PROCEDURE} [schema_name.]  
procedure_name([@parameter data_type [ OUT | OUTPUT |  
[READONLY]]  
[ WITH <procedure_option> ]  
[ FOR REPLICATION ]  
AS  
BEGIN  
    sql_statements  
END
```

შენახული პროცედურა შეიძლება შეიცავდეს ერთ ან მეტ ბრძანებას, როგორებიცაა მოთხოვნა, ჩასმა, განახლება ან წაშლა. ქვემოთ მოცემულია მარტივი შენახული პროცედურის მაგალითი, რომელიც აბრუნებს ჩანაწერებს ცხრილიდან SELECT მოთხოვნის გამოყენებით.

```
CREATE PROCEDURE GetStudentList
AS
BEGIN
    SELECT [StudentName],
           [StudentId],
           [StudentLastName]
    FROM [dbo].[Students]
END
```

გავუშვათ ზემოთ მოყვანილი T-SQL სკრიპტი შეკითხვის (Query) რედაქტორში, რათა შეადგინოთ და შექმნათ თქვენი პირველი შენახული პროცედურა მონაცემთა ბაზაში:



ზემოთ შექმნილი პროცედურა შეიძლება გამოვიძახოთ EXEC საბრძანებო სიტყვის გამოყენებით, როგორც ეს ნაჩვენებია ქვემოთ:

SQLQuery10.sql - IT...AL.Faculty (sa (55))\*

**EXEC GetStudentList**

113 %

Results Messages

	StudentName	StudentId	StudentLastName
1	ერეკლი	1000	ბერკაძე
2	ნინო	1001	ჭურდაძე
3	ანნა	1002	ბაბუა
4	გიგო	1003	პავლიძე
5	ნუცა	1004	ურიფისა
6	მაკა	1005	ჯოჯუა
7	დავითი	1007	ზვანაძე
8	ნიკა	1008	ცინცაძე
9	ლელა	1009	დაწელია

იმისთვის რომ პროცედურებმა შეასრულონ სხსვადასხვა მოქმედებები მათ სჭირდებათ პარამეტრები, ზემოთ განხილული პროცედურა იყო უპარამეტრო, შესაბამისად იგი გამოძახების დროს არ ითხოვდა ჩვენგან რაიმე მონაცემს, თუმცა თუ მოვინდომებთ შევქმნათ პროცედურა რომელიც მაგალითად ჩაწერს მონაცემებს ცხრილში, აუცილებლად მოგვიწევს პარამეტრების სახით გავადავცეთ მონაცემები რომელთა ჩაწერაც გვინდა ცხრილში. პროცედურას შეიძლება ჰქონდეს ერთი, რამოდენიმე ან სულაც არცერთი პარამეტრი(როგორც უკვე ვნახეთ) თითოეულ პარამეტრს აქვს საკუთარი სახელი და პრეფიქსი @, ასევე აუცილებელია თითოეულ პარამეტრს ჰქონდეს საკუთარი ტიპი (ჩვენთვის უკვე ნაცნობი მონაცემთა ტიპებიდან). ასეთი პარამეტრები თავსდება პროცედურის თავში (ქუდში) და გამოყოფილია მძიმით.

შენახული პროცედურის პარამეტრები: შემავალი გამომავალი, გაჩუმების პრინციპით:

- შენახულ პროცედურას შეიძლება ჰქონდეს არცერთი, ერთი ან მეტი INPUT (შემავალი) და OUTPUT(გამომავალი) პარამეტრი.
- შენახულ პროცედურას შეიძლება ჰქონდეს მითითებული მაქსიმუმ 2100 პარამეტრი.
- თითოეულ პარამეტრს ენიჭება სახელი, მონაცემთა ტიპი და მიმართულება, (შემავალია თუ გამომავალი) თუ მიმართულება არ არის მითითებული, მაშინ ნაგულისხმევად პარამეტრი არის არის შემავალი.
- თქვენ შეგიძლიათ მიუთითოთ ნაგულისხმევი მნიშვნელობა პარამეტრებისთვის.
- შენახულ პროცედურებს შეუძლიათ დააბრუნონ მნიშვნელობა პროგრამაში, თუ პარამეტრი მითითებულია როგორც OUTPUT.

- პარამეტრის მნიშვნელობები უნდა იყოს მუდმივი ან ცვლადი. არ შეიძლება მნიშვნელობა იყოს ფუნქციის სახელი.
- პარამეტრის ცვლადები შეიძლება იყოს მომხმარებლის მიერ განსაზღვრული ან სისტემის ცვლადები, როგორიცაა @spid

თუ გვსურს პროცედურის შექმნა რომელიც ჩვენს ნაცნობ Students ცხრილში შექმნის ახალ ჩანაწერს, ანუ გვსურს ახალი სტუდენტის დამატების პროცესის ავტომატიზაცია, აუცილებელია ასეთ პროცედურას როგორც მინიმუმ გადავცეთ აღნიშნული ცხრილის სავალდებულო ველების შესავსებად საჭირო ინფორმაცია: ჩვენს შემთხვევაში დაგვჭირდება პარამტერები:

```
@StudentName NVARCHAR(30),
@StudentLastName NVARCHAR(30),
@email VARCHAR(30),
@DirectionId INT
```

სტუდენტის სახელი, გვარი, მეილი და მიმართულების ვალიდური (ბაზაში არსებული) ნომერი. გაითვალისწინეთ რომ ცვლადების სახელები შეიძლება შეარჩიოთ ნებისმიერი სასურველი წესით, ჩვენს შემთხვევაში ვირჩევთ შესავსები ველების იდენტურ დასახელებებს რათა შემდგომ პროცედურის ტანში ადვილად მოხდეს დიფერენცირება რომელი ცვლადი რომელი პარამეტრისთვისაა.

როგორც უკვე აღვნიშნეთ სახელები შეიძლება იყოს ნებისმიერი (გარდა დუბლირებულისა) მაგრამ დამეთანხმებით ქვემოთ მოყვანილი სკრიპტის პარამეტრების იდენტიფიცირება შემდეგ კოდში ბევრად უფრო რთული იქნება:

```
@A NVARCHAR(30),
@B NVARCHAR(30),
@C VARCHAR(30),
@D INT
```

### პარამეტრის სახელები

- შენახული პროცედურის პარამეტრების სახელები უნდა დაიწყოს ერთი @-ით.
- სახელი უნდა იყოს უნიკალური შენახული პროცედურის ფარგლებში.
- თუ პარამეტრის მნიშვნელობები გადაეცემა როგორც @Param1 = value1, @Param2 = value2, როგორც ნაჩვენებია ზემოთ მოცემულ მაგალითში, მაშინ პარამეტრების გადაცემა შესაძლებელია ნებისმიერი თანმიმდევრობით.
- თუ ერთი პარამეტრი მოწოდებულია როგორც @param1 = მნიშვნელობა, მაშინ ყველა პარამეტრი უნდა იყოს მიწოდებული იმავე წესით.

სახელებისგან განსხვავებით ცვლადების მონაცემთა ტიპის არჩევა სურვილისამებრ არ ხდება. აუცილებელია მონაცემთა ტიპი ემთხვეოდეს ცხრილში არსებულ ტიპს, ვინაიდან თუ ცხრილში მონაცემი მთელი რიცხვია, პარამეტრი რომელიც მას ენიჭება ასევე უნდა იყოს მთელი რიცხვი.

პარამეტრების დასრულების შემდგომ სულდება პროცედურის თავი და იწყება ტანი **AS** საბრძანებო სიტყვის შემდგომ. აქვე ვახსენოთ **BEGIN** და **END** ოპერატორები რომლებიც SQL ში წარმოადგენენ კვიგურულ ფრჩხილებს შესაბამისად ქმნიან დირექტივების ბლოკს. როგორც ვიცით, ფიგურული ფრჩხილების მსგავსად თუ პროცედურის ტანში გვაქვს მხოლოდ ერთი ოპერატორი სავალდებულო არაა **BEGIN** და **END** ოპერატორების გამოყენება, მაგრამ თუ პროცედურა შედგება რამოდენიმე მოქმედებისგან აუცილებელია მათი **BEGIN** და **END** ოპერატორებით შემოსაზღვრა.

შემდეგი შენახული პროცედურა სვავს (Insert) მნიშვნელობებს ცხრილში Students:

```
CREATE PROCEDURE [dbo].[AddStudent]
```

```
@StudentName NVARCHAR(30),  
@StudentLastName NVARCHAR(30),  
@Email VARCHAR(30),  
@DirectionId INT
```

```
AS
```

```
BEGIN
```

```
    INSERT INTO [dbo].[Students]
```

```
(
```

```
[StudentName],  
[StudentLastName],  
[Email],  
[DirectionId]
```

```
)
```

```
VALUES
```

```
(@StudentName,  
    @StudentLastName,  
    @Email,  
    @DirectionId);
```

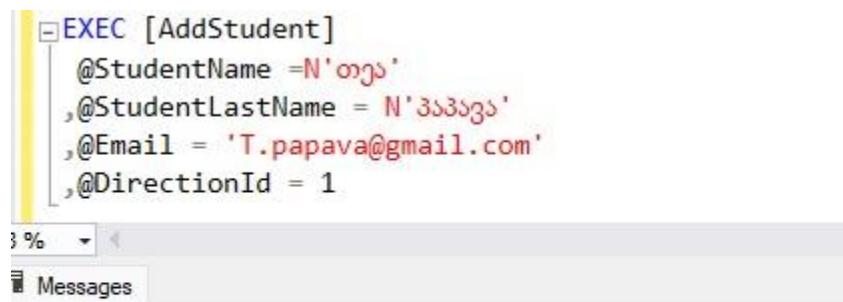
```
END;
```

აღნიშნული შენახული პროცედურა შეიძლება გამოყენებულ იქნას ცხრილში მნიშვნელობების ჩასასმელად **INSERT** ბრძანების ნაცვლად. მნიშვნელობები გადაეცემა შენახულ პროცედურას პარამეტრების სახით. სიმბოლო **@** როგორც უკვე ვთქვით გამოიყენება როგორც პარამეტრის ცვლადების პრეფიქსი.

შეგვიძლია გამოვიძახოთ AddStudent შენახული პროცედურა კვლავ EXEC საკვანძო სიტყვის გამოყენებით, იმ განსხვავებით რომ ამჯერად ჩვენი პროცედურა შეიცავს პარამეტრებს და შესაბამისად მისი ყოველი გამოძახების დროს უნდა მოვახდინოთ აღნიშული პარამეტრების ინიციალიზაცია:

```
EXEC [AddStudent]
    @StudentName =N'თეა'
    ,@StudentLastName = N'პაპავა'
    ,@Email = 'T.papava@gmail.com'
    ,@DirectionId = 1
```

პროცედურის შესრულებისას მიუთითოთ თითოეული პარამეტრი:



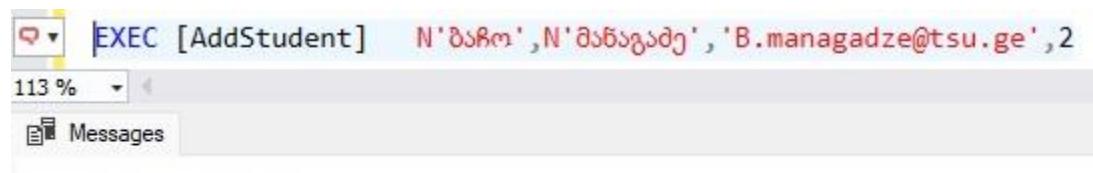
```
EXEC [AddStudent]
    @StudentName =N'თეა'
    ,@StudentLastName = N'პაპავა'
    ,@Email = 'T.papava@gmail.com'
    ,@DirectionId = 1
```

(1 row affected)

Completion time: 2023-04-24T16:17:05.2356284+04:00

თუ პროცედურის გამოძახებისას გადავცემთ ყველა პარამეტრს, და მათი გადაცემის თანმიმდევრობაც დაცულია, შეგვიძლია ცვლადი პარამეტრების სახელები აღარ მივუთითოთ:

```
EXEC [AddStudent]      N'ბაჩო',N'მანაგაძე','B.managadze@tsu.ge',2
```



```
EXEC [AddStudent]      N'ბაჩო',N'მანაგაძე','B.managadze@tsu.ge',2
```

(1 row affected)

Completion time: 2023-04-24T16:20:08.2300955+04:00

შენახული პროცედურის ნახვა

პროცედურების დასათვალირებლად გამოიყენეთ `sp_help` ან `sp_helptext` ბრძანებები:

```
sp_helptext AddStudent
```

SQLQuery10.sql - IT...AL.Faculty (sa (55)) \* → X

sp\_helptext AddStudent

113 %

Results Messages

	Text
1	CREATE PROCEDURE [dbo].[AddStudent]
2	@StudentName NVARCHAR(30),
3	@StudentLastName NVARCHAR(30),
4	@Email VARCHAR(30),
5	@DirectionId INT
6	AS
7	BEGIN
8	INSERT INTO [dbo].[Students]
9	(
10	[StudentName],
11	[StudentLastName],
12	[Email],
13	[DirectionId]
14	)
15	VALUES
16	(@StudentName,
17	@StudentLastName,
18	@Email,
19	@DirectionId);
20	END:

```
sp_help AddStudent
```

SQLQuery10.sql - IT...AL.Faculty (sa (55)) \* → X SQLQuery9.sql - IT...AL.master (sa (54)) SQLQuery189

sp\_help AddStudent

113 %

Results Messages

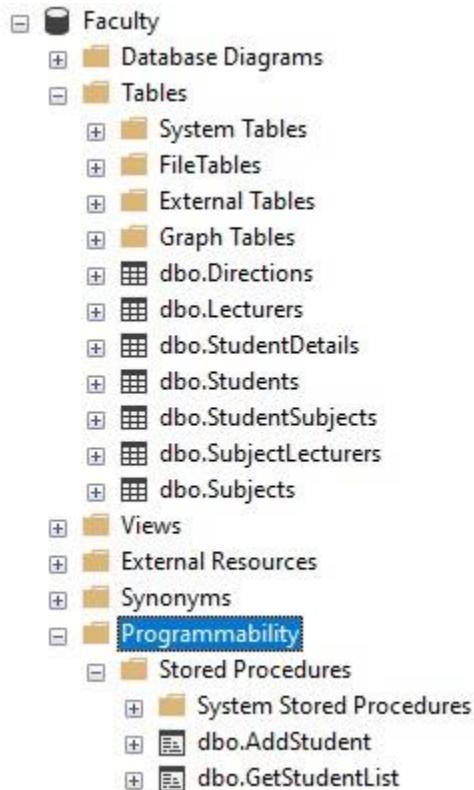
	Name	Owner	Type	Created_datetime
1	AddStudent	dbo	stored procedure	2023-04-24 16:15:56.780

	Parameter_name	Type	Length	Prec	Scale	Param_order	Collation
1	@StudentName	nvarchar	60	30	NULL	1	SQL_Latin1_General_CI_AS
2	@StudentLastName	nvarchar	60	30	NULL	2	SQL_Latin1_General_CI_AS
3	@Email	varchar	30	30	NULL	3	SQL_Latin1_General_CI_AS
4	@DirectionId	int	4	10	0	4	NULL

შედეგად შესაძლებელია დაათვალიეროთ თავად პროცედურის სკრიპტი ნახოთ როდის შეიქმნა, რა პარამეტრები აქვს და ა.შ

ყველა შენახული პროცედურა განთავსებულია მონაცემთა ბაზის Programmability > Stored Procedures საქაღალდეში.



## შენახული პროცედურის შეცვლა

ALTER PROCEDURE ბრძანება გამოიყენება შენახული პროცედურის შესაცვლელად.

