



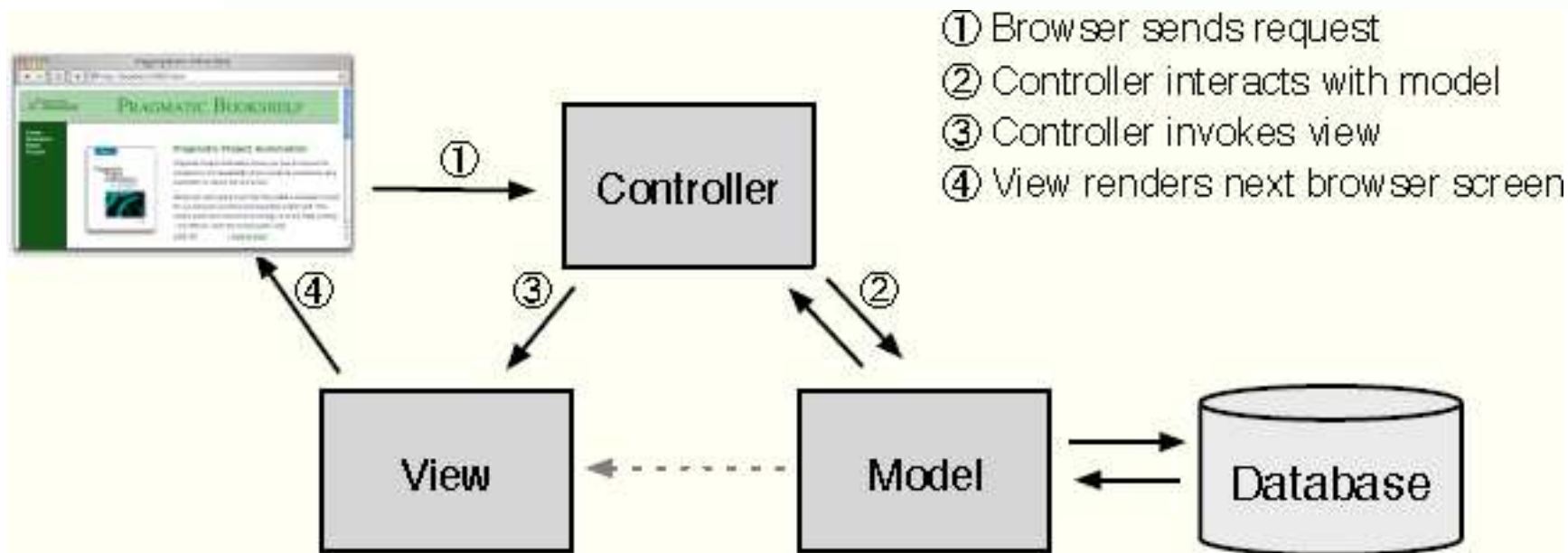
Adatbáziskapcsolat és modellek

Kovács Gábor

`kovacsg@tmit.bme.hu`

BME-TMIT

MVC



Adatbáziskezelés Rails-ben

⑥ Adatdefiníció – migráció

- △ Az adatbázis séma létrehozása, módosítása
- △ Verziókezelés

⑥ Adatmanipuláció – ActiveRecord

- △ ORM (Object Relational Mapping)
- △ CRUD – létrehoz, lekérdez, módosít, töröl
- △ Analitikus függvények
- △ Táblák közötti relációk és osztályok közötti kapcsolatok

Migráció 1

- ⑥ Migrációk végrehajtása `rails` (vagy `rake`) célokkal
- ⑥ Adatbázis létrehozása: `rails db:create`
- ⑥ Adatbázis létrehozása és inicializálása:
`rails db:setup`
- ⑥ Adatbázis törlése: `rails db:drop`
- ⑥ Adatbázis újrainicializálása: `rails db:reset`
- ⑥ Kapcsolódás létező adatbázishoz:
 - △ Ruby: `rails db:schema:dump, rails db:schema:load`
 - △ SQL: `rails db:structure:dump, rails db:structure:load`

Migráció 2

- ⑥ Agilis fejlesztés: az adatbázis séma is folyamatosan változik, fejlődik
- ⑥ Verziókezelés: új migráció minden egyes modellhez (`rails generate model`) és migrációhoz (`rails generate migration`)
- ⑥ A migráció végrehajtása `rails db:migrate`
- ⑥ Átállás egy másik verzióra, `VERSION` és `STEP` paraméterek

```
rails db:migrate VERSION=20120229213548
rails db:rollback STEP=1
rails db:redo STEP=1
```

