1. საინფორმაციო სისტემები
2. მონაცემთა ბაზები (მბ)
3. მუშაობა ცხრილებთან
4. ცხრილების დაპროექტება
5. ფორმები
6. მოთხოვნები
7. ანგარიშები
8. მაკროსები

თემა 1.
საინფორმაციო სისტემები

განსაზღვრებები

მონაცემთა ბაზები (მბ) –რაიმე საგნობრივი სფეროს შესახებ მონაცემების საცავი, რომელიც ორგანიზებულია სპეციალური სტრუქტურის სახით.

მნიშვნელოვანია:

- მონაცემები, რაიმე სფეროს შესახებ (არა ყველაფრის შესახებ)
- მოწესრიგებული

მონაცემთა ბაზების მართვის სისტემა (მბმს) – ეს არის მონაცემთა ბაზებთან სამუშაო პროგრამული უზრუნველყოფა.

ფუნქციები:

- მბ-ში ინფორმაციის მოძებნა
- მარტივი გამოთვლების შესრულება
- ანგარიშების გამოტანა ბეჭდვაზე
- მბ-ის რედაქტირება

საინფორმაციო სისტემა –ეს არის მბ+მბმს.

საინფორმაციო სისტემების ტიპები

- **ლოკალური საინფორმაციო სისტემები**

მბ და მბმს განთავსებულია ერთსა და იმავე კომპიუტერზე.

- **ფაილ-სერვერული**

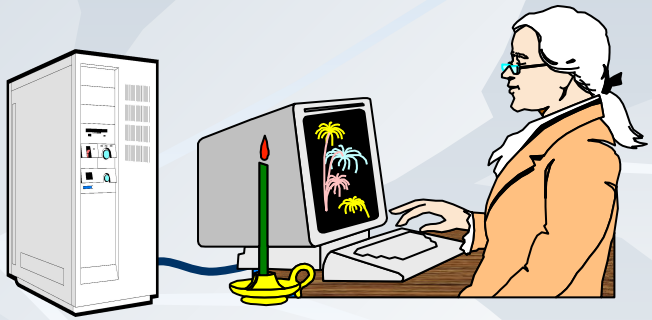
მბ განთავსებულია ქსელის სერვერზე (ფაილურ სერვერზე), ხოლო მბმს მომხმარებლის კომპიუტერზე.

- **კლიენტ-სერვერული**

მბ და ძირითადი მბმს განთავსებულია სერვერზე, ხოლო კლიენტური მბმს განთავსებულია სამუშაო სადგურზე, რომელიც უგზავნის მოთხოვნას სერვერს და ეკრანზე გამოყავს შედეგები.

ლოკალური საინფორმაციო სისტემა (ლსს)

მბ
მბმს



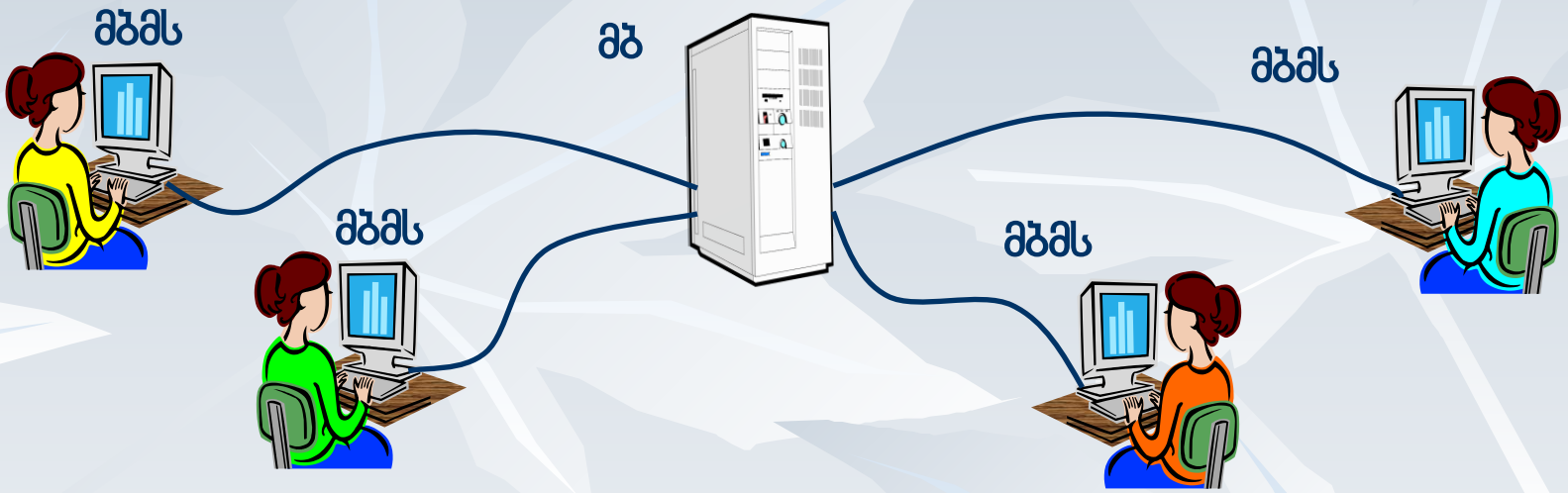
ავტონომიურობა
(დამოუკიდებლობა)

1) მბ-სთან მუშაობს მხოლოდ ერთი ადამიანი

2) მომხმარებლების დიდი რაოდენობის შემთხვევაში რთულდება განახლების პროცესი

3)პრაქტიკულად შეუძლებელია ერთდროულად რამოდენიმე მომხმარებლის მიერ შეტანილი ცვლილებების ”სინხრონიზება”

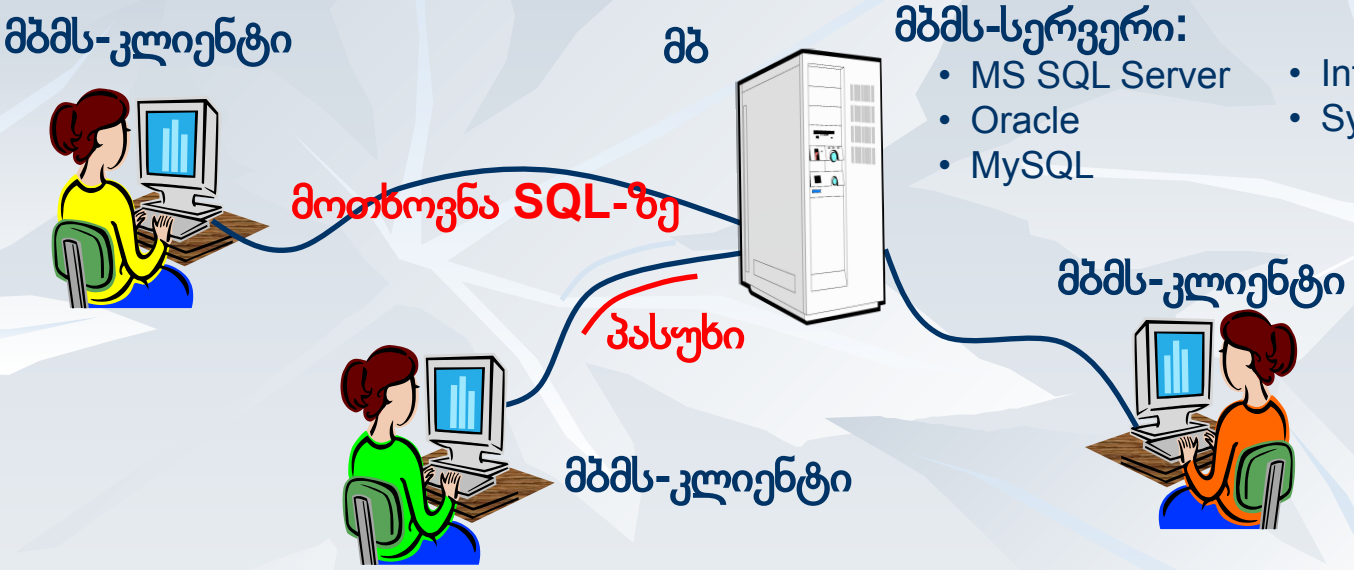
ფაილ-სერვერული სს-ები



ერთსა და იმავე ბაზასთან ერთდროულად რამოდენიმე მომხმარებელი მუშაობს

1) ძირითად სამუშაოს ასრულებენ სამუშაო სადგურები (სსადგ), ისინი მძლავრები უნდა იყვნენ
2) სტრიქონის მოსაძებნად პერსონალურ კომპიუტერზე ხდება მთელი მბ-ის კოპირება-ქსელის დატვირთვა იზრდება
3) არასანქცირებული შეღწევისაგან დაცვის დაბალი დონე
4) სხვადასხვა კომპიუტერებიდან ერთდროული ცვლილებების განხორციელების სირთულე

კლიენტ-სერვერული სს



SQL (*Structured Query Language*) – *სტრუქტურული მოთხოვნების ენა*

1) ძირითად სამუშაოს ასრულეს სერვერი. სამუშაო სადგურები დაბალი სიმძლავრის შეიძლება იყვნენ
2) მარტივია მოდერნიზაცია (მხოლოდ სერვერი)
3) ქსელში მოძრაობენ მხოლოდ საჭირო მონაცემები
4) დაცვა და უფლებების განაწილება ხდება სერვერზე (მაღალია არასანქცირებული შეღწევისაგან დაცვა)
5) წვდომის განაწილება (დავალებების რიგითობა)

1) გამართვის სირთულე
2) პროგრამული უზრუნველყოფის მაღალი ღირებულება (ათასობით აშშ დოლარი)

თემა 2. მონაცემთა ბაზები

მონაცემთა ბაზების ტიპები

- **ცხრილური მონაცემთა ბაზები**
  მონაცემები ერთი ცხრილის სახით

- **ქსელური მონაცემთა ბაზები**
  კვანძების ერთობლივობა, სადაც ყოველი ყოველთან შეიძლება იყოს დაკავშირებული

- **იერარქიული მონაცემთა ბაზები**
  მბ წარმოდგენილია მრავალდონიანი სტრუქტურის სახით

- **რელაციური მონაცემთა ბაზები (99,9%)**
  ურთიერთდაკავშირებული ცხრილების ერთობლივობა

ცხრილური მონაცემთა ბაზები

მოდელი – კართოთეკა

მაგალითები:

- უბის წიგნაკი
- საბიბლიოთეკო კატალოგი

ველები

ჩანაწერები

გვარი	სახელი	მისამართი	ტელეფონი
ბლიაძე	ნელი	თამარ მეფის 15, კორპ5, ბინა56	95-75-75
კაპანაძე	ლია	აღმაშენებლის 34, კორპ 6, ბინა32	96-76-76

1) მარტივი სტრუქტურა
2) მბ-ის ყველა დახარჩენი ტიპები იყენებენ ცხრილებს

ხშირ შემთხვევებში-მონაცემების დუბლირება:

ა.ს.ხომერიკი	მონაცემთა ბაზების საფუძვლები	200 გვ.
ა.ს.ხომერიკი	მონაცემთა ბაზების მართვის სისტემები	120 გვ.

ცხრილური მონაცემთა ბაზები

1. **ველების რაოდენობა განისაზღვრება დამპროექტებლის მიერ და მომხმარებელს არ შეუძლია მათი შეცვლა.**
2. **ნებისმიერ ველს უნდა გააჩნდეს უნიკალური სახელი.**
3. **ველებს შეიძლება გააჩნდეთ სხვადასხვა ტიპები:**
   - ტექსტური სტრიქონი (სიგრძე 255-მდე სიმბოლო)
   - ნამდვილი რიცხვები (მ.შ. წილადი ნაწილითაც)
   - მთელი რიცხვი (მაგ. 5,10,300,1500, ...)
   - ფულადი თანხა (მაგ.100ლარი, 50აშშ დოლარი, 500 რუბლი)
   - თარითი, დრო, თარიღი და დრო (მაგ.20.04.2009; 15:55; 10თებ1998.11:56:34)
   - ლოგიკური ველი (ჭეშმარიტი და მცდარი,"კი" ან "არა")
   - მრავალსტრიქონიანი ტექსტი (MEMO)
   - გამოსახულება,ბგერა ან სხვა ობიექტი (ობიექტი OLE)
4. **ველები შეიძლება იყვნენ ან არ იყვნენ შესავსებად აუცილებელნი** Noll; Not Noll
5. **ცხრილი შეიძლება შეიცავდეს ნებისმიერი რაოდენობის ჩანაწერს** (ეს რაოდენობა შეზღუდულია მხოლოდ დისკოს მოცულობით); შესაძლებელია ჩანაწერების დამატება, წაშლა, რედაქტირება, დახარისხება, ძებნა

გასაღები ველი (ცხრილის გასაღები)

გასაღები ველი (გასაღები) – ეს არის ველი (ან ველების კომბინაცია), რომელიც ცალსახად განსაზღვრავს ჩანაწერს.

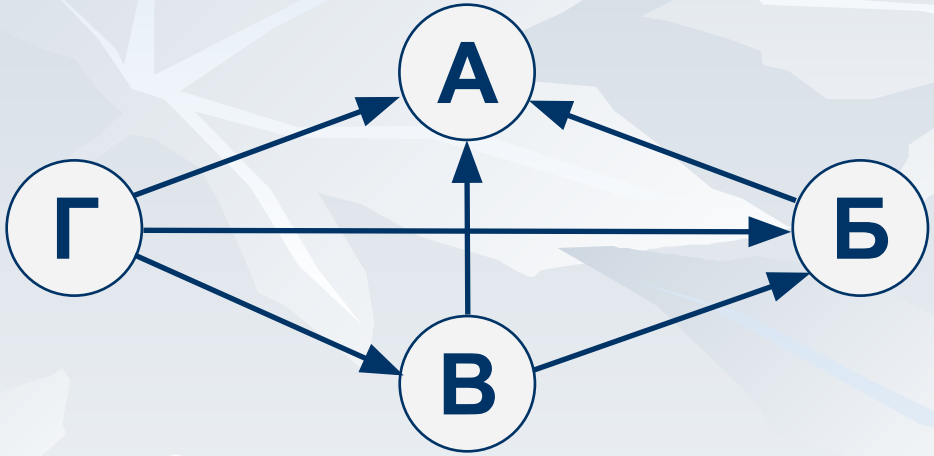
ცხრილში არ შეიძლება იყოს ერთი და იმავე გასაღები ველის მნიშვნელობის მქონე ორი ჩანაწერი

შეიძლება თუ არა ეს მონაცემები წარმოადგენდნენ გასაღებს

- გვარი
- სახელი
- პასპორტის ნომერი
- სახლის ნოერი
- მანქანის სარეგისტრაციო ნომერი
- ქალაქი სადაც ვცხოვრობთ
- სამუშაოს შესრულების თარიღი
- მანქანის მარკა

ქსელური მონაცემთა ბაზა

ქსელური მბ - ეს არის კვანძების ერთობლივობა, რომელშიც ყოველი შეიძლება დაკავშირებული იყოს ყოველთან.