```
ALTER PROCEDURE [dbo].[GetStudentList]
AS
BEGIN
    SELECT [StudentName],
           [StudentId] + ' ' + [StudentLastName] AS FullName,
           [Email]
    FROM [dbo].[Students]
END
```

ALTER PROCEDURE ბრძანებით ხდება პროცედურის ტანის ნებისმიერი ცვლილება. გაითვალისწინეთ რომ უნდა დაწეროთ არა მხოლოდ შესაცვლელი ნაწილი არამედ პროცედურის სრული ტანი ხელახლა (ცვლილების ჩათვლით). ხოლო პროცედურის

ისევე როგორ ბაზის სხვა ობიექტების სახელის გადასარქმევად გამოიყენება სისტემური პროცედურა:

```
sp_rename 'GetStudentList', 'GetStudents'
```

შენახული პროცედურის წაშლა

DROP PROCEDURE ბრძანება გამოჯიყენება შენახული პროცედურის წასაშლელად.

```
DROP PROCEDURE dbo.GetStudents;
```

OUTPUT პარამეტრები

OUTPUT პარამეტრი გამოიყენება, როდესაც გსურთ დააბრუნოთ გარკვეული მნიშვნელობა შენახული პროცედურისგან. პროცედურის გამოძახებისას პროგრამამ ასევე უნდა გამოიყენოს OUTPUT საკვანძო სიტყვა.

შემდეგი შენახული პროცედურა შეიცავს INPUT და OUTPUT პარამეტრს:

```
ALTER PROCEDURE [dbo].[AddStudent]
```

```
    @StudentName NVARCHAR(30),
    @StudentLastName NVARCHAR(30),
    @Email VARCHAR(30),
    @DirectionId INT,
    @IsAdded BIT output
```

```
AS
```

```
BEGIN
```

```
    INSERT INTO [dbo].[Students]
```

```
(
```

```
    [StudentName],
    [StudentLastName],
    [Email],
    [DirectionId]
```

```
)
```

```
VALUES
```

```
(@StudentName,
    @StudentLastName,
    @Email,
    @DirectionId);
```

```
    SET @IsAdded=1
```

```
END;
```

ზემოთ შენახულ პროცედურაში @IsAdded არის OUTPUT პარამეტრი. მას მნიშვნელობა მიენიჭება პროცედურის შესრულებისას და დაბრუნდება გამოძახების განცხადებაში. შემდეგი კოდი გამოიძახებს პროცედურას OUTPUT პარამეტრით:

```
DECLARE @IsAdded bit

EXEC [AddStudent]
    @StudentName =N'გიორგი'
    ,@StudentLastName = N'მაისურაძე'
    ,@Email = 'G.maisuradze@gmail.com'
    ,@DirectionId = 2

    ,@IsAdded=@IsAdded OUTPUT

PRINT @IsAdded
```

თუ დავაკვირდებით პროცედურის გამოძახების ტანს, მასში გამოყენებული ახალი დირექტივა: `DECLARE @IsAdded int`  
ვინდაინ ჩვენ გვჭირდება ცვლადი რომელიც არ წარმოადგენს პროცედურის პარამეტრს, და დამატებით უნდა შევქმათ ახალი ცვლადი ვიყენებთ საბრძანებო სიტყვას `DECLARE`.

ცვლადზე მნიშვნელობის მინიჭება შესაძლებელია SET და SELECT ბრძანებებით. სიტყვა `DECLARE`-ის საშუალებით შექმნილი ცვლადის სიცოხლისუნარიანობა განისაზღვრება ერთი ტრანზაქციით, შესაბამისად ერთი exec-ით და იგი პროცედურის გაშვების ყოველ ჯერზე თავიდან იქმნება და ქრება პროცედურის დასრულებისთანავე.

ჩვენს შემთხვევაში შეიქმნა ცვლადი რომელიც ინახავს პროცედურიდან დაბრუნებულ პარამეტრს. ხოლო `PRINT @IsAdded` ბრძანებით ხდება მისი გამოტანა ეკრანზე.

### ნაგულისმები პარამეტრები

SQL Server გაძლევთ საშუალებას მიუთითოთ პარამეტრების ნაგულისხმევი მნიშვნელობები. შესაბამისად გამოტოვოთ სასურველი პარამეტრები, რომლებსაც აქვთ ნაგულისხმევი მნიშვნელობები შენახული პროცედურის გამოძახებისას. ნაგულისხმევი მნიშვნელობა გამოიყენება მაშინ, როდესაც პარამეტრს არ გადაეცემა მნიშვნელობა ან როდესაც DEFAULT საკვანძო სიტყვა მითითებულია, როგორც მნიშვნელობა პროცედურის გამოძახებაში.

მიუთითეთ ნაგულისხმევი მნიშვნელობა, როდესაც აცხადებთ პარამეტრებს, როგორც ეს ნაჩვენებია ქვემოთ:

```
ALTER PROCEDURE [dbo].[AddStudent]
```

```

@StudentName NVARCHAR(30),
@StudentLastName NVARCHAR(30),
@email VARCHAR(30),
@DirectionId INT=1,
@IsAdded BIT output

AS
BEGIN
    INSERT INTO [dbo].[Students]
    (
        [StudentName],
        [StudentLastName],
        [Email],
        [DirectionId]
    )
    VALUES
    (@StudentName,
        @StudentLastName,
        @Email,
        @DirectionId);

    SET @IsAdded=1
END;

```

როგორც ხედავთ პარამეტრ @DirectionId-ს გაჩუმების პრინციპით მითითებული აქვს მნიშვნელობა 1, შესაბამისად თუ მას არ გადავცემთ პროცედურის გამოძახებისას, ავტომატურად მიენიჭება მიმართულების ნომერი -1.

```

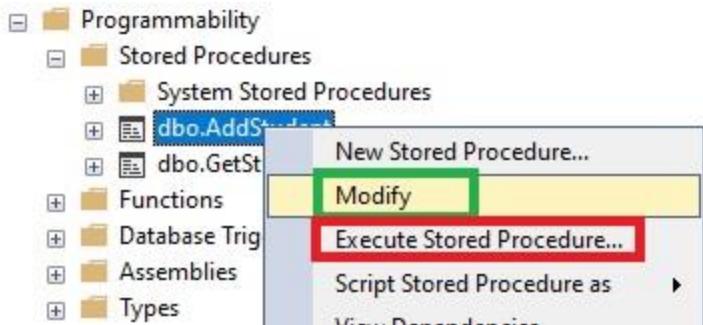
DECLARE @IsAdded bit

EXEC [AddStudent]
    @StudentName =N'გიორგი'
    ,@StudentLastName = N'მაისურაძე'
    ,@Email = 'G.maisuradze@gmail.com'
    ,@IsAdded=@IsAdded OUTPUT

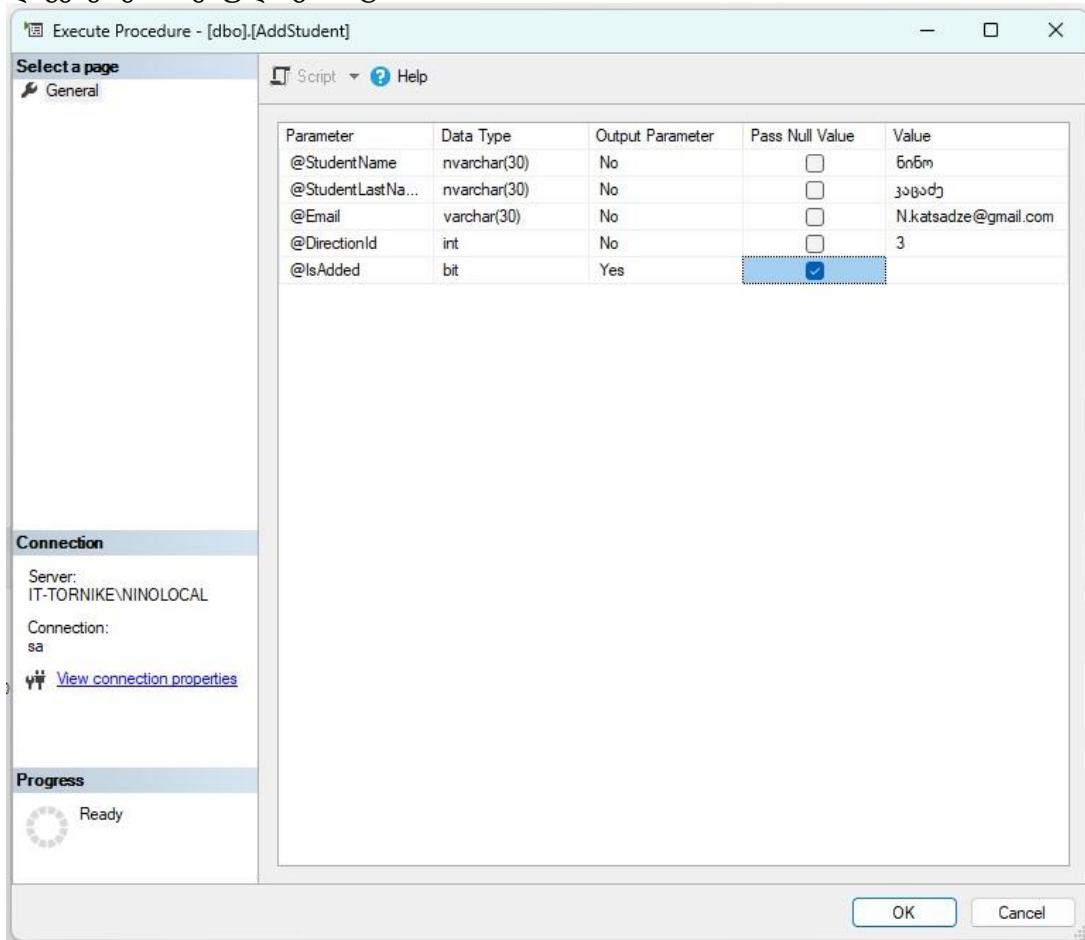
```

```
PRINT @IsAdded
```

პროცედურების გამოძახება ასევე შესაძლებელია დიზაინ რეჟიმში პროცედურაზე მარჯვენა ღილაკის და Execute ბრძანების გამოყენებით, ხოლო უკვე არსებული პროცედურის მოდიფიკაცია ასევე შესძლებელია მასზე მარჯვენა ღილაკის დაჭრით და Modify ბრძანების ამორჩევით.



Execute ის გაშვების შემთხვევაში მიიღებთ ფანჯარას რომელშიც უნდა ჩაწეროთ თითოეული პარამეტრი გარდა გამომავლებისა, ასევე შესრულების შემდგომ მიიღებთ გამოძახების დაგენერირებულ სკრიპტს.



```

USE [Faculty]
GO

DECLARE @return_value int,
        @IsAdded bit

EXEC    @return_value = [dbo].[AddStudent]
        @StudentName = N'ნინო',
        @StudentLastName = N'კაცაძე',
        @Email = N'N.N.katsadze@gmail.com',
        @DirectionId = 3,
        @IsAdded = @IsAdded OUTPUT

SELECT  @IsAdded as N'@IsAdded'

SELECT  'Return Value' = @return_value

GO

```

Results

	@IsAdded
1	1

## შეცდომების დამუშავება შენახულ პროცედურებში

როგორც უკვე აღვნიშნეთ პროცედურა წარმოადგენს ავტომატიზირებულ სკრიპტს, რომელიც უნდა მუშაობდეს ჩვენგან დამოუკიდებლად მაგალითად აპლიკაციაში ღილაკის დაჭერის ან საიტზე სარეგისტრაციო ფორმის შევსების შემდგომ, შესაბამისად აღნიშნული სკრიპტები უნდა დავწეროთ ძალიან ფრთხილად რომ მაქსიმალურად ავირიდოთ თავიდან შეცდომები, და მათი არსებობის შემთხვევაში სწორად გავუმკლავდეთ მათ.

იმისთვის რომ სწორად დავწეროთ კოდი საჭიროა გავთვალით ყველა შესაძლო შემთხვევა, იდეალურ შემთხვევაში ყველა არავალიდურ მონაცემებს დავხვდებით გამზადებული შესაბამისი „შეცდომის შეტყობინებით“ (Error Message) თუმცა ყოველთვის იარსებებს ალბათობა გუთვალისწინებელი შეცდომის, რა დროსაც შეცდომას დაგვიბრუნებს სისტემა სპეციალური ფუნქციებით, როგორიცაა ERROR\_MESSAGE(), ERROR\_NUMBER(), ERROR\_STATE(), ERROR\_SEVERITY().

აღნიშნული პრობლემის საპასუხოდ SQL Server-ში არსებობს TRY..CATCH ბლოკი რომელიც გამოიყენება შეცდომების დასამუშავებლად. მირითადი T-SQL ბრძანებების ჯგუფი იწერება TRY ბლოკში. თუ TRY ბლოკში მოხდა შეცდომა, კონტროლი გადაეცემა CATCH ბლოკს, რომელსაც ექნება SQL განცხადებების სხვა ნაკრები შეცდომის დასამუშავებლად.

მაგალითისთვის განვიხილოთ ისევ სტუდენტის დამატების მოქმედება, რომელიც შეიძლება სინამდვილეში სულაც არ შედგებოდეს ერთი Insert ოპერაციისაგან, მაგალითად სისტემაში დავამატოთ სტუდენტის ცხრილის სრული ლოგირების ცხრილი StudentsLog რათა ყოველთვის ვიცოდეთ რომელი ჩანაწერი ვის მიერ დაემატა, განახლდა თუ წაიშალა და როდის. თუ სტუდენტის ცხრილი გამოიყურება ასე:

```
CREATE TABLE [dbo].[Students](
    [StudentId] [INT] IDENTITY(1000,1) NOT NULL PRIMARY KEY,
    [StudentName] [NVARCHAR](30) NOT NULL,
    [StudentLastName] [NVARCHAR](30) NOT NULL,
    [Email] [VARCHAR](30) NULL,
    [DirectionId] [INT] NOT NULL,
    [Password] [VARBINARY](250) NULL)
```

მისი ლოგის ცხრილი შეიძლება გამოიყურებოდეს შემდეგნაირად:

```
CREATE TABLE [dbo].[StudentsLog](
    [StudentsLogId] [int] IDENTITY(1,1) NOT NULL PRIMARY KEY,
    [LogOperation] [varchar](50) NULL,
    [UserId] [int] NULL,
    [RecordDate] [datetime] NULL,
    [StudentId] [int] NULL,
    [StudentName] [nvarchar](30) NULL,
    [StudentLastName] [nvarchar](30) NULL,
    [Email] [varchar](30) NULL,
    [DirectionId] [int] NULL,
    [Password] [varbinary](250) NULL)
```

```
go
```

ვინაიდან მასში დამატებული იქნება ლოგის ჩანაწერის ნომერი (StudentsLogId), ლოგის ოპერაციის დასახელება (LogOperation), ოპერაციის შემსრულებელი მომხმარებლის ნომერი (UserId) (რომელიც სტანდარტულად ყოველთვის ვიცით სისტემაში ავტორიზაციის შემდგომ მუშაობის პროცესში)და ოპერაციის შესრულების თარიღი (RecordDate).

აღნიშნულ შემთხვევაში პროცედურის ტანი შეიცვლება ვინაიდან მასში უნდა მოხდეს არა ერთი არამედ ორი ცხრილის შევსება:

```
ALTER procedure [dbo].[AddStudent]
```

```
@StudentName nvarchar(30),
@StudentLastName nvarchar(30),
```

```
@Email varchar(30),
@DirectionId int,
@UserId int
as
begin

DECLARE @StudentId int

insert into [dbo].[Students]
([StudentName],
[StudentLastName],
[Email],
[DirectionId],
[Password])
values
(@StudentName,
@StudentLastName,
@email,
@DirectionId,
PWDENCRYPT('123456'))
)

set @StudentId=SCOPE_IDENTITY()

insert into [dbo].[StudentsLog]
(
[LogOperation],
[UserId],
[RecordDate],
[StudentId],
[StudentName],
[StudentLastName],
[Email],
[DirectionId],
[Password])
values
(
'Insert',
@UserId,
getdate(),
@StudentId,
@StudentName,
@StudentLastName,
@email,
@DirectionId,
PWDENCRYPT('123456')
)
```

```
end  
go
```

რაც შეეხება **SCOPE\_IDENTITY()** სისტემურ ფუნქციას, იწერება ჩასმის Insert ოპერაციის შემდგომ და თუ კი მოხდა ავტომატური გადანომვრის მქონე ცვლადის (Identity) მნიშვნელობის დაგენერირება, დაგვიბრუნებს მას. მარტივად რომ ვთქვათ სტუდენტის დამატების შემდეგ მას მიენიჭა სისტემაში თავისუფალი მომდევნო ნომერი (რომელიც ჩვენ არ ვიცით) ხოლო ბრძანება **set**

**@StudentId=SCOPE\_IDENTITY()** წინასწარ გამზადებულ ცვლადში ჩაწერს ამ ნომერს.

ასევე სისტემური ფუნქცია **PWDENCRYPT( '123456' )** დასკრიპტავს გადაცემულ სტრიქონს და ჩაწერს პაროლის ველში. შესაბამისად რეგისტრაციის დროს ყველას ენიჭება ერთი და იგივე პაროლი რომლის შეცვლაც შეეძლებათ შემდგომ.

გავითვალისწინოთ რომ პაროლების შენახვა დაუსკრიპტავი ფორმით დაუშვებელია!

რომ დავუბრუნდეთ მთავარ კოდს, როგორც ვთქვით მიშვნელოვანია წინასწარ გავითვალისწინოთ ყველა შეცდომა და მომხმარებელს დავუბრუნოთ შესაბამისი შეტყობინება, მაგრამ თუ მაინც მოხდა რაიმე გაუთვალისწინებელი საჭიროა გვქონდეს TRY..CATCH ბლოკი რათა ძირითად კოდში რაიმე შეცდომის მოხდენის შემთხვევაში დავიჭიროთ ეს შეცდომები, შევწყვიტოთ მოქმედები TRY ბლოკში და გადავიდეთ CATCH-ზე სადაც გვიწერია კოდი რომელიც გვინდა რომ შესრულდეს შეცდომების შემთხვევევაში.

დავამატოთ ლოგის ცხრილში Error nvarchar(max) ველი, შეცდომების შესანახად:

```
ALTER TABLE [dbo].[StudentsLog]  
ADD Error nvarchar(max)
```

ჩავასწოროთ ჩვენი პროცედურა დავამატოთ რამოდენიმე დასაბრუნებელი პარამეტრი, მათ შორის შეტყობინება შეცდომაზე :

```
ALTER procedure [dbo].[AddStudent]
```

```
@StudentName nvarchar(30),  
@StudentLastName nvarchar(30),  
@Email varchar(30),  
@DirectionId int,  
@StudentId INT OUTPUT,  
@IsAdded BIT OUTPUT,  
@Error NVARCHAR (MAX) OUTPUT
```

```
as
```

```
BEGIN TRY

insert into [dbo].[Students]
([StudentName],
[StudentLastName],
[Email],
[DirectionId],
[Password])
values
(@StudentName,
@StudentLastName,
@email,
@DirectionId,
PWDENCRYPT('123456')
)

SET @StudentId=SCOPE_IDENTITY()
SET @IsAdded=1

insert into [dbo].[StudentsLog]
(
[LogOperation],
[UserId],
[RecordDate],
[StudentId],
[StudentName],
[StudentLastName],
[Email],
[DirectionId],
[Password])
values
(
'Insert',
@UserId,
getdate(),
@StudentId,
@StudentName,
@StudentLastName,
@email,
@DirectionId,
PWDENCRYPT('123456')
)

END TRY

BEGIN CATCH
SET @IsAdded=0
SET @Error=ERROR_MESSAGE()

insert into [dbo].[StudentsLog]
(
[LogOperation],
[UserId],
[RecordDate],
[StudentId],
[StudentName],
```

```

[StudentLastName],
[Email],
[DirectionId],
[Password],
[Error])
values
(
'Insert Fail',
@UserId,
getdate(),
@StudentId,
@StudentName,
@StudentLastName,
@email,
@DirectionId,
PWDENCRYPT('123456')
,@Error)

```

```
END CATCH
```

```
go
```

ძირითად კოდში შეცდომის შემთხვევაში CATCH ბლოკში მაინც მოხდება შეცდომის შესახებ ინფორმაციის ლოგირება ასევე დასაბრუნებელი ცვლადები პროგრამას დაუზრუნებენ პასუხს შეცდომის შესახებ.  
გამოვიძახოთ პროცედურა გადავცეთ არარსებული მიმართულების ნომერი და ვნახოთ შედეგი:

```

DECLARE
    @StudentId int,
    @IsAdded bit,
    @Error nvarchar(max)

EXEC [dbo].[AddStudent]
    @StudentName = N'ნიკა',
    @StudentLastName = N'რუხაძე',
    @Email = N'N.rukhadze@gmail.com',
    @DirectionId = 12,
    @UserId = 1,
    @StudentId = @StudentId OUTPUT,
    @IsAdded = @IsAdded OUTPUT,
    @Error = @Error OUTPUT

SELECT      @StudentId as N'@StudentId',
            @IsAdded as N'@IsAdded',
            @Error as N'@Error'

```

აღნიშნული შეცდომის შემდეგ დავათვალიეროთ ძირითადი და ლოგირების ცხრილები:

```
SELECT * FROM [dbo].[Students];
```

```
SELECT * FROM [dbo].[StudentsLog];
```

როგორც უკვე აღვნიშნეთ მიშვნელოვანია წინასწარ გავითვალისწინოთ ყველა შეცდომა და მომხმარებელს დავუბრუნოთ შესაბამისი შეტყობინება, მაგალითად თუ შეუძლებელია ერთი და იმავე მეილით ორი სტუდენტის რეგისტრაცია სასურველია ჩამატებამდე შემოწმდეს მეილის უნიკალურობა და მომხმარებელმა მიღოს შესაბამისი შეცდომის შესახებ შეტყობინება. ჩავამატოთ If ლოგიკური შემოწმების პირობა პროცედურაში:

```
ALTER PROCEDURE [dbo].[AddStudent]
    @StudentName NVARCHAR(30),
    @StudentLastName NVARCHAR(30),
    @Email VARCHAR(30),
    @DirectionId INT = 1,
    @UserId INT,
    @StudentId INT OUTPUT,
    @IsAdded BIT OUTPUT,
    @Error NVARCHAR(MAX) OUTPUT
AS
BEGIN TRY

    IF EXISTS (SELECT [Email] FROM [dbo].[Students] WHERE [Email] = @Email)
    BEGIN
        SET @IsAdded = 0;
        SET @Error = N'სტუდენტი ასეთი მეილით უკვე დარეგისტრირებულია სისტემაში.';
        RAISERROR(@Error, 16, 1);
    END;

    INSERT INTO [dbo].[Students]
    (
        [StudentName],
        [StudentLastName],
        [Email],
        [DirectionId],
        [Password]
    )
    VALUES
    (@StudentName, @StudentLastName, @Email, @DirectionId, PWDENCRYPT('123456'));

    SET @StudentId = SCOPE_IDENTITY();
    SET @IsAdded = 1;

    INSERT INTO [dbo].[StudentsLog]
    (
        [LogOperation],
        [UserId],
        [RecordDate],
        [StudentId],
        [StudentName],
        [StudentLastName],
        [Email],
        [DirectionId],
```

```

        [Password]
    )
VALUES
('Insert', @UserId, GETDATE(), @StudentId, @StudentName, @StudentLastName, @Email,
@DirectionId,
PWDENCRYPT('123456'));

END TRY
BEGIN CATCH
    SET @IsAdded = 0;
    SET @Error = ERROR_MESSAGE();

    INSERT INTO [dbo].[StudentsLog]
(
    [LogOperation],
    [UserId],
    [RecordDate],
    [StudentId],
    [StudentName],
    [StudentLastName],
    [Email],
    [DirectionId],
    [Password],
    [Error]
)
VALUES
('Insert Fail', @UserId, GETDATE(), @StudentId, @StudentName, @StudentLastName,
@Email, @DirectionId,
PWDENCRYPT('123456'), @Error);

END CATCH;
GO

```

გავნიხილოთ **IF EXISTS** პირობა, თუ პირობის მერე მომავალი მოთხოვნა (select) დაბრუნდება არაცარიელი, ჩვენს შემთვევაში იპოვის ჩანაწერს იმავე მეილით რა მეილის მინიჭებასაც ახლა ვაპირებთ ახალი სტუდენტისთვის, გამოდის რომ მეილი დუბლირდება, უნდა შეწყდეს დაამატების ოპერაცია და გამოვიდეს შეცდომა. ასეც მოვიქცევით, თუ შემოწმების პირობა ჭეშმარიტია **IF**-ის ტანში, რომელი შემოსაზღვრულია **BEGIN** და **END** ოპერატორებით ვწერთ შეცდომის ტექსტს შესაბამის **@Error** ცვლადში და ხელოვნურად წარმოვქმნით შეცდომას **RAISERROR** ფუნქციით. შეცდომის წარმოქმნისთანავე **TRY** ბლოკი „ავარიულად“ წყვეტს მოქმედებას და იწყება **CATCH** ბლოკის შესრულება სადაც ვახდენთ შეცდომის შესახებ ინფორმაციის შენახვას ლოგირების ცხრილში. თუ დავაკვირდებით კოდს, არ დაგვჭირდა განშტოების **IF**-ოპერატორისთვის **ELSE** პირობის დაწერა, **TRY..CATCH** ბლოკის თავისებურებიდან გამომდინარე ავტომატურად შეწყდა შეცდომის წარმოქმნისთანავე მოქმედება და აღარ შესრულდა დანარჩენი შევსების ოპერატორები.

ხელახლა გამვიძახოთ პროცედურა და გადავცეთ ერთი და იგივე მეილი ორჯერ:

DECLARE

```
@StudentId int,  
@IsAdded bit,  
@Error nvarchar(max)
```

EXEC [dbo].[AddStudent]

```
@StudentName = N'ნიკა',  
@StudentLastName = N'რუხაძე',  
@Email = N'N.rukhadze@gmail.com',  
@DirectionId = 2,  
@UserId = 1,  
@StudentId = @StudentId OUTPUT,  
@IsAdded = @IsAdded OUTPUT,  
@Error = @Error OUTPUT
```

SELECT @StudentId as N'@StudentId',  
@IsAdded as N'@IsAdded',  
@Error as N'@Error'

შემდეგ კვლავ დავათვალიეროთ ძირითადი და ლოგირების ცხრილები:

```
SELECT * FROM [dbo].[Students];  
SELECT * FROM [dbo].[StudentsLog];
```

ვინაიდან TRY ბლოკის შესრულება წყდება შეცდომის მოხდენისას, შესაძლებელია რომ ოპერაციათა ნაწილი შერულდეს ხოლო ნაწილი არა. მაგალითად ჩვენი პროცედურის შემთხვევაში ერთი ჩასმა შესრულდეს ხოლო მეორე ვერა, რაც გამოიწვევს პროგრამის ხარვეზულ მუშაობას ვინაიდან თავიდანვე პროცედურის შექმნისას მთავარი მიზანია მთლიანი სკრიპტის ერთად შესრულება და არა ნაწილ-ნაწილ.

მაგალითად განვიხილოთ რაიმე რეგისტრაციის ფორმა სადაც შეიძლება ერთ ფორმაში ივსებოდეს რამოდენიმე ცხრილი, ძირითადი ცხრილი, დეტალურის ცხრილი და ასევე ფაილი იტვირთებოდეს და ამ დროს ამ სამიდან რომელიმე შევსება დასრულდა წარუმატებლად, რა გამოდის, რეგისტრაცია გავიარეთ ნაწილბრივად? ვერც თავიდან ვრეგისტრირებით იმიტომ რომ ერთ ცხრილში უკვე არის ჩვენზე ინფორმაცია და დუბლირდება ხელახლა ცდისას, ხოლო დანარჩენ ორ ცხრილში ვერ მოხდა შევსება და ავტორიზაციასაც ვეღარ გავდივარ? აღნიშნული პრობლემის თავიდან ასაცილებლად საჭიროა რომ პროცედურის ტანი ან შესრულდეს მთლიანად ან საერთოდ არ შესრულდეს და დაგვიბრუნდეს შეცდომა. ერთი შეხედვით ბოლო სკრიპტში გადავჭრით სტანდარტული შეცდომების შემთხვევები მაგრამ როგორ გავიგოთ სად მოხდა შეცდომა? რეალურად არ არის საჭრო ვიცოდეთ რომელი ცხრილი შეივსო/შეიცვალა და რომელი არა, უბრალოდ

უნდა დავამატოთ ახალი ლოგიკა. რომლის მიხედვითაც თუკი პროცედურაში მოხდა შეცდომა მთლიანი სკრიპტი „დაბრუნდება უკან“ და წაიშლება რაც კი ცვლილებები შევიდა ცხრილებში კონკრეტული პროცედურის შესრულებისას. გამოდის თუკი რომელიმე ცხრილში რამე ჩაიწერა ან შეიცვალა უნდა დაბრუნდეს იმ მდგომარეობაზე რაც პროცედურის დაწყებამდე იყო. ამისათვის უნდა დავიჭიროთ საიდან დაიწყო პროცედურის ტანი და სად დამთავრდა, გავაერთიანოთ რამოდენიმე მოქმედება ერთ მოქმედებაში (ტრანზაქციაში) და შემდეგ დავაბრუნოთ აღნიშნული ტრანზაქცია „უკან“ (ROLLBACK).

### SQL Server Transaction

ტრანზაქცია SQL Server-ში არის მონაცემთა ბაზაში ერთი ან რამდენიმე ამოცანის შესასრულებლად გათვალიწინებული ბრძანებების ან/და მოთხოვნების თანმიმდევრული ჯგუფი . თითოეული ტრანზაქციას შეიძლება შედგებოდეს წაკითხვის, ჩაწერის, განახლების ან წაშლის ერთი ოპერაციის ან ყველა ამ ოპერაციების კომბინაციისაგან. ყოველი ტრანზაქციისთვის აუცილებელია:

- ყველა მოქმედება წარმატებულია და ხდება ტრანზაქციის შესრულება/დასრულება (Commit).
- შეცდომის შემთხვევაშო ყველა ცვლილება უქმდება და ტრანზაქცია უკან დაბრუნდება (Rollback).

სტანდარტულად Insert, Update ან Delete -ის დროს სერვერი იწყებს ავტოტრანზაქციებს, სადაც თითო მოქმედება წარმოადგენს თითო ტრანზაქციას. ტრანზაქციის მექანიკურად შესაქმნელად უნდა გამოვიყენოთ ბრძანებები BEGIN TRANSACTION ან BEGIN TRAN და დავასრულოთ ბრძანებებით COMMIT ან ROLLBACK. თუკი ტრანზაქციის დასაწყისიდან COMMIT მდე შეცდომა არ დაფიქსირდა ითვლება რომ ტრანზაქცია წარმატებით დასრულდა.

განვიხილოთ ჩვენი პროცედურის სკრიპტი ტრანზაქციით:

```

ALTER PROCEDURE [dbo].[AddStudent]
    @StudentName NVARCHAR(30),
    @StudentLastName NVARCHAR(30),
    @Email VARCHAR(30),
    @DirectionId INT = 1,
    @UserId INT,
    @StudentId INT OUTPUT,
    @IsAdded BIT OUTPUT,
    @Error NVARCHAR(MAX) OUTPUT
AS
BEGIN TRY
BEGIN TRANSACTION

    IF EXISTS (SELECT [Email] FROM [dbo].[Students] WHERE [Email] = @Email)
    BEGIN
        SET @IsAdded = 0;
    END
    ELSE
    BEGIN
        INSERT INTO [dbo].[Students] ([Name], [LastName], [Email], [DirectionId], [UserId])
        VALUES (@StudentName, @StudentLastName, @Email, @DirectionId, @UserId);
        SET @IsAdded = 1;
        SET @StudentId = SCOPE_IDENTITY();
    END
END TRY
BEGIN CATCH
    IF @IsAdded = 0
    BEGIN
        ROLLBACK TRANSACTION;
        SET @Error = 'Email already exists';
        SET @IsAdded = 0;
    END
    ELSE
    BEGIN
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        SET @Error = 'An error occurred during transaction';
        SET @IsAdded = 0;
    END
END CATCH

```

```

SET @Error = N'სტუდენტი ასეთი მეილით უკვე დარეგისტრირებულია სისტემაში.' ;
RAISERROR(@Error, 16, 1);

END;

INSERT INTO [dbo].[Students]
(
    [StudentName],
    [StudentLastName],
    [Email],
    [DirectionId],
    [Password]
)
VALUES
(@StudentName, @StudentLastName, @Email, @DirectionId, PWDENCRYPT('123456'));

SET @StudentId = SCOPE_IDENTITY();
SET @IsAdded = 1;

INSERT INTO [dbo].[StudentsLog]
(
    [LogOperation],
    [UserId],
    [RecordDate],
    [StudentId],
    [StudentName],
    [StudentLastName],
    [Email],
    [DirectionId],
    [Password]
)
VALUES
('Insert', @UserId, GETDATE(), @StudentId, @StudentName, @StudentLastName, @Email,
@DirectionId,
PWDENCRYPT('123456'));
COMMIT

END TRY
BEGIN CATCH
    ROLLBACK TRANSACTION
    SET @IsAdded = 0;
    SET @Error = ERROR_MESSAGE();

    INSERT INTO [dbo].[StudentsLog]
(
    [LogOperation],
    [UserId],
    [RecordDate],
    [StudentId],
    [StudentName],
    [StudentLastName],
    [Email],
    [DirectionId],
    [Password],
    [Error]
)
VALUES

```

```
( 'Insert Fail', @UserId, GETDATE(), @StudentId, @StudentName, @StudentLastName,  
@Email, @DirectionId,  
PWDENCRYPT('123456'), @Error);
```

```
END CATCH;
```

```
GO
```

TRY ბლოკის დასაწყისშივე იხსნება ტრანზაქცია BEGIN TRANSACTION ბრძანებით და იხურება END TRY ბლოკის დასასრულში, ბრძანებით COMMIT. ეს ხდება რათა TRY -ს ტანი სერვერმა აღიქვას როგორც ერთი მთლიანი მოქმედება და იცოდეს რომ BEGIN TRANSACTION -დან COMMIT სიტყვამდე ყველა მოქმედება უნდა დასრულდეს წარმატებით რათა ჩაითვალოს რომ ტრანზაქცია შესრულდა. ახლა უკვე ჩვენი შევსების ოპერაციები სერვერისთვის აღარაა ცალცლაკე დამოუკიდებელი მოქმედებები. როგორც ვიცით თუ TRY ბლოკი დასრულდა უშეცდომოთ (გამოდის რომ BEGIN TRANSACTION -დან COMMIT სიტყვამდე არ მოხდა შეცდომა) ოპერაცია ითვლება წარმატებულად და აღარ სრულდება CATCH ბლოკი. მაგრამ ნებისმიერ შეცდომის შემთხვევაში წყდება TRY ბლოკი და შესაბამისად ტრანზაქცია, მართვა გადაეცემა CATCH ბლოკს, რომელშიც ვხედავ ბრძანებას ROLLBACK TRANSACTION.

ROLLBACK TRANSACTION ის შესრულებისას ტრანზაქციის შემადგენელი ყველა მოქმდება დაბრუნდება უკან. ზემოთ აღნიშნული პრობლემაც ნაწილობრივი რეგისტრაციის თაობაზე მოგვარდება ავტომატურად რადგან თუ ერთი ცხრილი შეივსო ხოლო დანარჩენებში მოხდა შეცდომა იმ ერთიდანაც ამოიშლება ჩანაწერი და დავიწყებთ რეგისტრაციის ხელახალ მცდელობას.

```

--select * from student
--select * from department
--select * from subjects

select * from [dbo].[StudentSubjects]
select * from [dbo].[StudentDepartment]

--დავამატოთ შეზღუდვა StudentSubjects Reg_Date გავხადოთ დეფოლტად getdate()
alter table [dbo].[StudentSubjects]
add
constraint df_regdate default getdate() for [reg_date]

====პროცედურები
--შევქმნათ სტუდენტის ცხრილში ჩანაწერის დამატები პროცედურა--

if object_id('InsertStudent', 'P')
is not null
Drop procedure InsertStudent
Go
create procedure InsertStudent
@StudentName nvarchar(30),
@StudentLastName nvarchar(30),
@DateofBirth date,
@personalId char(11),
@Gender nvarchar(20),
@city nvarchar(50),
@email varchar(30)
As
Begin
insert into student values
(
@StudentName, @StudentLastName, @DateofBirth, @personalId, @Gender, @city, @email
)
End

---შევხედოთ პროცედურის შექმნის კოდს

exec sp_helptext 'InsertStudent'

--ჩავსვათ სტუდენტის ცხრილში ჩანაწერი პროცედურით
exec InsertStudent N'ელენე', N'ბექაური', '2/2/2020', '11223344557', N'მდედრობითი',
N'ბათუმი', 'xxxx@gmail.com'

-- შევქმნათ StudentSubjects ცხრილში მონაცემის შეტანის პროცედურა
create procedure RegStudentSubjects
@StudentID int,
@SubjectID int
As
Begin
insert into StudentSubjects(StudentId, SubjectId)
values
(@StudentId, @SubjectId)
End
Go

```

```
--გავაკეთოთ ცხრილში ჩანაწერი პროცედურით  
exec RegStudentSubjects 10, 400
```

```
-- შექმნათ პროცედურა, რომელსაც გადავცემთ StudentId-ს და დაგვიბრუნებს სრულ  
--ინფორმაციას Student ცხრილიდან
```

```
Create procedure StudentInfo  
@StudentId int  
As  
Begin  
Select * from Student  
Where StudentID=@studentId  
End
```

```
exec StudentInfo 1
```

```
--დავწეროთ მოთხოვნა, რომელიც დააბრუნებს საგნებს და მასზე დარეგისტრირებულ სტუდენტების  
რაოდენობას.
```

```
--მხოლოდ ის საგნები რომლებზედაც სტუდენტები არიან დარეგისტრირებულები.  
select stsb.SubjectId, SubjectName, count(StudentId) as Sttotal  
from StudentSubjects stsb left join subjects sb  
on stsb. SubjectId=sb.Subjectid  
Group by stsb.SubjectId, SubjectName
```

```
--დავწეროთ მოთხოვნა, რომელიც დააბრუნებს სრულად საგნებს და მასზე დარეგისტრირებულ  
სტუდენტების რაოდენობას.
```

```
--ის საგნებიც, რომლებზედაც სტუდენტები არ არიან დარეგისტრირებულები.  
select sb.SubjectId, SubjectName, count(StudentId) as Sttotal  
from StudentSubjects stsb right join subjects sb  
on stsb. SubjectId=sb.Subjectid  
where sb.SubjectId=400  
Group by sb.SubjectId, SubjectName
```

```
--შევქმნათ პროცედურა, რომელსაც გადავცემთ საგნის კოდს და  
--გამოიტანს მასზე დარეგისტრირებული სტუდენტების რაოდენობას.
```

```
create procedure SubjectStudentTotal  
@SubjectId int  
As  
Begin  
select sb.SubjectId, SubjectName, count(StudentId) as Sttotal  
from StudentSubjects stsb right join subjects sb  
on stsb. SubjectId=sb.Subjectid  
where sb.SubjectId=@subjectID  
Group by sb.SubjectId, SubjectName  
End
```

```
exec SubjectStudentTotal 800
```

```
--შევქმნათ პროცედურა, რომელიც მხოლოდ იმ საგნებს დააბრუნებს,  
--რომლებზედაც დარეგისტრირებული სტუდენტების რაოდენობა >2-ზე (საგანი დახურულია)
```

```
Create procedure ClosedSubjects
```

```

as
Begin
select stsb.SubjectId, SubjectName, count(StudentId) as StTotal
From StudentSubjects stsb left join Subjects sb
    on Stsb.SubjectId=Sb.Subjectid
Group by stsb.SubjectId, SubjectName
having count(StudentId)>2
End

exec ClosedSubjects

```

--განვსაზღვროთ საგნისათვის აქტივაციის სტატუსი:

```

select sb.SubjectId, SubjectName, count(StudentId),
case
when count(StudentId)>2 then N'დახურულია'
Else N'აქტიურია'
End N'აქტივია'
From StudentSubjects stsb right join Subjects sb
    on Stsb.SubjectId=Sb.Subjectid
Group by sb.SubjectId, SubjectName

```

--შევქმნათ პროცედურა, რომელიც სტუდენტის კოდის გადაცემის შემთხვევაში  
--დაითვლის მის მიერ არჩეული საგნების კრეიტების ჯამს.

```

create procedure SumCredits
@StudentId int
As
Begin
select St.StudentId, sum(Subjectcredit) as total
from Student st left join StudentSubjects stsb on st.studentid=stsb.studentid
    left join subjects sb on stsb.subjectid=sb.subjectid
where St.StudentId=@StudentId
group by St.StudentId
End

```

```
exec SumCredits 5
```

--შევქმნათ პროცედურა, რომელიც სტუდენტის კოდის, ბველი და ახალი საგნის კოდების  
--გადაცემის შემთხვევაში, მოახდენს სტუდენტისატვის საგნის განახლებას.--

```

Create procedure updateStudentSubject
@StudentId int,
@oldSubjectid int,
@NewSubjectID int
as
Begin
update studentSubjects
Set SubjectId=@newsSubjectId
where studentId=@studentId
        and Subjectid=@oldSubjectId
End

```

```
exec updateStudentSubject 7,200,100  
exec updateStudentSubject 7,500
```

```
select * from StudentSubjects
```

```
update studentSubjects  
Set SubjectId=100 where studentId=7
```

## ტრიგერები

### ტრიგერები

ტრიგერები არის შენახული პროცედურების განსაკუთრებული ტიპი, რომელიც სრულდება ცხრილში განსაზღვრული ოპერაციის შესრულების შემდეგ. მისი გამოყენაება ბაზაში უფრო მონაცემების დაცვის მიზნით ხდება.

არსებობს ორი ტიპის ტრიგერები: DML (INSERT, UPDATE, DELETE) და DDL ტრიგერები - CREATE TABLE, DROP TABLE, და ALTER TABLE.

DML ტრიგერი რეაგირებს სამ ოპერაციაზე: ჩასმის, განახლების და წაშლის ოპერაციებზე. თუ ცხრილში ხდება ეს სამი ოპერაცია, მაშინ ხდება ამ ცხრილისთვის შექმნილი ტრიგერის ავტომატური გაშვება სისტემის მიერ. ტრიგერში პარამეტრები არ გვაქვს, განსხვავებით შენახული პროცედურისაგან.

ტრიგერებს ახასიათებთ შემდეგი თვისებები:

- დამოკიდებულ ცხრილებში მასიური ოპრაციების შესრულება;
- შეცდომის გამოცხადება;
- ცვლილებამდე და ცვლილების შემდეგ მონაცემების შედარება;
- განსხვავებით check შეზღუდვისაგან, ტრიგერებს შეუძლიათ მიმართვა სხვა ცხრილის ველებთან.

ტრიგერის შექმნისათვის გამოიყენება ოპერატორი CREATE TRIGER. ოპერატორში მიეთიტება ცხრილი, რომლისთვისაც ხდება ტრიგერის გამოცხადება. ტრიგერის შექმნის დროს ინფორმაცია მის შესახებ იწერება სისტემურ ცხრილებში: sysobjects, syscomments.

გასათვალისწინებელია ის ფაქტი, თუ ტრიგერის შექმნის დროს იგივე სახელიანი ტრიგერი არსებობს, მაშინ ახლად შექმნილი ტრიგერი ზედ გადაეწერება. ტრიგერების შექმნა სისტემური ცხრილებისათვის არ ხდება. ეს ოპერაცია შეზღუდულია სერვერის მიერ. ტრიგერით შეუძლებელია ისეთი ოპერაციის შესრულება, რომლების ცვლიან ბაზის სტრუქტურას, როგორიცაა:

- ALTER DATABASE;
- CREATE DATABASE;
- DROP DATABASE;
- RESTORE DATABASE

ტრიგერი შესაძლებელია შეიცავდეს ოპერატორს ROLLBACK TRANSACTION თუნდაც არ იყოს BEGIN TRANSACTION . თუ ეს ოპერატორი შესრულდა, მაშინ ხდება უარყოფა ყველა იმ ცვლილებების, რომლებმაც გამოიწვიეს ტრიგერის შეცვლა.

DML ტრიგერის შექმნის კოდის სინტაქსი:

```
CREATE TRIGGER [schema_name.]trigger_name  
ON {table_name | view_name}  
[WITH dml_trigger_option [...]]
```

## ტრიგერები

```
{FOR | AFTER | INSTEAD OF} { [INSERT] [,] [UPDATE] [,] [DELETE]}  
[WITH APPEND]  
{AS sql_statement | EXTERNAL NAME method_name}
```

ტრიგერის შექმნის კოდში არსებული AFTER და INSTEAD ოფცია არის ერთ-ერთი მნიშვნელოვანი ტრიგერის შექმნისას. AFTER ტრიგერი გაეშვება, როდესაც მოთხოვნით გათვალისწინებული მოქმედება შესრულდება. INSTEAD ტრიგერი ამუშავდება მოთხოვნაში განსახორციელებელი მოქმედების მაგივრად. AFTER ტრიგერები იქმნება მხოლოდ ცხრილებისატვის, INSTEAD ტრიგერები კი იქმნება როგორც ცხრილებზე ასევე ჩარმოდგენებზე.

```
CREATE TRIGGER u_tbstudents ON dbo.students
```

```
FOR UPDATE
```

```
AS
```

```
ROLLBACK TRANSACTION
```

განვმარტოთ ეს კოდი:

- ტრიგერის სახელის განსაზღვრისას იყენებენ სიმბოლოებს, რომლებიც მიანიშნებენ, თუ რა ტიპის ტრიგერია ეს. მაგ: u -განახლების, I-ჩასმის, d-წაშლის.
- ტრიგერის სახელის განსაზღვრის შემდეგ მოდის ცხრილის სახელი, რომლისთვისაც იქმნება ეს ტრიგერი. (ON საგასაღებო სიტყვის შემდეგ)
- ამის შემდეგ მოდის საგასაღებო სიტყვა FOR , რომელსაც მოყვება ის მოქმედება, რომლისთვისაც მუშავდება ტრიგერი.(ამ შემთხვევაში განახლება)
- ბოლოს მოდის საგასაღებო სიტყვა AS , რომელსაც მოყვება ტრიგერის ტანი, ანუ ის ბრძანებები, რომლებიც უნდა შესრულდეს. ზემოტ აღწერილ კოდში სრულდება მხოლოდ ერთი ბრძანება: ROLLBACK TRANSACTION.

ტრიგერის შედეგის ტესტირებისათვის გამოიყენება ვირტუალური ცხრილები:~

- DELETED
- INSERTED

პირველი გამოიყენება იმ შემთხვევაში, თუ ტრიგერის შექმნის ინსტრუქციაში არის ბრძანება : DELETE, UPDATE;

ხოლო მეორე იმ შემთხვევასი თუ ხდება ტრიგერის კოდში INSERT, UPDATE ბრძანებების გამოყენება.

AFTER (FOR) ტრიგერის გამოყენების ერთ-ერთი მიზანი არის მოქმედების აუდიტი ცხრილების შექმნა.

განვიხილოთ მაგალით:

**შექმნათ ცხრილი** Employee\_Test. ამ ცხრილისათვის გავაკეთოთ ტრიგერი, რომელიც გაეშვება ცხრილში ყოველი ახალი ჩანაწერის გაკეთების ან ჩანაწერის განახლებისას. ამავე დროს გააკეთებს ჩანაწერებს აუდიტ ცხრილში

## ტრიგერები

Employee\_Test\_Audit, თუ მთავარ ცხრილში როდის განხორციელდა კონკრეტული ქმედებები.

```
if OBJECT_ID ('Employee_Test', 'u')
is not null
drop table Employee_Test
go

CREATE TABLE Employee_Test
(
Emp_ID INT Identity,
Emp_name Varchar(30),
Emp_Sal Decimal (10, 2)
)

INSERT INTO Employee_Test
VALUES
('erekle', 1000),
('nino', 1500),
('giorgi', 1800),
('levani', 2100),
('tamari', 3000),
('ana', 2000),
('natia', 1500)
=====
=====
```

შევქმნათ აუდიტის ცხრილი::

```
if OBJECT_ID ('Employee_Test_Audit', 'u')
is not null
drop table Employee_Test_Audit
go

CREATE TABLE Employee_Test_Audit
(
Emp_ID int,
Emp_name varchar(30),
Emp_Sal decimal (10, 2),
Audit_Action varchar(50),
Audit_Time date
)
```

შევქმნათ ტრიგერი

```
if OBJECT_ID ('trgAfterInsert', 'tr')
is not null
```

## ტრიგერები

```
drop trigger trgAfterInsert
go
CREATE TRIGGER trgAfterInsert ON [dbo].[Employee_Test]
FOR INSERT
as
    declare @empid int;
    declare @empname varchar(50);
    declare @empsal decimal(10,2);
    declare @audit_action varchar(50);

    select @empid=Emp_ID from inserted ;
    select @empname=Emp_Name from inserted ;
    select @empsal=Emp_Sal from inserted ;
    set @audit_action='Inserted Record -- After Insert
Trigger.';

    insert into Employee_Test_Audit
        (Emp_ID,Emp_Name,Emp_Sal,Audit_Action,Audit_Time)
    values(@empid,@empname,@empsal,@audit_action,getdate());

    PRINT 'AFTER INSERT trigger fired.'
GO
```

ტრიგერის ტანში გამოყენებულია ცხრილის სახელი `inserted`, რომელიც არის ლოგიკური ცხრილი და შეიცავს იმ ჩანაწერებს, რომლებიც ჩაემატება. საბოლოოდ ამ ჩანაწერებით შეიცვება აუდიტი ცხრილი.

როდესაც კონკრეტული ქმედებისათვის იქმნება ტრიგერი, საჭიროა მითითებული იქნეს, თუ სად შეინახოს ველების მნიშვნელობები შეცვლის შემდეგ. ამისათვის გამოიყენება ორი ვირტუალური ცხრილი, რომელტაც გააჩნიათ სპეციფიკური სახელები, იმის მიხედვით, თუ რომელი ჩანაწერების ცვლილება ხდება მოთხოვნის შედეგად. შესაბამისად ამ ცხრილების სახელებია: **deleted, inserted**

გავაკეთოთ ჩანაწერი ცხრილში და შევამოწმოთ აუდიტის ცხრილი

```
insert into Employee_Test values('mari',1500);
```

The screenshot shows the 'Messages' window from SQL Server Management Studio. It displays the following output:

```
(1 row(s) affected)
AFTER INSERT trigger fired.

(1 row(s) affected)
```

## ტრიგერები

```
select * from Employee_Test_Audit
```

	Emp_ID	Emp_name	Emp_Sal	Audit_Action	Audit_Time
1	8	mari	1500.00	Inserted Record -- After Insert Trigger.	2014-05-18

=====  
გავაკეტოთ ტრიგერი განახლების ოპერაციაზე.

```
CREATE TRIGGER trgAfterUpdate ON [dbo].[Employee_Test]
FOR UPDATE
AS
    declare @empid int;
    declare @empname varchar(100);
    declare @empsal decimal(10,2);
    declare @audit_action varchar(100);

    select @empid=Emp_ID from inserted;
    select @empname=Emp_Name from inserted ;
    select @empsal=Emp_Sal from inserted;

    if update(Emp_Name)
        set @audit_action='Updated Record -- After Update
Trigger.';
    if update(Emp_Sal)
        set @audit_action='Updated Record -- After Update
Trigger.';

    insert into
Employee_Test_Audit(Emp_ID,Emp_Name,Emp_Sal,Audit_Action,Audit_T
ime)
    values(@empid,@empname,@empsal,@audit_action,getdate()) ;

    PRINT 'AFTER UPDATE Trigger fired.'
GO

update Employee_Test set Emp_Sal=1550 where Emp_ID=5

select * from Employee_Test_Audit
```

## ტრიგერები

	Emp_ID	Emp_name	Emp_Sal	Audit_Action	Audit_Time
1	8	mari	1500.00	Inserted Record -- After Insert Trigger.	2014-05-18
2	5	tamari	1550.00	Updated Record -- After Update Trigger.	2014-05-18

შევქმნათ ტრიგერი, რომელიც გაეშვება მონაცემის წაშლისას.

```
CREATE TRIGGER trgAfterDelete ON [dbo].[Employee_Test]
AFTER DELETE
AS
    declare @empid int;
    declare @empname varchar(100);
    declare @empsal decimal(10,2);
    declare @audit_action varchar(100);

    select @empid=d.Emp_ID from deleted d;
    select @empname=d.Emp_Name from deleted d;
    select @empsal=d.Emp_Sal from deleted d;
    set @audit_action='Deleted -- After Delete Trigger.';

    insert into Employee_Test_Audit
(Emp_ID,Emp_Name,Emp_Sal,Audit_Action,Audit_Time)
values(@empid,@empname,@empsal,@audit_action,getdate());

    PRINT 'AFTER DELETE TRIGGER fired.'
GO
=====
delete from Employee_Test
where Emp_ID=5
```

	Emp_ID	Emp_name	Emp_Sal	Audit_Action	Audit_Time
1	8	mari	1500.00	Inserted Record -- After Insert Trigger.	2014-05-18
2	5	tamari	1550.00	Updated Record -- After Update Trigger.	2014-05-18
3	5	tamari	1550.00	Deleted -- After Delete Trigger.	2014-05-18

## Instead ტრიგერები

## ტრიგერები

ამ ტიპის ტრიგერები გამოიყენება, მაშინ როდესაც იზღუდება სხვა  
მომხმარებლისათვის გარკვეული ოპერაციების შესრულება კონკრეტულ ცხრილებში  
ან წარმოდგენებში.(მაგ. წაშლა)

INSTEAD ტრიგერები კლასიფიცირდება შემდეგ ტიპებად:

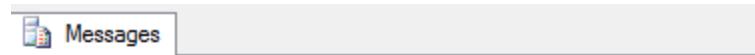
- INSTEAD OF INSERT Trigger.
- INSTEAD OF UPDATE Trigger.
- INSTEAD OF DELETE Trigger.

```
if OBJECT_ID ('trgInsteadOfDelete','tr')  
is not null  
drop trigger trgInsteadOfDelete  
go  
  
CREATE TRIGGER trgInsteadOfDelete ON [dbo].[Employee_Test]  
INSTEAD OF DELETE  
AS  
    declare @emp_id int;  
    declare @emp_name varchar(100);  
    declare @emp_sal int;  
  
    select @emp_id=d.Emp_ID from deleted d;  
    select @emp_name=d.Emp_Name from deleted d;  
    select @emp_sal=d.Emp_Sal from deleted d;  
  
    BEGIN  
        if(@emp_sal>1200)  
        begin  
            RAISERROR('Tu xelfasi <1200, ver cashli',16,1);  
            ROLLBACK;  
        end  
        else  
        begin  
            delete from Employee_Test where Emp_ID=@emp_id;  
            insert into  
Employee_Test_Audit(Emp_ID,Emp_Name,Emp_Sal,Audit_Action,Audit_Time)  
                values(@emp_id,@emp_name,@emp_sal,'Deleted --  
Instead Of Delete Trigger.',getdate());  
            PRINT 'Canawers ver cashli, Sezgudulia trigeris  
mier'  
        end  
    END
```

## ტრიგერები

GO

```
delete from Employee_Test  
where Emp_ID=1
```



```
(1 row(s) affected)  
AFTER DELETE TRIGGER fired.  
  
(0 row(s) affected)  
  
(1 row(s) affected)  
Canawers ver cashli, Sezgudulia trigeris mier
```

განვიხილოთ კიდევ რამდენიმე მაგალითი.

```
create table employee  
(emp_id int identity primary key,  
emp_name varchar(20),  
emp_lname varchar(20),  
date_of_birth date,  
phone char(20),  
)  
create procedure insert_employee  
@emp_name varchar(20),  
@emp_lname varchar(20),  
@date_of_birth date,  
@phone char(20)  
as  
insert into employee  
values  
(  
@emp_name,  
@emp_lname,  
@date_of_birth,  
@phone)  
exec insert_employee 'nino','goguadze','10/3/1992','123123'  
exec insert_employee 'daviti','alavize','11/3/1992','343434'  
exec insert_employee 'gigo','devize','9/5/1994','454545'  
exec insert_employee 'nino','xmalaze','6/4/1995','656565'  
exec insert_employee 'tamri','goguadze','7/5/1996','323132'  
exec insert_employee 'nino','kekelize','1/8/1992','1231122'
```

## ტრიგერები

```
select * from employee
```

### შევასრულოთ განახლების ოპერაცია

```
update employee  
set  
emp_name='maka' where emp_id=6  
  
select * from employee
```

შევქმნათ ტრიგერი, რომელიც შეზღუდავს განახლების ოპერაციას.