Migráció 3

6 A User modell lehetséges migrációja

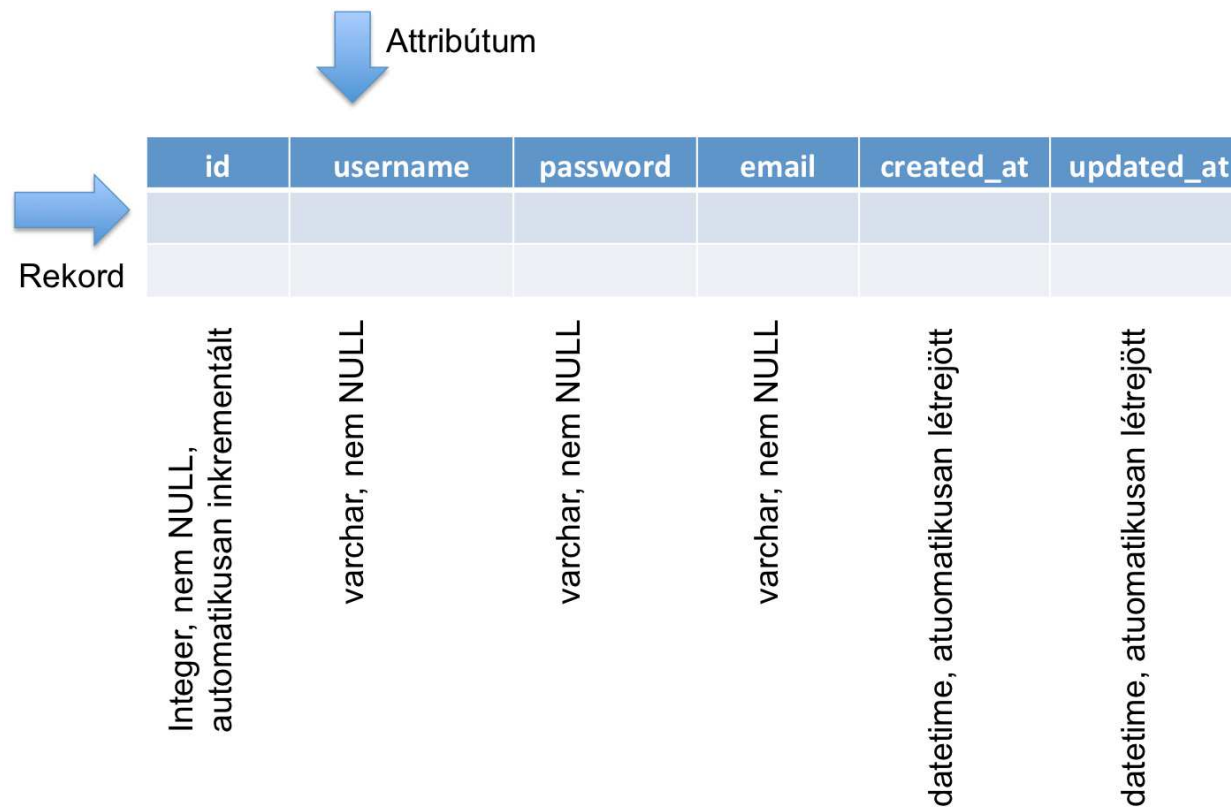
```
rails generate model User username:string password:string email:string
```

6 A migráció neve után felsorolhatók az attribútumok és típusuk `attributum:típus` formában

```
def change
  create_table :users do |t|
    t.string :username
    t.string :password
    t.string :email
    t.timestamps
  end
end
```

Migráció 4

6 A User modell lehetséges migrációjának eredménye



Migráció 5

⑥ A Rails által támogatott típusok: `:binary`, `:boolean`, `:date`, `:datetime`, `:decimal`, `:float`, `:integer`, `:references`, `:string`, `:text`, `:time`, `:timestamp` és `:primary_key`

⑥ Opciók

- △ `:null`, lehet `true` vagy `false`
- △ `:default`, default érték
- △ `:limit`, az adattípus hossza
- △ `:decimal` esetén `:precision` és `:scale`
- △ `:index`, lehet `true` vagy `false`
- △ `:polymorphic`, később

