



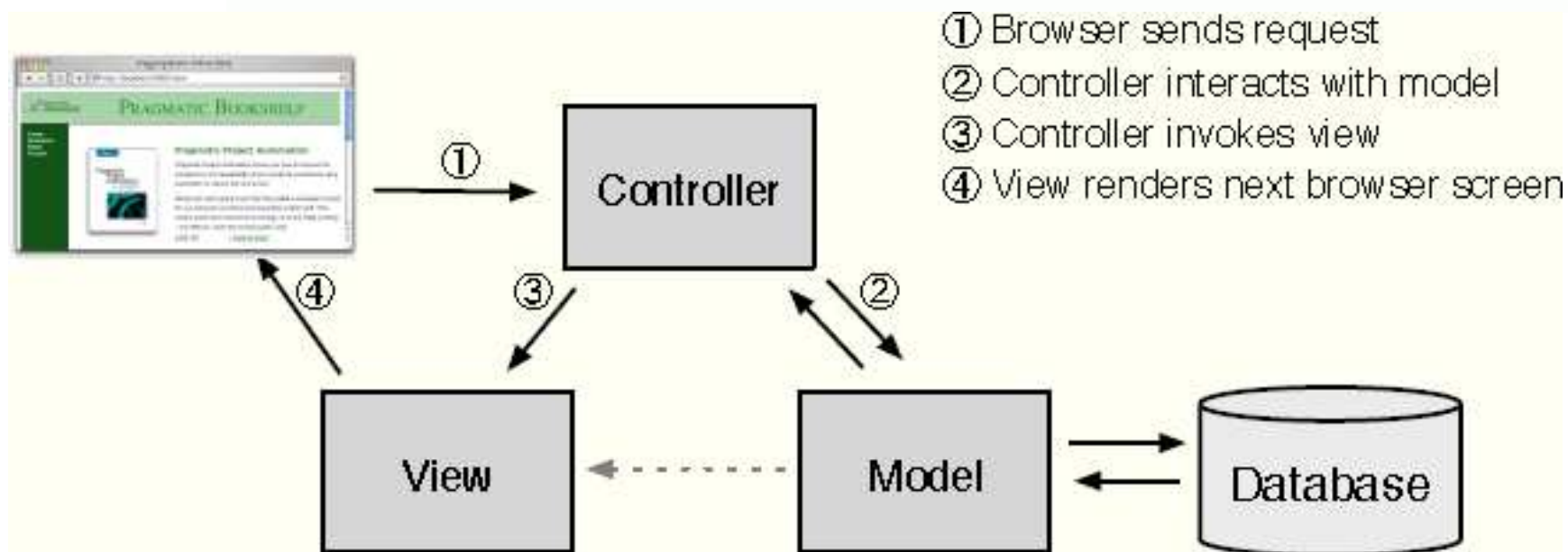
# ***Adatbázis kapcsolat és modellek***

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 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 `rake db:migrate`
- ⑥ Átállás egy másik verzióra  
`rake db:migrate VERSION=20110301121133`

## Migráció 2

- 6 A migráció két irányban lehetséges, amiket egy-egy osztálymetódus definiál

```
class MigrationName < ActiveRecord::Migration
  def self.up
    # ...
  end
  def self.down
    # ...
  end
end
```

## Migráció 3

- 6 A gyakorlaton megoldandó feladatban szereplő User modell migrációja

```
def self.up
  create_table :users do |t|
    t.string :neptun
    t.string :password, :limit => 40
    t.string :email
    t.boolean :student, :default => true
    t.timestamps
  end
end
```

# Migráció 4

- 6 A gyakorlaton megoldandó feladatban szereplő User modell migráció eredménye

attribútum

rekord

id	neptun	email	encrypted_password	student	created_at	updated_at	salt

Integer, nem NULL, automatikusan inkrementált

Varchar(6), nem NULL

Varchar(255)

Varchar(40)

tinyint(1), alapértelmezett érték 1

datetime, automatikusan létrejött

datetime, automatikusan létrejött

Varchar(40)