```
IF OBJECT_ID ('u_employee', 'TR') IS NOT NULL  
    DROP TRIGGER u_employee;  
GO  
CREATE TRIGGER u_employee  
ON employee  
AFTER UPDATE  
AS  
--RAISERROR (N'update ოპერაციების ის შესრულება არ შეგიძლიათ',  
15, 10);  
    ROLLBACK TRAN  
    GO
```

```
update employee  
set  
emp_name='giorgi' where emp_id=1
```

```
drop trigger u_employee
```

```
IF OBJECT_ID ('u_employee', 'TR') IS NOT NULL  
    DROP TRIGGER u_employee;  
GO  
CREATE TRIGGER u_employee  
ON employee  
AFTER UPDATE  
AS  
RAISERROR (N'update ოპერაციების ის შესრულება არ შეგიძლიათ',  
15, 10);  
    ROLLBACK TRAN  
    GO
```

```
update employee
```

## ტრიგერები

```
set  
emp_name='giorgi' where emp_id=1
```

ტრიგერის შეცვლა ხდება ALTER TRIGGER ბრძანებით.

შეცვალოთ ტრიგერი და დავამატოთ წაშლის ბრძანების შეზღუდვა

```
alter TRIGGER u_employee  
ON employee  
AFTER UPDATE, DELETE  
AS  
RAISERROR (N'update, delete ოპერაციების ის შესრულება არ  
შეგიძლიათ', 15, 10);  
ROLLBACK TRAN  
GO  
  
delete from employee  
where emp_id=6  
  
alter table employee  
disable trigger u_d_employee  
  
update employee  
set  
phone='111111' where emp_id=1
```

ცხრილში ტრიგერით შზღუდული ბრძანებების შესრულებისათვის, საჭიროა ტრიგერის გათიშვა. შეიძლება ცხრილში მოხდეს კონკრეტული ტრიგერის გათიშვა, ან გაითიშოს ყველა ტრიგერი. გათივის პროცედურა სრულდება ALTER TABLE ბრძანებით.

```
alter table employee  
enable trigger u_d_employee  
--  
  
alter table employee  
enable trigger ALL
```

---

ტრიგერების შექმნის მაგალითები:

## ØРІОГЈЕРЈО

```
CREATE TABLE project
(project_no CHAR(4) NOT NULL,
project_name CHAR(15) NOT NULL,
budget int ,
CONSTRAINT prim_proj PRIMARY KEY (project_no));
go

insert into project
values
('100', 'project1', 20000),
('101', 'project2', 120000),
('102', 'project3', 90000),
('103', 'project4', 100000),
('104', 'project5', 55000),


CREATE TABLE audit_budget
(project_no CHAR(4),
us_name CHAR(16),
date DATE,
budget_old int ,
budget_new int
)
GO

--drop trigger modify_budget

CREATE TRIGGER modify_budget
ON project
AFTER UPDATE
AS
IF UPDATE(budget)
BEGIN
DECLARE @budget_old int
DECLARE @budget_new int
DECLARE @project_number CHAR(4)

SELECT @budget_old = (SELECT budget FROM deleted)
SELECT @budget_new = (SELECT budget FROM inserted)
SELECT @project_number = (SELECT project_no FROM deleted)

INSERT INTO audit_budget
VALUES
(@project_number, USER_NAME(), GETDATE(), @budget_old, @budget_new)
```

ওরোগের্জো

END

```
select * from project

UPDATE project
SET budget = 700000
WHERE project_no = '104';

select * from audit_budget
```

## ცვლადები

ნებისმიერი სხვა პროგრამული ენების მსგავსად, T-SQL იძლევა საშუალებას ცვლადების სიმრავლის განსაზღვრის, რომელთა გამოყენებაც შემდგომში შესაძლებელია მოხდეს პროცედურაში ან სხვადასხვა ტიპის მოთხოვნის კოდში. ცვლადი ინახავს ინფორმაციის მცირე ნაწილს, მსგავსად რიცხვის ან სიმბოლოსი. ცვლადის არსებობა იძლევა საშუალებას მონაცემის დროებით შენახვის კოდის გაშვების მომენტში.

ცვლადების გამოყენების ძირითადი მიზანი მდგომარეობს შემდეგში:

- მოხდეს პარამეტრის გადაცემა პროცედურისათვის ან ფუნქციისათვის.
- Loop-ის პროცესის კონტროლი;
- IF გამოსახულებაში პირობის სტატუსის კონტროლი (True ან False)
- Where წინადადებაში პირობის პროგრამულად კონტროლი.

თუ საჭიროა მოთხოვნაში ცვლადის არსებობა, უნდა მოხდეს მისის გამოცხადება DECLARE წინადადებით., შემდეგ ეტაპზე ხდება მისთვის სახელის მიინიჭება. ლოკალური ცვლადის სახელები @ სიმბოლოთი. ბოლო ეტაპზე განისაზღვრება მონაცემის ტიპი.

ცვლადი არის ობიექტი, რომელიც იღებს სპეციფიურ მონაცემტა ტიპს, როგორიცაა: integer, date, character string.

ცვლადის გამოცხადების სინტაქსია ასეთია:

```
DECLARE @variable_name datatype [ = initial_value ],
        @variable_name datatype [ = initial_value ],
        ...;
```

variable\_name-ცვლადის სახელი

datatype-მონაცემის ტიპი, რომელიც უნდა მიენიჭოს ცვლადს

initial\_value-ეს პარამეტრი პირობითია. ეს არის საწყისი მნიშვნელობა რომელიც უნდა მიენიჭოს ცვლადს გამოცხადებისას(თუ ასეთი მნიშვნელობა არსებობს).

ცვლადის გამოყენება, გარდა ზემოთ აღნიშნული შემთხვევებისა, ხდება კიდევ შემდეგი მიზნით:

- როგორც მთვლელი, რომელიც ითვლის ციკლის შესრულების რაოდენობებს;
- შეინახოს მნიშვნელობა, რომელიც მოწმდება While წინადადებით.
- შეინახოს მნიშვნელობა, რომელიც ბრუნდება შენახული პროცედურით ან ფუნქციით.

განვიხილოთ ცვლადის შექმნის მარტივი მაგალით.

```
DECLARE @TestVariable AS NVARCHAR(100)=N'მე მომწონს SQL SERVER'
PRINT @TestVariable
```

შემდეგ მაგალითში გამოიყენება SELECT წინადადება ცვლადისათვის მონაცემის მისანიჭებლად.

```
DECLARE @TestVariable AS NVARCHAR(100)
SELECT @TestVariable =N'მე მომწონს SQL SERVER'
PRINT @TestVariable
```

შესაძლებელია რამდენიმე ცვლადის გამოცხადება:

```
declare @var1 nvarchar(40)=N'თანამშრომელთა რაოდენობა'
Declare @var2 int=10
Print @var1+' '+cast (@var2 as nvarchar(20))
```

ან

```
declare @var1 nvarchar(40)=N'თანამშრომელთა რაოდენობა'
Declare @var2 int=10
Print @var1+' '+convert(nvarchar(40), @var2)
```

ცვლადების თავისებურებები:

- ცოცხლობს მხოლოდ მოთხოვნის ბლოკში, სადაც მოხდა მისის გამოცხადება. მისი სასიცოცხლო ციკლი მთავდება GO-მდე
- თუ ცვლადს მივანიჭებთ მნიშვნელობას ცხრილიდან, რომელიც Select -ით აბრუნებს ერთზე მეტ ჩანაწერს, ამ შემთხვევაში ცვლადისათვის მინიჭებული მნიშვნელობა იქნება ბოლო მწკრივი.
- თუ დეკლარირებული ცვლადი მონაცემთა ტიპები და მინიჭებული მნიშვნელობის მონაცემთა ტიპები ერთმანეთს არ ემთხვევა, SQL Server ახდენს იმპლიციტურ კონვერტაციას<sup>1</sup> მნიშვნელობის მინიჭების პროცესში, თუ ეს შესაძლებელია. დაბალი პრიორიტეტის მონაცემთა ტიპი გარდაიქმნება SQL სერვერის მიერ უფრო მაღალი პრიორიტეტის მონაცემთა ტიპად, მაგრამ ამ ოპერაციამ შეიძლება გამოიწვიოს მონაცემთა დაკარგვა.

## ფუნქციები

ფუნქცია არის „შენახული პროგრამა“, რომელსაც შეუძლია მიიღოს პარამეტრი, შეასრულოს მოქმედებები და დააბრუნოს შედეგი (მნიშვნელობა). ფუნქციას შეუძლია მიიღოს რამდენიმე პარამეტრი და დააბრუნოს სკალარული მნიშვნელობა ან ცხრილი. შემავალი პარამეტრები შეიძლება იყოს ნებისმიერი ტიპის გარდა text, ntext, image, timestamp, cursor.

ჩვენ არ შეგვიძლია გამოვიყენოთ ფუნქცია მონაცემთა ბაზის ცხრილ(ებ)ში ჩანაწერების ჩასმის, განახლების და წაშლისთვის.

არსებობს ფუნქციების ორი კატეგორია:

- System Defined Function
- User Defined Function

<sup>1</sup> ავტომატური კონვერტაცია, რადგან ის ავტომატურად შესრულებულია მომხმარებლის ჩარევის გარეშე.

მომხმარებლის მიერ ხდება სამი ტიპის ფუნქციის განსაზღვრა:

- Scalar Function -სკალარული; აბრუნებს ერთ მნიშვნელობას, როგორიცაა string, integer, ან bit
- Inline Table-Valued Function -ფუნქცია, რომელიც აბრუნებს ცხრილს. აბრუნებს შედეგს select ოპერატორის ერთჯერადი გამოყენების შემთხვევაში. (იგი ჰარმოდგენას, მაგრამ უფრო მოქნილია პარამეტრების გამო);
- Multi-Statement Table-Valued Function -მრავალოპერატორიანი ფუნქცია, აბრუნებს ცხრილს, შექმნილს T-SQL ის რამდენიმე ოპერატორის მიერ.

ფუნქციის შექმნის სინტაქსი:

```
CREATE FUNCTION [schema_name.]function_name
( [ @parameter [ AS ] [type_schema_name.] datatype
    [ = default ] [ READONLY ]
    , @parameter [ AS ] [type_schema_name.] datatype
    [ = default ] [ READONLY ] ]
)
RETURNS return_datatype
[ WITH { ENCRYPTION
        | SCHEMABINDING
        | RETURNS NULL ON NULL INPUT
        | CALLED ON NULL INPUT
        | EXECUTE AS Clause }
[ AS ]
BEGIN
    [declaration_section]
    executable_section
    RETURN return_value
END;
```

### სკალარული ფუნქცია:

სკალარული ფუნქციის სინტაქსი

```
CREATE FUNCTION [schema_name.]function_name (parameter_list)
RETURNS data_type AS
BEGIN
    statements
    RETURN value
END
```

- create function ოპერატორის შემდეგ სქემის სახელი, რომლის გამოყენებაც არ არის სავალდებულო. თუ არ მოხდა სქემის სახელის მითითება, მაშინ გაჩუმებით გამოყენებული იქნება .dbo სქემა.
- შემდეგ მოდის ფუნქციის სახელი.
- შემდეგ მრგვალ ფრჩხილებში ხდება ჩამოთვლა აუცილებელი პარამეტრების, რომელთა გამოყენებაც ხდება ფუნქციაში, შემდეგ მოდის საგასაღებო სიტყვა RETURNS, რომელსაც მოყვება დასაბრუნებელი მნიშვნელობის ტიპი. კოდი, რომელიც უნდა შესარულოს ფუნქციამ,

მოთავსებულია Begin... End საგასაღებო სიტყვებს შორის. კოდში  
შესაძლებელია ნებისმიერი ოპერატორების გამოყენება.

განვიხილოთ სკალარული ფუნქციის შექმნის მაგალითი.

-Project ცხრილიდან დავაბუნოთ პროექტების ბიუჯეტების საერთო თანხა.

```
create function fn1()
returns int
As
Begin
Declare @getsum int
select @getsum=sum(budget)
from project
Return @getsum
End
```

სკალარული ფუნქციის გამოძახება ხდება **SELECT**-ით.

```
Select dbo.fn1()
```

სკალარული ფუნქციის განახლება ხდება Alter წინადადებით,

```
CREATE OR ALTER FUNCTION [schema_name.]function_name (parameter_list)
    RETURN data_type AS
    BEGIN
        statements
        RETURN value
    END
```

ხოლო წაშლა Drop ბრძნებით.

```
DROP FUNCTION [schema_name.]function_name;
```

სკალარული ფუნქციის თავისებურებანი:

- სკალარული ფუნქციები შეიძლება გამოყენებულ იქნას თითქმის ყველგან T-SQL ბლოკში.
- სკალარული ფუნქციები იღებს ერთ ან მეტ პარამეტრს, მაგრამ აბრუნებს მხოლოდ ერთ მნიშვნელობას, შესაბამისად, ისინი უნდა შეიცავდეს RETURN განაცხადეს.
- სკალარულ ფუნქციებში შეიძლება გამოყენებულ იქნას ლოგიკური IF ბლოკები ან WHILE.
- სკალარული ფუნქციები ვერ ახერხებს მონაცემების განახლებას. მათ შეუძლიათ მონაცემების წვდომა.
- სკალარული ფუნქციებს შეუძლიათ სხვა ფუნქციების გამოძახება.

შესაძლებელია ფუნქცია გამოვიყენოთ არა მხოლოდ Select-ში, არამედ იგი მივანიჭოთ ცვლადს.

მივანიჭოთ ჩვენს მიერ შექმნილი ფუნქციის მნიშვნელობა ცვლადს და დავბეჭდოთ.