6 Migráció típusok – tábla módosítása

- △ Új attribútum hozzáadása: `add_column` paraméterek sorrendben táblanév, az új attribútum neve, típusa, például `add_column :users, :salt, :string`
- △ Attribútum eltávolítása: `remove_column` paraméterek sorrendben táblanév, az eltávolítandó attribútum neve, például `remove_column :users, :salt`
- △ Attribútum átnevezése: `rename_column` paraméterek sorrendben táblanév, az régi attribútumnév, az új attribútumnév, például
`rename_column :users, :password, :encrypted_password`
- △ Attribútum típusának módosítása: `change_column` paraméterek sorrendben táblanév, attribútumnév, új típus, opciók, például
`change_column :users, :username, :string, :limit=>15`

Migráció 7

6 Migráció típusok – tábla módosítása (folyt.)

△ Tábla létrehozása: lásd az `User` modell

△ Kulcs

```
create_table :users, :primary_key => :username
```

△ Tábla törlése: `drop_table`, a paramétere a tábla neve, például

```
drop_table :users
```

△ Tábla átnevezése: `rename_table`, a két paraméter a régi táblanév és az új táblanév ebben a sorrendben, például `rename_table :users :new_users`

△ Kapcsolótábla: `create_join_table :tasks, :users`

6 Indexek

△ Nagy számú és gyakran hozzáfért adat esetén hasznos az indexek bevezetése:

```
add_index és remove_index, paramétereik a tábla neve és az index neve
```

Migráció 8

⌚ Natív SQL

- △ `execute` metódus string paraméterrel

⌚ Időpecsétek

- △ `t.timestamps` metódus automatikusan létrehozza a `created_at` `updated_at` attribútumokat

⌚ Elsődleges kulcs

- △ Egy egész típusú `id` attribútum automatikusan létrejön
- △ A `:primary_key` opcióval felüldefiniálható a neve
- △ A `:id => false` opcióval letiltható a generálása (pl. kapcsolótábla esetén)

⌚ Idegen kulcsok

- △ A `t.references :user` metódus létrehoz egy `user_id` attribútumot, ami a `user` tábla kulcsa
- △ Ez megvalósítható a modell osztályokon keresztül is
- △ Az idegen kulcs kényszerek hozzáadása nem szükséges (`add_foreign_key`, `remove_foreign_key`)

Migráció 9

- ⑥ A migráció alapvetően egyirányú (`change` metódus), de mindkét irány definíciója lehetséges az `up` és `down` metódusokkal a `change` helyett
- ⑥ A Rails automatikusan megvalósítja mindkét irányt a `change` metódussal a következő migrációk esetén: attribútum, index és időpecsét hozzáadása, illetve törlése, tábla létrehozása, tábla átnevezése
- ⑥ Rails 4-től a két függvény a `reversible` függvény blokk paraméterének `up` és `down` függvényével helyettesíthető
- ⑥ Az `AddValamiToTablanev` és `RemoveValamiFromTablanev` konvenciót követő migráció automatikusan létrehozza a migrációban az `add_column` `remove_column` hívásokat a paraméterként `attributum:tipus` formában átadott attribútumokra

Migráció 10

```
class MigrationName < ActiveRecord::Migration
  def up
    # ...
  end
  def down
    # ...
  end
  def change
    reversible do |dir|
      dir.up {...}
      dir.down {...}
    end
  end
end
```

Migráció 11

6 Hozzáadott attribútumok visszamenőleges inicializációja:

```
Users.all.each { |a| a.update_attributes! :password='changeme' }  
User.update_all ["password = ?", 'changeme']
```