# Migráció 5

- ⑥ A Rails által támogatott típusok: `:binary`, `:boolean`, `:date`, `:datetime`, `:decimal`, `:float`, `:integer`, `:string`, `:text`, `:time`, `:timestamp`
- ⑥ Opciók
  - △ `:null`, lehet `true` vagy `false`
  - △ `:limit`, az adattípus hossza
  - △ `:default`, default érték
  - △ `:decimal` esetén `:precision` és `:scale`



## 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, :student, :string`

## 6 Migráció típusok – tábla módosítása (folyt.)

- △ Tábla létrehozása: lásd az `User` modell

- △ Kulcs

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

- △ 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`

## 6 Natív SQL

- △ `execute` metódus

## 6 Indexek

- △ Nagyszámú adat esetén hasznos az indexek bevezetése: `add_index` és

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

# 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 többszámú változata.

# 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

id	neptun	email	encrypted_password	student	created_at	updated_at	salt
1	OWEYOA	kovacsg@tmit.bme.hu		0	2010-10-15 12:50	2010-10-15 12:55	

```
u = User.new( :neptun =>'OWEYOA', :email=>'kovacsg@tmit.bme.hu', :student =>
false )
u.id
1
```

# 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.neptun= 'oweyoa'
u.email= 'kovacsg@tmit.bme.hu'
u.student=false
u.save
```

- ⑥ Az ActiveRecord konstruktor elfogad blokk vagy hash paramétert is:
- ⑥ `User.new do |u| ... end`
- ⑥ `User.new( :neptun => 'oweyoa' )`



# 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` rekordok/objektumok halmazát/tömbjét adja vissza, amely egyedi kulcs alapú keresés esetén singleton
- △ Az összes rekord lekérdezése: `users=User.find :all`
- △ A visszaadandó attribútumok egy részhalmazának kérése:  
`students = User.find(:all, :select => "neptun,email,student")`

# ActiveRecord – CRUD 3

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

```
users=User.find(:all, :conditions => "neptun='oweyoa' and student=0")
```

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

```
n=params[:neptun]
u = User.find(:all, :conditions=>"neptun='#{n}'")
u = User.find(:all, :conditions=>["neptun=?", n])
u = User.find(:all, :conditions=>["neptun=:nk", { :nk=>n }])
u = User.find(:all, :conditions=>["neptun=:neptun", params[:user]])
```

- 6 Hasonló rekordok keresése, SQL LIKE

```
u = User.find(:all, :conditions => ["neptun like ?", "owe"+"%"])
```

# ActiveRecord – CRUD 4

## 6 Kiegészítések a kereséshez

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

```
students = User.find(:all, :conditions=>"student<>0", :order => "neptun DESC")
```

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

```
students = User.find(:all, :conditions=>"student<>0", :order => "neptun DESC", :limit => 10)
```

- △ Az első visszaadott rekord sorszáma a tömbben:

```
students = User.find(:all, :conditions=>"student<>0",  
  :order => "neptun DESC", :limit => page_size, :offset => page*page_size)
```

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

```
students=User.find(:all, :order=>"neptun DESC", :group=>"student")
```

- △ Keresés másik táblában: :from

- △ Más táblák adatainak hozzáfűzése: :joins

- △ Nem írható vissza: :readonly=>true

# ActiveRecord – CRUD 5

## 6 Csak egy sor keresése

- △ `u=User.find(:first)`
- △ Ugyanazok a kiegészítések használhatók, mint a `:all` esetben

## 6 Egyéb keresési módok

- △ Direct SQL kérés: `find_by_sql` metódus
- △ Dinamikus keresés: `find_by_` és `find_all_by_` metódusok a `:conditions` helyett, amelyeket osztály attribútumnevekkel bővítünk ki. Például  

```
find_by_neptun("oweyoa"), find_all_by_student(true),  
find_by_neptun_and_email("oweyoa", "kovacsg@tmit.bme.hu")
```

# ActiveRecord – CRUD 6

## 6 Rekordok frissítése

- △ A teljes rekord mentésével

