



# Bevezetés

---



# Tartalom

---

- **Adminisztratív információk**
- Félév áttekintése
- Esettanulmány



# Elérhetőség

## ➤ Kovács Ferenc

- Q.B 222

- kovacs@aut.bme.hu

## ➤ Honlap

- <http://www.aut.bme.hu/Course/VIAUA369>

- Belépés

- Neptun kód
  - Neptun jelszó

} Neptun autentikáció



# Oktatási forma

---

## ➤ Előadás

- Kedd (I.E 007)
- Páratlan hét csütörtök (I.E v007)

## ➤ Gyakorlat

- Páros hét szerda, péntek
- Terem: beosztás szerint



# Számonkérések

---

## ➤ Gyakorlatok

### ■ Jegyzőkönyv

- Gyakorlat napján be kell adni
- Csak az adhat be, aki részt vett a gyakorlaton

### ■ Részvétel

- Legalább 5 gyakorlat
- Legalább 5 gyakorlat legalább 2-es jegyzőkönyvvel

### ■ A legjobb 4 pontszáma beszámít a vizsgajegybe

## ➤ ZH

- április 23
- 50%-ot el kell érni az aláíráshoz



# Osztályozás

---

## Megszerezhető pontok

|                 |                 |
|-----------------|-----------------|
| Gyakorlat       | 20 pont         |
| ZH              | 20 pont         |
| Vizsga          | 60 pont         |
| <b>Összesen</b> | <b>100 pont</b> |

## Ponthatárok

|              |   |
|--------------|---|
| 0- 50 pont   | 1 |
| 50- 60 pont  | 2 |
| 60- 70 pont  | 3 |
| 70- 84 pont  | 4 |
| 84- 100 pont | 5 |



# Tartalom

---

- Adminisztratív információk
- **Félév áttekintése**
- Esettanulmány



# Adatréteg

---

- Szerver platformok
  - Oracle Server 11g R2
  - MS SQL Server 2012
- Architektúrák
- Párhuzamos adathozzáférés
- Szerveroldali programozás
- Lekérdezés optimalizálás



# Köztes réteg

---

- Adatelérési osztálykönyvtárak
- O/R leképzés
- Linq
  - Linq2XML
  - Linq2SQL
- Entity Framework
- WCF (Windows Communication Foundation)



# Megjelenítési réteg

---

## ➤ Vékony kliens

- ASP.Net MVC
- JQuery
- HTML 5

## ➤ Vastag kliens

- WPF (Windows Presentation Foundation)
- Silverlight

## ➤ Mobil kliens

- WP7 (Windows Phone 7)



# Gyakorlatok

---

1. Tranzakció kezelés
2. Szerveroldali programozás – Oracle Server
3. Szerveroldali programozás – MS SQL Server
4. Linq 2 SQL
5. ASP.Net 1
6. ASP.Net 2
7. WPF



# Tartalom

---

- Adminisztratív információk
- Félév áttekintése
- **Esettanulmány**

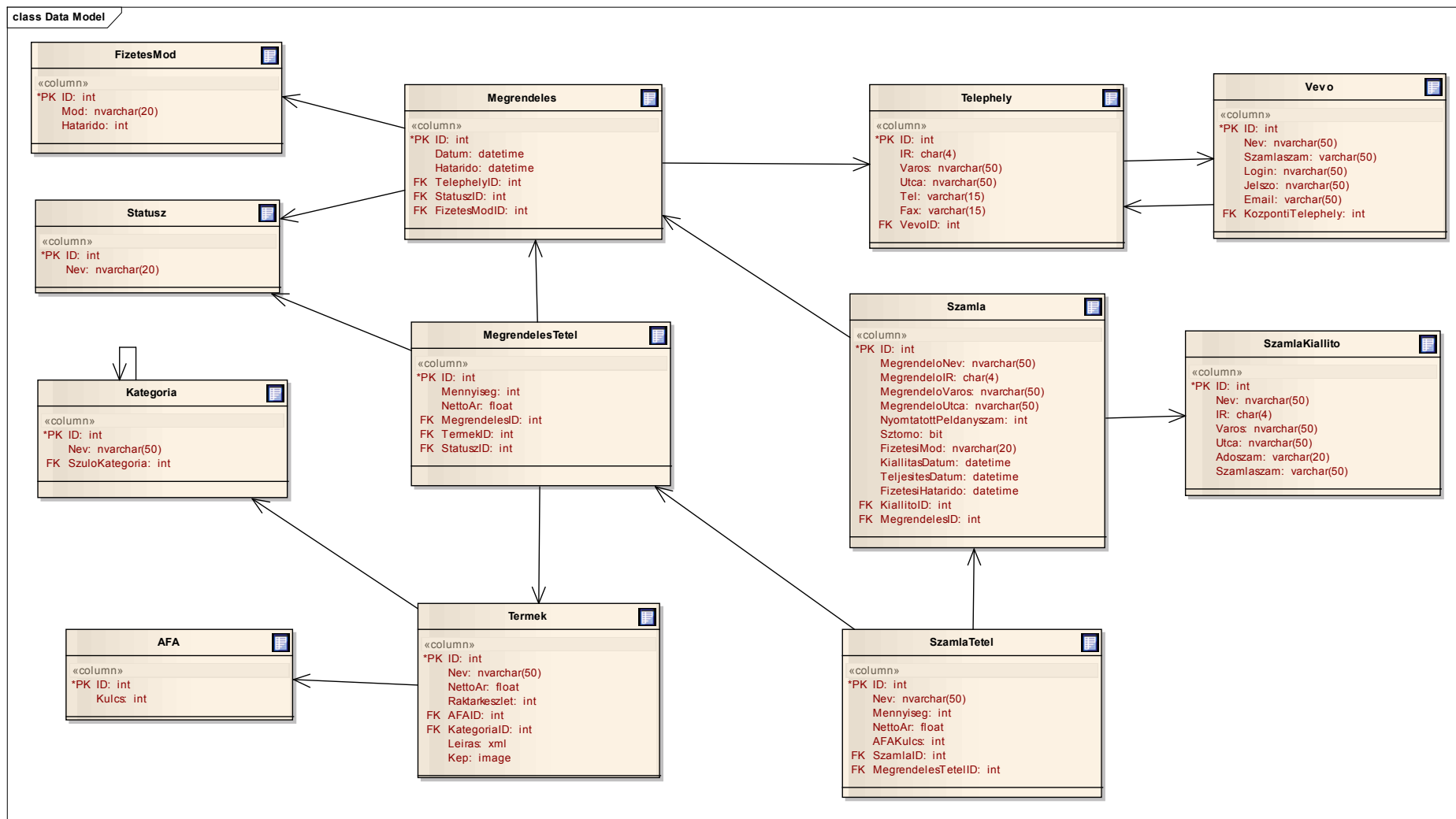


# Egyszerűsített ügyviteli rendszer

---

- Kereskedelmi cég (Játékbolt)
- Készletek nyilvántartása
- Vevők
- Megrendelések
- Számlázás

# Adatmodell





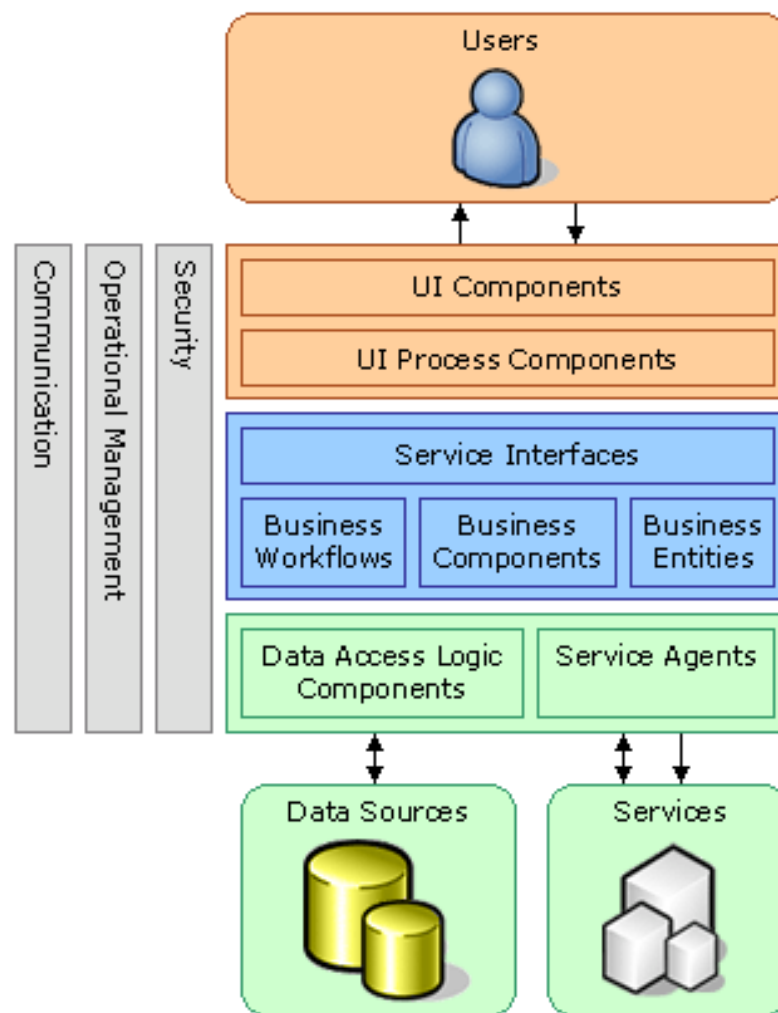
# Háromrétegű architektúra

---

## Áttekintés



# Architektúra elemei





# Megjelenítési réteg

---

- Tipikusan MVC architektúra
- A felület független a tranzakcióktól
  - Indítás, részvétel, szavazás



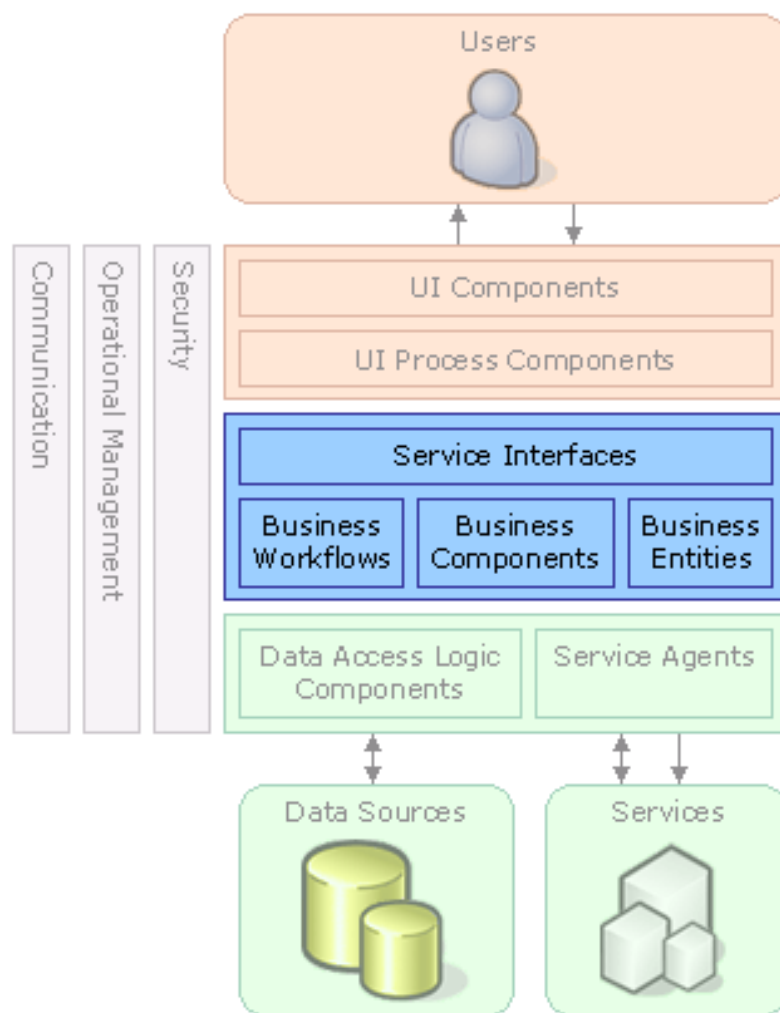
# Megjelenítési réteg feladatai

---

- Felhasználói input ellenőrzése
  - Formátum
  - Kötelező megadni
- Egyszerű transzformációk
  - Termék név megjelenítésben → Termék azonosító alsóbb rétegekben
- Adatok megjelenítése
  - Adatok lekérése a felhasználói interakció szerint
  - Státusz információ biztosítása
- Lokalizáció



# Üzleti logikai réteg





# Üzleti logikai réteg komponensei – 1

---

## ➤ Business Entities

- Alapobjektumok
- Információ hordozók, műveletek alapadatai

## ➤ Business Components

- Összetett komponensek
- Alapszolgáltatásokat implementál
- Tranzakciók
  - Része lehet tranzakciónak
  - Undo művelet biztosítása, ha nem tud részt venni



# Üzleti logikai réteg komponensei – 2

---

## ➤ Business Workflow

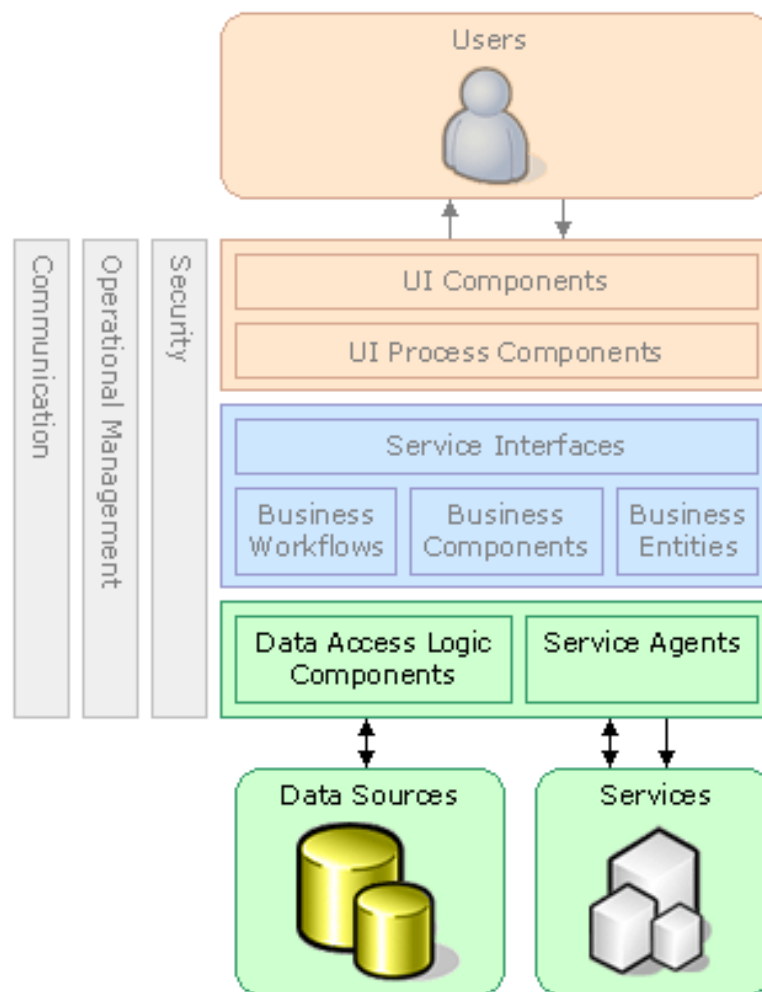
- Összetett üzleti folyamatok
- Komponensekből építkezik
- Tranzakció határok

## ➤ Service Interfaces

- Egységes hívási felület
- Belső rétegek elrejtése
- Üzleti műveletek



# Adatrétég





# Adatréteg komponensei

---

## ➤ DAL

- Adatbázis hozzáférés
- Elemi adatszolgáltatások

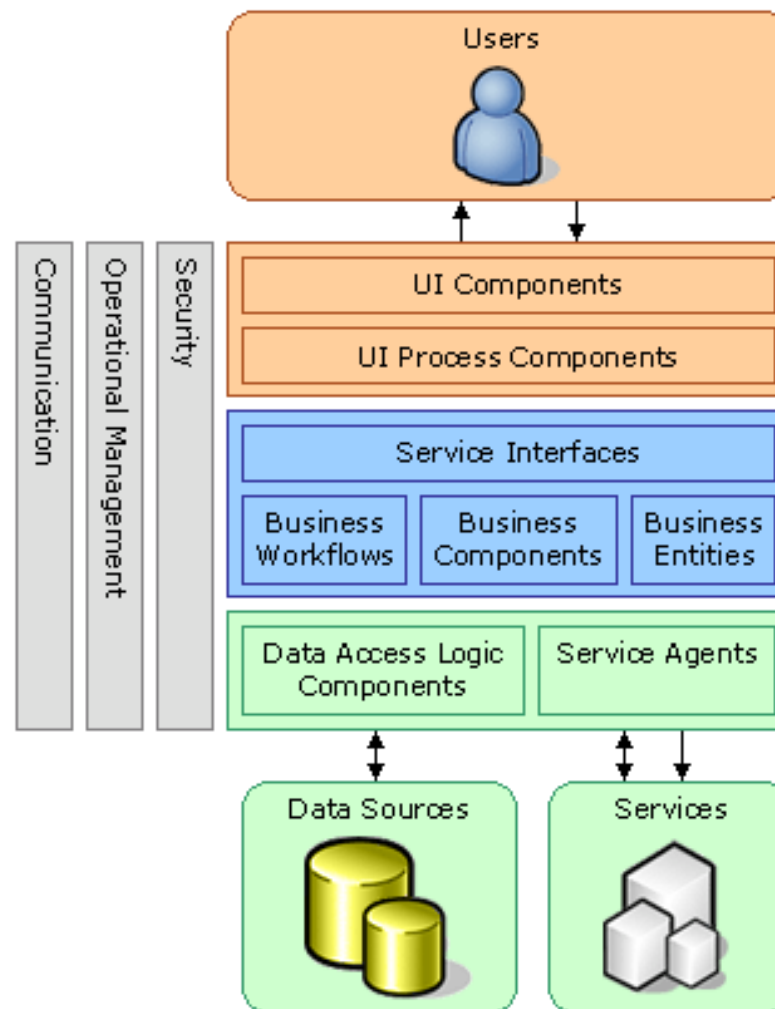
## ➤ Service Agents

- Szolgáltatások elérése
- Külső szolgáltatások „adatbázisként” látszanak
- Csomagolás



# Rétegszerkezet független szolgáltatások

- Biztonsági rendszer
- Üzemeltetési szolgáltatások
- Kommunikáció





# Biztonsági rendszer – 1

---





# Biztonsági rendszer – 2

---

## ➤ Felhasználó azonosítása

- OS ill. címtár
- Saját megoldás
- Single Sign on
  - Réteg közt is
  - Megszemélyesítés?

## ➤ Hozzáférés szabályozás

- Adatelérés
  - DBMS tábla ill. oszlop orientált
  - Sokszor sor szintű szükséges
- Szolgáltatás elérés



# Biztonsági rendszer – 3

---

## ➤ Nyomkövetés

- Felhasználók követése
  - Visszakereshető, hogy ki mikor mit csinált
  - Egy ember ne tudja eltüntetni a „nyomokat”
  - Megvalósítás
    - OS
    - DBMS
    - Üzleti folyamatok
- } Van beépítve, de sokszor kevés



# Üzemeltetési szolgáltatások – 1





# Üzemeltetési szolgáltatások – 2

---

## ➤ Kivételkezelés

- Egységes szemlélet
- Naplózás

## ➤ Monitorozás

- Működés ellenőrzése
- Technikai paraméterek
  - Válaszidő, átbocsájtó képesség ...



# Üzemeltetési szolgáltatások – 3

---

## ➤ Üzleti monitorozás

- SLA betartása
- SLA sértés okainak felderítése
  - Pl.: Késő beszállító

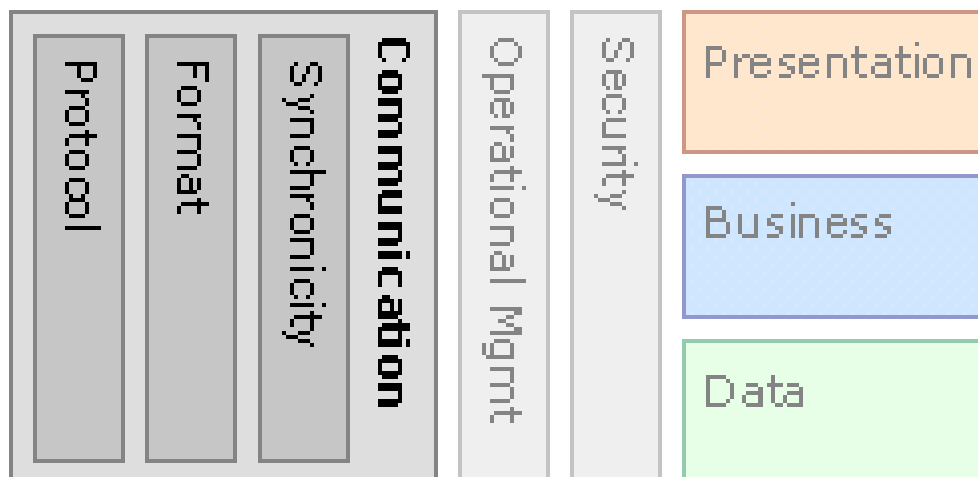
## ➤ Konfiguráció

- Egységes interfész az összes komponens számára
- Egységes konfigurációs megoldás
- Konfiguráció változtatás
  - Indításkor → konfigurációs fájlok
  - Futási időben → WMI, SNMP, ...



# Kommunikáció- 1

---





# Kommunikáció– 2

---

- Egyszerre több féle is lehet
- Protokoll beállítások
  - Hálózati protokoll típusa
  - Paraméterek
- Szinkronitás
  - Szinkron → „függvény hívás”
    - Tipikusan „belső” kommunikáció
  - Aszinkron
    - Üzenet alapú
    - Összetett, több komponensű rendszerek
    - Tranzakciók?



# DBMS Alapok

---



# Tartalom

---

- **Adatbázis**
- Adatmodell
- Fizikai adatbázis
- DBMS Architektúrák
  - MS SQL Server
  - Oracle Server



# Adatbázis definíció – 1

---

- Logikailag összefüggő adatok rendezett gyűjteménye
- Adatok - ismert tények, amelyek számítógépes adattárolón rögzíthetők
  - Hagyományos
    - Szöveg, Szám, Dátum
  - Multimédia
    - Kép, Hang, Video
  - Strukturált
    - XML



# Adatbázis definíció – 2

---

- Rendezett gyűjtemény
  - Könnyű tárolás
  - Könnyű módosítás
  - Könnyű lekérdezés
- Összefüggő adatok
  - Lefedik egy felhasználó csoport érdeklődési területét
  - Szükséges adatok



# Metaadatok – 1

---

|             |           |
|-------------|-----------|
| Kovács Béla | 12345678  |
| Nagy Aladár | 25898554  |
| Kiss István | 985332154 |

- Leírja az adott adat tulajdonságait
  - adatdefiníció
  - adatstruktúra
  - szabályok, és korlátozások
- Adatszótár (Data repository)



# Metaadatok – 2

---

## Oracle Server

### ➤ Tábla comment

- COMMENT ON TABLE  
Termek IS  
'Termék adatok '

## MS SQL Server

### ➤ Tábla comment

- EXEC  
sys.sp\_addextendedproperty  
@name=N'MS\_Description',  
@value=N'Termék adatok',  
@level0type=N'SCHEMA',  
@level0name=N'dbo',  
@level1type=N'TABLE',  
@level1name=N'Termek'



# Metaadatok – 3

---

## Oracle Server

### ➤ Oszlop comment

- `COMMENT ON COLUMN`  
Termek. Raktarkeszlet IS  
'Raktárban lévő mennyiség  
(DB)';

## MS SQL Server

### ➤ Oszlop comment

- `EXEC`  
`sys.sp_addextendedproperty`  
`@name=N'MS_Description',`  
`@value=N'A Raktárban lévő`  
`mennyiség (DB)' ,`  
`@level0type=N'SHEMA',`  
`@level0name=N'dbo',`  
`@level1type=N'TABLE',`  
`@level1name=N'Termek',`  
`@level2type=N'COLUMN',`  
`@level2name=N'Raktarkeszlet'`



# Tartalom

---

- Adatbázis
- **Adatmodell**
- Fizikai adatbázis
- DBMS Architektúrák
  - MS SQL Server
  - Oracle Server



# Relációs adatmodell

---

- Matematikai alapja van
- Alapvető komponensek
  - Tábla
    - Adattárolási struktúra
    - Adatok sorokba, és oszlopokba vannak szervezve
  - Integritási kritériumok
    - Érvényességi szabályok
  - Adatmanipuláció
    - Relációs algebra → SQL nyelv
    - Sor/oszlop kalkulus



# Relációk tulajdonságai

---

- Kétdimenziós tábla
- Megnevezett oszlopok
- Korlátlan számú sor
- Egyedi név
- Minden oszlopnak egyedi neve van a reláción belül
- Egy sor és oszlop kereszteződésében egyetlen érték szerepel
- Minden sor egyedi, nincs két egyforma sor
- A sorok sorrendje lényegtelen
- Attribútum
  - Nevezett oszlop
- Rekord
  - Sor
- Fokszám
  - Attribútumok Száma
- Kardinalitás
  - Sorok száma



# Integritási kritériumok – 1

---

## ➤ Tartományi integritás

- Oszlopban szereplő adatok ugyanabból a tartományból
- Adattípus
- Hossz
- Tartomány

## ➤ Entitás integritás

- Minden relációban van elsődleges kulcs
- Az elsődleges kulcs egyetlen részhalmaza sem lehet NULL



# Integritási kritériumok – 2

---

## ➤ Referenciális integritás

- A táblák közötti kapcsolatokat írja le
- Minden külső kulcs értékhez van megfelelő elsődleges kulcs érték a külső táblában
  - Csak létező rekordra hivatkozhat
  - Egy rekord nem törölhető, ha van rá hivatkozás

## ➤ Működési korlátozások

- Az üzletvitelből származtatható
- Közvetlenül nem írhatók le a relációs modellben
- Üzleti logika feladata



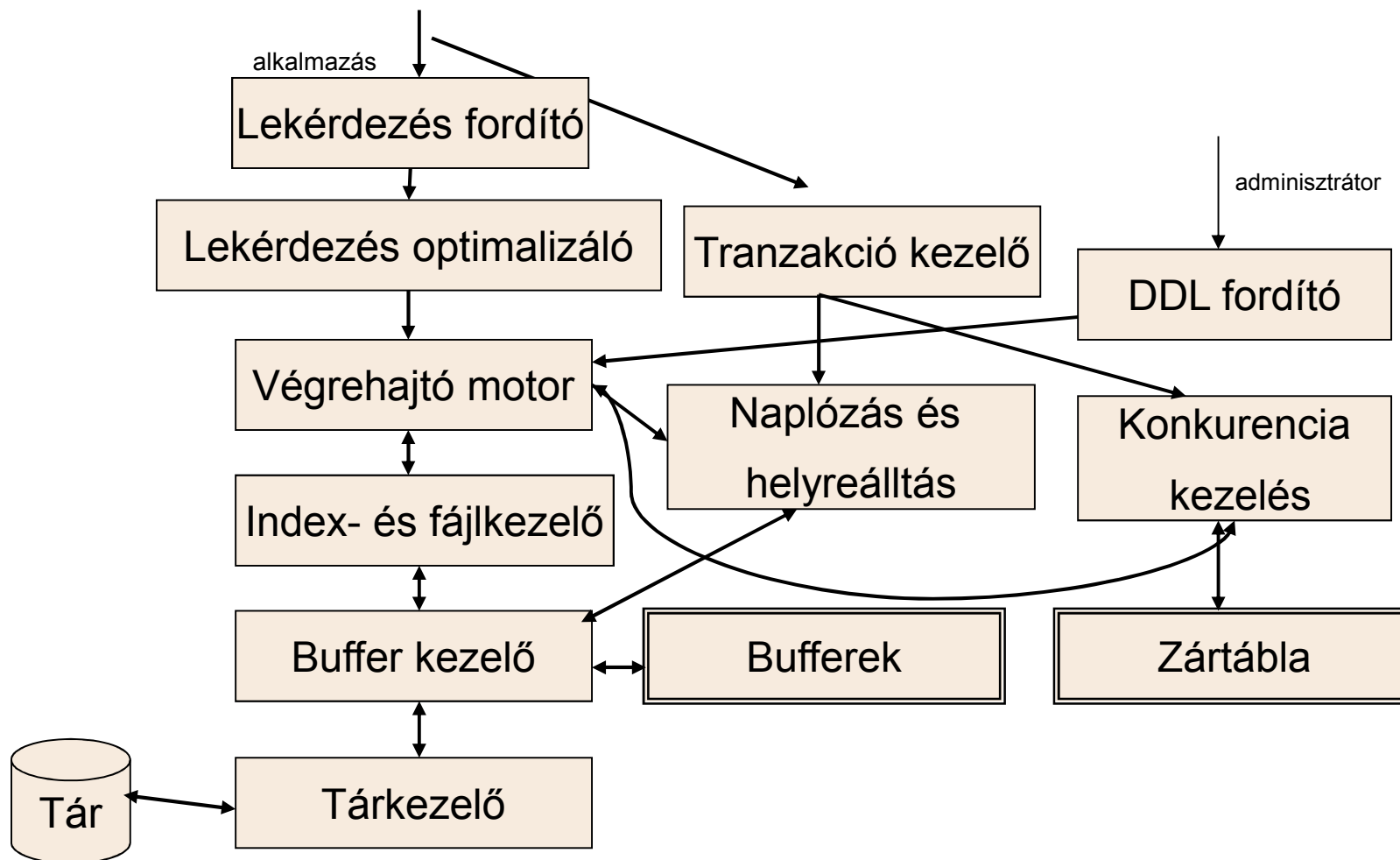
# Tartalom

---

- Adatbázis
- Adatmodell
- **Fizikai adatbázis**
- DBMS Architektúrák
  - MS SQL Server
  - Oracle Server



# Adatbáziskezelő- rendszer komponensei





# Relációs adatmodell a gyakorlatban

---

- Relációs adatbáziskezelő- rendszer
  - Modell támogatása
  - Tárolási struktúra absztrakciója → tábla
    - Csak definiálni kell
  - Integritási kritériumok
    - Kötelező betartani
    - Csak definiálni kell
  - Adatmanipuláció
    - SQL nyelv (DML)
    - Definíciós környezet



# Felhasználói séma

---

- Relációs adatmodellből adódó
  - Tábla
    - Adattípus platformfüggő lehet (SQL szabvány?)
  - Megszorítások
- Platformfüggő elemek



# Felhasználói séma elemei –MS SQL Server

## ➤ Tábla

- Oszlop
- Computed Column
  - Virtuális
  - Tárolt

## ➤ Nézet

- Indexelhető →Tárolódik

## ➤ Index

## ➤ Szekvencia

- SQL Server 2012

## ➤ Programmodul

- Eljárás
- Függvény
- Trigger
- Assembly



# Felhasználói séma elemei –Oracle Server

---

## ➤ Tábla

- Oszlop
- Virtual Column
  - Nem tárolt

## ➤ Nézet

- Nem indexelhető

## ➤ Materializált nézet

## ➤ Index

## ➤ Szekvencia

## ➤ Programmodul

- Eljárás
- Függvény
- Trigger
- Csomag
- Type
- Java package



# Adattípusok – 1

## MS SQL Server

### ➤ Szöveges típusok

- Char(n)
  - Nchar(n)
  - Varchar(n)
  - Nvarchar(n)
  - Varchar(max)
  - Nvarchar(max)
- } Byte  
Max 8000

## Oracle Server

### ➤ Szöveges típusok

- Char(n) → Byte | Char, max 2000 byte
- Nchar(n) → Byte, max 2000
- Varchar(n) → Byte, max 4000
- Varchar2(n) → Byte | Char, max 4000 byte
- Nvarchar2(n) → Byte, max 4000



# Adattípusok – 2

---

## MS SQL Server

### ➤ Numerikus

- Int
- Float
- Numeric(p,s)

### ➤ Dátum

- Datetime  
1753 január 1 ☺
- Datetime2

## Oracle Server

### ➤ Numerikus

- Integer
- Float
- Number(p,s)

### ➤ Dátum

- Date



# Adattípusok – 3

---

## MS SQL Server

### ➤ Nagyméretű objektumok

- Image
- TEXT

### ➤ Egyéb

- Money
- SQL\_VARIANT
- VARBINARY
- XML

## Oracle Server

### ➤ Nagyméretű objektumok

- BFILE
- BLOB
- CLOB
- NCLOB
- LONG
- LONG RAW

### ➤ Egyéb

- RAW
- ROWID
- XMLTYPE
- SYS.ANYDATA



# Elsődleges kulcsok generálása

## MS SQL Server

### ➤ Identity kulcsszó

```
create table Statusz(  
    ID int identity(1,1) primary key,  
    Nev nvarchar(20))
```

```
Insert into Statusz values ('Kész')
```

### ➤ Lekérdezése

- ident\_current('Statusz')
- @@IDENTITY
- SCOPE\_IDENTITY ()

## Oracle Server

### ➤ Szekvencia használatával

```
create sequence Statusz_seq  
start with 1 increment by 1;
```

```
create table Statusz(  
    ID int primary key,  
    Nev nvarchar(20));
```

```
Insert into Statusz values  
(Statusz_seq.nextval,'Kész');
```

### ➤ Lekérdezése

- Szekvencianév.Currval



# Tartalom

---

- Adatbázis
- Adatmodell
- Fizikai adatbázis
- **DBMS Architektúrák**
  - MS SQL Server
  - Oracle Server



# Szerver komponensek

---

## MS SQL Server

- SQL Server Service
- SQL Server Browser
- SQL Server Agent
- SQL Server Analysis Services
- SQL Server Reporting Services
- SQL Server Integration Services

## Oracle Server

- TNS Listener
- Oracle Server Service
  - OLAP option



# Adatbázisok tárolása

## MS SQL Server

### ➤ Adatbázis

- Adatfájl (.mdf)
  - Filegroup is lehet
- Tranzakciós napló (.ldf)
- Több sémát tartalmazhat

### ➤ Rendszer adatbázisok

- Master
- Model
- Distribution
- MSDB
- Temp

## Oracle Server

### ➤ Adatbázis nincs

### ➤ Felhasználói séma

### ➤ Táblahely

- Több felhasználói séma lehet
- Adatfájlokat fogja össze

### ➤ Beépített táblahelyek

- System
- Undo
- Users
- Temp



# Hozzáférés szabályozás

---

## MS SQL Server

- Rendszer szintű
- Adatbázis szintű
- Objektum szintű
  - Konkrét objektumhoz
  - Táblák és nézetek esetén oszlop szint is megadható
- Nemcsak engedélyezés, hanem tiltás is megadható

## Oracle Server

- Rendszer szintű
  - Összes sémára vonatkozik
  - Csak a saját sémára vonatkozik
- Objektum szintű
  - Konkrét objektumhoz
  - Táblák és nézetek esetén oszlop szint is megadható

Sor szintű hozzáférés nem szabályozható



# Tranzakciókezelés

---



# Tartalom

---

- **Tranzakciók alaptulajdonságai**
- Tranzakciók izolációja
- Tranzakciós naplózás



# Tranzakció fogalma

---

- A feldolgozás **logikai egysége**, olyan feldolgozási műveletek sorozata, mely csak együttesen értelmes
- Alaptulajdonságok:
  - Atomicity
  - Consistency
  - Isolation
  - Durability



# Atomitás és Konzisztencia

## ➤ Atomitás

- Egy egységet képez
- Oszthatatlan
  - Teljen végrehajtódik
  - Egyáltalán nem hajtódik végre
  - Köztes állapot nincs

## ➤ Konzisztencia

- Konzisztens állapotból konzisztens állapotba megy
- Hiba esetén is képes konzisztens állapotba visszaállni
  - Soft crash → Memória tartalom sérül
  - Hard crash → Háttértár sérül

Tranzakciós határ



# Tranzakciós határ – 1

---

- Meg kell tudni jelölni a tranzakciós határokat
  - Begin Transaction
  - Commit/ Rollback
- Specialitások
  - Savepoint
  - Rollback to savepoint



# Tranzakciós határ – 2

---

## ➤ MS SQL Server

- **Auto commit:** Minden utasítás önálló tranzakció
- Explicit tranzakciók: Begin tran, több utasítás esetén
  - Egymásba „ágyazható”
  - @@TRANCOUNT változó
- Implicit tranzakciók: Tranzakció vége jel után új tranzakció indul
- DML és DDL utasítások is tranzakció része
  - Néhány kivétel van (Create Database, backup, restore ...)



# Tranzakciós határ – 3

---

## ➤ Oracle Server

- **Implicit tranzakciók:** Tranzakció vége jel után új tranzakció indul
  - Nincs külön tranzakció kezdés utasítás
- Auto commit üzemmód
- Csak DML utasításokat tartalmazhat
  - DDL utasítások commitot implicálnak



# Demo

---

## Tranzakció határ



# Izoláció

---

- Tranzakciók hatása egymástól független
- Tranzakciók ütemezése
  - Mintha egymás után hajtódnának végre
  - Zárolás
  - Izolációs szintek
    - SQL Szabvány
    - Platformfüggő megoldások



# Tartós

---

- Hatása tartósan megmarad
- Nemcsak memóriában történik meg a módosítás
- Hiba
  - Soft crash → **Tranzakciós napló**
  - Hard crash → Mentés visszaállítás



# Tartalom

---

- Tranzakciók alaptulajdonságai
- **Tranzakciók izolációja**
- Tranzakciós naplózás



# Izolációs alapproblémák

---

- Sok párhuzamos tranzakció
- Úgy kell végrehajtani, mintha egymás után történnének és nem párhuzamosan
- Problémák
  - Elveszett módosítás
  - Piszkos olvasás
  - Nem megismételhető olvasás
  - Fantom rekordok



# Piszkos olvasás

---

- Egy tranzakció egy másik tranzakció nem commitált adatait használja
- Előfordulhat, hogy a tranzakció rollbacket hajt végre
- Nem lett volna szabad felhasználni az adatot



# Elvesztett módosítás

---

- Két párhuzamos tranzakció ugyanazon az adatelemen dolgozik
  - Commitált képből indulnak ki
  - Mindketten módosítanak
- Csak a később commitált hatása marad meg
  - A másik tranzakció hatása eltűnik



# Nem megismételhető olvasás

---

- Lekérdezés eredménye függ attól, hogy mikor adták ki egy tranzakcióban
  - Más tranzakció módosította az adatalelemet
  - Hibás programlogikát eredményezhet
    - Pl: Raktárkészlet ellenőrzése



# Fantom rekordok

---

- Kurzor megközelítés (rekordok halmaza)
  - Lekérdezés több rekordot ad vissza
  - Referenciát tárolunk a rekordokról
- Más tranzakciók hatása
  - Hivatkozott rekord törlődhet
  - Új rekord kerülhet be az adatbázisba, akinek a listában kellene szerepelnie



# Megoldás

---

- Tranzakciók ütemezése
- Csak olyan műveletek engedhetők meg, melyek nem sértik a helyes ütemezést
- Ha sérülne a helyes ütemezés, akkor a tranzakció vár
- Olyan ütemezés a megengedett, mely konfliktus ekvivalens egy soros ütemezéssel
  - Konfliktus mentes cserékkel soros ütemezéssé alakítható
  - Konfliktus
    - Ugyan arra az adatelemre vonatkoznak a műveletek
    - Legalább az egyik írás



# Ütemezés biztosítása

---

## ➤ Kétfázisú zárolás

### ■ Zár elhelyezés

- Kizáró zárolás az adatmódosítás előtt

### ■ Zár feloldás

- Csak a tranzakció végén

## ➤ Holtpont

### ■ Elkerülés

### ■ Detektálás

- Erőforrás foglalási gráf
- Irányított kör keresés



# Izolációs szintek

---

- Sorosítható ütemezés ritkán használt
  - Túl szigorú
  - Kevés párhuzamosságot enged meg
  - Üzleti logikából több párhuzamosíthatóság következik
- SQL szabvány szerinti izolációs szintek
  - Uncommitted read → mind a 4 probléma
  - Committed read → nincs piszkos olvasás
  - Repeatable read → nincs piszkos olvasás, nincs nem megismételhető olvasás
  - Serializable → egyik probléma sem fordulhat elő



# Demó

---

## Serializable izolációs szint



# Izolációs szintek támogatottsága

## SQL Server

- SQL szabvány
  - Uncommitted read
  - Committed read
  - Repeatable read
  - Serializable
- Szabványtól eltérő
  - Snapshot

## Oracle Server

- SQL szabvány
  - Committed read
  - Serializable
- Szabványtól eltérő
  - Read only
    - Serilazibale konzisztenciájú
    - Nem módosíthat
    - Tipikusan jelentésekhez



# Committed read implementációja

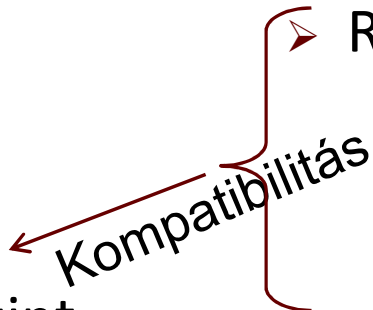
## MS SQL Server

- Select utasítás
  - Megosztott zárat használ
- Rekord módosul
  - Más nem olvashatja
- Snapshot izolációs szint

## Oracle Server

- Select utasítás
  - Nem helyez el zárat
- Rekord módosul
  - A commitált kép ***mindig*** olvasható
  - Rekord verziózás

Kompatibilitás





# Idegenkulcsok kezelése

---

- A hivatkozott rekordnak léteznie kell
- Zárolni kell a hivatkozott rekordot is
  - Platformfüggő
  - Nemcsak a hivatkozott rekord zárolódhat



# Demó

---

SQL Server zárolása

Külső kulcsok használata



# Tartalom

---

- Tranzakciók alaptulajdonságai
- Tranzakciók izolációja
- **Tranzakciós naplózás**



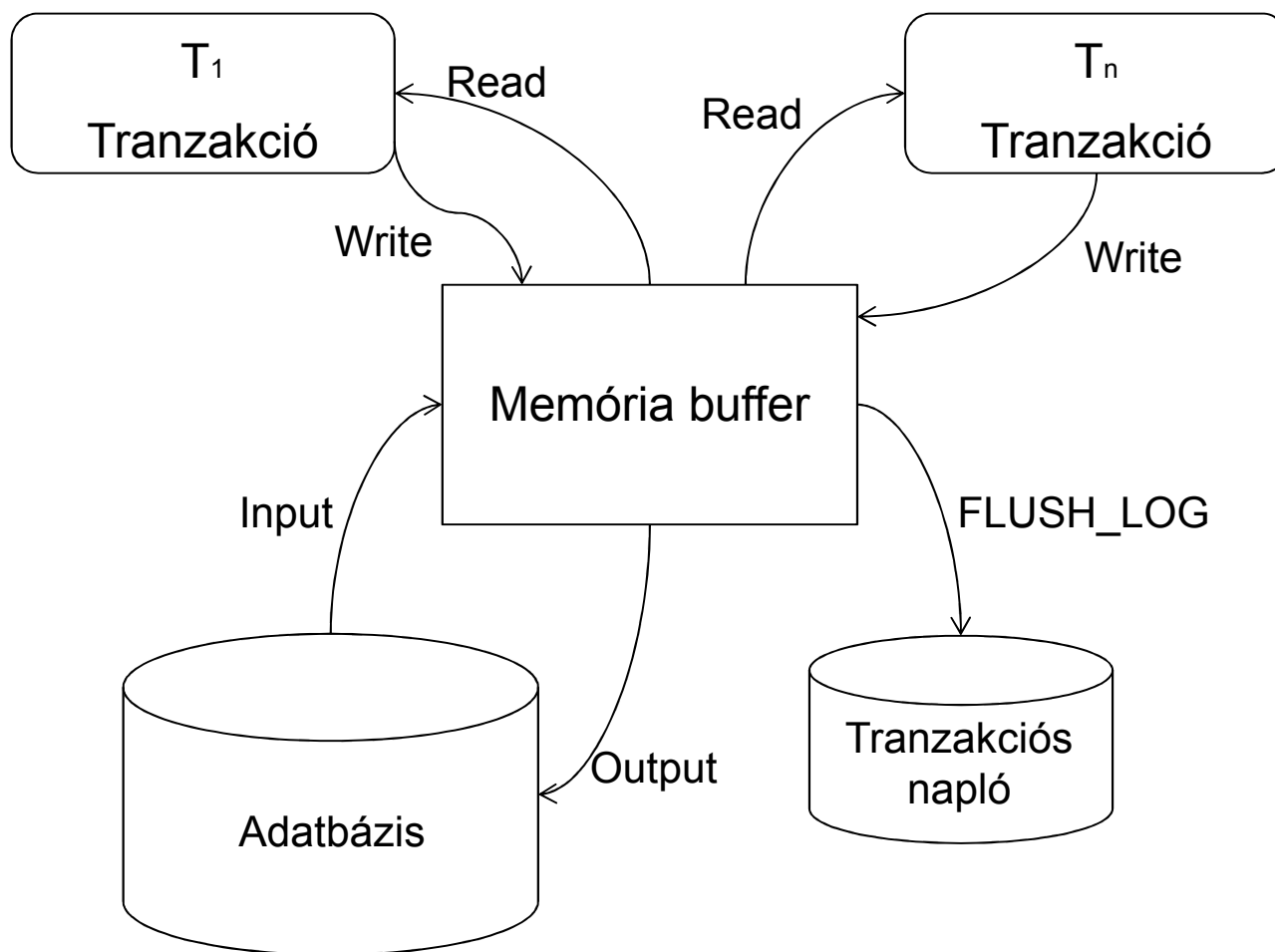
# Cél

---

- Soft crash elleni védekezés
  - Konzisztens állapot visszaállítása
  - Commiált tranzakciók megőrzése
  - Félbeszakadt tranzakciók visszagörgetése
  - Minimális overhead



# I/O modell – 1





# I/O modell – 2

---

- Input(A): Adatelem beolvasása
- Output(A) Adatelem kiírása
- Read(A,u): Tranzakció kiolvassa az adatelemet a memória bufferből
- Write(A,u): Tranzakció visszaírja az adatelemet a memória bufferbe
- FLUSH\_LOG: Tranzakciós napló diszkre írása



# Mintapélda

---

- Tranzakció két adatelemet módosít
  - A: 2-vel csökkent
  - B: 2-vel növel



# Undo típusú tranzakciós naplózás – 1

---

## ➤ Napló elemek:

- Begin T1: T1 tranzakció kezdete
- Commit T1: T1 tranzakció vége
- T1,A,u: T1 az A értékét u-ról megváltoztatta

## ➤ Szabályok

- Adatbázis nem írható át, amíg a tranzakciós napló nincs kiírva
- Commit jelet csak az adatbázis írás után lehet kitenni a naplóba



# Undo típusú tranzakciós naplózás – 2

---

## ➤ Visszaállítás

- Napló feldolgozása hátulról előre
- Félbemaradt tranzakciók esetén ott a régi érték a naplóban

## ➤ Hátrányok

- Kötött az adatbázis írás helye
- Kötött a commit jel helye
- Túl sok szinkronizáció a háttérfolyamatok között



# Mintapélda – Undo naplózás

|            | DB |    | Memória |    | Tranzakciós napló |
|------------|----|----|---------|----|-------------------|
| Művelet    | A  | B  | A       | B  |                   |
| Begin T1   | 10 | 20 | -       | -  | <Begin T1>        |
| Input(A)   | 10 | 20 | 10      | -  |                   |
| Input(B)   | 10 | 20 | 10      | 20 |                   |
| Read(A,u)  | 10 | 20 | 10      | 20 |                   |
| Write(A,u) | 10 | 20 | 8       | 20 | <T1,A,10>         |
| Read(B,u)  | 10 | 20 | 8       | 20 |                   |
| Write(B,u) | 10 | 20 | 8       | 22 | <T1,B,20>         |
| FLUSH_LOG  | 10 | 20 | 8       | 22 |                   |
| Output(A)  | 8  | 20 | 8       | 22 |                   |
| Output(B)  | 8  | 22 | 8       | 22 |                   |
|            |    |    |         |    | <Commit T1>       |

Commit



# Redo típusú tranzakciós naplózás – 1

---

## ➤ Napló elemek:

- Begin T1: T1 tranzakció kezdete
- Commit T1: T1 tranzakció vége
- T1,A,u: T1 az A értékét u-ra megváltoztatta

## ➤ Szabályok

- Adatbázis nem írható át, amíg a tranzakciós napló nincs kiírva
- Commit jelet az adatbázis írás előtt ki kell írni a naplóba



# Redo típusú tranzakciós naplózás – 2

---

## ➤ Visszaállítás

- Tranzakciós napló feldolgozása az elejétől
- Commitált tranzakciók újbóli végrehajtása

## ➤ Hátrányok

- Kötött az adatbázis írás helye
- Kevesebb, de még mindig sok szinkronizáció
- Hosszabb visszaállítás



# Mintapélda – Redo naplózás

|            | DB        |    | Memória |    | Tranzakciós napló |
|------------|-----------|----|---------|----|-------------------|
|            | A         | B  | A       | B  |                   |
| Begin T1   | 10        | 20 | -       | -  | <Begin T1>        |
| Input(A)   | 10        | 20 | 10      | -  |                   |
| Input(B)   | 10        | 20 | 10      | 20 |                   |
| Read(A,u)  | 10        | 20 | 10      | 20 |                   |
| Write(A,u) | 10        | 20 | 8       | 20 | <T1,A,8>          |
| Read(B,u)  | 10        | 20 | 8       | 20 |                   |
| Write(B,u) | 10        | 20 | 8       | 22 | <T1,B,22>         |
| Commit {   |           |    |         |    | <Commit T1>       |
|            | FLUSH_LOG | 10 | 20      | 8  | 22                |
|            | Output(A) | 8  | 20      | 8  | 22                |
|            | Output(B) | 8  | 22      | 8  | 22                |



# Undo/Redo tranzakciós naplózás – 1

---

## ➤ Napló elemek:

- Begin T1: T1 tranzakció kezdete
- Commit T1: T1 tranzakció vége
- T1,A,u,v: T1 az A értéket u-ról v-re megváltoztatta

## ➤ Szabályok

- Adatelem nem írható át az adatbázisban amíg a rá vonatkozó naplóbejegyzés kiírásra nem került
- Commit jel helye nem kötött



# Undo/Redo tranzakciós naplózás– 2

---

## ➤ Visszaállítás

- Undo és Redo módszereket egyszerre alkalmazva
- Commitált tranzakciók → after image
- Félbeszakadt tranzakciók → before image

## ➤ Előnyök

- Kevesebb szinkronizáció
- Adatelem előbb a tranzakció vége előtt átírható az adatbázisban
  - Általában több a sikeres mint a sikertelen tranzakció
  - Előnyösebb buffer kezelés



# Mintapélda – Undo/Redo naplózás

|            | DB |    | Memória |    | Tranzakciós napló |
|------------|----|----|---------|----|-------------------|
| Művelet    | A  | B  | A       | B  |                   |
| Begin T1   | 10 | 20 | -       | -  | <Begin T1>        |
| Input(A)   | 10 | 20 | 10      | -  |                   |
| Input(B)   | 10 | 20 | 10      | 20 |                   |
| Read(A,u)  | 10 | 20 | 10      | 20 |                   |
| Write(A,u) | 10 | 20 | 8       | 20 | <T1,A,10, 8>      |
| Read(B,u)  | 10 | 20 | 8       | 20 |                   |
| Write(B,u) | 10 | 20 | 8       | 22 | <T1,B,20, 22>     |
| FLUSH_LOG  | 10 | 20 | 8       | 22 |                   |
| Output(A)  | 8  | 20 | 8       | 22 |                   |
|            |    |    |         |    | <Commit T1>       |
| Output(B)  | 8  | 22 | 8       | 22 |                   |



# Tranzakciós napló csökkentése

---

- Checkpoint használata
  - A commitált tranzakciók kiírása az adatbázisba
- Új elemek a naplóban
  - Begin Checkpoint  $\langle T_1, \dots, T_n \rangle$ 
    - Checkpoint kezdődött
    - $T_1, \dots, T_n$  tranzakció aktív
  - End Checkpoint
    - Checkpoint folyamat befejeződött
- $T_1, \dots, T_n$  tranzakciók legelső bejegyzését megelőző részek eldobhatók



# Adatbázisok programozása

---

## Oracle Server programozása



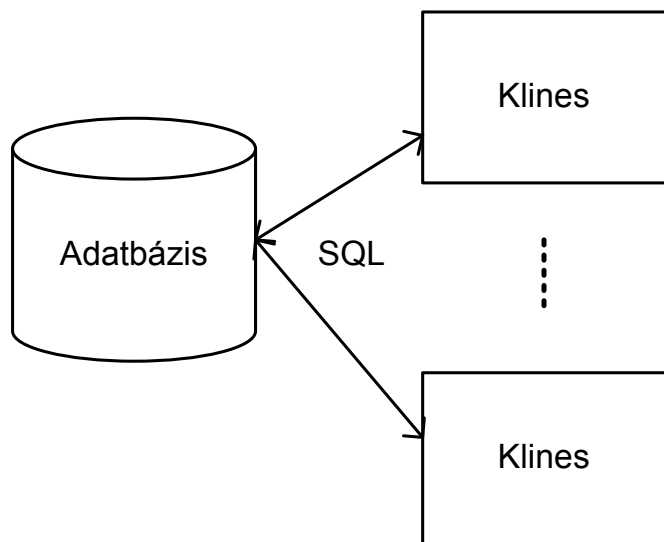
# Tartalom

---

- **Szerveroldali programozás szerepe**
- Oracle Server programozása
  - Tárolt eljárások
  - Triggerek
  - Csomagok
- PL/SQL nyelvi elemek
  - Alapelemek
  - Hibakezelés
  - Kurzorok használata
  - Tárolt eljárások készítése
  - Triggerek készítése



# Adatbázis szerepe



## ➤ Adatmodell

### ■ Táblák

## ➤ Adatmanipuláció

### ■ SQL nyelv

## ➤ Integritás

### ■ Tartomány

### ■ Entitás

### ■ Referenciális

Működési korlátozások!



# Működési korlátozások

---

- Kimutatnak a relációs adatmodellből
- Implementáció
  - Üzleti logikai réteg
  - Adatréteg → Szerveroldali programozás



# Szerveroldali programozás előnyei – 1

---

- Adatbázis felelős a konzisztenciáért
  - Adatbázis szerepe megváltozik
    - Adatforrás
    - Szolgáltatások
- Adatbiztonság
  - Adatmódosítás csak definiált interfészen keresztül
  - Biztonsági rendszer
    - Híváshoz csak futtatási jogosultság
    - Tulajdonos nevében fut
  - Zárt futtató környezet



# Szerveroldali programozás előnyei – 2

---

- Teljesítmény növelés
  - Csökkenő hálózati forgalom
  - Tárolt végrehajtási tervek
  - Cache
- Termelékenység
  - Több komponens hívhatja
  - Egyszerűbb karbantartás



# Szerveroldali programozás hátrányai

---

- Nem szabványos
  - Platformfüggő nyelvi elemek
  - Platformfüggő megoldások
- Interpretált
- Növeli a szerver terhelését
- Nem ill. nehezen skálázható



# Tartalom

---

- Szerveroldali programozás szerepe
- **Oracle Server programozása**
  - Tárolt eljárások
  - Triggerek
  - Csomagok
- PL/SQL nyelvi elemek
  - Alapelemek
  - Hibakezelés
  - Kurzorok használata
  - Tárolt eljárások készítése
  - Triggerek készítése



# Tárolt eljárások Oracle Serveren

---

## ➤ Belső

- PL/SQL nyelv
- Interpretált
- Zárt környezet
  - Futtatás
  - Adatbázisba zárt

## ➤ Külső

- Tetszőleges programozási nyelv
- Adatbázisból kimutató feladatok



# Tárolt eljárás létrehozása

---

- Create Procedure utasítás
- Szerver letárolja
  - Eljárás szövegét (fordítás sikerétől függetlenül)
  - Végrehajtási tervek
  - Futtatáshoz szükséges Pcode
  - Státusz (Érvényes/ érvénytelen)
  - Hivatkozott objektumok



# Fordítási folyamat

---

- Objektumok ellenőrzése
- Jogosultság ellenőrzés
  - Csak a közvetlen hozzárendelés
- Dinamikus SQL
  - Nem ellenőrzi
  - Nincs végrehajtási terv



# Tárolt eljárások érvényessége

---

## ➤ Érvénytelenné válik

- Hivatkozott objektum megváltozik
- Jogosultság megváltozik



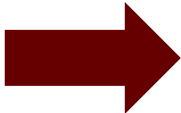
## ➤ Újra kell fordítani

- Automatikusan következő futtatáskor



# Külső tárolt eljárások

---

- Dinamikusan betölthető könyvtárak
  - Definiált interfészt implementál
  - Regisztrálni kell
  - Nincs futtató környezet
    - Bármit meg lehet tenni
    - Hiba lehetőség
      - Memória szivárgás
      - Védelmi hiba
-  Külön folyamat



# Tárolt függvény

---

- Olyan, mint a tárolt eljárás
- Visszatérési értéke van
- Bárhol használható, ahol az érték használható
  - Értékadás jobb oldalán
  - DML utasításban



# Néhány beépített függvény

- Nvl(kif1,kif2)
- Decode(kif1,  
                    érték1, érték2,...  
                    értékn értékn+1,  
                    defaultérték)
- Dátum kezelő függvények
  - SYSDATE
  - Add\_month()
  - Last\_day()
  - Month\_between
  - ...



# Tartalom

---

- Szerveroldali programozás szerepe
- **Oracle Server programozása**
  - Tárolt eljárások
  - **Triggerek**
  - Csomagok
- PL/SQL nyelvi elemek
  - Alapelemek
  - Hibakezelés
  - Kurzorok használata
  - Tárolt eljárások készítése
  - Triggerek készítése



# Triggerek használata

---

- Eseménykezelő tárolt eljárás
- Származtatott értékek karbantartása
  - Denormalizáció
    - Felfelé
    - Lefelé
    - Oldalra
- Naplózás
- Statisztikák gyűjtése
- Referenciális integritás szerverek közt



# Események

---

## ➤ DML esemény

- Insert, update, delete
- Táblához kötődik

## ➤ DDL trigger

- Create, alter, drop, ...
- Sémához kötődik

## ➤ Rendszeresemény

- Logon, logoff, SysError, ...

## ➤ Instead of trigger

- Speciláis DML trigger
- Nézetek adatmódosítása



# DML triggerek csoportosítása

---

## ➤ Felbontás

- Utasítás szintű
  - Módosított sorok nem elérhetők
- Sor szintű
  - Módosított rekord elérhető

## ➤ Ütemezés

- Adatmódosítás előtt
- Adatmódosítás után

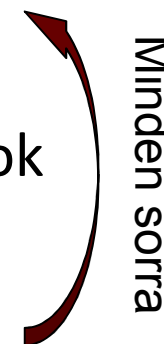


# DML triggerek futási sorrendje

---

## 1. Utasítás előtti triggerek (utasítás szintű)

- i. Sor módosítás előtti triggerek(sor szintű)
- ii. Adatmódosítás, zárolás, konzisztencia kritériumok ellenőrzése
- iii. Sor módosítás utáni triggerek(sor szintű)



## 2. Utasítás utáni triggerek (utasítás szintű)



# Módosult sorok elérése

- Trigger törzsében két rekord változó
  - Struktúrája illeszkedik az értékekhez
  - :new, :old
  - Adatmódosítás előtti triggerben a :new felülírható

|      | insert         | delete        | update             |
|------|----------------|---------------|--------------------|
| :new | Beszúrt rekord | NULL          | Rekord új értéke   |
| :old | NULL           | Törölt rekord | Rekord régi értéke |



# Triggerek egymásra hatásra

---

- Trigger kaszkádosítás
  - Olyan DML utasítást tartalmaz, mely triggert indít másik táblán
  - 32 mélységig megengedett
- Trigger rekurzió
  - Olyan DML utasítást tartalmaz a trigger, ami trigger újbóli meghívását eredményezi
  - Tiltott, futási hibát okoz
- Több trigger ugyanarra az eseményre
  - Lehet több triggert megadni
  - Végrehajtási sorrend
    - 10 g: nem definiálható
    - 11 g: megadható



# Triggerek és tranzakciók

---

- DML trigger
  - Részét képezi a tranzakciónak
- DDL trigger
  - Párhuzamos tranzakciót indít a felhasználó nevében
  - Automatikusan committal zárja le
- Rendszeresemény triggerek
  - Párhuzamos tranzakciót indít a Sys felhasználó nevében
  - Automatikusan committal zárja le



# Tartalom

---

- Szerveroldali programozás szerepe
- **Oracle Server programozása**
  - Tárolt eljárások
  - Triggerek
  - **Csomagok**
- PL/SQL nyelvi elemek
  - Alapelemek
  - Hibakezelés
  - Kurzorok használata
  - Tárolt eljárások készítése
  - Triggerek készítése



# Csomagok felépítése

---

- Eljárás gyűjtemény
- Package
  - Interfész
  - Kívülről hívható függvények
- Package Body
  - Interfész implementáció
  - Kívülről nem hívható függvények



# Csomag használatának előnyei

---

## ➤ Modularitás

- Logikai egységbe szervezhetők a tárolt eljárások
- Egységes menedzsment

## ➤ Egyszerűbb fejlesztés

- Elég a hívási interfészt kialakítani
- A többi modult lehet fejleszteni az interfész ismeretében

## ➤ Információ elrejtés

- Csak az interfész publikus

## ➤ Bővített funkcionalitás

- Session szintű változók definiálhatók

## ➤ Teljesítmény

- Használatkor a teljes csomag betöltődik a memóriába



# Csomagok elemei

---

- Eljárások
- Függvények
- Változó
  - Session szinten globális
  - Session folyamán végig megőrzi értékét
- Inicializációs blokk
  - Egyszer fut le egy sessionben
  - Amikor először használjuk a csomagot



# Néhány előre definiált csomag

---

- DBMS\_OUTPUT
  - „kiíratás a standard outputra”  
(SGA kimeneti bufferba)
- DBMS\_PIPE
  - Sessionök közötti kommunikáció
- UTL\_File
  - OS fájlok kezelés
- DBMS\_LOCK
  - Sessionök közötti szinkronizáció
- DBMS\_FGA
  - Audit
- DBMS\_SCHEDULER
  - Ütemezett feladatok
- ...



