



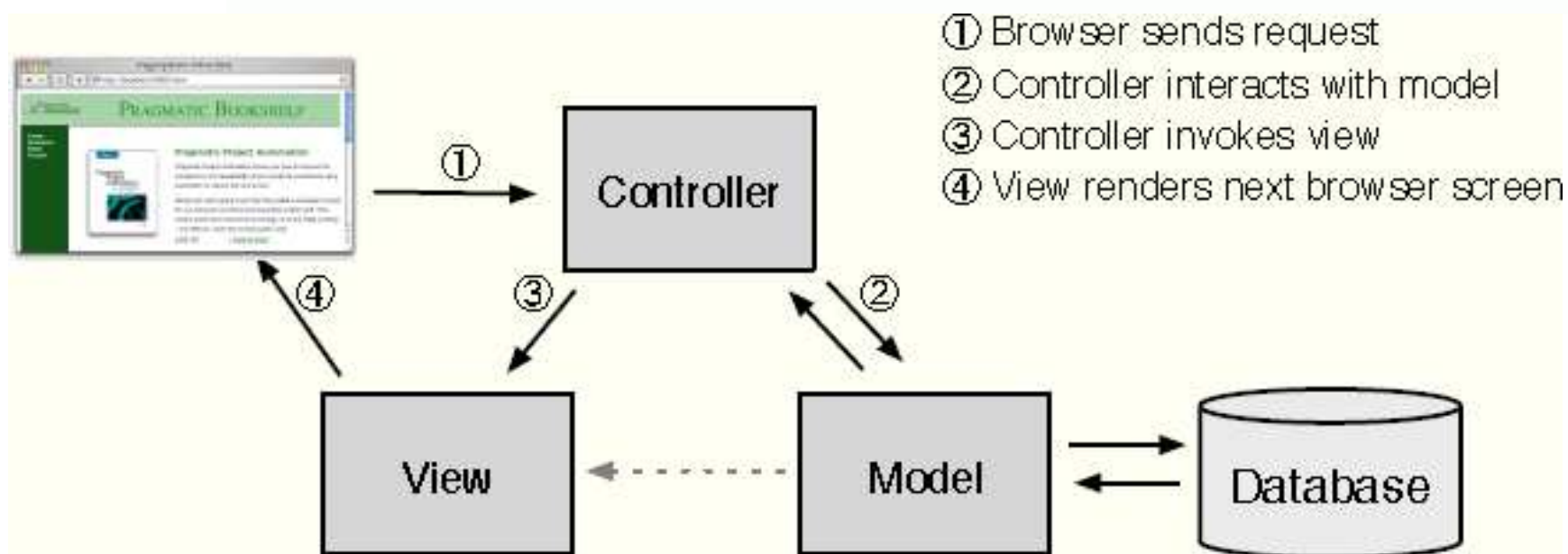
Adatbáziskapcsolat és modellek

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 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, `VERSION` és `STEP` paraméterek

```
rake db:migrate VERSION=20120229213548
```

```
rake db:rollback STEP=1
```

```
rake db:redo STEP=1
```

Migráció 2

- 6 A gyakorlaton megoldandó feladatban szereplő User modell 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 3

- 6 A gyakorlaton megoldandó feladatban szereplő User modell migráció eredménye

	id	username	password	email	created_at	updated_at

Integer, nem NULL, automatikusan inkrementált

varchar, nem NULL

varchar, nem NULL

varchar, nem NULL

datetime, automatikusan létrejött

datetime, automatikusan létrejött

Migráció 4

- ⑥ A Rails által támogatott típusok: `:binary`, `:boolean`, `:date`, `:datetime`, `:decimal`, `:float`, `:integer`, `: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`

6 Migráció típusok – tábla módosítása

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

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`

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 7

6 Natív SQL

- △ `execute` metódus string paraméterrel

6 Időpecsétek

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

6 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)

6 Idegen kulcsok

- △ A `t.references :issue` metódus létrehoz egy `issue_id` attribútumot, ami a `issues` tábla kulcsa
- △ Ez megvalósítható a modell osztályokon keresztül is

Migráció 8

- 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 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 8

6 Hozzáadott attribútumok visszamenőleges inicializációja:

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

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

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 típusná

id	username	password	email	created_at	updated_at
1	Senki	titok	senki@mail.bme.hu	2011-10-04 11:20:00	2011-10-04 11:20:00

u = User.new :username: "Senki", password: "titok", email: senki@mail.bme.hu
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.username= 'Senki'
u.email= senki@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` 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 => "username,email")`

ActiveRecord – CRUD 3

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

```
users=User.find(:all,  
  :conditions => "username='Senki' and email='senki@mail.bme.hu'")
```

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

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

- 6 Hasonló rekordok keresése, SQL LIKE

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

ActiveRecord – CRUD 4

6 Kiegészítések a kereséshez

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

```
users = User.find(:all, :conditions=>"email<>'",  
  :order => "username DESC")
```

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

```
students = User.find(:all, :conditions=>"email<>'",  
  :order => "username DESC", :limit => 10)
```

- △ Az első visszaadott rekord sorszáma a tömbben:

```
students = User.find(:all, :conditions=>"email<>'",  
  :limit => page_size, :offset => page*page_size)
```

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

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

- △ 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 attribútumneveivel bővítünk ki.