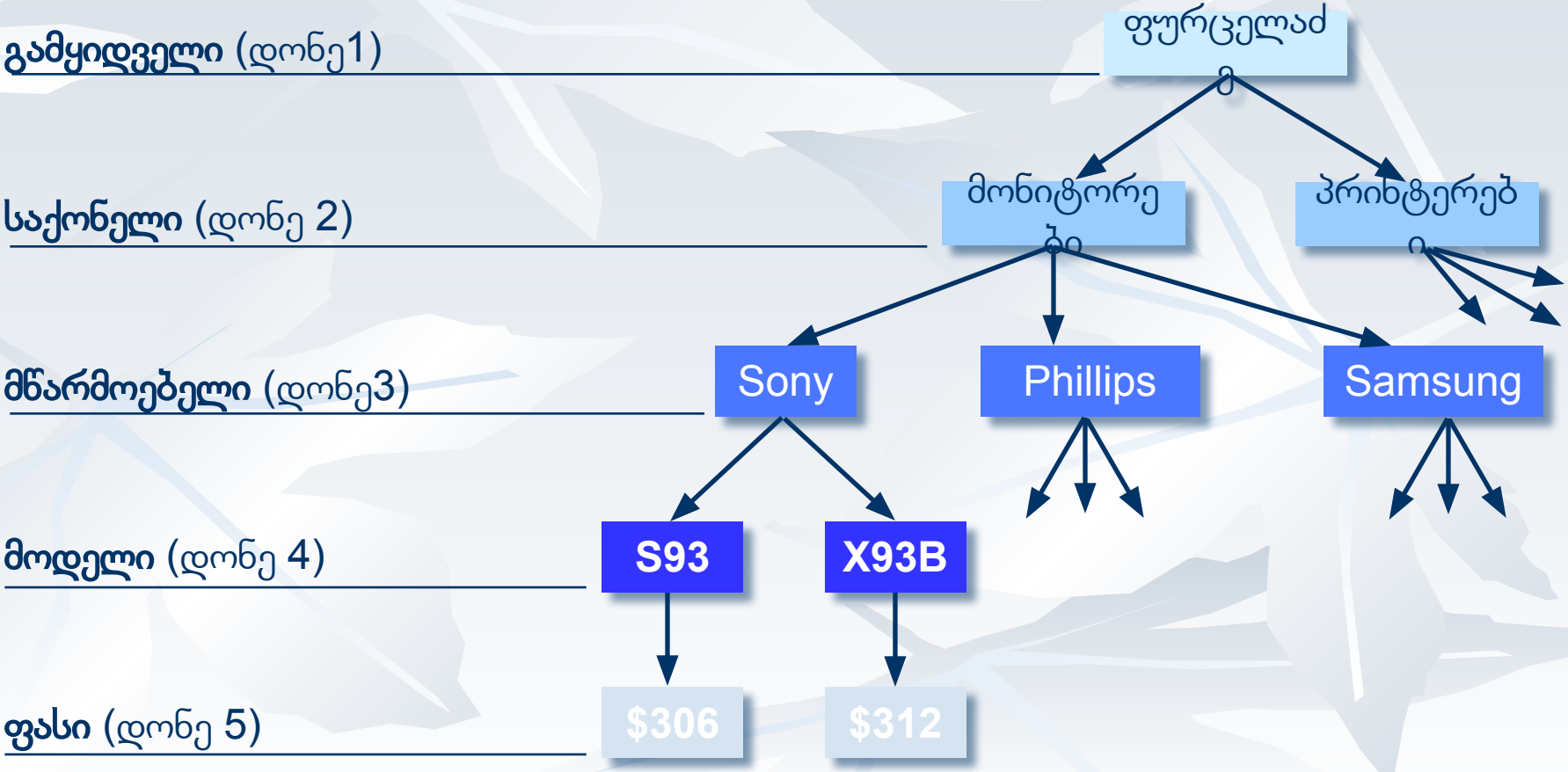
სრულად ასახავს გარკვეული ამოცანების სტრუქტურას (მაგალითად ქსელური დაგეგმარება ეკონომიკაში)

1) რთულია ყველა კავშირების შესახებ ინფორმაციის შენახვა და მათი მოძებნა
2) სტრუქტურის სირთულე (ჩახლართულობა)

შესაძლებელია ცხრილების სახით შენახვა, მაგრამ მონაცემების დუბლირებით

იერარქიული მონაცემთა ბაზა

იერარქიული მბ-ეს არის მრავალდონიანი სტრუქტურის სახით წარმოდგენილი მონაცემების ერთობლივობა



იერარქიული მონაცემთა ბაზები

ცხრილურ ფორმაზე დაყვანა:

გამყიდველი	საქონელი	მწარმოებელი	მოდელი	ფასი
ფურცელაძე	მონიტორი	Sony	S93	$306
ფურცელაძე	მონიტორი	Sony	X93B	$312
პურსელაზე	მონიტორი	Phillips	190 B5 CG	$318
ფურცელაძე	მონიტორი	Samsung	SyncMaster 193P	$452
…				

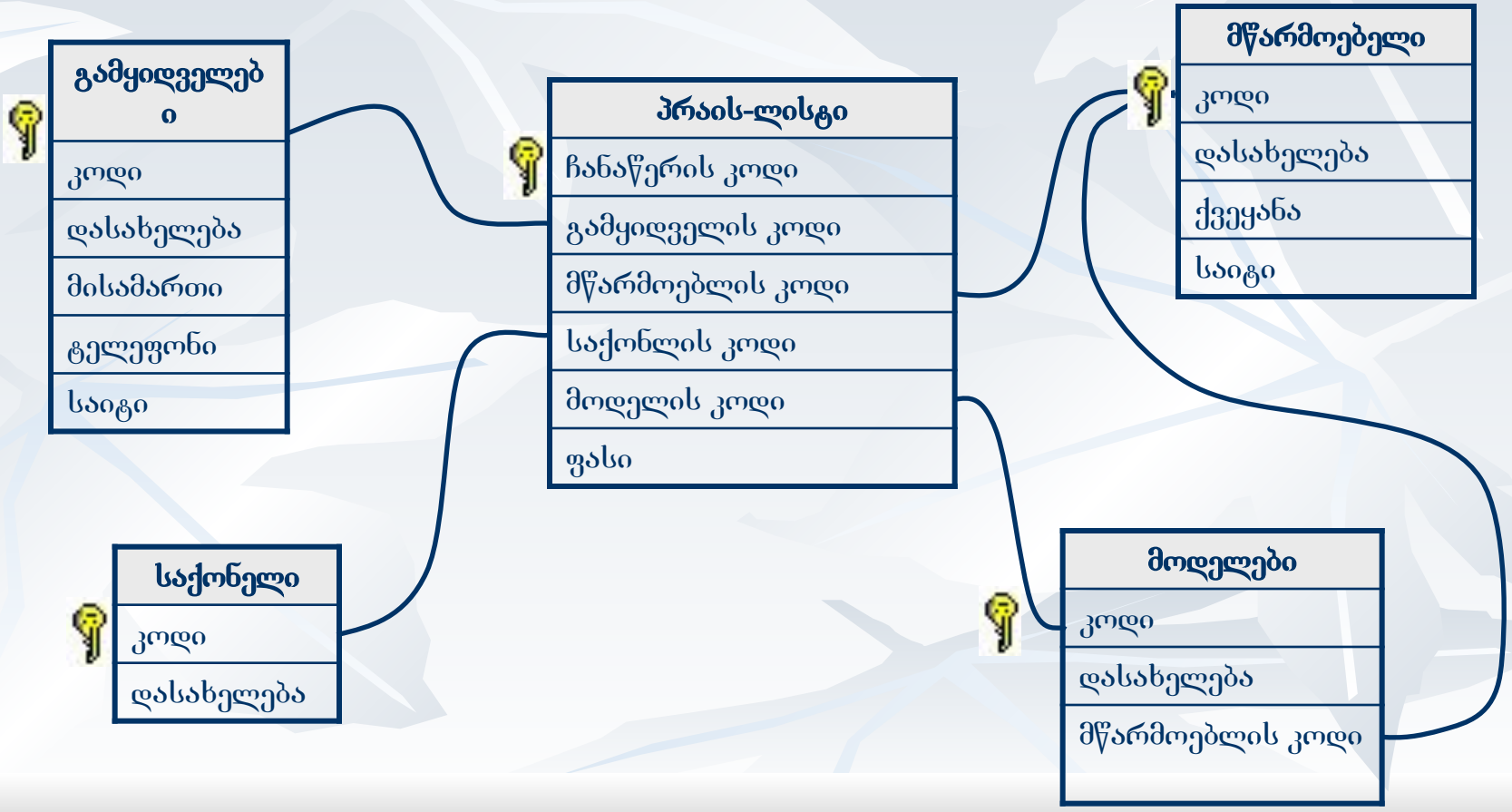
1) მონაცემების დუბლირება
2) არ არსებობს ოპერატორის შეცდომებიდან დაცვის მექანიზმი (ფურცელაძე-პურსელაზე), უკეთესი იქნებოდა სიიდან არჩევა
3) ფირმის მისამართის შეცვლისას, საჭიროა მისი შეცვლა ყველა სტრიქონში

თემა 3. მონაცემთა რელაციური ბაზები

რელაციური მონაცემთა ბაზები

1970-იანი წწ. ე. კოდდი, ინგლისურიდან. *relation – კავშირები.*

რელაციური მონაცემთა ბაზა – ეს არის მარტივი ცხრილების ერთობლივობა, რომლებიც ერთმანეთს რიცხვითი კოდების საშუალებით არიან დაკავშირებულნი.



რელაციური მონაცემთა ბაზები

1) არ ხდება ინფორმაციის დუბლირება;
2) ფირმის მისამართის შეცვლისას, საკმარისია მისი შეცვლა მხოლოდ ცხრილში **გამყიდველები**;
3) გათვალისწინებულია არასწორი შეტანისაგან დაცვა: შესაძლებელია ფირმის იმ დასახელების არჩევა, რომელიც წინასწარ არის შეტანილი ცხრილში **გამყიდველები**;
4) ტრანზაქციის მექანიზმი: ბაზაში ნებისმიერი ცვლილება ხდება მხოლოდ მაშინ, როდესაც ისინი მთლიანად დასრულებულია.

1) სტრუქტურის სირთულე (არა უმეტეს 40-50 ცხრილისა);
2) ინფორმაციის მოძებნისას საჭიროა რამოდენიმე ცხრილისათვის ერთდროული მიმართვა;
3) საჭიროა მთლიანობის დაცვა: **გამყიდველი** ფირმის წაშლისას, საჭიროა ყველა ცხრილიდან ყველა დაკავშირებული ჩანაწერების წაშლა (მბმს-ებში ეს ავტომატიურად ხდება კასკადური წაშლის მექანიზმის გამოყენებით).

კავშირები ცხრილებს შორის

ერთი-ერთთან («1-1») –ერთ ჩანაწერს პირველი ცხრილიდან შეესაბამება ზუსტად ერთი ჩანაწერი მეორე ცხრილიდან.

გამოყენება: ხშირად გამოყენებადი მონაცემების გამოყოფა.

1 — 1

კოდი	გვარი	სახელი
1	გაბუნია	ივანე
2	ესაძე	მიხეილ
…		

კოდი	დაბადების წ.	მისამართი
1	1992	დავიტაშვილის ქ.4, ბინა89
2	1993	აღმაშენებლის გამზ.56, ბინა35
…		

ერთი-მრავალთან («1- ∞») –ერთ ჩანაწერს პირველი ცხრილიდან, შეესაბამება მრავალი ჩანაწერი მეორე ცხრილიდან.

საქონელი

1 — ∞

პრაის-ლისტი

კოდი	დასახელება
1	მონიტორი
2	ვინჩესტერი
…	

კოდი	საქონლის კოდი	ფასი
123	1	10 999
345	1	11 999
…		

ცხრილებს შორის კავშირები

მრავალი-მრავალთან («∞ - ∞») – ერთ ჩანაწერს პირველი ცხრილიდან შეესაბამება მრავალი ჩანაწერი მეორე ცხრილიდა და პირიქით.

მასწავლებლები

კოდი	გვარი
1	ფერაძე
2	ესებუა
…	

∞ ∞

საგნები

კოდი	დასახელება
1	ისტორია
2	გეოგრაფია
3	ბიოლოგია
…	

რეალიზაცია – მესამე ცხრილისა და ორი «1-∞» კავშირის საშუალებით.

ცხრილი

1 ∞ ∞ 1

კოდი	გვარი
1	ფერაძე
2	ესებუა
…	

Код	Код учителя	Код предмета	Класс
1	1	1	9-А
2	1	2	8-Б
3	2	3	7-В
…			

Код	Название
1	ისტორია
2	გეოგრაფია
3	ბიოლოგია
…	

მონაცემთა ბაზების ნორმალიზაცია

ნორმალიზაცია – ეს ისეთი მონაცემტა ბაზების სტრუქტურის შემუსავებას ნიშნავს, რომელშიც არ იქნება მონაცემებისა და კავშირების სიჭარბე.

ძირითადი პრინციპები:

- არც ერთი ველი არ უნდა იყოს გაყოფადი

Фамилия и имя
თორთლაძე მერაბ
ცირამუა კობა
…

Фамилия	Имя
თორთლაძე	მერაბ
ცირამუა	კობა
	…

- არ უნდა იყოს ისეთი ველები, რომლებიც აღნიშნავენ ერთი და იმავეს (მაგალითად საქონლის) სხვადასხვა სახეობას.

წელი	ბანანები	კივი
2006	3200	1200
2007	5000	1500
…		

∞ 1

წელი	საქონლის კოდი	რაოდენობა
2006	1	1200
2007	2	1500
…		

კოდი	საქონელი
1	ბანანები
2	კივი
…	

მონაცემთა ბაზების ნორმალიზაცია

ძირითადი პრინციპები:

- ნებისმიერი ველი დამოკიდებული უნდა იყოს მხოლოდ **გასაღებზე** (გასაღები-ეს არის ველი ან ველების კომბინაცია, რომელიც ცალსახად განსაზღვრავს ჩანაწერს).