```
Declare @var1 int
```

```
Set @var1=dbo.fn1()
Print @var1
```

ფუნქციის შექმნისას შესაძლებელია გამოყენებული იყოს SCHEMABINDING, ENCRYPTION

თუ ფუნქცია შექმნილია ოპციით SCHEMABINDING, მაშინ ბაზაში არ მოხდება იმ ობიექტების შეცვლა, რომლებზეც ხდება ფუნქციის მიმართვა. (ALTER ოფციის გამოყენებით). შეზღუდულია აგრეთვე DROP -ის გამოყენებაც.

როდესაც ფუნქციაში გამოყენებულია [schemabinding](#), აუცილებელია, რომ:

- ობიექტების სახელებში, რომლებზეც ხდება ფუნქციით მიმართვა აუცილებლად იყოს მითთებული სქემის სახელი.
- ფუნქცია და ობიექტები უნდა იყოს ერთ ბაზაში განთავსებული;
- მომხმარებელს, რომელიც ქმნის ფუნქციას, აქვს ყველა უფლება იმ ობიექტებთან მიმართებაში, რომლებსაც მიმართავს ფუნქცია.

### განსხვავება ფუნქციასა და პროცედურას შორის

User Defined function	Stored Procedure
ფუნქცია აუცილებლად აბრუნებს რაღაც მნიშვნელობას	პროცედურით შესაძლებელია არ მოხდეს კონკრეტული მნიშვნელობის დაბრუნება
ფუნქცია იყენებს Select წინადადებას, მასში არ ხდება DML ბრძანებების გამოყენება	პროცედურა ერთნაირად იყენებს როგორც select, ასევე DML-ის ბრძანებებს, როგორიცაა insert, update, delete
იყენებს მხოლოდ input პარამეტრს, არ აქვს მხარდაჭერა output პარამტრთან.	აქვს ორივე input და output პარამტრები

ტრანზაქციები არ არის დაშვებული ფუნქციის გამოყენების ფარგლებში	დაიშვება ტრანზაქციების გამოყენება
გამოიყენება მხოლოდ ცხრილური ტიპის ცვლადები, დროებითი ცხრილების გამოყენება არ ხდება.	გამოიყენება ორივე: ცხრილური ტიპის ცვლადებიც და დროებითი ცხრილებიც.

ფუნქციიდან არ ხდება პროცედურის გამოძახება	პროცედურით შესაძლებელია ფუნქციის გამოძახება
---	---

## ცვლადები და ფუნქციები

ფუნქცია შესაძლებელია გამოძახებული იქნას <b>select</b> წინადადებით	პროცედურა შესაძლებელია გამოძახებულ იქნას <b>Select/Where/Having</b> წინადადებებით. მის
შესაძლებელია გამოყენებული იქნეს შეერთების ბრძანებაში, როგორც შედეგი	პროცედურის გამოყენება შეერთების ბრძანებაში არ ხდება

## დროის და თარიღის ფუნქციები

ველების მონაცემების ტიპების შეცვლა  
ფუნქციები CAST, CONVERT

ამ ფუნქციების გამოყენებით შესაძლებელია მონაცემების ტიპის შეცვლა. პირველი განსხვავება ამ ორ ფუნქციას შორის არის ის, რომ CAST არის ANSI სტანდარტი, ხოლო CONVERT არის SQL Server-ის სპეციფიური ფუნქცია. CONVERT ფუნქცია გამოიყენება date/time, data type, და money ტიპების შესაცვლელად, მაშინ როდესაც CAST ეს ფუნქცია არ გააჩნია.

მათი ზოგადი სინტაქსი ასეთია:

**CAST ( გამოსახულება AS მონაცემის ტიპი [ ( სიგრძე ) ] )**

**CONVERT ( ტიპი [ (სიგრძე) ], გამოსახულება [ , სტილი ] )**

CONVERT –ს შეუძლია დროის და თარიღის გარდაქმნა შესაბამისი სტილების მიხედვით.  
(ქვემოთ მოცემულია სტილების ცხრილი)

Without century (yy) <sup>1</sup>	With century (yyy)	Standard	Input/Output <sup>2</sup>
-	0 or 100 <sup>1, 2</sup>	Default	mon dd yyyy hh:miAM (or PM)
1	101	U.S.	mm/dd/yyyy
2	102	ANSI	yy.mm.dd
3	103	British/French	dd/mm/yy
4	104	German	dd.mm.yy
5	105	Italian	dd-mm-yy
6	106 <sup>(1)</sup>	-	dd mon yy
7	107 <sup>(1)</sup>	-	Mon dd, yy
8	108	-	hh:mi:ss
-	9 or 109 <sup>1, 2</sup>	Default + milliseconds	mon dd yyyy hh:mi:ss:mmmAM (or PM)
10	110	USA	mm-dd-yy
11	111	JAPAN	yy/mm/dd
12	112	ISO	ymmd
-	13 or 113 <sup>1, 2</sup>	Europe default + milliseconds	dd mon yyyy hh:mi:ss:mmm(24h)
14	114	-	hh:mi:ss:mmm(24h)
-	20 or 120 <sup>(2)</sup>	ODBC canonical	yyyy-mm-dd hh:mi:ss(24h)
-	21 or 121 <sup>(2)</sup>	ODBC canonical (with milliseconds)	yyyy-mm-dd hh:mi:ss.mmm(24h)

```
select convert (date, '10/04/2022 12:34:22',101)
select convert (date, '10/04/2022 12:34:22',103)
select convert (datetime,'10/04/2022 12:34:22',105)
```

1. --GETDATE ()

აბრუნებს მიმღინარე დროს.

```
SELECT GETDATE()
```

2. --DATEADD (datepart, number, date)

```
SELECT DATEADD(DAY, 30, GETDATE())
```

(დაუმატებს 30 დღეს)

3. --DATEDIFF (datepart, startdate, enddate)

```
SELECT DATEDIFF(DAY, '01/01/2022', GETDATE())
1240
```

4. --DATEPART (datepart, date)

```
SELECT DATEPART(MONTH, GETDATE())
SELECT DATEPART(day, GETDATE())
```

5. -- DATENAME (datepart, date)

```
SELECT DATENAME(MONTH, GETDATE())
```

6. ISDATE (expression)

```
--The ISDATE() function checks an expression and returns 1 if it is a valid date,  
otherwise 0  
SELECT ISDATE (getdate())  
SELECT ISDATE('Hello world!')
```

სტილების ცხრილი: თუ მონაცემის ეკუთვნის Date ან Time ტიპს, მაშინ შეიძლება გამოყენებულ იქნეს ცხრილში მოცემული სტილები.  
განვიხილოთ დროის და თარიღის ფუნქციის გამოყენების სხვადასხვა მაგალითი:

```
/*CURRENT_TIMESTAMP is an ANSI SQL function whereas GETDATE  
is the T-SQL version of that same function*/
```

```
select GETDATE() as 'მიმდინარე თარიღი'  
select Current_Timestamp as 'მიმდინარე თარიღი და დრო'
```

```
select year('12/05/2022') as 'წელი'  
select month('12/05/2022') as 'თვე'  
select day('12/05/2022') as 'რიცხვი'
```

```
/*  
*/
```

```
select Sysdatetime()  
select GETDATE()
```

```
SELECT CAST (GETDATE() AS DATE)  
SELECT CAST (GETDATE() AS time)
```

```
select  
    CAST(getdate() as datetime),  
    cast(getdate() as datetime2)
```

```
SELECT DATENAME(year, getdate()),  
       DATENAME(month, '12/05/2022')  
select DATENAME(day, '12/05/2022')
```

```
SELECT GETDATE() as getdate,  
DATENAME(qq, GETDATE()) AS Quarter,  
DATENAME(mm, GETDATE()) AS Month,  
DATENAME(dw, GETDATE()) AS Weekday,  
DATENAME(hh, GETDATE()) AS Hour,  
DATENAME(mi, GETDATE()) AS Minute,  
DATENAME(ss, GETDATE()) AS Seconds;
```

```
----
```

```
SELECT GETDATE() as getdate,  
DATEPART(dy, GETDATE()) AS DayOfYear,  
DATEPART(dd, GETDATE()) AS DayNum,  
DATEPART(ww, GETDATE()) AS WeekNum,  
DATEPART(dw, GETDATE()) AS Weekday,  
DATEPART(hh, GETDATE()) AS Hour,  
DATEPART(mi, GETDATE()) AS Minute,  
DATEPART(ss, GETDATE()) AS Seconds;
```

```
SELECT GETDATE() as getdate,  
DATEADD(yy, 3, GETDATE()) AS AddedYears,  
DATEADD(mm, 6, GETDATE()) AS AddedMonths,  
DATEADD(dd, 6, GETDATE()) AS AddedDays;
```

```
SELECT FORMAT (getdate(), 'dd/MM/yyyy ')
SELECT FORMAT (getdate(), 'MM/dd/yyyy ')
```

ფორმატების ცხრილი

Query	Sample output
SELECT FORMAT (getdate(), 'dd/MM/yyyy ') as date	21/03/2021
SELECT FORMAT (getdate(), 'dd/MM/yyyy, hh:mm:ss ') as date	21/03/2021, 11:36:14
SELECT FORMAT (getdate(), 'dddd, MMMM, yyyy') as date	Wednesday, March, 2021
SELECT FORMAT (getdate(), 'MMM dd yyyy') as date	Mar 21 2021
SELECT FORMAT (getdate(), 'MM.dd.yy') as date	03.21.21
SELECT FORMAT (getdate(), 'MM-dd-yy') as date	03-21-21
SELECT FORMAT (getdate(), 'hh:mm:ss tt') as date	11:36:14 AM
SELECT FORMAT (getdate(), 'd','us') as date	03/21/2021
SELECT FORMAT (getdate(), 'yyyy-MM-dd hh:mm:ss tt') as date	2021-03-21 11:36:14 AM
SELECT FORMAT (getdate(), 'yyyy.MM.dd hh:mm:ss t') as date	2021.03.21 11:36:14 A
SELECT FORMAT (getdate(), 'dddd, MMMM, yyyy','es-es') as date --Spanish	domingo, marzo, 2021
SELECT FORMAT (getdate(), 'dddd dd, MMMM, yyyy','ja-jp') as date --Japanese	日曜日 21, 3月, 2021

## STRING FUNCTIONS

სტრიქონთან სამუშაო ფუნქციები)

### ASCII

ეს არის სტანდარტული სიმბოლოების კოდი. ასო ინახება როგორც რიცხვი, თითო ასო არის დაშიფრული. მისი სინტაქსია:

**ASCII (<character\_expression>)**

მაგ:

```
select ascii ('A') - 65
```

**CHAR** ეს არის ASCII –ის შებრუნებული ვარიანტი, ე.ი.გადაჰყავს სიმბოლო ASCII კოდებში  
**ASCII (<expression>)**

**CHARINDEX** სტრიქონში ეძებს ქვესტრიქონს,

**CHARINDEX(<expression>,<character\_string>[‘start\_location’])**

expression საძიებელი სტრიქონი, character\_string-სტრიქონი, სადაც ვეძებთ, start\_location სიმბოლოთა პოზიცია, საიდანაც უნდა დავიწყოთ ძებნა

```
select charindex('ma','matematika',2)
```

```
5
```

**DIFFERENCE-** აბრუნებს განსხვავებას **soundex**-ის ორ მნიშვნელობას შორის, თუ ორი სიტყვა ჟღერადობით ერთნაირია მაშინ მნიშვნელობა=4-ს, =4, თუ არა 0,

**DIFFERENCE(<expression1>,<expression2>)**

```
select difference('blue','blow')
```

```
4
```

**LEFT** სტრიქონში ეძებს სიმბოლოებს მარცხენა ბოლოდან მითითებული რაოდენობით.

**LEFT (<expression>,integer)**

```
select left ('tbilisi', '2')
```

```
tb
```

**LEN-** ითვლის სიმბოლოთა რაოდენობას.

**LEN (<expression>)**

**LOWER** expression-ში მითითებულ გამოსახულებაში ახდენს სიმბობლოების რეგისტრების შეცვლას, ზედა გადაჰყავს ქვედაში.

**LOWER(<expression>)**

**LTRIM**-სპობს ცარიელ ადგილებს (პრობელებს) მარცხნიდან

**LTRIM(<characteer\_exspresion>)**

**NCHAR**-აბრუნებს სიმბოლოებს უნიკოდებში.

**NCHAR(<integer\_code>)**

პარამეტრი (`<integer_code>`) უნდა იყოს დადებითი მთელი რიცხვი 0 - 65 535-მდე

```
select nchar('3')
```

### **PATINDEX**

სტრიქონში ეძებს ქვესტრიქონის პოზიციას შაბლონის საფუძველზე. თუ ასეთი სტრიქონი ვერ მოძებნა, აბრუნებს 0-ს

**PATINDEX('<%pattern%>',<expression>)**

`%pattern%` არის შაბლონი, რასაც ეძებს, **expression** სიმბოლოთა მიმდევრობა, სადაც ეძებს. :

```
select patindex('%ma%', 'matematika')
```

**1**

**QUOTENAME** –მისი გამოყენებით შესაძლებელია სიტყვის ჩაწერა სასურველ სიმბოლოებში.( ბრჭყალებში, კვადრატულ ფრჩხილებში, ერთმაგ ბრჭყალებში)

**quotename('<character\_string>'['<quote\_character>'])**

**character\_string** –უნიკოდში ჩაწერილი სტრიქონია, **quote\_character**-შემომსაზღვრელი სიმბოლოები

```
select quotename('hello','[]')  
[hello]
```

**REPLACE**– მოძებნის პირველში, მეორეში სიმბოლოთა მიმდევრობას და შეცვლის მესამეთი.

**RAPLACE ('<string\_exspression1>', '<string\_exspression2>', '<string\_exspression3>')**

```
select ename, replace(job,'man','women')
```

```
from emp
```

```
SMITH    CLERK  
ALLEN   SALESwomen  
WARD     SALESwomen
```

**REPLICATE** –იმეორებს სიმბოლოებს მითითებული რაოდენობით.

**REPLICATE (<character\_expression>,<integer>)**

```
select replicate('hello',3)
```

**hellohellohello**

პრობელების გასაკეთებლად იწერება +

```
select replicate('hello'+ ' ',3)
```

**hello hello hello**

**REVERSE** –უკუღმა ბეჭდვა

**REVERSE (<character\_expression>)**

```
select reverse('tbilisi')
```

**isilibt**

**RIGHT**-იგივეა, რაც **LEFT** მხოლოდ სიმბოლოები მითითებული რაოდენობით გამოაქვს მარჯვნიდან.

**RIGHT(<character\_expression>,<integer>)**

```
select right('tbilisi',5)
```

ilisi

**RTRIM**-სპობს პრობელებს მარჯვნიდან

**RTRIM(<character\_expression>)**

LTRIN-ის შებრუნებული ვარიანტი

**SOUNDEX** –აბრუნებს ჯდერადობას ოთხსიმბოლოაინი კოდით

**SOUNDEX(<character\_expression>)**

**select soundex('tbilisi')**

**T142**

**SPACE** –იმეორებს პრობელებს:

**SPACE(<integer>)**

**select 'hello'+space(7)+'**

**world**

STR-რიცხვითი მონაცემი გადაჰყავს ტექსტურში

**STR(<numeric expression>[,<length>,[<desimal>]])**

**select ename+ ' +job,' '+str(sal)**  
**from emp**

<b>SMITH CLERK</b>	<b>800</b>
<b>ALLEN SALESMAN</b>	<b>1600</b>
<b>WARD SALESMAN</b>	<b>1250</b>
<b>JONES MANAGER</b>	<b>2975</b>

**STUFF**- შლის ტექსტში მითითებულ სიმბოლოთა რაოდენობას მითითებული პოზიციიდან და ცვლის მას სასურველი სიმბოლოებით.

**STUFF(<expression>,<start>,<length>)**

**select ename, stuff(job,6,2,'...')**  
**from emp** (ჩაშლის-მე-6 დან 2 სიმბოლოს შეცვლის სამი წერტილით.)

<b>SMITH</b>	<b>NULL</b>
<b>ALLEN</b>	<b>SALES...N</b>
<b>WARD</b>	<b>SALES...N</b>
<b>JONES</b>	<b>MANAG...</b>

**SUBSTRING**-

სტრიქონში ეძებს ქვესტრიქონს. ვუთითებთ სად ვეძებთ, რომელი პოზიციიდან და რამდენ სიმბოლოს

**SUBSTRING(<expression>,<start>,<length>)**

**select ename, substring(job,1,3)**  
**from emp**

<b>SMITH</b>	<b>CLE</b>
<b>ALLEN</b>	<b>SAL</b>
<b>WARD</b>	<b>SAL</b>

UNICODE-აკეთებს იგივე რასაც CHAR

**UNICODE(<character\_expression>)**