Adatbáziskezelés Rails-ben

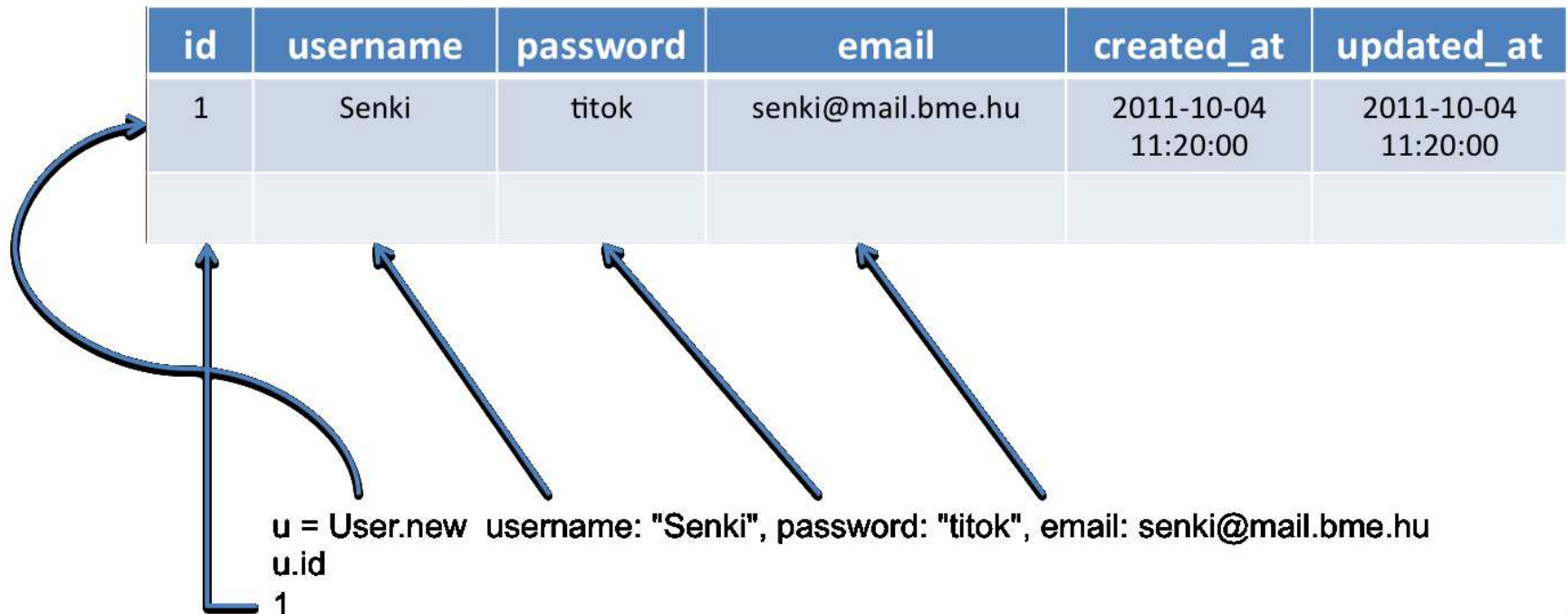
- ⑥ Adatdefiníció – migráció
 - △ Az adatbázis séma létrehozása, módosítása
 - △ Verziókezelés
- ⑥ **Adatmanipuláció – ActiveRecord**
 - △ ORM (Object Relational Mapping)
 - △ CRUD – létrehoz, lekérdez, módosít, töröl
 - △ Analitikus függvények
 - △ Táblák közötti relációk és osztályok közötti kapcsolatok

ActiveRecord – ORM 1

- ⑥ Az ActiveRecord az ORM réteg a Railsben
 - △ Tábla – osztály
 - △ Rekord – objektum
 - △ Tábla attribútum – objektum attribútum
- ⑥ Egy ActiveRecord::Base leszármazott osztály definíciója egy tábla létrehozásának felel meg.
- ⑥ Railsben konvenció szerint a tábla neve a modell osztály nevének többesszámú változata.
- ⑥ Nincsenek: triggerek, tárolt eljárások, kényszerek

ActiveRecord – ORM 2

- 6 Az attribútumok setterei és getterei automatikusan elérhetőek, a boolean attribútumok gettere ?-re végződik
- 6 Az attribútum típusa automatikusan konvertálódik Ruby típusává



ActiveRecord – ORM 3

6 Elsődleges kulcs

- △ A Rails minden tábla létrehozásakor automatikusan létrehoz egy `id` attribútumot
- △ A Rails garantálja, hogy a `id` egyedi minden rekordra
- △ Felüldefiniálható a `primary_key=` setterrel, vagy a migráció során a `create_table :id=>false` paraméterével.