```
u=User.find_by_neptun("oweyoa")
u.email="kovacsg@db.bme.hu"
u.save
```

- △ Egyetlen attribútum frissítésével:

```
u.update_attribute(:email,"kovacsg@db.bme.hu")
```

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

- △ Egyetlen rekord frissítése osztálymetódussal

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

- △ Az összes rekord frissítése osztálymetódussal:

```
User.update_all("student=2","student<>0"). 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(["student <> 0"])
```

- △ 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 – Analitikus függvények

## 6 Analitikus függvények, statisztika a tábla oszlopairól

```
average=User.average(:submitted)
max=User.maximum(:submitted)
min=User.minimum(:submitted)
total=User.total(:student)
num=User.count
num=User.count :conditions =>
  "student<>0", :distinct
```



# ActiveRecord – Relációk 1

## 6 Aggregáció: `composed_of`, `:class_name`, `:mapping`

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 Táblák közötti reláció: idegen kulcsok a migrációban

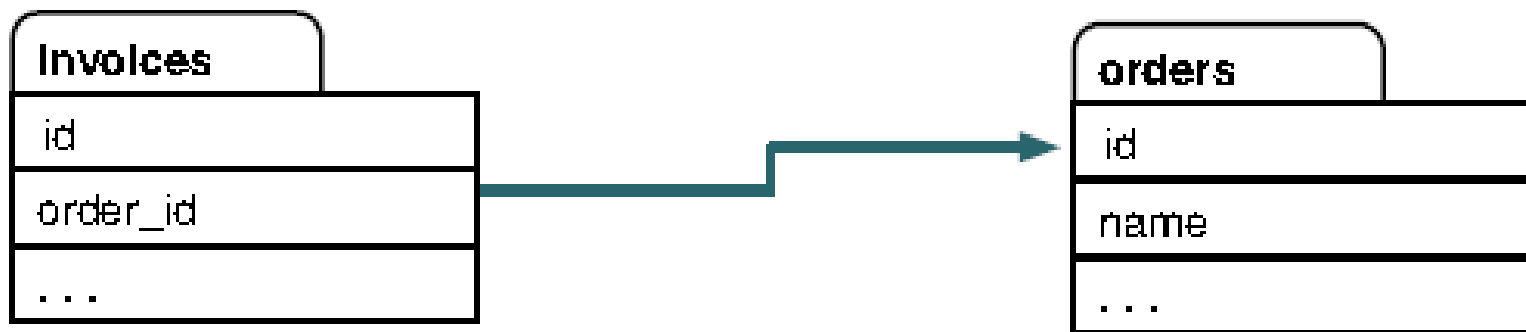
Users
id
neptun
email
encrypted_password
student
created_at
updated_at
salt

Solutions
id
user_id
task_id
late
missing
created_at
updated_at

Tasks
id
number
url
Deadline
created_at
updated_at

## ActiveRecord – Relációk 3

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

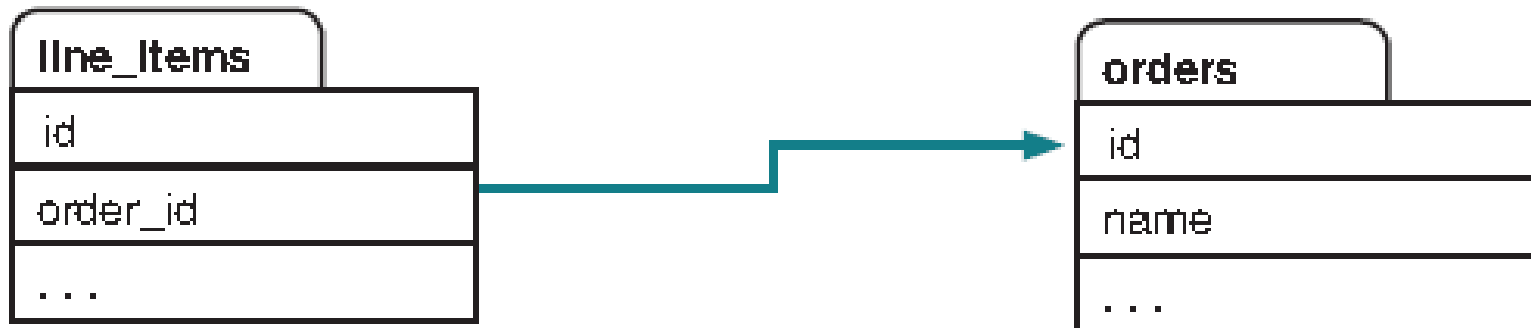