# Tartalom

---

- Szerveroldali programozás szerepe
- Oracle Server programozása
  - Tárolt eljárások
  - Triggerek
  - Csomagok
- **PL/SQL nyelvi elemek**
  - **Alapelemek**
  - Hibakezelés
  - Kurzorok használata
  - Tárolt eljárások készítése
  - Triggerek készítése



# PL/SQL nyelv elemei

---

- Oracle Server programozási nyelve
- DML nyelv
  - Insert, Update, Delete, Select, Merge
  - Paraméterezhető
- Vezérlési szerkezetek
  - Pascal jellegű
  - Elágazások, ciklusok
- Változók
  - Egyszerű, összetett
- Strukturált hibakezelés
- Kurzorok



# PL/SQL blokk felépítése

---

[DECLARE

<konstansok;>

<változók;>

<kurzorok;>

<felhasználó által definiált kivételek;>]

BEGIN

<PL/SQL futtatható utasítások;>

[EXCEPTION

<hibakezelés;>]

END;



# PL/SQL blokk típusok

---

## Anonymous

```
[DECLARE]

BEGIN
    --statements

[EXCEPTION]

END;
```

## Procedure

```
PROCEDURE name
IS

BEGIN
    --statements

[EXCEPTION]

END;
```

## Function

```
FUNCTION name
RETURN datatype
IS

BEGIN
    --statements
    RETURN value;

[EXCEPTION]

END;
```



# Változók megadása

---

## ➤ Deklaráció

- `nev VARCHAR2(20):='Kovacs Bela';`
- `ferfi BOOLEAN;`
- Inicilaizáció nélkül `NULL`

## ➤ Értékadás

- `ferfi:=true;`
- `SELECT name INTO nev  
FROM ember WHERE azonosito=5;` → Csak egy rekord esetén



# DML utasítások használata

---

➤ Szintaktika megegyezik az SQL nyelvi szintaktikával

➤ Változók használata

- Insert

- Values listán

- Delete

- Where feltételben

- Update

- Értékadás
  - Where feltételben

- Select

- Where feltétel
  - Having feltétel
  - Select listán

- Ahol nem befolyásolja az utasítás struktúráját



Dinamikus SQL



# Vezérlési szerkezetek – IF utasítás

---

**IF** feltétel **THEN**

<Utasítások1;>

**ELSE**

<Utasítások2;>

**END IF;**

**IF** feltétel1 **THEN**

<Utasítások1;>

**ELSIF** feltétel2 **THEN**

<Utasítások2;>

**ELSE**

<Utasítások3;>

**END IF;**



# Vezérlési szerkezetek – ciklusok

---

**LOOP**

...

**[EXIT]**

...

**END LOOP;**

**LOOP**

...

**[EXIT WHEN feltétel]**

...

**END LOOP;**

**WHILE <feltétel> LOOP**

...

**END LOOP;**

**FOR ciklus\_számláló IN [REVERSE]  
alsó\_határ..felső\_határ LOOP**

...

**END LOOP;**



# Demó

---

## Egyszerű tárolt eljárás



# Tartalom

---

- Szerveroldali programozás szerepe
- Oracle Server programozása
  - Tárolt eljárások
  - Triggerek
  - Csomagok
- **PL/SQL nyelvi elemek**
  - Alapelemek
  - **Hibakezelés**
  - Kurzorok használata
  - Tárolt eljárások készítése
  - Triggerek készítése



# Hibakezelése

---

## ➤ Kivételek

- Előre definiált hibák
- Felhasználó által definiált hibák

## ➤ Hibaüzenet generálása

- Feldolgozás megszakítása
- `RAISE_APPLICATION_ERROR` tárolt eljárás



# Kivételek kezelése

## ➤ Kivétel eldobása

- Raise utasítás
  - Felhasználó által definiált hiba
  - EXCEPTION típusú változó

## ➤ Kivétel elkapása

- Exception blokk
- Minden PL/SQL blokhöz tartozhat Exception blokk

**BEGIN**

utasítások\_sorozata1;

**EXCEPTION**

**WHEN** kivétel\_név1 **THEN**

utasítások\_sorozata2;

**WHEN** kivétel\_név2 **THEN**

utasítások\_sorozata3;

...

**WHEN OTHERS THEN**

utasítások\_sorozata4;

**END;**



# Néhány definiált kivétel

---

- ZERO\_DIVIDE
  - Nullával való osztás
- TOO\_MANY\_ROWS
  - Select utasítás egynél több sorral tért vissza
  - Select ... into
- NO\_DATA\_FOUND
  - Select utasítás nem talált rekordokat
  - Csoportfüggvények mindig adnak értéket
- DUP\_VAL\_ON\_INDEX
  - Elsődleges ill. egyedi kulcs sérült
- CURSOR\_ALREADY\_OPEN
  - Egy megnyitott kurzort ismét megnyitottunk
- INVALID\_CURSOR
  - Érvénytelen kurzor művelet (pl.: bezárunk egy nem nyitott kurzort)



# Alkalmazás hiba generálása

---

- `Raise_application_error(hiba_szám, üzenet);`
  - `hiba_szám`: -20000....-20999
  - Üzenet: 2048 byte
- Feldolgozás megszakad
- Visszakerül a vezérlés a hívóhoz



# Demó

---

## Egyszerű hibakezelés



# Tartalom

---

- Szerveroldali programozás szerepe
- Oracle Server programozása
  - Tárolt eljárások
  - Triggerek
  - Csomagok
- **PL/SQL nyelvi elemek**
  - Alapelemek
  - Hibakezelés
  - **Kurzorok használata**
  - Tárolt eljárások készítése
  - Triggerek készítése



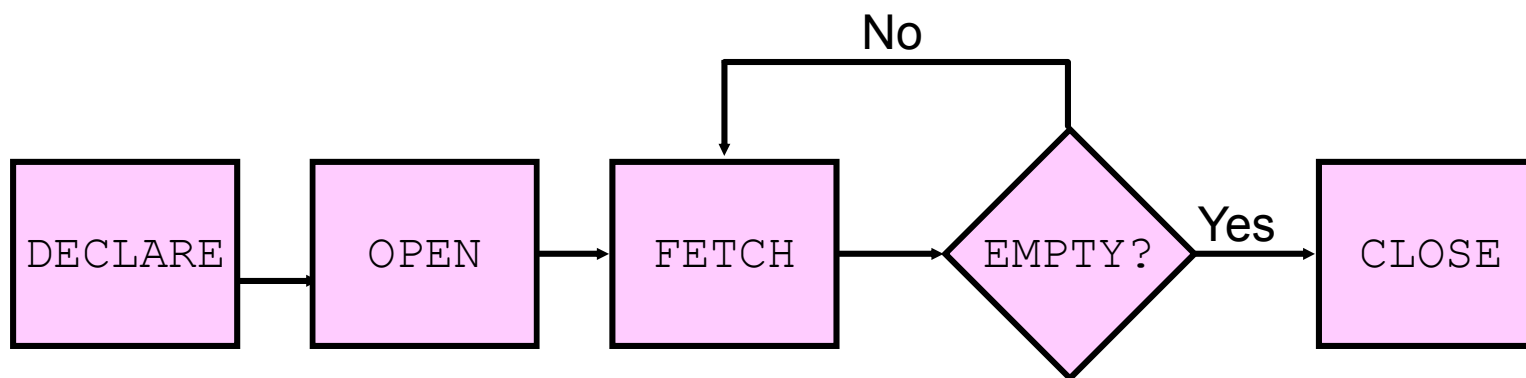
# Kurzorok működése

---

- Több rekordot visszaadó lekérdezések feldolgozása
- Rekordonkénti enumeráció
  - Megnyitás
  - Rekord feldolgozása
  - Kurzor léptetése



# Kurzorok használata





# Kurzor példa – 1

---

**DECLARE**

```
CURSOR termék_cursor IS  
  SELECT Nev, NettoAr FROM Termek  
  WHERE Raktarkeszlet >0;  
TermekNev nvarchar2;  
Ar float;
```

**BEGIN**

```
OPEN termék _cursor;  
LOOP  
  FETCH termék _cursor INTO TermekNev, Ar;  
  EXIT WHEN termék _cursor%NOTFOUND;
```

...

```
END LOOP;
```

```
CLOSE termék _cursor;
```

...

```
END;
```



## Kurzor példa – 2

---

**DECLARE**

**CURSOR** termék\_cursor **IS**

**SELECT** Nev, NettoAr **FROM** Termek

**WHERE** Raktarkeszlet >0;

Termek\_Record termék\_cursor%ROWTYPE

**BEGIN**

**OPEN** termék \_cursor;

**LOOP**

**FETCH** termék \_cursor **INTO** Termek\_Record;

**EXIT WHEN** termék \_cursor%**NOTFOUND**;

...

**END LOOP**;

**CLOSE** termék \_cursor;

...

**END**;



## Kurzor példa – 3

---

```
DECLARE CURSOR termék_cursor (Mennyiség  
integer) IS  
    SELECT Nev, NettoAr FROM Termek  
    WHERE Raktarakeszlet >Mennyiség;  
  
BEGIN  
    FOR rec in termék_cursor (5) LOOP  
        ...  
    END LOOP;  
    ...  
END;
```



# Demó

---

## Kurzorok



# Tartalom

---

- Szerveroldali programozás szerepe
- Oracle Server programozása
  - Tárolt eljárások
  - Triggerek
  - Csomagok
- **PL/SQL nyelvi elemek**
  - Alapelemek
  - Hibakezelés
  - Kurzorok használata
  - **Tárolt eljárások készítése**
  - Triggerek készítése



# Tárolt eljárások készítése

## Létrehozás

- Create or replace procedure
  - Lásd eddigi demók
- Bemenő paraméterek
  - Típusra nem lehet méretkorlátozás
    - Varchar2 → OK
    - Varchar2(30) → Hibás

## Paraméter módok

- In
  - Konstansként viselkedik (nem kaphat értéket)
  - Átveszi az értéket a hívótól
- Out
  - Változóként viselkedik
  - Érték visszaadására szolgál
  - NULL-ként inicializálódik
  - Alapértelmezésben nem veszi át az értéket a hívótól
- In Out
  - Változóként viselkedik
  - Átveszi az értéket a hívótól
  - Érték visszaadására is szolgál



# Demó

---

Lásd eddigi példák 😊



# Tartalom

---

- Szerveroldali programozás szerepe
- Oracle Server programozása
  - Tárolt eljárások
  - Triggerek
  - Csomagok
- **PL/SQL nyelvi elemek**
  - Alapelemek
  - Hibakezelés
  - Kurzorok használata
  - Tárolt eljárások készítése
  - **Triggerek készítése**



# DML Triggerek létrehozása

---

CREATE [OR REPLACE] TRIGGER trigger\_név

{BEFORE | AFTER}

{INSERT | DELETE | UPDATE} [OR [...n]]

ON táblanév

[FOR EACH ROW]

[WHEN (feltétel)]

{PL/SQL blokk}



# Demó

---

Elsődleges kulcsok automatikus generálása  
Származtatott érték karbantartása



# Adatbázisok programozása

---

## MS SQL Server programozása



# Tartalom

---

## ➤ MS SQL Server programozása

### ■ Tárolt eljárások

### ■ Triggerek

## ➤ TSQL nyelvi elemek

### ■ Alapelemek

### ■ Hibakezelés

### ■ Kurzorok használata

### ■ Tárolt eljárások készítése

### ■ Triggerek készítése



# Tárolt eljárás létrehozása

---

## ➤ Create procedure

- Létrehozza a tárolt eljárást a sémában
- Értelmezés, ellenőrzés
- Eltárolja a kódot

## ➤ Első futtatáskor

- Fordítás Minden SQL Server újrainduláskor!
- Optimalizálás

## ➤ Stored procedure cache

- Használaton kívüli tervek előregednek



# Tárolt eljárás újrafordítása

---

## ➤ Okok

- Új indexek bevezetésekor
- Az input adatok függvényében nagyon különböző részeit érinti az adatbázisnak
- Nem tipikus paraméterrel történő hívás esetén

## ➤ Újrafordítás

- sp\_recompile tárolt eljárás
- Execute with recompile futtatással



# Külső tárolt eljárások

---

- Szerepük hasonló, mint az Oracle-nél
  - SQL Server 2005-től már nem ajánlott
    - Megszüntetését tervezik
    - Kompatibilitás miatt maradt meg
- 
- .Net Assembly-k használata



# Tárolt függvények

---

- Visszatérési értékük van
- Adathatása nem lehet
- Típusok
  - Scalar- valued
    - Értéket számol ki
  - Table- valued
    - Rekordhalmazzal tér vissza
    - Táblaként használhatók lekérdezésekben
    - Komplex logikával lehet rekordokat összeválogatni
  - Aggregate
    - Saját oszlopfüggvény
    - .Net Assembly



# Néhány beépített tárolt függvény

- Isnull(kif1,kif2)
- CASE *input\_expression*  
WHEN *when\_expression* THEN  
*result\_expression* [ ...*n* ] [ ELSE  
*else\_result\_expression* ]  
END
- CAST ( *expression* AS *data\_type* [  
*length* ] )
- CONVERT ( *data\_type* [ ( *length* ) ] ,  
*expression* [ , *style* ] )
  - Style → Konverziót szabályozó konstans
- SQL Server 2012
  - Try\_cast
  - Try\_convert
- Dátum
  - Getdate()
  - DateAdd(datepart , number, date )
  - Month(date)
  - Year(date)
  - ...
- String
  - Replace
  - Trim
  - Ltrim
  - Rtrim
  - Len
  - Format
  - ...



# Tartalom

---

## ➤ MS SQL Server programozása

- Tárolt eljárások

- **Triggerek**

## ➤ TSQL nyelvi elemek

- Alapelemek

- Hibakezelés

- Kurzorok használata

- Tárolt eljárások készítése

- Triggerek készítése



# Trigger típusok

---

- DML trigger
  - Utasítás szintű
  - Tábla
  - Adatmódosítás után hajtódik végre
    - Insert, update, delete
- DDL trigger
  - SQL Server 2005
  - Felhasználói séma
- Instead of trigger
  - Utasítás szintű
  - Nézet, tábla
  - Insert, update, delete



# Módosított rekordok elérése

## ➤ Napló táblákon keresztül

- Szerkezetük megegyezik azzal a táblával, melyen fut a trigger
- Memóriában létező tábla, csak a triggerben értelmezett
- Minden sessionnek saját

|          | Insert      | Delete          | Update                |
|----------|-------------|-----------------|-----------------------|
| Inserted | Új rekordok | Üres            | Rekordok új értékei   |
| Deleted  | Üres        | Törölt rekordok | Rekordok régi értékei |



# Triggerek egymásra hatása

---

## ➤ Kaszkád triggerek

- Megengedett (→ nested triggers szerver paraméter)
- 32 mélységig

## ➤ Trigger rekurzió

- Megengedett (→ RECURSIVE\_TRIGGERS adatbázis paraméter)
- 32 mélységig

## ➤ Ugyanahhoz az eseményhez kapcsolt triggerek

- Teljes sorrend nem definiálható
- sp\_settriggerorder tárolt eljárás
  - Első
  - Utolsó



# Triggerek és tranzakciók

---

- A szülő tranzakció részét képezi
- SQL Server sajátosság
  - DDL utasítás is a tranzakció része
  - Majdnem minden utasítás tranzakcióban fut
    - Néhány kivétel
      - Create | Alter | Drop Databse
      - Backup, Restore
      - ...



# Instead of triggerek

---

- Adatmódosítás nézetekben
  - Insert, update, delete megvalósítása
- Táblákon is értelmezett
  - Adatmódosítás előtti triggerként működik
  - Ha a táblára hivatkozunk nem okoz rekurziót



# Tartalom

---

## ➤ MS SQL Server programozása

- Tárolt eljárások
- Triggerek

## ➤ **TSQL nyelvi elemek**

- **Alapelemek**
- Hibakezelés
- Kurzorok használata
- Tárolt eljárások készítése
- Triggerek készítése



# SQL Nyelvi elemek

---

## ➤ DML utasítások

- Select, Insert, Update, Delete, Merge
- Változók használata
  - Ahol nem befolyásolják a struktúrát

## ➤ DDL utasítások

- Használható
- Változó nem lehet benne



# Lokális változók

- Változó: @-al kezdődik, DECLARE utasítással deklaráljuk

**DECLARE** @változónév1 típus1, @változónév2 típus2

- Értékadás változónak:

- Deklarálás után a változó NULL, nem lehet alapértelmezett értéket adni neki

- SET utasítással

**DECLARE** @szam int

**SET** @szam = 3

- SELECT utasításon belül

**DECLARE** @szam int, @nev nvarchar(30)

**SELECT** @szam = id, @nev = nev **FROM** termek **WHERE**...

- Ha a lekérdezés több sorral tér vissza, a változó értéke az utolsó sor értékével egyezik meg



# Vezérlési szerkezetek

---

## ➤ IF...ELSE

**IF** boolean\_kifejezés

Utasítás\_blokk

**[ELSE**

Utasítás\_blokk]

## ➤ Utasítás blokk

**BEGIN**

TSQL utasítások

**END**

## ➤ A WHILE ciklus

**WHILE** boolean\_kifejezés

Utasítás blokk

## ➤ Ciklusvezérlő utasítások

■ **BREAK**

■ **CONTINUE**



# Demó

---

## Egyszerű tárolt eljárás



# Tartalom

---

## ➤ MS SQL Server programozása

- Tárolt eljárások
- Triggerek

## ➤ **TSQL nyelvi elemek**

- Alapelemek
- **Hibakezelés**
- Kurzorok használata
- Tárolt eljárások készítése
- Triggerek készítése



# Hibakezelés

---

## ➤ @@Error függvény

- Minden utasítás után lekérdezhető
- Ha nincs hiba, akkor 0-t ad vissza

## ➤ Kivétel kezelés

- Nincs Exception adattípus

**BEGIN TRY**

Utasítások

**END TRY**

**BEGIN CATCH**

Utasítások

**END CATCH**



# Hiba okának lekérdezése

---

- ERROR\_NUMBER()
  - Hibakód lekérdezése
- ERROR\_PROCEDURE()
  - Melyik programmodulban történt a hiba
- ERROR\_LINE()
  - Hányadik sorban
- ERROR\_MESSAGE()
  - Hibaüzenet szövege



# Hiba generálás - Raiserror

- sys.messages táblában lévő előre definiált hibák küldése

- Egyedi hibák

RAISERROR ( { *msg\_id* | *msg\_str* }  
 ,*severity* ,*state* [ ,*argument* [ ,...*n* ] ] )

- Severity

- 0...18: Felhasználói
- 19...25: csak sysadmin generálhatja
- 20...25: súlyos hiba, kapcsolatot is lezárja

- State

- Több helyről küldhetjük ugyan azt a hibát
- Helyek azonosítása
- 1...127



# Hiba generálás - Throw

---

- SQL Server 2012 újdonság
- Inkább már ezt használjuk

THROW error\_number, message, state

- Sajátosságok
  - Severity=16
  - Hibaüzenet nem formátum string alapú
    - FORMATMESSAGE függvény
  - Tovább dobható
    - Catch blokkban throw;



# Demó

---

## Hibagenerálás



# Tartalom

---

## ➤ MS SQL Server programozása

- Tárolt eljárások
- Triggerek

## ➤ **TSQL nyelvi elemek**

- Alapelemek
- Hibakezelés
- **Kurzorok használata**
- Tárolt eljárások készítése
- Triggerek készítése



# Kurzorok használata

---

- Több soros lekérdezések eredményének kezelésére szolgáló nyelvi eszköz
- A kurzorok kezelésének lépései
  1. Deklarálás (**DECLARE**), itt adjuk meg a lekérdezést
  2. Megnyitás (**OPEN**), ekkor fut le a lekérdezés
  3. Navigálás a sorok között (**FETCH**)
  4. Bezárás (**CLOSE**)
  5. Felszabadítás(**DEALLOCATE**)



# Deklarálás

---

**DECLARE** kurzornév **CURSOR**

[ **FORWARD\_ONLY** | **SCROLL** ]

[ **STATIC** | **KEYSET** | **DYNAMIC** | **FAST\_FORWARD** ]

[ **READ\_ONLY** | **SCROLL\_LOCKS** | **OPTIMISTIC** ]

**FOR** lekérdezés

[ **FOR UPDATE** [ **OF** oszlopnév [ ,...n ] ] ]



# Megnyitás és navigálás

## ➤ OPEN

**OPEN** kurzornév

## ➤ FETCH

**FETCH**

[ [ **NEXT** | **PRIOR** | **FIRST** | **LAST**

| **ABSOLUTE** { n | @nvar }

| **RELATIVE** { n | @nvar }

]

**FROM**

]

cursor\_name [ **INTO** @variable\_name [ ,...n ] ]

## ➤ @@FETCH\_STATUS

■ A legutolsó FETCH utasítás státuszát adja vissza

- 0 – sikeres FETCH
- -1 – sikertelen FETCH
- -2 – a lekért sor hiányzik



# Bezárás és felszabadítás

---

## ➤ CLOSE

- Aktuális rekordhalmaz felszabadítása
- Zárak elengedése
- Adatstruktúra megmarad, később megnyitható

**CLOSE** kurzornév

## ➤ DEALLOCATE

- Levesz egy referenciát a kurzorról
- Ha az összes referenciát levették, akkor megszűnik a struktúra

**DEALLOCATE** kurzornév



# Kurzor példa – 1

---

```
DECLARE Termekek CURSOR FAST_FORWARD READ_ONLY  
FOR  
SELECT Nev, NettoAr FROM Termek  
    WHERE Raktarkeszlet >0
```

```
DECLARE @Nev nvarchar(20)  
DECLARE @NettoAr
```

```
OPEN Termekek  
FETCH FROM Termekek INTO @Nev, @NettoAr  
WHILE @@FETCH_STATUS=0  
BEGIN  
    ...  
    FETCH FROM Termekek INTO @Nev, @NettoAr  
END  
CLOSE Termekek  
DEALLOCATE Termekek
```



## Kurzor példa – 2

---

**DECLARE @Raktarkeszlet int**

**DECLARE @Nev nvarchar(20)**

**DECLARE @NettoAr float**

**SET @Raktarkeszlet=5**

**DECLARE Termekek CURSOR FAST\_FORWARD READ\_ONLY  
FOR**

**SELECT Nev, NettoAr FROM Termek  
WHERE Raktarkeszlet > @Raktarkeszlet**

**OPEN Termekek**

**FETCH FROM Termekek INTO @Nev, @NettoAr**

**WHILE @@FETCH\_STATUS=0**

**BEGIN**

**...**

**FETCH FROM Termekek INTO @Nev, @NettoAr**

**END**

**CLOSE Termekek**

**DEALLOCATE Termekek**



# Demó

---

## Kurzorok használata



# Tartalom

---

## ➤ MS SQL Server programozása

- Tárolt eljárások
- Triggerek

## ➤ **TSQL nyelvi elemek**

- Alapelemek
- Hibakezelés
- Kurzorok használata
- **Tárolt eljárások készítése**
- Triggerek készítése



# Tárolt eljárások kezelése

---

## ➤ Létrehozás

- **CREATE PROCEDURE** → Önálló batch

**CREATE PROC [ EDURE ]** *eljárás\_név*  
[ { *@paraméter adat\_típus* }  
] [ ,...*n* ]

**AS** *sql\_utasítások* [ ...*n* ]

## ➤ Módosítás

- **ALTER PROCEDURE**

- meg kell adni az egész új eljárást.

## ➤ Törlés

- **DROP PROCEDURE**



# Információ tárolt eljárásokról

---

- Sp\_helptext
  - az eljárás kódja, ha nem titkosított
- Sp\_depends
  - hivatkozott objektumok listája
- Sp\_rename
  - Eljárás átnevezése



# Demó

---

Lásd eddigi példák 😊



# Tartalom

---

## ➤ MS SQL Server programozása

- Tárolt eljárások
- Triggerek

## ➤ **TSQL nyelvi elemek**

- Alapelemek
- Hibakezelés
- Kurzorok használata
- Tárolt eljárások készítése
- **Triggerek készítése**



# DML trigger létrehozása

---

**CREATE TRIGGER** *trigger\_név*

**ON** { *tábla* | *nézet* }

**FOR** { [ **DELETE** ] [ , ] [ **INSERT** ]

[ , ] [ **UPDATE** ] }

**AS**

*sql\_utasítás* [ ...*n* ]



# Demó

---

## Triggerek használata



# Lekérdezés optimalizálás

---



# Tartalom

---

- **Lekérdezések feldolgozása**
  - **I/O költség**
  - Logikai lekérdezési terv
  - Fizikai végrehajtási terv
- Oracle Server lekérdezés optimalizációja
- MS SQL Server lekérdezés optimalizációja
- Néhány jó tanács



# Válaszidőt befolyásoló tényezők

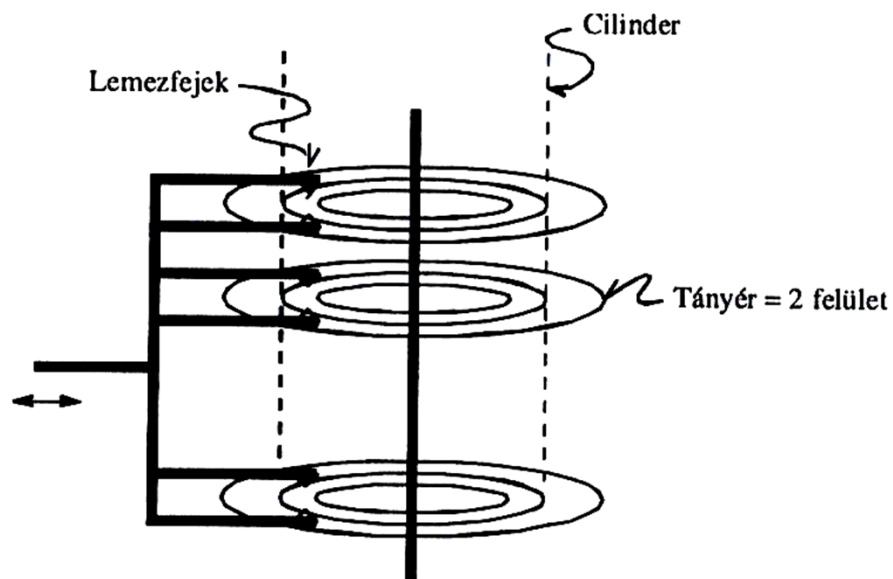
---

- I/O költség
  - Adatbázisokban meghatározó
  - Moore törvény nem igaz rá ☹
  - Speciális kezelési módok
- CPU használat
  - Komplex lekérdezések
  - Összetett számítások
- Memória használat
  - Cache hatás



# Blokk olvasás költségei

- Keresési idő: fej megfelelő cylinderre állása
- Rotációs késés: keresett blokk fej alá érkezése
- Adatátviteli idő





# I/O költség csökkentése

---

- Korai beolvasás
  - Nagyobb bufferek
  - Előre ismert a beolvasandó blokkok sorrendje
- Cylinder alapú szervezés
  - Egymást követő adatblokkok egy cylinderen
  - Rotációs késés csökkentése
- Több lemez használata (RAID)
  - Több fejszerelvény
  - Keresési idő csökkentése
- Lift algoritmus
  - Fejszerelvény folyamatosan „liftez” a cylindereken
  - Keresési idő csökkentése



# Tartalom

---

- **Lekérdezések feldolgozása**
  - I/O költség
  - **Logikai lekérdezési terv**
  - Fizikai végrehajtási terv
- Oracle Server lekérdezés optimalizációja
- MS SQL Server lekérdezés optimalizációja
- Néhány jó tanács



# Lekérdezés feldolgozás menete

## ➤ Elemző

- Lekérdezés fordítása
- Logikai terv készítése

## ➤ Optimalizáló

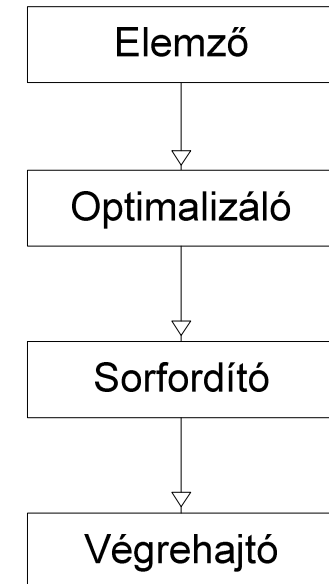
- Fizikai terv elkészítése
- Táblák bejárása
- Táblák összekapcsolása

## ➤ Sorfordító

- Fizikai terv leképezése I/O műveletekre

## ➤ Végrehajtó

- Műveletek végrehajtása





# Logikai végrehajtási terv elemei

---

- Elemző fa (átalakítás után)
  - Relációk (levél elemek)
  - Műveletek (csomópontok)
  - Adatok áramlása (csak lentől fölfelé)
- Relációs algebrai műveletek
  - Descartes-szorzat ( $R \times S$ )
  - Projekció ( $\pi_L(R)$ )
  - Szelekció ( $\sigma_F(R)$ )
  - Összekapcsolás ( $R \bowtie S$ )
  - Ismétlődések szűrése ( $\delta(R)$ )
  - Csoportosítás ( $\gamma_L(R)$ )
  - Rendezés ( $\tau_L(R)$ )

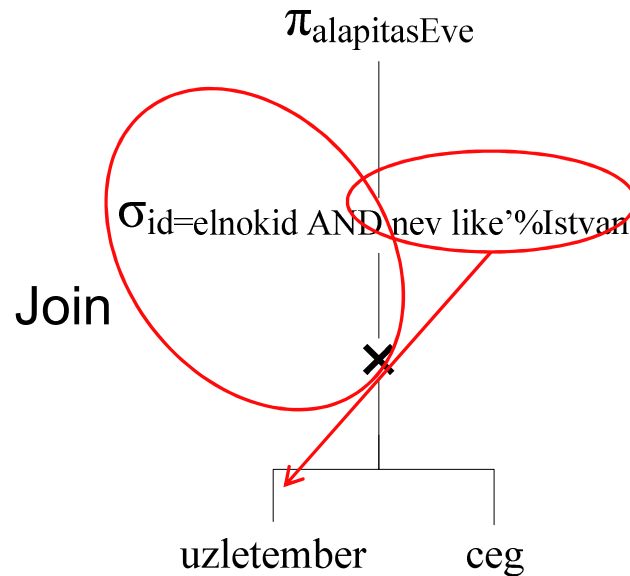


# Egyszerű lekérdezés elemző fája

select alapitasEve

from uzletember u, ceg c

where u.id = c.elnokid AND nev like '%István'





# Elemzőfa átalakítása – 1

---

- Optimális logikai terv kialakítása
- Fizikai végrehajtási terv keresési terének vágása
- Alapkonceptió
  - Kiválasztás műveletek lefelé mozgatása a fában
  - Vetítések felfelé mozgatása
  - Join operátorok használata
    - Direkt szorzat csak akkor ha a lekérdezés erre utasít
  - Join operátorok egyik attribútuma mindig tábla



# Elemzőfa átalakítása – 2

## ➤ Kiválasztás

- Felcserélési szabály:  $\sigma_{F1}(\sigma_{F2}(R)) = \sigma_{F2}(\sigma_{F1}(R))$
- Szétvágási szabály:
  - $\sigma_{F \text{ and } G}(R) = \sigma_F(\sigma_G(R))$
  - $\sigma_{F \text{ or } G}(R) = \sigma_F(R) \text{ UNION } \sigma_G(R)$  (Ha R-ben nincsenek ismétlődések)

## ➤ Összekapcsolás

- $R \bowtie_F S = \sigma_F(R \times S)$
- $R \bowtie S = S \bowtie R$   $O(n!)$  sorrend
- $(R \bowtie S) \bowtie U = R \bowtie (S \bowtie U).$



## Elemzőfa átalakítása – 3

### ➤ Összekapcsolás és kiválasztás

- $\sigma_F(R \bowtie S) = \sigma_F(R) \bowtie S$ , ha R-ben szerepel minden F-ben vizsgált attribútum;
- $\sigma_F(R \bowtie S) = R \bowtie \sigma_F(S)$ , ha S-ben szerepelnek az F-ben vizsgált attribútumok;
- $\sigma_F(R \bowtie S) = \sigma_F(R) \bowtie \sigma_F(S)$ , ha R-ben és S-ben is szerepelnek F attribútumai.

### ➤ Ismétlődések

- $\delta(\gamma_L(R)) = \gamma_L(R)$
- $\delta(R \times S) = \delta(R) \times \delta(S)$  (Join operátorokra is igaz)



# Tartalom

---

- **Lekérdezések feldolgozása**
  - I/O költség
  - Logikai lekérdezési terv
  - **Fizikai végrehajtási terv**
- Oracle Server lekérdezés optimalizációja
- MS SQL Server lekérdezés optimalizációja
- Néhány jó tanács



# Fizikai terv

---

## ➤ Fizikai terv elemei

- Relációt beolvasó operátorok
  - Logikai terv levél eleminek beolvasása
- Relációs algebrai műveletet végrehajtó operátor

## ➤ Tervek készítése

- Szabály alapú
- Költségbecslés alapú
  - Tábla elérési módok
  - Join operátorok megvalósítási módjai
  - Join sorrend



# I/O költség modellezése

---

## ➤ $B(R)$

- Az  $R$  reláció által elfoglalt blokkok száma

## ➤ $T(R)$

- $R$  reláció sorainak száma

## ➤ $V(R,a)$

- Az  $R$  reláció „ $a$ ” attribútumának variáltsága



# Tábla hozzáférési módok (I/O operátorok)

- Teljes átvizsgálás (Full table scan)
  - Nyalábolt esetben  $\sim O(B)$
  - Nem nyalábolt tárolás  $\sim O(R)$
- Index alapú átvizsgálás (Index scan)
  - $\sigma$  operátor megvalósítása
    - Egyenlőség esetén
    - Kisebb ill. Nagyobb operátor esetén
  - Rendezés megvalósítása



# Join operátorok megvalósítása

---

- Beágyazott ciklusok (Nested Loops Join)
- Hash alapú összekapcsolás (Hash Join)
- Rendezés összefésülés (Sort Merge Join)



# Nested loop join

---

- Egymásba ágyazott kettős for ciklus
- I/O költség
  - Nyalábolt esetben:  $O(B(R) \cdot B(S))$
  - Nem nyalábolt esetben:  $O(T(R) \cdot T(S))$
- Tetszőleges méretű táblákra működik
  - Nagy méret esetén: a két tábla egy- egy blokkját tartja memóriában
  - Kis méret: a táblát bent lehet tartani



# Hash join

---

## ➤ Első menetben

- Kisebb (R) reláció beolvasása
- Vödrös hash építése a memóriában
  - Kulcs a join operátorban szereplő oszlop

## ➤ Második menet

- A nagyobbik (S) reláció beolvasása
- Kapcsolódó rekordok keresése a vödrös hashben

## ➤ I/O költség

- $O(B(R)+S(R))$



# Sort Merge Join

---

- Mindkét relációt beolvassa a memóriába
- Rendezi az összekapcsolási kulcs szerint
- A két rendezett listát összefésüli
  - Listák közös bejárása
  - Egyszerű algoritmus
- Kis méretű relációk esetén
- Index a rendezés miatt
- I/O költség
  - $O(B(R)+S(R))$



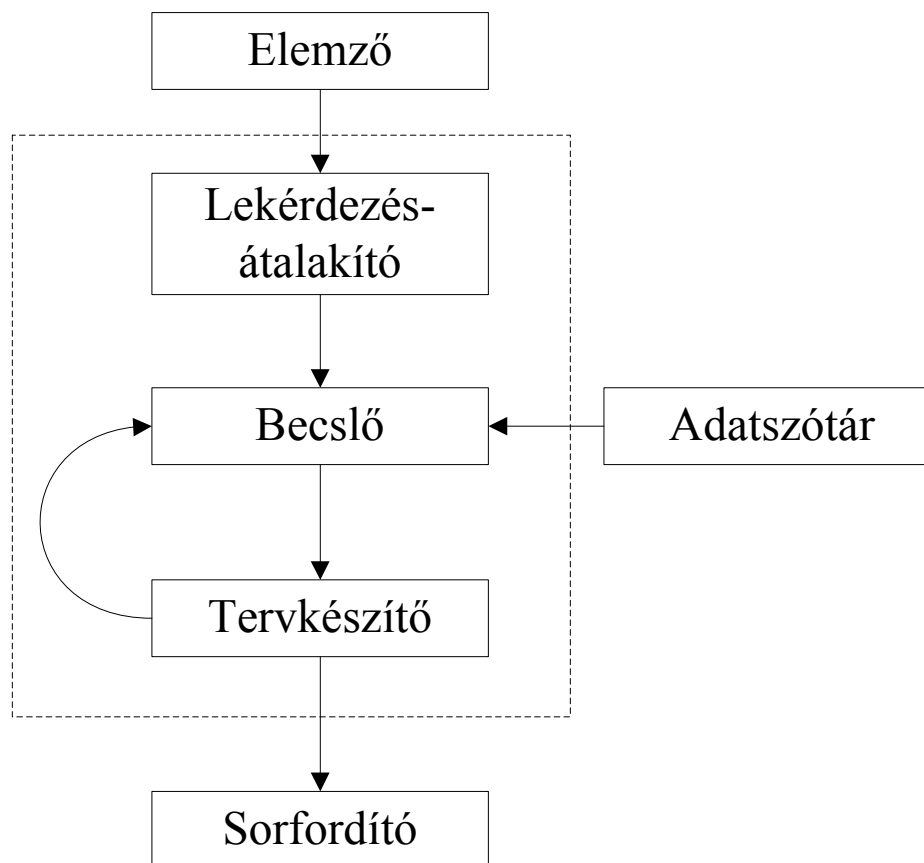
# Tartalom

---

- Lekérdezések feldolgozása
  - I/O költség
  - Logikai lekérdezési terv
  - Fizikai végrehajtási terv
- **Oracle Server lekérdezés optimalizációja**
- MS SQL Server lekérdezés optimalizációja
- Néhány jó tanács



# Oracle Server lekérdezés optimalizációja



**Optimalizáció menete**



# Lekérdezés átalakító – Logikai optimalizálás

- Nézetek befésülése
  - Befésülés allekérdezésként
  - Nézetek külön optimalizálása
    - Suboptimális tervhez vezet
- Feltételek lefelé tolása a kifejezés fában
- Allekérdezés felemelése
  - Join operátor preferálása
- Lekérdezés újraírása materializált nézetekre



# Becslő szerepe

---

- Fizikai tervhez költségbecslés készítése
- Becslés alapja
  - Default értékek használata
  - Dinamikus mintavételezés
  - Statisztikák
    - Adatszótárban tárolja
    - Hisztogramok az oszlop értékekre
    - Indexek telítettsége
    - Hiányában nagyon rossz terv születik



# Optimalizálás célja

---

## ➤ Optimalizációs módok

- Átbocsájtó képesség (ALL\_ROWS)
  - Reportok készítése
- Első n sor megtalálása (FIRST\_ROWS(n))
  - Interaktív felületek

## ➤ Beállítási lehetőségek

- Session szinten
  - OPTIMIZER\_MODE session paraméter
- Egyes lekérdezésre
  - SQL hint



# Tábla elérési módok – 1

---

## ➤ Full table scan

- Nincs index a táblán
- Várhatóan sok rekord kell a táblából
- Kicsi a tábla egyszerűbb beolvasni

## ➤ Rowid scan

- Sorazonosító alapú olvasás
- Index adja a sorazonosítót

## ➤ Unique index scan

- Elsődleges kulcs használata
- Ha az index összes oszlopára szűrünk
- Preferálja használatát

## ➤ Index range scan

- Index alapú szűrés
- =,<,>,like 'A%'
- Index szerint rendezett az eredményhalmaz



# Tábla elérési módok – 2

---

## ➤ Full index scan

- Index szerint a tábla teljes végigolvasása
- Kulcs szerint rendezett eredményhalmaz

## ➤ Fast full index scan

- Ha csak az indexben szereplő attribútumok kellenek
- A táblát nem is éri el

## ➤ Bitmap index scan

- Bitmap index alapú bool kifejezés kiértékelése



# Tábla összekapcsolási módok – 1

---

- Nested loop
  - Preferált kevés számú külső sor esetén
- Hash join
  - Equi join
  - Egyik reláció elfér a memóriában
- Sort merge join
  - Rendezett forrástáblák
  - Valamelyik későbbi művelet rendezett eredményt vár ill állít elő



# Tábla összekapcsolási módok – 2

---

## ➤ Index join

- Táblán lévő indexek hash alapú összekapcsolása
- Az összekapcsolt indexek tartalmazzák a szükséges attribútumokat
- Több index használata ugyanazon a táblán
  - Nélküle csak egy index/ tábla



# Optimalizáció befolyásolása

---

- Indexek definiálásával
- SQL hint
- Lekérdezés átírásával
  - → Lásd jó tanácsok rész



# Oracle Server indexei

---

- B\* fa alapú indexek
  - Egyszerű
  - Összetett
    - Hierarchikus
  - Függvény alapú
- Index oragized table
  - Blokkok sorrendje index szerint
- Bitmap index



# Bitmap index működése – 1

| CUSTOMER # | MARITAL_ STATUS | REGION  | GENDER | INCOME_ LEVEL |
|------------|-----------------|---------|--------|---------------|
| 101        | single          | east    | male   | bracket_1     |
| 102        | married         | central | female | bracket_4     |
| 103        | married         | west    | female | bracket_2     |
| 104        | divorced        | west    | male   | bracket_4     |
| 105        | single          | central | female | bracket_2     |
| 106        | married         | central | female | bracket_3     |

| REGION='east' | REGION='central' | REGION='west' |
|---------------|------------------|---------------|
| 1             | 0                | 0             |
| 0             | 1                | 0             |
| 0             | 0                | 1             |
| 0             | 0                | 1             |
| 0             | 1                | 0             |
| 0             | 1                | 0             |



# Bitmap index működése – 2

```
SELECT COUNT(*)
FROM CUSTOMER
WHERE MARITAL_STATUS = 'married'
AND REGION IN ('central','west');
```

| status =<br>'married' |     | region =<br>'central' |    | region =<br>'west' |   |
|-----------------------|-----|-----------------------|----|--------------------|---|
| 0                     |     | 0                     |    | 0                  |   |
| 1                     |     | 1                     |    | 0                  |   |
| 1                     | AND | 0                     | OR | 1                  | = |
| 0                     |     | 0                     |    | 1                  |   |
| 0                     |     | 1                     |    | 0                  |   |
| 1                     |     | 1                     |    | 0                  |   |



# SQL Hint

- Optimalizáló befolyásolása
- Első SQL kulcsszó után kell megadni
- Nagymértékben befolyásolja a keresési tér bejárását
- Nem kötelező az optimalizáló számára
- Statisztikák változását nem követi
  - Hard coded megoldások
- Tipikus hintek
  - Preferált index használata
  - Index használatának tiltása
  - Join típus megadás
  - Join sorrend megadása
  - Táblaelérési mód megadása
  - Párhuzamos végrehajtás
  - Lekérdezés átírása
    - OR → Lekérdezések + Union all
    - Materializált nézetek használata



# Tartalom

---

- Lekérdezések feldolgozása
  - I/O költség
  - Logikai lekérdezési terv
  - Fizikai végrehajtási terv
- Oracle Server lekérdezés optimalizációja
- **MS SQL Server lekérdezés optimalizációja**
- Néhány jó tanács



# Optimalizálás alapelvek

---

- Statisztikák alapján értékel
- Költség = Válaszidő
- Triviális terv
  - Egyszerűbb lekérdezéshez egyértelműen generálható
  - „Szabály alapú”
- Nem készíthető triviális terv
  - Összetett lekérdezések
  - 3 fázisú optimalizáló



# Háromfázisú optimalizáció

---

## ➤ 0. Fázis

- Egyszerű átalakítások
- Preferált hash join
- Ha a költség  $< 0,2 \rightarrow$  végrehajtás

## ➤ 1. Fázis

- Kibővített átalakítások
- Ha a költség  $< 1 \rightarrow$  végrehajtás

## ➤ 2. Fázis

- Párhuzamos végrehajtás vizsgálata



# Tábla-elérési módok – 1

---

## ➤ Table scan

- Ha nincs semmilyen index
- Táblára vonatkozó szűrési feltételt is kiértékeli

## ➤ Clustered index scan

- Nyalábolt adatolvasás
- Adatblokkok index szerint rendezve
- Clustered index primary key mentén létrejön
- Table scan helyett ezt preferálja



# Tábla-elérési módok – 2

---

- Nonclustered index scan
  - Mint a clustered index scan
  - Alapvetően = operátor kiértékelésére
- Clustered/ Nonclustered index seek
  - Hasonló az index scan-hez
  - B\* fa leveleinek bejárása egy kezdőelemtől
    - >, between, < operátor kiértékelése



# Táblák összekapcsolása

---

- Nested loops
- Hash Match
- Merge Join



# Optimalizáció befolyásolása

---

- Indexek definiálásával
- SQL hint
  - Funkciójában hasonló az Oracle Serverhez
  - Eltérő szintaktika
- Lekérdezés átírásával
  - → Lásd jó tanácsok rész



# MS SQL Server indexei

---

## ➤ B\* fa alapú indexek

- Egyszerű
- Összetett
  - Hierarchikus
- Clustered
  - Adatblokkok sorrendje index szerint
  - Egy táblán egy lehet
  - Definiált primary key mentén automatikusan létrejön



# Indexek sajátosságai – 1

---

- Cover index (included column)
  - B\* fa levelének bővítése oszlopokkal
  - Nem kell kiolvasni a tényleges rekordot
- Clustered és non clustered indexek együttes használata
  - Non clustered index levél eleme
    - Nem fizikai címet tartalmaz
    - Kulcs érték a clustered indexre
  - Dupla index olvasás



# Indexek sajátosságai – 2

---

- Indexelt nézetek
  - Nézet eredményének tárolása
  - Index is rendelhető hozzá



# Tárolt eljárások sajátosságai – 1

---

## ➤ Plan cache

- Végrehajtási terv cache
- Ha ugyan olyan struktúrájú lekérdezés jött
- Statisztikák nem változtak



# Tárolt eljárások sajátosságai – 2

---

**create proc test**

**@db int as**

**select \***

**from Termek**

**where Raktarkeszlet<@db**

**create proc test**

**@db int as**

**select \***

**from Termek**

**where Raktarkeszlet<@db**

**option (optimize for (@db=2))**

**exec test 20**

**exec test 2 with recompile**



# Tartalom

---

- Lekérdezések feldolgozása
  - I/O költség
  - Logikai lekérdezési terv
  - Fizikai végrehajtási terv
- Oracle Server lekérdezés optimalizációja
- MS SQL Server lekérdezés optimalizációja
- **Néhány jó tanács**



# Jó tanácsok – 1

---

- Statisztikáink legyenek naprakészek
  - Elavult statisztika → rossz végrehajtási terv
- Lekérdezés struktúrája
  - SQL deklaratív környezet
    - Gondolkodjunk procedurálisan is!
  - Többféleképp is megfogalmazható ugyanaz
  - Törekedjünk az egyszerűsége
  - Kerüljük a select \* -ot
  - „Jó struktúrával” sokat lehet nyerni
    - Csak ezután kísérletezzünk a hintek használatával



## Jó tanácsok – 2

---

- Jobb ma egy join (még ha outer is), mint
  - In / Not in
  - Exists / Not exists
- Exists helyett inkább in
  - Nem korrelatív
- Nézetek
  - Ha lehet kerüljük
  - Főleg ne kapcsoljuk egymáshoz
- Kerüljük a vagy feltételeket → Union all
- Union helyett Union all (ha lehet)



## Jó tanácsok – 3

```
select *  
from szamla sz  
where not exists  
(  
    select 1  
    from SzamlaTetel szt  
    where sz.id=szt.SzamlaID  
)
```

```
select sz.*  
from szamla sz  
where sz.id not in  
(  
    select SzamlaID  
    from SzamlaTetel  
)
```

```
select sz.*  
from szamla sz left outer join szamlatetel szt  
on sz.id=szt.SzamlaID  
where szt.id is null
```

Egyszerű esetekben ma már mindegy, de általában nem



# Jó tanácsok – 4

---

## ➤ Indexek használata

- Egy táblán egy lekérdezésben általában csak egyet tud használni → Join művelet el is használhatja
- Összetett index
  - Hierarchia számít
- Kulcs bármilyen kifejezésben szerepel akkor nem tudja használni az optimalizáló
  - Akár: kulcs+0 ← Ezt már észreveszik ☹



# Jó tanácsok – 5

---

## ➤ Függvények használata

### ■ Select listán nyugodtan

- Nem befolyásolja a végrehajtási tervet

### ■ Where feltételben lehetőleg ne használjuk

- Minden rekordra le kell futtatni
- Nehezen mozgatható a kifejezés fában
- Kimenetére nem készül statisztika → nehéz optimalizálni



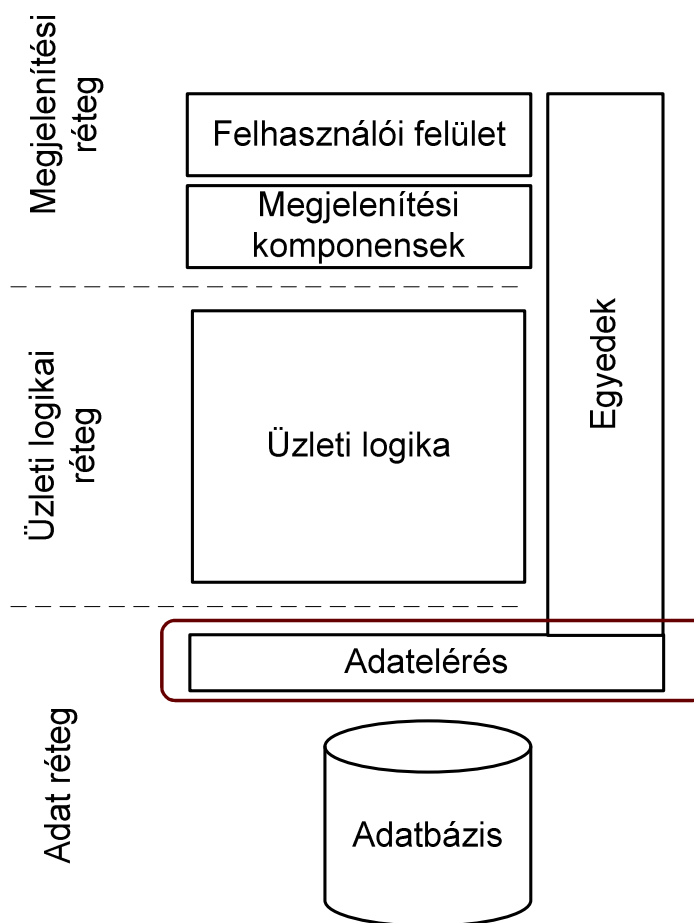
# Adatelérési osztálykönyvtárak

---

## ADO.NET

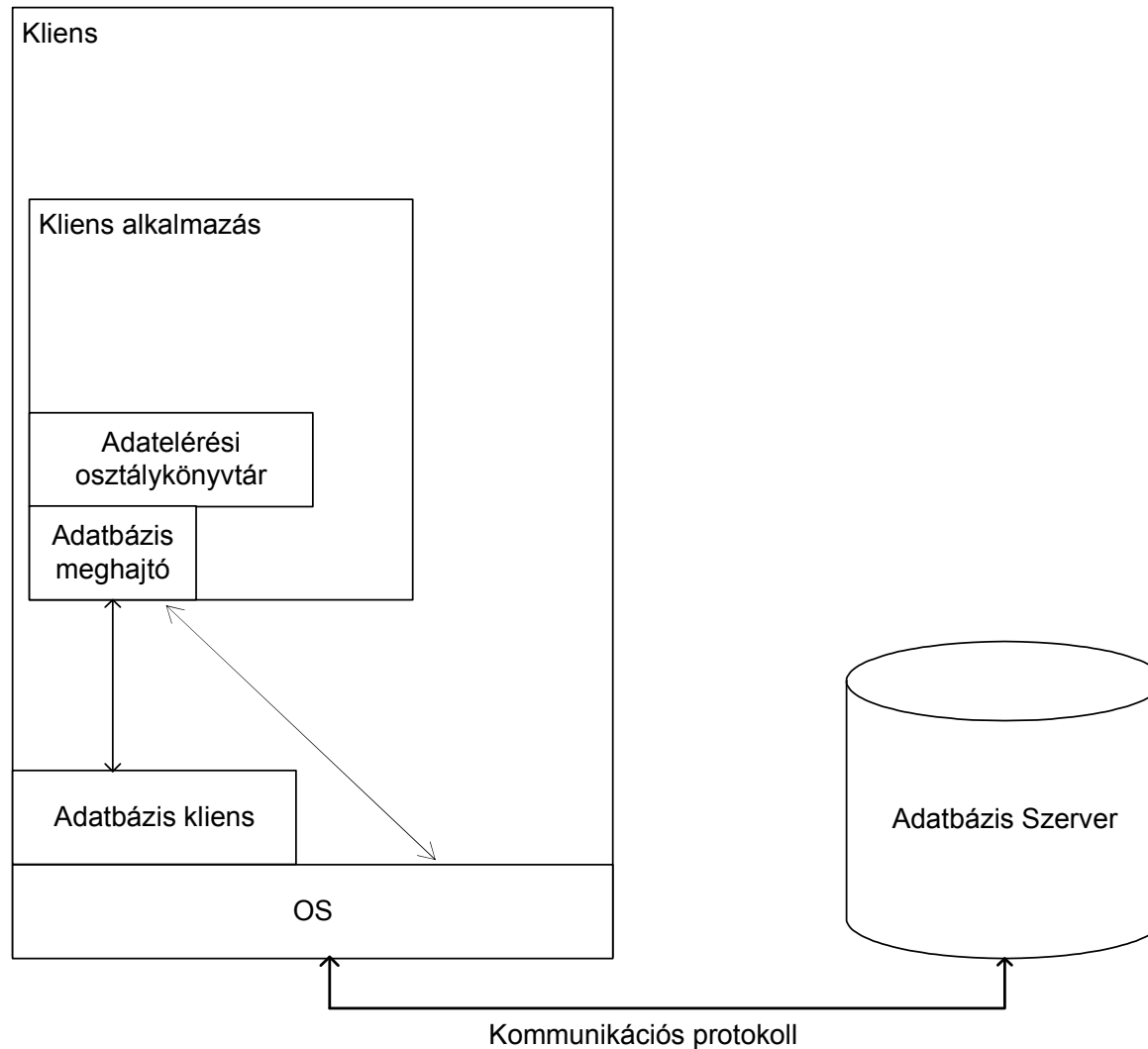


# Adatelérési osztálykönyvtárak szerepe





# Adatelérési osztálykönyvtárak működése





# Adatelérési könyvtár feladata

---

- Adatbázis használat absztrakció
  - Relációs adatbázisok elérése
  - Adatbázis független
  - Egységes, adatbázis független kódolás
- Tipikus elmei
  - Connection
    - Adatbázis meghajtók
  - Command
    - Paraméterek
  - ResultSet
  - Exceptions



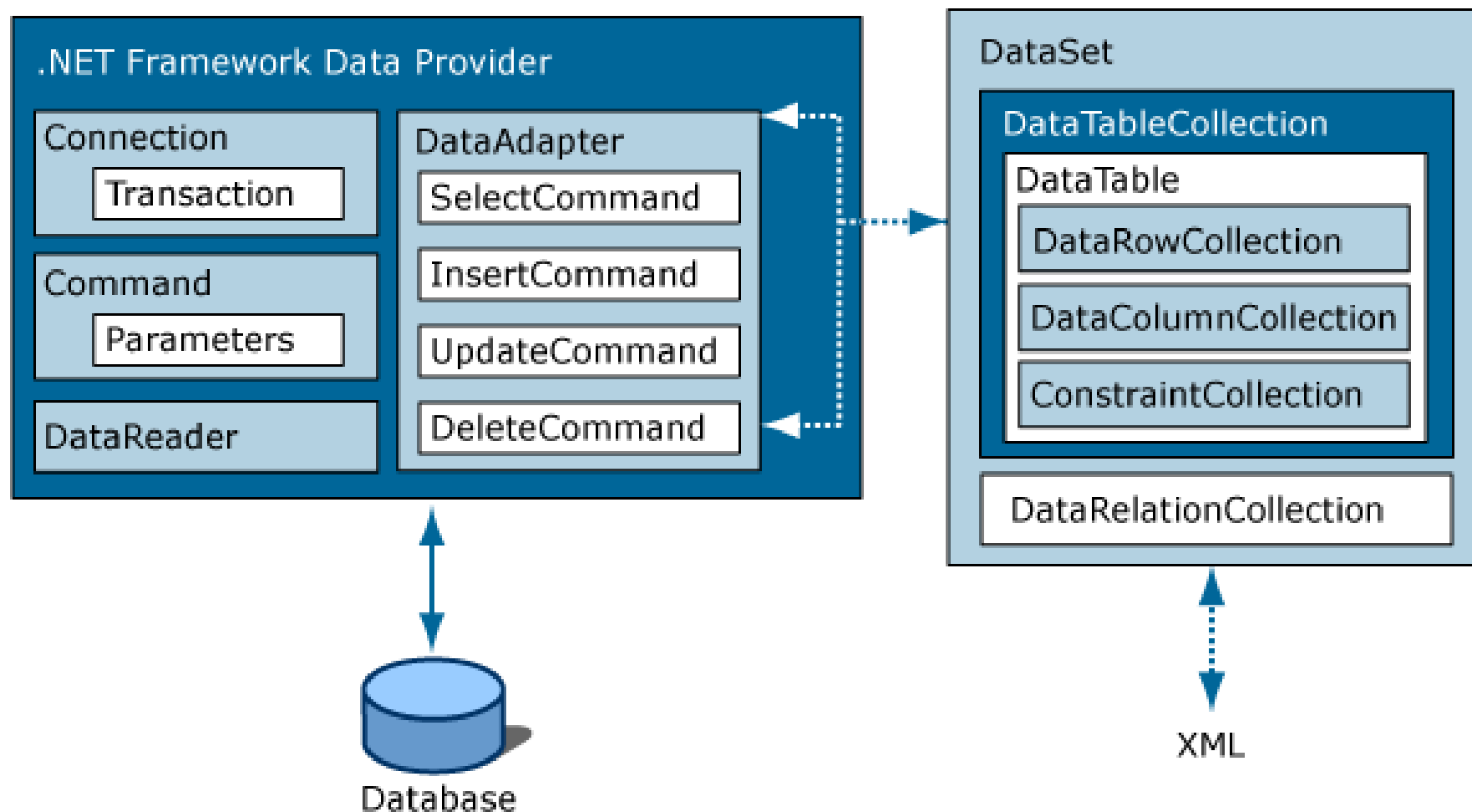
# ADO.NET alap koncepció

---

- Egységes, adatbázis független kódolás
  - Adatbázis interfész független
  - Interfészek és absztrakt osztályok
- Implementációk
  - Megvalósítják az alap funkciókat
  - Ki is terjeszthetik az alap funkciókat
  - A .NET framework részeként
    - OleDb
    - Microsoft SQL Server
  - Egyéb implementációk a Microsofttól
    - ODBC
    - Oracle
  - Más cégek implementációi
    - Pl: MySQL



# Alap objektumok





# DataReader vs DataSet

---

## **DataReader**

- Egyirányú, csak olvasható
- Ha az adatokat azonnal feldolgozzuk
- Csak egy lehet megnyitva

## **DataSet**

- Lokális adat cache
- Ha az adatok közt navigálunk
- Ha az adatokat módosítjuk is



# Kapcsolati modellek

---

## Kapcsolat alapú

- Folyamatos kapcsolat az adatbázis szerverrel
- Előnyök
  - Egyszerűbb a konkurencia kezelése
  - Az adatok mindenhol a legfrissebbek
- Hátrányok
  - Folyamatos hálózati kapcsolat
  - Skálázhatóság

## Kapcsolat nélküli

- Kapcsolat csak az adatmanipuláció idejére
- Előnyök
  - Nem szükséges folyamatos hálózati kapcsolat
  - Skálázhatóság
- Hátrányok
  - Az adatok nem mindig a legfrissebbek
  - Ütközések lehetségesek



# Adatbázis kapcsolat felépítése

---

## ➤ IDbConnection interfész

- Open: Megnyitás
- Close: Lezárás
- BeginTransaction: Tranzakció indítás

## ➤ Connection pooling

- Az OleDb, MS SQL implementációnál is van
- Minél később megnyitni a kapcsolatot
- Minél előbb lezárni



# Connection String összeállítása

---

## ➤ DB szervertől függ a szintaktika

```
static string _connectionStringValue = "User  
ID=LOGIN;Password=PASSWORD;Persist Security Info=false;Initial  
Catalog=AdventureWorks;Data Source=DATASOURCE;Packet Size=4096";
```

## ➤ <http://www.connectionstrings.com/> 😊

## ➤ Connection string alapú támadások

- All input is evil
- Connection String Builder
  - DB platformonként
  - Paraméterek használata



# Adatbázis parancs használata

---

## ➤ IDbCommand interfész

- 3 különböző típus (CommandType)
  - Tárolt eljárás
  - Tábla teljes tartalma
  - SQL query
- A parancs szövege (CommandText)
- Az adatbázis kapcsolat (Connection)
- A tranzakció (Transaction)
- Paraméterek
  - All input is evil



# Parancs végrehajtása

---

- ExecuteReader
  - Több rekord lekérdezése (kurzor)
- ExecuteScalar
  - Skalár érték lekérdezése
- ExecuteNonQuery
  - Nincs visszatérési érték (Pl: INSERT)
- ExecuteXmlReader (SQL szerver)
  - XML dokumentumot ad vissza



# Parancs használata – Példa

---

```
SqlCommand command = new SqlCommand();  
command.Connection = connection;  
command.CommandText = "SalesByCategory";  
command.CommandType = CommandType.StoredProcedure;
```

```
SqlParameter parameter = new SqlParameter();  
parameter.ParameterName = "@CategoryName";  
parameter.SqlDbType = SqlDbType.NVarChar;  
parameter.Direction = ParameterDirection.Input;  
parameter.Value = categoryName;
```

```
command.Parameters.Add(parameter);  
connection.Open();
```

```
SqlDataReader reader = command.ExecuteReader();
```