საქონელი

კოდი	დასახელება	ფასი
1	მონიტორი	2 000 р.
2	ვინჩესტერი	11 000 р.
…		

დამოკიდებულია არა მხოლოდ საქონლის დასახელებაზე!

პრაის-ლისტიტი

- არ უნდა იყოს ისეთი ველები, რომელთა მოძებნა შესაძლებელი იქნება სხვა დანარჩენი ველების საშუალებით.

კოდი	საქონელი	ერთი ტონის ფასი	რაოდენობა ტონებში	ღირებულება
1	ბანანები	1200	10	12 000
2	კივი	1500	20	30 000
…				

ძებნა მონაცემთა ბაზებში

ხაზოვანი ძებნა –ეს არის ყველა ჩანაწერის გადარჩევა, მანამ სანამ არი იქნება მოძებნილი საჭირო ჩანაწერი

კოდი	გვარი
1	თუთბერიძე
2	ზედგინიძე
…	
1024	ხომერიკი

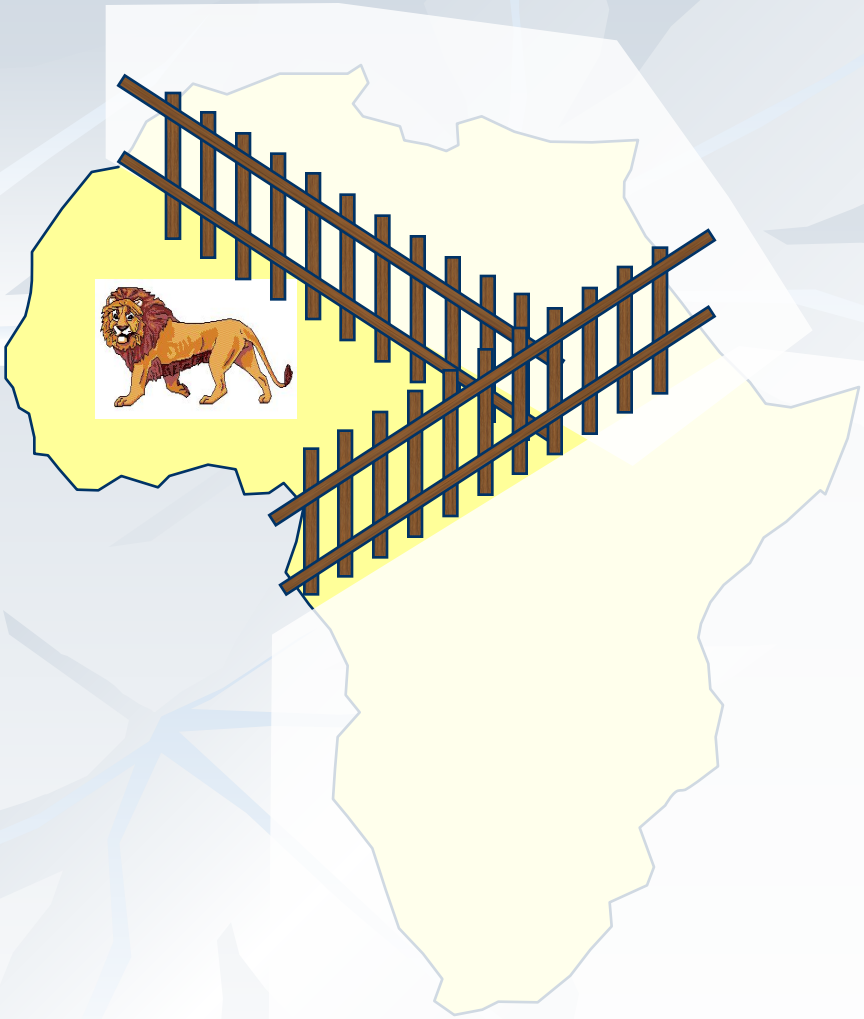
მოდებაძე?

1024 შედარება!

არ არის საჭირო მონაცემების წინასწარი მომზადება

ძებნის დაბალი სიჩქარე

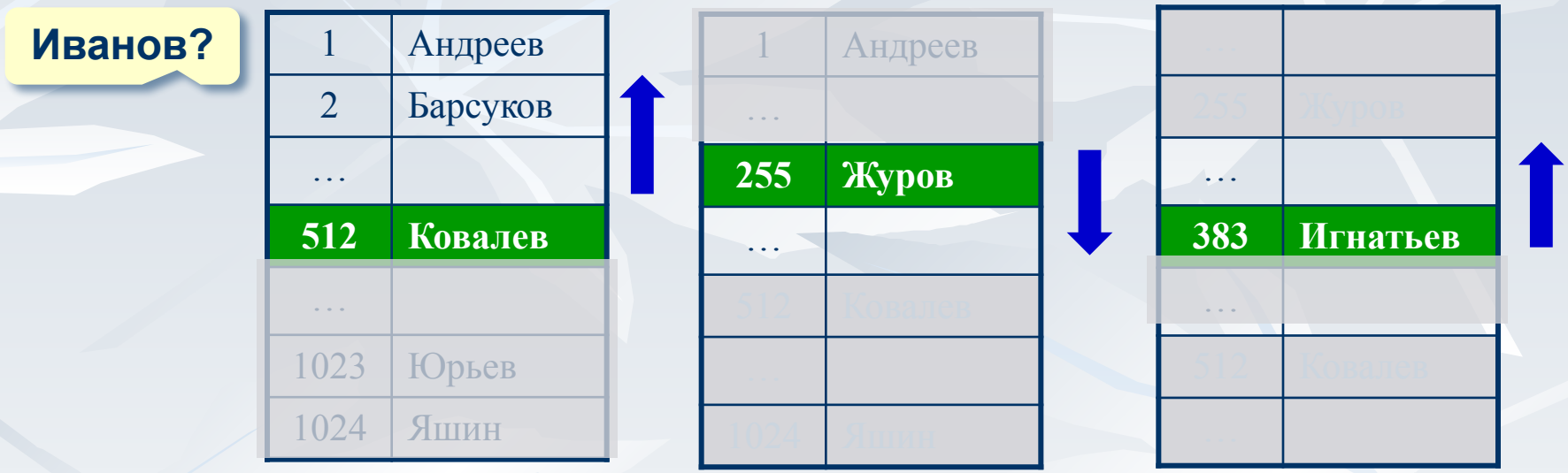
ორობითი ძებნა



1. საძიებო არე გავყოთ ორ ტოლ ნაწილად.
2. განვსაზღვროთ, თუ რომელ ნახევარშია ჩვენთვის საჭირო ობიექტი.
3. ამ ნახევრისათვის გადავიდეთ ისევ პირველ ნაბიჯზე.
4. გავიმეოროთ 1-3 ნაბიჯი სანამ არ ”დავიჭერთ” საჭირო ობიექტს.

ძებნა მონაცემთა ბაზებში

ორობითი ძებნა მბ-ში–მოითხოვს წინასწარ დახარისხებას.



რამდენი შედარებაა?

11 შედარებაა!

სწრაფი მოძებნა

1) საჭიროა ჩანაწერების დახარისხება სასურველი ველის მიხედვით;
2) შესაძლებელია გამოყენება მხოლოდ ერთი ველისათვის.

ძებნა ინდექსების მიხედვით

ინდექსი—ეს არის დამხმარე ცხრილი, რომლის დანიშნულება მდგომარეობს არჩეული სვეტის მიხედვით ძირითად ცხრილში ძიების სწრაფად განხორციელებაში.

ცხრილი

ნომერი	თარიღი	საქონელი	რაოდენობა
1	02.02.2006	კივი	6
2	01.11.2006	ბანანები	3
3	12.04.2006	აპელსინები	10

ინდექსები:

თარიღის მიხედვით

ნომერი	თარიღი
1	02.02.2006
3	12.04.2006
2	01.11.2006

საქონლის მიხედვით

ნომერი	საქონელი
3	აპელსინები
2	ბანანები
1	კივი

რაოდენობის მიხედვით

ნომერი	საქონელი
2	3
1	6
3	10

ძებნა ინდექსების მიხედვით

ძებნის ალგორითმი:

1) **ორობითი ძებნა ინდექსის მიხედვით**–საჭირო ჩანწერების ნომრების მოძებნა;
2) ძირითადი ცხრილიდან აღნიშნული ჩანაწერების არჩევა ნომრების მიხედვით.

ორობითი ძებნა ყველა სვეტის მიხედვით, რომელთათვისაც შედგენილია ინდექსები

1) ინდექსები იკავებენ ადგილს მეხსიერებაში;
2) ცხრილის შეცვლისას საჭიროა ყველა ინდექსის გადაწყობა (მბმს-ებში ხორციელდება ავტომატურად).

Access (*Microsoft Office*)-ის მონაცემტა ბაზები

გაფართოება: *.mdb, ერთი ფაილი

შემადგენლობა:

- **ცხრილები**;
- **ფორმები** –დიალოგური ფანჯრები მონაცემების შეტანისა და რედაქტირებისათვის;
- **მოთხოვნები** – მიმართვები მონაცემთა ბაზებითან საჭირო ინფორმაციის ასარჩევად ან ბაზების შესაცვლელად;
- **ანგარიშები** – ბეჭდვაზე გამოსატანი დოკუმენტები;
- **მაკროსები** –მუსაობის ავტომატიზაციის საშუალებები;
- **მოდულები** – დამატებითი პროცედურები *Visual Basic*-ზე.

MySQL-ის ბრძანებები და ოპერატორები

MySQL-ში გამოყენებული მონაცემთა ტიპები

MySQL-ის მონაცემთა ბაზის შექმნა (CREATE DATABASE)

MySQL-ის მონაცემთა ბაზის წაშლა(DROP DATABASE)

MySQL-ის მონაცემთა ბაზაში ცხრილის შექმნა (CREATE TABLE)

MySQL-ის მონაცემთა ბაზიდან ცხრილის წაშლა (DROP TABLE)

ცხრილის თვისებების შეცვლა

- ცხრილებისათვის სახელების გადარქმევა (ALTER TABLE RENAME)
- ცხრილებში სვეტების შექმნა (ALTER TABLE ADD)
- სვეტის თვისებების შეცვლა (ALTER TABLE CHANGE)
- სვეტების წაშლა (ALTER TABLE DROP)

ცხრილში სტრიქონების ჩასმა INSERT

ცხრილიდან სტრიქონების წაშლა(DELETE FROM)

ცხრილებში ჩანაწერების განახლება(UPDATE)

ცხრილებში ჩანაწერების მოძებნა(SELECT)

MySQL-ში გამოყენებული მონაცემთა ტიპები

- 1 მთელი რცხვები
- 2 წილადი რიცხვები
- 3 სტრიქონები
- 4 ბინარული მონაცემები
- 5 თარიღი და დრო

მთელი რიცხვები

მონაცემების ტიპების განსაზღვრის ზოგადი სახე:

პრეფიქსი INT [UNSIGNED]

არა აუცილებელი ალამი *UNSIGNED* განსაზღვრავს იმას, რომ შექმნილი იქმნება უნიშნო რიცხვების (დიდი რიცხვების ან ნულის) შესანახად განკუთვნილი ველი.

TINYINT	-128-დან	127-მდე
SMALLINT	-32 768-დან	32 767-მდე
MEDIUMINT	-8 388 608-დან	8 388 607-მდე
INT	-2147 483 648-დან	2 147 483 647-მდე
BIGINT	-9 223 372 036 854 775 808-დან	9 223 372 036 854 775 807-მდე

წილადი რიცხვები

- ძირითადი სინტაქსია:
- ტიპის სახელი[(length, decimals)] [UNSIGNED]

აქ:

- *length* - ციფრებისათვის განკუთვნილი ადგილების რაოდენობაა (ველის სიგანე),რომლებშიც განთავსებული იქნება წილადი რიცხვი მისი გადაცემის შემთხვევაში.
- *decimals* - ათობითი მძიმის შემდეგ მდგომი, მხედველობაში მისაღები ციფრების რაოდენობა.
- UNSIGNED - განსაზღვრავს უნიშნო რიცხვებს.

- FLOAT- მცირე სიზუსტის რიცხვი, მცოცავი მძიმით.
- DOUBLE- ორმაგი სიზუსტის რიცხვი მცოცავი მძიმით.
- REAL- სინონიმი DOUBLE-ისათვის.
- DECIMAL- სტრიქონის სახით დამახსოვრებული წილადი რიცხვი.
- NUMERIC- სინონიმი DECIMAL-ისათვის .

სტრიქონები

- **სტრიქონები წარმოადგენენ სიმბოლოების მასივებს**.

- როგორც წესი ტექსტურ ველებში ძიებისას *SELECT*-ის გამოყენებით არ ხდება რეგისტრის გათვალისწინება. ანუ სტრიქონები "Вася" და "ВАСЯ" ერთნაირად ითვლება.
- დასაწყისისათვის გავეცნოთ სტრიქონის ტიპს, რომელსაც შეუძლია დაიმახსოვროს არა უმეტეს *length* სიმბოლოებისა, სადაც *length* მიეკუთვნება დიაპაზონს 1-დან 255-მდე.

- VARCHAR (length) [BINARY]

- ასეთი ტიპის ველში რაიმე მნიშვნელობის შეტანისას მისგან ავტომატურად ამოიჭრება დამაბოლოებელი ჰარები.

- თუ მითითებულია ალამი *BINARY*, მაშინ *SELECT*-ის გამოყენებისას, სტრიქონი შედარებული იქნება რეგისტრის გათვალისწინებით.
- VARCHAR ინახავს არა უმეტეს 255 სიმბოლოს.
- TINYTEXT ინახავს არა უმეტეს 255 სიმბოლოს.
- TEXT ინახავს არა უმეტეს 65 535 სიმბოლოს.
- MEDIUMTEXT ინახავს არა უმეტეს 16 777 215 სიმბოლოს.
- LONGTEXT ინახავს არა უმეტეს 4 294 967 295 სიმბოლოს.

- ყველაზე ხშირად იყენებენ ტიპს *TEXT*, მაგრამ თუ არა ვართ დარწმუნებული, რომ მონაცემები არ იქნებიან 65 536 სიმბოლოს უნდა გამოვიყენოთ *LONGTEXT*.