```
class Invoice < ActiveRecord::Base
  belongs_to :order
  # ...
end
```

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

## ActiveRecord – Relációk 4

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

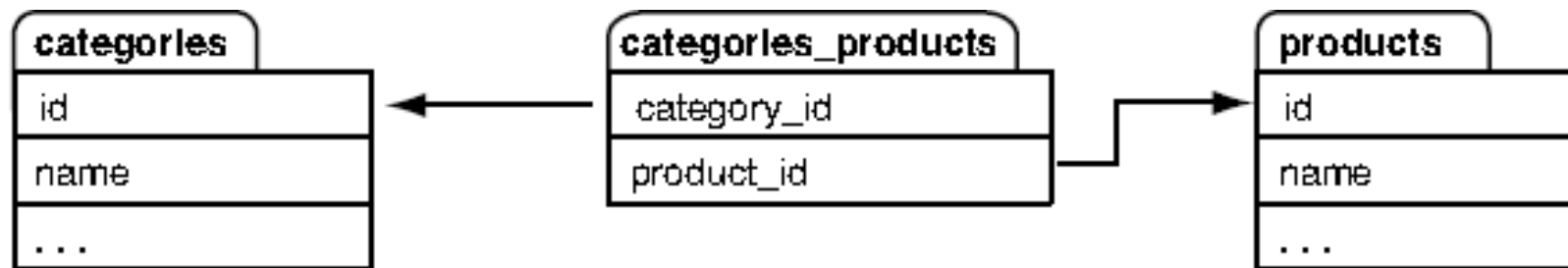


```
class LineItem < ActiveRecord::Base
  belongs_to :order
  # ...
end
```

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

# ActiveRecord – Relációk 5

- 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.



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

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

# ActiveRecord – Relációk 6

## 6 Konvenciók:

- △ A `belongs_to` deklaráció: `belongs_to :user`, az idegen kulcs `user_id`, a vonatkozó tábla neve `users`, a vonatkozó osztály neve `User`
  - △ A `has_one` deklaráció: `has_one :invoice`, az idegen kulcs a másik táblában `order_id`, a vonatkozó tábla neve `invoices`, a vonatkozó osztály neve `Invoice`
  - △ A `has_many` deklaráció: `has_many :solutions`, az idegen kulcs a másik táblában `user_id`, a vonatkozó tábla neve `solutions`, a vonatkozó osztály neve `Solution`
- 6 A `:class_name`, a `:foreign_key` és a `:conditions` opciókkal konfigurálhatók a relációk

# ActiveRecord – Relációk 7

- ⑥ Kapcsolótábla létrehozása egy-több relációkból: `:through`
- ⑥ A reláció elnevezése: `:source`
- ⑥ Feltételes reláció: `:conditions`
- ⑥ Duplikátumok eltávolítása: `:unique`
- ⑥ Önreferencia a `:class_name` és `:foreign_key` paraméterek konfigurációjával lehetséges
- ⑥ Egy-több reláció listaként: `acts_as_list :scope => :user_id` a reláció több oldalán, `has_many :tasks, :order => :position` 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 8

## 6 Relációk mentése

- △ `has_one` asszociációban létező objektumhoz történő hozzárendelés automatikusan menti azt
- △ `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 9

## 6 A példánk

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

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

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