ActiveRecord – ORM 4

6 Adatbázis-kapcsolat

△ A database.yml alapján

```
ActiveRecord::Base.establish_connection(  
  :adapter => "mysql2" ,  
  :host => "localhost" ,  
  :database => "gyakorlat_development" ,  
  :username => "root" ,  
  :password => ""  
)
```

ActiveRecord – CRUD 1

- ⑥ Új rekord létrehozása = Új példány mentése (`save`)

```
u= User.new
u.username= 'Valaki'
u.email= 'valaki@mail.bme.hu'
u.password = 'titok'
u.save
```

- ⑥ Az ActiveRecord konstruktor elfogad blokk vagy hash paramétert is:
 - △ `User.new do |u| ... end`
 - △ `User.new(:username => 'Senki')`
- ⑥ Konstruktor és mentés egyszerre: `create` metódus

ActiveRecord – CRUD 2

6 Rekord(ok) keresése, olvasása

- △ Legegyszerűbb mód: hozzáférés az elsődleges kulccsal
`User.find(1)`. Vagy visszatér egy `User` objektummal, vagy `RecordNotFound` kivételt generál
- △ A `find` és a `take` rekordok/objektumok halmazát/tömbjét adja vissza, ami egyedi kulcs alapú keresés esetén singleton, az utóbbi nem végez implicit rendezést
- △ Az összes rekord lekérdezése: `users=User.find :all` vagy `User.all`
- △ Singleton halmaz lekérése: `User.take`, `User.first` vagy `User.last`, egész paraméterrel megadható a visszaadandó rekordok száma
- △ A visszaadandó attribútumok egy részhalmazának kérése:

```
students = User.select("username,email")
```

ActiveRecord – CRUD 3

- 6 Az összes rekord egy részhalmazának lekérdezése (ez az SQL WHERE-nek felel meg):

```
u = User.find_by username: 'Valaki'  
u = User.where(username: 'Valaki').take
```

- 6 Rekordok paraméterezett keresése (a paraméterek például egy HTTP POST-ból jönnek)

```
n=params[:username]  
u = User.where("username='#{n}'").take  
u = User.where("username=?", n).take  
u = User.where("username=:un", { un: n }).take  
u = User.where("username=:username", params[:user]).take
```

- 6 Hasonló rekordok keresése, SQL LIKE

```
u = User.where("username like ?", "Se"+"%").take
```

ActiveRecord – CRUD 4

6 Kiegészítések a kereséshez

- △ Rendezett tömb visszatérési értéként, az SQL ORDER BY mintájára:

```
users = User.order(username: :desc)
users = User.order("username DESC")
```

- △ A visszaadott elemek számának korlátozása, az SQL LIMIT mintájára:

```
users = User.limit(10)
```

- △ Az első visszaadott rekord sorszáma a tömbben:

```
users = User.limit(page_size).offset(page*page_size)
```

- △ A visszaadott elemek csoportosítása, az SQL GROUP BY mintájára:

```
users=User.order("username DESC").group("email")
```

- △ Nem írható vissza: `User.readonly`

ActiveRecord – CRUD 5

6 Egyéb keresési módok

- △ Keresés és nem létező rekord létrehozása: `find_or_create_by`
- △ Direct SQL kérés: `find_by_sql`
- △ Egyedi rekordok: `Task.find_by(:state).distinct,`
`Task.find_by(:state).uniq`
- △ Üres eredménytábla: `User.none`
- △ Létezik-e rekord: `User.find(1).exists?`

ActiveRecord – CRUD 6

6 Rekordok frissítése

- △ A teljes rekord mentésével

```
u=User.where(username:"Senki").take
u.email="senki2@mail.bme.hu"
u.save
```

- △ Egyetlen attribútum frissítésével:

```
u.update(:email,"senki2@mail.bme.hu")
```

- △ Több rekord frissítése: `u.update_attributes(params[:user])`

- △ Egyetlen rekord frissítése osztálymetódussal

```
User.update(1, :email => "senki2@mail.bme.hu")
```

- △ Az összes rekord frissítése osztálymetódussal:

```
User.update_all("password='changeme'", "password<>'")
```

Az első paraméter az SQL UPDATE kérés SET tagja, a második a WHERE tagja.

ActiveRecord – CRUD 7

- ⑥ Rekord visszaírása: `save`, `save!`. Az utóbbi sikertelenség esetén kivételt dob
- ⑥ Új rekord létrehozása: `create`, `create!`
- ⑥ A változtatások elfelejtése: `reload`. Teszteléskor hasznos.