ბინარული მონაცემები

- ბინარული მონაცემები **– ეს თითქმის იგივეა, რაც მონაცემები *TEXT* ფორმატი, მაგრამ ძიებისას მათში ხდება სიმბოლოების რეგისტრის გათვალისწინება.**

- TINYBLOB-ინახავს არაუმეტეს 255 სიმბოლოების. BLOB- ინახავს არაუმეტეს 65 535 სიმბოლოს.
- MEDIUMBLOB -ინახავს არაუმეტეს 6 777 215 სიმბოლოს
- LONGBLOB-ინახავს არაუმეტეს 4 294 967 295 სიმბოლოს.
- *BLOD*-მონაცემების კოდირების შეცვლა არ ხდება ავტომატურად, თუ, მოცემული კავშირის შემთხვევაში ჩართულია ტექსტის კოდირების "на лету“ შეცვლის შესაძლებლობა.

- **MySQL-ის მიერ მხარდაჭერილია ველების რამოდენიმე ტიპი, რომლებიც გათვალისწინებული არიან თარიღებისა და დროის სხვა და სხვა ფორმატში დასამახსოვრებლად.**

- DATE- თარიღი ფორმატში ᲧᲧᲧᲧ-თთ-დდ
- TIME - დრო ფორმატში სს:ᲧᲧ:ᲧᲧ
- DATETIME თარიღი და დრო ფორმატში ᲧᲧᲧᲧ-თთ-დდ სს:ᲧᲧ:ᲧᲧ
- TIMESTAMP თარიღი და დრო ფორმატში timestamp.

- მაგრამ ველი, მნიშვნელობის მიღებისას გამოისახება არა timestamp ფორმატში, არამედ ᲧᲧᲧᲧთთდდსსᲧᲧᲧᲧ სახით, რაც მნიშვნელოვნად ლახავს PHP-ში მისი გამოყენების უპირატესობებს.

CREATE DATABASE

ოპერატორის სინტაქსი

- **CREATE DATABASE [IF NOT EXISTS] db_name [CHARACTER SET charset] [COLLATE collation];**
- db_name - სახელი,რომელიც მიენიჭება შესაქმნელ მონაცემთა ბაზას.
- IF NOT EXISTS - თუ ეს პარამეტრი არ იქნება მითითებული,მაშინ უკვე არსებული სახელით მონაცემთა ბაზის შექმნის მცდელობისას, წარმოიშობა ბრძანების შესრულების შეცდომა.
- CHARACTER SET, COLLATE - გამოიყენება ცხრილისა და სორტირების წესის სტანდარტული კოდირების განსაზღვრისათვის.
  - თუ ცხრილის შექმნისას ეს პარამეტრები არ იქნება მითითებული, მაშინ ახლად შესაქმნელი ცხრილისათვის კოდირება და სორტირების წესები აიღებიან იმ მნიშვნელობებიდან, რომლებიც მითითებული იყო მთლიანი ბაზისათვის.
  - თუ მოცემულია პარამეტრი *CHARACTER SET*, მაგრამა არ არის მოცემული პარამეტრი *COLLATE*, მაშინ გამოიყენება სორტირების სტანდარტული წესი.
  - თუ განსაზღვრულია პარამეტრი *COLLATE*, მაგრამ არ არის განსაზღვრული *CHARACTER SET*, მაშინ კოდირების წესს განსაზღვრავს *COLLATE - ში მოცემული კოდირების წესის სახელწოდების პირველი ნაწილი.*
  - *CHARACTER SET-ში განსაზღვრული კოდირება მხარდაჭერილი უნდა იყოს სერვერის მიერ,* (latin1 ან sjis), ხოლო სორტირების წესი დასაშვები უნდა იყოს მიმდინარე კოდირებისათვის.

DROP DATABAS

ოპერატორის სინტაქსი

- DROP DATABASE [IF EXISTS] db_name
- db_name - განსაზღვრავს იმ მონაცემთა ბაზის სახელს, რომლიც წაშლაც არის საჭირო.
- IF EXISTS - თუ ეს პარამეტრი არ იქნება მითითებული, მაშინ არ არსებული მონაცემთა ბაზის წაშლის მცდელობისას წარმოიშობა ბრძანების შესრულების შეცდომა.
- **DROP DATABASE** ბრძანების შესრულების შედეგად იშლება როგორც თვით მონაცემთა ბაზა, ასევე მასში მოთავსებული ყველა ცხრილი.

CREATE TABLE

- **ოპერატორის სინტაქსი:**

- CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name [(create_definition,...)] [table_options] [select_statement]

- tbl_name - განსაზღვრავს იმ ცხრილის სახელს, რომელიც უნდა შეიქმნას მიმდინარე მონაცემთა ბაზაში.
- თუ **CREATE TABLE** ბრძანების გამოძახების მომენტისათვის არც ერთი ბაზა არ იქნება მიჩნეული მიმდინარედ, მაშინ წარმოიშობა ბრძანების შესრულების შეცდომა.

- **MySQL** 3.22 -დან დაწყებული შემოღებულია *db_name.tbl_name სინტაქსის საშუალებით,*
- იმ მონაცემთა ბაზის ცხადად მითითების შესაძლებლობა, რომელშიც უნდა შეიქმნას ახალი ცხრილი.

- TEMPORARY - ეს პარამეტრი გამოიყენება *tbl_name სახელით* დროებითი ცხრილის შესაქმნელად მხოლოდ მიმდინარე სეანსისათვის.
- სცენარის შესრულების დასრულებისთანავე შექმნილი ცხრილი იშლება.
- ეს შესაძლებლობა გაჩნდა **MySQL** 3.23-ში.

- **MySQL** 4.0.2-ში დროებითი ცხრილების შესაქმნელად საჭიროა CREATE TEMPORARY TABLES **პრივილეგიები** .

- IF NOT EXISTS - თუ მითითებულია ეს პარამეტრი და ხდება დუბლირებული სახელით (ანუ მიმდინარე ბაზაში ცხრილი ასეთი სახელით უკვე არსებობს) ცხრილის შექმნის მცდელობა, მაშინ ცხრილი არ შეიქმნება და არც შეცდომის შესახებ შეტყობინება არ გაჩნდება.
- წინააღმდეგ შემთხვევაში ცხრილი ასევე არ შეიქმნება, მაგრამ მივიღებთ შეტყობინებას შეცდომის შესახებ.

- აღსანიშნავია ის გარემოება, რომ ცხრილების შექმნისას ხდება მხოლოდ მათი სახელების შედარება. შიდა სტრუქტურები არ მიიღებიან მხედველობაში.

create_definition (3-1)

create_definition -განსაზღვრავს შესაქმნელი ცხრილის შიდა სტრუქტურას (ველების დასახელებები და ტიპები, გასაღებები, ინდექსები და ა.შ.)

- *create_definition* - ***ის შესაძლო სინტაქსები***:

- col_name type [NOT NULL | NULL] [DEFAULT default_value] [AUTO_INCREMENT] [PRIMARY KEY] [reference_definition]
- ან
- PRIMARY KEY (index_col_name,...)
- ან
- KEY [index_name] (index_col_name,...)
- ან
- INDEX [index_name] (index_col_name,...)
- ან
- UNIQUE [INDEX] [index_name] (index_col_name,...)
- ან
- FULLTEXT [INDEX] [index_name] (index_col_name,...)
- ან
- [CONSTRAINT symbol] FOREIGN KEY [index_name] (index_col_name,...) [reference_definition]
- ან
- CHECK (expr)

create_definition (3-2)

- **col_name** - განსაზღვრავს სვეტის სახელს შესაქმნელ ცხრილში.
- **Type** - *col_name სვეტისათვის განსაზღვრავს მონაცემების ტიპს.*
  - *type - პარამეტრის შესაძლო მნიშვნელობები*:
    - TINYINT[(length)] [UNSIGNED] [ZEROFILL]
    - SMALLINT[(length)] [UNSIGNED] [ZEROFILL]
    - MEDIUMINT[(length)] [UNSIGNED] [ZEROFILL]
    - INT[(length)] [UNSIGNED] [ZEROFILL]
    - INTEGER[(length)] [UNSIGNED] [ZEROFILL]
    - BIGINT[(length)] [UNSIGNED] [ZEROFILL]
    - REAL[(length,decimals)] [UNSIGNED] [ZEROFILL]
    - DOUBLE[(length,decimals)] [UNSIGNED] [ZEROFILL]
    - FLOAT[(length,decimals)] [UNSIGNED] [ZEROFILL]
    - DECIMAL(length,decimals) [UNSIGNED] [ZEROFILL]
    - NUMERIC(length,decimals) [UNSIGNED] [ZEROFILL]

    - CHAR(length) [BINARY]
    - VARCHAR(length) [BINARY]
    - DATE
    - TIME
    - TIMESTAMP
    - DATETIME
    - TINYBLOB
    - BLOB
    - MEDIUMBLOB
    - LONGBLOB
    - TINYTEXT
    - TEXT
    - MEDIUMTEXT
    - LONGTEXT
    - ENUM(value1,value2,value3,...)
    - SET(value1,value2,value3,...)

create_definition (3-3)

- [NOT NULL | NULL] - მიუთითებს იმაზე, შეიძლება, რომ მოცემული სვეტი შეიცავდეს მნიშვნელობას NULL თუ არა. თუ კი მითითებული არ არის, მაშინ გაჩუმებით მიიღებს მნიშვნელობას NULL (ანუ შეიძლება შეიცავდეს NULL-ს).
- [DEFAULT default_value] - განსაზღვრავს გაჩუმებით მნიშვნელობას მოცემული სვეტისათვის.
- **INSERT** ბრძანების საშუალებით ცხრილში ახალი ჩანაწერის ჩასმისას, თუ col_name ველისათვის მნიშვნელობა ცხადად არ იყო განსაზღვრული, იგი მიიღებს მნიშვნელობას *default_value*.
- [AUTO_INCREMENT] - ცხრილში ახალი ჩანაწერის ჩამატებისას, ასეთი ატრიბუტის მქონე ველი ავტომატიურად მიიღებს რიცხვით მნიშვნელობას, რომელიც 1-ით მეტი იქნება მოცემული მომენტისათვის ამ ველის ყველაზე დიდი მნიშვნელობისა..
- მოცემული შესაძლებლობა, როგორც წესი გამოიყენება სტრიქონების უნიკალური იდენტიფიკატორების გენერირებისათვის.
- სვეტს, რომლისთვისაც გამოიყენება არგუმენტი *AUTO_INCREMENT* უნდა გააჩნდეს მთელ რიცხვიანი ტიპი.
- ცხრილში შეიძლება იყოს მხოლოდ ერთი სვეტი *AUTO_INCREMENT ატრიბუტით*.
- ასევე ეს სვეტი უნდა იყოს ინდექსირებული.
- AUTO_INCREMENT- ისათვის რიცხვების თანმიმდევრობის ათვლა იწყება 1-დან. ეს მხოლოდ დადებითი რიცხვები უნდა იყოს.

ცხრილის შექმნის მაგალითები

- შემდეგი მაგალითი ქმნის 3 ველიან ცხრილს-*users*, სადაც პირველი ველი - ჩანაწერის უნიკალური იდენტიფიკატორია, მეორე ველი - მომხმარებლის სახელი, ხოლო მესამე ველი - მისი ასაკი:

```
1 CREATE TABLE
2   `users` (
3      `id` INT(11) NOT NULL AUTO_INCREMENT,
4      `name` CHAR(30) NOT NULL,
5      `age` SMALLINT(6) NOT NULL,
6      PRIMARY KEY(`id`)7    )
```

- ჩავსვათ ოთხი ახალი ჩანაწერი

```
1 INSERT INTO
2   `users` (`name`, `age`)
3VALUES
4   ('Катя', 12),
5   ('Лена', 18),
6   ('Миша', 16),
7   ('Саша', 20)
```

- გამოვიტანოთ *users* ცხრილის ყველა ჩანაწერი:

```
1   SELECT
2    *
3    FROM
4    `users`
```

AUTO_INCREMENT

შედგენილი გასაღების მეორადი სვეტისათვის

- MyISAM და BDB ცხრილებში, *შედგენილი გასაღების მეორადი სვეტისათვის, AUTO_INCREMENT პარამეტრის განსაზღვრის შესაძლებლობა არსებობს.*
- ამ შემთხვევაში ჩანაწერის გასაღები (მისი უნიკალური მნიშვნელობა) იქნება ერთდროულად ორი ველის მნიშვნელობა.
- ამასთან პირველი ველი იქნება თავისებური პრეფიქსი, ხოლო მეორე სწორედ უნიკალური რიცხვითი მნიშვნელობა ამ პრეფიქსისათვის.
- ამ თავისებურების გამოყენება მოხერხებულია, თუ საჭიროა ახალი ჩანაწერების დამატება ჯგუფების მიხედვით.
- ვნახოთ ეს თავისებურება შემდეგ მაგალითზე:
- დავუშვათ, რომ ჩვენ გვჭირდება ჩვენი კონტაქტების ჩაწერა ცხრილში. შევქმნათ ცხრილი *users, ჩანაწერებისათვის განკუთვნილი* ოთხ ველით.
- პირველი ველი შეიცავს კონტაქტის ტიპს (სახლი, სამუშაო,სწავლა);
- მეორე ველი -ჩანაწერის უნიკალური იდენტიფიკატორი;
- მესამე ველი -ადამიანის სახელი;
- მეოთხე ველი - მისი ასაკი.

ცხრილის შექმნის მაგალითი

AUTO_INCREMENT

```
1 CREATE TABLE
2   `users` (
3       `category` ENUM('სახლი', 'სამსახური', 'სასწავლებელი') NOT NULL,
4       `id` MEDIUMINT NOT NULL AUTO_INCREMENT,
5       `name` CHAR(30) NOT NULL,
6       `age` SMALLINT(6) NOT NULL,
7       PRIMARY KEY(`id`, `category`)
8   )
```

ახალი ჩანაწერების ჩასმა

- შექმნილ ცხრილში ჩავსვათ ახალი ჩანაწერები

- 01 INSERT INTO
- 02 `users` (`category`, `name`, `age`)
- 03 VALUES
- 04 ('სახლი', 'ოლია', 26),
- 05 ('სახლი', 'ნასტია', 20),
- 06 ('სამსახური', 'არტემი', 26),
- 07 ('სასწავლებელი', 'დიმა', 25),
- 08 ('სამსახური', 'საშა', 27),
- 09 ('სასწავლებელი', 'მიშა', 25),
- 10 ('სამსახური', 'ლენა', 35)

ჩანაწერების დათვალიერება

- დავათვალიეროთ *users* ცხრილის ყველა ჩანაწერი, *category* და *id* ველების მიხედვით მათი დალაგების გზით

```
1 SELECT
2   *
3 FROM
4   `users`
5 GROUP BY
6   `category`, `id`
```

მიღებული შედეგი

- **შედეგად მივიღებთ**:

```
+----------+----+-------+--------------------+
|        category     | id |  name   |age |
+----------+----+-------+--------------------+
| სახლი              |1|  ოლია     | 26 |
| სახლი              |2|  ნასტია   | 20 |
| სამსახური          |3|  არტემი   | 26 |
| სამსახური          |5|  საშა      | 27 |
| სამსახური          |7|  ლენა      | 35 |
| სასწავლებელი       |4|  დიმა      | 25 |
| სასწავლებელი       |6|  მიშა      | 25 |
+----------+----+-------+--------------------+
7 rows in set (0.00 sec)
```

- იმისათვის, რომ მივიღოთ უკანასკნელად ჩამატებული ჩანაწერის *ID*, უნდა გამოვიყენოთ **MySQL**-ის შემდეგი ბრძანება.

```
1  SELECT
2  LAST_INSERT_ID()
```

- ან API ფუნქცია **mysql_insert_id()**.