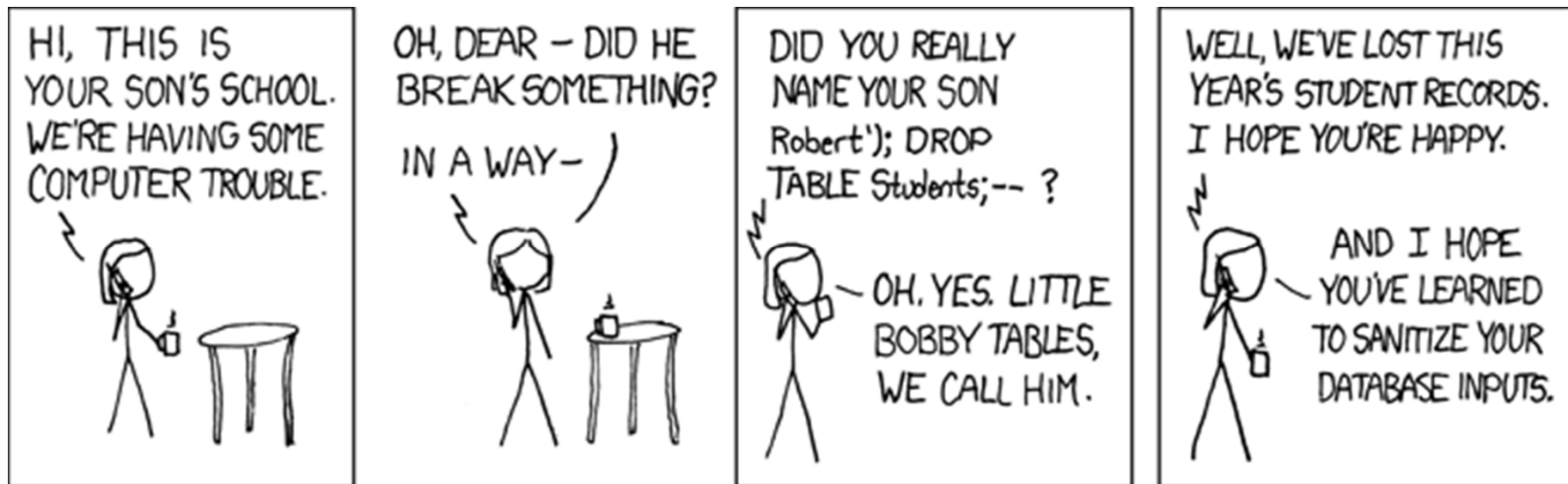
# SQL Injection – 1

---

- Biztonsági probléma
  - Input adatok ellenőrizetlen felhasználása
- SQL utasítás string jellegű összefűzése
  - “Select \* from aru where ar =”+Ar.Text;
  - Ar.Text-ben bármi szerepelhet
    - 1; drop table szamla;--
- Súlyos hiba!
  - Paraméterek használata



# SQL Injection – 2





# Demó

---

## SQL Injection



# Tranzakciók ADO.NET alatt

---

- Tranzakció létrehozása
  - BeginTransaction metódus
- Parancsok tranzakcióhoz rendelése
  - Transaction tulajdonság
- Parancsok futtatása
- Tranzakció befejezése
  - CommitTransaction
  - RollbackTransaction
- Izolációs szintek
  - A BeginTransaction paramétere
  - Szerver függő izolációs szintek



# Hibakezelés ADO.NET alatt

- Fontos hibák kivételeket okoznak
- A reader-t, és a kapcsolatot le kell zárni!
  - Megoldás kivétel kezeléssel
  - Lezárások a finally blokkban
- Dispose tervezési minta
  - A using kulcsszó használata létrehozáskor
  - A blokk végén lezáródik a kapcsolat/reader

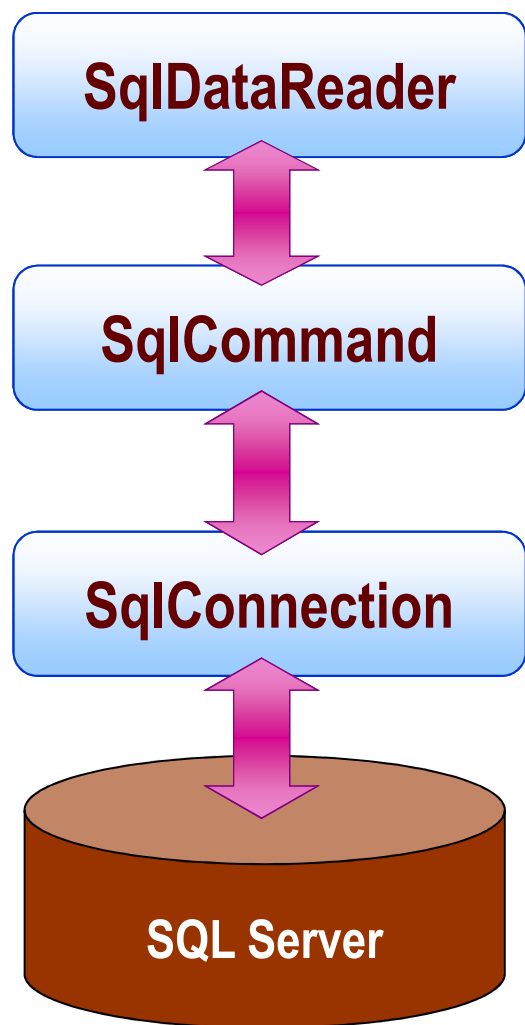
```

try
{
    SqlConnection connection =
        new SqlConnection(connectionString)

    connection.Open();
    ...
}
finally
{
    if (connection != null)
        connection.Dispose();
}
-----
using (SqlConnection connection = new
    SqlConnection(connectionString))
{
    connection.Open();
    ...
}
    
```



# Kapcsolat alapú modell



- Aktuális adatok érkeznek az adatbázisból
- Feldolgozás lépései
  1. Kapcsolat megnyitása
  2. Parancs futtatása
  3. Eredmény feldolgozása
  4. Reader lezárása
  5. Kapcsolat lezárása



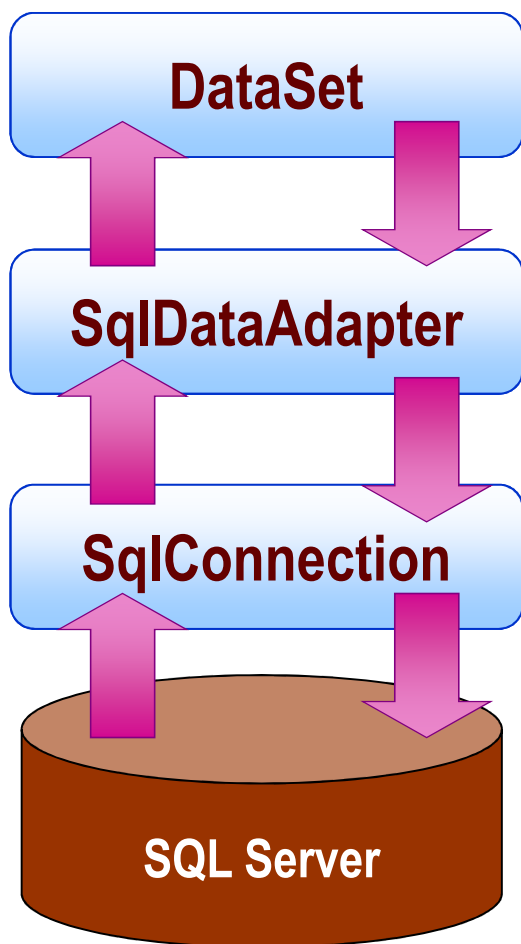
# Kapcsolat alapú modell – Példa

---

```
using (SqlConnection conn=new SqlConnection(connectionString))
{
    SqlCommand command = new SqlCommand("SELECT * FROM Termek", conn);
    connection.Open();
    using(SqlDataReader reader = command.ExecuteReader())
    {
        if (reader.HasRows)
        {
            while (reader.Read())
            {
                Console.WriteLine("{0}\t{1}", reader["ID"], reader["Nev"]);
            }
        }
    }
}
```



# Kapcsolat nélküli modell



- Cache-nek tekinthető
- Más is módosíthat
- Feldolgozás lépsei
  1. A kapcsolat megnyitása
  2. A DataSet feltöltése
  3. A kapcsolat lezárása
  4. A DataSet feldolgozása
  5. A kapcsolat megnyitása
  6. Változtatások visszatöltése
  7. A kapcsolat lezárása



# Kapcsolat nélküli modell – Példa

---

```
DataSet dataSet = new DataSet();
SqlDataAdapter adapter = new SqlDataAdapter();

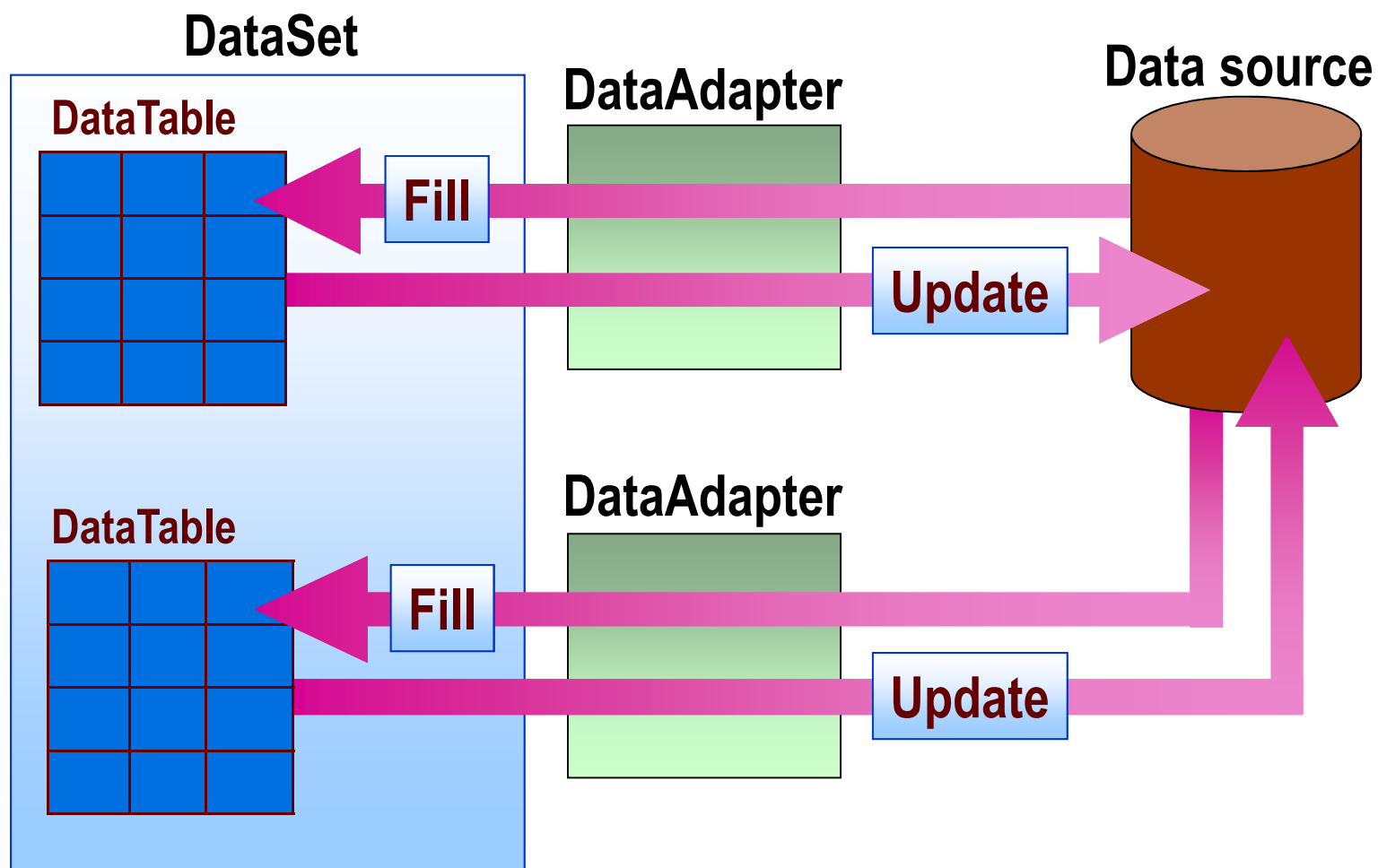
using (SqlConnection conn= new SqlConnection(connectionString))
{
    adapter.SelectCommand = new SqlCommand ("SELECT * FROM Termek",conn);

    SqlCommandBuilder builder = new SqlCommandBuilder(adapter);

    connection.Open();
    adapter.Fill(dataSet);
}
...
using (SqlConnection conn= new SqlConnection(connectionString))
{
    dataSet.AcceptChanges(); // SqlCommandBuilder nélkül nem működne
    //adapter.Update(dataSet) is hívható, de akkor csak az adapter táblája frissül
}
```



# DataAdapter szerepe





# DataAdapter feladata

---

- A DataSetet illeszti az adatbázishoz
- Feltöltés
  - SelectCommand tulajdonság
  - Fill metódus
- Szinkronizálás
  - InsertCommand tulajdonság
  - UpdateCommand tulajdonság
  - DeleteCommand Tulajdonság
  - Update metódus



# DataSet felépítése

## ➤ System.Data névtér (adatbázis független)

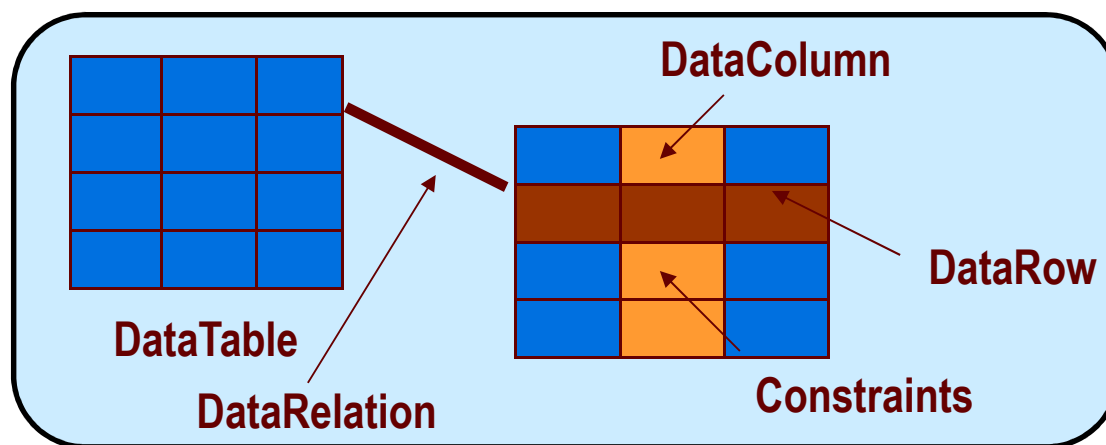
- DataRow
- DataColumn
- DataTable
- DataView
- DataAdapter
- DataSet

## ➤ Gyűjtemények

- Tables
- Relations

## ➤ Séma

- XSD alapján
- Kódból





# DataSet egyéb lehetőségei

## ➤ Kényszerek

- Egyediség biztosítása (UniqueConstraint)
- Elsődleges kulcs
- Külső kulcs (ForeignKeyConstraint)

## ➤ Kapcsolatok (DataRelation)

- Közös értékű mező alapján
- Elsődleges kulcs – Külső kulcs
- Szülő gyermek kapcsolatok

## ➤ Számított értékű mezők

- Felhasználhatóak
  - Más mezők értékei
  - Konstansok
  - Matematikai műveletek
- DataColumn osztály
  - Expression tulajdonság



# Típusos DataSet

---

- Definiált struktúrájú DataSet
- Struktúra leírása: XML séma
- Generált osztályok
  - Az előzőekben bemutatott általános osztályokból származnak
  - Entitásra jellemző tulajdonságok, metódusok
- Előnyei
  - Gyorsabb fejlesztés
  - Típusosság fordítási időben



# Konkurencia kezelés

---

- Probléma: többen módosíthatják az adatokat
- Pesszimista konkurencia kezelés
  - Biztos más is módosít
  - Kizárólagos hozzáférés kell a módosított adatokhoz
  - Tipikusan kapcsolat alapú modellben
- Optimista
  - Más ügysem módosít
  - Ütközés detektálás
  - Tipikusan kapcsolat nélküli modellben



# Pesszimista konkurencia kezelés

---

## ➤ Kapcsolat alapú modell

- Rekordok zárolása
- Adatbázis tranzakciók biztosítják

## ➤ Kapcsolat nélküli esetben

- Tipikusan BLL-ben merül fel a probléma
- Zártáblák megvalósítása a BLL-ben
  - Kis lockmanager osztály, bejegyzi, hogy melyik ID-t módosítják
  - A nyilvántartáshoz elég egy Hashtable
    - Kulcs: tábla + ID



# Optimista konkurencia kezelés

---

- Módosított rekordokat ellenőrizni kell
- Rekord tartalom alapján döntünk
  - Rekord verzió
    - Számláló (logikai óra)
  - Teljes tartalomvizsgálat
- DataAdapteren állítható be
  - Különböző SQL kódot generál



# DataSet állapot követés

---

- Egy adott sor többféle állapotban lehet
- DataRowState
  - Unchanged: változatlan
  - Added: újonnan lett létrehozva
  - Deleted: törölődött
  - Modified: módosult
  - Detached: a sor nem része egy DataRowCollection-nek
- Ezt az információt használja fel DataAdapter a módosítások átvezetéséhez az adatbázisba



# DataSet – Rekord verzió

---

- Az egyes soroknak több verziója létezhet
- DataRowVersion
  - Original: a kiindulási érték
  - Current: aktuális érték
  - Default: DataRowState-től függ
  - Proposed: szerkesztés alatt álló érték
- Lekérdezés
  - `row.HasVersion(DataRowVersion.Proposed)`
  - `row["Nev", DataRowVersion.Original]`



# DataSet – Adatok frissítése

---

## ➤ FEGURM probléma

- Fill / Edit / Get Changes / Update / Refresh / Merge

## ➤ DataTable, DataSet és DataAdapter szinten támogatott

- GetChangeSet()

- Load()

- FillOption

- OverwriteChanges (mint a Fill)

- Az „Original” és a „Current” verziót módosítja

- PreserveChanges (alapértelmezett):

- Az „Original” verziót módosítja

- Upsert

- A „Current” verziót módosítja

- Merge()



# ADO.NET optimalizálás – 1

---

## ➤ Connection Pooling

- ADO.NET része, érdemes kihasználni
- Min Pool Size ➔ ConnectionString-enként

## ➤ Commit-ok kezelése

- AutoCommit / Tranzakciók?
- A commit időigényes művelet (disk I/O + hálózat)
- Hosszú tranzakciók erőforrás igényesek (zárolások)
- Megfelelő egyensúlyt kell tartani a kettő között



# ADO.NET optimalizálás – 2

---

- Megfelelő tranzakciós modell választása
  - Kerüljük az elosztott tranzakciókat!
- Megfelelő parancstípus választása
  - INSERT / UPDATE / DELETE műveletek gyorsabbak, ha ExecuteNonQuery-vel futtatjuk őket
  - Egy adatmező lekéréséhez használjuk az ExecuteScalar függvényt
- DataProvider választás
  - A 100%-osan .Net alapú az ideális
    - Egyre több ilyen van
  - Kevert megoldások (ODBC kerülendő)



# ADO.NET optimalizálás – 3

---

## ➤ Parancs újrafelhasználás

### ■ Command.Prepare()

- Szerver oldalon előkészíti a futtatást
- A későbbi végrehajtás gyorsabb lesz, de cserébe idő- és erőforrásigényes
- Csak akkor van haszna, ha többször futtatjuk a parancsot

## ➤ DataSet / DataReader

- DataReader: hatékonyabb, célirányosabb, szerveroldali erőforrásokat foglal, kis memóriaigény
- DataSet: kapcsolat nélküli munkához, többretegű architektúrához, nagyon nagy kliens oldali overhead, elsősorban memóriában

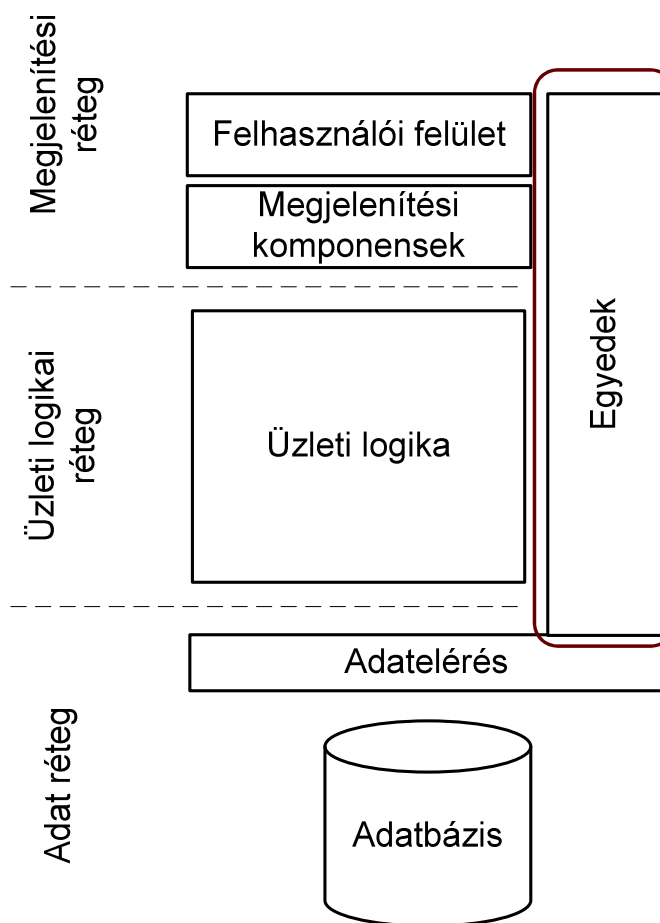


# O/R leképzés

---



# Háromrétegű architektúra





# Modellezés

---

## ➤ Üzleti logika

- OO modellezés
- UML
- Tervezési minták (Design Patterns)
- Nem csak statikus tagok, folyamatok is

## ➤ Adatréteg

- E/K diagramok
- UML data modelling profile (→2005 december)
- Statikus, attribútum szemlélet



# O/R leképezés feladata

---

## ➤ Object Relational Mapping

- Üzleti objektumok leképezése relációs adatmodellre
- Adattárolás és üzleti folyamatok összekötése

## ➤ Problémák

- Eltérő koncepciók
- Öröklődés
- Shadow információk
- ...



# Irodalom

---

➤ Felhasznált irodalom, ábrák, példák:

- <http://www.agiledata.org/essays/mappingObjects.html>



# Miért probléma?

---

## ➤ Egyszerű megoldás

- Objektum → Tábla (?)
- Adattag → Oszlop(?)

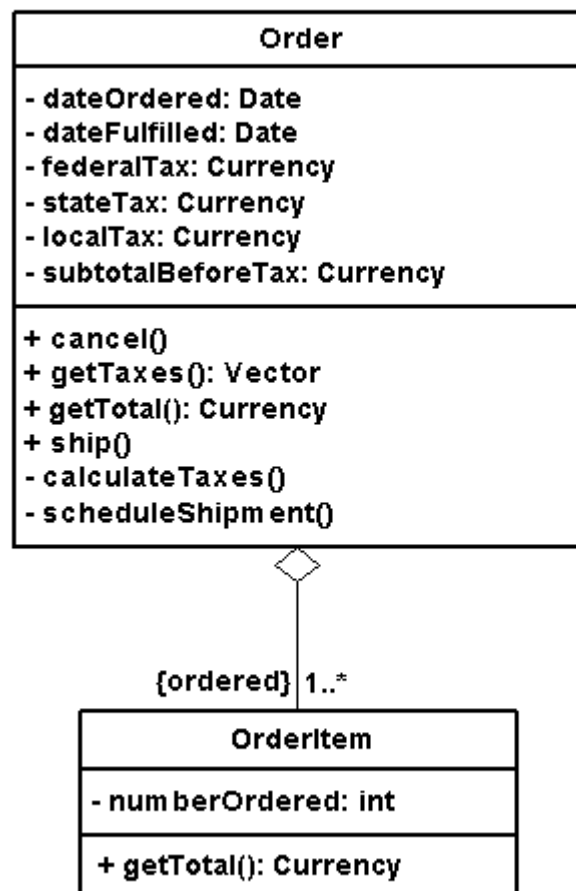
## ➤ Probléma

- Nem minden attribútum perzisztens
  - Ideiglenes kalkulálódik
- Vannak attribútumok, melyek önmagukban objektumok
- Adattípusok leképezése
- Öröklés kezelése



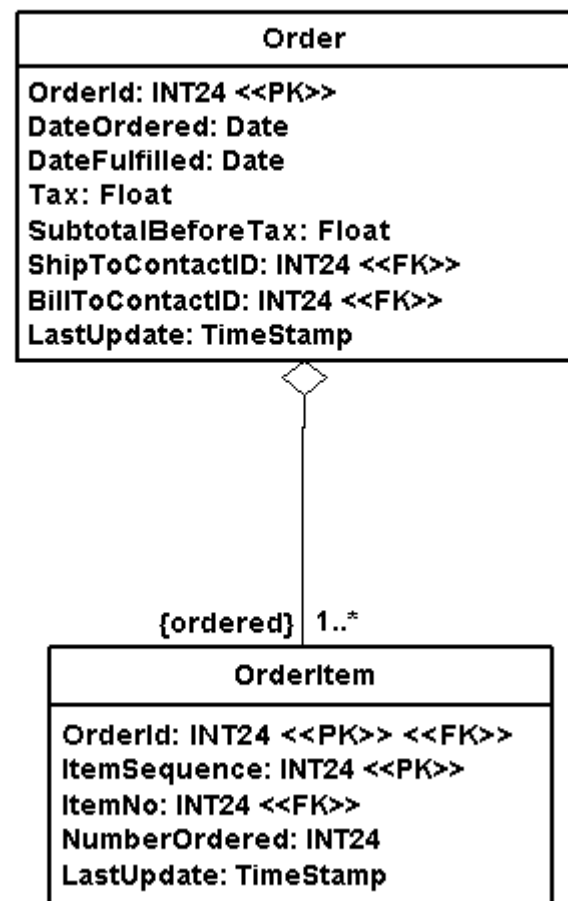
# Egyszerű mintapélda

<<Class Model>>



Copyright 2002-2006 Scott W. Ambler

<<Physical Data Model>>





# Egyszerű mintapélda – Problémák

---

## ➤ Tax mező

- Szét kellene szedni három oszlopba
- Ki kell/ lehet számolni

## ➤ Relációs modellben vannak kulcsok

- Hogy kapcsolódik az objektum példányhoz?
  - Shadow information

## ➤ Eltérő adattípusok

- Currency → Float leképezés az áraknál
- Konverzió?



# Shadow információk

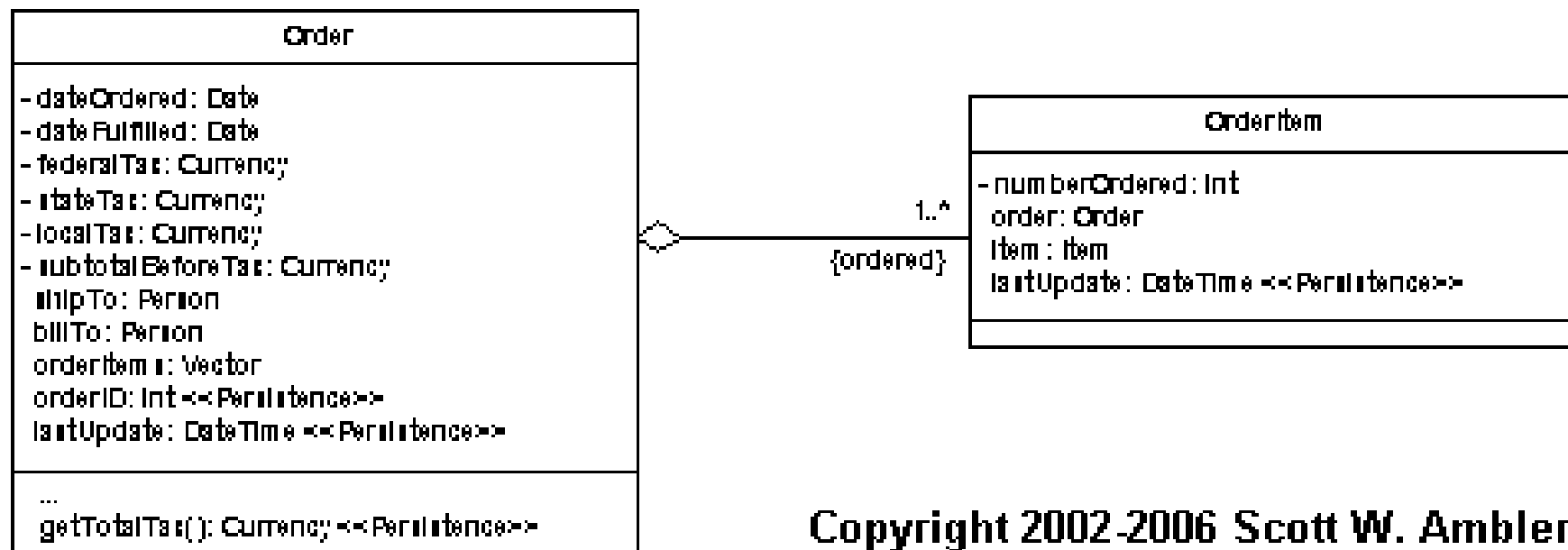
---

## ➤ Szükségesek a perzisztencia megvalósításához

- Kulcsok
- Időbélyegek (optimista konkurencia kezelés)
- Új/ már korábban mentett objektum
  - Insert/ Update
- Nem fontos az üzleti objektumba helyezni
  - Kezelni kell valahogy



# Egyszerű mintapélda – Shadow információk



Copyright 2002-2006 Scott W. Ambler



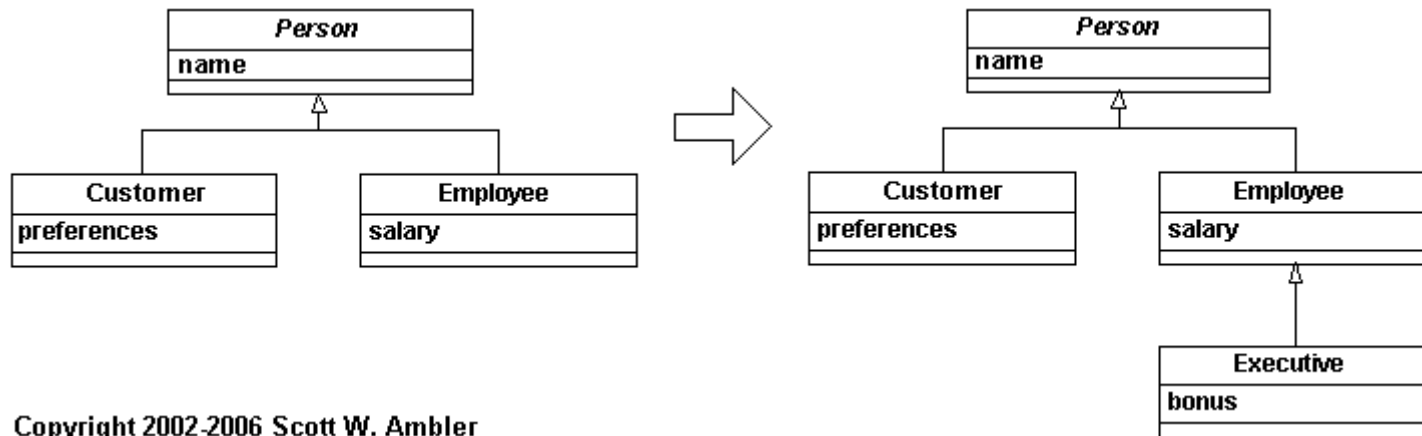
# Öröklődés modellezése

---

- Hierarchia leképezése egy közös táblába
- Minden valós osztály leképezése saját táblába
  - Akikből objektum példányok képződhetnek
- Minden osztály leképezése saját táblába
  - Absztrakt osztályok is
- Osztályok és hierarchia szintek általános leképezése



# Öröklődés – Mintapélda



Copyright 2002-2006 Scott W. Ambler

- Absztrakt Person osztály
- CR miatt új osztályt kell implementálni
  - Executive



# Egy táblába történő leképezés – 1

---

- Összes attribútum felsorolása a hierarchiát bejárva
- Típust azonosítás
  - Egy, oszlopban kódolt értékkel
  - IsCustomer, IsEmployee,... oszlopokkal
- Példány azonosító oszlop felvitele
- Bővítés kezelése
  - Új attribútumok felvitele



# Egy táblába történő leképezés – 2

| Person            |
|-------------------|
| PersonPOID <<PK>> |
| PersonType        |
| Name              |
| Preferences       |
| Salary            |

Copyright 2002-2006 Scott W. Ambler

| Person            |
|-------------------|
| PersonPOID <<PK>> |
| IsCustomer        |
| IsEmployee        |
| Name              |
| Preferences       |
| Salary            |

Copyright 2002-2006 Scott W. Ambler



# Egy táblába történő leképezés – 3

---

## ➤ Előnyök

- Egyszerű
- Könnyű új osztályt bevenni a hierarchiába
- Objektum példány szerepének változása könnyen követhető
  - Employee → Executive
  - Employee és Customer egyszerre

## ➤ Hátrányok

- Helypazarlás
- Egy osztály változása miatt az összes tárolása megváltozik
- Komplex struktúra esetén nehezen áttekinthető

## ➤ Célszerű használni

- Egyszerű hierarchiák esetén



# Valós osztályok leképzése táblába – 1

---

- Osztályonként egy tábla
- Osztály összes attribútumának eltárolása
- Példányazonosító
- Változás követése
  - Új osztály → új tábla
  - Attribútum változás → Hierarchia szint mentén végig kell vinni



# Valós osztályok leképezése táblába – 2

---

| Customer            |
|---------------------|
| CustomerPOID <<PK>> |
| Name                |
| Preferences         |

| Employee            |
|---------------------|
| EmployeePOID <<PK>> |
| Name                |
| Salary              |

Copyright 2002-2006 Scott W. Ambler



# Valós osztályok leképzése táblába – 3

---

## ➤ Előnyök

- Átláthatóbb
- Jobban illeszkedik az objektum modellhez
- Gyors adatelérés

## ➤ Hátrányok

- Osztály módosítása → Hierarchia szintek
- Több szerepet is betöltő példányok kezelése
  - Employee → Executive
  - Employee és Customer

## ➤ Célszerű használni

- Ritkán változó struktúrák esetén



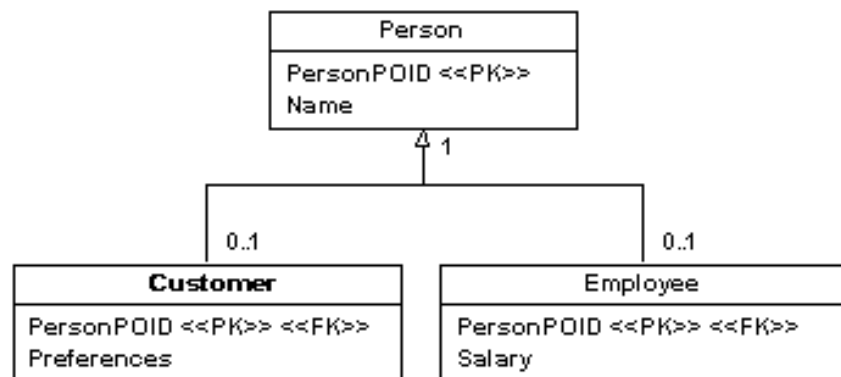
# Összes osztály leképezése táblába – 1

---

- Osztály hierarchiát követik a táblák
- Szülő – gyerek viszony leképzése idegen kulccsal
- Példány azonosító



# Összes osztály leképezése táblába – 2



Copyright 2002-2006 Scott W. Ambler



# Összes osztály leképezése táblába – 3

---

## ➤ Előnyök

- Könnyű megérteni (egy az egyben leképezés)
- Könnyű módosítani a szülő osztályok struktúráját

## ➤ Hátrányok

- Összetett adatbázis séma
- Egy példány adatai több táblában vannak
  - Összetett lekérdezés

## ➤ Célszerű használni

- Komplex hierarchia esetén
- Változó struktúra esetén



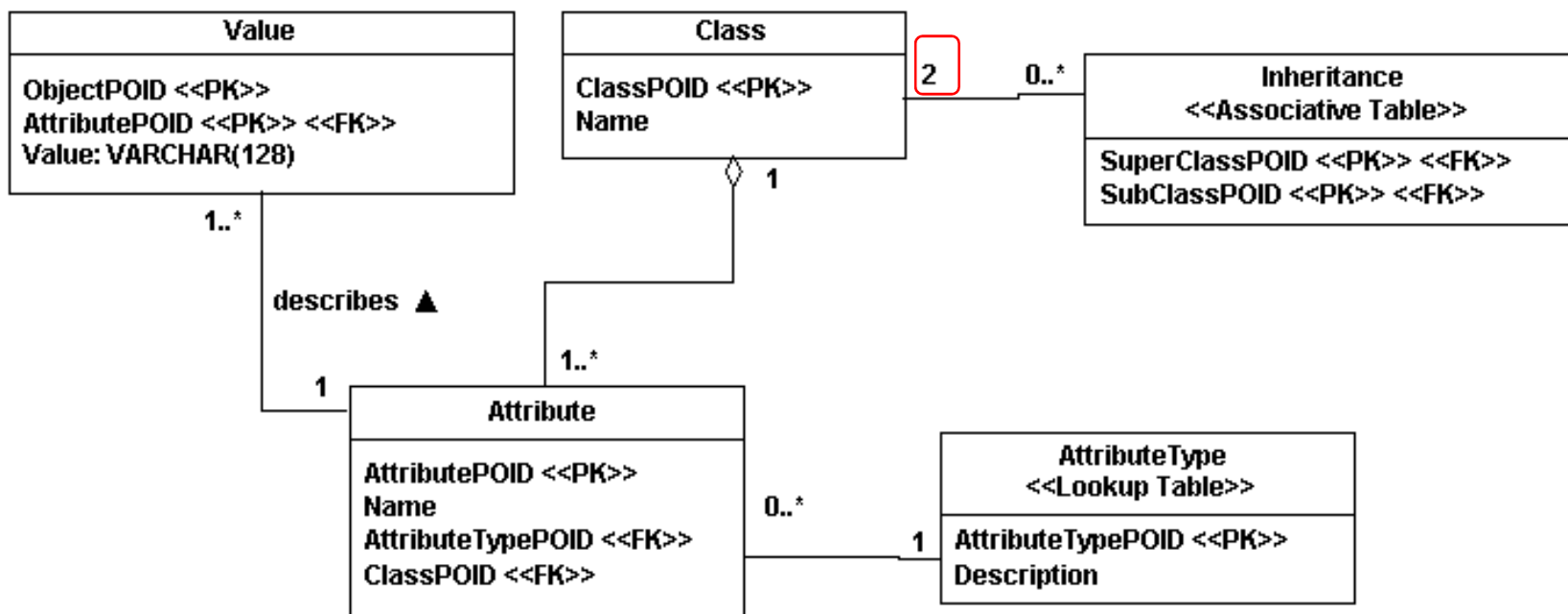
# Leképezés általános struktúrába – 1

---

- Meta data driven megoldás
- Általános séma
  - Tetszőleges hierarchia leírható
  - Független a konkrét osztályoktól
    - Osztály hierarchia → Meta adat
    - Osztály példányok → Attribútumok manifesztálódása



# Leképezés általános struktúra – 2



Copyright 2002-2006 Scott W. Ambler



# Leképezés általános struktúrába – 3

---

## ➤ Előnyök

- Perzisztencia kezelő keretrendszerekhez illeszkedik
- Flexibilis
- „Bármilyen” leírható benne

## ➤ Hátrányok

- Elsőre nehéz „megemészteni”
- Nehéz „összeszedni” egy objektum példány adatait
- Meta adatok közvetlen hatása → karbantartás, telepítés
- Nagy adatmennyiség esetén nem hatékony

## ➤ Célszerű használni

- Komplex alkalmazások
- Kis mennyiségű adatok
- „Minden változhat” akár futási időben is



# Többszörös öröklés – 1

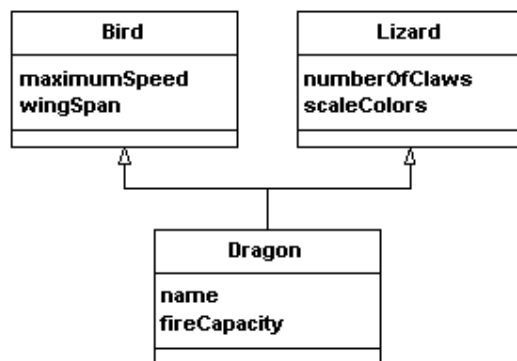
---

- Nem nagyon támogatott, de
  - C++ → Még mindig
  - ...
- Ugyanazok a technológiák használhatók
  - Általános megoldásban már benne van



# Többszörös öröklés – 2

**Class Model**



**Possible Physical Data Models**

| <b>Creature</b>                                                                                                                                                      |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>CreaturePOID &lt;&lt;PK&gt;&gt;</b><br><b>Name</b><br><b>FireCapacity</b><br><b>MaximumSpeed</b><br><b>WingSpan</b><br><b>NumberOfClaws</b><br><b>ScaleColors</b> |



# Objektum kapcsolatok leképzése – 1

## ➤ Kapcsolatok

- Asszociáció
- Aggregáció
- Kompozíció

## ➤ Típusok

- Egy – Egy
- Egy – Több
- Több – Több

→ Referenciális integritás

## ➤ Irány

- Egy irányú
- Két irányú

→ Nem képezhető le

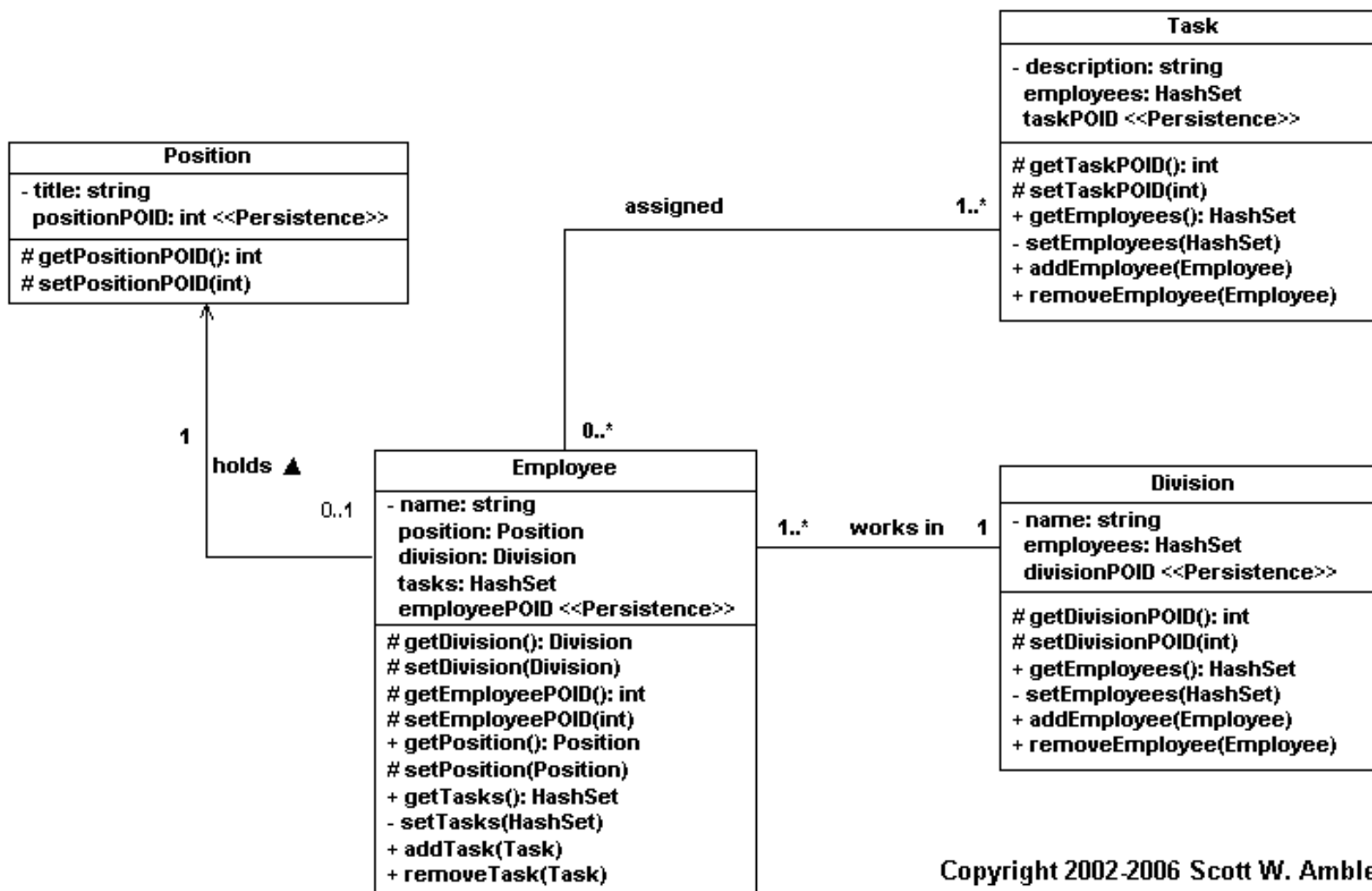


# Objektum kapcsolatok leképzése – 2

---

- Egy – Egy kapcsolat
  - Külső kulcs az egyikbe
    - Közömbös
    - Egy – több lehetőségét magában hordja → Lehet, hogy nem közömbös ☺
- Egy – Több kapcsolat
  - Külső kulcs az „egy”-re
- Több – Több kapcsolat
  - Közvetlenül nem képezhető le
  - Kapcsoló tábla használata
- Kardinalitások
  - Mindkét oldal kötelezőt nem célszerű leképezni
  - Elindulási probléma adatbázis szinten
  - Számolás adatréteg szinten: 0,1,több

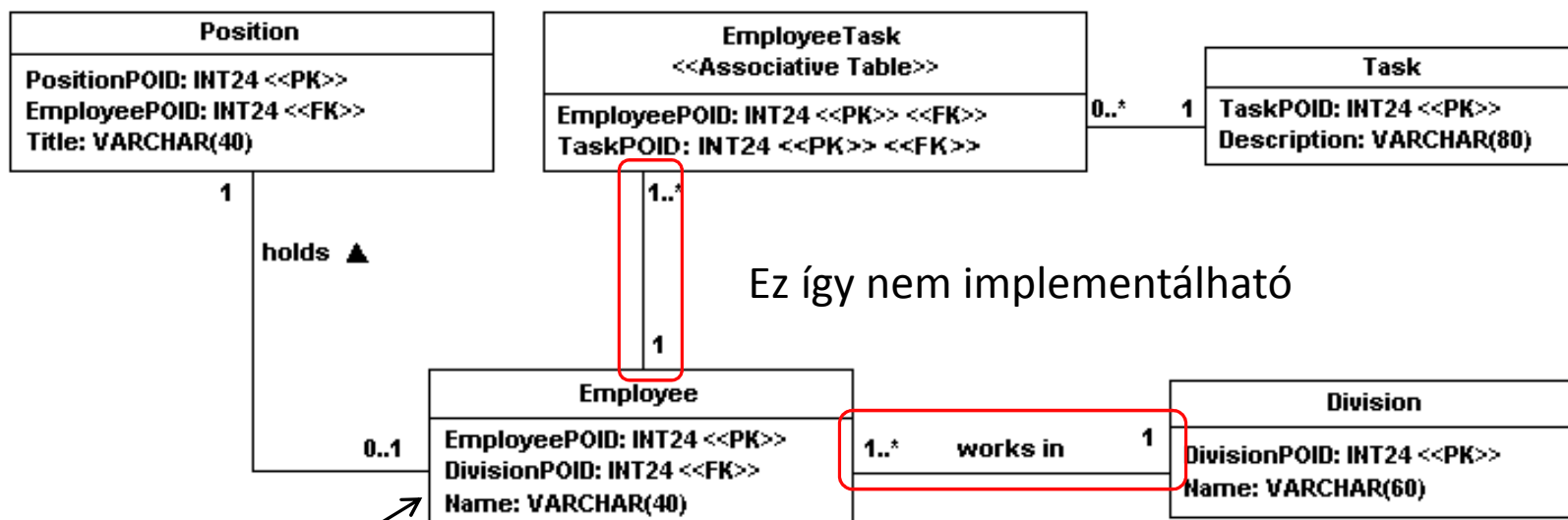
# Objektum kapcsolatok leképzése – 3



Copyright 2002-2006 Scott W. Ambler



# Objektum kapcsolatok leképzése – 4



Copyright 2002-2006 Scott W. Ambler

Egy – több kapcsolat is lehetne az adatréteg alapján



# Rekurzió – 1

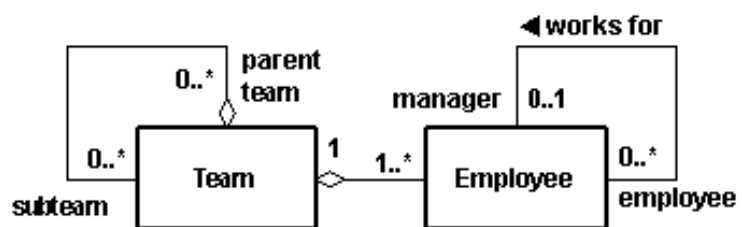
---

- Más néven reflexió
- Kapcsolat kezdő és végpontja ugyan az az entitás
- Hasonlóan a többi kapcsolat
  - Több – több leképezésben kis eltérés



# Rekurzió – 2

<<Class Model>>



Copyright 2002-2006 Scott W. Ambler



# Rendezett gyűjtemények

---

- Sorrendezést adó attribútum
- Sorrend helyesen olvassuk fel
  - Order by ...
- Sorrendet adó attribútum ne legyen része a kulcsnak
  - Sorrend változás hivatkozásokat is érintené
- Sorrend változtatás hatása
  - Több rekordot is módosítani kell
- Elem törlése
  - Nem fontos újra „indexelni”
    - Ekkor csak sorrendet jelent pozíció indexet nem
- Hézagos kiosztás (pl 10-esével)
  - Van szabad pozíció
  - Kevesebb módosítás, ritkábban kell több rekordot



# Osztály szintű tulajdonságok – 1

---

- Osztályra jellemzők
- Nem kötődnek példányhoz
- Példa
  - Következő számla sorszám
- Osztály szintű konstansok kezelése is hasonló



## Osztály szintű tulajdonságok – 2

- Minden tulajdonságnak külön tábla
  - Gyors
  - Sok kicsi tábla → áttekinthetetlen adatmodell
- Minden tulajdonság ugyanabban a táblában különböző oszlopokban
  - Gyors
  - Egyszerű (egy tábla az összesnek)
  - Konkurencia (mivel sor alapú és nem oszlop)
- Osztályonként egy tábla az értékek különböző oszlopokban
  - Gyors
  - Sok kicsi tábla → áttekinthetetlen adatmodell

Nem flexibilis



# Osztály szintű tulajdonságok – 3

---

## ➤ Általános megoldás

- Minden tulajdonság új rekord
  - Osztály
  - Tulajdonság név
  - Érték
- Adatkonverziót meg kell oldani
- Egyszerű bővíthetőség
  - Új tulajdonság → Új rekord



# Osztály szintű tulajdonságok – 4

**Class Model**

| Customer                                                                                                                                  |
|-------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> <li>- customerNumber: int</li> <li>- name: string</li> <li>- <u>nextCustomerNumber: int</u></li> </ul> |
| # calculateNextCustomerNumber(): int                                                                                                      |

**Possible Physical Data Models**

| CustomerNumber            |
|---------------------------|
| NextCustomerNumber: INT24 |

Copyright 2002-2006 Scott W. Ambler



# Néhány jó tanács

---

## ➤ Átláthatóság

- Konzekvens elnevezések
- Meta adatok karbantartása
  - Kapcsolatoknál is

## ➤ Teljesítmény

- „Egyszerűbb” általában jobb
- Késletett beolvasás
  - Nagyméretű attribútumok csak ha kellenek (pl.: kép)
  - Objektumok csak ha kellenek
    - Nem biztos, hogy az összes hivatkozott objektum kell
      - Tervezői döntés



# LINQ

---



# Alapprobléma

---

## ➤ Adat != Objektum

### ■ ORM

## ➤ SQL nyelv

☺ Adatorientált

☺ Egyszerű lekérdezést megfogalmazni

☹ Nem objektum alapú

☹ Nem típusos

☹ Nem épül be nyelvi elemként



# Hagyományos adatelérés

```
SqlConnection c = new SqlConnection(...);  
c.Open();  
SqlCommand cmd = new SqlCommand(  
    @"SELECT c.Name, c.Phone  
        FROM Customers c  
        WHERE c.City = @p0"  
    );  
cmd.Parameters.AddWithValue("@po", "London");  
DataReader dr = c.Execute(cmd);  
while (dr.Read()) {  
    string name = dr.GetString(0);  
    string phone = dr.GetString(1);  
    DateTime date = dr.GetDateTime(2);  
}  
dr.Close();
```

## Problémák

- SQL szöveg
- Paraméter kötés
- Típusosság hiányzik
- Fordító nem tud ellenőrizni
- Intellisense hiányzik



# Adatelérés LINQ segítségével

```
public class Customer
{
    public int Id;
    public string Name;
    public string City;
    public string Phone;
}
```

```
-----

GridView1.DataSource = from customer in db.Customers
                        where customer.City == "London"
                        select customer;

GridView1.DataBind();
```

## Előnyök

- Típusos
- Objektumokra épül
- Típus ellenőrzés fordítási időben



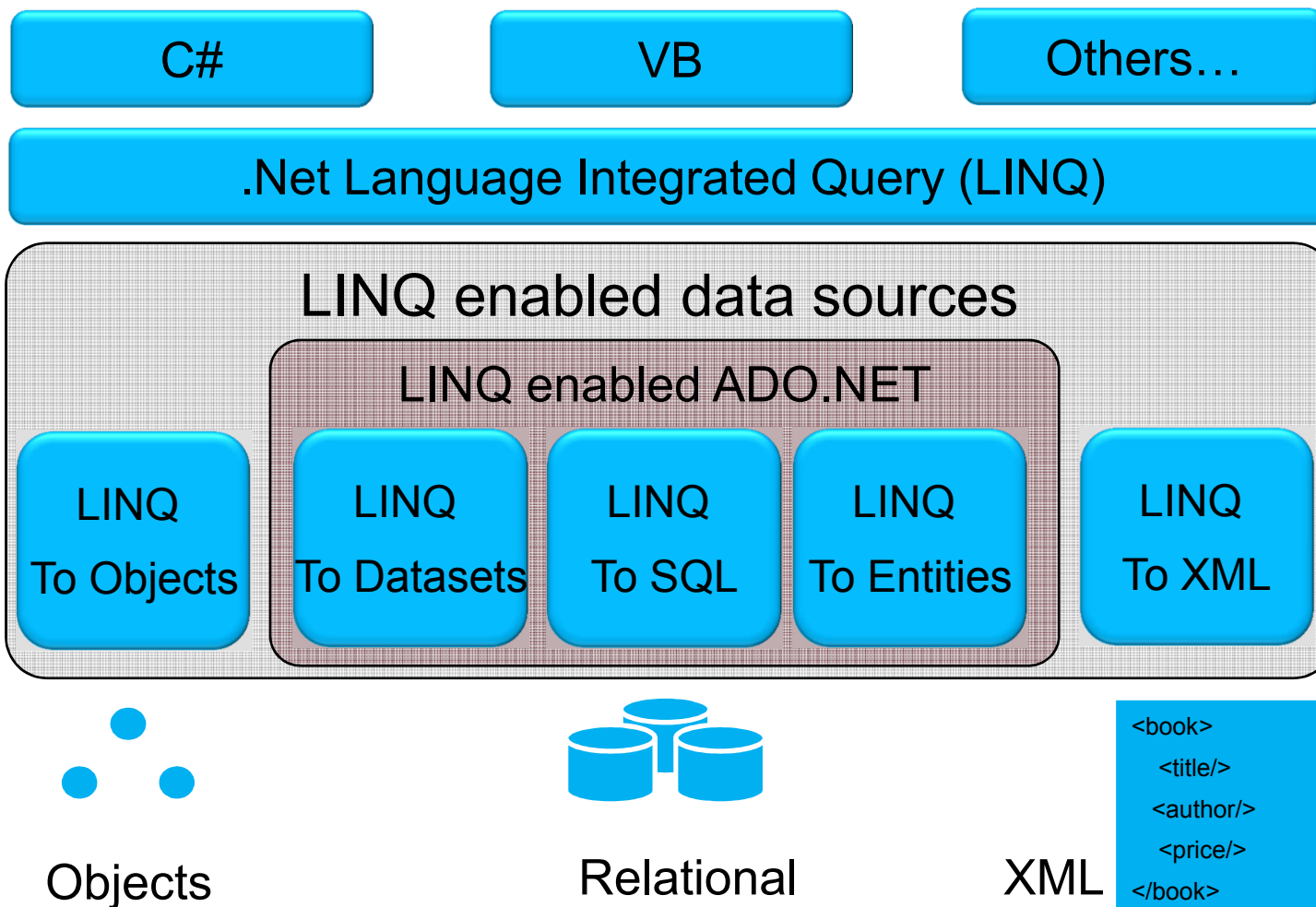
# Mi a LINQ?

---

- .NET Language-Integrated Query
- .Net Framework 3.0-ra épül
- Nyelvi elemként megjelenő lekérdezés
- Visual Studio 2008
  - C# 3.0
  - VB 9.0



# LINQ Felépítése





# LINQ to Objects használata

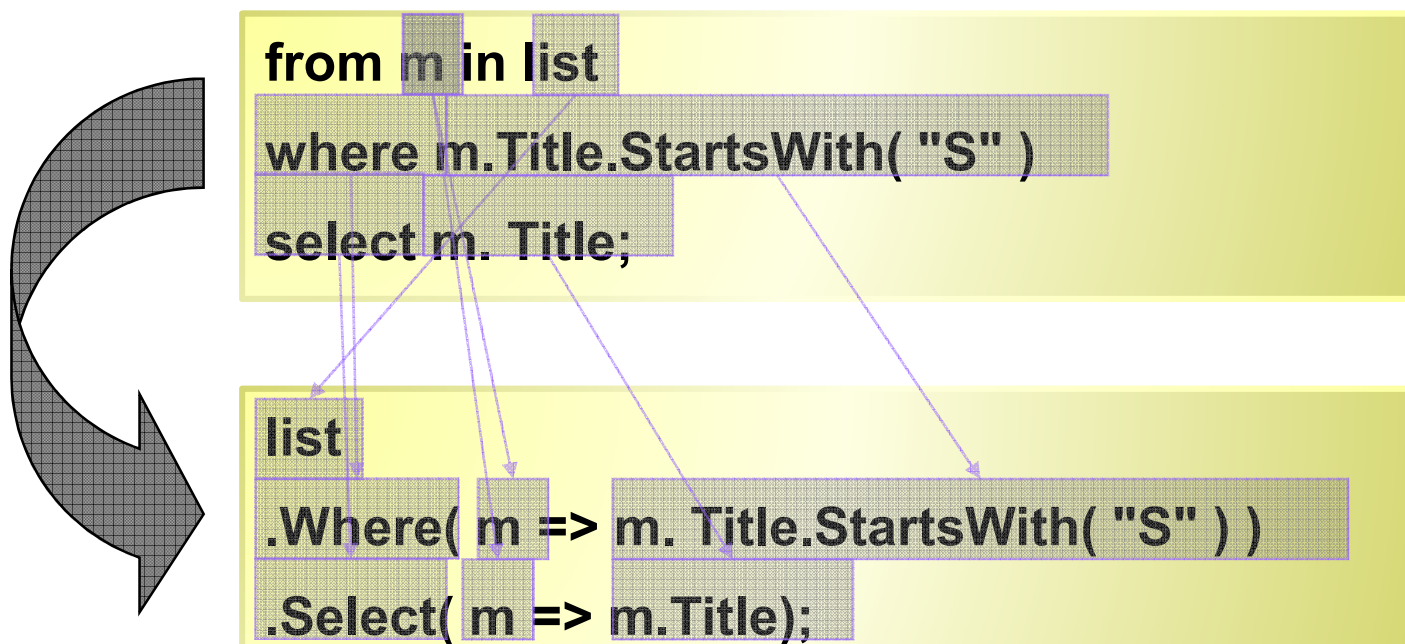
---

- Minden gyűjteményen használható
  - IEnumerable interfész
    - Select, where, join, GroupBy, ...
  - Késletett kiértékelés
    - Eredményhalmaz iterálásakor
  - Lekérdezés megfogalmazása független a gyűjteménytől
    - Kifejezés fa
- Nyelvi elemként jelenik meg
  - Függvényhívássá alakul át



# Lekérdezések átalakítása

- A nyelvbe ágyazott lekérdezéseket metódushívásokká alakítja a fordító





# LINQ to Objects példa – 1

Tömb megvalósítja  
IEnumerable<T> interfészt

```
string [] cities = { "Auckland", "Oslo", "Sydney",  
                    "Seattle", "Paris", "Los Angeles" };
```

```
IEnumerable<string> places = from city in cities  
                             where city.Length > 5  
                             orderby city descending  
                             select city;
```

```
GridView1.DataSource = places;  
GridView1.DataBind();
```

LINQ  
lekérdezés

IEnumerable<string>  
Data binding!



# Custom City Class

---

```
public class City
{
    public string Name;
    public string Country;
    public int DistanceFromSeattle;
}

List<City> locations = GetLocations();
```



# LINQ to Objects példa – 2

Gyűjtemény megvalósítja az `IEnumerable<T>` interfészt

```
List<City> locations = GetLocations();
```

```
IEnumerable<City> places = from city in locations
                           where city.DistanceFromSeattle > 1000
                           orderby city.Country, city.Name
                           select city;
```

```
GridView1.DataSource = places;
GridView1.DataBind();
```

`IEnumerable<City>` gyűjtemény tartalmazza a lekérdezés eredményét



# LINQ to XML

---

- XML mindenhol megjelent
  - Konfigurációs állományok
  - Dokumentumok
  - Platformfüggetlen kommunikáció
  - Hierarchikus adatok
  - Web szolgáltatások
  - ...