ActiveRecord – CRUD 8

6 Rekordok törlése

- △ Egyetlen rekord törlése osztálymetódussal:

```
User.delete(1)
```

- △ Több rekord törlése osztálymetódussal:

```
User.delete([1,2])
```

- △ Feltételes törlés osztálymetódussal:

```
User.delete_all(["password = ''"])
```

- △ A `destroy` a `delete`-en túlmenően visszahívja a `before_delete` metódust, és törli az objektumot a memóriából is az összes asszociált és leszármazott objektummal együtt.

ActiveRecord – Scope

6 Gyakran használt lekérdezésekből függvényt készíthetünk

- △ `scope` függvény

- △ Első paraméter a függvény neve, a második paraméter egy lambda, amely egy lekérdezést definiál

- △ `scope :is_admin, -> {where(admin: true)}`

- △ **Paraméterrel:**

- `scope :created_before, ->(date) {where("created_at < ?", date)}`

- △ **Használat:** `User.is_admin, User.created_before(Date.today)`

ActiveRecord – Enum

- ⑥ Véges doménnel rendelkező attribútum értékeit elnevezhetjük a modellben

```
class User < ApplicationRecord
  enum type: [:student, :lecturer, :admin]
end
```

- ⑥ Scope-okat definiál az értékek alapján

- ⑥ **Setter:** `User.student!`

- ⑥ **Getter:** `User.student? # => true`

- ⑥ **Attribútum getter:** `User.type # => "student"`

- ⑥ `User.where(type: [:student, :lecturer])`

ActiveRecord – Analitikus függvények

6 Analitikus függvények, statisztika a tábla oszlopairól

```
average=User.average(:tasks)
max=User.maximum(:tasks)
min=User.minimum(:tasks)
total=User.sum(:tasks)
num=User.count
num=User.count :conditions =>
  "password<>'changeme'", :distinct
```

ActiveRecord – Relációk 1

- 6 Aggregáció: `composed_of`, `:class_name`, `:mapping`
- 6 Elavult

customers
id
created_at
credit_limit
first_name
initials
last_name
last_purchase
purchase_count

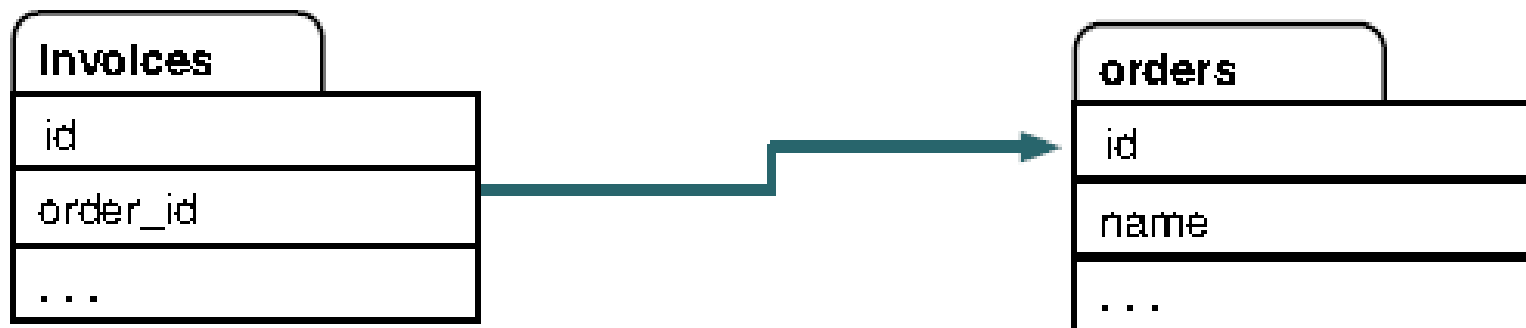
```
class Customer < ActiveRecord::Base
  composed_of :name,
    :class_name => Name,
  :mapping => [
    [ :first_name, :first ],
    [ :initials, :initials ],
    [ :last_name, :last ]
  ]
end
```

```
class Name
  attr_reader :first, :initials, :last

  def initialize(first, initials, last)
    @first = first
    @initials = initials
    @last = last
  end
end
```

ActiveRecord – Relációk 2

- 6 Egy-egy reláció: az objektum egy attribútuma (a tábla egy sora) egyetlen másik objektumra hivatkozik (egy másik tábla egyetlen sorára)