პირველადი გასაღები

- [PRIMARY KEY]-განსაზღვრავს ცხრილის პირველად გასაღებს.
  - ცხრილში მხოლოდ ერთი გასაღები ველის განსაზღვრაა შესაძლებელი.
  - პირველად გასაღებ სვეტებად მიჩნეული სვეტის არც ერთი მნიშვნელობა არ უნდა შეიცავდეს მნიშვნელობას NULL.
  - თუ კი ცხრილის შექმნისას პირველადი გასაღები ველი ცხადად არ იყო მითითებული, ხოლო პროგრამა მას მოითხოვს, მაშინ მბ MySQL ავტომატურად აყენებს *UNIQUE* პარამეტრიან პირველ სვეტს, თუ ამ სვეტის არც ერთი მნიშვნელობა არ არის NULL-ის ტოლი.
  - პირველადი გასაღების სახით შესაძლებელია განისაზღვროს როგორც ერთი, ასევე რამოდენიმე სვეტი ერთდროულად:
  - PRIMARY KEY(col_1, col_2, ...)
  - მხოლოდ ამ შემთხვევაში არც ერთი სხვა სვეტი არ შეიძლება იყოს პირველადი გასაღები, ანუ არ შეიძლება იქნას აღწერილი:
  - PRIMARY KEY(col_1), PRIMARY KEY(col_1, col_2)
  - *PRIMARY KEY ველები წარმოადგენენ ინდექსირებული ველებს (უფრო დეტალურ ინფორმაციას ინდექსების შესახებ მივიღებთ მოგვიანებით INDEX-ებისათვის მიძღვნილ ნაწილში).*

- KEY-წარმოადგენს INDEX - ის სინონიმს.
- INDEX - ი განსაზღვრავს იმ ველებს, რომელთა ინდექსირებაც არის გათვალისწინებული.
  - ველების ინდექსირება სასარგებლოა **SELECT ბრძანების მუშაობის დასაჩქარებლად.**

ინდექსები

- ინდექსების სარგებლიანობის თვალსაჩინო მაგალითია წიგნი.
- წიგნში ინდექსის როლს ასრულებ სარჩევი.
- სარჩევის მიხედვით ჩვენ სწრაფად ვპოულობთ საჭირო თავს ან პარაგრაფს.
- ინდექსირებული ველების განსაზღვრის შემთხვევაში, MySQL-ი ქმნის სპეციალურ საცავს, რომელშიც შეინახება ცხრილის ინდექსირებული ველების ყველა მნიშვნელობა და ამ მნიშვნელობების ზუსტი ადგილმდებარეობა.
- ანუ მნიშვნელობის მოძიება ხდება პრაქტიკულად მყისიერად, რაც ბუნებრივია გავლენას ახდენს სკრიპტის შესრულების სიჩქარეზე.
- სამაგიეროდ მონაცემთა ბაზის მოცულობა იზრდება დაახლოებით ორჯერ.
- **MySQL-ში შესაძლებელია ნებისმიერი ტიპის ველების ინდექსირება.**
- მუშაობის დასაჩქარებლად *CHAR და VARCHAR* ტიპის ველებში შესაძლებელია მხოლოდ რამოდენიმე პირველი სიმბოლოების ინდექსირება.
- ინდექსების განსაზღვრისას უნდა გავითვალისწინოთ, რომ მხოლოდ *MyISAM*, *InnoDB* и *BDB* *ტიპის ცხრილებშია შესაძლებელი ინდექსირებული ველის* NULL *მნიშვნელობა.*
- შეცდომებისაგან თავის დაზღვევის მიზნით, უმჯობესია, რომ ინდექსირებულ ველებს ყოველთვის მივანიჭოთ მნიშვნელობა *NOT NULL*.
- თუ პარამეტრი index_name, რომელიც განსაზღვრავს ინდექსის სახელს მითითებული არ არის, მაშინ ინდექსს მიენიჭება პირველი ინდექსირებული სვეტის სახელი.

ცხრილის ინდექსირების მაგალითი

- შემდეგ მაგალითში შევქმნათ ცხრილი *users* , *name* და *age* ველებით და მოვახდინოთ ცხრილის ინდექსირება name ველის პირველი 12 ასოს მიხედვით:

```
1 CREATE TABLE
2    `users` (
3         `name` CHAR(200) NOT NULL,
4         `age` SMALLINT(3),
5         INDEX (`name`(12))6    )
```

- თუ *CHAR* და *VARCHAR* *სვეტებისათვის მხოლოდ სასურველი იყო სვეტების ნაწილის ინდექსირება, TEXT* და *BLOB* ტიპის ველებისათვის ეს აუცილებელია. ამასთან *TEXT* და *BLOB* ტიპის ველების ინდექსირება შესაძლებელია მხოლოდ *MyISAM* *ტიპის ცხრილებში.*

- *tbl_name* ცხრილის ინდექსების შესახებ ცნობების მიღება შესაძლებელია შემდეგი SQL-მოთხოვნის შესრულებით:

```
1  SHOW INDEX FROM
2    `tbl_name`
```

გასაღები ველი

UNIQUE

- UNIQUE - ეს გასაღები მიუთითებს იმაზე, რომ მოცემულ სვეტს შეუძლია მხოლოდ უნიკალური მნიშვნელობების მიღება. ცხრილის *UNIQUE გასაღების მქონე ველში* განმეორებითი მნიშვნელობის დამატების შემთხვევაში, ეს ოპერაცია დასრულდება შეცდომით.

  - უნიკალურად შეიძლება განისაზღვროს, როგორც ერთი, ასევე რამოდენიმე სვეტი:

```
1 CREATE TABLE
2   `users` (
3       `name` VARCHAR(200) NOT NULL,
4       `address` VARCHAR(255) NOT NULL,
5       UNIQUE(`name`, `address`)
6   )
```

სრულტექსტოვანი ძიება

- **FULLTEXT**-განსაზღვრავს ველებს, რომელთა მიმართაც შემდგომში შესაძლებელია გამოყენებული იქნას სრულტექსტოვანი ძიება.
  - **სრულტექსტოვანი ძიება** წარმოადგენს MySQL-ის საშუალებას, რომელიც განკუთვნილია მონაცემთა ბაზაში საჭირო ინფორმაციის მოსაძებნად, და მიღებული შედეგების გამოსატანად, მოძებნილი სტრიქონების საძიებო მოთხოვნასთან მიმართებაში, რელევანტურობის მიხედვით.
  - სრულტექსტოვანი ძიება შემოღებულია **MySQL-ის** 3.23.23 ვერსიიდან დაწყებული, *MyISAM-ის ტიპის ცხრილებისათვის და ვრცელდება მხოლოდ VARCHAR და TEXT ტიპის ველებზე.*
  - *FULLTEXT-გასაღებიანი ველების ინდექსირებისას ხდება მთლიანი მნიშვნელობის ინდექსაცია და არა მხოლოდ მისი ნაწილის,* (ანუ ინდექსაციისათვის პირველი **n**-სიმბოლოების განსაზღვრა არ შეიძლება).
  -
- **FOREIGN KEY** და **CHECK** - შემოღებულია მხოლოდ თავსებადობის უზრუნველსაყოფად სხვა SQL-ბაზებიდან კოდის გადმოტანის შემთხვევაში, მინიშნებიანი ცხრილების შემქმნელი პროგრამების გასაშვებად.
- ფაქტიურად ისინი არაფერს არ აკეთებენ.
- **table_options** - განსაზღვრავს დამატებით პარამეტრებს შესაქმნელი ცხრილისათვის.

ცხრილების შესაძლო ტიპები MySQL-ში

- BDB - გვერდების ტრანზაქციისა და ბლოკირებების მხარდამჭერი ცხრილები.
- HEAP - ამ ტიპის ცხრილების მონაცემები ინახება მხოლოდ მეხსიერებაში.
- ISAM - ცხრილების ორიგინალური დამმუშავებელი.
- InnoDB - სტრიქონის ტრანზაქციისა და ბლოკირების მხარდამჭერი ცხრილები.
- MERGE - MyISAM-ის ცხრილების ანაკრები, რომელიც გამოიყენება, როგორც ერთი ცხრილი.
- MRG_MYISAM - ფსევდონიმი MERGE-ისათვის.
- MyISAM - ახალი დამმუშავებელი, რომელიც უზრუნველყოფს ცხრილების გადატანითობას ბინარული სახით, რომელიც ცვლის ISAM

ველების პარამეტრები

- AUTO_INCREMENT- მოცემული ცხრილისათვის აყენებს შემდეგ მნიშვნელობას *AUTO_INCREMENT*.
- AVG_ROW_LENGTH -განსაზღვრავს მოცემული ცხრილისთვის სტრიქონის საშუალო სიგრძის მნიშვნელობა.მის განსაზღვრას აზრი აქვს მხოლოდ ძალიან დიდი ცხრილებისათვის, რომელთაც გააჩნიათ ცვლადი სიგრძის ჩანაწერები.
- CHECKSUM - ის უნდა განისაზღვროს 1-იანით,რათა **MySQL-ში მხარდაჭერილი იყოს საკონტროლო ჯამის შემოწმება ყველა სტრიქონისათვის (ეს ცხრილებს განახლებისას რამდენადმე ანელებს, მაგრამ სამაგიეროდ, დაზიანებული ცხრილების იოლად მოძებნის საშუალებას იძლევა**). (*MyISAM*).
- COMMENT- 60 სიმბოლოს სიგრძის კომენტარი მოცემული ცხრილისათვის.
- MAX_ROWS -იმ სტრიქონების მაქსიმალური რაოდენობა, რომელთა დამახსოვრებაც არის დაგეგმილი მოცემულ ცხრილში..
- MIN_ROWS - იმ სტრიქონების მინიმალური რაოდენობა, რომელთა დამახსოვრებაც არის დაგეგმილი მოცემულ ცხრილში.
- PACK_KEYS- უფრო მცირე ინდექსის მისაღებად საჭიროა განისაზღვროს 1-იანით. როგორც წესი ეს ანელებს ცხრილის განახლებას და აჩქარებს წაკითხვას. (*MyISAM, ISAM*).
- 0-ის დაყენება გამორთავს გასაღებების გამკვრივებას. *DEFAULT-ში დაყენებისას* (MySQL 4.0) დამმუშავებელი გაამკვრივებს მხოლოდ გრძელ სვეტებსნ *CHAR/VARCHAR*.
- PASSWORD- ახდენს ფაილის `.frm ' შიფრაციას პაროლის საშუალებით.
- DELAY_KEY_WRITE – 1-იანში დაყენება აფერხებს გასაღებების ცხრილის განახლების ოპერაციას, სანა აღნიშნული ცხრილი არ დაიხურება. (*MyISAM*).
- ROW_FORMAT- განსაზღვრავს სტრიქონების შენახვის წესებს. დღეისათვის ეს ოფცია მუშაობს მხოლოდ *MyISAM ცხრილებთან, რომლებშიც მხარდაჭერილია სტრიქონების ფორმატები DYNAMIC* и *FIXED*.

RAID_TYPE, UNION, INSERT_METHOD

- RAID_TYPE - *RAID_TYPE ოპციის გამოყენებით*, შესაძლებელია *MyISAM* მონაცემთა ფაილის დაშლა ნაკვეთებად, რათა დაძლეულ იქნას ისეთი ოპერაციული სისტემით მართვადი ფაილური სისტემის 2გბ/4გბ-იანი ლიმიტი, რომელიშიც არ არის მხარდაჭერილი დიდი ფაილები. დაყოფა არ ეხება ინდექსების ფაილს.
- გასათვალისწინებელია, რომ ფაილური სისტემებისათვის რომელთა მიერაც ხდება დიდი ფაილების მხარდაჭერა, ეს ოფცია არ არის რეკომენდებული!
- შეტანა-გამოტანის უფრო მაღალი სიჩქარეების მისაღებად შესაძლებელია RAID-დირექტორიების განთავსება სხვადასხვა ფიზიკურ დისკოებზე.
- *RAID_TYPE* იმუშავებს ნებისმიერ ოპერაციულ სისტემასთან, თუ კი **MySQL** - კონფიგურაცია შესრულებულია *with-raid-პარამეტრებით.*
- დღეისათვის *RAID_TYPE* ოფციისათვის შესაძლებელია მხოლოდ STRIPED პარამეტრი (1 და RAID0 მისთვის ფსევდონიმებს წარმოადგენენ).
- თუ მითითებულია RAID_TYPE=STRIPED *MyISAM ცხრილისათვის*, მაშინ *MyISAM* მონაცემთა ბაზის დირექტორიაში შექმნის RAID_CHUNKS ქვედირექტორიებს სახელწოდებებით `00', `01', `02'.
- თითოეულ ამ დირექტორიაში MyISAM შექმნის ფაილს `table_name.MYD'.
- მონაცემების ჩაწერისას მონაცემთა ფაილში დამმუშავებელი RAID დაამყარებს შესაბამისობას პირველი RAID_CHUNKSIZE*1024 ბაიტებისა პირველად დასახელებულ ბაიტთან, ხოლო მომდევნო RAID_CHUNKSIZE*1024 ბაიტებისა - შემდეგ ფაილებთან და ა.შ..
- UNION - ოფცია *UNION* გამოიყენება იმ შემთხვევაში, თუ საჭიროა იდენტური ცხრილების ერთობლივობის, როგორც ერთი ცხრილის გამოყენება. იგი მუშაობს მხოლოდ MERGE ცხრილებთან.
- მოცემული მომენტისათვის MERGE ცხრილთან შესადარებელი ცხრილებისათვის , საჭიროა, SELECT, UPDATE и DELETE პრივილეგიების ქონა.
- ყველა შესადარებელი ცხრილები უნდა მიეკუთვნებოდნენ იმავე მონაცემთა ბაზას, რომელსაც MERGE ცხრილი მიეკუთვნება.
- INSERT_METHOD-MERGE ცხრილში მონაცემების შესატანად, საჭიროა INSERT_METHOD- ის საშუალებით მიეთითოს, თუ რომელ ცხრილში უნდა იქნას შეტანილის მოცემული სტრიქონი.