Például

```
find_by_username("Senki"),  
find_all_by_email("senki@mail.bme.hu"),  
find_by_username_and_email("Senki","senki@mail.bme.hu")
```

ActiveRecord – CRUD 6

6 Rekordok frissítése

- △ A teljes rekord mentésével

```
u=User.find_by_username("Senki")
u.email="senki2@mail.bme.hu"
u.save
```

- △ Egyetlen attribútum frissítésével:

```
u.update_attribute(: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 – Analitikus függvények

6 Analitikus függvények, statisztika a tábla oszlopairól

```
average=User.average(:issues)
max=User.maximum(:issues)
min=User.minimum(:issues)
total=User.sum(:issues)
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 vonatkozik (egy másik tábla egyetlen sorára)

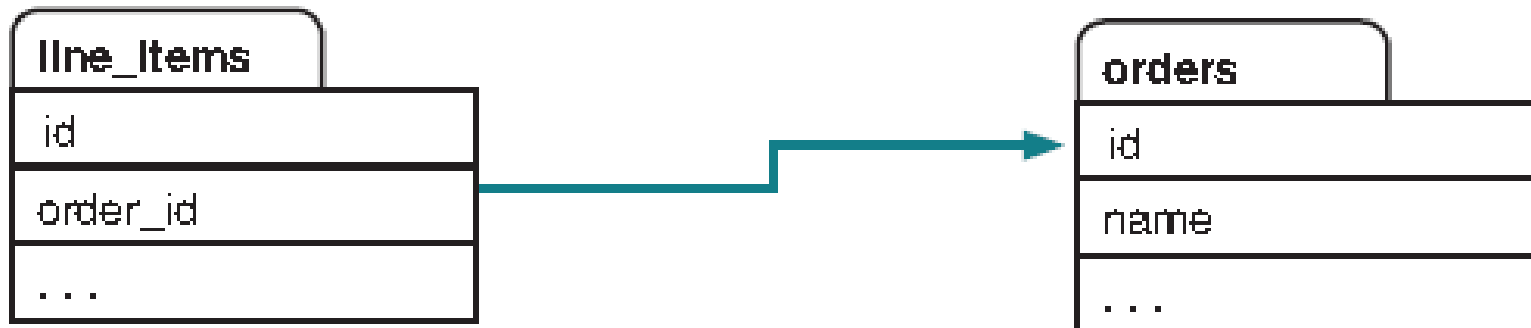


```
class Invoice < ActiveRecord::Base
  belongs_to :order
  # ...
end
```

```
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 vonatkozik (egy másik tábla több sorára)

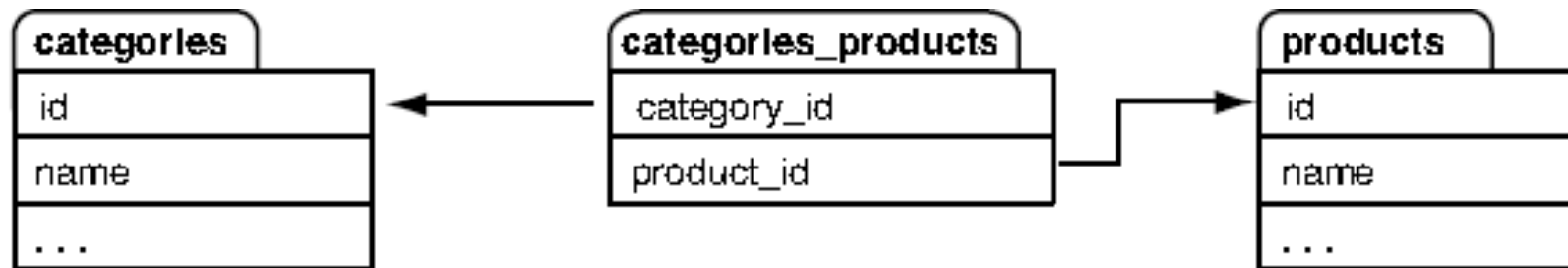


```
class LineItem < ActiveRecord::Base
  belongs_to :order
  # ...
end
```

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

ActiveRecord – Relációk 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 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 :issues`, az idegen kulcs a másik táblában `user_id`, a vonatkozó tábla neve `issues`, a vonatkozó osztály neve `Issue`

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

ActiveRecord – Relációk 6

6 Automatikusan létrejövő metódusok:

- △ Setter, `user.issues=i`
- △ Getter, `user.issues`
- △ Gyorsítótár kikapcsolása, `user.issues(true)`
- △ Asszociált objektum(ok) létezésének ellenőrzése: `user.issue.nil?`

ActiveRecord – Relációk 7

- ⑥ Több-több reláció modell osztályokkal: `:through` opció
- ⑥ A reláció elnevezése opció: `:source` opció

```
class User < ActiveRecord::Base
  has_many :issues
  has_many :comments, :through => :issues
end
```

```
class Issue < ActiveRecord::Base
  belongs_to :user
  has_many :comments
end
```

```
class Comment < ActiveRecord::Base
  belongs_to :issue
end
```

ActiveRecord – Relációk 8

- ⑥ Feltételes reláció opció: `:conditions => "priority = 5"`
- ⑥ Duplikátumok eltávolítása: `:unique`
- ⑥ Függőségek: `:dependent => :destroy`
- ⑥ Önreferencia a `:class_name` (modell osztály neve) és `:foreign_key` (idegen kulcs attribútum neve) paraméterek konfigurációjával lehetséges
- 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 9

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 10

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 11

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

Kezdeti adatok

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