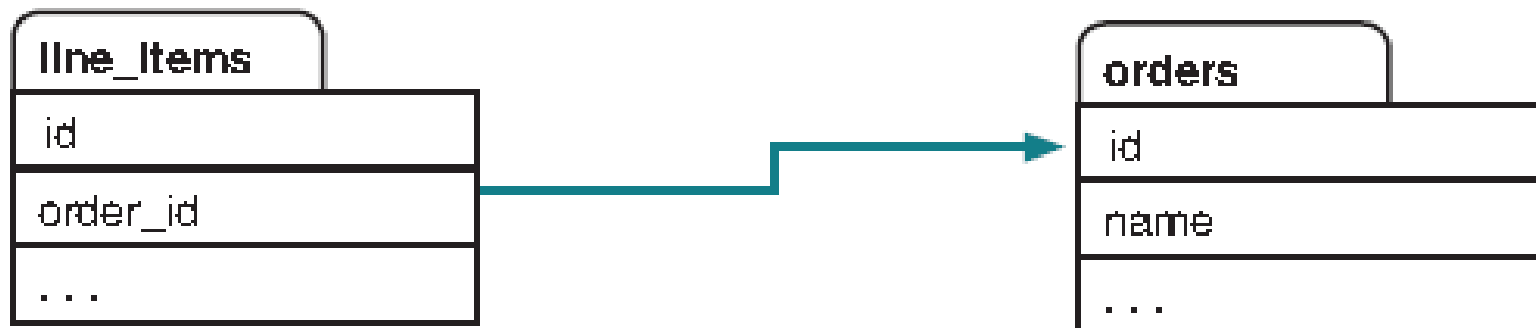


```
class Invoice < ActiveRecord::Base
  belongs_to :order
  # ...
end
```

```
class Order < ActiveRecord::Base
  has_one :invoice
  # ...
end
```


ActiveRecord – Relációk 3

- 6 Egy-több reláció: az objektum egy attribútuma (a tábla egy sora) azonos típusú objektumok listájára hivatkozik (egy másik tábla több sorára)

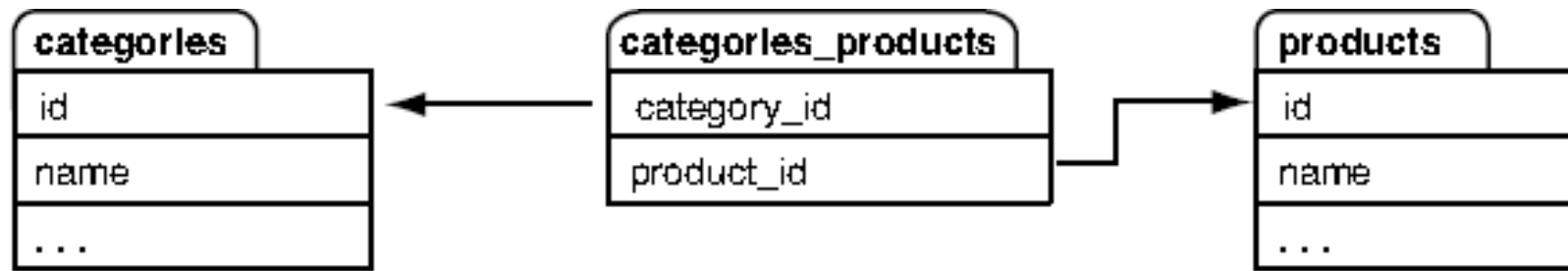


```
class LineItem < ActiveRecord::Base
  belongs_to :order
  # ...
end
```

```
class Order < ActiveRecord::Base
  has_many :line_items
  # ...
end
```

ActiveRecord – Relációk 4

- 6 Több-több reláció: az A típusú objektum egy attribútuma B típusú objektumok listájára hivatkozik, és a lista egy elemére több A típusú objektum is hivatkozhat.
- 6 A migrációban egy megfelelő kapcsolótábla létrehozandó! A neve konvenció szerint a két tábla nevének _ jellel összefűzve, ahol az ábécében előbb lévő tábla neve szerepel elől.



```
class Category < ActiveRecord::Base
  has_and_belongs_to_many :products
  # ...
end
```

```
class Product < ActiveRecord::Base
  has_and_belongs_to_many :categories
  # ...
end
```