```
select unicode('t')
```

```
116
```

UPPER-ქვედა რეგისტრის სიმბოლოები გადაპყავს ზედაში

**UPPER(<character\_expression>)**

ტექსტებთან სამუშაო (სიმბოლური) ფუნქციები

დააბრუნეთ ინგლისური პატარა ასოები '

\* /

```
declare @A int
set @A=65
while @A<=90
begin
    print char(@A)
    set @A=@A+1
end
```

--- ფუნქცია ascii

```
select ASCII('d') as ' d - ს შესაბამისი კოდია'
Print ASCII('d')
```

სავარჯიშო 2

/\* ფუნქცია ascii - ის და char - ის გამოყენებით დააბრუნეთ  
ინგლისური პატარა ასოები\*/

```
declare @A int
set @A=ASCII('a')
while (@A<=ASCII('z'))
begin
    print char(@A)
    set @A=@A+1
end

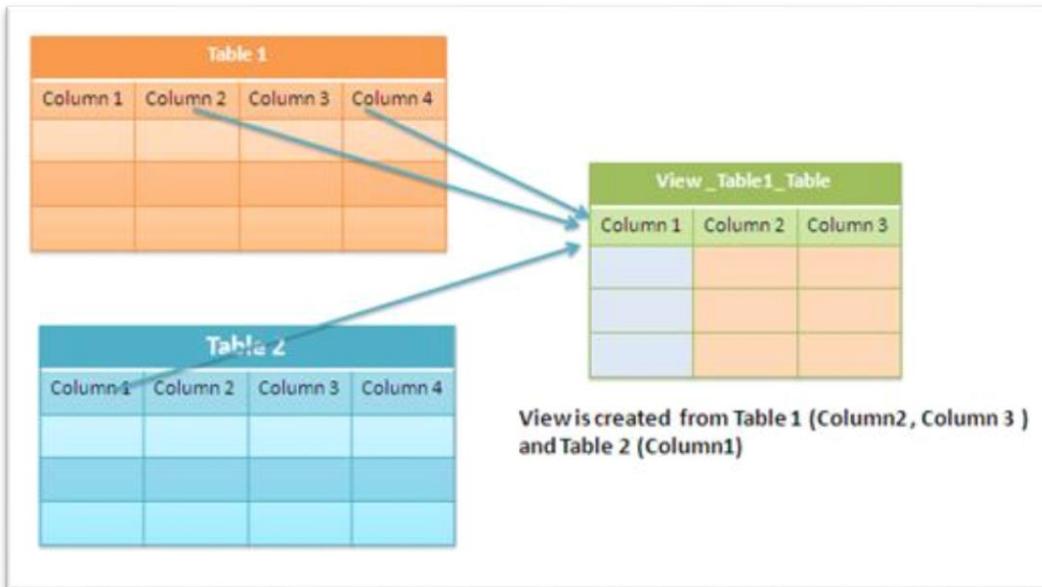
select ASCII(d)
select ASCII(0)
```

წარმოდგენები( ვირტუალური ცხრილები) და დროებითი ცხრილები

## ვირტუალური და დროებითი ცხრილები

### View არის ვიტუალური ცხრილი,

რომელიც იქმნება ბაზაში არსებული ძირითადი ცხრილის მონაცემებით კონკრეტული მოთხოვნის საფუძველზე.



შეიძლება ითქვას,

რომ წარმოდგენა სბაზაში აქვს უფორო დაცვის ფუნქცია. მისი საშუალებით ხორციელდება შემდეგი ოპერაციები:

- ცხრილის კონკრეტული ველების მონაცემებთან წვდომის შეზღუდვა და შესაბამისადგან საზღვრული ინფორმაციის მიღება ცხრილიდან.
- სხვადასხვა ცხრილებში არსებული მონაცემების შეერთება და შედეგის წარმოდგენა, როგორც ერთი მთლიანი ცხრილის.
- გარდა ქმნების შედეგად მიღებული ინფორმაციის წარმოდგენა.

წარმოდგენით შესაძლებელია განხორციელება ისეთი მოქმედებების,

როგორიცაა: Insert, Delete, Update

მისი სინტაქსი ასეთია:

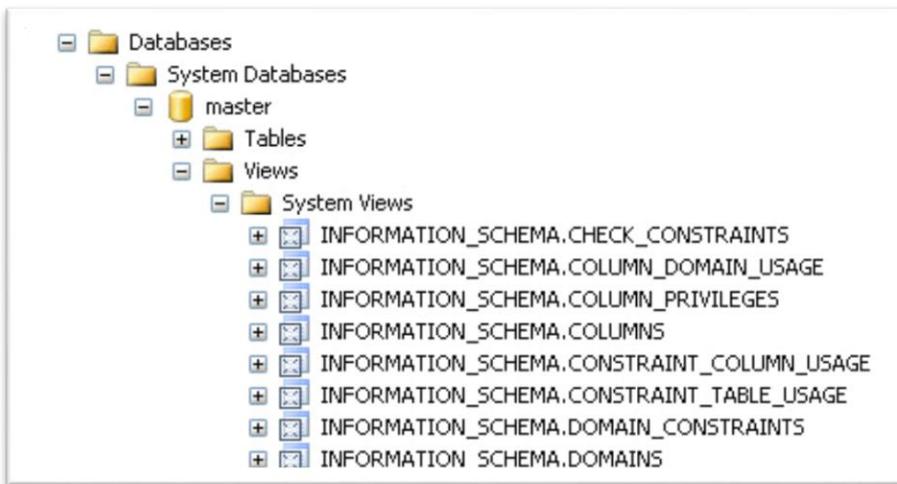
```
CREATE VIEW view_name [(column_list)]
[WITH {ENCRYPTION | SCHEMABINDING | VIEW_METADATA}]
AS
select_statement
[WITH CHECK OPTION]
```

წარმოდგენები( ვირტუალური ცხრილები) და დროებითი ცხრილები

WITH CHECK OPTION თვისება უზრუნველყობს ყველა UPDATE და INSERT ოპერაციების შესრულებას წარმოდგენით.

არსებობს წარმოდგენის ორი განსხვავებული ტიპი:

- System Views
  - Information Schema View
  - Catalog View
  - Dynamic Management View (DMV)
- User Defined Views
  - Simple View
  - Complex View



ხდება სისტემური წარმოდგენების კატეგორიზაცია:

- Information Schema View
- Catalog View
- Dynamic Management View (DMV)

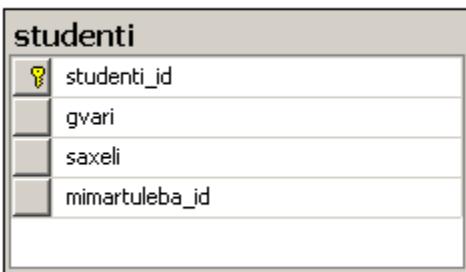
### Information View

ეს არის წარმოდგენების ძალიან მნიშვნელოვანი ჯგუფი, რომელიც გამოიყენება ფიზიკური ინფორმაციის გამოსატანად ბაზიდან.(მაგ. ცხრილები და სვეტები). ამ ჯგუფის წარმოდგენის სახელი განისაზღვრება შემდეგი სინტაქსით:

INFORMATION\_SCHEMA.[View Name].

მაგ: მოცემულია ცხრილი (სურათის მიხედვით)

წარმოდგენები( ვირტუალური ცხრილები) და დროებითი ცხრილები



გავუშვამოთხოვნა:

```
SELECT*
FROM INFORMATION_SCHEMA.COLUMNS
Where TABLE_NAME='studenti'
```

The screenshot shows the results of the SQL query. The query itself is displayed at the top, followed by a results grid. The results grid has columns: TABLE\_CATALOG, TABLE\_SCHEMA, TABLE\_NAME, COLUMN\_NAME, ORDINAL\_POSITION, COLUMN\_DEFAULT, IS\_NULLABLE, DATA\_TYPE, and CHARACTER\_MAXIMUM\_LENGTH. The data shows four columns for the 'studenti' table: 'studenti\_id' (int, NULL, NO, int, NULL), 'gvari' (nvarchar, 50, YES, nvarchar, 50), 'saxeli' (nvarchar, 50, YES, nvarchar, 50), and 'mimartuleba\_id' (int, NULL, YES, int, NULL).

TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	COLUMN_NAME	ORDINAL_POSITION	COLUMN_DEFAULT	IS_NULLABLE	DATA_TYPE	CHARACTER_MAXIMUM_LENGTH
studenti	dbo	studenti	studenti_id	1	NULL	NO	int	NULL
studenti	dbo	studenti	gvari	2	NULL	YES	nvarchar	50
studenti	dbo	studenti	saxeli	3	NULL	YES	nvarchar	50
studenti	dbo	studenti	mimartuleba_id	4	NULL	YES	int	NULL

## Catalog View

ამ ტიპის წარმოდგენები გამოიყენება როგორც ბაზის „თვითაღმწერი“ ინფორმაციის გამოსატანად. მაგ

```
select* from sys.tables
where name='studenti'
```

შედეგი:

The screenshot shows the results of the sys.tables query. The results grid has columns: name, object\_id, principal\_id, schema\_id, parent\_object\_id, type, type\_desc, create\_date, modify\_date, is\_ms\_shipped, is\_published, and is\_schema\_published. There is one row for the 'studenti' table, with values: name='studenti', object\_id=2105058535, principal\_id=NULL, schema\_id=1, parent\_object\_id=0, type='U', type\_desc='USER\_TABLE', create\_date='2013-04-04 15:59:25.347', modify\_date='2013-04-04 16:20:14.670', is\_ms\_shipped=0, is\_published=0, and is\_schema\_published=0.

name	object_id	principal_id	schema_id	parent_object_id	type	type_desc	create_date	modify_date	is_ms_shipped	is_published	is_schema_published
studenti	2105058535	NULL	1	0	U	USER_TABLE	2013-04-04 15:59:25.347	2013-04-04 16:20:14.670	0	0	0

წარმოდგენები( ვირტუალური ცხრილები) და დროებითი ცხრილები

## Dynamic Management View

ამ ჯგუფის წარმოდგენებით ხდება ინფორმაციის გამოტანა ბაზის ადმინისტრირების შესახებ სერვერის მიმდინარე კონექტირების პირობებში.

```
SELECT  
connection_id,  
session_id,client_net_address,  
auth_scheme  
FROMsys.dm_exec_connections
```

## User Defined View

ამ ტიპის წარმოდგენა იქმნება იუზერის მიერ.

შეზღუდვები, რომლებიც გამოიყენება წარმოდგენის შექმნისას:

- არ მუშაობს `orderby`
- წარმოდგენა არ იქმნება დროებით ცხრილებზე;
- წარმოდგენაში არ გამოიყენება პარამეტრი,

განვიხილოთ წარმოდგენის შექმნის მაგალითები.

```
ifexists(select name from sys.views  
where name='emp_dept')  
dropview emp_dept  
go  
go  
createview emp_dept  
as  
select ename, deptno  
from emp  
where deptno=20  
  
select*from emp_dept  
-----  
  
createview empdeptname  
as  
select ename, dept.deptno, dname  
from emp innerjoin dept on emp.DEPTNO=dept.deptno  
where dept.deptno=20  
select*from empdeptname  
  
go  
createview empwitoutsalary
```

წარმოდგენები( ვირტუალური ცხრილები) და დროებითი ცხრილები

```
as
select ename, job, deptno
from emp

select*from empwithoutsalary
-----
```

შესაძლებელია წარმოდგენის შექმნა წარმოდგენის საფუძველზე:

```
createview empwithoutsalary1
as
select*from empwithoutsalary
where DEPTNO=10

select*from empwithoutsalary1

select*from emp
```

ჩანაწერების განახლება წარმოდგენით:

```
createview empnoename
as
select empno,ename, job
from emp

select*from empnoename
-----
update empnoename
set empno=1
where empno=730
```

ჩანაწერის დამატება წარმოდგენით:

```
insertinto empnoename
values
(2,'sdfsdfs','sdff')

select*from empnoename
select*from emp
```

---ჩანაწერის წაშლა წარმოდგენით:

```
deletefrom empnoename
where empno=2

=====
createview dnamedeptno
as
```

წარმოდგენები( ვირტუალური ცხრილები) და დროებითი ცხრილები

```
select deptno, dname
from dept

select* from dnamedeptno

alteredview dnamedeptno
as
select deptno, dname
from dept
orderby deptno
```

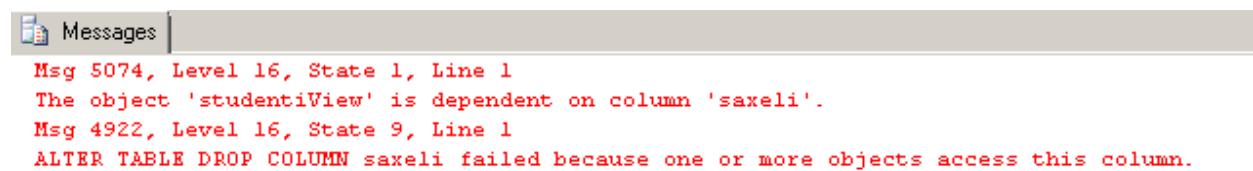
დაბრუნდება შეცდომა, რადგან მოთხოვნაში არის order by, რომელიც არ ფუნქციონირებს ვირტუალურ ცხრილში.

## SCHEMABINDING

SCHEMABINDING- თვისება ბლოკავს განახლების, წაშლის ოპერაციებს ცხრილის იმ ველებზე, რომლების მონაწილეობენ წარმოდგენის განსაზღვრაში.

```
CREATEVIEW studentiView
WithSCHEMABINDING
As
SELECT
studenti_ID,
      gvari,
      saxeli
FROM dbo.studenti

alteredtable studenti
dropcolumn saxeli
```



The screenshot shows the 'Messages' window in SQL Server Management Studio. It contains the following error messages:

```
Msg 5074, Level 16, State 1, Line 1
The object 'studentiView' is dependent on column 'saxeli'.
Msg 4922, Level 16, State 9, Line 1
ALTER TABLE DROP COLUMN saxeli failed because one or more objects access this column.
```

## Encryption

ახდენს წარმოდგენის დანსაზღვრის კოდირებას. მომხმარებელი ვერ ხედავს წარმოდგენის კოდს. ეს არის ერთ-ერთი დადებით მხარე წარმოდგენის თვისებებში. როცა წარმოდგენა კოდირებულია, მისი დეკოდირება არ ხდება.

```
CREATEVIEW stview
WithENCRYPTION
```

წარმოდგენები( ვირტუალური ცხრილები) და დროებითი ცხრილები

```
as
select studenti_id, saxeli, gvari
from studenti

execsp_helptextstview
```

პროცედურის შედეგი