# XML dokumentumok kezelése

---

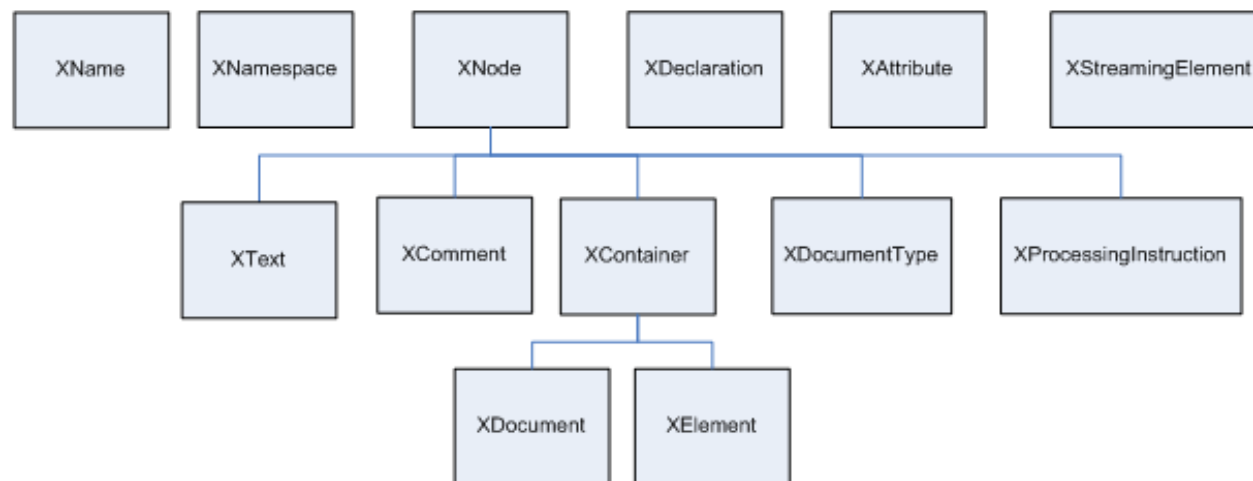
- XmlReader / XmlWriter
  - ☺ Hatékony, de nehézkes
- DOM
  - ☹ Sok adatnál lassú
- XSLT
  - ☺ Kevésbé mély struktúrájú adatokra
  - ☹ Nehéz megtanulni
  - ☹ Sok adatnál
- LINQ to XML
  - ☺ Egységes szemlélet
  - ☺ Egyszerű
  - ☺ Gyors



# LINQ to XML felépítése

## ➤ System.Xml.LINQ

## ➤ Új osztályok



## ➤ Közvetlen nyelvi támogatás

### ■ Nyelvbe ágyazott lekérdezések

□ Ancestors

□ Descendants

} → „Fa bejárása”

### ■ VB 9: XML a nyelvben idézőjelek nélkül → nyelvi elem



# Xml létrehozása

```
XElement xml = new XElement("contacts",
    new XElement("contact",
        new XAttribute("contactId", "1"),
        new XElement("firstName", "Béla"),
        new XElement("lastName", "Nagy")
    ),
    new XElement("contact",
        new XAttribute("contactId", "2"),
        new XElement("firstName", "Béla"),
        new XElement("lastName", "Kicsi")
    )
);

Console.WriteLine(xml);
```



# Az elkészített Xml

```
<contacts>
  <contact contactId="1">
    <firstName>Béla</firstName>
    <lastName>Nagy</lastName>
  </contact>
  <contact contactId="2">
    <firstName>Béla</firstName>
    <lastName>Kicsi</lastName>
  </contact>
</contacts>
```



# XDocument létrehozása

```
XDocument doc = new XDocument(  
    new XDeclaration("1.0", "utf-8", "yes"),  
    new XComment(„RSS Feed”),  
    new XElement("rss",  
        new XAttribute("version", "2.0"),  
        new XElement("channel",  
            new XElement("title", "RSS csatorna címe"),  
            new XElement("description", "RSS csatorna leírás"),  
            new XElement("link", "http://example.com"),  
            new XElement("item",  
                new XElement("title", „Első cikk”),  
                new XElement("description", „Első leírás”),  
                new XElement("pubDate", DateTime.Now.ToString()),  
                new XElement("guid", Guid.NewGuid())) ) ) );  
doc.Save(@"c:\sample.xml");
```



# Az elkészített Xml doksi

```
<!-- RSS Feed -->
<rss version="2.0">
  <channel>
    <title>RSS csatorna cím</title>
    <description>RSS csatorna leírás.</description>
    <link>http://example.com</link>
    <item>
      <title>Első cikk címe</title>
      <description>Első leírás</description>
      <pubDate>2011-05-21 15:12:30</pubDate>
      <guid>11e7e86b-1ad6-465a-8f74-cf00be28aaba</guid>
    </item>
  </channel>
</rss>
```



# Xml lekérdezés C#

```
// Lekérdezés összeállítás és eredmény kiírása
var q = from c in xml.Descendants("contact")
        where (int)c.Attribute("contactId") <= 2
        select (string)c.Element("firstName") + " " +
               (string)c.Element("lastName");

foreach (string name in q)
    Console.WriteLine("Név = {0}", name);
```

Név = Nagy Béla  
Név = Kicsi Béla



# Demó

---

Avagy ki szerelmesebb Rómeó vagy Júlia?



# LINQ to SQL

---

- Relációs adatbázisok és OO világ összekötése
- Egyfajta ORM
- Csak MS SQL Szerverrel működik
- Más platformok
  - Entity Framework
- Fejlesztése lezárt, helyette EF



# LINQ to SQL Architektúra

```
from c in db.Customers
where c.City == "London"
select
    new { c.Name, c.Phone }
```

LINQ  
Lekérdezés

Alkalmazás

Objektumok

SubmitChanges()

LINQ to SQL  
(ADO.NET)

Egyéb szolgáltatások  
- változás követés  
- konkurencia kezelés  
- Objektum azonosítás

SQL Lekérdezés

Sorok

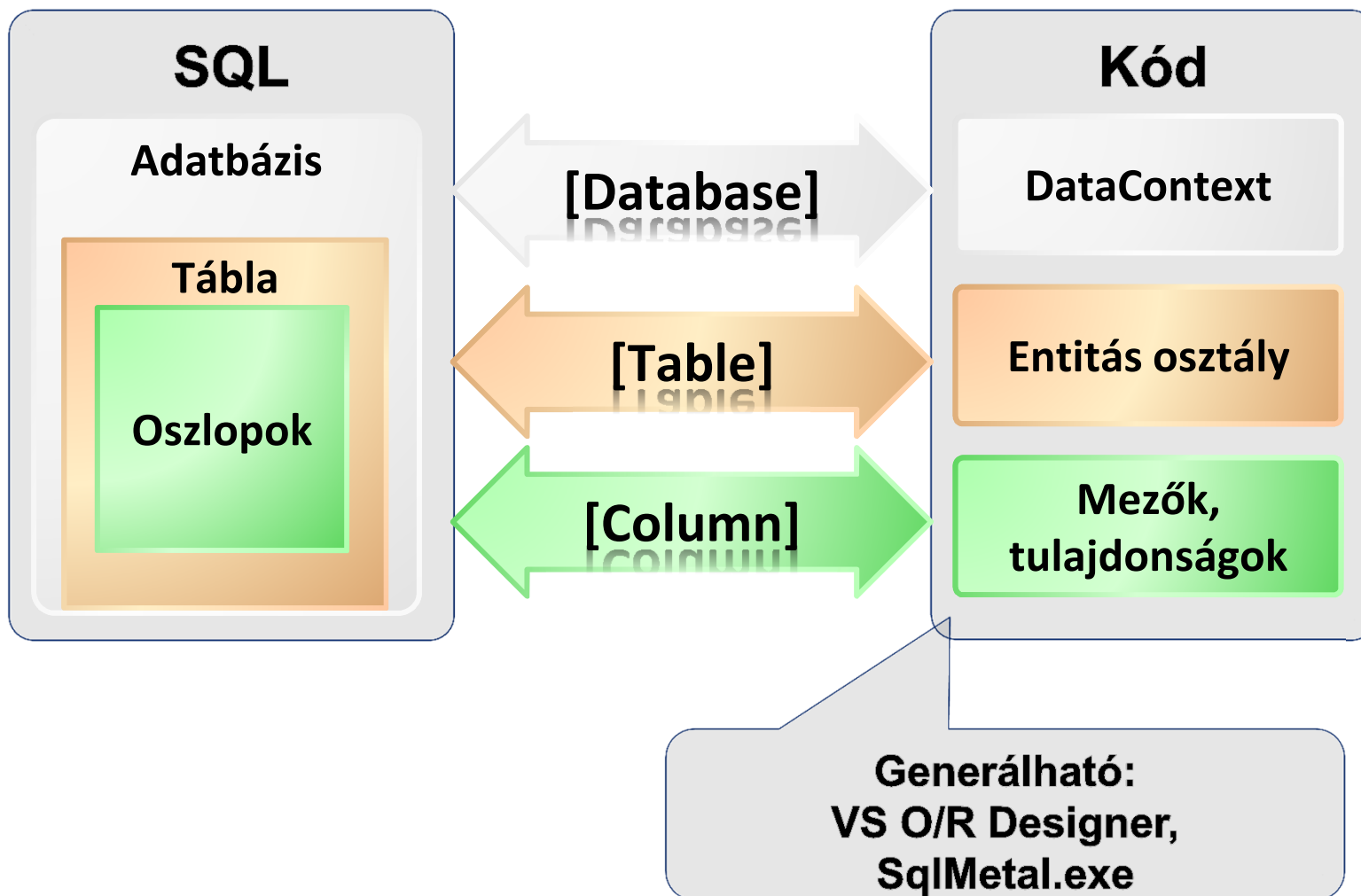
SQL / Tárolt eljárás

```
select Name, Phone
from customers
where city = 'London'
```

SQLServer



# Leképezés metaadatokkal





# Leképzés – Entitás osztály

---

```
[Table(Name="Customers")]
```

```
public class Customer
```

```
{
```

```
    [Column(IsPrimaryKey=true)]
```

```
    public string CustomerID;
```

```
    [Column]
```

```
    public string City;
```

```
}
```



# Leképzés – DataContext

---

```
DataContext db = new
    DataContext("c:\\northwind\\northwnd.mdf");

// Get a typed table to run queries
Table<Customer> Customers = db.GetTable<Customer>();

// Query for customers from London
var q = from c in Customers
    where c.City == "London"
    select c;
foreach (var cust in q)
    Console.WriteLine("id = {0}, City = {1}",
        cust.CustomerID, cust.City);
```



# Leképzés – Strongly Typed DataContext

```
public partial class Northwind : DataContext
{
    public Table<Customer> Customers;
    public Table<Order> Orders;
    public Northwind (string connection):
        base(connection) {}
}

...

Northwind ndc = new Northwind();
var q2 = from c in ndc.Customers
        where c.Orders.Count > 15
        select c;
```



# Lekérdezések

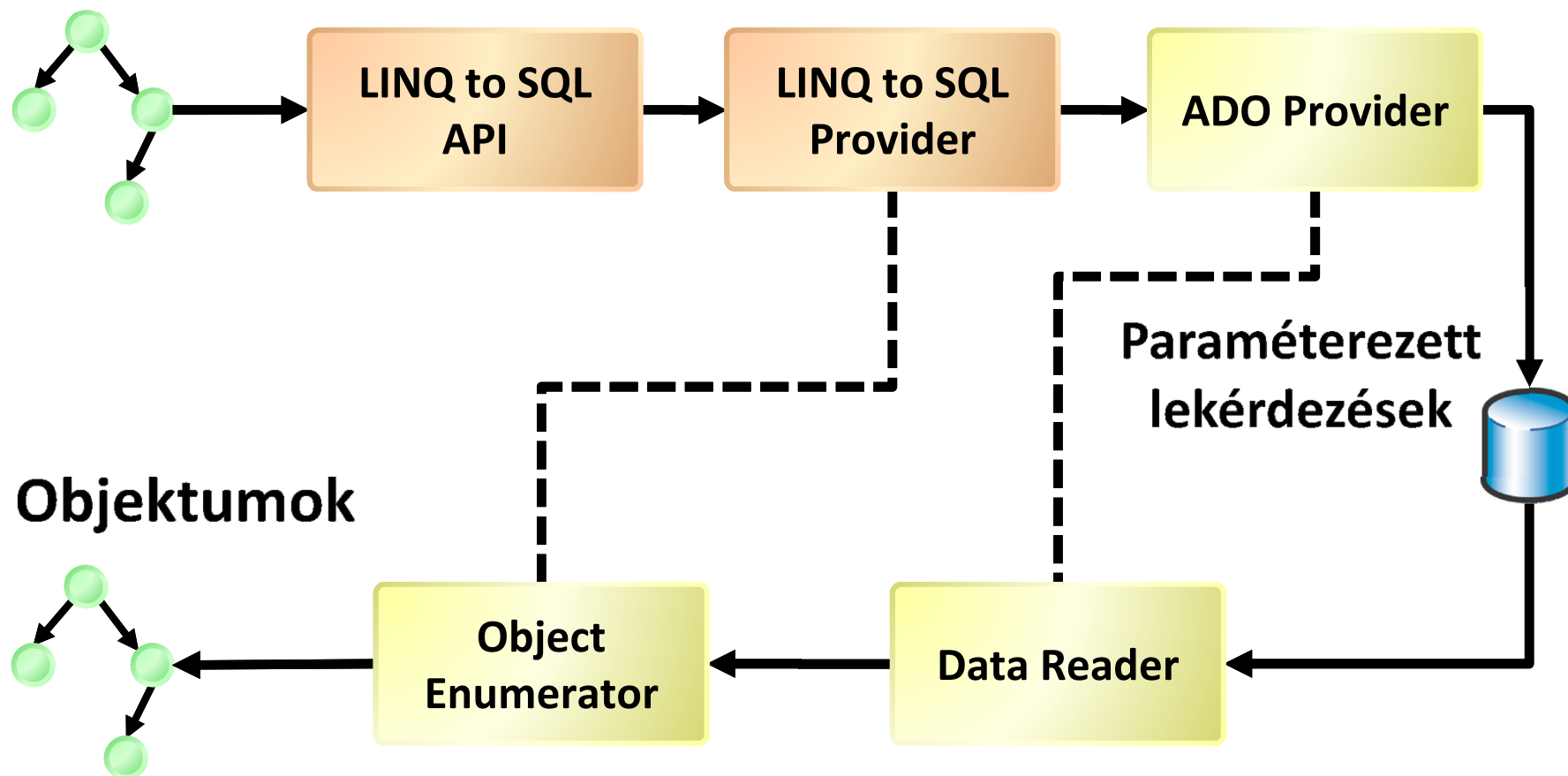
- IQueryable<T>
  - .Expression
  - .Execute()
- A *query* csak egy leírás
- Késleltetett végrehajtás: iteráláskor
- Relációk betöltése az első hivatkozásnál:
  - DataContext.DeferredLoadingEnabled
- PI:

```
var q = (  
    from c in dc.Customers  
    where c.City == "London"  
    select c).Including( c => c.Orders );
```



# Lekérdezések végrehajtása

## Lekérdezés





# Vetítés

---

- Az adatok tetszőleges formában visszakérhetők
  - Csak oszlopok
  - Nevesített típusok
  - Névtelen típusok
- Csak lekérdezésre!

```
var q = from c in dc.Categories
        where c.Products.Count > 5
        select new {
            Name = c.CategoryName,
            Count = c.Products.Count
        };
```



# Vetítés – Példák

---

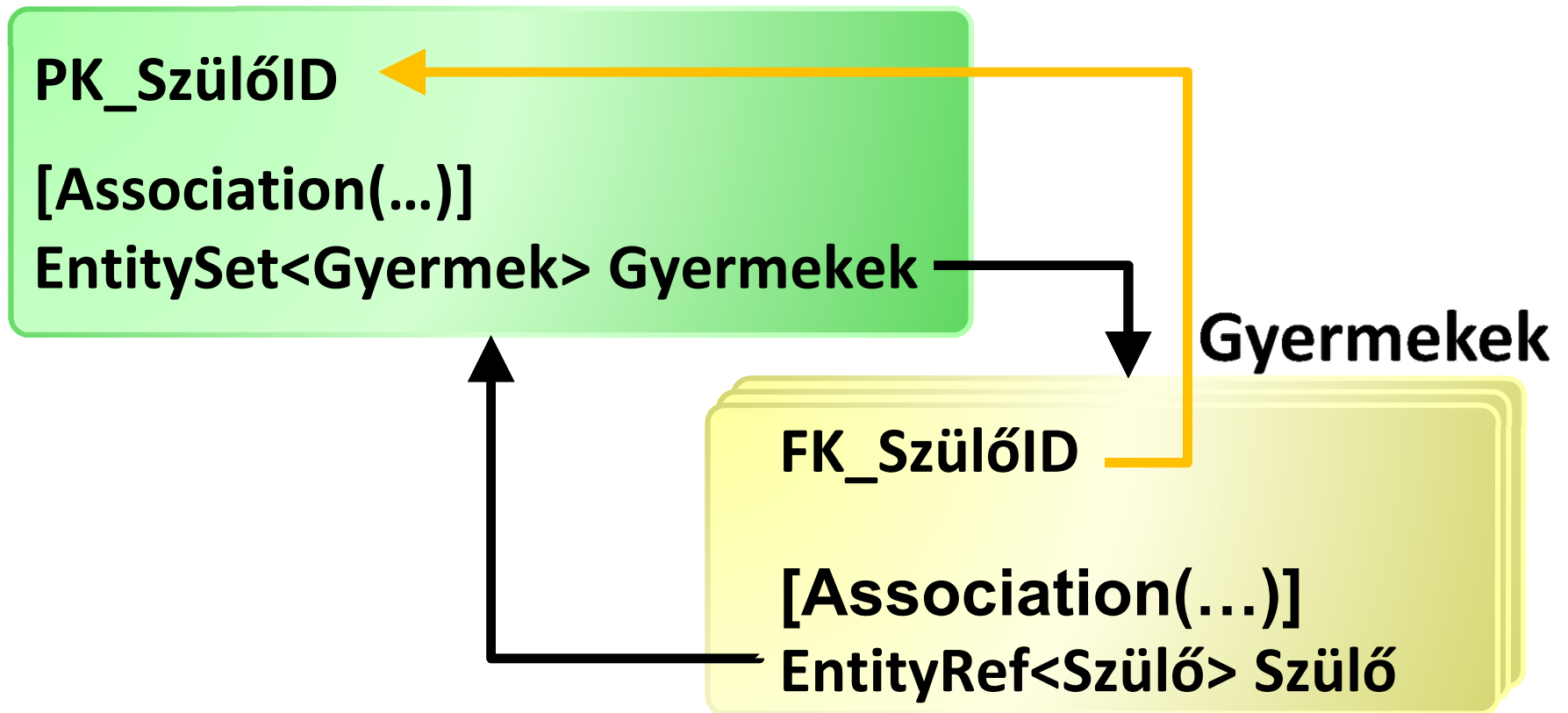
```
var q = from c in db.Customers  
        where c.City == "London"  
        select c.CompanyName;
```

```
var q =    from c in db.Customers  
           where c.City == "London"  
           select new { c.CompanyName, c.Phone };
```



# Relációk

## Szülő



Cél: kapcsolatok bejárása tulajdonságokon keresztül



# Relációk – Példa

---

```
[Table(Name="Customers")]
```

```
public class Customer
```

```
{
```

```
    [Column(IsPrimaryKey=true)]
```

```
    public string CustomerID;
```

```
    ...
```

```
    private EntitySet<Order> _Orders;
```

```
[Association(Storage="_Orders", OtherKey="CustomerID")]
```

```
    public EntitySet<Order> Orders {
```

```
        get { return this._Orders; }
```

```
        set { this._Orders.Assign(value); }
```

```
    }
```

```
}
```



# Relációk – Példa folytatás

```
[Table(Name="Orders")]
public class Order
{
    [Column(IsPrimaryKey=true)]
    public int OrderID;
    [Column]
    public string CustomerID;
    private EntityRef<Customer> _Customer;

    [Association(Storage="_Customer", ThisKey="CustomerID")]
    public Customer Customer {
        get { return this._Customer.Entity; }
        set { this._Customer.Entity = value; }
    }
}
```



## Relációk – Példa folytatás 2

---

```
var q = from c in db.Customers
        from o in c.Orders
        where c.City == "London"
        select new { c, o };
```

```
var q = from o in db.Orders
        where o.Customer.City == "London"
        select new { c = o.Customer, o };
```



# Relációk betöltése

---

- EntitySet mikor töltődjön be?
- **Távoli lekérdezés**
  - IQueryable<T>
  - Többször fordulhat a szerverhez
  - Mindig friss adatok
- **Helyi lekérdezés**
  - IEnumerable<T>
  - EntitySet.Load()



# Join

---

- Tulajdonságok, idegen kulcsok → reláció
- Ad hoc kapcsolat → join



# Join Példa – 1

---

```
var q =  
    from s in db.Suppliers  
    join c in db.Customers on s.City equals c.City  
    into scusts  
    select new { s, scusts };
```



# Join – Példa 2

---

```
var q =      from s in db.Suppliers
              join c in db.Customers on s.City equals c.City
              into custs
              orderby custs.Count() descending
              where custs.Count() > 0
              select new
              {
                  Supplier = s,
                  Customers = custs
              };
```



# Join – Példa 2 folytatás

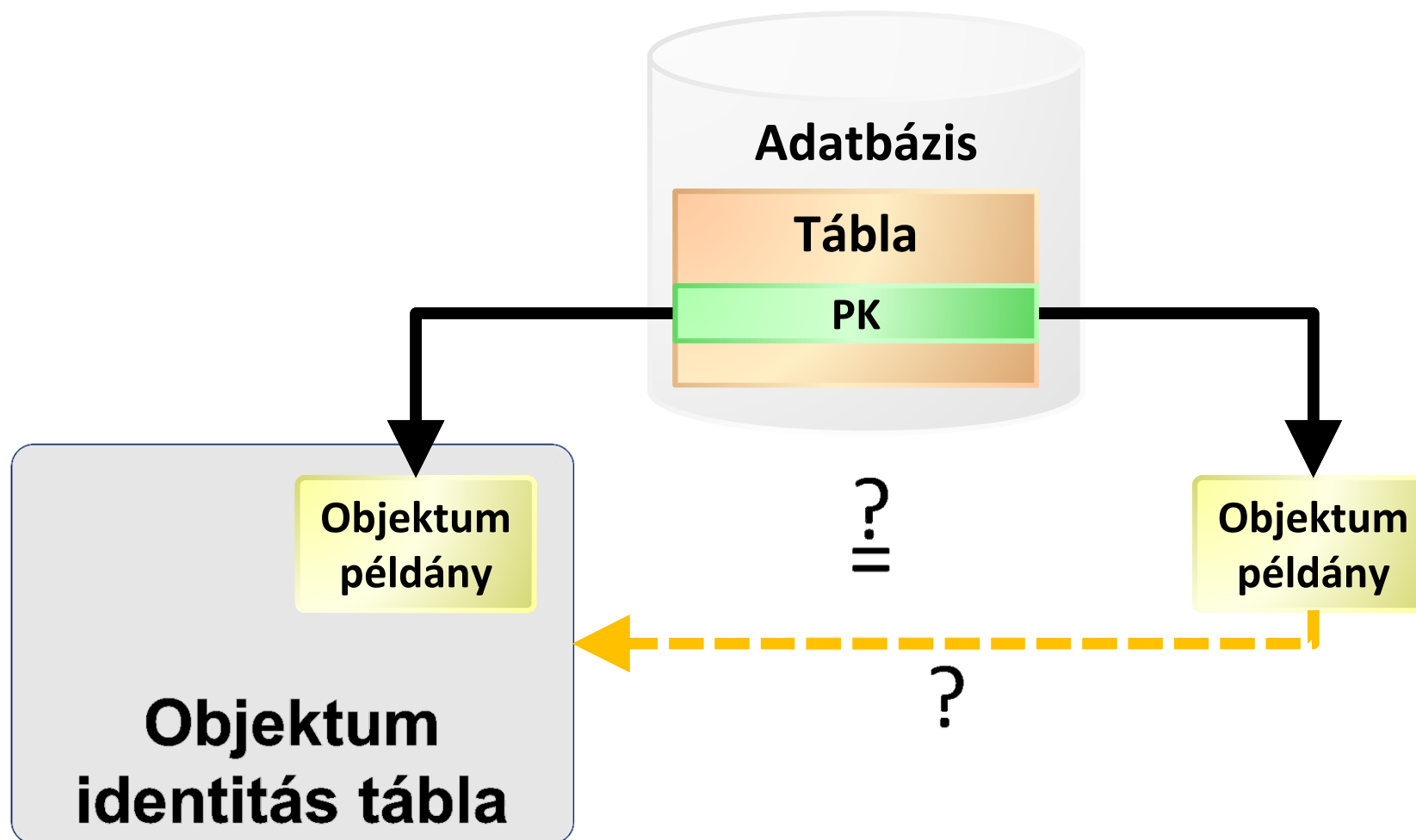
---

```
foreach( var result in q )
{
    // szállító adatai
    Console.WriteLine( "\n{0, -50}{1, -20}{2} db",
                        result.Supplier.CompanyName,
                        result.Supplier.City,
                        result.Customers.Count() );

    // A szállító városában lévő vevők adatai
    foreach( Customer c in result.Customers )
    {
        Console.WriteLine( "    {0}", c .CompanyName );
    }
}
```



# Objektumok egyedisége





# Entitások több rétegben

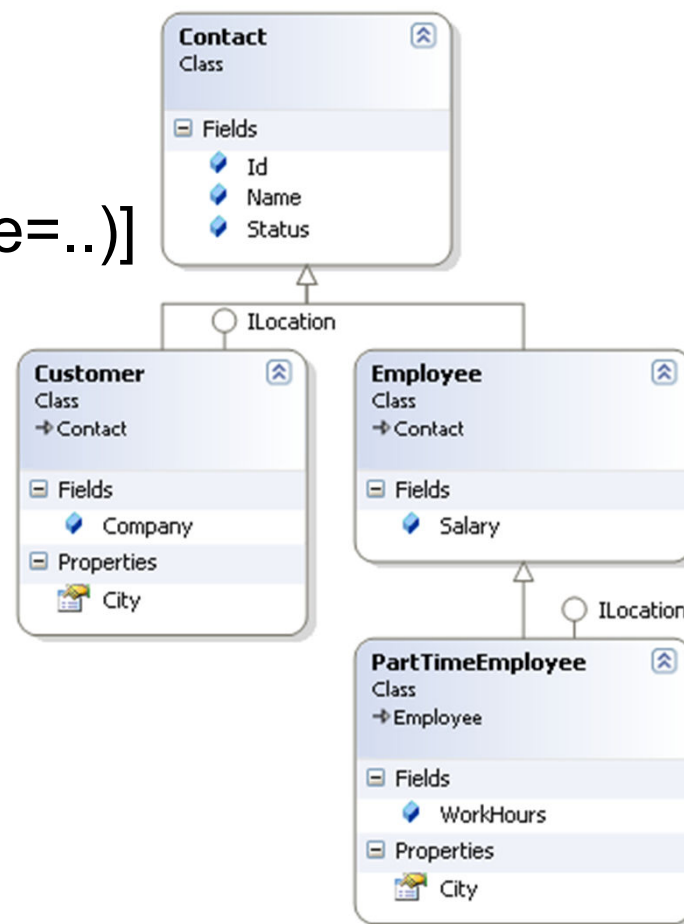
---

- Lekérdezés és módosítás több körben
- `Table.Attach( T item )`



# Öröklés

- Egyetlen táblában
- [Column (IsDiscriminator = true )]
- [InheritanceMapping(Code=.., Type=..)]





# Öröklés Példa

---

```
[Table]
```

```
[InheritanceMapping( Code = "C", Type = typeof( Customer ) )]
```

```
[InheritanceMapping( Code = "E", Type = typeof( Employee ) )]
```

```
[InheritanceMapping( Code = "P", Type = typeof( PartTimeEmployee  
    ) )]
```

```
[InheritanceMapping( Code = "X", Type = typeof( Contact ),  
    IsDefault = true )]
```

```
class Contact
```

```
{
```

```
    [Column( IsPrimaryKey = true )]
```

```
    public int Id;
```

```
    [Column]
```

```
    public string Name;
```

```
    [Column( IsDiscriminator = true )]
```

```
    public string Status;
```

```
}
```



# Öröklés Példa folytatás – 1

---

```
class Customer : Contact
{
    [Column]
    public string Company;
    [Column]
    public string City { get; set; }
}
class Employee : Contact
{
    [Column]
    public int salary;
}
class PartTimeEmployee : Employee, Ilocation
{
    [Column]
    public int workHours;
    [Column]
    public string City { get; set; }
}
```



# Öröklés Példa folytatás – 2

---

```
class PartnerDB : DataContext
{
    public Table<Contact> Contacts;
    public PartnerDB( string connStr )
        : base( connStr ) {}
}
```



# Öröklés Példa folytatás – 3

---

```
// Összes Customer lekérdezése  
PartnerDB db = new PartnerDB( connStr );  
var q = from c in db.Contacts  
        where c is Customer  
        select c;
```



# Változáskövetés

---

- Automatikusan a gráf bármely pontján
  - Add(), Remove(), new, null
- Memóriában
  - DataContext.SubmitChanges()
- Egyedi logika
  - [InsertMethod], [UpdateMethod], [DeleteMethod]
- DataContext.ObjectTrackingEnabled → read-only



# Változási értesítések

---

- **INotifyPropertyChanging**
  - PropertyChanging esemény
  - System.Data.Linq névtér
- **INotifyPropertyChanged**
  - PropertyChanged esemény
  - System.ComponentModel névtér
- **VS O/R Designer generálja**



# Egyidejű módosítások

---

- Optimista konkurencia kezelés
  - [Column( UpdateCheck=Always|Never|WhenChanged, IsVersion=True | False )]
- DataContext.SubmitChanges( ConflictMode )
  - FailOnFirstConflict (default), ContinueOnConflict
- ChangeConflictException,  
OptimisticConcurrencyException
- DataContext.Refresh( RefreshMode )
  - KeepChanges, KeepCurrentValues,  
OverwriteCurrentValues



# Tranzakció

---

- Automatikusan read committed
  - DataContext.Transaction
- Saját TransactionScope használható
  - Pesszimista zárolás



# Tárolt eljárások és függvények

---

- [StoredProcedure(..)]
- [Function(..)]
- [Parameter(..)]
- Erősen típusos eredmény osztályok
- Többféle eredmény kezelése
- Nem támogatott:
  - Ha dinamikus SQL-től függ az eredmény formája



# Tárolt eljárások meghívása

- DataContexten keresztül hívhatók meg
- Visszatérési érték alapértelmezésként int

## ■ Tárolt eljárás

```
Declare @result int  
Select @ result= count(*) from customers  
Return @result
```

## ■ Meghívása

```
NorthwindDataContext db =  
    new NorthwindDataContext();  
int count = db.GetCustomerCount();
```



# Tárolt eljárások – paraméter átadás

---

```
// Output paraméter ref-ként jön vissza
int? id = null;
db.AddCategory( "Új kategória", ref id );
Console.WriteLine( "\nÚj kategória azonosítója: {0}", id );

// Entitás gyűjtemény eredményhalmaz
foreach( Customer c in db.GetCustomersInCity( "London" ) )
    Console.WriteLine( "{0}\t{1}", c.CustomerID, c.ContactName);

// Egyedi típus eredményhalmaz
foreach( GetCustomerContactsInCity c in
    db.GetCustomerContactsInCity( "London" ) )
    Console.WriteLine( "{0,-20} {1}", c.ContactName, c.ContactTitle
    );
```



# Adatbázis létrehozása

---

- DataContext.CreateDatabase()
- Csak az adatszerkezetet, logikát nem
- Korábbi adatbázis eldobható
  - DataContext.DatabaseExists()
  - DataContext.DeleteDatabase()
- Jogosultságok!



# Adatbázis létrehozás példa

---

```
[Table(Name="DVDTable")]
public class DVD
{
    [Column(Id = true)]
    public string Title;
    [Column]
    public string Rating;
}
public class MyDVDs : DataContext
{
    public Table<DVD> DVDs;

    public MyDVDs(string connection) : base(connection) {}
}
```



# Adatbázis létrehozás példa folytatás

```
MyDVDs db = new MyDVDs("c:\\mydvds.mdf");  
  
if (db.DatabaseExists()) {  
    Console.WriteLine("Deleting old database...");  
    db.DeleteDatabase();  
}  
  
db.CreateDatabase();
```



# Érdekeség – Lapozás

---

```
NorthwindDataContext db = new NorthwindDataContext();
```

```
var results = from c in db.Customers join o in db.Orders
               on c.CustomerID equals o.CustomerID into custOrders
               from o in custOrders
               select new {
                   Customer = c.CompanyName,
                   OrderDate = o.OrderDate,
                   OrderTotal = o.OrderDetails.Sum(d=>d.UnitPrice)
               };
```

```
GridView1.DataSource = results.Skip(startRow).Take(10);
```



# Demó

---

## LINQ to SQL



# Entity Framework

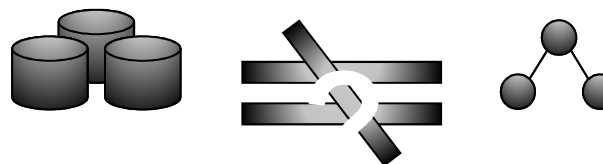
---



# Előzmények

## ➤ Adat != Objektum

### ■ ORM



## ➤ LINQ to SQL

### ■ Közvetlen leképezés az adattáblák és az osztályok között

□ tábla ⇔ osztály

😊 Egyszerűbb adatmanipuláció

☹ **Az adatmodell továbbra is erősen kihat az üzleti logika szerkezetére**



# Adatmodell és üzleti logika

---

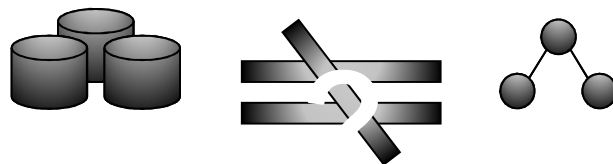
- Az adatbázissémák nem mindig ideálisak az éppen készülő alkalmazások számára
  - Meglévő sémákra kell épülniük az új alkalmazásoknak
  - Alul- vagy felülnormalizált sémák működési vagy teljesítményszempontok miatt
  - Az alkalmazás és az adatbázis is fejlődik az idővel
- Az adatbázisséma gyakran átítatja az egész alkalmazást
  - Minden alkalmazás próbál számára logikus nézeteket létrehozni
    - Tárolt eljárások, nézetek és (a leggyakrabban) ad hoc lekérdezések
  - A séma helyenként döntően visszahat az alkalmazás felépítésére



# Alapprobléma

---

- A fogalmi szint (üzleti logika) a valóságot közvetlenül modellezi
- A logikai szint (adatbázis) tartalmilag azonos, de normalizált formában
- A fogalmi szint és a logikai szint közti különbség áthidalására lehetővé tesszük a kettő közti automatikus leképezést





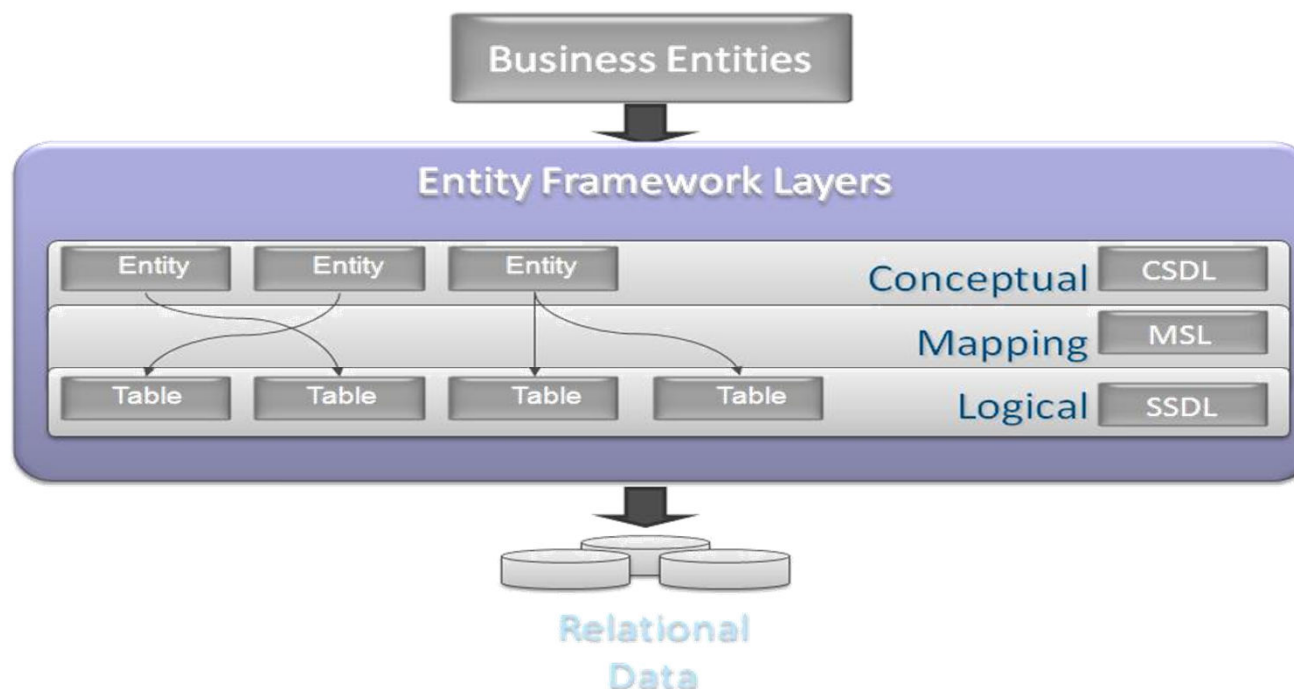
# Entity Framework (EF)

---

- ORM rendszer
- Hasonló a LINQ 2 SQL-hez, csak sokkal testre szabhatóbb
- Lehetővé teszi a logikai (*adatbázis*) és a fogalmi (*üzleti logika*) modellek szétválasztását
- Függetleníti az alkalmazásunkat az adatbázismotortól



# EF leképezés



- Öröklés
  - Többféle
- Egy entitás
  - Akár több táblába
- Egy táblába
  - Akár több entitás
- Egyedi nézetek
- Asszociációk



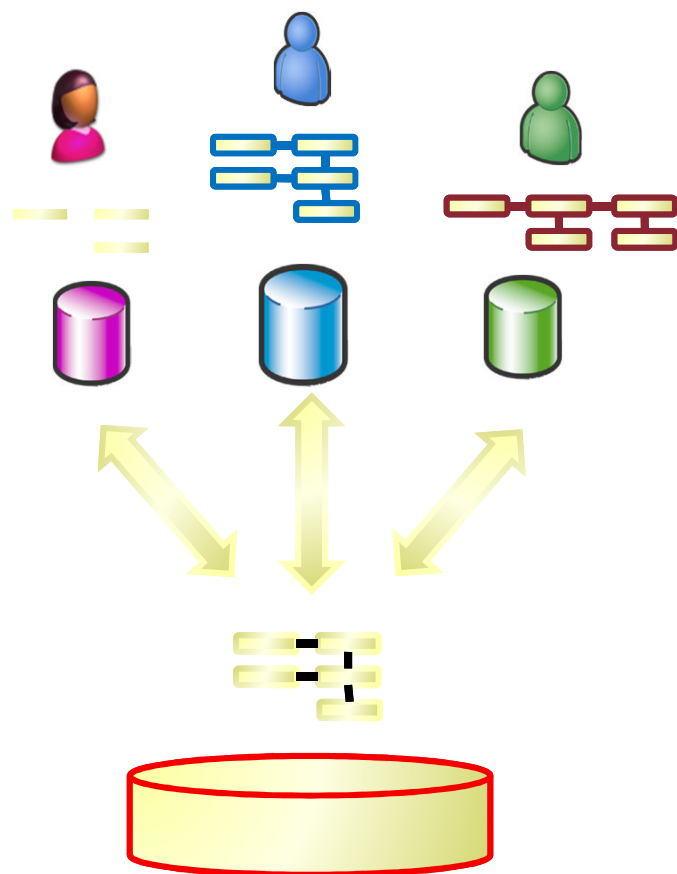
# Entity Data Model

---

- Az EF modelljét EDM-nek hívjuk (**E**ntity **D**ata **M**odel)
- Kliens oldali adatmodell
- Absztrakciós réteg az adatbázis fölött
- Explicit „entitás” és „kapcsolat” fogalmak
  - Entitás: valamely „entitás típus” egy példánya
  - Struktúrált rekordok egy kulccsal
  - Az entitás típusok *örökölhetnek* más entitás típusoktól
- EDM modell gyakorlatilag „futtatható”
  - Nem csak arra jó, hogy a falra ragasszuk 😊



# Kliens nézetek



- Minden alkalmazás saját nézetet kaphat
  - Alkalmazásonként más-más nézetet definiálhatunk ugyanahhoz az adatforráshoz
- A nézetek megvalósítása tisztán a kliensoldalon történik
  - Az adatbázisséma „tisztá” marad
  - Nincs adatbázis módosítás
- Nagy kifejezőerő
  - Frissíthető, táblákat, relációkat átölelő nézetek
  - A frissíthetőség statikusan ellenőrizhető



# Észrevétel

---

- Az indirekció bevezetése az adatelérési rétegben nem újdonság
  - DBMS-nek használata is hasonló
  - Már 20+ éve ezt csinálják!
    - Fizikai adatmodell (fájlok) ⇔ Logikai adatmodell (adattáblák)



# Entitások

---

- Az EDM által leírt elemek az entitások
- Az EDM az entitásokhoz tulajdonságokat rendel, de viselkedésüket nem definiálja
- Az entitások más entitásokhoz relációkkal kapcsolódhatnak



# EDM a kliens oldalon

---

- Osztályokat generál az entitásokhoz
- Kezeli az adatbázis-kapcsolatokat
- Általános lekérdezési eszközök az adatok eléréséhez
- Változáskövetés



# Adatbázis

---

- Az EDM nem rendelkezik közvetlen ismeretekkel az adatbázismotorról
- A konkrét adatbázismotor típusa (Oracle, MSSQL stb.) (elvileg) nincs közvetlen hatással az EDM működésére
- Az entitásokon végzett műveleteket egy külön almodul (provider) fordítja le az adatbázismotor műveleteire
- Támogatott szolgáltatók:
  - SQL Server
  - Oracle
  - MySQL
  - Stb.



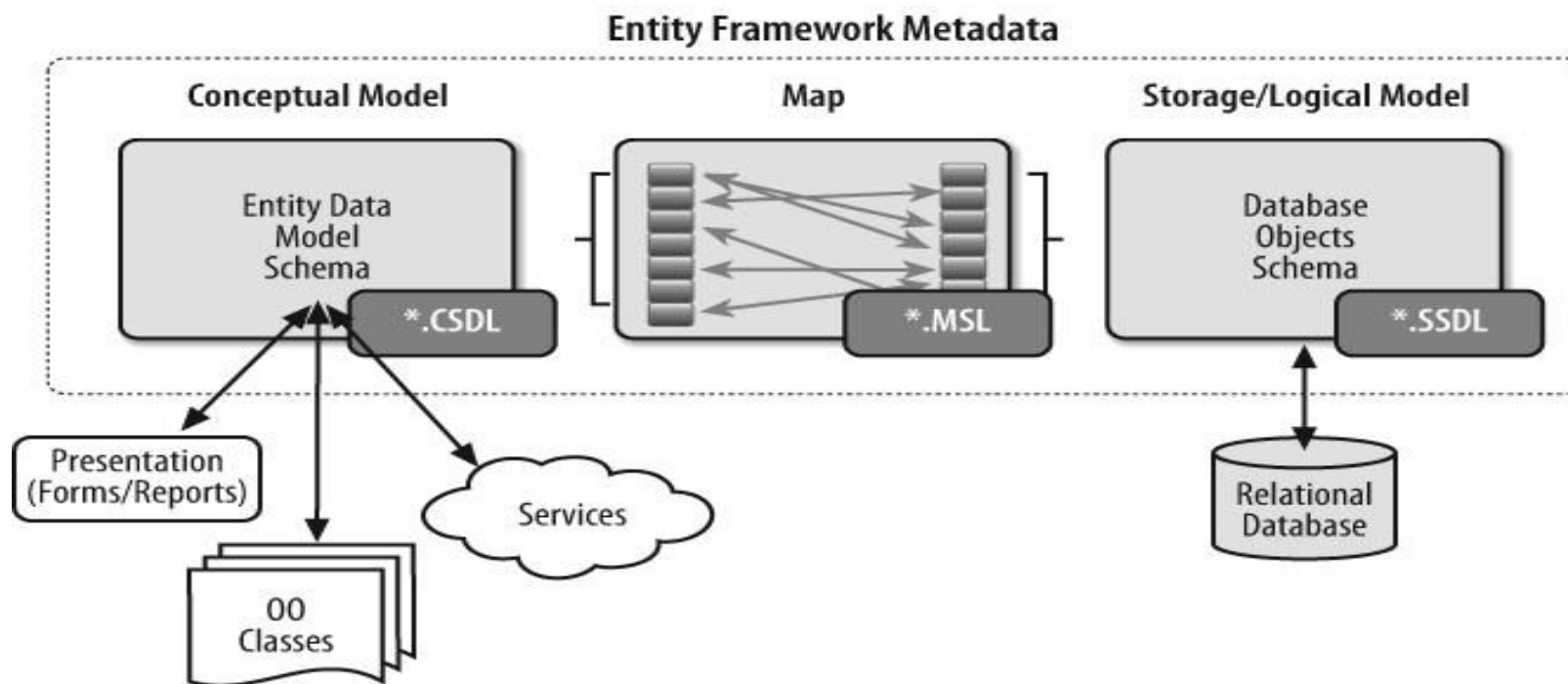
# EDM Designer

---

- Az EDM grafikus megjelenítése
- Tulajdonságok, entitások beállításainak szerkesztés
- Új elemek (pl. relációk) hozzáadása
- Adatbázis séma változásainak lekövetése
- Beépített validáció



# EDM szerkezete



- Conceptual Schema Definition Language (CSDL)
- Store Schema Definition Language (SSDL)
- Mapping Specification Language (MSL)



# EDM – XML leírása

```
<?xml version="1.0" encoding="utf-8"?>
<edmx:Edmx Version="1.0" xmlns:edmx="http://schemas.microsoft.com/ado/2007/06/edmx">
  <!-- EF Runtime content -->
  <edmx:Runtime>
    <!-- SSDL content -->
    <edmx:StorageModels>...</edmx:StorageModels>
    <!-- CSDL content -->
    <edmx:ConceptualModels>...</edmx:ConceptualModels>
    <!-- C-S mapping content -->
    <edmx:Mappings>...</edmx:Mappings>
  </edmx:Runtime>
  <!-- EF Designer content (DO NOT EDIT MANUALLY BELOW HERE) -->
  <edmx:Designer xmlns="http://schemas.">...</edmx:Designer>
</edmx:Edmx>
```



# Kódgenerálás és EDM

---

- Az EF automatikusan osztályokat generál az entitástípusoknak
- A modellt ezeken a generált osztályokon keresztül tudjuk manipulálni
- A modell későbbi változtatásai közvetlenül kihathatnak a generált osztályokra



# Demó

---

EDM séma

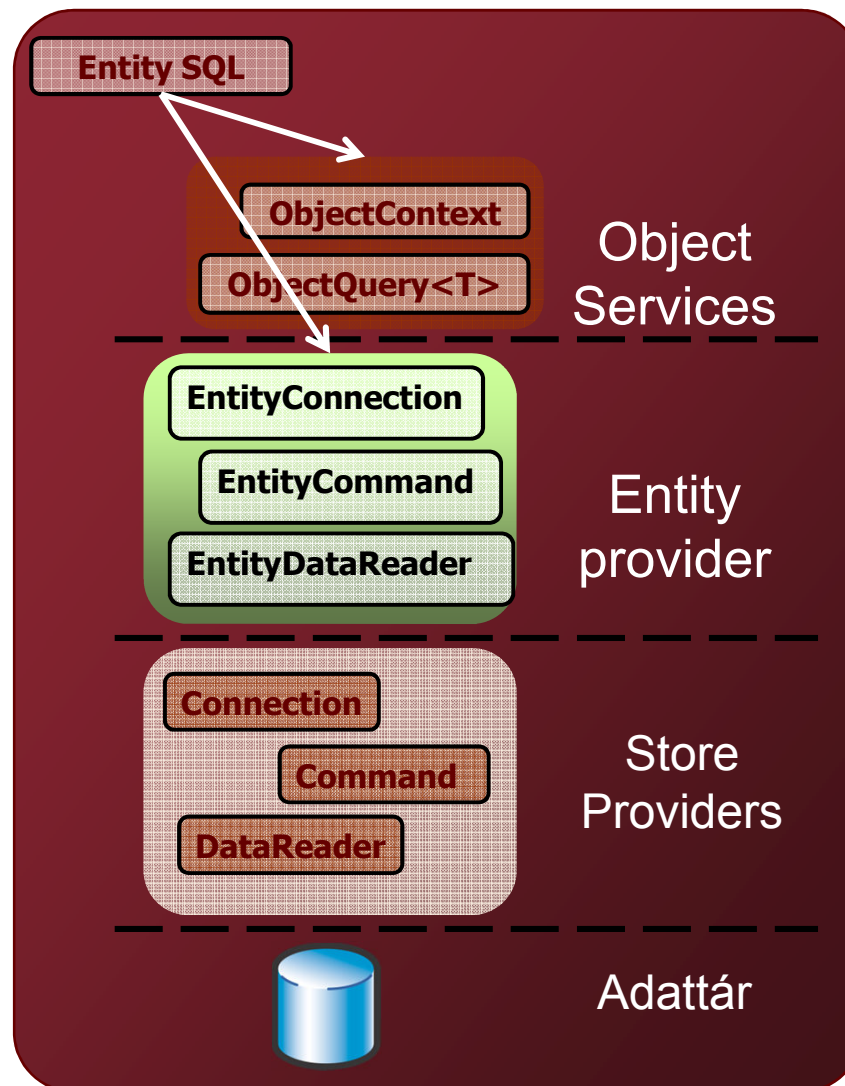
Kód generálás



# EDM működése

## ➤ObjectContext

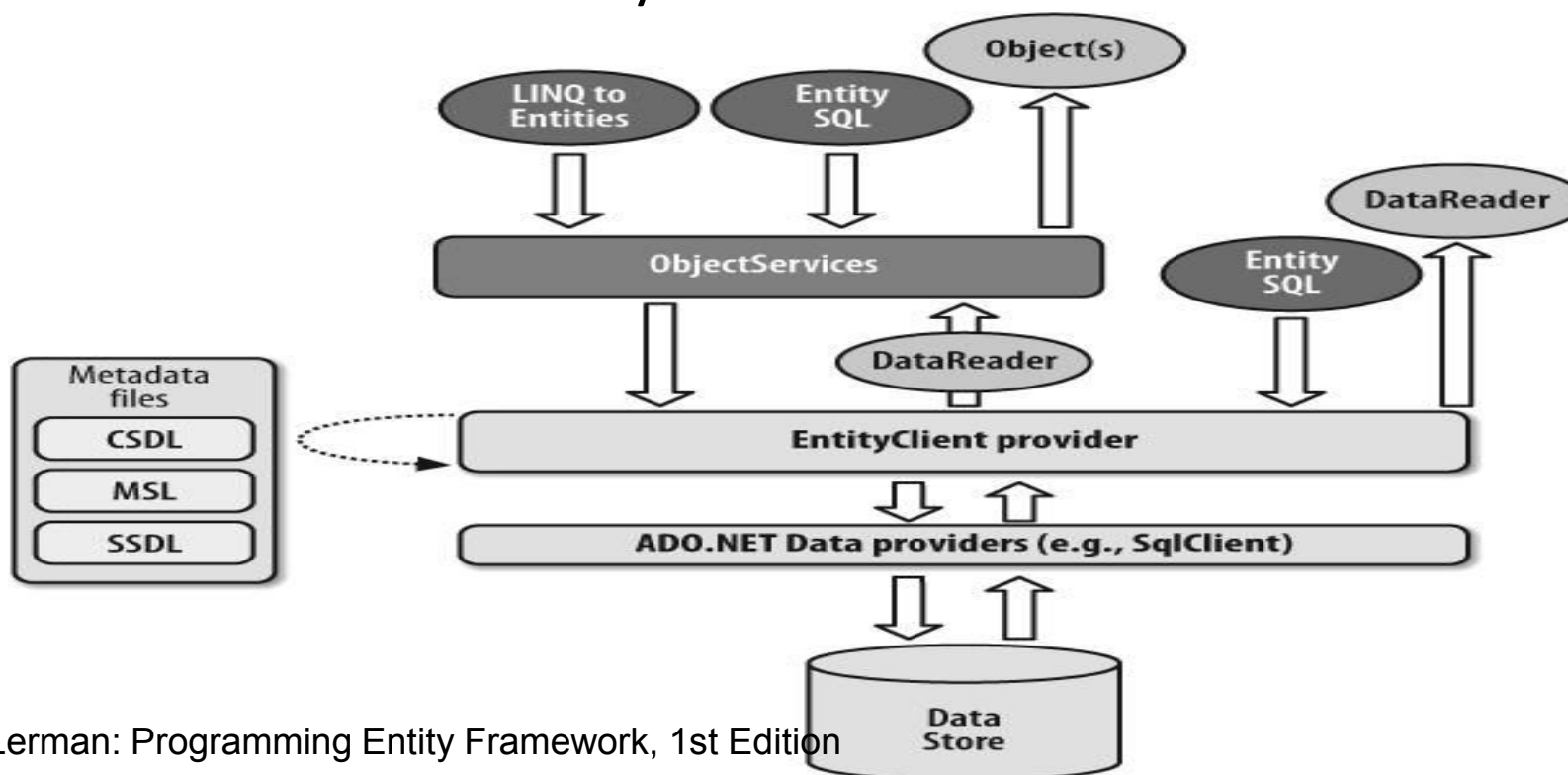
- Közös belépési pont
- Ismeri a sémákat és a leképezést
- Kapcsolódik a tényleges adattárhoz
- Állapotkezelést végez
  - Változáskövetés, objektumazonosítás





# Adatelérés

- A lekérdezéseket az adatmodellen fogalmazzuk meg
- Lekérdezés a modellen → adatbázis-specifikus lekérdezés
- A lekérdezés eredményei entitásokként materializálódnak



Julia Lerman: Programming Entity Framework, 1st Edition



# Lekérdezések készítése

---

## ➤ Lehetőségek

- LINQ to Entities
- Entity SQL + Object Services
- Entity SQL + Entity Client



# LINQ to Entities

---

- LINQ to SQL-ből megszokott szintakszis
- IntelliSense támogatás
- A lekérdezések Entity SQL lekérdezéseként értelmeződnek

```
var courses = from course in context.Courses
               where course.Title.StartsWith("C")
               orderby course.Title ascending
               select new
               {
                   Title = course.Title,
                   Location = course.Location
               };
```



# Entity SQL

---

- T-SQL-szerű lekérdezési nyelv
- EF az Entity SQL-t adatbázis specifikus lekérdezésre fordítja

```
var qStr = @"SELECT VALUE c
                FROM SchoolEntities.Courses AS c
                WHERE c.Title='Calculus'";

var courses = context.CreateQuery<Course>(qStr);
```



# Entity Client lekérdezés

## ➤ EntityDataReader-t használ

### ■ az objektumok entitásként materializálódnak

```
using (var conn = new EntityConnection("name=ProgrammingEFDB1Entities"))
{
    conn.Open();
    var qStr = "SELECT VALUE c FROM SchoolEntities.Courses AS c ";
    var cmd = conn.CreateCommand();
    cmd.CommandText = qStr;
    using (var rdr = cmd.ExecuteReader(CommandBehavior.SequentialAccess))
    {
        while (rdr.Read())
        {
            Console.WriteLine(rdr.GetString(1));
        }
    }
}
```



# Lekérdezési módok összehasonlítása

|                       | LINQ to Entities | Entity SQL with Object Services | Entity SQL with Entity Client |
|-----------------------|------------------|---------------------------------|-------------------------------|
| LINQ support          | √                | –                               | –                             |
| IntelliSense          | √                | –                               | –                             |
| Model dynamic queries | –                | √                               | √                             |
| Return type           | Objects          | Objects or DbDataRecords        | DbDataReader                  |
| Performance           | √                | √√√                             | √√                            |



# Melyiket használjuk?

---

- Entity SQL + Entity Client
  - Az adatokat sorban (streamelve) szeretnénk lekérdezni
  - Meglévő alkalmazások átírásakor
- Entity SQL + Object Services
  - LINQ-nél nagyobb kifejezőerőre van szükség
  - Dinamikusan összeállított lekérdezésekre van szükség
  - A teljesítmény nagyon fontos
- LINQ to Entities
  - Minden más esetben



# Demó

---

## Lekérdezések



# Változáskövetés

---

- Az entitások változásait azObjectContext tartja nyilván.
- ObjectContext referenciát tart minden példányosított entitásra
- Az adatváltozások visszaírása az adatbázisba a SaveChanges metódussal történik

```
context.SaveChanges();
```

OR

```
// doesn't refresh the entities state after save  
context.SaveChanges(false);
```



# Entitások létrehozása

## ➤ Entitás létrehozása a memóriában

- New operátorral
- Generált Create... metódussal

```
var department = new Department();  
var course = Course.CreateCourse(...);
```

```
course.Department = department;  
department.Courses.Add(course);  
context.AddToCourses(course);
```

## ➤ Entitás csatolása

- Létező entitás tulajdonságának értékül adva
- EntityCollection.Add(...)
- ObjectContext.AddTo(...)



# Entitások módosítása

---

- Tulajdonság / hivatkozás módosítása
- A változást ObjectContext nyilvántartja
- SaveChanges meghívásával a változások átvezetődnek az adatbázisba



# Entitások törlése

---

- Csak betöltött entitást tudunk törölni
- `ObjectContext.DeleteObject(...)`
- `SaveChanges` hívás itt is szükséges

```
context.DeleteObject(course);
```



# Demó

---

## Adatmódosítás



# Öröklés

---

- Támogatott megoldások
  - Öröklési hierarchia egy táblába
  - Minden típus saját táblába
  - Minden valós osztály saját táblába



# Öröklés – egy hierarchia egy táblába

---

- Egy teljes öröklési fa minden eleme ugyanabba a táblába kerül
- Hasonló a LINQ to SQL-es modellhez
- Feltételes leképzésen alapul
  - Minden entitástípusnál meg van adva egy feltétel, mely alapján a típus beazonosítható (pl.: discriminator = "Person")
- EDM designer támogatja



# Öröklés – minden típus saját táblába

---

- Alapértelmezett
- Mapping generálódik
  - Táblából kiindulva
  - Entitásokból kiindulva
- EDM designer támogatja



# Öröklés – Minden valós osztály saját táblába

---

- Több táblába történő leképzése alapul
- Feltételes leképzésen alapul
- EDM Designer nem támogatja
  - Öröklődő attribútumok mappingjét manuálisan kell megadni



# Demó

---

## Öröklés



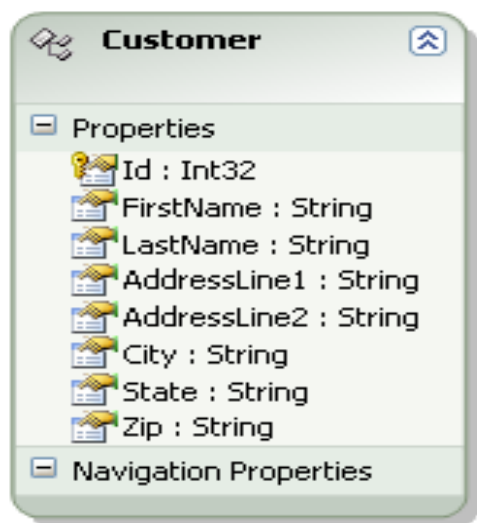
# Komplex típusok – 1

---

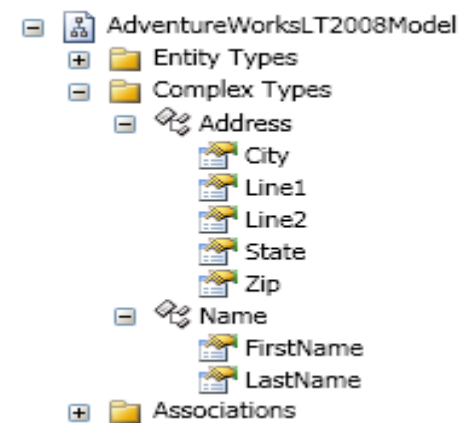
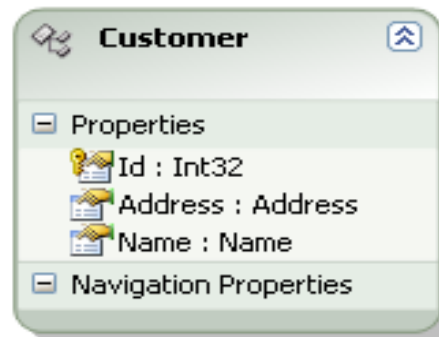
- Több skalár (vagy összetett) típusú érték vonható össze egyetlen komplex típusba
  - például cím = ország+város+utca+...
  - Összetett típus tartalmazhat másik összetett típust
- Az összetett típusok nem lehetnek üresek (null)
- Ha az összetett tulajdonságon belül bármelyik skalár érték megváltozik, az egész komplex property megváltozottá válik



# Komplex típusok – 2



VS.