DATA DIRECTORY, INDEX DIRECTORY

- DATA DIRECTORY=“კატალოგი" და INDEX DIRECTORY=“კატალოგი “ ოფციების გამოყენებით ცხრილების დამმუშავებელს შესაძლებელია მიეთითოს, თუ სად უნდა მოათავსოს მან თავისი ცხრილური და ინდექსური ფაილები.
- გასათვალისწინებულია, რომ მითითებული პარამეტრი directory უნდა წარმოადგენდეს სრულ გზას საჭირო კატალოგამდე (და არა ფარდობით გზას).
- მოცემული ოფციები მუშაობენ მხოლოდ MyISAM ცხრილებისათვის MySQL 4.0-ში,თუ ამ დროს არ გამოიყენება ოფცია-*skip-symlink*.

- select_statement -ამატებს შესაქმნელ ცხრილში SELECT ბრძანების მუშაობის შედეგად მიღებულ ველებს და მნიშვნელობებს.

მაგალითი

ვთქვათ მოცემულია ცხრილი ქალაქების სახელწოდებებით:

```
1 CREATE TABLE
2    `city`(
3        `name` CHAR(200) NOT NULL
4    )
```

```
1 INSERT INTO
2    `city`3VALUES
4    (‘მოსკოვი'),
5    (‘რიაზანი'),
6    (‘ლუხოვცი'),
7    (‘კოლომნა')
```

და ჩვენ გვსურს ცხრილის შექმნა, რომელშიც მოცემული იქნება მომხმარებლების სახელები და იმ ქალაქების სახელები, სადაც ისინი ცხოვრობენ:

მაგალითი (გაგრძელება)

```
01 CREATE TABLE
02   `users`(
03       `id` INT(11) NOT NULL AUTO_INCREMENT,
04       `name` CHAR(200) NOT NULL,
05       PRIMARY KEY(`id`)
06   )
07 SELECT
08   *
09 FROM10   `city`
```

- ახლა უკვე ცხრილს user გააჩნია სვეტები და შეიცავს მნიშვნელობებს:
- მოტანილი მაგალითი არავითარ შინაარსობრივ დატვირთვას არ ატარებს, ვინაიდან ველს name არავითარი მნიშვნელობები არ მიუღია.
- აქ მხოლოდ ნაჩვენებია მარცხნიდან სვეტების მიერთების პრინციპი **SELECT** კონსტრუქციის გამოყენებით..

კიდევ ერთი მაგალითი

```
1 CREATE TABLE
2   `city_new`
3 SELECT
4   `id`,
5   `city_name` AS `name`
6 FROM7   `users`
```

```
1 SELECT
2   *
3 FROM
4   `city_new`
```

```
+----+-------------+
| id |   name      |
+----+-------------+
| 1 | მოსკოვი    |
| 2 | რიაზანი    |
| 3 | ლუხოვცი |
| 4 | კოლომნა  |
+----+-------------+
4 rows in set (0.00 sec)
```

DROP TABLE

- **ოპერატორის სინტაქსი**
- DROP TABLE [IF EXISTS] tbl_name [, tbl_name,...] [RESTRICT | CASCADE]
- ეს ცხრილი შლის ცხრილს ან ცხრილებს მიმდინარე მონაცემთა ბაზიდან.
- tbl_name - წასაშლელი ცხრილის სახელი.
- IF EXISTS- თუ ეს პარამეტრი მითითებულია, მაშინ არარსებული ცხრილის წაშლის მცდელობისას შეცდომა არ წარმოიშვება.
- წინააღმდეგ შემთხვევაში წარმოიქმნება ბრძანების შესრულების შეცდომა.
- RESTRICT и CASCADE - არ გააჩნიათ არავითარი ფუნქციონალური დატვირთვა. გამოიყენებიან მხოლოდ პროგრამის გადატანის გასამარტივებლად.
- ქვემოთ მოცემული მაგალითი ახორციელებს *users ცხრილის წაშლას*.
- *1 DROP TABLE `users`*

ALTER TABLE

ბრძანების სინტაქსი

- **ALTER [IGNORE] TABLE tbl_name alter_specification [, alter_specification ...]**
- Командаბრძანება **ALTER TABLE** გამოიყენება უკვე არსებული ცხრილის შიდა სტრუქტურის შეცვლის საშუალებას.
- tbl_name- განსაზღვრავს იმ ცხრილის სახელს რომელშიც უნდა განხორციელდეს ცვლილებები.
- IGNORE- თუ ეს პარამეტრი არ არის მითითებული, მაშინ უნიკალურ გასაღებებში დუბლირებული მნიშვნელობების აღმოჩენისას, ახალ ცხრილში ხდება ყველა ცვლილების გაუქმება.
- წინააღმდეგ შემთხვევაში უნიკალურ გასაღებებში დუბლირებული მნიშვნელობების აღმოჩენისას, პირველი ჩანაწერი დუბლირებული გასაღებით დარჩება, ხოლო დანარჩენები წაიშლებიან.
- alter_specification-განსაზღვრავს უშუალოდ თვით ქმედებას, რომელიც უნდა განხორციელდეს ცხრილზე..

alter_specification

შესაძლო სინტაქსები:

- ADD [COLUMN] create_definition [FIRST | AFTER column_name]
- ADD [COLUMN] (create_definition, create_definition,...)
- ADD INDEX [index_name] (index_col_name,...)
- ADD PRIMARY KEY (index_col_name,...)
- ADD UNIQUE [index_name] (index_col_name,...)
- ADD FULLTEXT [index_name] (index_col_name,...)
- ADD [CONSTRAINT symbol] FOREIGN KEY index_name (index_col_name,...) [reference_definition]
- ALTER [COLUMN] col_name {SET DEFAULT literal | DROP DEFAULT}
- CHANGE [COLUMN] old_col_name create_definition [FIRST | AFTER column_name]
- MODIFY [COLUMN] create_definition [FIRST | AFTER column_name]
- DROP [COLUMN] col_name
- DROP PRIMARY KEY
- DROP INDEX index_name
- DISABLE KEYS
- ENABLE KEYS
- RENAME [TO] new_tbl_name
- ORDER BY col
- table_options

ახალი ველის დამატება

ADD [COLUMN] create_definition [FIRST | AFTER column_name]

- გამოიყენება ცხრილში ახალი ველის დასამატებლად.
- ამასთან შესაძლებელია ახალი ველის პოზიციის ცხადად მითითება.
  - COLUMN - არააუცილებელი პარამეტრია, რომელიც შეიძლება გამოტოვებულ იქნას.
  - create_definition-ახალი სვეტის სახელისა და თვისებების მითითება. სინტაქსი იგივეა, რაც ცხრილის შექმნისას სვეტის განსაზღვრის შემთხვევაში (**CREATE TABLE**).
  - FIRST- მიუთითებს იმაზე, რომ ცხრილში ახალი ველის დამატება საჭიროა ველების სიის დასაწყისშივე (გაჩუმებით ახალი ველი ემატება სიის ბოლოში).
  - AFTER column_name-განსაზღვრავს ცხრილში იმ ველის სახელს, რომლის შემდეგაც იქნება დამატებული ახალი ველი.

მაგალითი-(3-1)

- ვთქვათ მოცემულია ცხრილი *users* შემდეგი ველებით: *name*, *age*
- დავამატოთ ახალი ველი *country* სიის ბოლოში:

```
1 ALTER TABLE
2    `users`
3 ADD
4    `country` VARCHAR(64) NOT NULL
```

- *users ცხრილის ველები*:

```
1 SHOW COLUMNS FROM `users`;
+---------+---------------+------+-----+---------+------------+
| Field   |    Type       | Null |Key  | Default | Extra |
+---------+---------------+------+-----+---------+------------+
| name  | varchar(50) | YES |          | NULL    |          |
| age     | int(3)          | YES |          | NULL    |          |
|country| varchar(64) | NO   |          |           |          |
+---------+---------------+-------+--------+----------+-------+
 3 rows in set (0.03 sec)
```

მაგალითი (3-2)

დავამატოთ ახალი ველი *id* სიის დასაწყისში

```
1 ALTER TABLE
2   `users`
3 ADD
4   `id` INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY
5 FIRST
```

users ცხრილის ველები:

```
1 SHOW COLUMNS FROM `users`;
```

```
+----------+----------------+------+----------+-----------+--------------------+
| Field    | Type           | Null | Key      | Default   | Extra              |
+----------+----------------+------+----------+-----------+--------------------+
| id       | int(11)        | NO   | PRI      | NULL      | auto_increment |
| name     | varchar(50)    | YES |          | NULL      |                    |
| age      | int(3)         | YES |          | NULL      |                    |
| country  | varchar(64)    | NO   |          |           |                    |
+----------+----------==--+------+------------+----------+--------------------+
4 rows in set (0.00 sec)
```

მაგალითი (3-3)

- დავამატოთ ახალი ველი *city, country* ველის წინ(ანუ. *age* ველის შემდეგ):

```
1 ALTER TABLE
2    `users`
3 ADD
4    `city` VARCHAR(64)
5 AFTER
6    `age`
```

- *users* ცხრილის ველების სია:

```
1SHOW COLUMNS FROM `users`;
+-------------+----------------+--------+------+-----------+----------------------+
|    Field    |      Type      | Null   | Key | Default |         Extra        |
+-------------+----------------+--------+------+-----------+----------------------+
| id          | int(11)        | NO    | PRI  | NULL    | auto_increment |
| name        | varchar(50)   | YES |        | NULL    |                      |
| age         | int(3)         | YES |        | NULL    |                      |
| city        | varchar(64)    | YES |        | NULL    |                      |
| country     | varchar(64)    | NO    |       |           |                      |
+-------------+----------------+--------+------+---------+----------------------+
5 rows in set (0.00 sec)
```

ველების ჯგუფის დამატება

- ADD [COLUMN] (create_definition, create_definition,...) - ამატებს ცხრილში ერთ ველს ან ველების ჯგუფს.
- COLUMN-არააუცილებელი პარამეტრი, რომელიც შეიძლება გამოტოვებული იქნას.
- create_definition-ახალი სვეტის სახელისა და თვისებების დამატება განსაზღვრა.
- სინტაქსი იგივეა, რაც სვეტის განსაზღვრისას ცხრილის შექმნის დროს (**CREATE TABLE**).
  - ვთქვათ მოცემული ცხრილი *users* შემდეგი ველებით:

- 1 SHOW COLUMNS FROM `users`;
- +-------+--------------+---------+------+---------+-------------------+
- | Field | Type | Null | Key | Default | Extra |
- +-------+--------------+---------+------+---------+-------------------+
- | id | int(11) | NO | PRI | NULL | auto_increment |
- | name | varchar(50) | YES | | NULL | |
- | age | int(3) | YES | | NULL | |
- +-------+--------------+---------+------+---------+-------------------+
- 3 rows in set (0.00 sec)

მაგალითი (1)

- ცხრილში users დავამატოთ ახალი ველები *city* და *country*:

```
1 ALTER TABLE
2   `users`
3 ADD
4   (
5      `city` VARCHAR(64) NOT NULL,
6      `country` VARCHAR(64) NOT NULL
7   )
```

- **users ცხრილის ველების სია:**

```
1 SHOW COLUMNS FROM `users`;

+---------+----------------+--------+--------+---------+-------------------------------+
| Field   | Type           | Null   | Key    | Default | Extra                         |
+---------+----------------+--------+--------+---------+-------------------------------+
| id      | int(11)        | NO     | PRI    | NULL    | auto_increment                |
| name    | varchar(50)    | YES    |        | NULL    |                               |
| age     | int(3)         | YES    |        | NULL    |                               |
| city    | varchar(64)    | NO     |        |         |                               |
| country | varchar(64)    | NO     |        |         |                               |
+---------+----------------+--------+--------+---------+-------------------------------+
5 rows in set (0.00 sec)
```

ALTER TABLE

- **ALTER TABLE table_name_old RENAME table_name_new**
  - *table_name_old* - ცხრილის ძველი სახელი, რომლის შეცვლაც გვჭირდე;
  - *table_name_new* - ცხრილის ახალი სახელი.