ActiveRecord – Relációk 5

6 Konvenciók:

- △ A `belongs_to` deklaráció: `belongs_to :user`, az idegen kulcs `user_id`, a hivatkozó tábla neve `users`, a hivatkozó osztály neve `User`
- △ A `has_one` deklaráció: `has_one :invoice`, az idegen kulcs a másik táblában `order_id`, a hivatkozó tábla neve `invoices`, a hivatkozó osztály neve `Invoice`
- △ A `has_many` deklaráció: `has_many :tasks`, az idegen kulcs a másik táblában `user_id`, a hivatkozó tábla neve `tasks`, a hivatkozó osztály neve `Task`

ActiveRecord – Relációk 6

⑥ A `:class_name`, `a :foreign_key` és `a :conditions` opciókkal konfigurálhatók a relációk

⑥ Önreferencia:

```
class User
  has_many :friends,
    class_name: "User",
    foreign_key: "friend_id"
    association_foreign_key: "user_id",
    join_table: 'users_friends'
end
```

⑥ Migráció:

```
def change
  create_table :users_friends do |t|
    t.references :user
    t.references :friend
  end
end
```

ActiveRecord – Relációk 7

6 Automatikusan létrejövő metódusok:

- △ Setter, `user.tasks=i`
- △ Getter, `user.tasks`
- △ Gyorsítótár kikapcsolása, `user.tasks(true)`
- △ Asszociált objektum(ok) létezésének ellenőrzése: `user.task.nil?`
vagy `user.tasks.empty?`

ActiveRecord – Relációk 8

⑥ **Illesztés:** joins, left_outer_joins, includes, preload

⑥ `Order.joins(:invoices).where(sent: 1)`

⑥ **Közvetlen SQL-lel:**

```
Order.joins(  
  'INNER JOIN invoices  
  ON invoices.order_id = order.id AND  
  invoices.sent = 1'  
)
```

ActiveRecord – Relációk 9

- ⑥ Egy-több vagy több-több reláció modell osztályokkal:
:through opció
- ⑥ A reláció elnevezése opció: :source opció

```
class User < ActiveRecord::Base
  has_many :tasks, :through => :submissions
  has_many :submissions
end
```

```
class Task < ActiveRecord::Base
  has_many :users, :through => :submissions
  has_many :submissions
end
```

```
class Submission < ActiveRecord::Base
  belongs_to :task
  belongs_to :user
end
```

ActiveRecord – Relációk 10

- ⑥ Feltételes reláció opció: `:conditions => "priority = 5"`
- ⑥ Duplikátumok eltávolítása: `:unique`
- ⑥ Függőségek: `:dependent => :destroy`
- ⑥ Egy-több reláció listaként: `acts_as_list :scope => :user_id` a reláció több oldalán, `has_many :issues, :order => :priority` az egy oldalán
- ⑥ Fa struktúra (SQL CONNECT BY): `parent_id` oszlop definíciója a migrációban, és `acts_as_tree :order => ":user"` az ActiveRecord leszármazott osztályban

ActiveRecord – Relációk 11

6 Relációk mentése

- △ `has_one` asszociációban létező objektumhoz történő hozzárendelés automatikusan menti azt és a felülírtat
- △ `belongs_to` asszociációban nincs automatikus mentés
- △ Egy-több és több-több reláció esetén, ha a szülő objektum az adatbázisban van, akkor egy gyerek objektum hozzáadása a listához automatikusan menti a gyerek objektumot. Ha nincs, akkor a szülő mentésekor mentődik el.

ActiveRecord – Relációk 12

6 Polimorfizmus

6 Az :addresses tábla létrehozásakor:

`t.references :addr , :polymorphic => true,`
létrejön egy egész típusú `addr_id` és egy string típusú `addr_type` attribútum

```
class User < ActiveRecord::Base
  has_one :address, :as => :addr
end
```

```
class Order < ActiveRecord::Base
  has_one :address, :as => :addr
end
```

```
class Address < ActiveRecord::Base
  belongs_to :addr, :polymorphic => true
end
```

ActiveRecord – Relációk 13

- 6 Callback opciók: `:before_save`, `:after_save`, `:before_delete`, `:after_delete`
- 6 Paraméterük egy metódusnév szimbólumként vagy egy metódusneveket tartalmazó tömb
- 6 Például: `has_one :address, :before_save => :anonymize`

Kezdeti adatok

- ⑥ Kezdeti adatok betöltése az adatbázisba
- ⑥ A `db/seed.rb` fájl
- ⑥ Betöltés a `rails db:seed` vagy a `rails db:setup` paranccsal
- ⑥ Alternatíva: tesztadatok betöltése – később