1) 1:1 leképezés

2) Túlzsúfolt

1) Komplex leképezés

2) Rendezett



# Demó

---

## Komplex típusok



# DefiningQuery

---

- Az SSDL része
- Amikor egy SQL nézetet adunk az EDM-hez, automatikusan ilyen jön létre
- Csak olvasható entitásokat ad vissza
- Tárolt eljárásokkal a módosítást is lehetővé tehetjük
- Egyedi lekérdezéseket is készíthetünk
- Jobban testre szabhatók a visszaadott objektumok



# Lazy loading

---

## ➤ Explicit

- Hivatkozást betöltését kezdeményezni kell
- IsLoaded → Load

## ➤ Implicit

- Automatikusan történik
- Navigation propertyk mentén



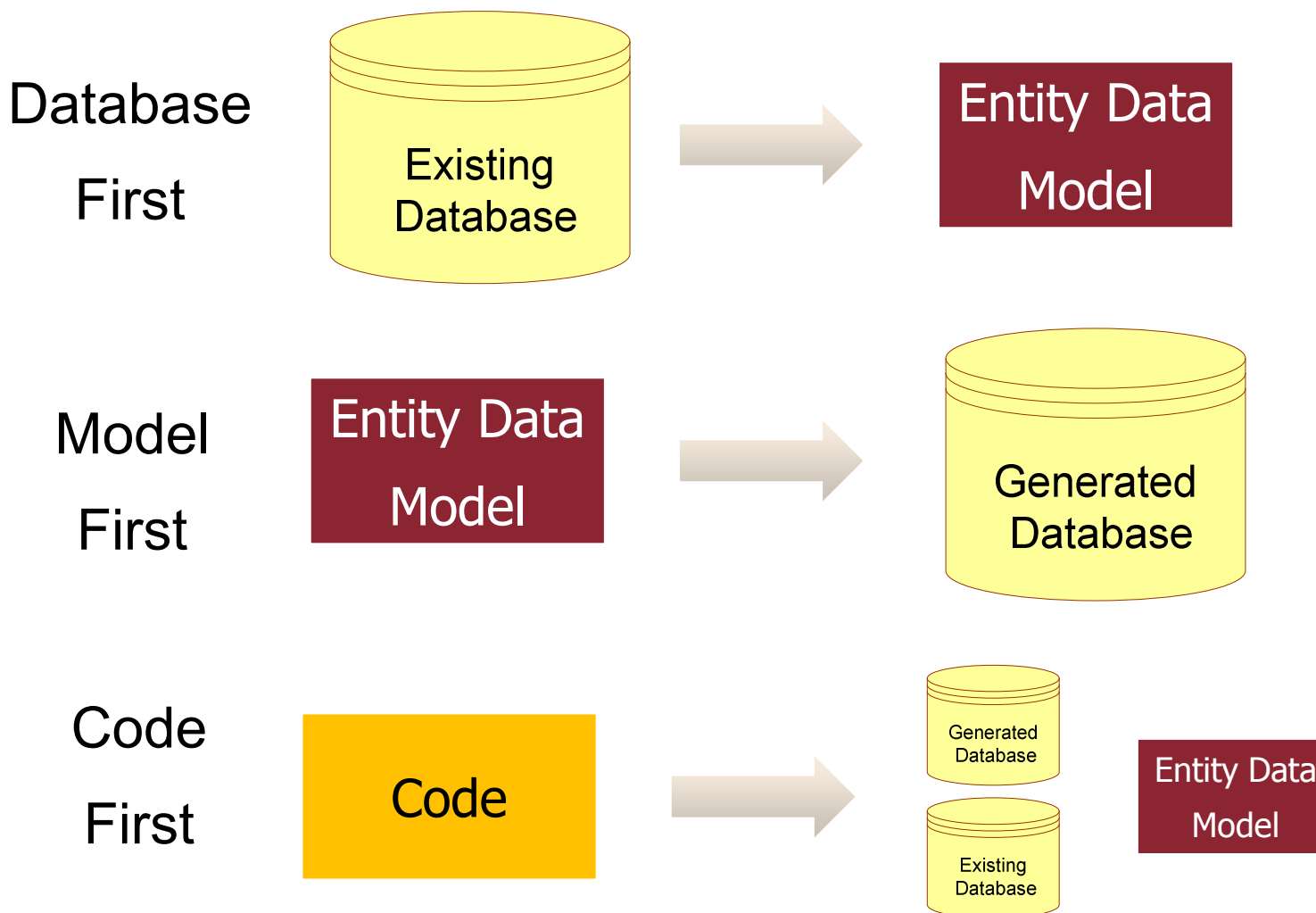
# Demó

---

## Lazy loading

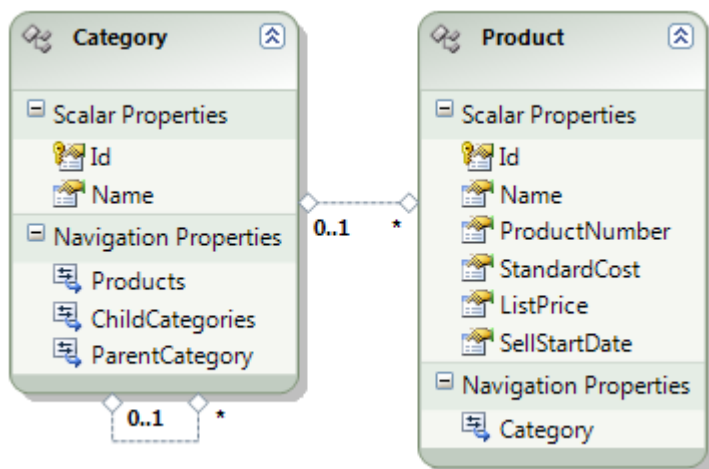


# Modellezés iránya

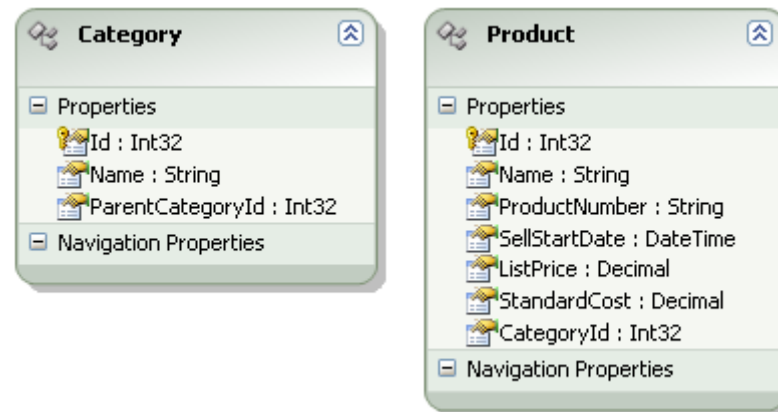




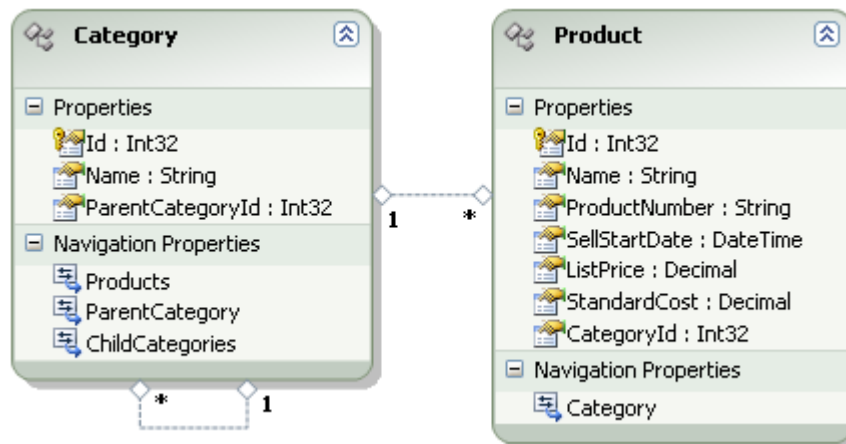
# Külső kulcsok



Elméleti megoldás



Gyakorlati igény



A megoldás



# Relációk

---

- Az entitásokat asszociációk kapcsolják össze
  - A asszociáció két végén két entitás van
  - Mindkét végponton meg kell adni a számosságot:
    - egy vagy nulla / egy / több
- Két fajta asszociációt támogat az EF
  1. **Külső kulcsos asszociáció:** az entitáshoz tartozik egy skaláris property ami az entitás asszociációhoz tartozó külső kulcsa
  2. **Független asszociáció:** az asszociáció csak entitás referenciaként jelenik meg az entitáson, külső kulcsként nem



# Külső kulcsos asszociáció

---

- Az asszociációt a külső kulcs módosításával (is) lehet változtatni
- Navigáció property tartozhat az entitásokhoz
- Az asszociáció megváltozása a függő entitás állapotát **Modifiedra** változtatja
- Megjegyzések
  - A több-többes kapcsolatokat mindig független asszociációk kezelik
  - Újonnan hozzáadott entitások esetén a külső kulcs beállítása nem szinkronizálja a referenciát, mert az entitásnak még csak ideiglenes kulcsa van



## Független asszociáció

---

- A külső kulcs nem érhető el közvetlenül
- Mindenképp tartozik navigációs property az entitáshoz
- Az asszociációt a navigációs property-n keresztül lehet megváltoztatni
- Az ObjectStateManager-ben minden független asszociációhoz külön ObjectStateEntry tartozik
- Az asszociáció megváltozása a függő entitás állapotát nem változtatja



# Tárol eljárások

---

- A tárolt eljárást leképezhetjük
  - ObjectContext függvényeként
    - Entitásokat is adhat vissza
  - Entitások adatmódosító függvényeire
    - Insert
    - Update
    - Delete



# Demó

---

## Tárolt eljárások



# Object Context szerepe

---

- Az adatbázis elérés központi osztálya
  - Adatbázis kapcsolat (EntityConnection)
  - Meta adatok, leképezés (MetadataWorkspace)
  - Objektumtár (ObjectStateManager)
- Nyilvántartja az entitásokat, hogy a rajtuk végzett változtatásokat le tudja küldeni az adatbázisba
- A generált ...Container osztály az OC-ból származik
- Mindig „using”-gal használjuk!



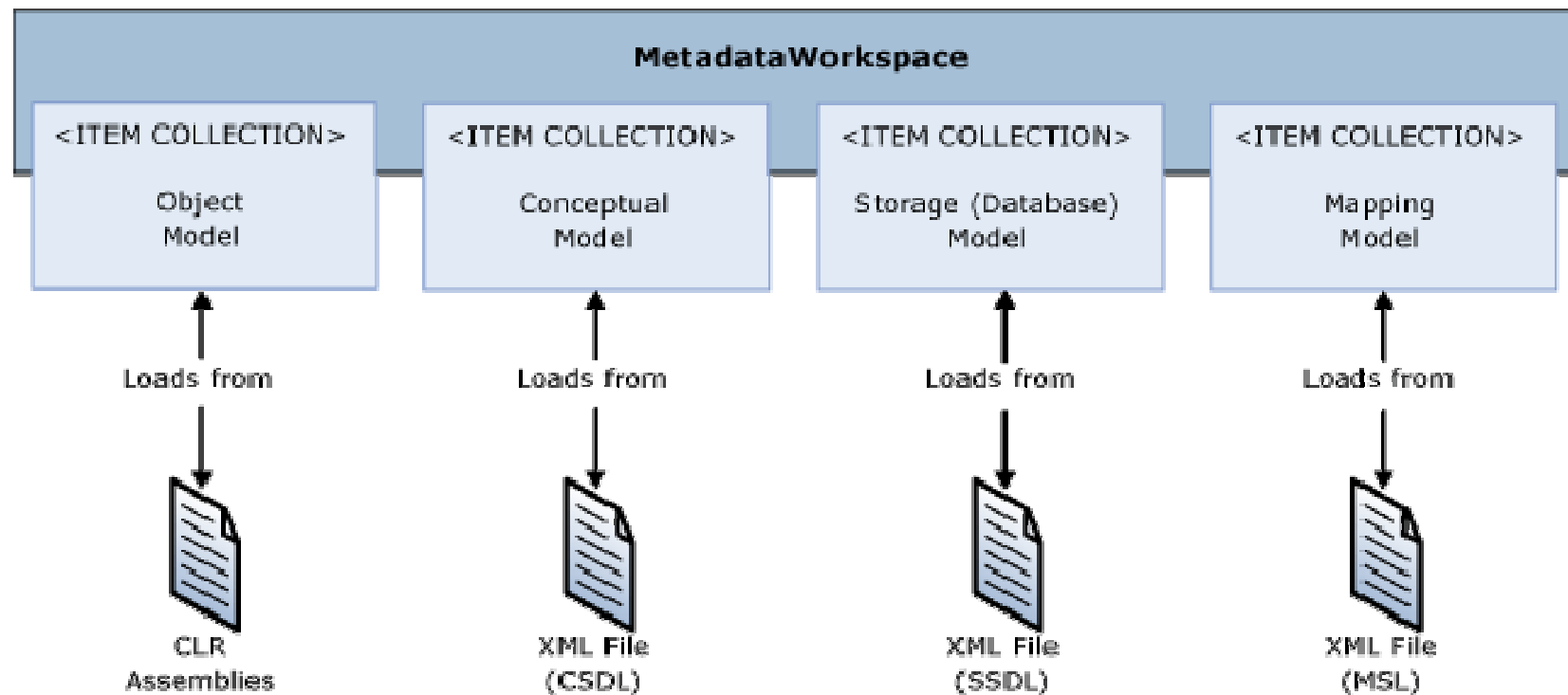
# Object Context – meta adatok

---

- MetadataWorkspace osztály
  - 3 szint kérdezhető le
    - objektum, koncepcionális, adatbázis
  - Entitások és tagjaik
  - Asszociációk
  - Felhasználható adattípusok
  - Funkciók
- A különböző szintekhez saját típusrendszer tartozik
  - Pl: nvarchar-string, image-Binary



# Object Context – meta adatok





# Object Context – ObjectStateManager

---

- ObjectStateManager – OC példányhoz tartozik
- Nyilvántartja a lekérdezett entitásokat
  - Összefűzi a részben átfedő eredményhalmazokat
  - További lekérdezéseknél figyelembe veszi a módosításokat – az adatbázisból felolvasott értékeket felülírja a tárban lévőekkel
- Listát vezet az újonnan felvett és törölt entitásokról
- Nyilvántartja az objektumokon történt változtatásokat



# Object Context – Kulcsok

---

- Minden entitásnak van egyedi kulcsa
  - EntityKey osztály reprezentálja
  - IEntityWithKey interfészen keresztül lekérdezhető
    - EntityObject implementálja az interfészt
  - A conténer és tábla név mellett több property adható meg (összetett kulcs esetén)
- Ha van kulcs, akkor
  - Lehet csatolni az entitást az OC-hez
  - Le lehet kérdezni az entitást az adatforrásból vagy az OC-ból



# Eredményhalmazok összefűzése

- **NoTracking:** az OC nem tartja nyilván a betöltött entitásokat
- **AppendOnly:** csak azok az entitások kerülnek bele az OC-be, amelyek eddig nem voltak benne. Alapértelmezett
- **OverwriteChanges:** a már meglévő entitások felülíródnak a betöltöttekkel, állapotuk **unchanged** lesz.
- **PreserveChanges:**
  - Unchanged entitások felülíródnak.
  - Modified entitásoknál a nem módosított mezők *eredeti* értéke átíródik és a property állapota modified lesz ha az adatbázis értékek különböznek a jelenlenlegitől. A módosított mezők nem íródnak felül.



# Változáskövetés – 1

---

- Az ObjectContext-ben minden entitáshoz tartozik egy állapotleíró objektum (ObjectStateEntry)
- Az objektumban bekövetkezett változásokat ezen állapotleíró alapján tudja beazonosítani és nyomon követni az Entity Framework



## Változáskövetés – 2

---

- **ObjectStateEntry**: egy entitás (vagy reláció) állapotát tartja nyilván az OC-ben
  - EntityKey – azonosítja az entitást
  - EntityState – leírja az entitás állapotát (lásd később)
  - Eredeti tulajdonságok értékei
  - Módosított tulajdonságok listája
  
- **DBDataRecord**: adatmezőket tároló osztály
  - Mezőnevek, értékek
  - GetInt32/SetInt32, stb.
  - Egy osztály értékeinek nem típusos másolata



# Változáskövetés – EntityState

---

- **Detached:** az objektum nem tartozik OC-hez.
- **Unchanged:** az entitást nem módosították a hozzárendelés vagy utolsó mentés (SaveChanges) óta.
- **Added:** új entitás, eredeti értékei nem lekérdezhetőek.
- **Deleted:** az entitást törölték az OC-ből.
- **Modified:** az entitást módosították az OC-hez rendelés óta és még nem hívtak SaveChangest.



# Állapotkövetés – Entitás típusok

---

- Az állapotkövetés megkönnyítésére 4 féle entitás típus közül választhatunk:
  - EntityObject
  - POJO
  - POJO Proxy
  - Self Tracking Entities



# Entitás típusok

---

1. **EntityObjectból származó entitások (alapértelmezett)**
  - Az OC automatikusan végzi a változás követést, támogatja a lazy loadingot
2. **POCO (Plain-Old CLR Object)**
  - Nincs automatikus változás követés, DetectChanges metódust kell hívni, nincs lazy loading támogatás
3. **POCO Proxy**
  - POCO automatikus változáskövetéssel és lazy loading támogatással
  - Speciális követelmények a POCO osztályokra
4. **Self-Tracking Entities**
  - Template alapú, generált megoldás
  - Nem függ az EF-től



# POCO Entitások

---

## ➤ Változáskövetés

- Nem automatikus, snapshot alapú változás detektálás
- Amikor az entitás csatolásra kerül, az OC lekérdezi és tárolja a tulajdonságainak értékeit
- A **DetectChanges** metódus hívás összeveti az entitásokhoz tárolt eredeti értékeket és az aktuális propertyket
  - Közben felveszi az új entitásokat is

## ➤ Szinkronból kiesett entitásokon végzett műveletek hibás működést okozhatnak



# POCO Proxy Entitások

---

## ➤ Követelmények

### ■ Általános:

- Publikus, nem absztrakt és nem sealed osztály
- Public/protected paraméter nélküli konstruktor
- Nem implementálhatja az IEntityWithChangeTracker és az IEntityWithRelationships interfészeket

### ■ Késleltetett betöltéshez:

- Érintett tulajdonság (get): public, virtual, non sealed

### ■ Változáskövetéshez:

- Érintett tulajdonság (get, set): public, virtual, non sealed
- Relációknál a \* oldalat ICollection<T> reprezentálja
- OC **CreateObject** függvényét használjuk **new** helyett



# Self-tracking Entitások

---

- A programozási modell azonos a hagyományos EF-fel, a generált osztályok a következő állapotinformációkat hordozzák magukban:
  - Státusz (Added, Deleted, Modified, Unchanged)
  - A relációik eredeti értékei
  - A relációkon végzett hozzáadás és törlés műveletek



# Új entitás felvétele – 1

---

- Amíg az entitás csak az OC-ben létezik, addig a kulcs `IsTemporary` tulajdonsága `true`.
- Adatbázis által generált kulcsok esetén (tipikusan a mesterséges elsődleges kulcsok) a **StoreGeneratedPattern** tulajdonság = `Identity`
  - `Computed`: update esetén is változhat
  - `None`: nem adatbázis által generált
- Az `Identity` és `Computed` értékeket a `SaveChanges` hívás után az OC frissíti



## Új entitás felvétele – 2

---

1. Új entitás létrehozása, a kulcs még üres
2. Az entitás OC-hez rendelése, ideiglenes kulcs
  - AddObject hívás vagy a reláción megfelelő oldalán egy új elem hozzáadása
3. SaveChanges hívása az OC-n
4. EF lefuttatja az INSERT SQL-t
  - Az új értékek beírása az OC-be, a megfelelő ObjectStateEntry-kbe
5. A friss értékek bekerülnek az OC-hoz tartozó entitásokba is



# ObjectContext – SaveChanges

---

- Elmenti a változtatásokat az adatbázisba
  - Csak azokról tud, amely entitások az OC-hoz vannak rendelve
- A metódus egyetlen nagy tranzakcióban dolgozik
- A sikeres hívás AcceptAllChanges-zel zárul
  - Az OC változás követése alaphelyzetbe áll: minden entitás állapota **Unchanged**re változik



# ObjectContext – néhány tulajdonság

---

- Az OC nem szálbiztos
- Példányosított OC tipikusan nyitott adatbázis kapcsolatot jelent
- Az OC objektum tár nem arra lett tervezve, hogy nagy mennyiségű objektumot hatékonyan kezeljen (nem objektum adatbázis)



# ObjectContext – tipikus használat

---

- Az OC-t egyetlen funkció lefutásához példányosítsuk, majd engedjük el
  - Nincs konkurencia probléma
  - Nincs sok objektum
  - Az adatbázis kapcsolat minimális ideig van nyitva
- Használjunk „using”-ot
- Amikor a lekérdezett objektumokat nem akarjuk módosítani, használjuk a NoTracking opciót



# Két és többretegű alkalmazások

---

## ➤ Kétretegű alkalmazás (GUI + DB)

- Az entitások nem hagyják el az OC alkalmazástartományát (nincs sorosítás)
- Az állapotkezelést az OC megoldja ha a funkciók kompaktak (például egy dialógus ablak)

## ➤ Többretegű alkalmazás

- Az entitás kikerül az OC alkalmazástartományából
- Az állapotkövetés nem bízható az OC-ra
- A változásokat az OC tudomására kell hozni



# Többrétegű alkalmazások

---

- Használjunk állapotmentes szolgáltatást
- Minden kéréshez hozzunk létre új OC-t
- A kliens szolgáltatson minden adatot a módosításokhoz
  - Például az új objektummal együtt az eredeti objektumot is, de legalább az időbélyeget



# Többrétegű alkalmazások – sorosítás

---

- Lazy loading-ot ki kell kapcsolni: a sorosítás bejárja és mindent lekérdez.
- A bináris és WCF datacontract alapú sorosítás a relációval kapcsolódó és az OC-ban létező objektumokat is sorosítja, az XML sorosítás nem
- Entitás státusz, változás követési információk nem sorosítódnak, csak az objektum aktuális állapota



# Leválasztás

---

- Entitások leválasztása az OC-ről
  - Sorosítás
  - ObjectSet / OC . Detach( object )
  - Relációban kapcsolódó objektumokat nem választja le
  - A GC összegyűjtheti a leválasztott objektumot, az ObjectStateManager nem hivatkozik rá tovább



# Csatolás

---

## ➤ Entitások csatolása az OC-hez

- Egy entitás egy időben egyetlen OC-hez lehet rendelve
- ObjectSet/OC . Attach( IEntityWithKey )
- OC . AttachTo( string entitySet, object entity )
- ObjectSet/OC . AddObject
- Lekérdezések NoTracking opció nélkül



# Csatolás utáni státusz beállítása

---

- Minden felcsatolt entitás **unchanged** státusszal kerül bele az OC-be
  - A SaveChanges így még semmit nem ír az adatbázisba
- ChangeObjectState( entity, state )
  - Megváltoztatja a csatolt entitás (és a hozzá tartozó relációk) státuszát
  - Módosítás esetén minden tulajdonsága módosítottá válik
- SetModifiedProperty( entity, propertyName )
  - Egyesével lehet módosítotttra állítani a tulajdonságok státuszát



# Tulajdonságok állítása

---

- `ApplyCurrentValues( currentEntity )`
  - A OC-ban megkeresi az azonos kulccsal rendelkező csatolt entitást. A *currentEntity* detached állapotú.
  - Beleírja az *currentEntity* tulajdonságainak értéket a csatolt entitásba és módosítottként megjelelőli a propertyket, amik különböztek az *eredeti* értékektől.
- `ApplyOriginalValues( originalEntity )`
  - A OC-ben lévő azonos kulcsú entitás eredeti értékeit módosítja, ami változás a jelenlegihez képest, azokat módosított státuszba állítja



# Többrétegű alkalmazások – állapotkövetés

- Többrétegű alkalmazás
  - Az entitás kikerül az OC alkalmazástartományából
  - Az állapotkövetés nem bízható az OC-ra
  - A változásokat az OC tudomására kell hozni
- 3 mintamegoldást mutatunk, de természetesen további lehetőségek is vannak



# Megoldás – 1

---

- A kliens csak a módosított objektumot adja át
- **Feltételezzük, hogy minden tulajdonság megváltozott**
- Attach( modifiedEntity )
- ChangeObjectState( modifiedEntity, Modified )
- SaveChanges
- Ez minden tulajdonságot leküld az adatbázisba



## Megoldás – 2

---

- A kliens csak a módosított objektumot adja át
- **Nem feltételezhetjük, hogy minden tulajdonság megváltozott**
- Lekérdezzük az objektumot az adatbázisból
- ApplyCurrentValues( modifiedEntity )
  - Megjelöli, hogy mi változott
- SaveChanges
- Ez plusz egy adatbázislekérdezéssel jár



## Megoldás – 3

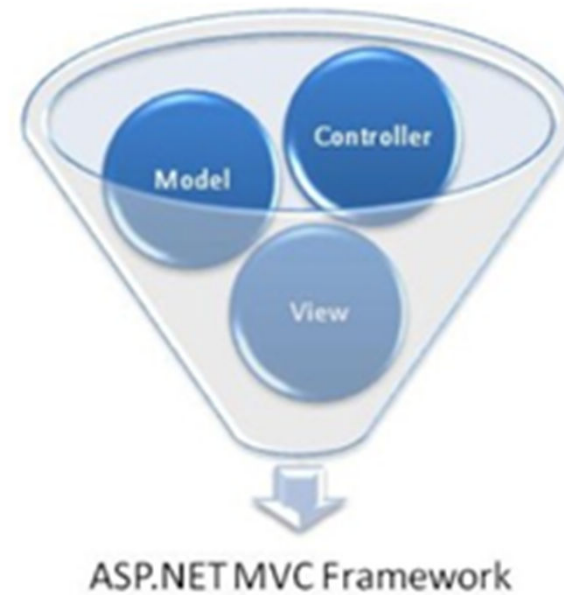
---

- A kliens a módosított és az eredeti objektumot is átadja
  
- Attach( original )
  
- ApplyCurrentValues( modifiedEntity )
  - Megjelöli, hogy mi változott
  
- SaveChanges
  
- Ez kommunikációs és interfész overhead



# ASP.NET MVC

---

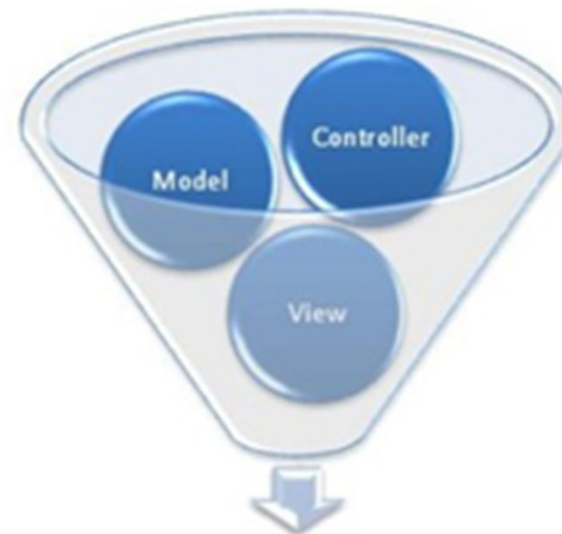




# Tartalom

---

- ASP.NET WebForms és ASP.NET MVC
  - Történelem, hasonlóságok és különbségek
- Pontosan mit jelent itt a
  - Model, View, Controller
- Hogyan lesz az URL-ből metódus hívás
  - REST, Routing, Area
- Különböző View enginek használata
- Egyedi helperek készítése
- Globális filterek
- ASP.NET MVC alkalmazások tesztelése
  - Egység teszt készítése



ASP.NET MVC Framework

# ASP.NET MVC

---

## ASP.NET WebForms és ASP.NET MVC

- Történelem
- Hasonlóságok
- Különbségek

# ASP.NET verziók

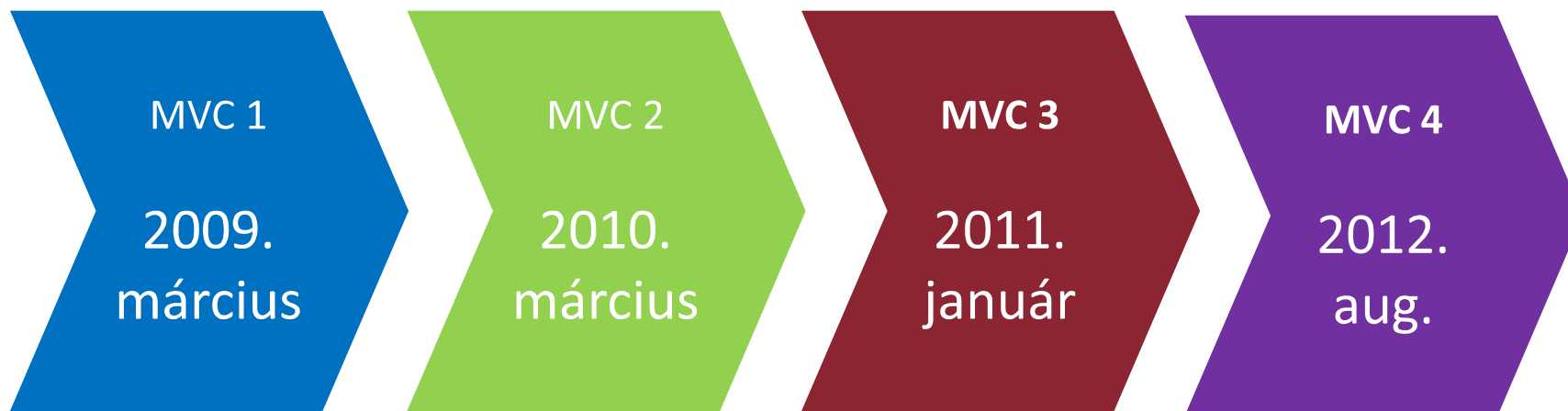


- Dinamikus webalkalmazások fejlesztése
- ASP.NET MVC, Web API, Entity Framework, LINQ
- NET által támogatott nyelv használható (C#, VB...)
- LINQ Query, IQueryable, IQueryableDataSource
- ASPX = HTML Markup + Code behind
- Login és navigációs vezérlők
- ListView, Wizard vezérlő továbbfejlesztése
- Kód és a markup szétválasztása
- Mester oldalak
- Html kódolásra expression
- Markup és skin



# ASP.NET MVC történelem

---



Web Platform Installer: <http://www.microsoft.com/web>

Nyílt forráskód: <http://aspnet.codeplex.com>

Microsoft Source License



# WebForms vs MVC

---

- Nem az ASP.NET helyett van, hanem egy alternatíva.
- System.Web.Mvc
- Egy projekt felépítése nagyon hasonló:
  - Van web.config
  - Mester oldal (Razor: Site layout)
  - Beépített jogosultságkezelés használható
  - ASP.NET után nagyon hamar meg lehet szokni!
    - ASPX view enginnel, Razort kicsivel nehezebb



# WebForms vs MVC

---

## ASP.NET WebForms

- Vezérlők
- Eseménykezelés
- Adatbevitel
- Markup generálás
- UI állapotmegőrzés
- Magasabb absztrakciós szint
- RAD

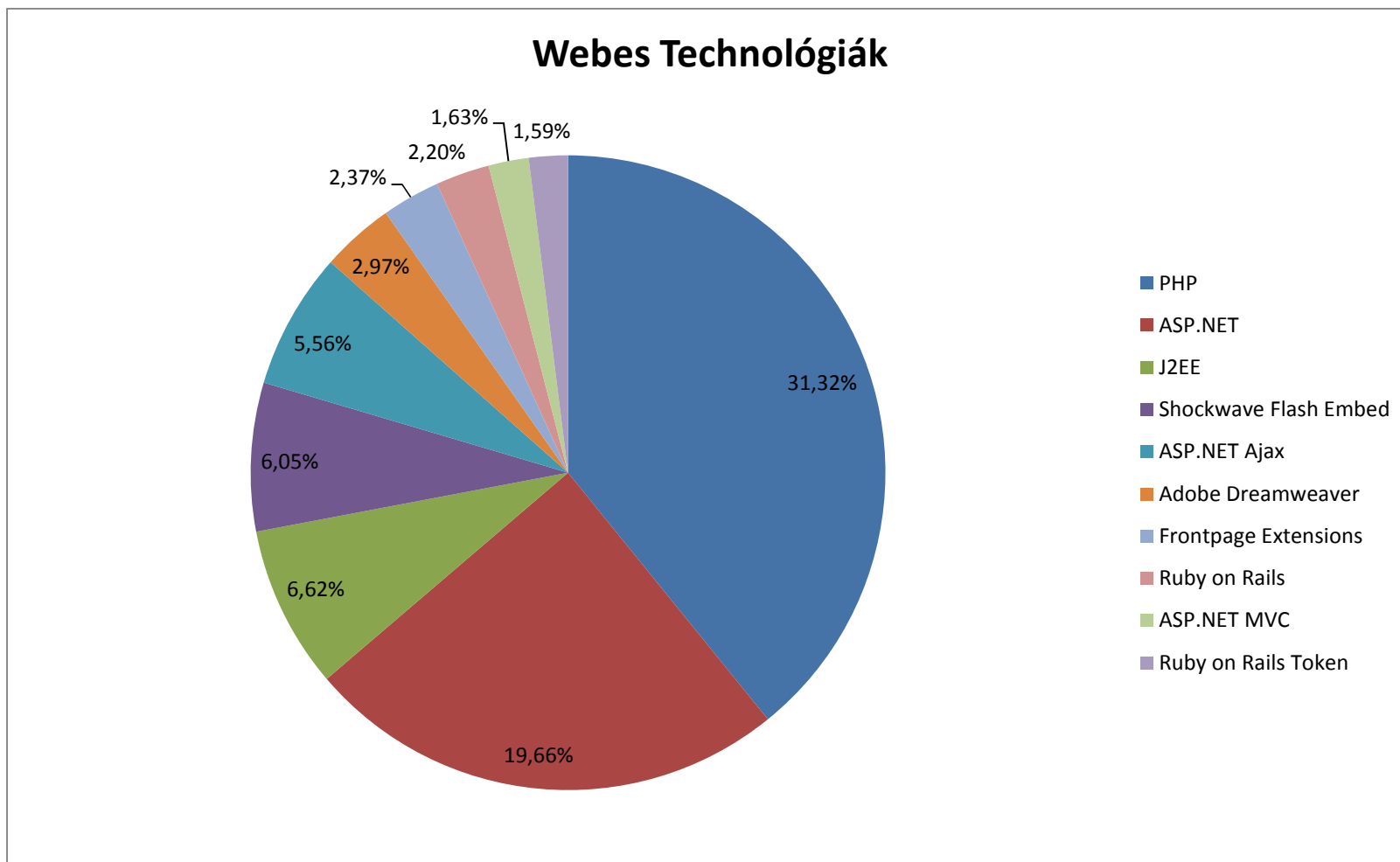
## ASP.NET MVC

- Kevesebb „varázslás”
- Teljes kontroll a markup felett
- Funkciók szétválasztása
- Tervezési minták
- Kiterjeszthetőség
- Tesztelhetőség
- Többféle view engine

**Közös ASP.NET platform szolgáltatások**



# Webes technológiák elterjedtsége



Forrás: <http://trends.builtwith.com/> (2013.)



# DEMÓ

---



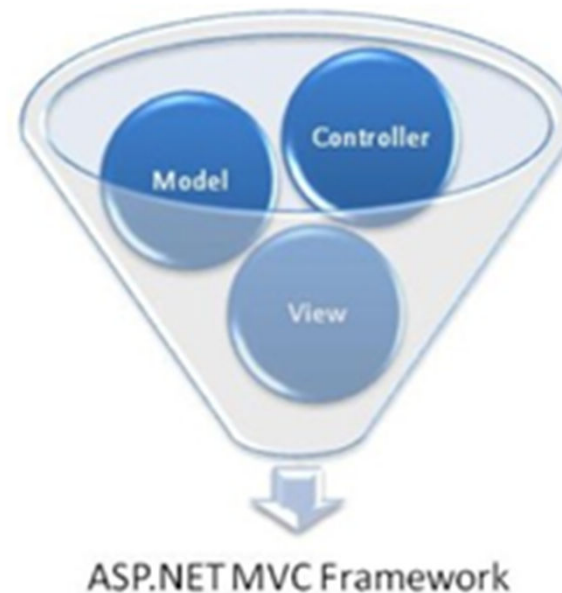
## MCV példaalkalmazás megtekintése

- Project felépítése
- Model, View, Controller hol helyezkedik el
- Közös szolgáltatások
  - FormsAuthentication és Membership



# ASP.NET MVC

---

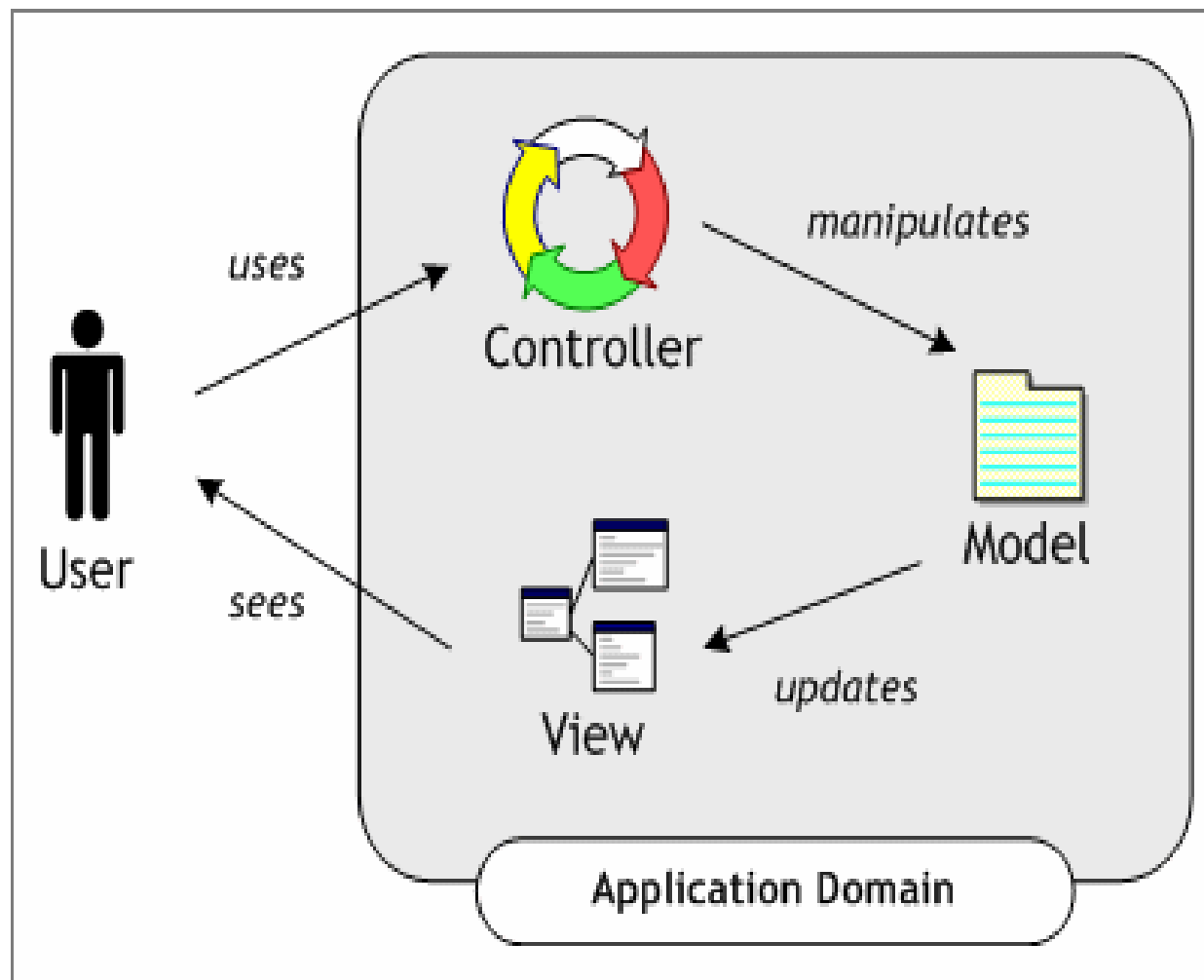


Pontosán mit jelent a

- Model
- View
- Controller

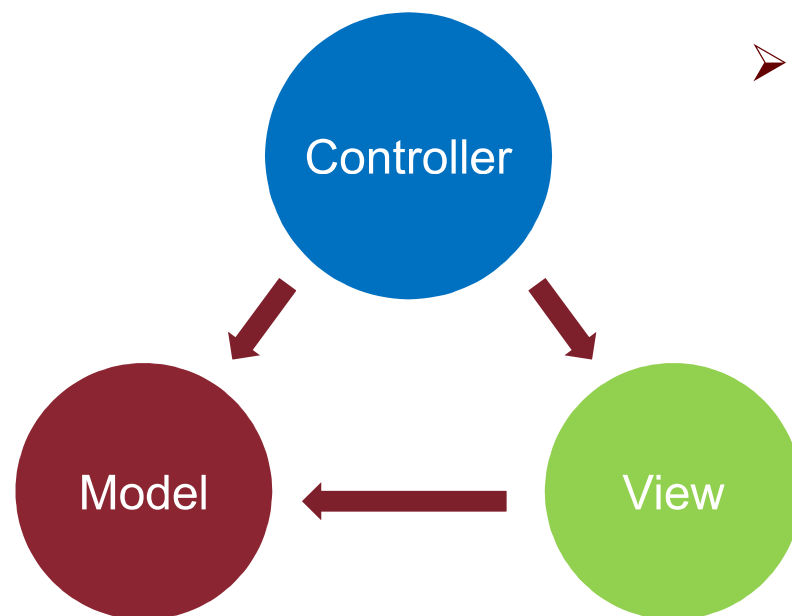


# MVC architektúra





# Mitől MVC



## ➤ Controller

- Beérkező kérések feldolgozása
- View paraméterezése
- Model módosítása

## ➤ Model

- Alkalmazás adatai
- Üzleti szabályok
- Validáció

## ➤ View

- Razor (.cshtml)
- ASPX (.aspx)
- CodeBehind = Controller



# Controller és Action

---

- Amire URL mappelhető
  - <http://mycdshop.hu/categories/details/blues>
  - CategoriesController.Details( "blues" )
- Egy kérés feldolgozásának belépési pontja
  - A kérést kezelő osztály = **Controller**, metódus = **Action**( *paraméterek* )
- Módosítja a modellt
- A Controller paraméterezi a View, hogy vissza tudja küldeni a választ a kliensnek.
- A kliensnek küldött válasz lehet akár string vagy JSON is nem csak HTML



# View

---

- Sablon a válasz előállításához
- „Üres helyeit” a Controller tölti ki a **Model** alapján
- Több különböző sablon szintaxis (ViewEngine)
  - Razor (.cshtml)
  - ASPX (.aspx)
- Kényelmes és pontos HTML előállítás
- Lehet erősen típusos, vagy típus nélküli



# Model

---

- Az adat amire a View-ban hivatkozhatunk
- Üzleti logika állítja elő.
  - Egyszerű esetekben megegyezhet az EF objektumokkal
  - Általában azonban kifejezetten a View-hoz készül
- Adatok validációja a modellhez kötődik
- A ViewBag-ben adhatunk át olyan adatokat, ami nem része a modellnek, de el kell érni a View-n.



# Tisztaság

---

## ➤ Tiszta URL

- Testreszabható URL képzés (Routing)
- Representational State Transfer [REST]
- Search engine optimization [SEO]

## ➤ Tiszta HTML

- A kimenet strukturált és olvasható
- Teljes mértékben a mi kezünkben van



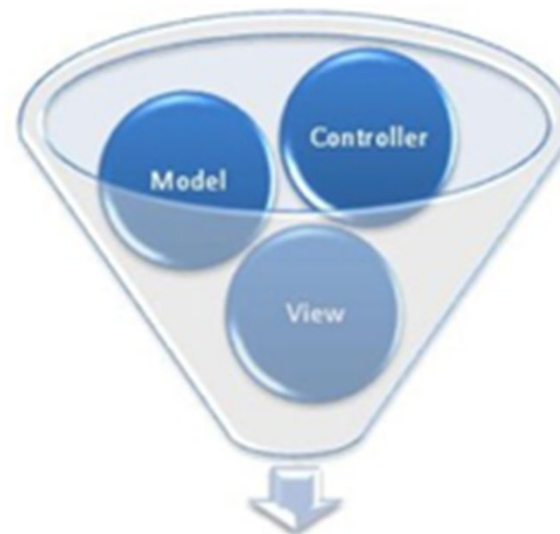
# DEMÓ

---



## Hogyan néz ki egy regisztráció

- Mit látunk a Controllerben (GET és POST)
- View megismerése
- Mi van a Model-ben



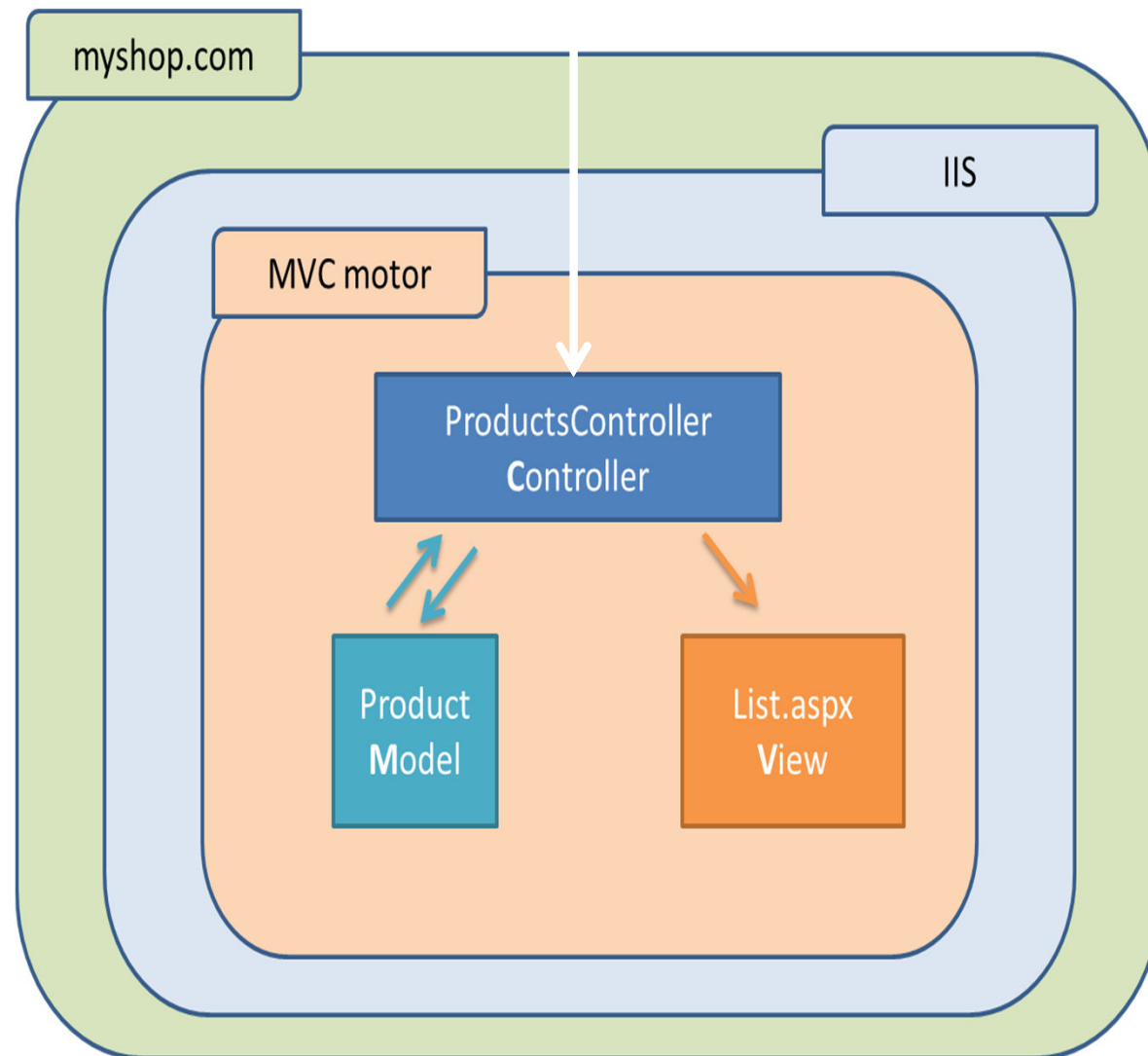
ASP.NET MVC Framework

# ASP.NET MVC

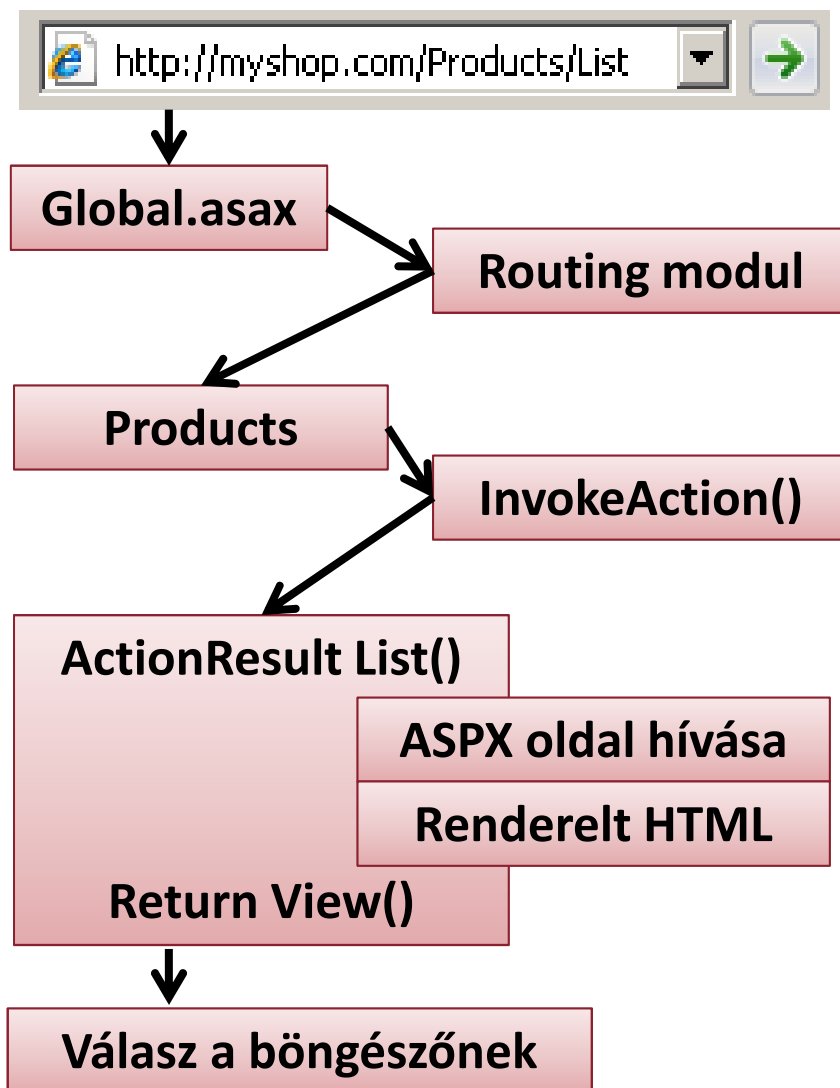
---

Hogyan lesz az URL-ből metódus hívás

- REST
- Routing
- Area



# Beérkező kérés feldolgozása



Beérkező kérés a **Products/List**-re

**URL → mely metódust kell hívni**

Controller: **Products**, Action: **List**

Megkeresi a Controller, megfelelő Actionjét.  
(paraméter nélküli, attribútum is)

Meghívja a **Prdoducts.List()**-et

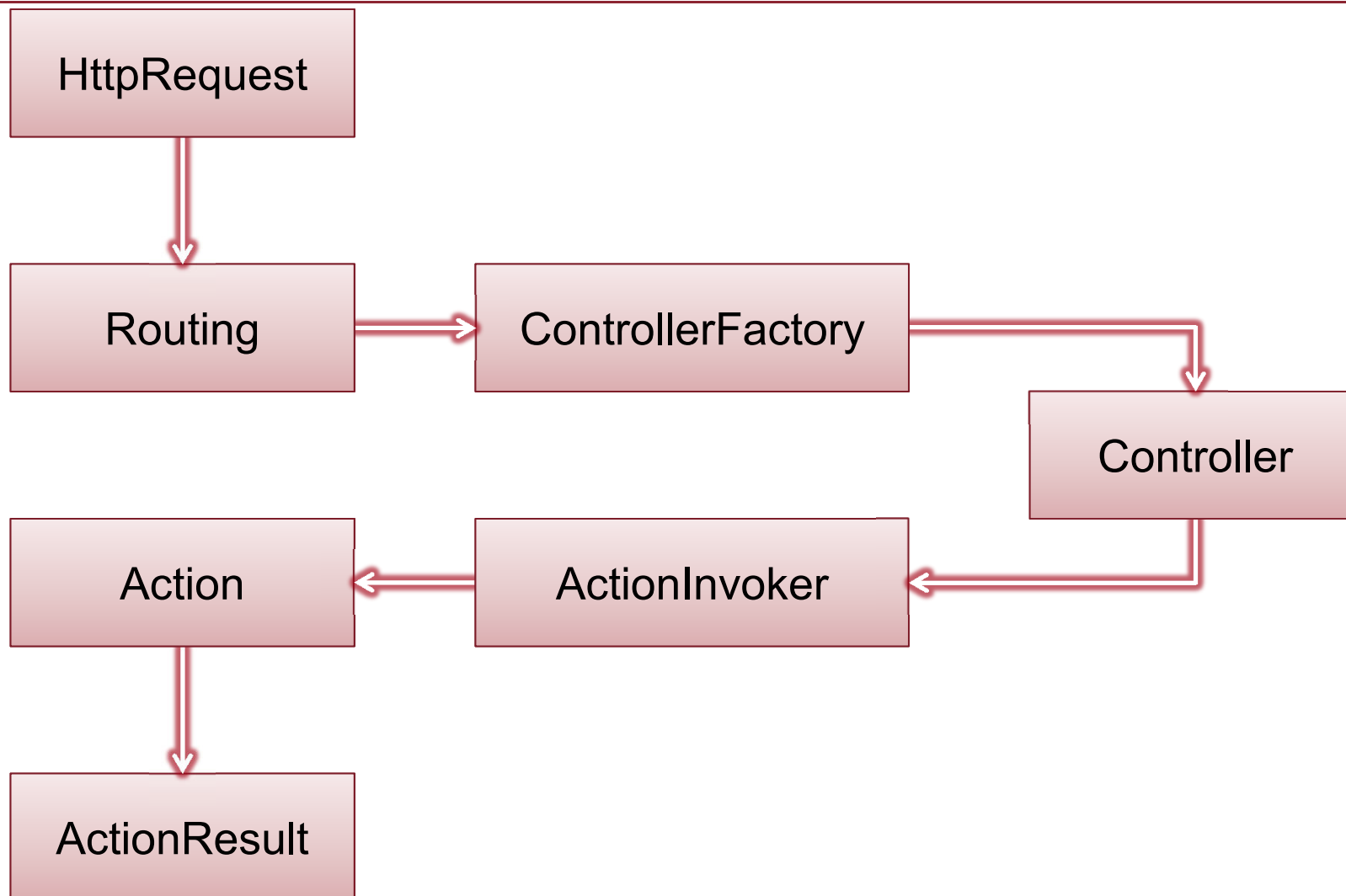
Action összeállítja a modell alapján a megfelelő Viewnak az adatokat

**Views/Products/List.aspx**, ami  
kirendereli a HTML-t amit visszaad az  
Actionnek

Action visszaküldi a HTML-t a  
böngészőnek



# Kérés élelciklus





# Hogy is van ez?

---

- Az alkalmazás API-ja = az elérhető URL-ek
- REST-esen elérhető minden Action
- Routing: URL → Controller/Action/Id megfeleltetés
  - URL → Osztály/Metódus/Paraméterek ami nem más, mint a Controller/Action/Id
- Külön Action a HTTP GET-re és POST-ra



# Hogyan működik a Routing

- Global.asax Application\_Start-ban regisztrálni
  - IgnoreRoute és MapRoute
  - Fontos a sorrend, constraintek is megadhatók

```
public static void RegisterRoutes(RouteCollection routes) {  
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");  
    routes.MapRoute(  
        "Default", // Név  
        "{controller}/{action}/{id}", // Url paraméterekkel  
        // Paraméter default értékei  
        new { controller = "Home", action = "Index", id = "" },  
        new { id = @"\d{1,6}" } // Megkötések  
    ); }  
}
```



# Irányítás egy URL-re

- A routing táblába URL → Controller + Action
- Visszafele irányban
  - Action
  - ActionLink

```
<%= Html.ActionLink("Red items", "List", "Products",  
new { color="Red", page=2 }, null) %>
```

```
<a href="/Products/List?color=Red&page=2">Red items</a>
```

- RedirectToAction

```
public ActionResult MyActionMethod()  
{ return RedirectToAction("List", "Products"); }
```



# Routing előre definiált beállításokkal

## ➤ Named routes

- Routing listához hozzáadni a sablont

```
routes.MapRoute("intranet",  
    "staff/{action}",  
    new { controller = "StaffHome" });
```

- Link megadása ASPX szintaxissal

```
<%= Html.RouteLink( "Katt",  
    "intranet",  
    new { action = "StaffList" } ) %>
```

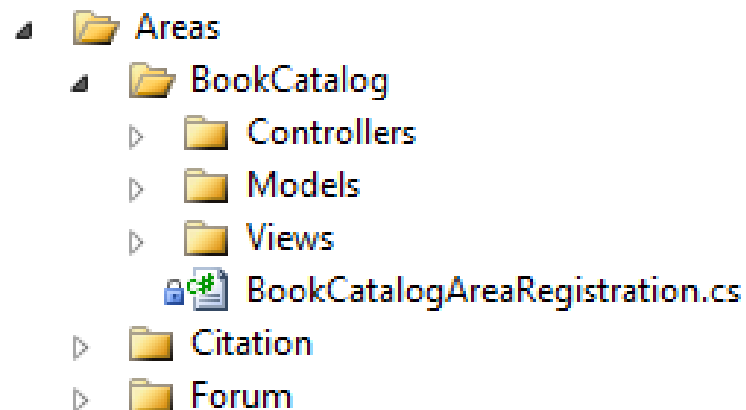
- Generált link

```
<a href="/staff/StaffList">Katt</a>
```



# Area

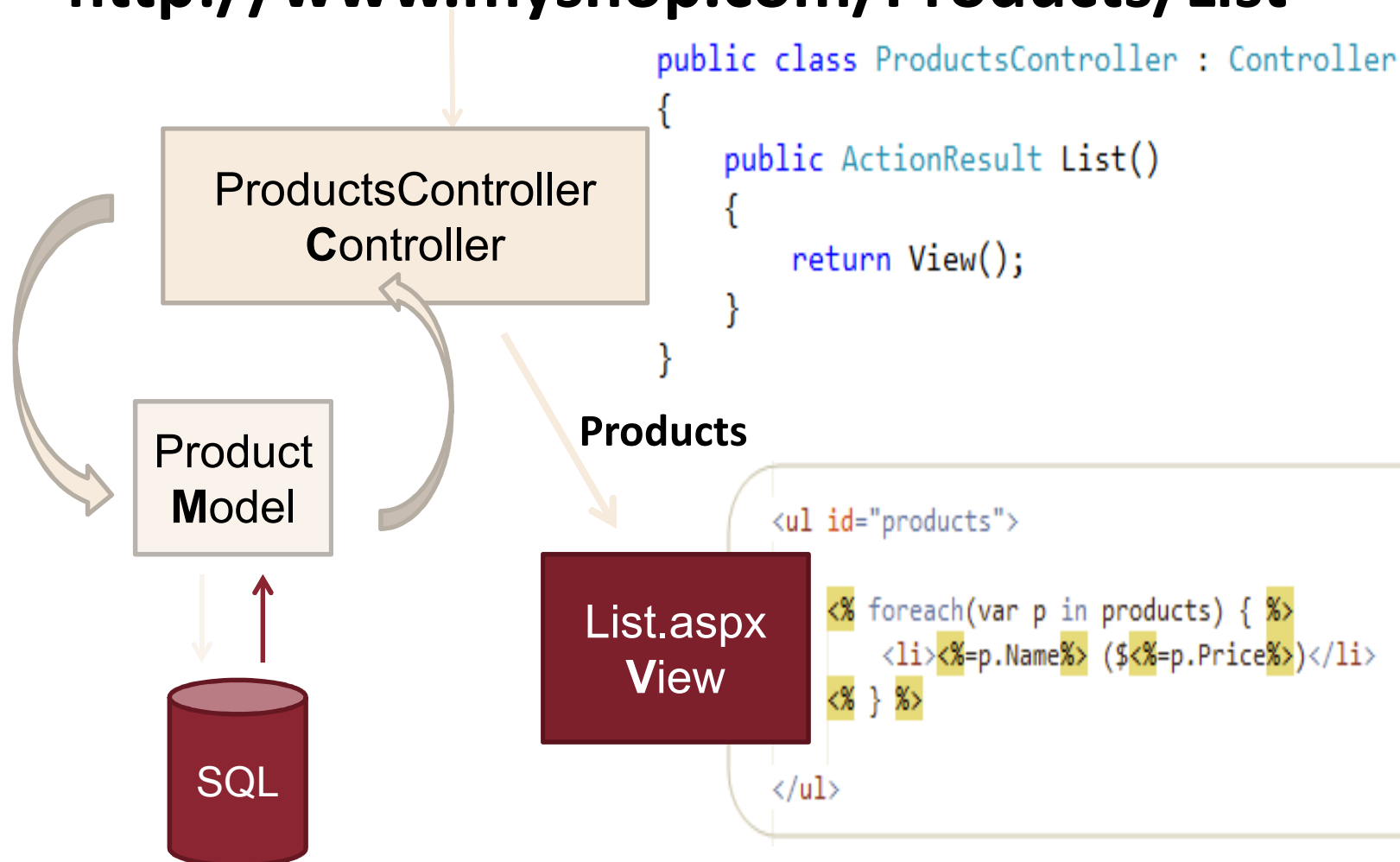
- Logikai szeparációs egység
  - Könyvtár szinten
  - Routing lista regisztrálás alapján
  - Külön web.config
- Beágyazott kis MVC alkalmazás





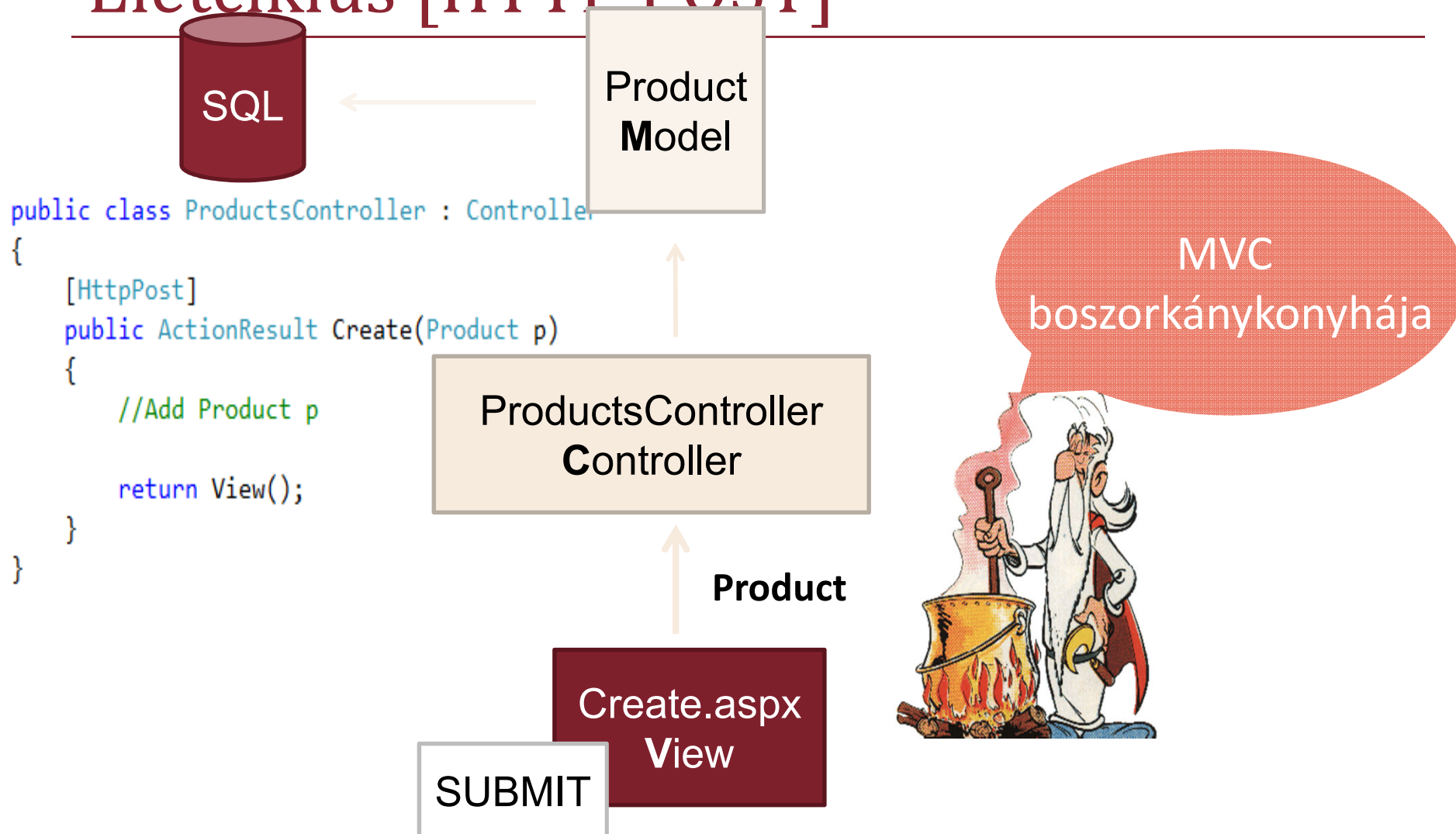
# Életciklus [HTTP GET]

<http://www.myshop.com/Products/List>





# Életciklus [HTTP POST]



<http://www.myshop.com/Products/Create>



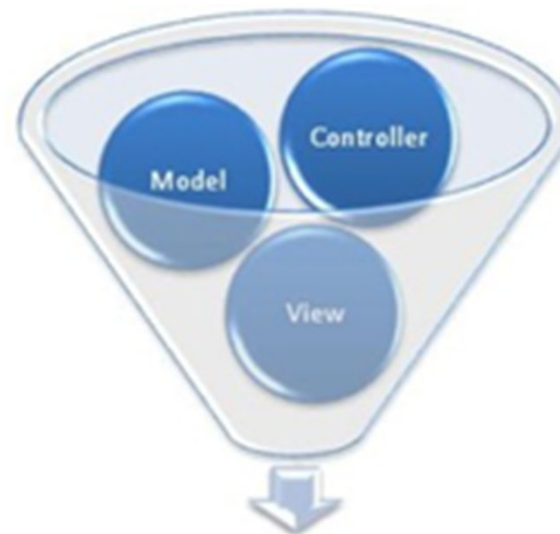
# DEMÓ

---



Debuggoljuk végig a regisztrációs folyamatot

- GET és POST Actionök vizsgálata.