- ვთქვათ ჩვენ გვჭირდება ცრილს *search* შევუცვალოთ სახელი და დავარქვათ მას *search_en:*

- 1 $sql="ALTER TABLE search RENAME search_en";
- 2 mysql_query($sql);

INSERT

INSERT ახორციელებს ახალი სტრიქონების ჩასმას ცხრილში

ბრძანების სინტაქსი

- **INSERT [LOW_PRIORITY | DELAYED] [IGNORE] [INTO] tbl_name [(col_name,...)] VALUES (expression,...),(...),...**
- ან
- **INSERT [LOW_PRIORITY | DELAYED] [IGNORE] [INTO] tbl_name SET col_name=expression, col_name=expression, ...**
- ან
- **INSERT [LOW_PRIORITY | DELAYED] [IGNORE] [INTO] tbl_name [(col_name,...)] SELECT ...**

INSERT

ბრძანების მუშაობის ზოგადი დებულებები

- tbl_name - განსაზღვრავს იმ ცხრილის სახელს, რომელშიც მოხდება ახალი სტრიქონის ჩამატება. ბრძანება **INSERT**- ის გაშვების მომენტისათვის მონაცემთა ბაზაში უნდა არსებობდეს ცხრილი ამ სახელწოდებით
- LOW_PRIORITY - თუ მითითებულია ეს პარამეტრი , მაშინ ახალი ჩანაწერის ჩასმა დაყოვნდება მანამ სანამ სხვა სცენარები არ დაასრულებენ წაკითხვას ამ ცხრილიდან. უნდა აღინიშნოს, რომ თუ ხშირად ხდება ცხრილის გამოყენება, მაშინ ამ პარამეტრის მითითებისას შესაძლებელია დიდი დროის გასვლა სანამ მოცემული ბრძანება შესრულდება.
- DELAYED- თუ მითითებულია ეს პარამეტრი, მაშინ **INSERT** ბრძანების შესრულების შემდეგ სცენარი მაშინათვე მიიღებს პასუხს მონაცემთა ბაზიდან ახალი ჩანაწერის წარმატებით შესრულების შესახებ, ხოლო ჩანაწერი ჩასმული იქნება მხოლოდ მას შემდეგ რაც დასრულდება მოცემული ცხრილის გამოყენება სხვა სცენარის მიერ. ეს ხელსაყრელია მაშინ, როდესაც საჭიროა სკრიპტის მუშაობის დიდი სიჩქარე.
- მოცემული პარამეტრი მუშაობს მხოლოდ *ISAM* და *MyISAM ტიპის ცხრილებთან* .
- უნდა აღინიშნოს, რომ, თუ ცხრილი, რომელშიც ხდება ჩანაწერის ჩასმა, მოცემულ მომენტში არ გამოიყენება სხვა მოთხოვნების მიერ, მაშინ ბრძანება *INSERT DELAYED* იმუშავებს უფრო ნელა, ვიდრე **INSERT**. ასე, რომ *DELAYED* პარამეტრის გამოყენება რეკომენდებულია მხოლოდ ცხრილზე დიდი დატვირთვის შემთხვევაში.
- IGNORE -თუ ცხრილის ზოგიერთ ველებს გააჩნიათ PRIMARY და UNIQUE ტიპის გასაღებები , და ხდება ახალი ჩანაწერის ჩამატება, რომელშიც ამ ველებს დუბლირებული მნიშვნელობები გააჩნიათ, მაშინ ბრძანების მოქმედება ავარიულად დასრულდება და გამოვა შეტყობინება №1062 (*"Duplicate entry 'val' for key N"*) შეცდომის შესახებ.
- თუ INSERT ბრძანებაში მითითებულია გასაღები სიტყვა IGNORE, მაშინ ჩანაწერების ჩამატება არ წყდება, ხოლო სტრიქონები დუბლირებული მნიშვნელობებით უბრალოდ არ ჩაისმებიან.
- თუ კი **MySQL**- ი დაკონფიგურირებული იყო DONT_USE_DEFAULT_FIELDS ოფციის გამოყენებით, მაშინ ბრძანება INSERT მოახდენს შეცდომის გენერირებას იმ შემთხვევაში, თუ ცხადად არ იქნება მითითებული ყველა იმ სვეტების სიდიდეები, რომლებიც მოითხოვენ მნიშვნელობებს NOTNULL.
- იმისათვის, რომ გავიგოთ AUTO_INCREMENT გასაღებიანი ველისათვის მინიჭებული მნიშვნელობა, შეგვიძლია გამოვიყენოთფუნქცია mysql_insert_id().
- **INSERT** ბრძანების საშუალებით, არსებულ ცხრილში ახალი ჩანაწერების ჩასამატებლად, სამი ძირითადი სინტაქსი არსებობს:
- INSERT ... VALUES - ამ შემთხვევაში ბრძანებაში აშკარად მიეთითება შესატანი ველებისა და მათი მნიშვნელობების თანმიმდევრობის.

ცხრილში ჩანაწერის ჩასმის მაგალითები

- შემდეგი ბრძანება *users* ცხრილში დაამატებს ახალ ჩანაწერს, *name*, *age*, *country*, *city* ველებისათვის შესაბამისად *Evgen*, *26*, *Russia*, *Ryazan* მნიშვნელობების მინიჭებით:

```
1 INSERT INTO
2    `users` (`name`, `age`, `country`, `city`)
3 VALUES
4    ('Evgen', 26, 'Russia', 'Ryazan')
```

- თუ ჩხრილში არსებული ველის ან ველთა ს ჯგუფისათვის არ იქნება განსაზღვრული მნიშვნელობები, მაშინ გამოყენებული იქნება ცხრილის შექმნისას გაჩუმებით განსაზღვრული მნიშვნელობა:

```
1INSERT INTO
2    `users` (`name`, `age`, `city`)
3VALUES
4    ('Evgen', 26, 'Ryazan')
```

  - ამ ბრძანების შესრულების შემდეგ ველი *country მიიღებს გაჩუმებით მნიშვნელობას.*
  - თუ **INSERT** ბრძანების შესრულებისას არ იყო მითითებული ველების დასახელებები, მაშინ, *VALUES()-ში მითითებული უნდა იყოს მნიშვნელობები ცხრილის ყველა ველებისათვის.*

- თუ კი ცხრილის ველების სია წინასწარ არ არის ცნობილი, მაშინ მისი გაგება შესაძლებელია შემდეგი ბრძანების მეშვეობით:

```
1 DESCRIBE `users`
```

- ამ ბრძანების შედეგი იქნება დაახლოებით შემდეგი შინაარსის ცხრილი:

```
+----------+-----------------+----------+---------------+---------+---------+
| Field    | Type            | Null     | Key           | Default | Extra   |
+----------+-----------------+----------+---------------+---------+---------+
| name     | varchar(50)     | YES      |               | NULL    |         |
| age      | int(3)          | YES      |               | NULL    |         |
| country  | varchar(64)     | NO       |               |         |         |
| city     | varchar(64)     | NO       |               |         |         |
 +---------+-----------------+----------+---------------+---------+---------+
4 rows in set (0.01 sec)
```

INSERT ... SET

ამ შემთხვევაში, ბრძანებაში, ცხრილში არსებულ ყოველ ველს, ენიჭება ”ველის სახელი =”მნიშვნელობას” სახის მნიშვნელობა.

- შემდეგი მაგალითი შედეგის მიხედვით იდენტურია პირველი მაგალითისა **INSERT ... VALUE** - სათვისს.:

```
1 INSERT INTO
2   `users`
3 SET
4   `name` = 'Evgen',
5   `age` = 26,
6   `country` = 'Russia',
7   `city` = 'Ryazan‘
```

- ისევე, როგორც INSERT ... VALUES -ს შემთხვევაში, თუ ერთ ან რამოდენიმე ველს არ განესაზღვრება მნიშვნელობას მაშინ, ძალაში იქნება გაჩუმებითი მნიშვნელობა.
- მნიშვნელობების სახით ველებს შეიძლება განესაზღვროთ არა მხოლოდ მნიშვნელობები, არამედ გამოსახულებებიც.
- გამოსახულებებში დაშვებულია ცხრილის იმ მნიშვნელობების გამოყენება, რომლებიც უკვე იყვნენ გამოყენებული ამ ბრძანებაში:

```
1 INSERT INTO
2   `tbl_name`
3SET
4   `field1` = 4,
5   `field2` = `field1`*`field1`
```

ან

```
1 INSERT INTO
2   `tbl_name` (`field1`, `field2`)
3VALUES
4   (4, `field1`*`field1`)
```

INSERT ... SELECT

- ასეთი სინტაქსი ცხრილში ერთი მოქმედებით, დიდი რაოდენობის ჩაწერების დამატების საშუალებას იძლევა, თანაც სხვადასხვა ცხრილებიდან
- შემდეგ მაგალითში ნაჩვენებია *users_new* ცხრილში, *users* ცხრილის ყველა იმ ჩანაწერების ჩაწერა, რომლებშიც ველი, *country-ს* მნიშვნელობა ტოლია "*Russia*"-სი.

```
1 INSERT INTO
2    `users_new`
3 SELECT
4    *
5 FROM
6    `users`
7 WHERE
8    `country` = 'Russia'
```

- თუ ცხრილისათვის, რომელშიც ხდება ჩანაწერების ჩამატება, არ არის მითითებული ველების სია, მაშინ ყველა ველისათვის მნიშვნელობები განისაზღვრებიან *SELECT ბრძანების მუშაობის შედეგების საფუძველზე.*
- თუ განსაზღვრული არ არის მხოლოდ რამოდენიმე ველი, მაშინ მათვის მიღებული იქნება გაჩუმებითი მნიშვნელობები.

INSERT ... SELECT
სინტაქსია თავისებურება

- იმ ცხრილის სახელი, რომელშიც ხდება ჩანაწერის ჩამატება, არ უნდა იყოს მითითებული *SELECT-* ის ნაწილის, *FROM-* ის ცხრილების სიაში, ვინაიდან ამან შეიძლება გამოიწვიოს ჩასმის შეცდომა *(*ხომ *SELECT-*ის ნაწილის, *WHERE* პირობას, თვითონ შეუძლია მოძებნოს ჩანაწერები, რომლებიც ამ ბრძანებით უკვე ადრე იყვნენ ჩამატებული*)*.

- სხვა სცენარებს ეკრძალებათ მოთხოვნაში მონაწილე ცხრილებში ჩანაწერების ჩამატება, მათი შესრულების დროს.

- *AUTO_INCREMENT-* სვეტები მუშაობენ ჩვეულებრივად.

Синтаксис команды UPDATE

- ჩანაწერის განახლება ხორციელდება ბრძანებით UPDATE.
- ბრძანების სინტაქსი
- UPDATE [LOW_PRIORITY] [IGNORE] tbl_name SET col_name1=expr1 [, col_name2=expr2, ...] [WHERE where_definition] [LIMIT #]

- tbl_name - განსაზღვრავს იმ ცხრილის სახელს, რომელშიც უნდა განხორციელდეს ჩანაწერების განახლება.
- На момент запуска команды UPDATE ბრძანების გაშვების მომენტისათვის ცხრილი ასეთი სახელით უნდა არსებობდეს მონაცემთა ბაზაში.

- LOW_PRIORITY- თუ მითითებულია ეს პარამეტრი, მაშინ ჩანაწერის განახლება გადაიწევს იმ დროიმდე, სხვა სცენარები არ დაასრულებენ წაკითხვას ამ ცხრილიდან.

- IGNORE - თუ კი ცხრილის ზოგიერთ ველებს გააჩნიათ გასაღებები *PRIMARY* და *UNIQUE*, და ამ დროს ხდება იმ სტრიქონის განახლება, სადაც ამ ველებს გააჩნიათ ერთმანეთის მადუბლირებელი მნიშვნელობები, მაშინ ბრძანების მოქმედება სრულდება ავარიულად და გამოიტანება შეტყობინება №1062 (*"Duplicate entry 'val' for key N"*).

- თუ ბრძანება *INSERT-ში მითითებულია* გასაღები სიტყვა *IGNORE*, მაშინ ჩანაწერების განახლება არ წყდება, ხოლო დუბლირებული მნიშვნელობების შემცველი სტრიქონები უბრალოდ არ შეიცვლებიან.
- SET - ამ გასაღები სიტყვის შემდეგ მითითებული უნდა იყოს ცხრილის იმ ველების სია, რომელთა განახლებაც უნდა მოხდეს და უშუალოდ ველების თვით მნიშვნელობები შემდეგი სახით:

- **ველის სახელი='მნიშვნელობა'**

მაგალითები (2-1)

- შემდეგი მაგალითი ახორციელებს *country* ველის განახლებას *users* ცხრილის ყველა ჩანაწერებში:

- **1 UPDATE**
- **2 `users`**
- **3 SET**
- **4 `country`='რუსეთი'**

- **აქ კი ხდება *country* და *city* ველების განახლება *users* ცხრილის ყველა ჩანაწერებში:**

- **1 UPDATE**
- **2 `users`**
- **3 SET**
- **4 `country`='Russia',**
- **5 `city`='Ryazan'**

- თუ კი ახალი მნიშვნელობა, რომელსაც ანიჭებს **UPDATE** ბრძანება, შეესაბამება ძველ მნიშვნელობას, მაშინ ამ ველის განახლება არ ხდება.

- ახალი მნიშვნელობის განსასაზღვრავად შესაძლებელია გამოსახულებების გამოყენება.

მაგალითები (2-2)

შემდეგი მაგალითი გაზრდის ყველა *users* ცხრილში დაფიქსირებული ყველა მომხმარებლის ასაკს ერთი წლით:

```
1 UPDATE
2   `users`
3 SET
4   `age`=`age`+1
```

- WHERE- განსაზღვრავს ცვლილებას დაქვემდებარებული ჩანაწერების შერჩევის პირობას.

- შემდეგი მაგალითი შეცვლის მომხმარებლების შესახებ ჩანაწერებში ქალაქის სახელწოდებას "*Ryazan*“-ს "*Рязань*“-თ:

```
1 UPDATE
2   `users`
3 SET
4   `city`='Рязань‘
5 WHERE6   `city`='Ryazan‘
```

- **LIMIT- განსაზღვრავს ცვლილებებს დაქვემდებარებული ჩანაწერების მაქსიმალურ რაოდენობას.**

```
1 UPDATE
2   `users`
3 SET
4   `age`=`age`+1
5 LIMIT
6   5
```

SELECT

- **ჩანაწერების მოძებნა ხორციელდება ბრძანებით SELECT**
- **ბრძანების სინტაქსი**
- **SELECT * FROM table_name WHERE (გამოსახულება) [order by field_name [desc][asc]]**
- ეს ბრძანება ეძებს ყველა ჩანაწერებს ცხრილში *table_name*, რომლებიც აკმაყოფილებენ გამოსახულებას "გამოსახულება".
- თუ ჩანაწერების რაოდენობა რამდენიმეს შეადგენს, მაშინ *order by* წინადადების მითითებისას, მათი დახარისხება მოხდება იმ ველის მიხედვით, რომლის სახელიც იქნება მითითებული ამ გასაღები სიტყვის მარჯვენა მხარეს (თუკი მოცემული იქნება სიტყვა *desc*, მაშინ დახარისხება მოხდება საპირისპირო მიმართულებით).
- *order by წინადადებაში შესაძლებელია რამოდენიმე ველის მითითებაც.*
- განსაკუთრებული მნიშვნელობა გააჩნია სიმბოლოს *.
- ის მიგვანიშნებს იმაზე,რომ შერჩეული ჩანაწერებიდან ამოღებულ უნდა იქნას ყველა ჩანაწერი, როგორც კი შესრულებული იქნება ანაკრების მიღების ბრძანება.
- მეორეს მხრივ, ვარსკვლავის ნაცვლად შესაძლებელია უშუალოდ ყველა იმ ველების ჩამოთვლა, რომელთა ამოღებაც არის საჭირო და მათი ერთმანეთისაგან მძიმეებით გამოყოფა.
- მაგრამ ყველაზე ხშირად მაინც იყენებენ სწორედ *.