# ASP.NET MVC

---

## Különböző View enginek használata

- Razor
- ASPX



# Erős típusosság

---

- Az ASP.NET MVC-ben minden erősen típusos
- Karbantartható, logikus, semmit sem érzünk feleslegesnek
- View-hoz többféle tároló property tartozik
  - ViewBag – bármilyen adat tárolására
  - Model – azonos típusú objektumok tárolására



# Példa az erősen típusosságra

- Inheritsben nem a kódfájl, hanem a Model szerepel

```
<%@ Page ... Inherits="System.Web.Mvc.ViewPage<IEnumerable<MVCHandsOn.Models.Product>>" %>
```

```
<% foreach (var item in Model) { %>
```

```
<tr>
```

```
<td>
```

```
<%= item.Name %>
```

```
</td>
```

```
<td>
```

```
<%= item.Price %>
```

```
</td>
```

```
</tr>
```

```
<% } %>
```



# Tartalom megjelenítése Viewban

---

## ➤ Inline kód

- HTML-be ágyazott `<%= érték %>`
- `<title><%= Model.Name %></title>`

## ➤ HTML helper metódus

- `<%= Html.TextBox("comment") %>`

## ➤ Szerver oldali vezérlő (!)

- Teszőleges ASP.NET-es szerver oldali vezérlő használható

## ➤ Partial View

- Újrafelhasználható vezérlő, amiben nem lehet üzleti logika

## ➤ Html.RenderAction()

- Újrafelhasználható vezérlő, amibe üzleti logika is lehet



# Html helper metódusok

---

- Input vezérlők
  - `Html.CheckBox`, `Html.Hidden`, `Html.RadioButton`, `Html.Password`, `Html.TextArea`, `Html.TextBox`
- Link és URL
  - `Url.Content`, `Html.ActionLink`, `Url.Action`, `Url.RouteUrl`, `Html.RouteLink`
- Html kódolás
  - `Html.Encode`, `Html.AttributeEncode`
- Több választtós input
  - `Html.DropDownList`, `Html.ListBox`
- Futures szerelvénymetódusai (`Microsoft.Web.Mvc.dll`)
  - `Html.Image`, `Html.Button`, `Html.MailTo`, `Html.SubmitButton`, `Html.SubmitImage`
- További metódusok
  - `Html.BeginForm`, `Html.RenderAction`, `Html.RenderRoute`, `Html.RenderPartial`, `Html.ValidationMessage`, `Html.ValidationSummary`, `Html.AntiForgeryToken`



# Szerver oldali vezérlők használata

```
<asp:Repeater ID="MyRepeater" runat="server">
  <ItemTemplate>
    <MyApp:PeopleControl runat="server"
      ViewDataKey='<%# Container.DataItem["peopleid"] %>' />
  </ItemTemplate>
</asp:Repeater>

<script runat="server">
  // Hack! WebForm vezérlő beállításai...
  protected void Page_Load(object sender, EventArgs e)
  {
    MyRepeater.DataSource = ViewBag["peopleid"];
    MyRepeater.DataBind();
  }
</script>
```



# Html.RenderAction (Controller)

- Meghív egy Controller Action-t aminek a tartalmát beilleszti a generált HTML-be

```
public class WorldClockController : Controller
{
    public ActionResult Index() {
        return View(new Dictionary<string, DateTime> {
            { "UTC", DateTime.UtcNow },
            { "New York", DateTime.UtcNow.AddHours(-5) },
            { "Hong Kong", DateTime.UtcNow.AddHours(8) }
        });
    }
}
```

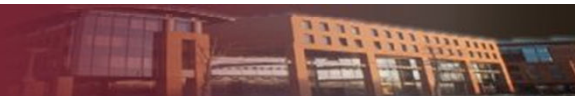


# Html.RenderAction (Razor view)

```
@{ ViewBag.Title="Index"; }

<h2>Index</h2>

<table>
    <thead><tr>
        <th>Location</th>
        <th>Time</th>
    </tr></thead>
    @foreach(var pair in Model) {
        <tr>
            <td>@pair.Key</td>
            <td>@pair.Value.ToShortTimeString()</td>
        </tr> }
</table>
```



# Razor

- Új View Engine
  - WebMatrixban jelent meg
- ASP.NET MVC újdonság
- CÉLOK:
  - Egyszerűsítés
  - Célközönség növelése
  - PHP-nak konkurencia
  - Kevesebb <% %>





# WebForms vs Razor

```
<ul id="products">

    <% foreach(var p in products) { %>
        <li><%=p.Name%> ($<%=p.Price%>)</li>
    <% } %>

</ul>
```

Webforms (8 <%)

Razor (3 @)

```
<ul id="products">

    @foreach(var p in products) {
        <li>@p.Name ($@p.Price)</li>
    }

</ul>
```



# DEMÓ

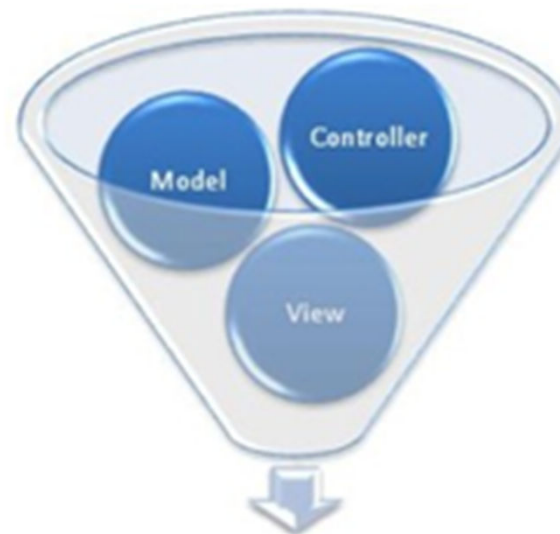
---



Különböző View enginek használata

Termékek, kategóriák listázása

- Razor
- ASPX



ASP.NET MVC Framework

# ASP.NET MVC

---

## DRY

- Egyedi helperek
- Partial View
- Template-ek
- Model validáció



# Egyedi HTML helperek

```
public static class MyHelpers {  
    private const string ScriptIncludeFormat =  
        "<script src=\"{0}\"></script>";  
  
    public static string IncludeScript(string virtualPath) {  
        string clientPath = VirtualPathUtility.ToAbsolute(virtualPath);  
        return String.Format(ScriptIncludeFormat, clientPath);  
    }  
}
```

```
<%= MyHelpers.IncludeScript("~/Scripts/SomeScript.js") %>
```

```
<script src="/Scripts/SomeScript.js"></script>
```



# HTML helper bővítő metódussal

```
public static string IncludeScript(this HtmlHelper helper,
    string virtualPath)
{
    string clientPath = VirtualPathUtility.ToAbsolute(virtualPath);
    return String.Format(ScriptIncludeFormat, clientPath);
}
```

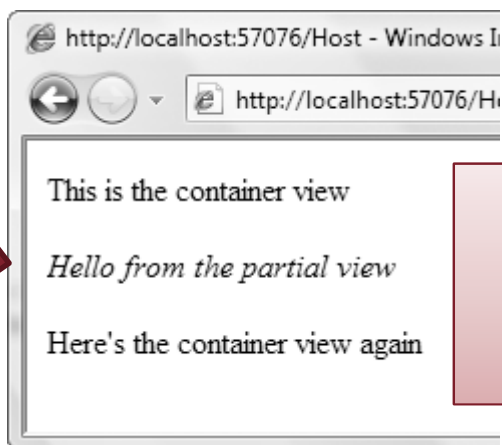
```
<%= Html.IncludeScript("~/Scripts/SomeScript.js") %>
```



# Partial View

- Újra felhasználható vezérlő
  - Nem tartalmaz üzleti logikát
  - View központú megközelítés

```
<%@ Control Language="C#"
    Inherits="System.Web.Mvc.ViewUserControl" %>
<i>Hello from the partial view</i>
```



```
<p>This is the container view</p>
<% Html.RenderPartial("MyPartial"); %>
<p>Here's the container view again</p>
```



# Edit és Display Template

## ➤ Újra felhasználható vezérlő egy típushoz

### ■ Model centrikus megközelítés

@Html.EditorFor( BirthDate

e )

2011.03.25

@model DateTime?

@Html.TextBox( ""

<script type="te

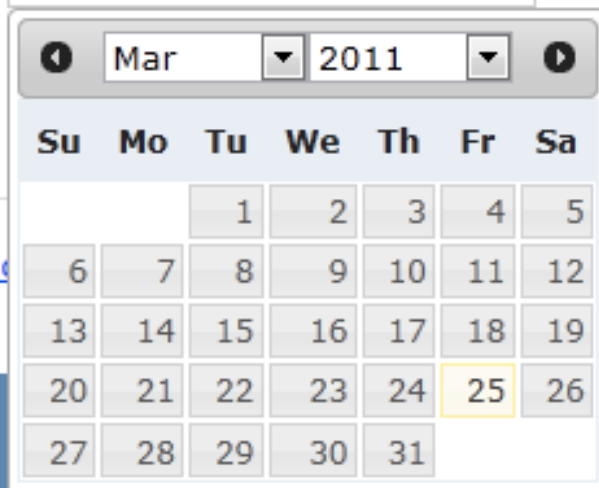
\$(document).re

\$(".dp").dat

changeMonth: true, changear: true,

dateFormat: 'yy.mm.dd' }); });

</script>



MM.dd}",  
: DateTime.Today ),



# Model szintű validáció

- Attribútumokkal lehet megadni
- Szerver: ModelState.IsValid kliens: jQuery

```
[Required(ErrorMessage = "A keresztnév megadása kötelező")]
[StringLength(50, ErrorMessage = "Max 50 karakter")]
public string FirstName { get; set; }
```

```
[HttpPost]
public ActionResult Create( Person friendToCreate ) {
    // Model validálása
    if (ModelState.IsValid) {
        return Redirect("~/");
    }

    // Hibás
    return View(friendToCreate);
}
```

## Person

FirstName

A keresztnév megadása kötelező

LastName

A vezetéknév megadása kötelező



# Összetett validáció

## ➤ Több tulajdonság közti validáció

```
[Required]
```

```
public string Passowrd { get; set; }
```

```
[Required, Compare("Passowrd")]
```

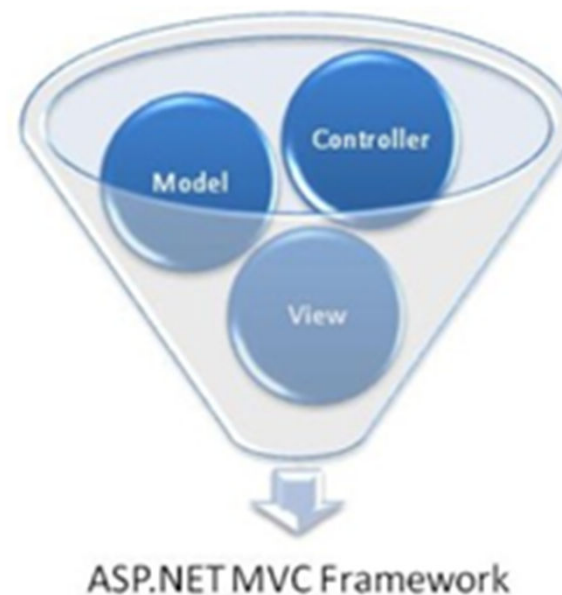
```
public string ConfirmPassowrd { get; set; }
```

```
public class Person : IValidatableObject
{
    public IEnumerable<ValidationResult> Validate(
        ValidationContext validationContext)
    {
        if( YearOfBirth != DateTime.Now.Year + Age )
            yield return new ValidationResult("Születési év vagy kor hibás!");
    }
}
```



# ASP.NET MVC

---



Globális filterek használta  
Egység teszt készítése



# Filterek

---

- MVC 1.0-ban került be.
- Action elé attribútum ami lefut az Actionnel
- Macerás minden Action elé kitenni → **Global filters**
  - Global.asax-ban a RegisterGlobalFilters-ben kell megadni
  - Egy helyen adható meg minden Actionre

## Globális filter

- Minden kéréshez lefut

## FilterProvider

- Feltételtől függően fut le.
  - Pl csak debug módban



# Egység teszt (TDD)

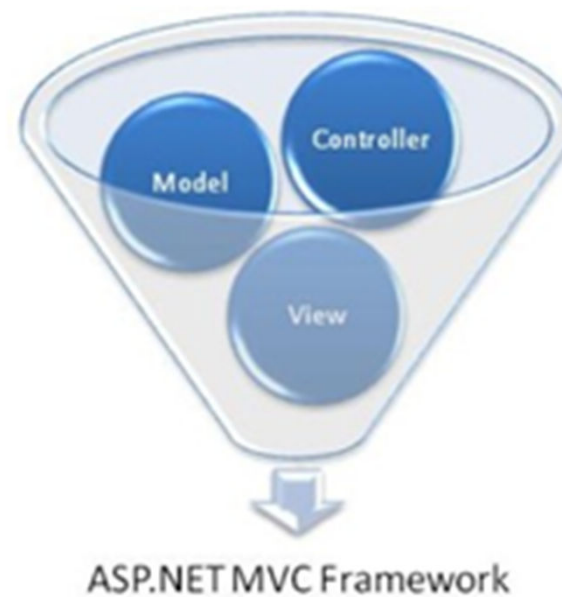
---

- A projecttel együtt létre lehet hozni
- „Ellenőrizd a kódot mielőtt megírnád”
- Helyesen működnek-e a Controllerek?
- Nem indítunk külön ASP.NET processzt
- Web szerver sem fut feleslegesen
- Több Unit Test keretrendszer is támogatott:
  - NUnit
  - MBUnit
  - MS Test
  - Visual Studio Unit Test (default)



# ASP.NET MVC

---



Amit megtanultunk 😊



# Amit megtanultunk 😊

---

- Miért MVC az MVC
- URL Mapping
- Két View engine (aspx és razor)
- DRY
  - Partial View,
  - Template,
  - HTML helper
- Model szintű validáció



# Tehát

---

- ASP.NET Model – View – Controller architektúrára épülő változata
- Hiába 4.0, még mindig új!
- Kis irodalom:
  - Hivatalos oldal:
    - <http://www.asp.net/mvc>
  - Scott Guthrie blogja:
    - <http://weblogs.asp.net/scottgu/>



# jQuery

---

*„The Write Less, Do More, JavaScript Library”*



# Miről lesz szó...

---

- A jQuery létjogosultsága és célja
- Selectorok
- Dokumentum bejárás / HTML manipuláció
- Események
- Effektek
- Ajax
- Pluginek



## Létjogosultság





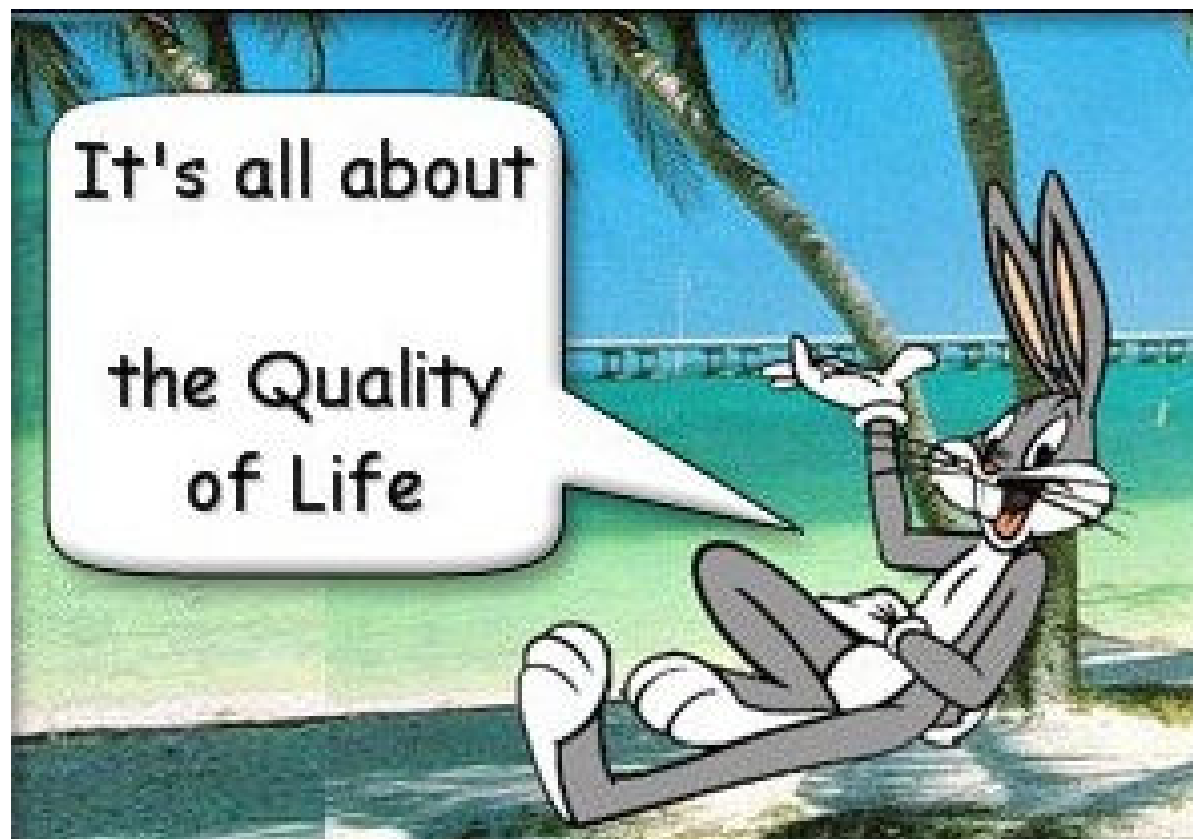
# JavaScript

---

- 1995. december
- Gyengén típusos
- Nincsenek osztályok
- Prototipizált OO nyelv
- Nem DOM manipulációra találták ki



## Használjunk jQuery-t



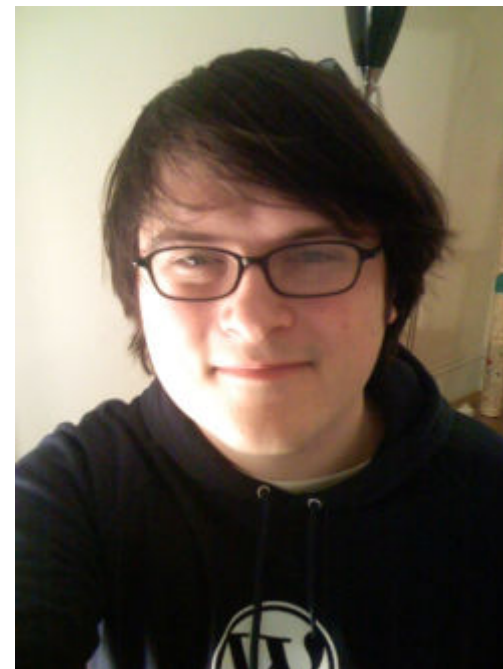
Cél



# A jQuery

---

- 2006. óta létezik
- Egyszerű
- Nyílt forráskódú JavaScript könyvtár
- Egyszerűvé teszi a HTML és JavaScript közti interakciót
- Rengeteg plugin
- Microsoft is támogatja
- VS-ban IntelliSense támogatás

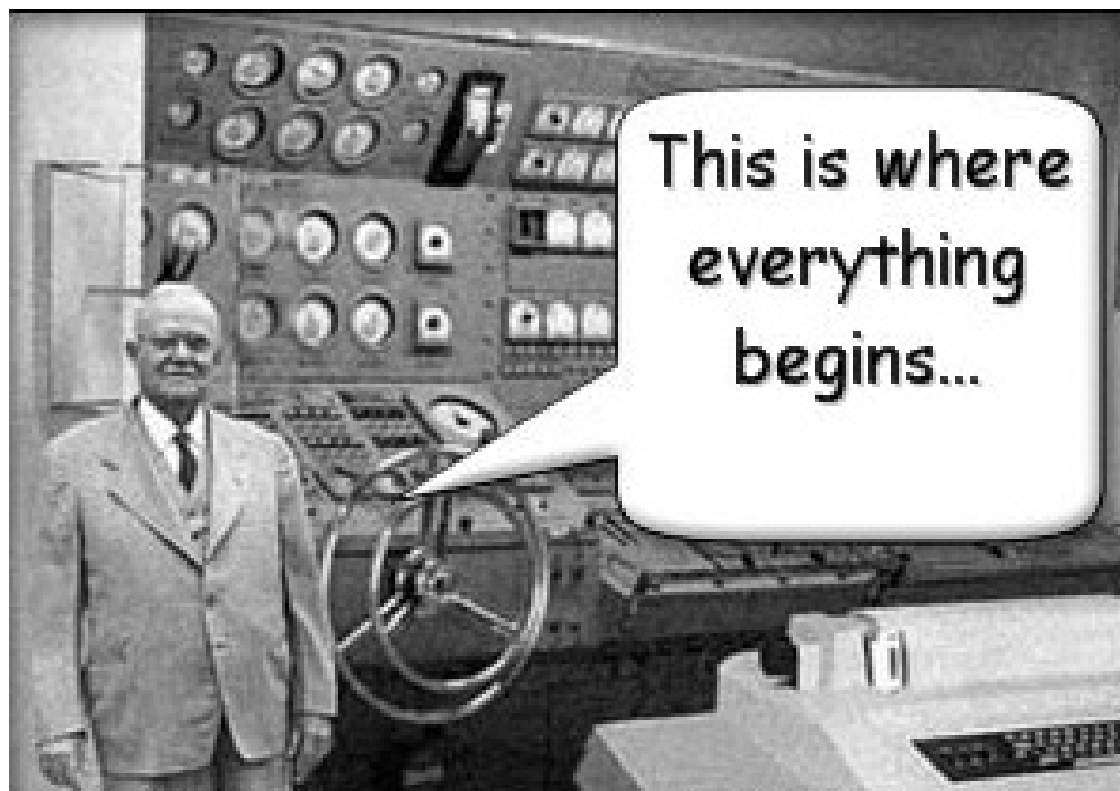


John Resig



# Térjünk a lényegre

---

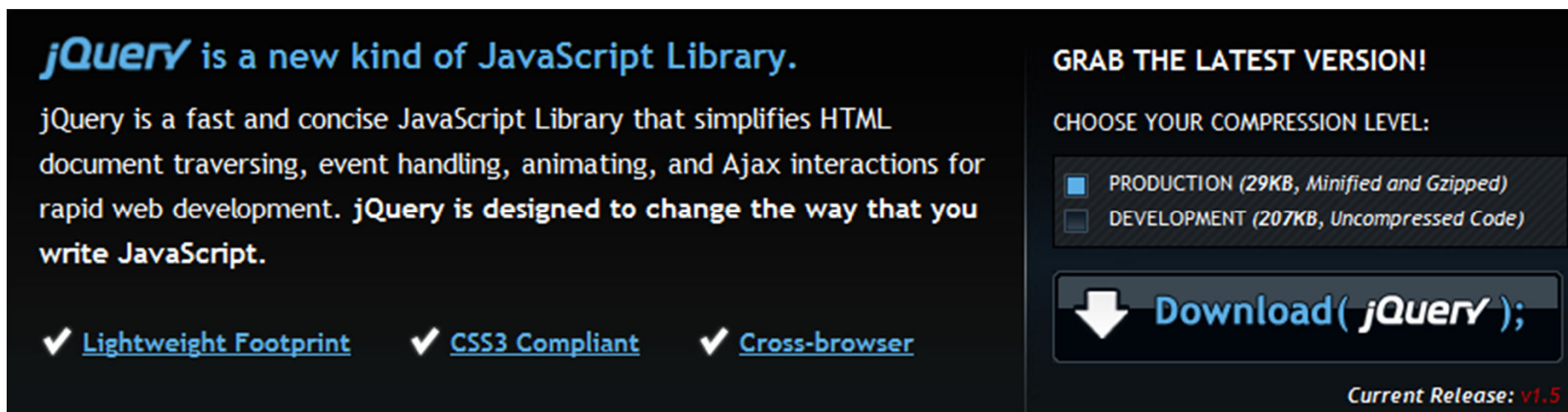




# Első használat előtti teendők

Töltsük le a legfrissebb verziót:

<http://jquery.com>



**jQuery** is a new kind of JavaScript Library.

jQuery is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development. **jQuery is designed to change the way that you write JavaScript.**

✓ [Lightweight Footprint](#)    ✓ [CSS3 Compliant](#)    ✓ [Cross-browser](#)

**GRAB THE LATEST VERSION!**

CHOOSE YOUR COMPRESSION LEVEL:

- ☒ PRODUCTION (29KB, Minified and Gzipped)
- ☐ DEVELOPMENT (207KB, Uncompressed Code)

 **Download ( jQuery );**

Current Release: **v1.5**

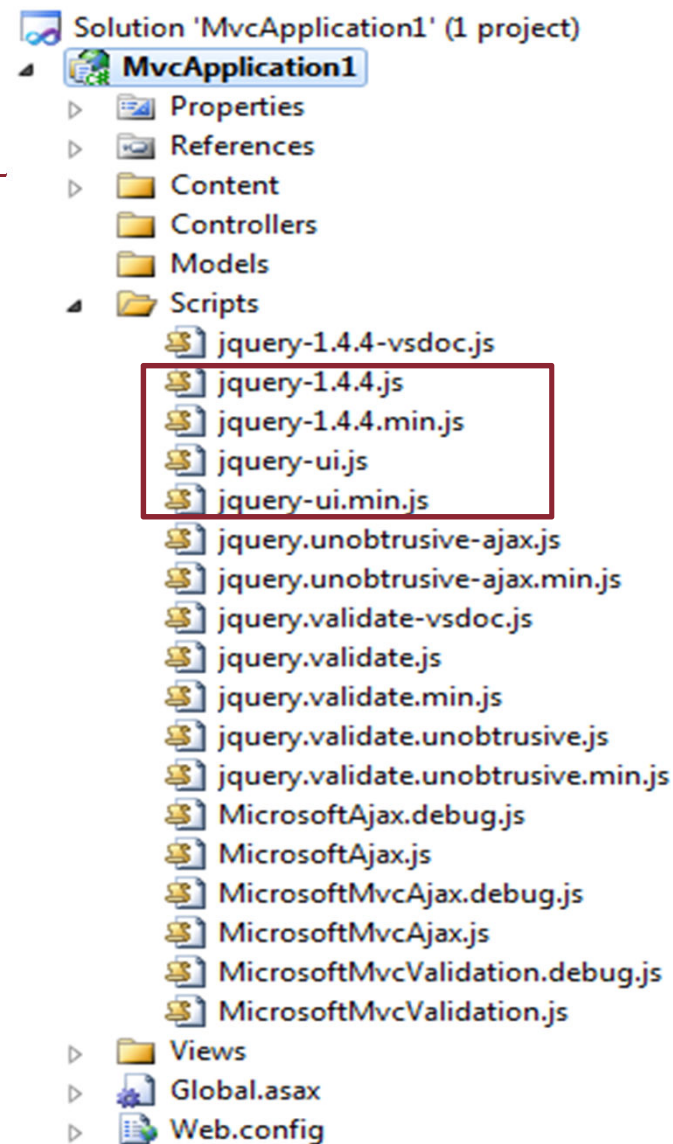
# jQuery és a VS 2010

## ➤ Webes projectekben van

- \*.js – Debug verzió
- \*.min.js – Minimalizált változat
- \*.vsdoc –  
VS IntelliSensehez kell

## ➤ Ajánlott eszközök

- FireQuery (Firebug kiegészítő)
- Fiddler (Ajaxhoz)





# jQuery selectorok és filterek

---





# DEMÓ

---

## jQuery selectorok



# Egyszerű és összetett selectorok

---

## ➤ Egyszerű selectorok

- Minden kiválasztása: `$("*")`
- Tag alapján: `$("table")`
- Id alapján: `$("#tableHeader")`
- CSS osztály alapján: `$(".active")`

## ➤ Összetett selectorok

- Tag és CSS osztály alapján: `$("tr.active")`
- Tag és id alapján: `$("table#tableHeader")`
- Id vagy CSS osztály alapján: `$("#tableHeader, .active")`

## ➤ Ezek után már tudjuk, hogy mit választ ki?

`$("h1, h2, h3, tr.active")`



# Hierarchikus selectorok

---

- Leszármazottak ( ): `$("table td")`
- Gyerekek (>): `$("tr > td")`
- Következő (+): `$("label + input")`
- Testvérek (~): `$("#main ~ div")`



# Szűrések

## ➤ Szűrés **index** alapján

- Első/utolsó: `$("#tr:first")` `$("#tr:last")`
- Index kisebb/nagyobb mint: `$("#tr:lt(2)")` `$("#tr:gt(2)")`
- Index egyenlő: `$("#tr:eq(2)")`

## ➤ Szűrés **láthatóság** alapján

- Látható/rejtett: `$("#div:visible")` `$("#div:hidden")`

## ➤ Szűrés **attribútum** alapján

- Van olyan attribútuma: `$("#div[id]")`
- Az attribútum értéke egyenlő: `$("#div[id='main']")`
- Az attribútum értéke kezdődik / végződik / tartalmazza:  
`$("#div[id^='mai']")`      `$("#div[id$='ain']")`  
`$("#div[id*='ai']")`



# Selectorok és filterek az űrlaphoz

## Selectorok

- Jelölőnégyzet

`$("input:checkbox")`

- Rádió gomb

`$("input:radio")`

- Gomb

`$(":button")`

- Szöveg beviteli mező

`$(":text")`

## Filterek

- Bepipált

`$("input:checked")`

- Kiválasztott

`$("input:selected")`

- Engedélyezett

`$("input:enabled")`

- Letiltott

`$("input:disabled")`



# Legördülő menü kiválasztott eleme

- Legördülő menü HTML kódja:

```
<select name="cities">  
  <option value="1">Budapest</option>  
  <option value="2" selected="selected">  
    Nyíregyháza</option>  
  <option value="3">Debrecen</option>  
</select>
```

- Egy megoldás a sok közül

```
$("#select[name='cities'] option:selected").val()
```



# HTML dokumentum bejárása

---





# DEMÓ

---

## HTML dokumentum bejárása jQueryvel



## jQuery vagy DOM elem

- A selectorok jQuery objektumok tömbjét adják vissza.

```
$("div").length;
```

- A **length** visszaadja, hogy mennyi elemet választottunk ki
- Fontos tudni, hogy **DOM vagy jQuery** elemet kapunk

### DOM elem

```
$("div").get(2);  
$("div")[2]
```

### jQuery elem

```
$("div").eq(2);
```



## each(fn)

- Az **each(fn)** végighívja a megadott függvényt a lista minden elemén
  - A függvény tartalmazhatja a sorszámot, de nem kötelező.
    - `fn()` vagy `fn( index )`
  - Az alábbi kód egy tábla minden páros sorát ellátja az **odd** CSS osztállyal

```
$("table tr").each(  
    function(i){  
        if (i % 2) $(this).addClass("odd");  
    });
```

**A \$(this) DOM objektumból jQuery objektumot készít**



# HTML bejárását segítő függvények

---

- Testvérek: **.siblings( expr )**
- Következő testvér: **.next( expr )**
- Következő testvérek: **.nextAll( expr )**
- Előző testvér: **.prev( expr )**
- Testvérek: **.siblings( expr )**
- Gyerekek: **.children( expr )**
- Leszármazottak: **.find( expr )**
- Szülő: **.parent( expr )**
- Ősök **.parents( expr )**



# Mit csinál az alábbi kód?

---

```
$("#tr td").each(  
    function(){  
        if ( $(this).is(":first-child") )  
            $(this).addClass("firstCol");  
    });
```

```
$("#p").find(".header").hide();
```

```
$(".header", $("#p")).show();
```



# Láncolás

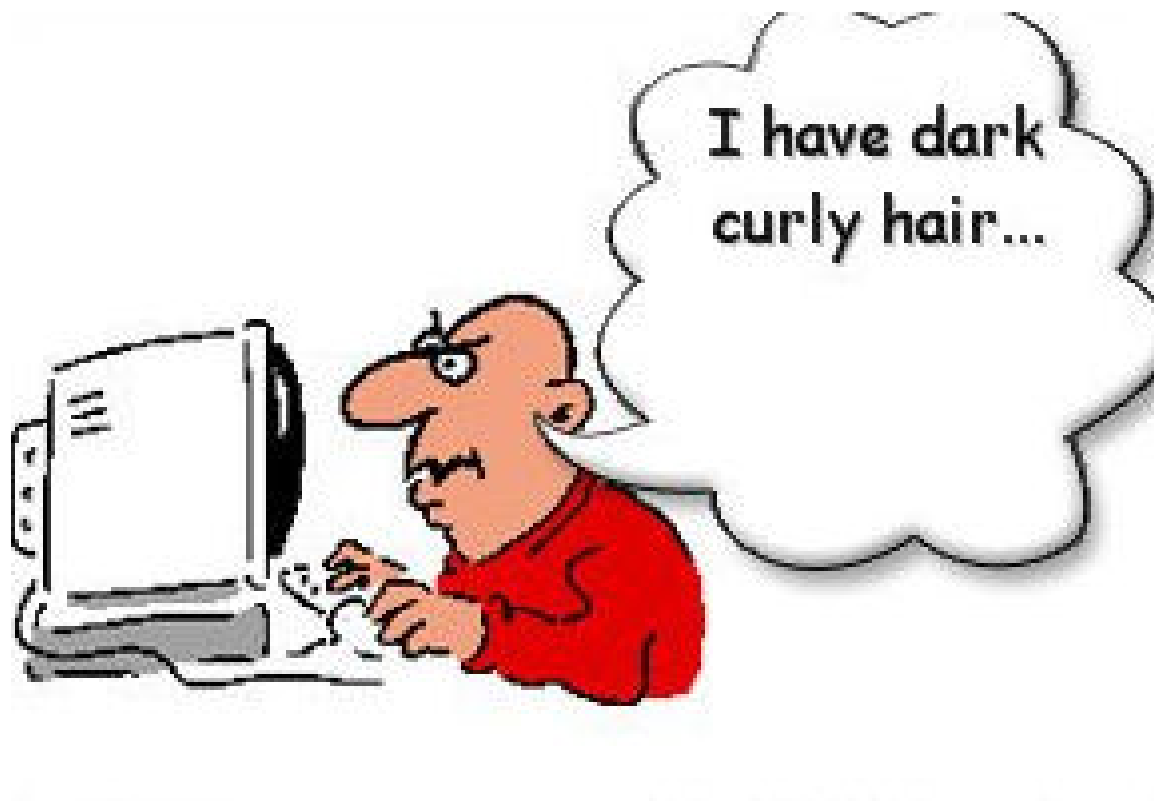
---

```
$( "<li><a></a></li>" )  
  .find( "a" )  
    .html( "About" )  
    .attr( "href", "/Home/About" )  
    .andSelf()  
      .addClass( "menu" )  
    .end()  
  .end()  
  .appendTo( "#menu" );
```



# HTML manipuláció

---





# DEMÓ

---

## HTML manipuláció jQuery-vel



# HTML elemek értéke

---

- Jelölőnégyzet értékének lekérdezése

```
$("input:checkbox:checked").val();
```

- Szövegdoboz értékének beállítása

```
$(":text[name='txt']").val("Hello");
```

- Listaelemek beállítása (kiválasztás, vagy bejelölés)

```
$("#lst").val(["NY", "IL", "NS"]);
```

- HTML tartalom beállítása

```
$("p").html("<div>Hello $!</div>");
```

```
$("div.a").text($("div.b").html());
```



# CSS osztályok manipulálása

- CSS osztály hozzáadása és eltávolítása

```
$("#p").removeClass("blue").addClass("red");
```

```
$("#div").toggleClass("main");
```

- Lekérdezés, hogy rendelkezik-e egy CSS osztállyal

```
if ($("#div").hasClass("main")) { //... }
```

- CSS attribútum lekérdezése / beállítása

```
$("#div").css("background-color");
```

```
$("#div").css("float", "left");
```



# HTML elemek beszúrása és módosítása

- Új HTML elem beszúrása: `append(To)` / `prepend(To)`

```
$("h1").append("<li>Hello $!</li>");  
$("<li/>").html("1").prependTo("ul");
```

- HTML elem lecserélése

```
$("h1").replaceWith("<div>Hello</div>");  
$("<div>Hello</div>").replaceAll("h1");
```

- HTML elem cseréje úgy, hogy a tartalma megmarad

```
$("h3").each(function(){  
    $(this).replaceWith(  
        "<div>" + $(this).html() + "</div>");  
});
```



# HTML elemek törlése

---

- Minden gyerekkel együtt

```
$("#tableHeader").empty();
```

- Kiválasztott elemek törlése

```
$("p.header").remove();
```

```
$("p").remove(":not(.red)")
```

- Kiválasztott elemek pozíciójának módosítása

```
$("p").remove().appendTo("#main");
```



# HTML attribútumok kezelése

---

- Link létrehozása:

```
$("a").attr("href", "home.htm");
```

- Attribútum eltávolítása:

```
$("button").removeAttr("disabled")
```

- Több attribútum beállítása egyszerre

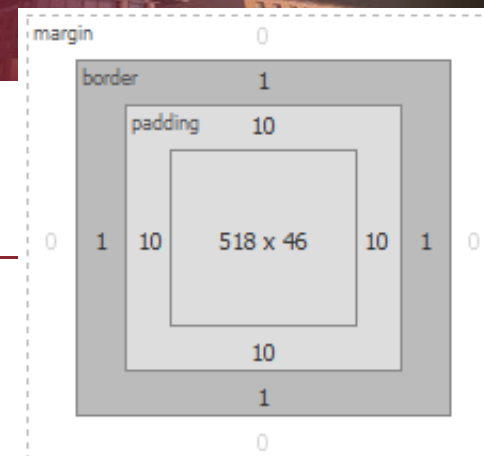
```
$("img").attr({ "src" : "/images/smile.jpg",  
                "alt" : "Smile" });
```

```
$("div").css({ "color"      : "blue",  
               "margin-right" : "10px"  
               marginLeft    : "10px" });
```

# Szélesség és pozíció

## ➤ Szélesség lekérdezésének lehetőségei:

- Elem szélessége: `.width()` `.height()`
- Belső szélesség: `.innerWidth()` `.innerHeight()`
- Szélesség a kerettel: `.outerWidth()` `.outerHeight()`
- Teljes szélesség: `.outerWidth(true)`  
`.outerHeight(true)`



## ➤ Pozíció meghatározása

- A dokumentumtól: `$("div").offset().top;`
- A szülő elemtől: `$("div").position().left;`
- Csúszka pozíció: `$(window).scrollTop();`

**Alapértelmezés szerint minden pixel (px)**



# Események

---





# DEMÓ

---

## jQuery események



# A mágikus `$()` függvény

---

- Az egyik legfontosabb esemény a **document.ready**

`$(function(){...});`

- Akkor váltódik ki amikor a dokumentum kész a kliens oldali kód futtatására.
- Ugyanennek az eseménynek a teljes szintaxisa

`$(document).ready(function(){...});`



# jQuery események

---

- Böngésző események
  - error(), resize(), scroll()
- Dokumentum betöltődése
  - load(), ready(), unload()
- Esemény fel- és leiratkozások
  - (un)bind(), (un)delegate(), one(), live(), die(), trigger(), proxy
- Űrlap eseményei
  - blur(), change(), focus(), select(), submit()
- Billentyűzet események
  - focusin(), focusout(), keydown(), keyup(), keypress()
- Egér események
  - click(), dbclick(), hover(), mousedown(), mouseup(), mouseenter(), mouseleave(), mousemove(), mouseover(), toggle()



## Feliratkozás egy elem eseményre

- A false visszatérési érték ugyanaz, mintha meghívnánk a `.preventDefault()` és `.stopPropagation()`
- `.bind()` mindig lefut, `.one()` első lefutás után leiratkozik automatikusan az eseményről.
- `.live()` azokra az elemekre is lefut, amit később adunk az oldalhoz. A `.die()`-al tudjuk eltávolítani.

```
$(document).ready(function () {  
    $('#foo').bind('click', function (event) {  
        alert('Katt (' + event.pageX + ', '  
              + event.pageY + '))');  
    });  
});
```



# Események kiváltása manuálisan

➤ Tetszőleges eseményt a `.trigger()` tud kiváltani

- Böngésző eseményekre is működik
- Saját eseményeket is kiválthatunk
- A `.trigger()`-rel kiváltott események is felgyűrűznek

```
$("div").trigger("click");
```

➤ A `.triggerHandler()`-rel is kiválthatunk eseményeket, ám

- Ezek nem gyűrűznek tovább
- Csak az első egyező esemény fog lefutni
- Nem fut le a default eseménykezelő (pl. form submit)

```
$("#new").click(function () {  
    $("input").triggerHandler("focus");  
});
```



# Effektek

---





# DEMÓ

---

## jQuery effektek



# Alapvető effektek

---

## ■ Elem megjelenítése / eltüntetése

```
.show(); .hide("fast"); .toggle(100);
```

## ➤ Sliding

```
.slideUp() .slideDown("fast") .slideToggle(1000);
```

## ➤ Fading, átlátszóság állítása

```
.fadeIn(); .fadeOut("normal"); .fadeTo ("fast", 0.5);
```

## ➤ Adjuk meg callbackben, mit csináljon az animáció után

```
$("div").hide("slow", function(){alert("Eltűnt"); });
```



## Egyéb beállítások

- Egyedi animációhoz: `.animate( beállítás, időtartam)`

```
$("div").animate(  
    { width: "90%", opacity: 0.5 }, 1000 );
```

- Animációk láncolása

- Alapértelmezés szerint egymás után hajtódnak végre

```
$("div").animate( { width: "90%" }, 100 )  
    .animate({ opacity: 0.5 },  
            { queue: false, duration: 200 })  
    .animate({ borderWidth: "5px" } );
```



# Ajax és a jQuery





# Az AJAX célja

**Felhasználói élmény  
növelése**





# Szinkron végrehajtás



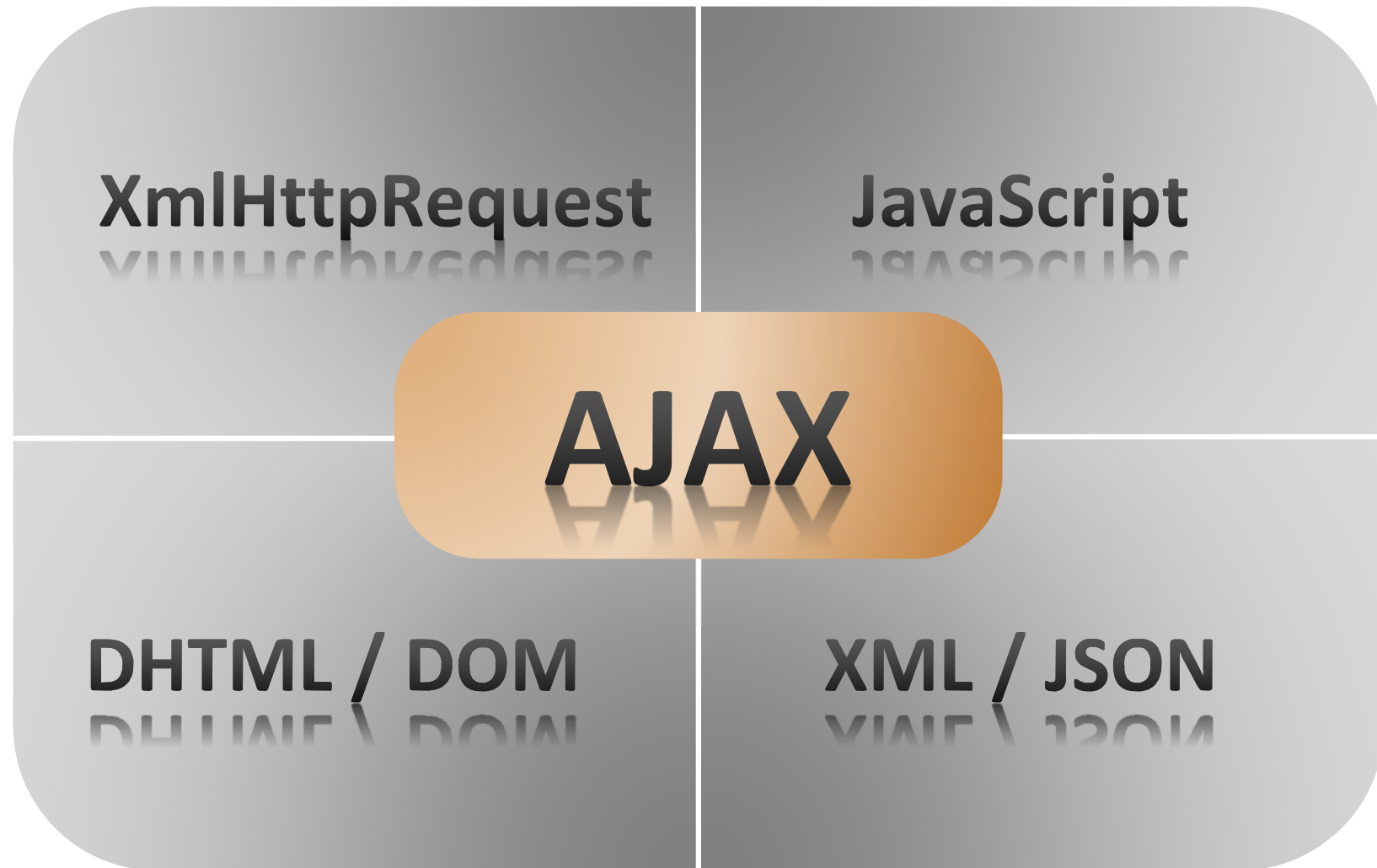


# Aszinkron végrehajtás





# Asynchronous JavaScript and XML

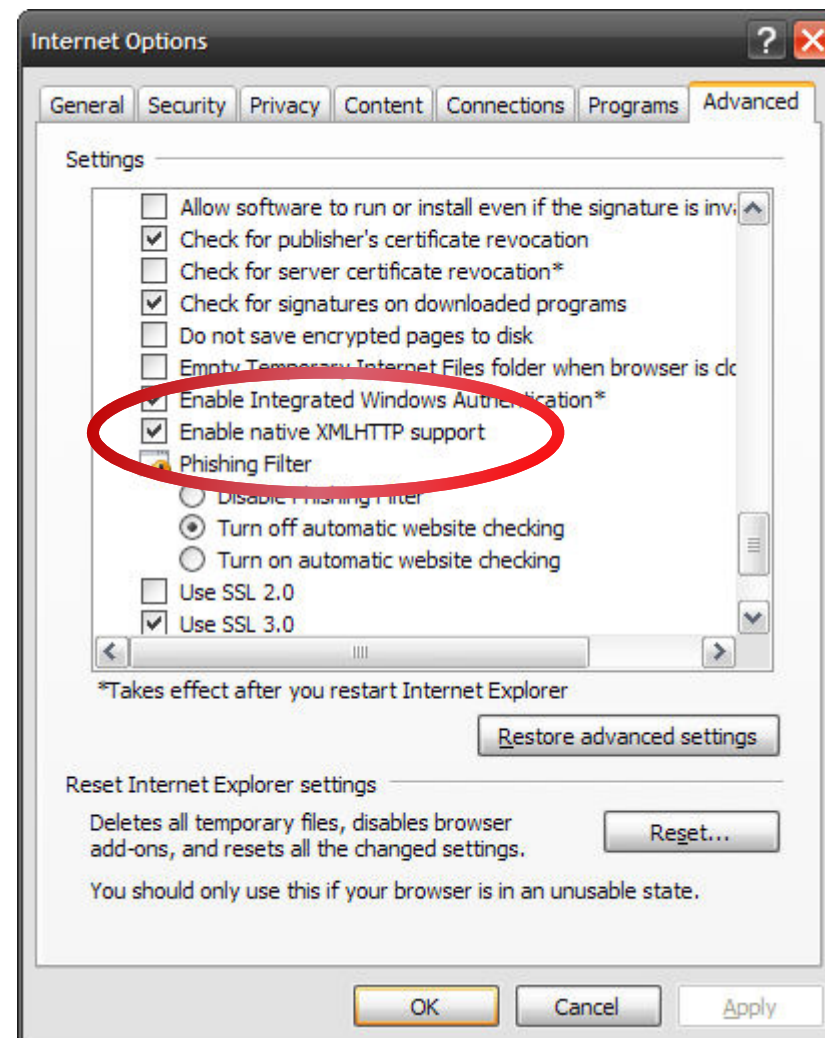
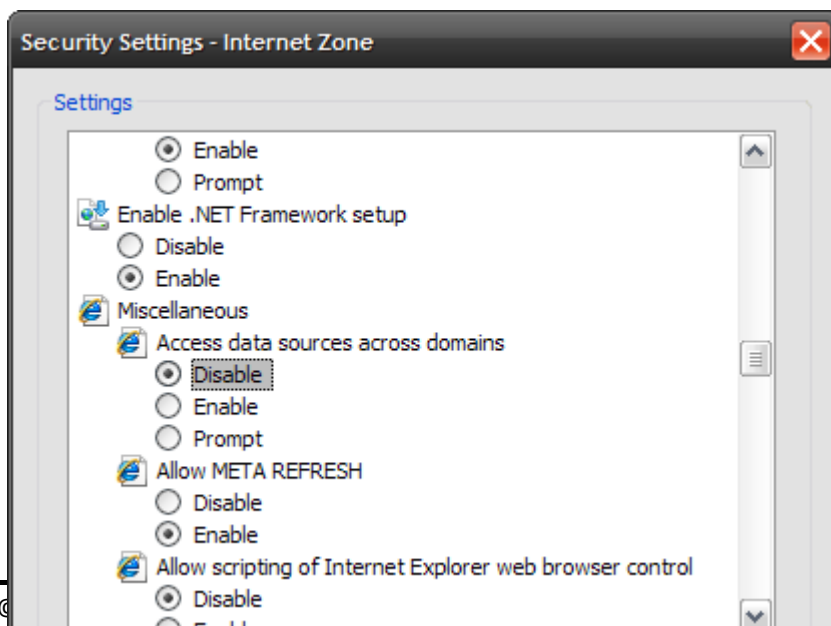


# XmlHttpRequest

## ➤ Böngészőfüggő

- ActiveX: IE 5.x, 6.x
- Natív: Mozilla, Safari, IE 7

## ➤ „Same origin” szabály





# AJAX pro és kontra

## Előnyök

- Aszinkronitás
- Sebesség
  - Csökkentett szerver terhelés\*
  - Jobb hálózat kihasználás
- Felhasználói élmény
  - Rövidebb válaszidők
  - Munkaállapot megőrzése
  - Nincs blokkolás

## Nehézségek

- Szabvány limitációk
- Kereső motorok
- Permalinkek
- Böngésző funkciók
- Forgalom számlálás, méretezés és tesztelés
- Új felhasználói szokások, új design elvárások
- Mobil támogatás



## Egy oldal letöltése .load()

- Egy oldal tartalmának letöltése egy HTML elembe
  - .innerHTML-t használ, (<html>, <title>, <head> szűrés)
  - Same origin szabály
  - Ha adunk át paramétert, akkor POST egyébként GET.

```
$("#div").load("test.php");
```

```
$("#content").load("getcontent.aspx",  
    {"i": "33", "type": "main" } );
```

- Az oldal egy darabjának letöltésére is képes

```
$("#div").load("test.php #mainDiv");
```



# \$.ajax()

➤ Bonyolult, testre szabott Ajaxos lekérdezésekhez.

- global: globális események ne süljenek el (ajaxStart/ajaxStop)
- dataType: xml, json(p), **html**, text, script

```
bodyContent = $.ajax({  
    url: "script.php",  
    global: false,  
    type: "POST",  
    data: ({ id: this.getAttribute('id') }),  
    dataType: "html",  
    async: false,  
    success: function (msg) { alert(msg); }  
} ).responseText;
```



## \$.ajax() rövidített / egyszerűbb formái

- \$.get(): HTTP GET
- \$.post(): HTTP POST
- \$.getScript(): Szkriptet tölt le és futtat
- \$.getJSON(): JSON adathalmazt tölt le



# jQuery összefoglalás

---

- Egyszerű, könnyen tanulható
- Gyorsan készíthető gazdag felhasználói felület
  - Kész pluginek használata
- Kiterjeszthető
- ASP.NET MVC is használható
  - Ajaxos hívások esetén közvetlenül hívható controller
  - JsonResult
- Folyamatosan fejlődik



# jQuery pluginek

---

Csak néhány, ízelítőül



# jQuery UI

---

- <http://jqueryui.com>
- Widget-ek
  - Accordion, Autocomplete, Button, Datepicker, Dialog, Progressbar, Slider, Tabs
- Interakciók
  - Draggable, Droppable, Resizable, Selectable, Sortable
- Effektek, animációk
- Segédosztályok
  - Position



# jQuery Templates

---

- <http://api.jquery.com/category/plugins/templates/>
- Egyirányú adatkötés sablonokkal



```
<!-- Helyőrző a teljes listához -->
```

```
<ul id="movieList"></ul>
```

```
<!-- Egy elem sablonja, helyőrzőkkel az adatoknak -->
```

```
<script id="movieTemplate" type="text/x-jquery-tmpl">
```

```
  <li><b>${Name}</b> (${ReleaseYear})</li>
```

```
</script>
```

```
<script type="text/javascript">
```

```
  // A megjelenítendő adatok
```

```
  var movies = [ { Name: "The Red Violin", ReleaseYear: "1998" },  
                  { Name: "Eyes Wide Shut", ReleaseYear: "1999" },  
                  { Name: "The Inheritance", ReleaseYear: "1976" } ];
```

```
  // Renderelés
```

```
  $( "#movieTemplate" ).tmpl( movies ).appendTo( "#movieList" );
```

```
</script>
```



# jQuery Data Link

---

- <http://api.jquery.com/category/plugins/data-link/>
- Objektumok tulajdonságainak egymáshoz kötése
- Ha az egyik változik, frissül a másik

```
var hallgato = {}; // Adat objektum
$("form").link(hallgato); // Kötés egy űrlaphoz

$(" [name=kernev] ").val("Béla"); // Űrlap kitöltése
alert(hallgato.kernev); // Adat objektumba bekerül
```



# Open Data Protocol

---

OData





**Nyílt adat** (open data):  
mindenki számára\*, standard és  
egyszerűen felhasználható formában  
elérhető adat

\* Csak technológiai szempontból, *open data* != *free data*.





# Probléma: interoperabilitás

---

## Szerver oldal (Producer)

- Operációs rendszerek
- Szerver termékek
  - WebSphere, SharePoint...
- Web platformok
  - ASP.NET, JSP, PHP...
- Szolgáltatások
  - eBay, Netflix, Twitpic...
- ...

## Kliens oldal (Consumer)

- Vastag kliensek
  - Outlook, PowerPivot...
  - .NET, Java, C++...
- Weblapok, böngészők
  - JavaScript, jQuery...
- Beépülő modulok
  - Silverlight, Flash, Java...
- Mobil eszközök
  - iPhone, iPad, WinPhone7...
- ...



# Adat nélkül nem megy

---

## ➤ Top 5 iPhone alkalmazás (2009)

1. Tweetie 2
  - Twitter kliens
2. Yelp
  - Szórakozóhelyek, városi szolgáltatások értékelése
3. Slacker
  - Audio streaming
4. FlightTrack Pro
  - Repülőjáratok követése
5. Mint
  - Pénzügyi árfolyam és tranzakció követő





*“...good data management  
can generate good science,  
and bad data management  
can inhibit the process of  
scientific discovery.”*

*- Ray P. Norris*



# Open Data Protocol (OData)

---

- Web protokoll adatok lekérdezésére és módosítására
- Nyílt protokoll az adatok megosztására
- SQL az interneten át



# Történelem

---

- Microsoft projekt
  - Project „Astoria”
  - ADO.NET Data Services
  - WCF Data Services
- Microsoft Open Specification Promise (OSP)
  - Open Data Protocol elnevezés
- Számos producer és consumer
  - Ld. <http://odata.org>





# Mitől „open” ?

---

- Létező szabványokon alapul
  - Adatátvitel
  - Címzés
  - Adat reprezentáció



# Adatátvitel

---

## ➤ HTTP

- HTTP GET/POST/PUT/DELETE
- Hitelesítés (authentication)
  - Basic
  - Cookie alapú
- Titkosítás
  - HTTPS



# Címzés

---

- Milyen adatot akarunk elérni?
- REST
  - REpresentational State Transfer
  - Entitásoknak egyedi URI
  - Műveletek és paramétereik URL-ben (query string)



# Adat reprezentáció

---

- Milyen formában menjenek át az adatok?
- Atom Publishing Protocol (AtomPub)
  - XML
- JavaScript Object Notation (JSON)
  - Javascriptre, Ajaxra optimalizált



# OData

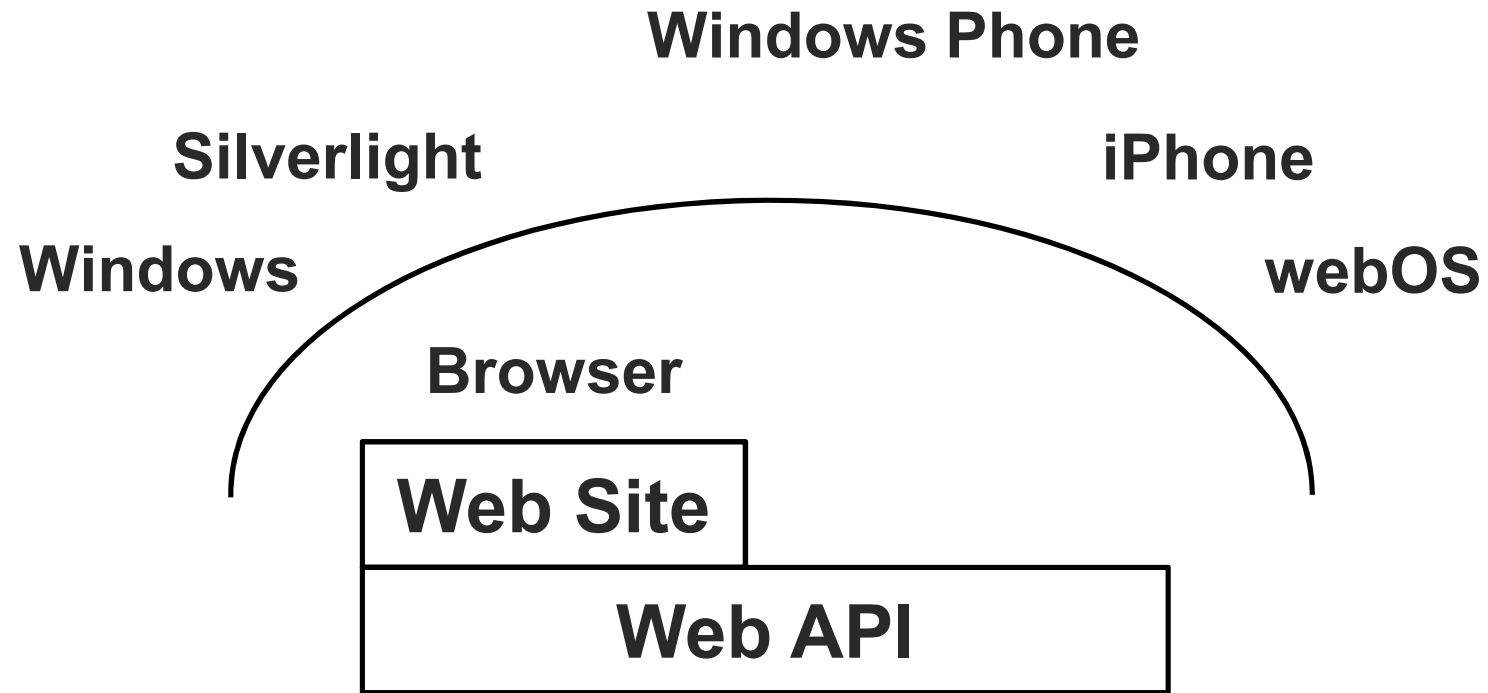
---

HTTP / Atom

Lekérdezés  
(query)

JSON

Metaadatok





# DEMÓ

---

## OData



# Lekérdezés

## ➤ HTTP GET kérés az URI-re

Lekérdezés

Entitás URI

**GET /OData/OData.svc/Products HTTP/1.1**

Host: services.odata.org

Accept: **application/atom+xml**, application/xml

DataServiceVersion: 1.0

MaxDataServiceVersion: 2.0

AtomPub választ kér

Opcionális



# Válasz: feed (AtomPub)

HTTP/1.1 200 OK

Content-Length: 5685 Date: Sat, 27 Feb 2010 20:03:28 GMT

Content-Type: **application/atom+xml**;charset=utf-8

DataServiceVersion: 1.0;

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
```

```
<feed xml:base="http://services.odata.org/OData/OData.svc/"
  xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
  xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
  xmlns="http://www.w3.org/2005/Atom">
```

```
<title type="text">Products</title>
```

```
<id>http://services.odata.org/OData/OData.svc/Products</id>
```

```
<updated>2010-02-27T20:03:28Z</updated>
```

```
<Link rel="self" title="Products" href="Products" />
```

```
<Entry> ... </Entry>
```

```
<Entry> ... </Entry>
```

```
<Entry> ... </Entry>
```

```
</feed>
```

Egyedi URI



# JSON kérés

---

GET /OData/OData.svc/Products HTTP/1.1

Host: services.odata.org

Accept: **application/json**

DataServiceVersion: 1.0

MaxDataServiceVersion: 2.0



# JSON válasz

HTTP/1.1 200 OK

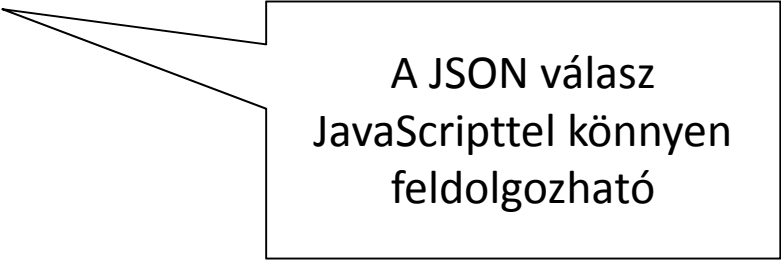
Content-Length: 2481

Date: Sat, 27 Feb 2010 20:08:33 GMT

Content-Type: **application/json**; charset=utf-8

DataServiceVersion: 1.0;

```
{  
  "d" : [  
    { ... },  
    { ... },  
    { ... },  
  ]  
}
```



A JSON válasz  
JavaScripttel könnyen  
feldolgozható



# Egy tulajdonság lekérdezése

---

Gyűjteményen  
belül az entitás  
egyedi URI-je

Tulajdonság  
neve

GET /OData/OData.svc/**Products(1)/Description** HTTP/1.1

Host: services.odata.org

Accept: application/xml

DataServiceVersion: 1.0

MaxDataServiceVersion: 2.0



# Válasz: lehet összetett tulajdonság is

---

HTTP/1.1 200 OK

Content-Length: 158

Date: Sat, 27 Feb 2010 21:10:15 GMT

Content-Type: **application/xml**;charset=utf-8

DataServiceVersion: 1.0;

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
```

```
<Description xmlns="http://schemas.microsoft.com/ado/2007/08/dataservices">
```

```
    Low fat milk
```

```
</Description>
```



# Egyszerű tulajdonság esetén

---

GET /OData/OData.svc/Products(1)/Description/\$value HTTP/1.1

HTTP/1.1 200 OK

Content-Length: 12

Date: Sat, 27 Feb 2010 21:16:05 GMT

Content-Type: **text/plain**; charset=utf-8

DataServiceVersion: 1.0;

**Low fat milk**



# Kapcsolatok

## ➤ Entitások URI-vel hivatkozhatnak egymásra

Hivatkozás egy Category  
típusú entitásra

```
<Link rel="http://schemas.microsoft.com/ado/2007/08/dataservices  
/related/Category"
```

```
type="application/atom+xml;type=Entry"
```

```
title="Category"
```

```
href="Categories(10)" />
```

A Category property  
mutat a másik entitásra

A „10” azonosítójú  
Category entitásra



# További műveletek

---

## ➤ Módosítás

- HTTP PUT: cseréli a régi adatot
- HTTP MERGE: csak a különbséget
- Válasz: HTTP/1.1 204 No Content

## ➤ Törlés

- HTTP DELETE → HTTP/1.1 200 OK

## ➤ Kapcsolatok felvétele

- HTTP POST → HTTP/1.1 204 No Content



# URI

---

`http://services.odata.org/OData/OData.svc`

Szolgáltatás gyökér URI

`/Category(1)/Products`

Kért erőforrás

`?$top=2&$orderby=name`

Lekérdezés opciók



# Kért erőforrás

---

- /\$metadata
- /Categories
- /Categories(1)
- /Categories(1)/Name
- /Categories(1)/Products
- /Categories(1)/Products/\$count
- /Categories(1)/Products(1)/Supplier/Address/City/\$value



# Lekérdezés opciók

---

## ➤ Rendezés: **\$orderby**

- **\$orderby** =Rating
- **\$orderby** =Rating asc
- **\$orderby** =Rating,Category/Name desc

## ➤ Lapozás:

- **\$top**=5
- **\$skip**=2



# Lekérdezés opciók (folyt.)

---

## ➤ Szűrés: **\$filter=**

- Price gt 20
- Address/City eq 'London'
- Price le 200 and Price gt 100
- (Price add 5) gt 10
- substringof('Alfreds', CompanyName) eq true
- length(CompanyName) eq 19
- year(BirthDate) lt 1948
- round(Price) eq 32



## Lekérdezés opciók (folyt.)

---

➤ Hivatkozott entitás betöltése: **\$expand**

- /Categories?\$expand=Products
- /Categories?\$expand=Products/Suppliers
- /Products?\$expand=Category, Suppliers

➤ Formátum megadása: **\$format**

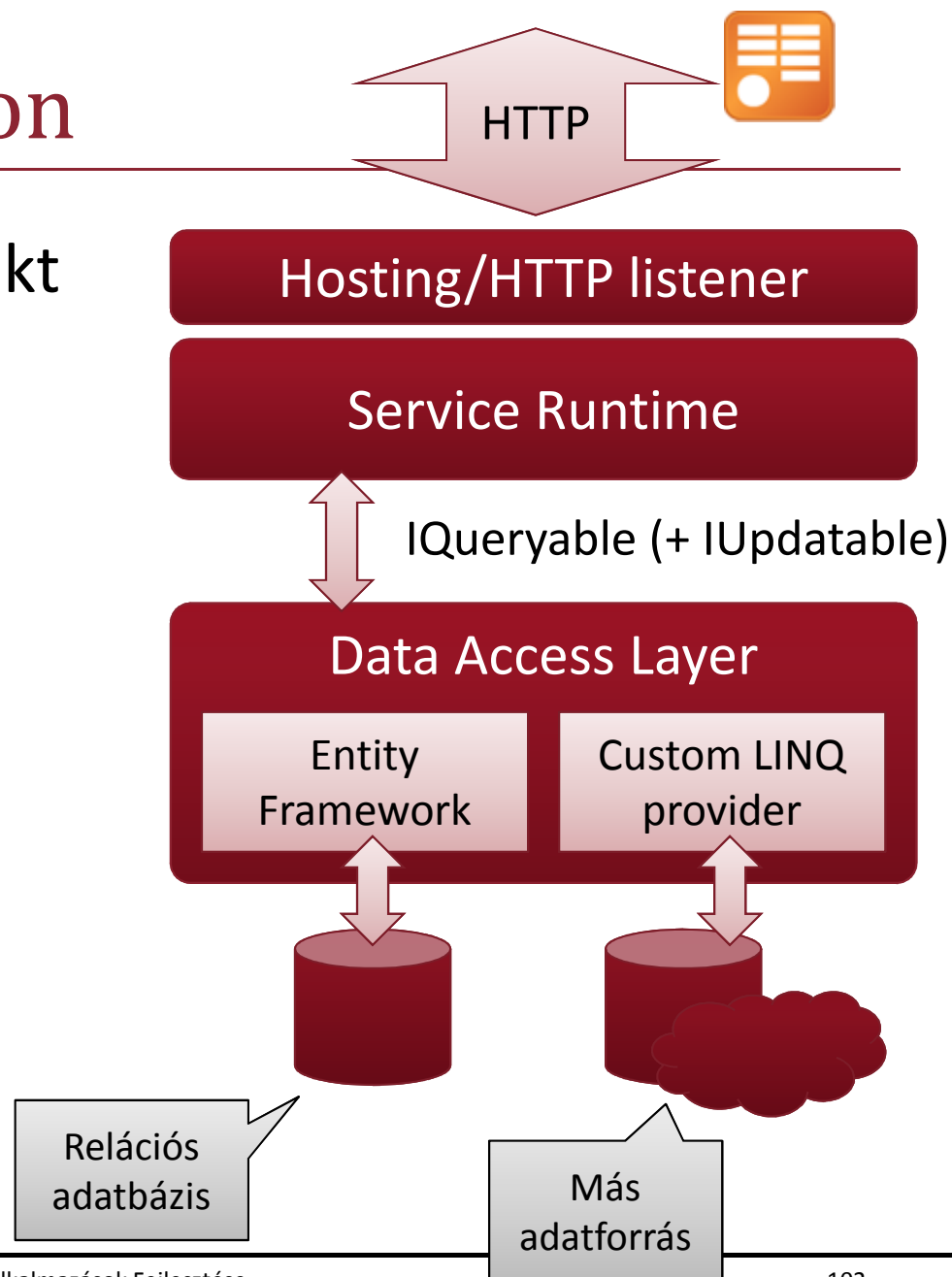
- \$format=atom vagy json vagy xml

➤ Kiválasztás: **\$select**

- /Products?\$select=Name, Category
- /Categories?\$select=Name, Products&\$expand=Products

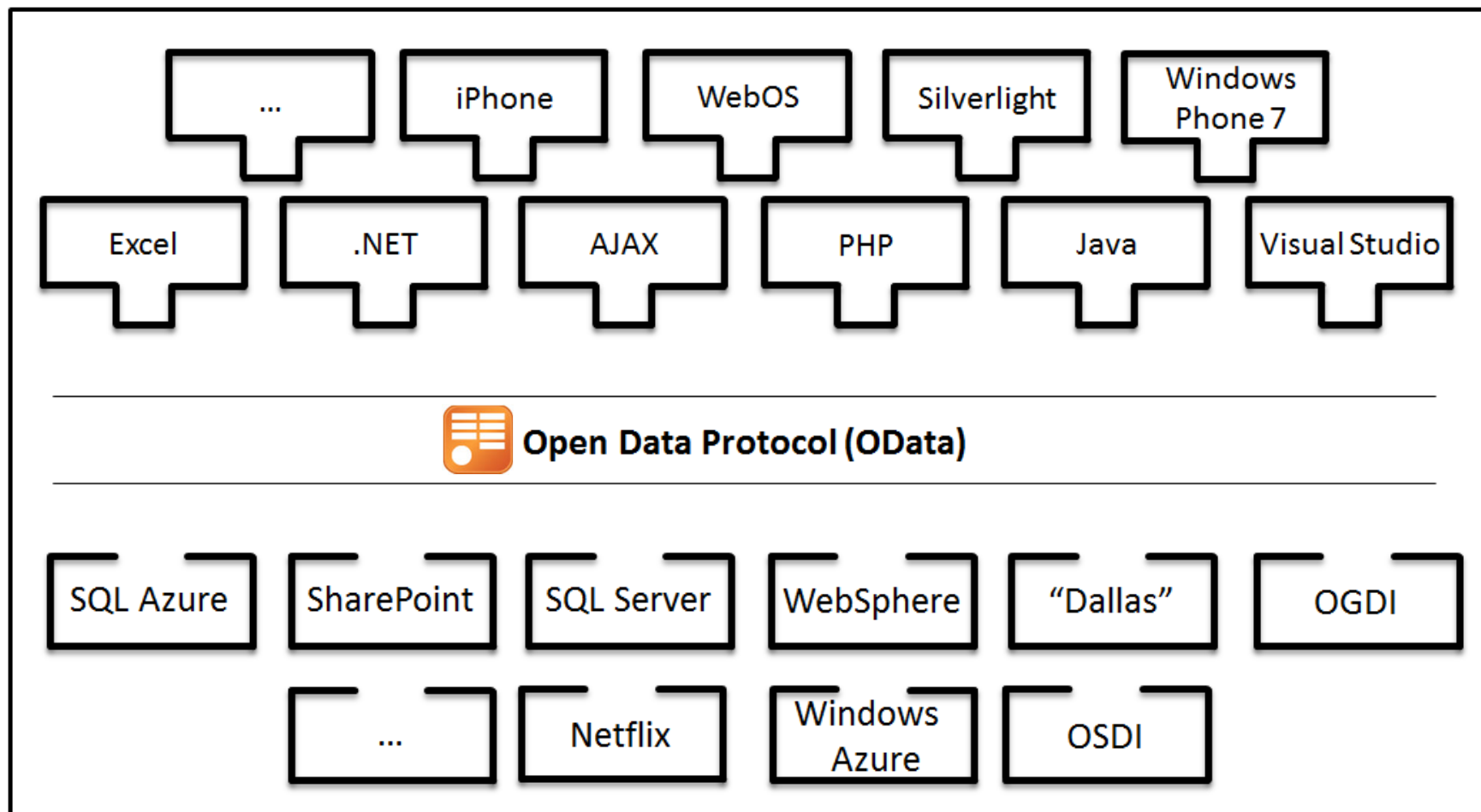
# Microsoft platformon

- WCF Data Services projekt típus Visual Studioban
- Entity Framework
- Entity Data Model





# OData ma



Aktuális: <http://odata.org>



# Windows Communication Foundation - WCF

---



# Tartalom

---

- Definíció, jellemzők
- Architektúra, fogalmak
- Futásidejű architektúra
- Szolgáltatások és kliensek fejlesztése



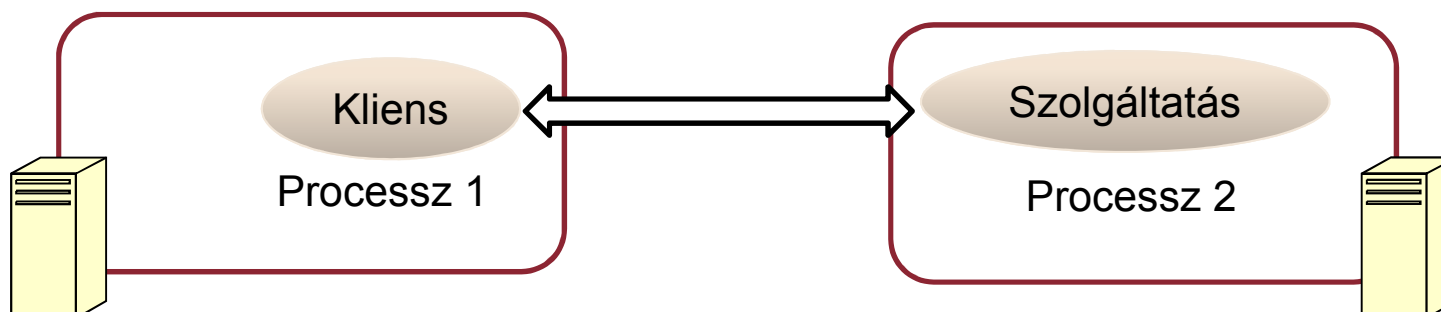
# Definíció, jellemzők

---



# Definíció

- Újgenerációs technológia **elosztott** alkalmazások fejlesztéséhez .NET platformon
- Elosztott
  - Folyamatok (processzek) között
  - A folyamatok futhatnak külön számítógépen is
- Kliens (kezdeményező) és kiszolgáló oldalon futó szolgáltatás





# Jellemzők

---

- .NET környezet
  - A kommunikáció „túloldalán” bármilyen platform lehet!
- Magasabb szint, mint a socketek alkalmazása
- Mire?
  - Kliens-szerver távoli „eljáráshívás”, egyfajta OO RPC (egy másik folyamatban/gépen élő objektum adott metódusának hívása)
  - Nyers XML üzenetek
  - RSS
  - Peer-to-peer
  - Bármilyen, ami elosztott ...
- Transzport lehet: TCP/IP, UDP, shared memory, HTTP, stb.
- Egy egységes programozási modellt nyújt a fentiekre



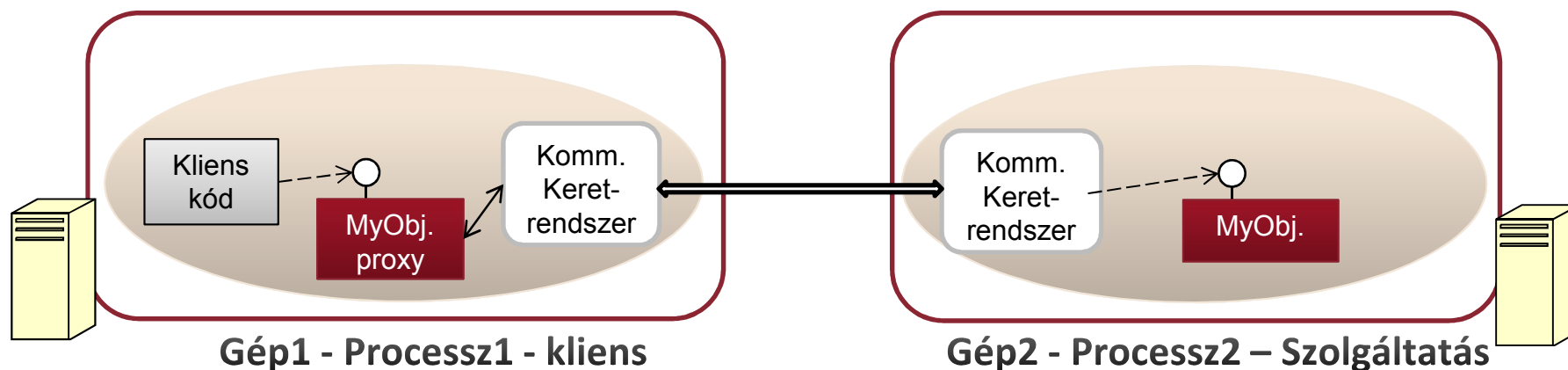
## ➤ Hozzáadott szolgáltatások (opcionálisan)

- Biztonság
- Tranzakciókezelés
- Megbízhatóság
- Üzenetsorok
- ...



# Alapfeladat - távoli objektumelérés

- Számos technológia létezik: .NET Remoting, .NET ASMX webszolgáltatás, WCF, Java RMI, CORBA, ...
- Távoli objektum műveleteinek meghívása, transzparens módon
  - ❑ Úgy hívom meg a műveleteit, mintha egy lokális objektum lenne
  - ❑ A példányosítás NEM transzparens (nem is cél)
  - ❑ (Remote) Proxy tervezési minta

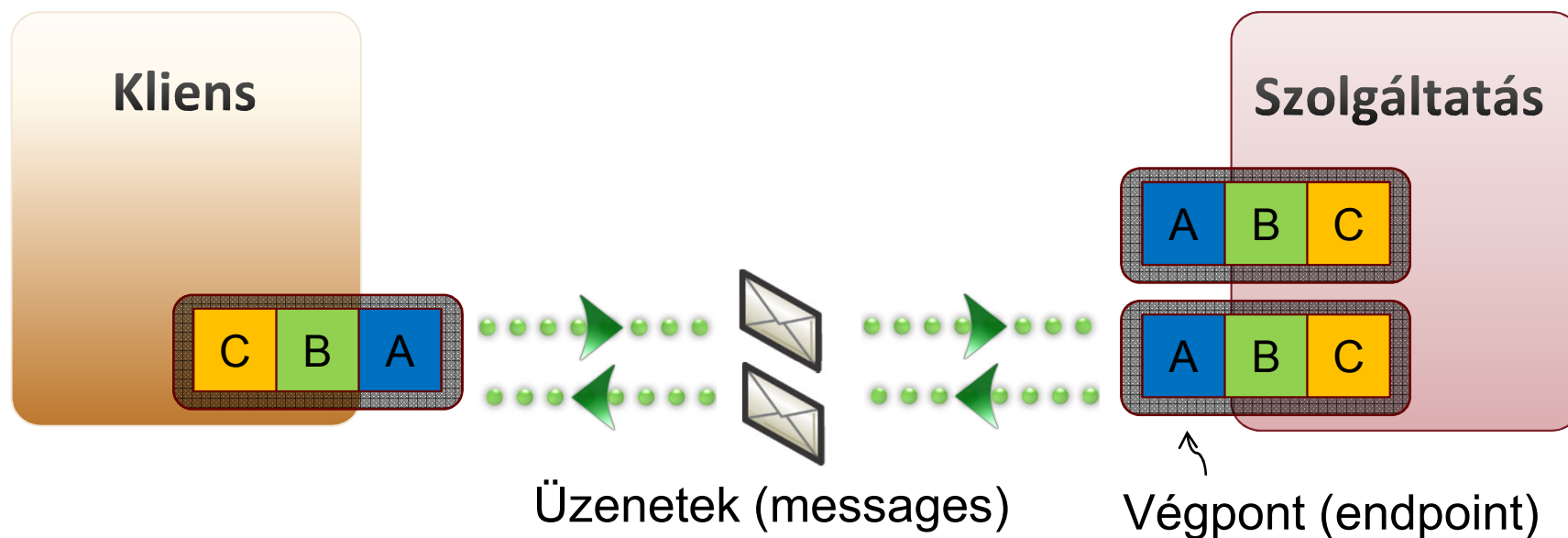




# Architektúra, fogalmak

---

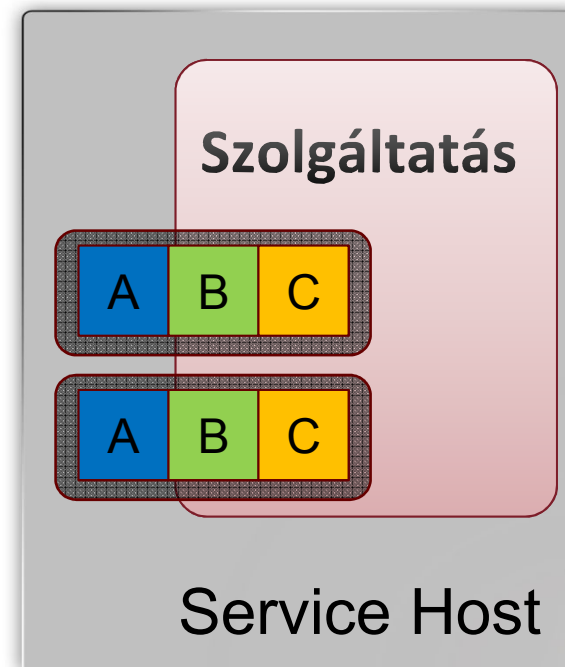
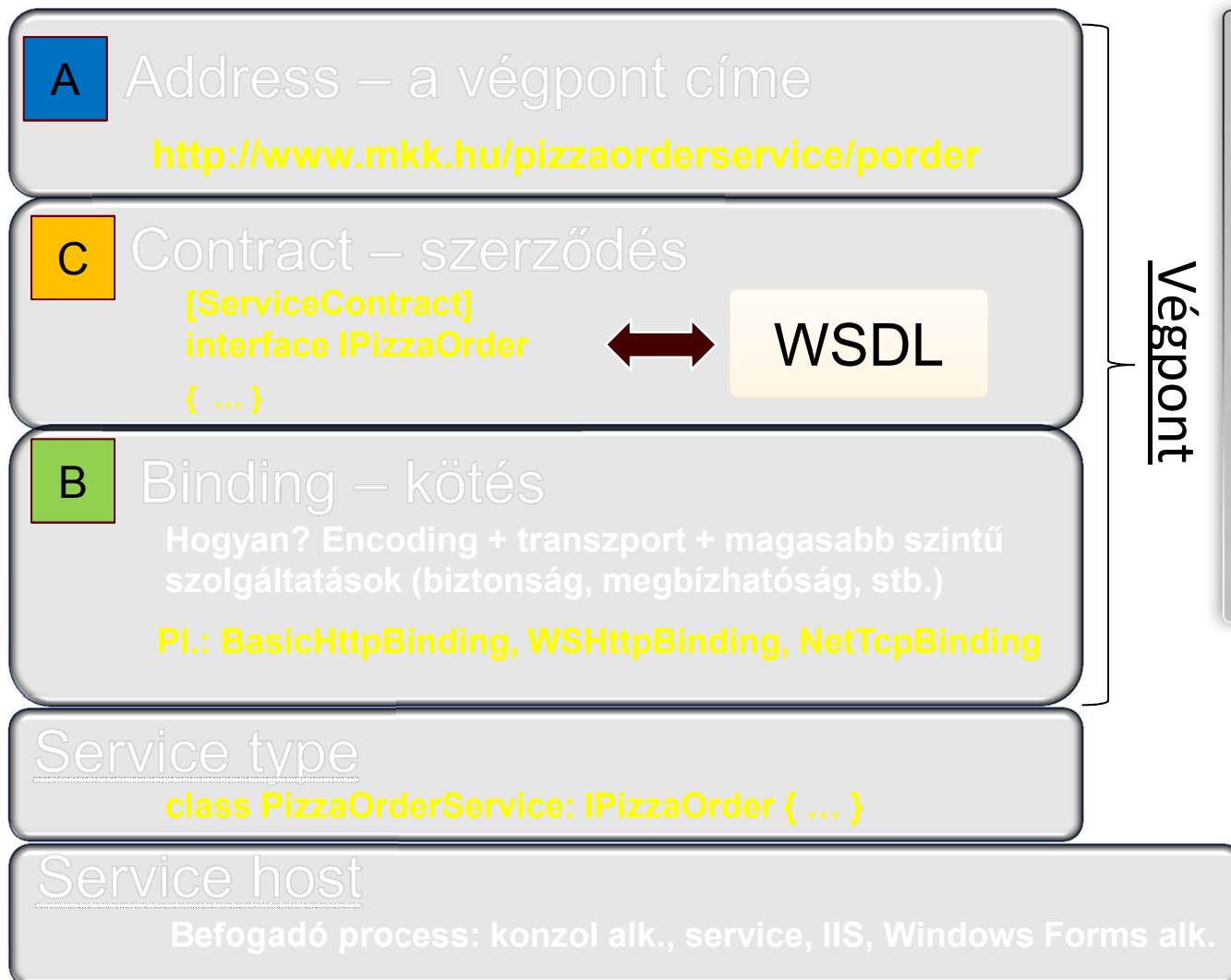
# Kommunikáció



- Egy szolgáltatás egy vagy több végpontot publikál
- A kliens is rendelkezik végpont(ok)al
- A végpontok között üzenetek mozognak
  - Tipikusan a kliens küld egy kérés üzenetet, amire a szolgáltatás egy válaszüzenettel válaszol



# Architektúra - szolgáltatások



Address: Hol?

Contract: Mit?

Binding: Hogyan?



# Architektúra - szolgáltatások

---

## ➤ Address - cím

- A cím, ahol a szolgáltatás elérhető
- Egy URL
- Két részből áll: IP cím/DNS név + szolgáltatás url
- A „HOL?” kérdésre ad választ



# Architektúra - szolgáltatások

---

## ➤ Contract – szerződés

- Mit lehet a cím alatt elérni
- Szerződés a kliensek felé, a szolgáltatás leírása
- Interfész a műveletekkel (pl. NewOrder, DeleteOrder) + adatok leírása (pl. NewOrder művelet Order paramétere)
- Leírás kétféleképpen:
  - .NET nyelven: interfész + entitás osztályok
  - WSDL (Web Service Description Language – szabványos, XML alapú leírás)
- A „MIT?” kérdésre ad választ



# Architektúra - szolgáltatások

---

## ➤ Binding – kötés

- Egy leírása annak, hogyan történjen a kommunikáció
- Pl. hogyan legyen encodolva az adat: XML, bináris; mi a transzport: nyers TCP/IP, HTTP, shared memory; további számos aspektus...
- Számos beépített binding áll a rendelkezésünkre
- A „HOGYAN?” kérdésre ad választ.

## ➤ Service type

- A szerződés interfészt implementáló közösleges .NET osztály

## ➤ Service host

- Befogadó folyamat, futási környezet



# Futásidejű architektúra

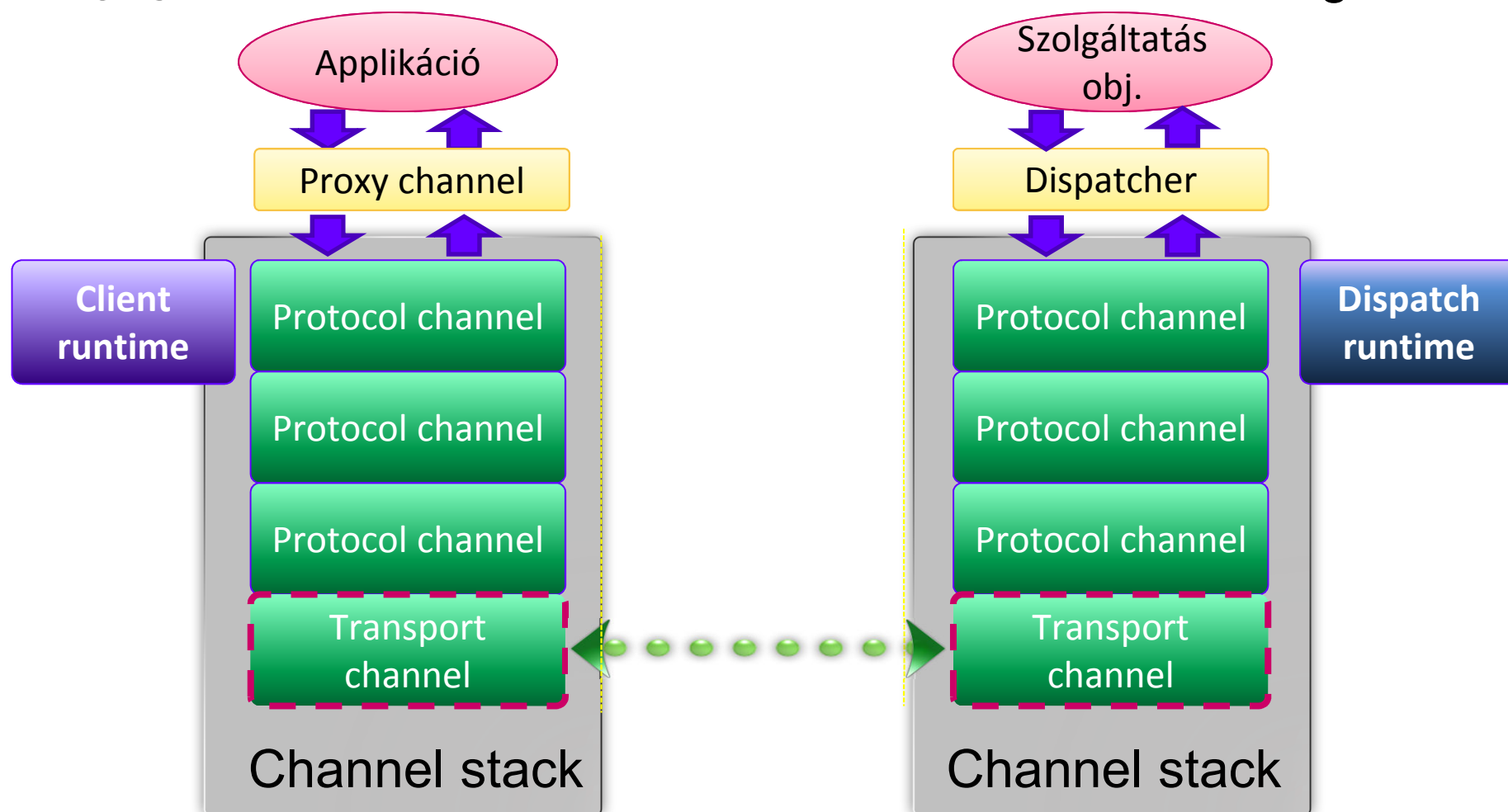
---



# Architektúra - csatornák

Kliens

Szolgáltatás





# Architektúra - csatornák

---

- A kliens applikáció meghívja a proxy (channel) adott műveletét
- A proxy a művelethívás hatására egy Message objektumot készít (XML, a művelethívás és paraméterei)
  - A művelet paraméterei sorosítódnak (serialization): .NET objektumból XML infóset
- Ez a Message objektum végighalad a kliens oldali channel stack rétegein (lefelé)
- A legalsó, transport channel adatfolyammá alakítja, és elküldi a kiszolgáló oldalnak



# Architektúra - csatornák

---

- A kiszolgáló oldalon a Dispatch runtime legyártja a kiszolgáló oldali channel stack-et
- A transport channel fogadja az adatokat, egy Message objektumot készít
- Ez a Message objektum végighalad a kiszolgáló oldali channel stack rétegein (felfelé)
- A Dispatch Runtime a Message objektum alapján meghívja a megfelelő szolgáltatás objektum megfelelő műveletét (ehhez deszerializálja a paramétereket)
- A visszatérési értékből (és out paraméterek) Message objektum készül, ami a szolgáltatás oldali channel stacken végighalad, eljut a klienshez, ott is végighalad a channel stacken (felfelé), és proxy művelet visszatér.



# Architektúra - csatornák

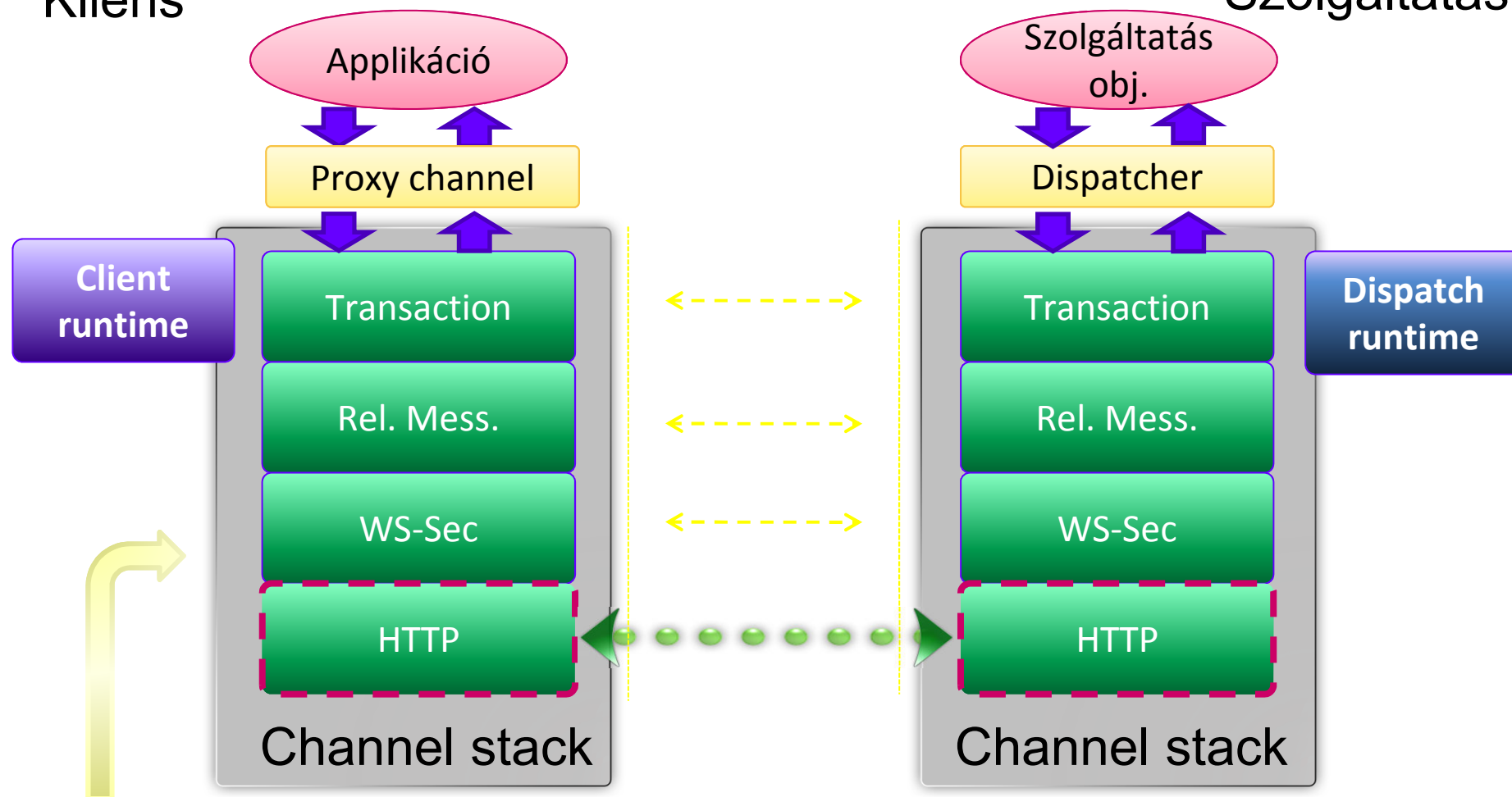
---

- Láttuk: a channel stack réteges felépítésű
- Nagyon hasonlít a hálózati protokollokéhoz
- Minden réteg szolgáltatást nyújt a felette levőnek, és az alatta levő szolgáltatásait pedig igénybe veszi. Ilyen szolgáltatás pl.:
  - Transzport: adat küldés és fogadás
  - Biztonság
  - Megbízhatóság,
  - Tranzakciókezelés
  - Stb.
- Minden réteg a „túloldal” ugyanazon rétegével kommunikál (a Message objektumba tesz ehhez adatokat)

# Architektúra – csatornák: példa

Kliens

Szolgáltatás



**Pl. WSHttpBinding adott módon paraméterezve**



# Architektúra – csatornák: példa

---

- Az előző ábrán egy konkrét csatornaconfigurációt láthatunk
- Ez a konfiguráció egy pl. a WSHttpBinding használatakor áll elő (adott módon paraméterezve)
- A binding alapvetően nem más, mint egy csatornaconfiguráció leírása



# WCF szolgáltatások és kliensek fejlesztése

---

Hogyan kell egy egyszerű szolgáltatást elkészíteni?

Hogyan kell egy klienst elkészíteni?



# Fontosabb lépések áttekintése

---

- Előkészítés (Visual Studio solution, projektek)
- Contract definiálása
  - Szolgáltatás interfész
  - Adatszerkezetek (paraméterek és visszatérési érték)
- Szolgáltatás implementáció
- Szolgáltatás hosztolás
  
- Kliens elkészítése

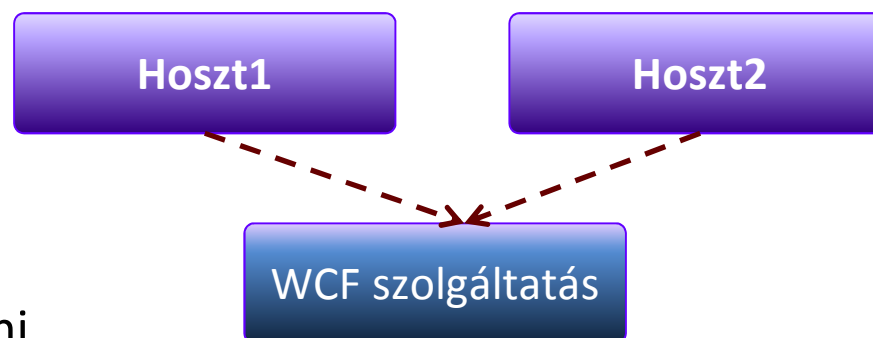


# A szolgáltatás előkészítése

## ➤ Komponens struktúra (szerelvények, projektek)

### ■ Hol hosztoljuk?

- Célszerű a szolgáltatást egy DLL-ben (class library) implementálni, hogy tetszőleges módon hosztolni lehessen



## ➤ Szerelvények:

- System.ServiceModel

## ➤ Névterek

- System.ServiceModel



# Szerződés definiálása

---

- Szolgáltatás szerződés (Service Contract) definiálása
  - Műveletek: ServiceContractAttribute és OperationContractAttribute
  - Adatszerkezetek: DataContractAttribute és DataMemberAttribute



# Szerződés definiálása

---

## ➤ Műveletek

```
[ServiceContract(Namespace = "http://mycompany.hu/restaurantservices")]  
public interface IPizzaOrderService  
{  
    [OperationContract]  
    Guid NewOrder(Order order);  
  
    [OperationContract]  
    Order[] GetOrdersOfUser(string userName);  
}
```



# Szerződés definiálása \*

## ➤ Adatszerkezetek

```
[DataContract(Namespace = "http://mycompany.hu/restaurantservices")]
public partial class Order
{
    [DataMember]
    public Guid OrderId;
    [DataMember]
    public string UserName;
    [DataMember]
    public List<OrderItem> Items;
}
```

```
[DataContract(Namespace = "http://mycompany.hu/restaurantservices")]
public partial class OrderItem
{
    [DataMember]
    public string ItemCode;
    [DataMember]
    public double Quantity;
}
```



# Szolgáltatás implementáció, hosztolás\*

## ➤ Implementáció

```
class PizzaOrderService : IPizzaOrderService { ... }
```

## ➤ Hosztolás

```
Uri baseAddress = new Uri("http://localhost:8080/RestaurantServices");  
ServiceHost seviceHost = new ServiceHost(  
    typeof(RestaurantServices.PizzaOrderService),  
    baseAddress  
);
```

## ➤ A, Végpontok felvétele kódból

```
seviceHost.AddServiceEndpoint(  
    typeof(RestaurantServices.IPizzaOrderService),  
    new BasicHttpBinding(),  
    "PizzaOrderService");  
seviceHost.Open();
```



# Szolgáltatás implementáció, hosztolás\*

## ➤ B, Végpontok megadása app.config fájlban

```
<system.serviceModel>
  <services>
    <service name="RestaurantServices.PizzaOrderService">
      <endpoint address="PizzaOrderService" binding="basicHttpBinding"
        contract="RestaurantServices.IPizzaOrderService">
      </endpoint>
      <!--további végpontok itt-->
    </service>
  </services>
</system.serviceModel>
```

- A végpontokat és a báziscímet nem jó a kódba beégetni, inkább konfigurációs fájlban adjuk meg.
- Tetszőleges számú végpont (ABC kombináció) megadható



# A kliens elkészítése

---

- Mire van szükség a kliensen a szolgáltatás meghívásához?
  - A szolgáltatás címe
  - Szerződés kódja C# nyelven
    - IPizzaOrderService – hiszen meg kell tudjuk hívni a műveleteit
    - Order, OrderItem – hiszen szerepelnek a műveletek paramétereiben
  - Olyan binding, ami kompatibilis a szolgáltatásával

Vegyük észre, hogy ez a három dolog pont egy végpont leírás, egy „ABC” kombináció.



# Kliens és a szerződés

---

- A kliensnek szüksége van a szerződések leírására (interfész, adat és konfiguráció)
  - Eddig a kliens megkapta a szerződések **kódját** (*forráskód vagy szerelvény*)
    - Nem lehetséges mindig (pl. más technológia)
  - A szolgáltatás publikálhat egy **metadata-exchange** végpontot (WS-Metadata Exchange)
    - Address: mi döntjük el
    - Contract: IMetadataExchange
    - Binding: MexHttpBinding, MexHttpsBinding, MexNetTcpBinding, MexNamedPipeBinding
  - Vagy: **WSDL + XSD**



# Kliens és a szerződés

---

- SDK Tool: Service Model Metadata Utility (**svcutil.exe**)
  - Metadata exchange végpont → konfigur. + szerződés kód + proxy
  - Metadata exchange végpont → wsdl + xsd
  - wsdl + xsd → konfigur. + szerződés kód + proxy
- Visual Studio: **Add Service Reference** ... a kliensben
  - Egyszerűbb, mint az svcutil.exe használata



# Metaadatok publikálása\*

---

## ➤ Engedélyezése

- Opcionálisan: Http get támogatás engedélyezés (a WSDL letöltéséhez)
- ServiceBehavior

```
<behavior name="MyServiceTypeBehaviors" >  
    <serviceMetadata httpGetEnabled="true" />  
</behavior>
```

## ➤ MetadataExchange végpont publikálása

- Contract: IMetadataExchange
- Binding: mexHttpBinding



# Néhány fontosabb binding áttekintése

| <u>Binding</u>                                              | <u>Interop</u>    | <u>Biztonság</u>       | <u>Session</u>                                                                        | <u>Tranz.</u>                                                                         | <u>Duplex</u>                                                                         |
|-------------------------------------------------------------|-------------------|------------------------|---------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|
| <b>BasicHttpBinding</b>                                     | Basic Profile 1.1 | Transzport, Üzenet*    |                                                                                       |                                                                                       |                                                                                       |
| <b>WSHttpBinding, WS2007HttpBinding</b>                     | WS                | Transzport, Üzenet, v. |    |    |                                                                                       |
| <b>WSDualHttpBinding</b>                                    | WS                | Üzenet                 |    |    |    |
| <b>WSFederationHttpBinding, WS2007FederationHttpBinding</b> | WS-Federation     | Üzenet                 |    |    |                                                                                       |
| <b>NetTcpBinding</b>                                        | .NET              | Transzport, Üzenet     |  |  |  |
| <b>NetNamedPipeBinding</b>                                  | .NET              | Transzport             | Transzp.                                                                              |  |  |
| <b>NetMsmqBinding</b>                                       | .NET              | Transzport, Üzenet     |                                                                                       |  |                                                                                       |
| <b>NetPeerTcpBinding</b>                                    | Peer              | Transzport             |                                                                                       |                                                                                       |  |
| <b>MsmqIntegrationBinding</b>                               | MSMQ              | Transzport             |                                                                                       |  |                                                                                       |



# Mikor mit válasszunk?

---

## ➤ BasicHttpBinding

- Fontos az együttműködés más platformmal (pl. Java) és nincs szükség extra szolgáltatásokra

## ➤ WSHttpBinding

- Fontos az együttműködés más platformmal (pl. Java) és szükség van extra szolgáltatásokra is (előfordulhatnak együttműködési problémák)

## ➤ NetTcpBinding

- .NET van mindkét oldalon, ez a leghatékonyabb, és minden extra szolgáltatást is tud
- De csak .NET  $\leftrightarrow$  .NET



# Windows Presentation Framework (WPF) – architektúra

---



# A WPF előtt

---

## ➤ GDI – Graphics Device Interface

### ■ Egy API a kimeneti eszközökhöz

- Monitor
- Printer

### ■ Feladatai

- Vonalak rajzolása
- Fontok renderelése

### ■ Hátrány

- Animáció készítése nehézkes
- Nincs szinkronizáció a framebufferrel
- Nincs 3D raszterizáció



# A WPF előtt

---

## ➤ GDI+

- Leváltja a GDI-t
- Windows XP-től
- Új funkciók
  - Élsimítás (2D)
  - Float koordináták
  - Gradient shading
  - JPG/PNG támogatás



# A WPF előtt

---

- Felhasználói élmény: User Experience
  - Nincs designer támogatás (sem eszköz sem ember)
  - Nem kényelmes/bővíthető a programozói modell
- Nagyon régi technológiákra épít
  - GDI (1985): hardveresen támogatott a műveletek nagy része, de keveset tud
  - GDI+ (2001, XP): csak szoftveresen, de van például élsimítás, transzparens megjelenés is
  - Direct2D (2008, Vista): hardveresen gyorsított de van software fallback
    - IE9
- Nem használja a mai grafikus kártyákat, rossz teljesítmény:
  - GDI: 10
  - GDI+: 1
  - DirectX/2D: 100-1000



# GDI „problémák”

---

- Raszter alapú megoldás
  - A rajzolás eredménye tipikusan egy bitmappre kerül rá
  - A rajzolás imperatív modellt követ: a kiadott utasítások azonnal végrehajtnak
- Nehézkes programozási modell
  - WM\_PAINT, újrarajzolás, egy szálon
  - A rajzolás az alkalmazás „idejéből” megy, a teljes felhasználói élmény függ a futó alkalmazásoktól
- Jól működött korlátozott memória esetén
  - Biztonság: közvetlenül a képernyőre rajzolhat
- Egyéb problémák
  - Rosszul skálázódik: DPI függő
  - Egész szám koordináták, stb



# Limitációk

---

## ➤ Limitált számú GDI objektum

### ■ Windows 95/98/Millennium

- 1200

### ■ Windows 2000

- 16384

### ■ Windows XP/Vista/7:

- 10000 / process (konfigurálható)

- 65536 / session

## ➤ GDI overflow



# GDI / GDI+ programozói modell

---

## ➤ GDI: C

1. GetDC: DeviceContext (rajzolási környezet)
2. SelectObject (Brush / Pen) -> old
3. DrawLine, ...
4. SelectObject ( old )

## ➤ GDI+: C++

1. GetDC
2. Graphics( DC )
3. New Pen / Brush
4. Graphics.DrawLine( ... pen, brush, ... )



# Az integráció hiánya

---

- Operációs rendszer szinten nincs integráció
  - Rajzolás (GDI/GDI+ / DirectX, OpenGL, ... )
  - Eseménykezelés (Window alapú)
    - Nincs köze a megjelenített objektumokhoz
  - Alkalmazás (nincs adatkötés)
- Egyéb sziget technológiák
  - Képfeldolgozás
  - Videó és hang
  - 3D grafika
  - Érintés, gesztusok, beszédfelismerés, ...
  - Dokumentum kezelés



# Programozói modell

---

- Vezérlők és a rajzolás nem objektum orientált
  - Nincs származtatás, egységbe zárás
    - Nincs újrafelhasználás, csak segéd függvények
  - Nincs polimorfizmus, nincsenek virtuális metódusok
    - Nehézkes eseménykezelés
- Nehéz a megjelenés testreszabása
  - Minden rajzolást újra kell írni
  - Imperatív rajzolási modell
- Nem komponálható vezérlők
  - A vezérlő nem meglévő másik vezérlőkből, elemekből áll össze



# Agenda

---

- „Problems” with GDI
- The new declarative modell
- Threading (Dispatcher)
- Dependency / Attached properties
- Visuals
- Physical architecture
- DPI independent coordinates



# Integráció

---

- Teljeskörű integráció
  - 2D / 3D grafika
  - Mediakezelés
  - Beszédfelismerés, érintéskezelés
- Hardveres gyorsítás
  - DirectX
  - Szoftveres render pipeline fallback
  - Új driver modell: WDDM, Vista-tól
- Felbontás független koordinátakezelés



# Objektum orientált megközelítés

---

- Minden vezérlő .NET-es osztály
  - Komplex öröklődő osztály hierarchia
  - A saját osztályok is ebbe épülnek bele
  - Jobban testreszabható származtatással és a virtuális metódusok átírásával
  - A funkcionalitás tisztábban megjelenik az egyes hierarchia ágakon
- A vezérlőhöz nem kötődik megjelenés
  - „Lookless controls”: a megjelenés testreszabható
  - Cél: a vezérlő csak a viselkedésért felelős



# Komponálhatóság

---

- A vezérlők tetszőleges tartalmat meg tudnak jeleníteni
  - Például: Button, Content
  - Content bármi lehet: szöveg, kép, más vezérlők:
    - StackPanel + szöveg + videó
- Az összetett vezérlők más vezérlőkből állnak
  - Az összeépítésre nagyon kevés megkötés van
  - Például: ListView
    - ScrollView a görgetéshez
    - Minden csomópont egy másik tetszőleges vezérlő



# Deklaratív rajzolási modell

---

- A megjelenítendő tartalmat objektumok írják le
  - A programozó az objektumokat hozza létre és azok tulajdonságait állítja be
  - Az objektumok az alkalmazás élelciklusa alatt végig léteznek és elérhetőek a program számára
    - Vs. GDI - mentés Windows metafile-ba
- Alap vizuális elemek
  - Geometriai formák: vonal, téglalap, poligon, stb.
  - Szöveg
  - Videó, hang
  - Effektek



# Demo

---



# Testreszabható megjelenés

---

- Deklaratíván, objektumokkal leírható az egyes vezérlők megjelenése
  - Vonal, effektek, stb.
- Leírhatók a különböző állapotokhoz tartozó változások
  - Például letiltott vezérlő
- Az állapotok eseményekhez köthetők
  - Például megnyomják a gombot
- A változások animálhatók



# Agenda

---

- „Problems” with GDI
- The new declarative modell
- Threading (Dispatcher)
- Dependency / Attached properties
- Visuals
- Physical architecture
- DPI independent coordinates



# STA – egy szálú működés

---

- GUI: vezérlők állapotai, eseménykezelés
- Többszálú megközelítés:
  - Az állapot módosító metódusok párhuzamosan futhatnak – nagyon sok zárolás, konzisztencia probléma
  - Eseménykezelők futása – manuálisan kiváltott események párhuzamosak lesznek a felhasználóval
- Megpróbálták
  - Túl lassú
  - Nehéz programozói modell
  - Nagyon hibák – nem determinisztikus deadlockok



# Architektúra – osztályok

---

- Minden objektum (vezérlő stb) egy szárhoz van rendelve, csak arról a szálról elérhető
  - Így van WinFormsban és XWindowban is
  - A DispatcherObject tartja nyilván
- System.Threading.DispatcherObject
  - A legtöbb WPF osztály őse
  - Párhuzamosság ellenőrzés: VerifyAccess, CheckAccess
  - Dispatcher : prioritásos üzenetsor



# Dispatcher - használat

```
ThreadStart start = delegate
```

```
{
    status.Text = "...";
};
new Thread(start).Start();
```

```
DispatcherOperation op =
    Dispatcher.BeginInvoke(
        DispatcherPriority.Normal,
        new Action<string>(SetStatus),
        "... (Async)");
```

```
DispatcherOperationStatus status = op.Status;
while (status != DispatcherOperationStatus.Completed)
{
    status = op.Wait(TimeSpan.FromMilliseconds(1000));
    if (status == DispatcherOperationStatus.Aborted)
    { // Alert }
}
```

```
ThreadStart start = delegate
```

```
{
    Dispatcher.Invoke(
        DispatcherPriority.Normal,
        new Action<string>(SetStatus),
        "...");
};
new Thread(start).Start();
```



# Agenda

---

- „Problems” with GDI
- The new declarative modell
- Threading (Dispatcher)
- Dependency / Attached properties
- Visuals
- Physical architecture
- DPI independent coordinates



# DependencyProperty

---

- A property rendszer további jellemzői
  - Egységes, kiszámítható funkcionalitás
  - Optimális tárolás
  - Aszimmetrikus teljesítmény
- Érték precedencia (egyszerűsített !)
  1. Animáció
  2. Explicit értékállítás XAML-ből vagy programozottan
  3. Stílus
  4. Öröklés: a logikai fában lévő szülőtől
  5. Alapértelmezett érték



# A property keretrendszer

- Keretrendszer szolgáltatások
  - Adatkötés, változás értesítési mechanizmussal (INPC)
  - Animáció, sablon, stílus, trigger, öröklés, default érték, ...
- WPF specifikus property rendszer
  - *DependencyObject* : alaposztály
  - *DependencyProperty* : a property leírója

```
public static readonly DependencyProperty BackgroundProperty =  
    DependencyProperty.Register(„Background”, typeof( Brush ),  
        typeof( Control ), ... FrameworkPropertyMetadataOptions.AffectsRender,  
        ...Brushes.White, ... validationCallback );  
  
...  
  
public Brush Background  
{  
    get { return (Brush)this.GetValue( BackgroundProperty ); } }
```



# Attached Property

- Általában olyan objektumhoz tartozó beállítás amit a más (pl Parent) használ, például:
  - Szülő: *DockPanel* osztály
  - Információ: melyik oldalra dokkoljon
  - AttachedProperty: *DockProperty*, *Dock.Left*, stb.
  - Bármelyik gyerek elemen be lehet állítani

```
<DockPanel>  
  <Button DockPanel.Dock=„Left" />
```

```
DockProperty = DependencyProperty.RegisterAttached( ... );  
DockPanel.SetDock( uiElement, Dock.Left );  
uiElement.SetValue( DockPanel.DockProperty, Dock.Left );
```

- Általános, kiterjeszthető



# Agenda

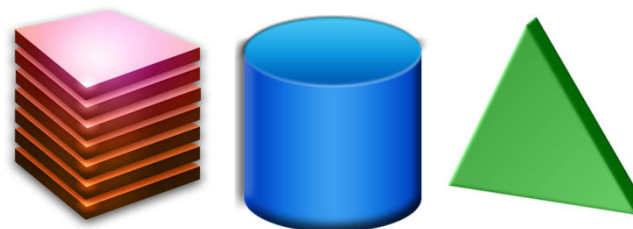
---

- „Problems” with GDI
- The new declarative modell
- Threading (Dispatcher)
- Dependency / Attached properties
- Visuals
- Physical architecture
- DPI independent coordinates



# WPF megjelenítési csatornák

---



Segoe UI

Szöveg





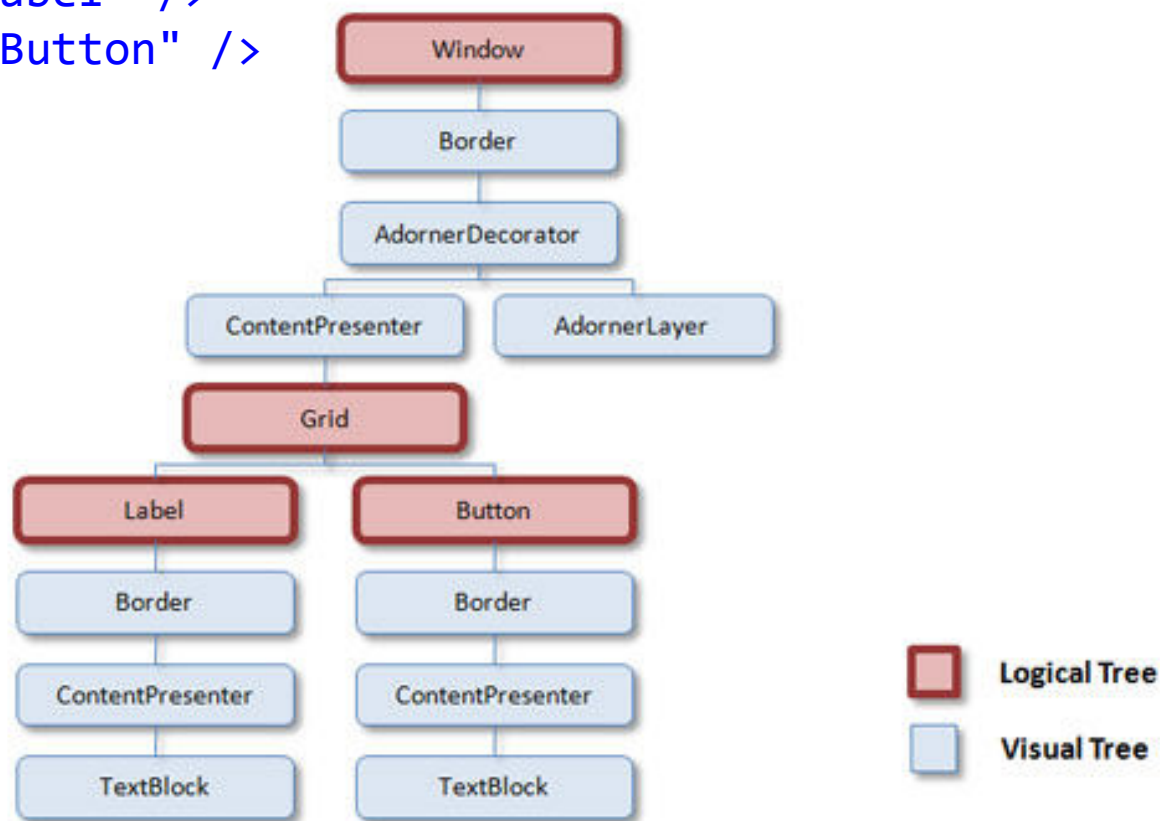
# A vezérlőktől a grafikus primitívekig

- Logical tree: a vezérlők hierarchiája
  - Alapvetően a XAML definiálja, tartalom modell
- Visual tree: a megjelenítésért felelős Visual-ök fája
  - Visual: a megjelenés vektorgrafikus összállításáért felel
- A UIElementek tipikusan az OnRender metódusban adják meg megjelenésüket
  - A rajzolt primitíveket a Visual osztály tárolja el
  - A rajzolás a DrawingContexttel történik



# Visual tree és logical tree

```
<Window>
  <Grid>
    <Label Content="Label" />
    <Button Content="Button" />
  </Grid>
</Window>
```





# DrawingContext

---

- A DrawingContext tipikus metódusai:
  - DrawLine, DrawEllipse, DrawGeometry, ...
  - DrawText, DrawGlyphRun
  - DrawVideo, PushEffect, PushOpacityMask, ...
    - Paraméterként animáció is megadható !
- A DrawingContext készíti el és tárolja a grafikus primitíveket a Visual osztályban
- A Visualtól Drawing osztályokként lehet lekérdezni a tárolt vektorgrafikus tartalma



# Visualból örökölt osztályok

---

- DrawingVisual: alakzat, kép, szöveg renderelés
- DrawingVisual3D: 3D
- ContainerVisual: más Visualöket tartalmaz



# Visual osztály

---

- System.Windows.Media.Visual
  - Nyilvántartja a rajzolt grafikus primitíveket
  - Rajzolás a képernyőre
  - Kapcsolat a managed API és a milcore között
    - *Composition node*
  - Metaadatok
    - Vágás, transzformáció, bounding rect, hittesting
  - Fa szerkezet => *Visual Tree*
    - Z-order, alfa csatorna, stb



# Szöveg megjelenítés

---

- `DrawingContext . DrawText( szöveg, pozíció )`
  - Szöveg megjelenítés `MilCore` szinten, `DirectX`-szel
- `FormattedText`: a formázott szöveget reprezentál
  - Betűtípus, méret, fékövér, dőlt, aláhúzás, stb.
  - `Brush`: tetszőleges kitöltés
  - A fenti tulajdonságok akár betűnként változhatnak
  - Méretek, max szélesség, sortáv, igazítás, trimming, ...
  - Lekérdezések: méretek, soronkénti szélességek
  - *`BuildGeometry`*: alakzat készítése a szövegből



# Grafika készítése

---

1. Közvetlenül a `DrawingContext` metódusaival
  - `UIElement` esetén override-olni kell az `OnRender`-t
2. `DrawingVisual`: 'alakzat' megjelenítésére képes
  - `Drawing` osztályok készítésével, például `VideoDrawing`, `GeometryDrawing`, ...
  - A `GeometryDrawing` 2D alakzatokat tartalmaz
    - `Brush` és `Pen` az alakzatok kontúrjához és kitöltéséhez
    - Például `LineGeometry`: végpont koordináták
3. Vagy `Shape` osztályokkal, például `Line`, `Rectangle`
  - Eseménykezelés, layout, transzformációk, stb.