SELECT-სრული სინტაქსი

- SELECT [STRAIGHT_JOIN]
- [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT] [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS] [HIGH_PRIORITY]
- [DISTINCT | DISTINCTROW | ALL]
- expression,...
- [INTO {OUTFILE | DUMPFILE} 'file_name' export_options]
- [FROM table_references
- [WHERE where_definition]
- [GROUP BY {unsigned_integer | col_name | formula} [ASC | DESC], ...]
- [HAVING where_definition]
- [ORDER BY {unsigned_integer | col_name | formula} [ASC | DESC], ...]
- [LIMIT [offset,] rows]
- [PROCEDURE procedure_name]
- [FOR UPDATE | LOCK IN SHARE MODE]]

ANSI SQL

- ყველა პარამეტრები, რომლებიც იწყებიან SQL_, STRAIGHT_JOIN და HIGH_PRIORITY-ით, წარმოადგენენ
- MySQL-ის გაფართოებას ANSI SQL -ისათვის.

- ოფციები DISTINCT, DISTINCTROW და ALL მიუთითებენ, იქნებიან თუ არა დაბრუნებული დუბლირებული ჩანაწერები.
- გაჩუმებით დაყენებულია პარამეტრი (ALL), რაც ნიშნავს იმას, რომ ხდება ყველა ჩანაწერის დაბრუნება.

- DISTINCT და DISTINCTROW წარმოადგენენ სინონიმებს და მიანიშნებენ იმაზე, რომ, მონაცემთა ჯამურ ანაკრებში დუბლირებული ჩანაწერები ამოღებული უნდა იქნან.

- გამოსახულება expression განსაზღვრავს სვეტებს, რომლებშიც საჭიროა შერჩევის განხორციელება.

- ბრძანება INTO OUTFILE 'file_name‘ ახორციელებს არჩეული სტრიქონების ჩაწერას ფაილში, რომელიც მითითებულია file_name-ში.
- მოცემული საიტი იქმნება სერვერზე და მოცემულ მომენტამდე ის არ უნდა არსებულიყო.

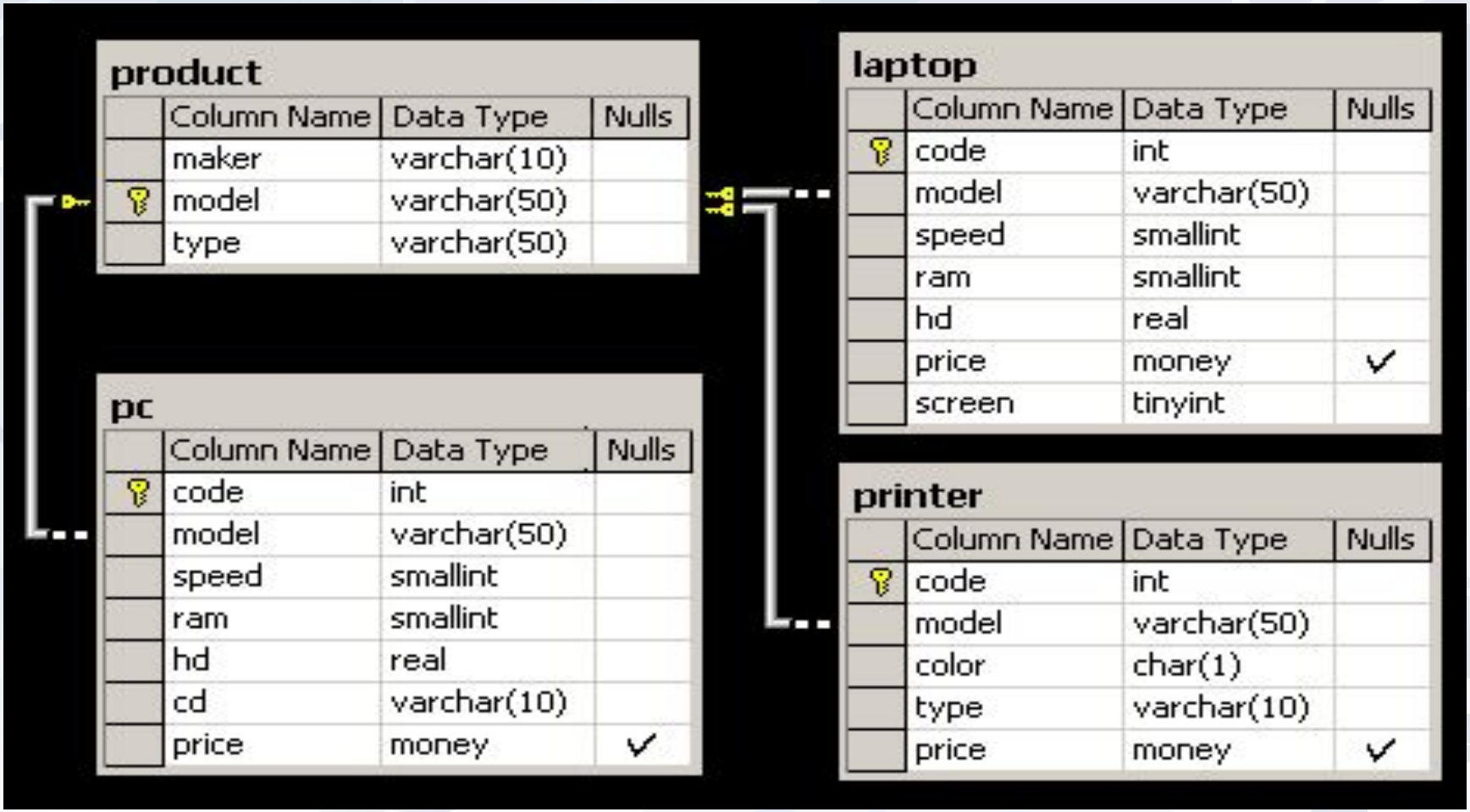
- SELECT ბრძანების ასეთი ფორმის გამოსაყენებლად საჭიროა FILE-პრივილეგიები.

- თუ INTO OUTFILE-ის ნაცვლად გამოვიყენებთ INTO DUMPFILE-ს ,მაშინ MySQL-ი ფაილში ჩაწერს მხოლოდ ერთ სტრიქონს, სვეტებისა და სტრიქონების დასრულების სიმბოლოების გარეშე, და რაიმე ეკრანირების გარეშე.

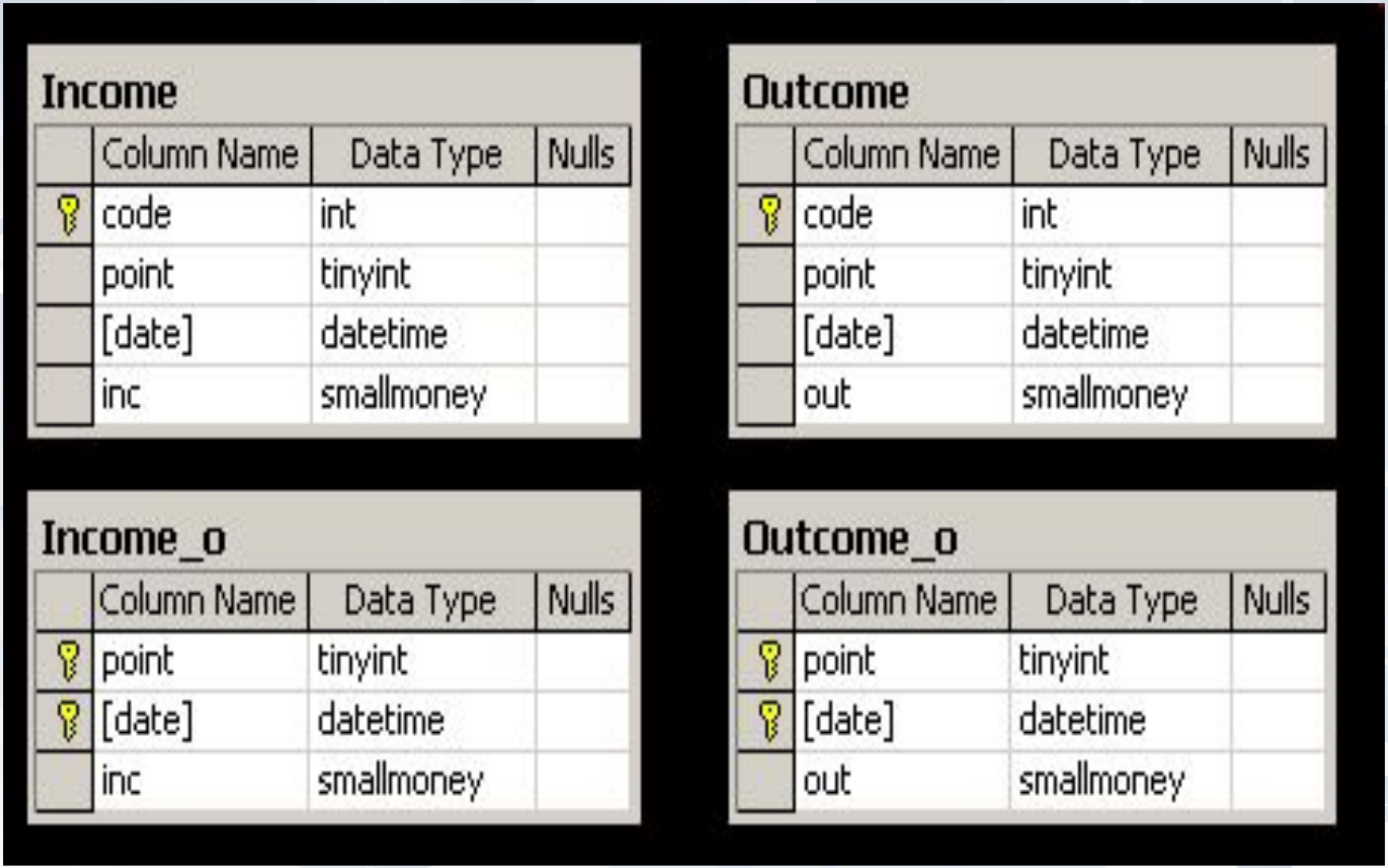
- გასათვალისწინებელია, რომ INTO OUTFILE და INTO DUMPFILE- ის საშუალებით შექმნილი ნებისმიერი ფაილი, ხელმისაწვდომი იქნება ყველა მომხმარებლისათვის.
- გამოსახულება FROM table_references განსაზღვრავს ცხრილს, საიდანაც უნდა მოხდეს სტრიქონების ამოღება.

- ოფცია WHERE განსაზღვრავს პირობას მონაცემების შერჩევისათვის.

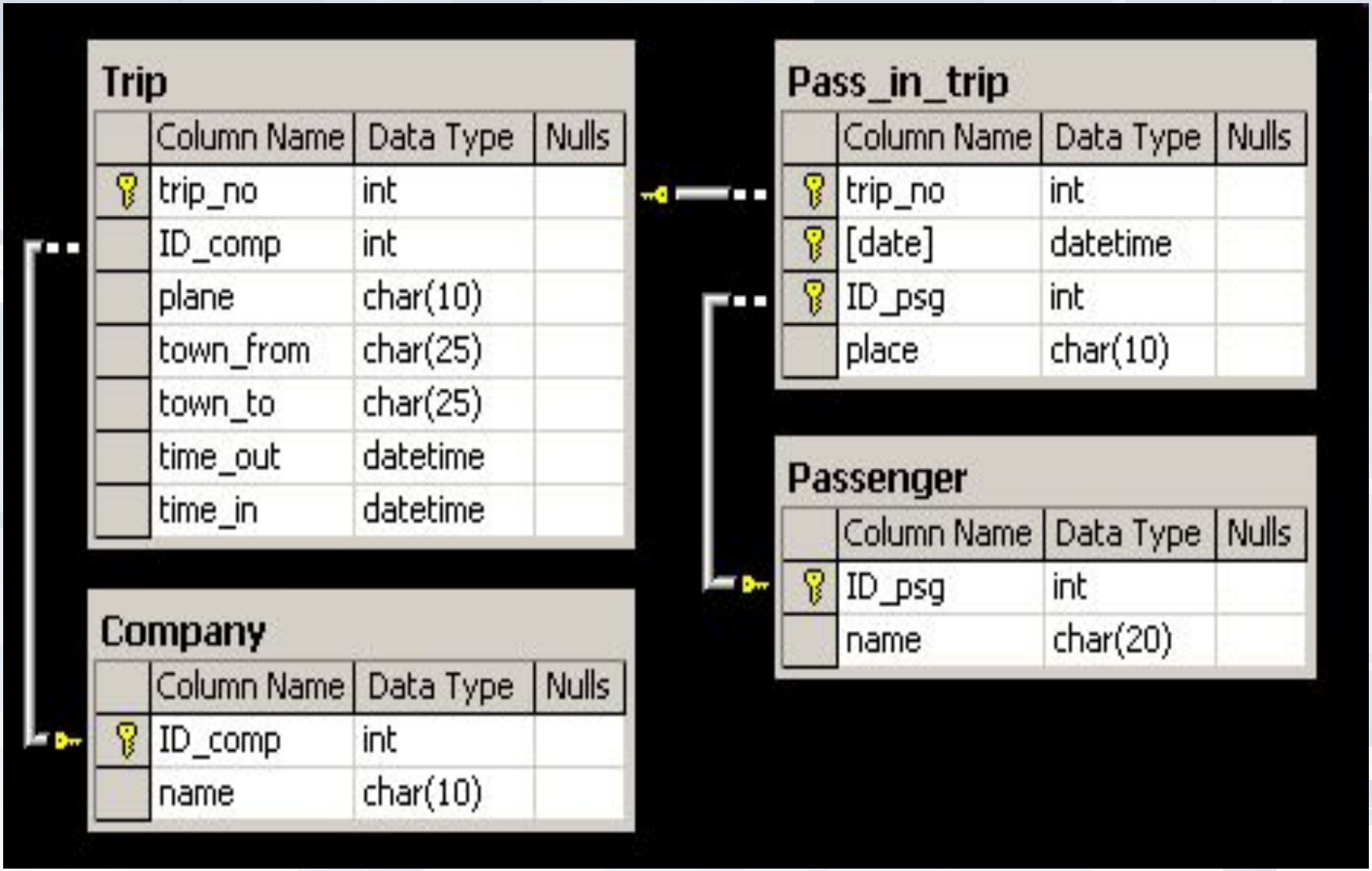
კომპიუტერული ფირმა



მეორადი რესურსების ფირმა



აეროფლოტი


Trip
Column Name
Data Type
Nulls
trip_no
int
ID_comp
int
plane
char(10)
town_from
char(25)
town_to
char(25)
time_out
datetime
time_in
datetime
Pass_in_trip
Column Name
Data Type
Nulls
trip_no
int
[date]
datetime
ID_psg
int
place
char(10)
Passenger
Column Name
Data Type
Nulls
ID_psg
int
name
char(20)
Company
Column Name
Data Type
Nulls
ID_comp
int
name
char(10)