# Pen és Brush

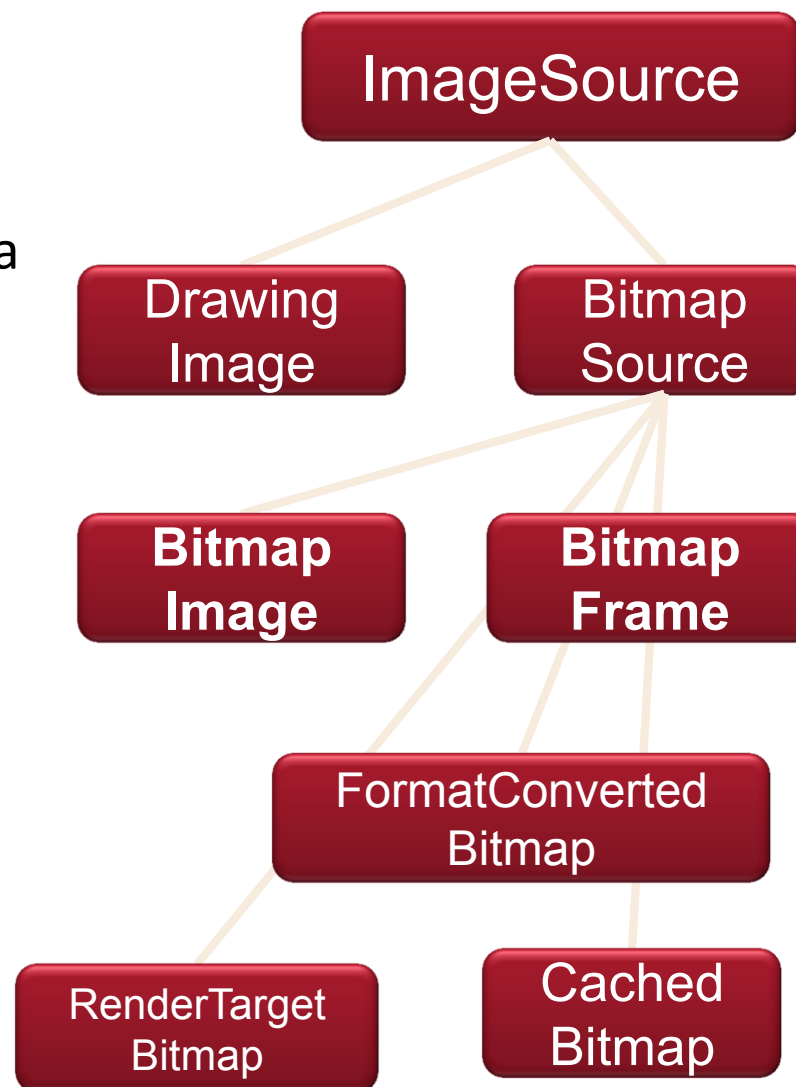
---

- Pen (vagy Stroke): kontúr vonalak formázása
  - Vonalvastagság, végpontok és találkozások
  - Vonalkitöltés, ami egy Brush
- Brush: alakzat kitöltésének formázása
  - SolidColorBrush, Linear/RadialGradientBrush
  - ImageBrush: kép (tipikusan bitmap)
  - DrawingBrush: alakzat, videó, szöveg – skálázódik
  - VisualBrush: egy vezérlő aktuális képe
  - Tile, stretch, transform, stb.



# Kép kezelés

- ImageSource alaposztály
  - Szélesség, magasság, metaadatok
  - ImageBrush, ImageDrawing használja
- DrawingImage
  - vektorgrafikus alakzat van rajta
- BitmapSource: rasztergrafikus kép
- BitmapImage: kép forrás, pl. fájl
  - XAML-ben jól használható
- <Image>: kép megjelenítése
  - XAML-ben
  - ImageSource adja meg a forrást





# Párhuzamok

---

- Leíró osztályok – adat jellegűek
  - LineGeometry, GeometryGroup, MeshGeometry3D
- Grafikus primitíveket leíró osztályok
  - Megjelenítéshez szükséges attribútumok (pl. Brush)
  - MediaPlayer, ImageSource, FormattedText, Drawing (pl. GeometryDrawing), GeometryModel3D, ...
- Framework szintű osztályok: layout, események
  - Shape, MediaElement, ModelUIElement3D
- DrawingContext – DeviceContext, Graphics, HDC



# Elnevezési konvenciók

---

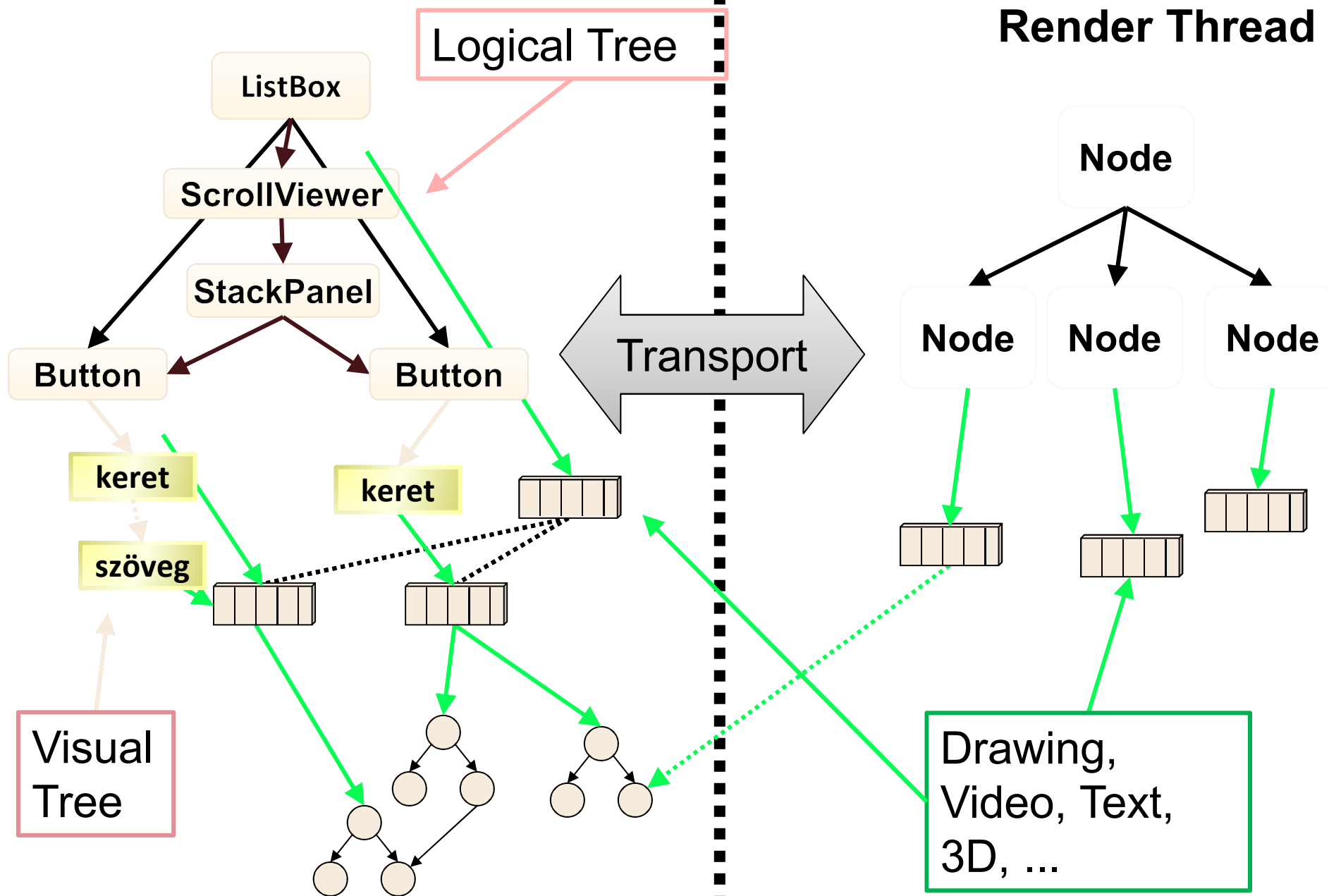
- *Drawing*Visual
  - „olyan Visual osztály amiben *drawing* van”
  - ‘*alakzatot*’ megjelenítő Visualból származó osztály
- ImageSource: kicsit csalóka, csak Image-nek hívják
  - BitmapImage (-Source): raszteres kép
  - DrawingImage: alakzatot megjelenítő vektorgrafikus kép
- ImageDrawing: képet reprezentáló ‘alakzat’
- GeometryDrawing: alakzatot reprezentáló ‘alakzat’
- LineGeometry: vonalat reprezentáló geometria



# Agenda

---

- „Problems” with GDI
- The new declarative modell
- Threading (Dispatcher)
- Dependency / Attached properties
- Visuals
- Physical architecture
- DPI independent coordinates



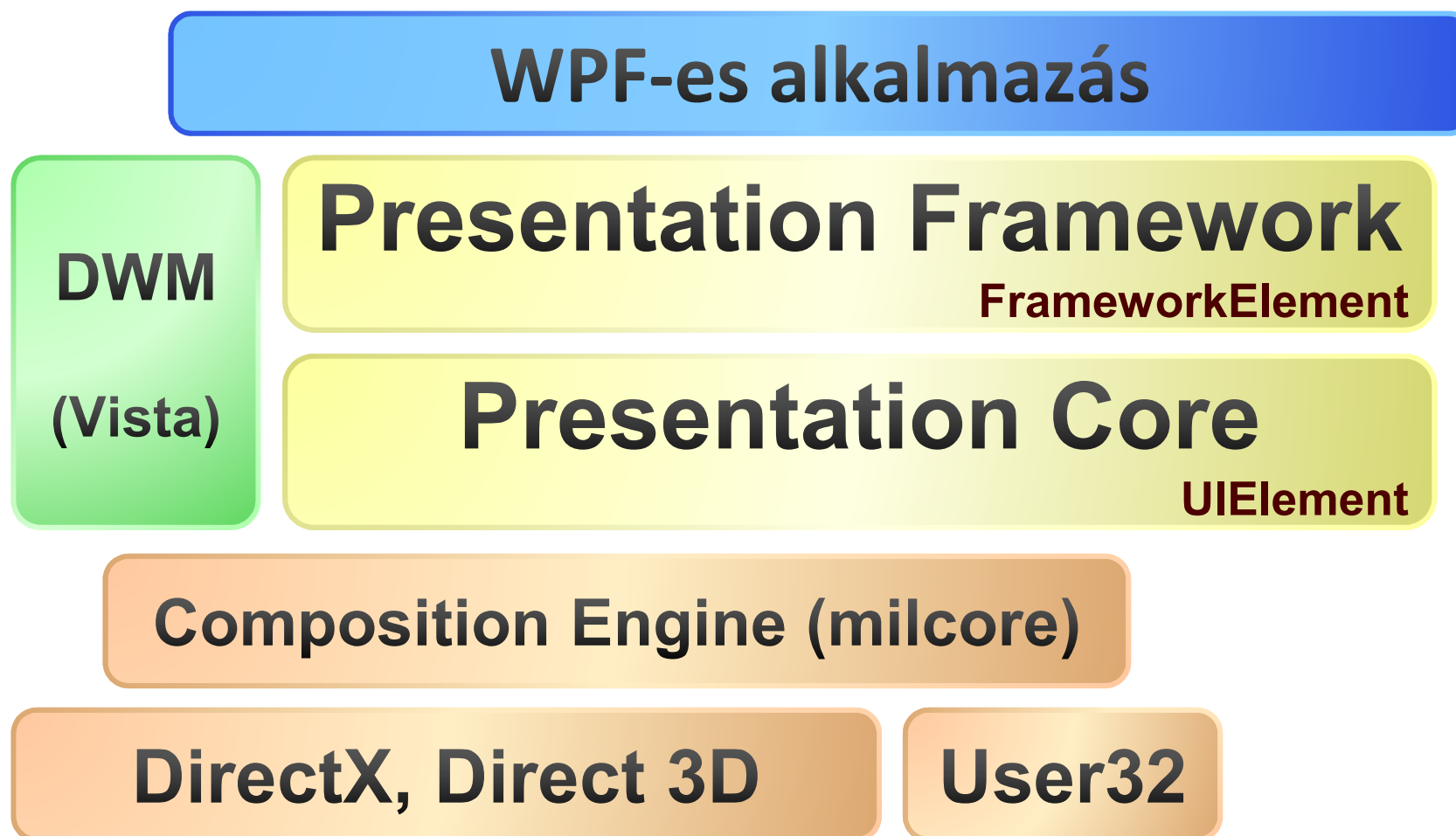


# Vektorgrafikus megjelenítés

---

- WPF: vektorgrafikus felhasználói felület modell
  - Az alkalmazásban nincs WM\_PAINT !
  - Minden vezérlő grafikus primitívekből épül fel
    - Win32/WinForms: csak a vezérlők épülnek hierarchiába.
    - *WPF esetén a vezérlőket alkotó grafikus elemek is objektumok és hierarchiába szerveződnek !*
    - Integráció: 3D objektumok, szöveg, kép, videó, effekt !
  - A grafikus elemek hierarchiába rendeződnek, fát alkotnak → megjelenés deklaratív modellje, VisualTree
  - A grafika végül egy **DirectX**-es felületen jelenik meg

# képpont a képernyőre ?

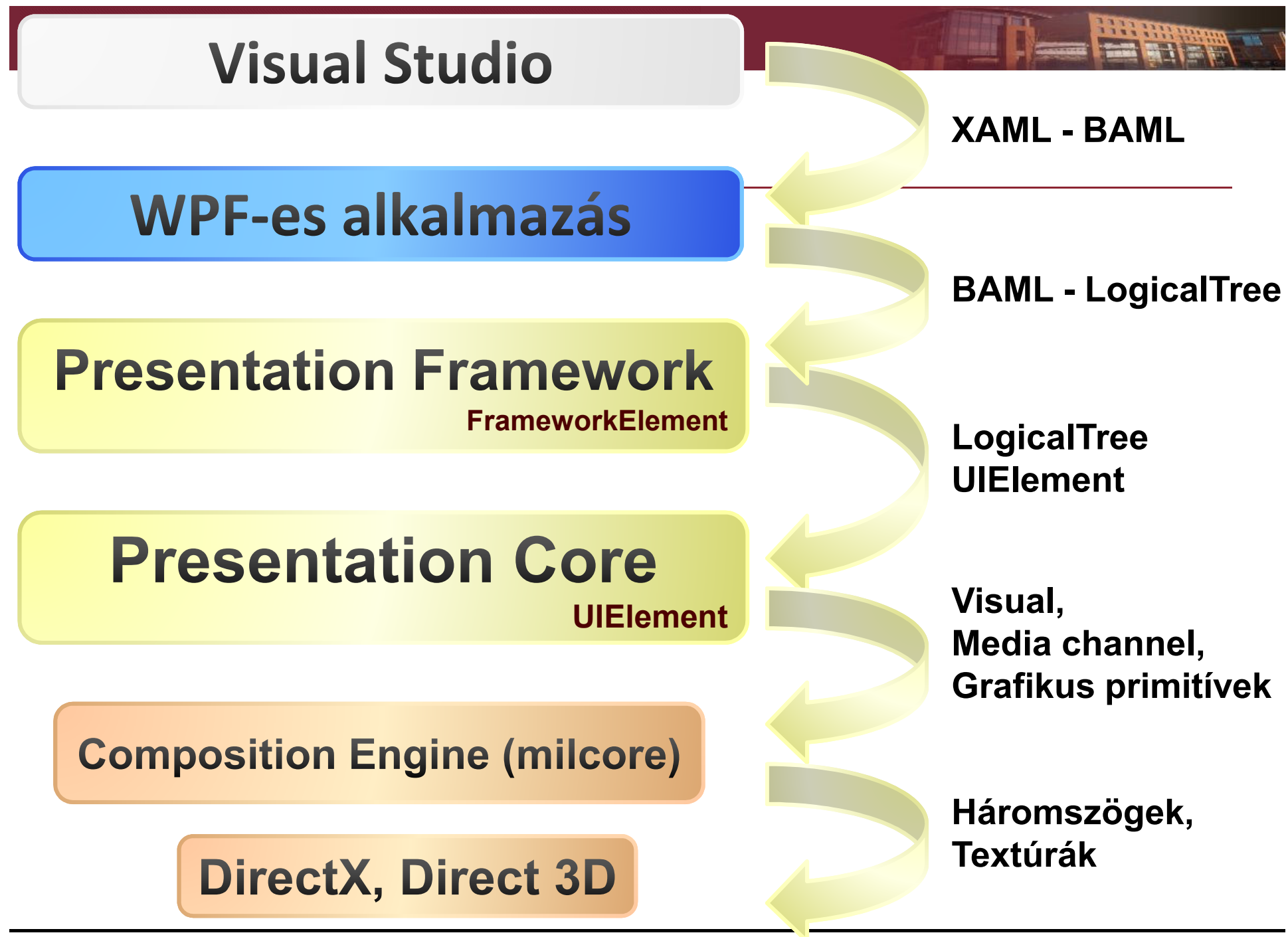




# WPF architektúra

---

- PresentationCore: megjelenítés, esemény kezelés
  - MilCore: grafikus primitívek DirectX alapú renderelése
  - *Grafikus primitívek: média-, alakzat- és képkezelés*
  - *Visual: a renderelés felügyelt interfésze, hit-testing*
  - UIElement: pozicionálás, eseménykezelés, animáció
- PresentationFramework: layout, adatkötés, vezérlők
  - FrameworkElement: stílus, storyboard, triggererek
  - Layout panelek: StackPanel, Grid, DockPanel, ...
  - Vezérlők: Button, TextBox, Slider, ...





# Aero, WPF és a DWM

---

- Aero Glass – Vista Home Basic feletti változatok felhasználói felülete
  - Áttetsző „non-client” terület – pixel shader
  - Win + TAB : flip 3D – 3D felületek
  - Live thumbnails
- Ki a felelős?
  - A Desktop Window Manager (DWM)
    - Natív komponens



# Desktop Window Manager (DWM)

---

- A teljes desktop egy visual tree
- Minden toplevel ablak két csomópont
  - „non-client” és a „client” terület
  - A „client” terület WPF esetén az a *megosztott* DirectX-es felület amire az alkalmazásban futó milcore.dll rajzol
  - Régi alkalmazásnál egy memória puffer
    - A HDC oda rajzol, nem közvetlenül a primary buffer-re
    - Miért? Pixel formátum konverzió, XOR, egyéb írás/olvasás műveletek lassúak lennének. GDI nem tud DirectX felületre közvetlenül írni.
    - A memória puffer időnként átkerül a videómemóriába, textúraként használja a DWM úgy, mint WPF esetén



# Windows Display Driver Model (WDDM)

---

- Az új videó driver modell, DirectX 10.0
- WDDM nélkül nincs DWM !
  - Aero glass, thumbs, flip3d, stb.
- Főbb funkciók
  - Videómemória virtualizáció
    - Nem fogy el, csak lassú lesz ☺
  - Osztott DirectX felület támogatás (shared DX surface)
    - A WPF-es alkalmazás saját felületét a DWM közvetlenül használja textúraként megjelenítéshez
  - Megszakítható GPU műveletek
  - Újrainduló videódriver !



# Agenda

---

- „Problems” with GDI
- The new declarative modell
- Threading (Dispatcher)
- Dependency / Attached properties
- Visuals
- Physical architecture
- DPI independent coordinates



# High DPI támogatás

---

- Ha a Windows nem 96 DPI-n fut, akkor *minden korábbi* alkalmazást úgy futtat, mintha 96 DPI lenne, és a képét felnagyítja
  - Módosítja az egér koordinátákat, stb.
  - Csak a kész DirectX-es képet tudja nagyítani ...
  - ... így a szöveg elmosódik ☹
  - Kikapcsolható ! 😊
    - Alkalmazás Property oldala, Compatibility fül, „Disable display scaling on high DPI settings”
    - Fejlesztők: manifeszt fájlal vagy API hívással



# DPI semleges programozási modell

- Cél: ablak méretezése felbontástól függetlenül !
  - Ehhez a Windows-ban megfelelő DPI-t kell beállítani
  - Például: 17"-os LCD, 1600x1200 → 120 DPI
- WPF: *mindent* csak képpont független egységben lehet megadni (Device Independent Pixel, DIP)
  - A hüvelyk 1/96-od része
  - Az egységhez tartozó pixelek számát a Windows DPI beállítása határozza meg
    - Alapértelmezett a 96 DPI, így egy egység pont egy pixel
    - Megfelelő beállításnál a DIP valódi hosszt jelent
  - *Mindenütt*: egér koordináta, vezérlő elemek mérete, betűméret, grafika, rendszer méretek, stb.



# Képernyőképek különböző DPI-vel

**72 DPI****24pt Text**

200 x 25 Rectangle



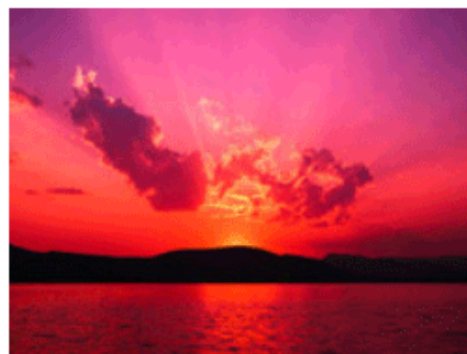
200 x 150 Image

**96 DPI****24pt Text**

200 x 25 Rectangle



200 x 150 Image

**120 DPI****24pt  
Text**200 x 150  
Image

200 x 25 Rectangle





# Lebegőpontos méretek

---

- Mindenütt DIP
- Mindenütt dupla pontos lebegőpontos számok
- XAML-ben beépített konverterek (in, cm, pt)
  - Például: `<TextBlock Width=„12 cm" />`
- **Vigyázat!** A double nem úgy viselkedik mint az egész szám!
  - Vigyázzunk a ==, !=, <=, >= operátorokkal !
  - A kerekítés és ábrázolás meglepetéseket okozhat !



# WPF – XAML, vezérlők

---



# Agenda

---

1. WPF architektúra, GDI, DirectX, DWM
  - Többszálúság, property system
  - Grafikus primitívek, rajzolás alacsony szinten
  - Média/kép kezelés
2. Vezérlő hierarchia, XAML, layout
3. Adatkötés, sablonok, erőforrások, stílusok, triggerek, eseménykezelés
4. MVVM, 3D, dokumentum kezelés



# Agenda

## ➤ XAML

- XAML részletesen
- Vezérlők hierarchiája
- Content controls
- HeaderedContentControls
- Items controls
- HeaderedItemsControls
- Range controls
- Text and link controls



# XAML alapok

---

<Button

```
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
Content="OK" Click="button_Click"/>
```

```
using System.Windows.Controls;  
using System.Windows;  
Button b = new Button();  
b.Click += new RoutedEventHandler(button_Click);  
b.Content = "OK";
```



# XAML

---

- XAML: XML alapú, deklartív objektum példányosító nyelv
  - A WPF mellett a Workflow Foundation is ezt használja
  - Nincs mágia, nincs kódból el nem érhető funkció
- WPF esetén a programozók és designerek, fejlesztőeszközök és designer eszközök közös nyelve
- DE: a WPF vektorgrafikus modellje NEM a XAML-ből fakad!
  - A XAML „csak” egy eszköz, nyelv a felhasználói felület deklaratív leírására
  - Kicsit hasonlít az XHTML-re, de sokkal kifejezőbb, erőteljesebb
  - Elemek hierarchiája, layout, grafika, média, animáció, adatkötés, triggererek, stílusok, ... mind leírhatók benne!



# XAML alapok 1.

---

- A tegek osztályokat példányosítanak
  - Nem WPF specifikus !
- Az attribútumok tulajdonságokat állítanak be
- Névterek, szerelvények
  - Az alapértelmezett névtér a WPF vezérlők névtére
  - Az 'x' névtér az általános XAML motor
    - például az x:Name – a generált változó neve
  - Egyéb szerelvények és névtereik importálhatók
    - Bármelyik szerelvényhez attribútummal névtér rendelhető



## XAML alapok 2.

- Egy teg tartalma (gyerek elemek) az „alapértelmezett” tulajdonság értékét határozza meg
- Az attribútum értelmezéséhez lehet írni konvertert

```
<TextBlock Width=„12 cm" />
```

- Az attribútum értéke lehet összetett objektum is
  - Pl: LinearGradientBrush
  - (PropertyElementSyntax)
- Eseménykezelők

**Cél:**

*XAML-t NE KELLJEN KÉZZEL ÍRNI !*

***2009: Vagy mégis???***



# XAML alapok

```
<Button>
  <Button.Content>
    OK
  </Button.Content>
</Button>
```



```
<Button>
  OK
</Button>
```

```
<Button>
  <Button.Content>
    <Rectangle Height="40"
      Width="40" Fill="Black"/>
  </Button.Content>
</Button>
```



```
<Button>
  <Rectangle Weight="40"
    Width="40" Fill="Black"/>
</Button>
```



# A XAML feldolgozása

---

- A XAML-t a fordító tömör bináris formává alakítja:  
BAML
  - Előtte kétfázisú fordítás idejű ellenőrzés
  - Futásidőben már nincs XML értelmezés
- A BAML erőforrásként kerül tárolásra
- A WPF keretrendszer gondoskodik a BAML értelmezéséről és annak alapján az objektumfa és a kötések létrehozásáról
  - A XAML-ből nem generál közvetlenül kódot



# Agenda

---

## ➤ XAML

### ■ XAML részletesen

## ➤ Vezérlők hierarchiája

### ➤ Content controls

### ➤ HeaderedContentControls

### ➤ Items controls

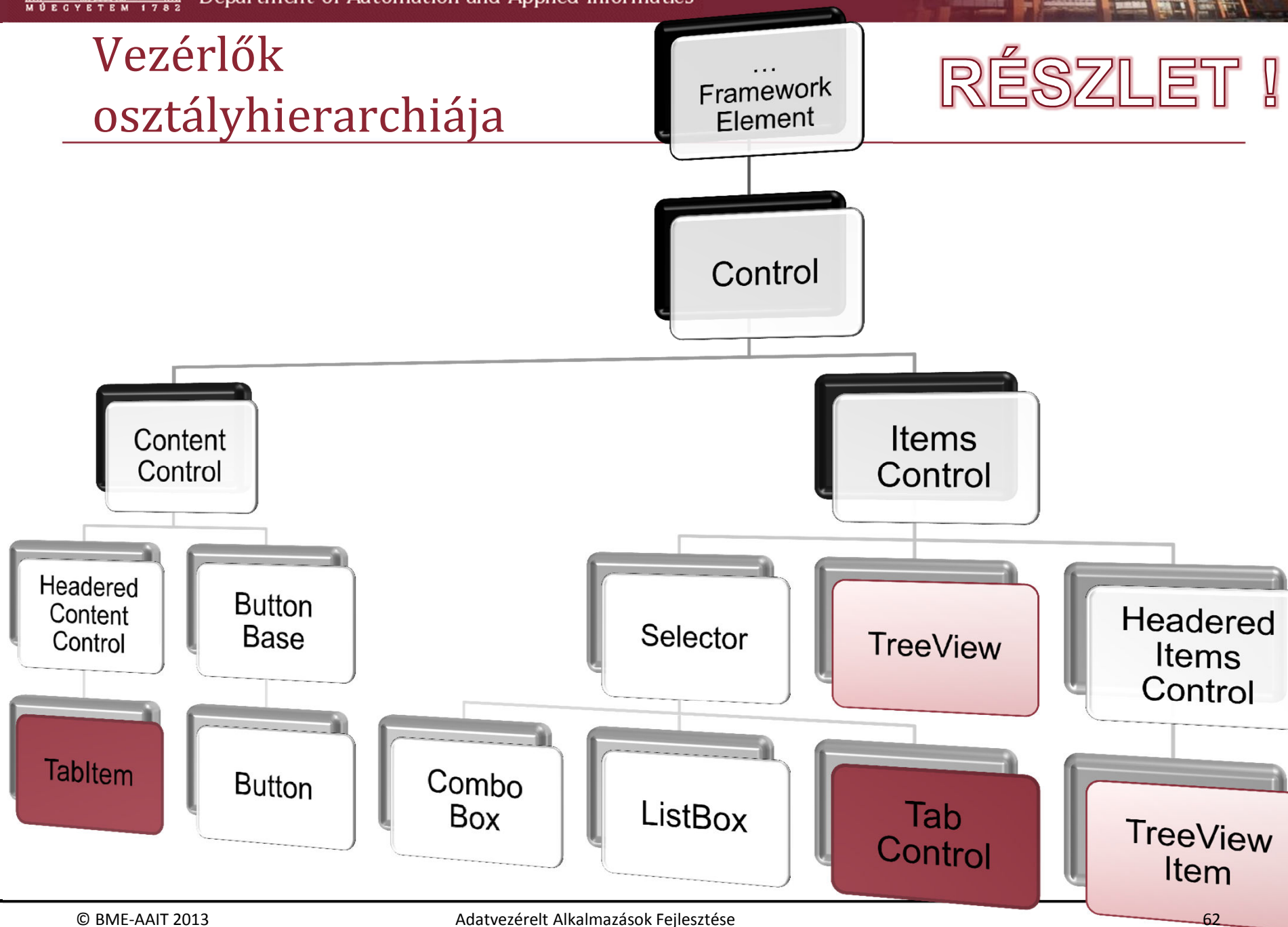
### ➤ HeaderedItemsControls

### ➤ Range controls

### ➤ Text and link controls

# Vezérlők osztályhierarchiája

RÉSZLET !





# A megjelenés testreszabása

---

- Minden vezérlő **tartalma** testreszabható
- Többféle tartalom a vezérlő típusától függően
  - Egyszerű, például gomb ( : ContentControl)
  - Fejléces, például tabfül ( : HeaderedContentControl)
  - Listás, például ListBox ( : Selector : ItemsControl)
  - Hierarchikus, pl. TreeView ( : HeaderedItemsControl)
- A tartalmat a ContentPresenter jeleníti meg
  - Alapértelmezett tartalom: ToString() – szöveg
  - De testreszabható !



# Agenda

---

- XAML
  - XAML részletesen
- Vezérlők hierarchiája
- Content controls
- HeaderedContentControls
- Items controls
- HeaderedItemsControls
- Range controls
- Text and link controls



# ContentControl : Control

---

- Content tulajdonság (object): a tartalom
  - 1. ContentTemplate adott: használja a sablont
  - 2. UIElement: megjeleníti azt az objektum fát
  - 3. Ellenőrzi a típus alapú DataTemplate-eket
  - 4. ToString-et használ
    - ContentStringFormat: formázó sztring
- ContentTemplate (DataTemplate)
  - A tartalom megjelenítését lehet vele testreszabni
- ContentTemplateSelector: sablonválasztó logika



# ButtonBase : ContentControl

---

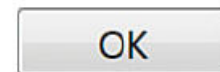
- Kezeli az egéreseeményeket
- Tulajdonságok
  - IsPressed
  - ClickMode: Release, Press, Hover
- Integrálódik a Command mintával
  - Command, CommandParameter, CommandTarget



# Button, RepeatButton : ButtonBase

---

## ➤ Button



- IsDefault, IsCancel
- Handles OnClick

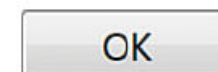
## • RepeatButton

- Delay property
- Interval property
- Nyomva tartva többször lövi el a Click eseményt
- System.Windows.Controls.Primitives névtérben van



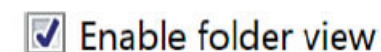
# ToggleButton, CheckBox

## ➤ ToggleButton : ButtonBase



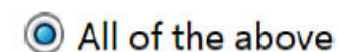
- IsChecked: lehet null is
- IsThreeStat

## ➤ CheckBox : ToggleButton



- Saját megjelenése van

## ➤ RadioButton : ToggleButton



- Saját megjelenése van
- GroupName-el lehet csoportba foglalni őket és csak egyet lehet kiválasztani



# Label : ContentControl

---

- Szöveget jelenít meg
- Target property
  - A focus beállítja a hivatkozott vezérlőre
  - A gyorsbillentyűt az aláhúzás karakter jelzi
    - AccessKeys : mnemonics
- Máshol is használhatók a gyorsbillentyűk
  - AccessText-et kell a Content helyére tenni
  - Állítható a megjelenése



# UserControl : ContentControl

---

- Más vezérlőket fog össze és jelenít meg
  - Nincs template támogatás: korlátozott külső testreszabhatóság
- Nincs további saját tulajdonsága vagy viselkedése



# Agenda

---

- XAML
  - XAML részletesen
- Vezérlők hierarchiája
- Content controls
- HeaderedContentControls
- Items controls
- HeaderedItemsControls
- Range controls
- Text and link controls



# HeaderedContentControl : CC

---

- A tartalom mellett van egy fejléc is (Header)
  - Header és HeaderFormatString tulajdonságok
  - HeaderTemplate, HeaderTemplateSelector



# GroupBox : HCC

---

➤ Rajzol egy keretet a tartalom köré

## Grammar

- ☐ Check grammar as you type
- ☐ Hide grammatical errors in this document
- ☐ Check grammar with spelling

## Grammar

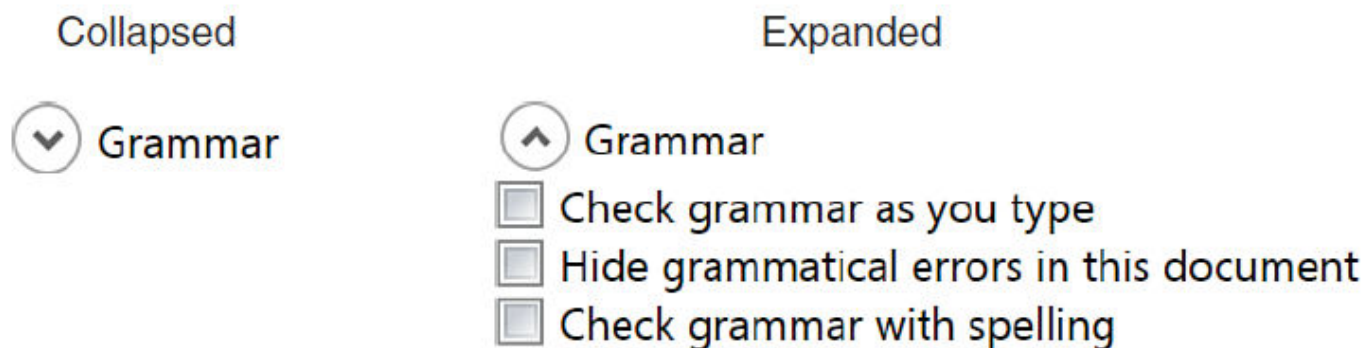
- ☐ Check grammar as you type
- ☐ Hide grammatical errors in this document
- ☐ Check grammar with spelling



# Expander : HCC

---

- Becsukható / kinyitható tartalom
  - A fejléc mindig látszik
- IsExpanded tulajdonság
- ExpandDirection: négy irány
- Expanded / Collapsed események





# Agenda

---

- XAML
  - XAML részletesen
- Vezérlők hierarchiája
- Content controls
- HeaderedContentControls
- Items controls
- HeaderedItemsControls
- Range controls
- Text and link controls



# ItemsControl : Control

---

- Items (ItemCollection típus)
  - Ha ItemsSource megvan adva, akkor read-only
  - Egyébként itt lehet beállítani a tartalmat
- ItemsSource írható property
  - Ha null, akkor Items-et használ
- HasItem, IsGrouping
  - Items tulajdonság (ItemCollection (:CollectionView)) képes csoportosítani, rendezni, stb.
- DisplayMemberPath
  - Az elemek megjelenítését egy property-hez kötjük
  - Ha nincs megadva és nincs ItemTemplate, akkor a ToString-et jeleníti meg (type)



# Container osztályok

---

- Minden megjelenített listaelemhez van egy Container osztály
  - Pl ListBoxItem, ami egy ContentControl
- Az ItemContainerGenerator-on keresztül lehet elérni a container itemeket
  - Explicit is létre lehet hozni (a forráshoz hozzáadni), akkor a logical tree része lesz, enumerálni lehet
- A container elemek felelősek a kiválasztott elem háttérszínéért, stb.
- Az ItemsPanellel változtatható az elrendező panel



# Selector : ItemsControl

---

- Kiválasztható elemek
- SelectedIndex: 0 alapú index
- SelectedItem
  - A kiválasztott objektum
- SelectedValue
  - SelectedValuePath által hivatkozott tulajdonsága a kiválasztott objektumnak
- Attached property:
  - IsSelected
  - IsSelectedActive -> fókusz rajta van-e a kiválasztott elemen
  - SelectionChanged esemény



# ListBox : Selector

---

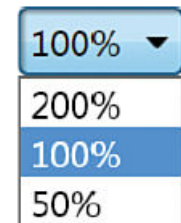
- SelectionMode
  - Single/Multiple/Extended (Shift-et vagy a Ctrl-t le kell nyomni ha ki akarunk többet jelölni))
- ScrollViewer . CanContentScroll





# ComboBox : Selector

- DropDownOpened, DropDownClosed események
- IsDropDownOpen (read-only prop)
- IsEditable
- IsReadOnly
- TextSearch.TextPath ComboBox-ban
- TextSearch.Text -> ComboBoxItem-ben





# ListView : ListBox

---

## ➤ View property

■ Például Grid

| Date     | Day of Week | Year |  |
|----------|-------------|------|--|
| 1/1/2007 | Monday      | 2007 |  |
| 1/2/2007 | Tuesday     | 2007 |  |
| 1/3/2007 | Wednesday   | 2007 |  |



# Agenda

---

- XAML
  - XAML részletesen
- Vezérlők hierarchiája
- Content controls
- HeaderedContentControls
- Items controls
- HeaderedItemsControls
- Range controls
- Text and link controls



# HeaderedItemsControl : ItemsControl

---

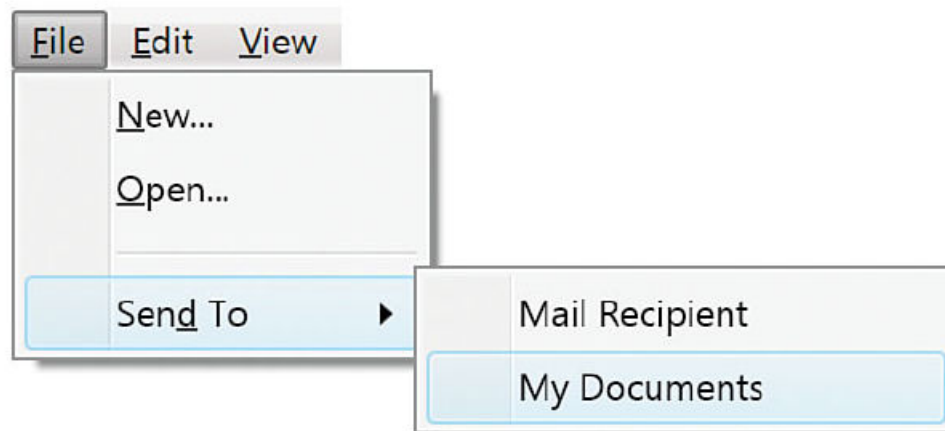
- Header tulajdonság a fejléchez



# Menu : HIC

## ➤ Tulajdonságok

- Icon
- InputGestureText
- IsCheckable
- Separator

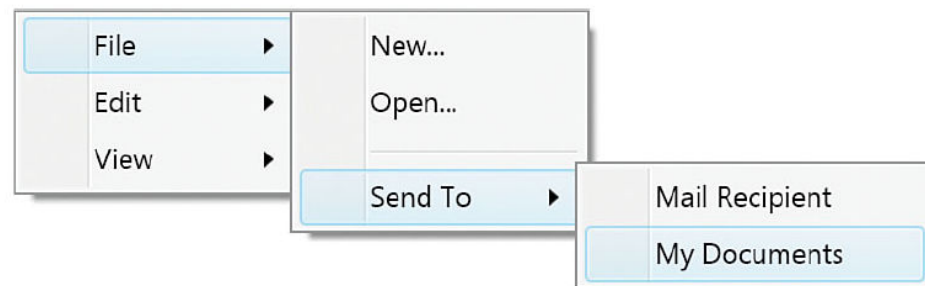


## ➤ IsMainMenu ->ha true, akkor Alt-ra vagy F10-re fókuszál



# ContextMenu : HIC

- Ugyanaz jellemző rá, mint a menu-re.
- Open/Closed események
- ContextMenuService.ShowOnDisabled





# Agenda

---

- XAML
  - XAML részletesen
- Vezérlők hierarchiája
- Content controls
- HeaderedContentControls
- Items controls
- HeaderedItemsControls
- Range controls
- Text and link controls



# RangeBase

---

- Max, min
- SmallChange, LargeChange
- Value lekérdezés, validálás
- ValueChanged esemény



# Range Controls

---



IsIndeterminate, ha true, akkor folyton megy a zöld csík



# Range Controls

## Slider:



- TickPlacement (TopLeft, BottomRight, None)
  - A vonalkák hol legyenek
- IsSelectionRangeEnabled
- SelectionStart, SelectionEnd



```
<Slider Minimum="0" Maximum="10" Value="5"
    TickPlacement="BottomRight"
    IsSelectionRangeEnabled="True"
    SelectionEnd="8" SelectionStart="2"/>
```



# Agenda

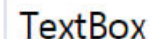
---

- XAML
  - XAML részletesen
- Vezérlők hierarchiája
- Content controls
- HeaderedContentControls
- Items controls
- HeaderedItemsControls
- Range controls
- Text and link controls



# Text and Ink Controls

## TextBox:

TextBox

- Copy,Paste,Cut,Undo,Redo és spell checking alapban benne van
- SpellCheck.IsEnabled
- TextWrapping (Wrap/WrapWithOverflow) (Wrap esetén ha egy szó nem fér ki a sorban, akkor törik, WrapWithOverflow esetén nem engedi szétszedni a szót)

## RichTextBox:

*Rich*TextBox

- FlowDocument-ben tárolja a szöveget
- Sokkal többet tud, mint a TextBox
- Lesz rá példa később!

## PasswordBox:



- Password-ben tárolja a szöveget!
- PasswordChanged esemény



# Text and Ink Controls

---

- EditingMode:
  - Ink
  - InkAndGesture
  - GestureOnly
  - EraseByStroke
  - EraseByPoint
  - Select
  - None
- 15 esemény
- DrawingAttributes-al testre szabhatóak a Stroke-ok
- Nagyon gazdag vezérlő





# Agenda

---

## ➤ XAML

- XAML részletesen

## ➤ Vezérlők hierarchiája

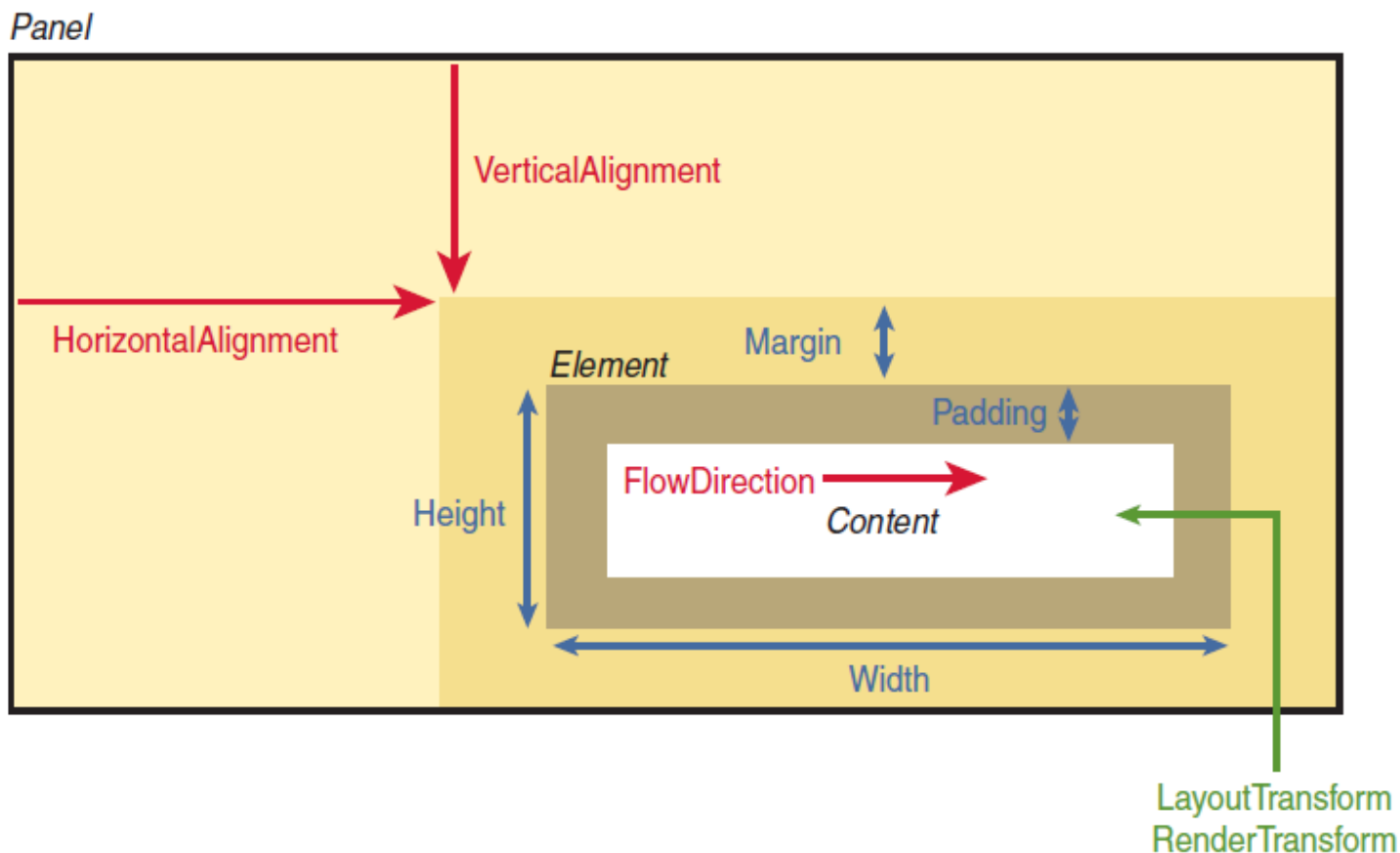
- Content controls
- HeaderedContentControls
- Items controls
- HeaderedItemsControls
- Range controls
- Text and link controls

## ➤ Elhelyezés

- Méretezés, pozicionálás
- Transzformáció
- Layout engine



# Méretezés





# Felhasználói felület, layout

---

- Gyerek elemeken lévő általános módosítók
  - Horizontal/Vertical Alignment
    - A szülőn belül hova kerüljön
    - Left/Top, Right/Bottom, Center, Stretch
  - Margók: Left, Right, Top, Bottom
    - Az elem széleinek távolsága a szülő vagy egyéb vezérlők megfelelő széleitől
    - Az ActualHeight/Width-en kívül értelmezettek
  - Padding: csak néhány elemen van (pl border)
    - A benne lévő elemek távolsága a panel szélétől
  - Hasonlóak a horgonyzáshoz (anchor) csak többet tudnak



# Méretezés – Width, Height

---

## ➤ Tulajdonságok:

- Width, Height ( alapértelmezett Auto = Double.NaN)
- MinWidth, MinHeight ( alapértelmezett 0)
- MaxWidth, MaxHeight ( alapértelmezett Double.PositiveInfinity)

## ➤ Számold tulajdonságok:

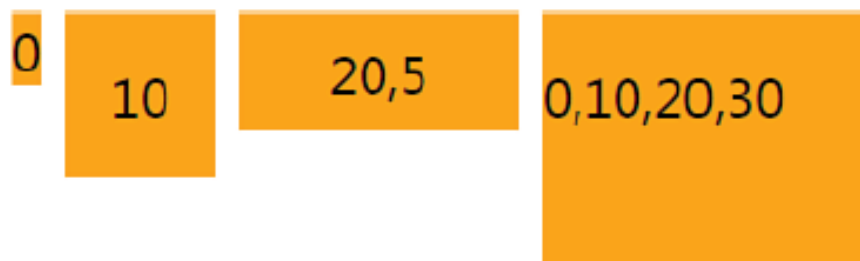
- DesiredSize (UIElement) -> layout készítéskor számolja
- RenderSize (UIElement) -> renderelés után számolja
- ActualHeight, ActualWidth -> renderelés után



# Méretezés – Margin, Padding

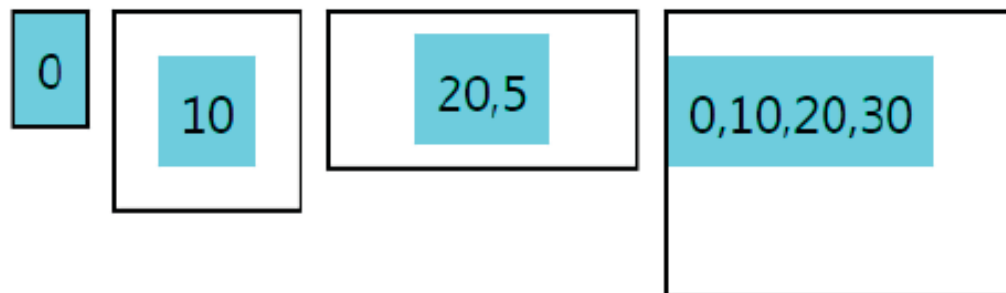
## ➤ Padding:

Four different Paddings:



## ➤ Margin (Thick)

Four different Margins:



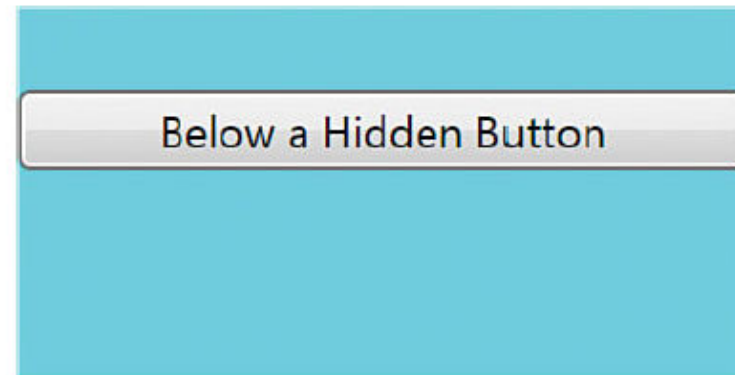
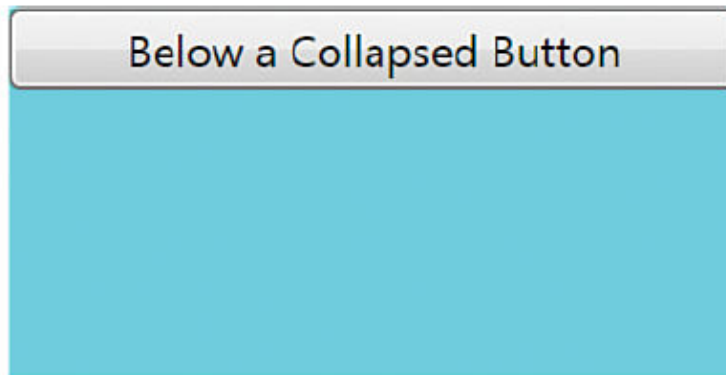


# Méretezés – UIElement . Visibility

---

## ➤ System.Windows.Visibility:

- Visible
- Collapsed
- Hidden

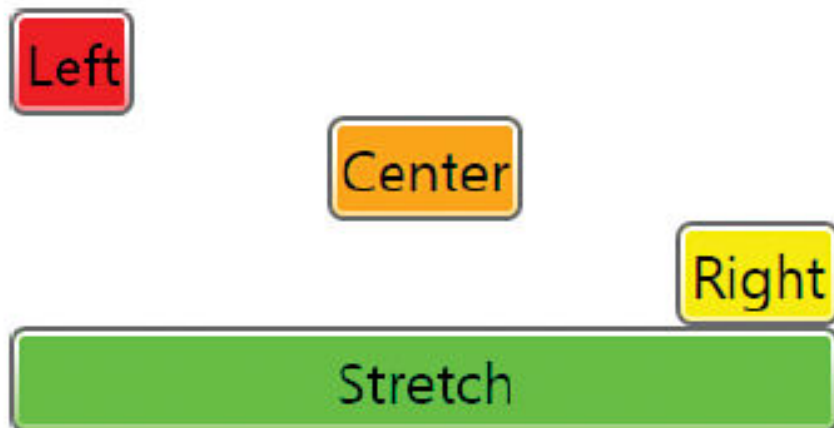




# Pozícionálás

## ➤ Alignment:

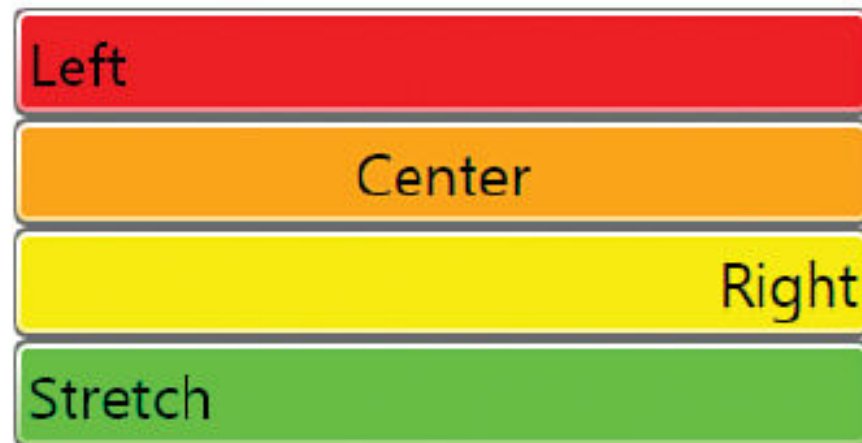
- HorizontalAlignment: Left, Center, Right, Stretch
- VerticalAlignment: Top, Center, Bottom, Stretch





# Pozícionálás

- Tartalom igazítás a Control osztályban
  - HorizontalAlignment / VerticalAlignment propertyk
  - HorizontalContentAlignment: Left, Center, Right, Stretch
  - VerticalContentAlignment: Top, Center, Bottom, Stretch





# Pozícionálás

---

## ➤ FlowDirection:

- RightToLeft
- LeftToRight

LeftToRight

RightToLeft



# Agenda

---

## ➤ XAML

- XAML részletesen

## ➤ Vezérlők hierarchiája

- Content controls
- HeaderedContentControls
- Items controls
- HeaderedItemsControls
- Range controls
- Text and link controls

## ➤ Elhelyezés

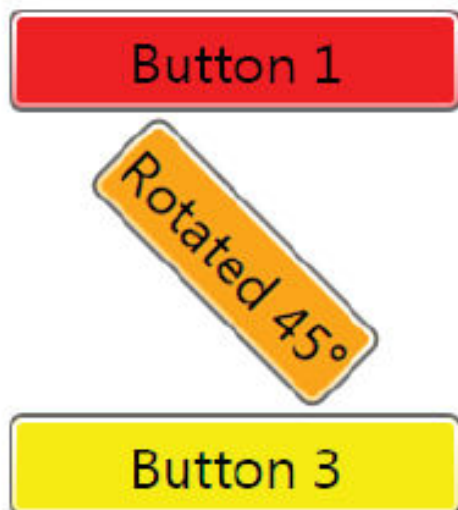
- Méretezés, pozicionálás
- Transzformáció
- Layout engine



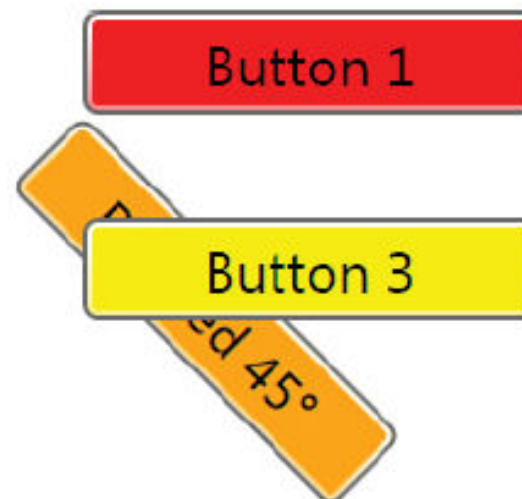
# Transformok

---

## LayoutTransform



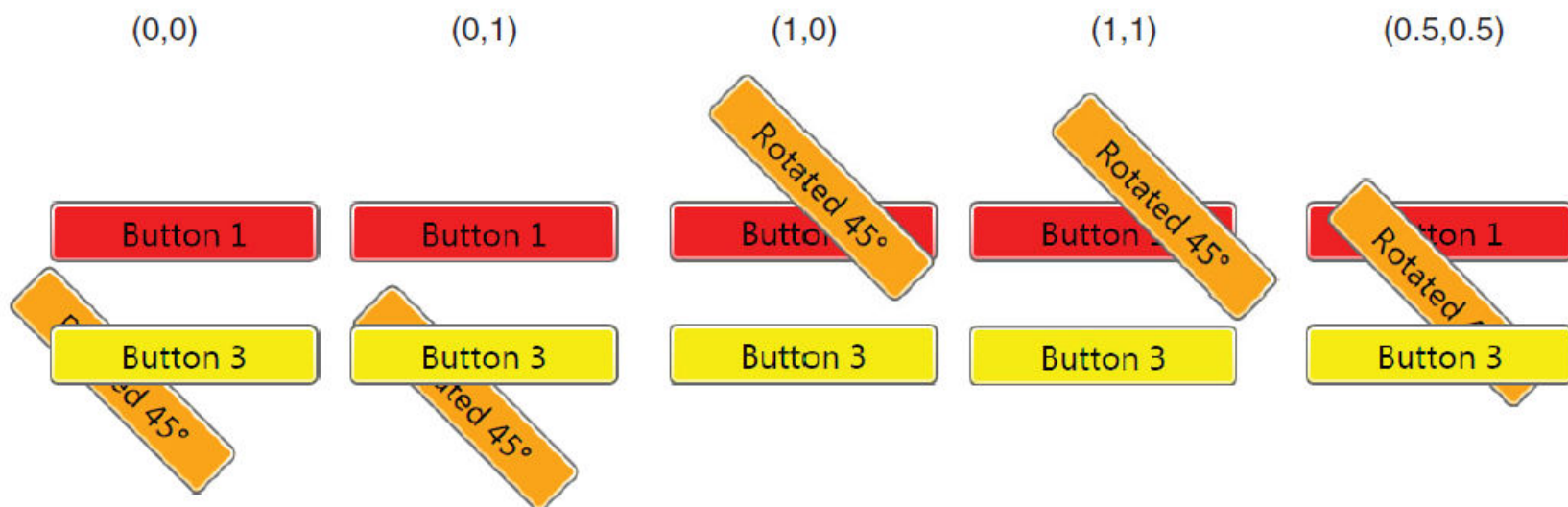
## RenderTransform





# Transformok

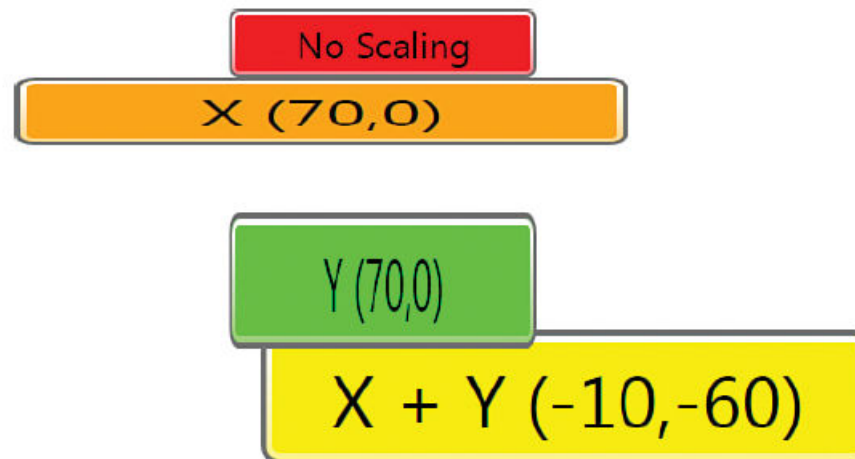
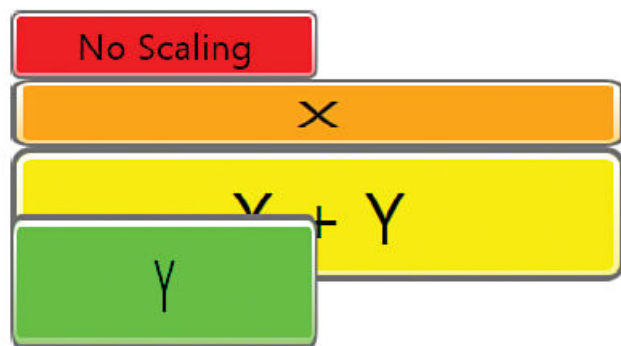
## ➤ RenderTransformOrigin





# ScaleTransform

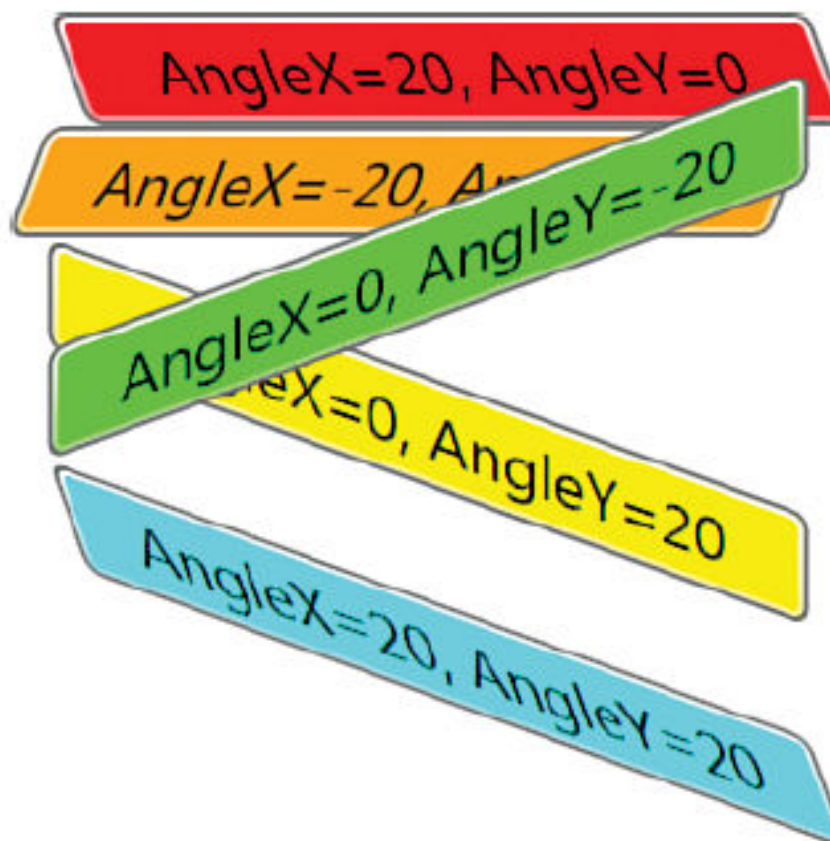
- ScaleX
- ScaleY
- CenterX
- CenterY
- Padding skálázódik, de a Margin nem!





# SkewTransform

- AngleX
- AngleY
- CenterX
- CenterY





# TranslateTransform

---

- X
- Y
- Csak a RenderTransform van rá hatással!!!





# Agenda

---

## ➤ XAML

- XAML részletesen

## ➤ Vezérlők hierarchiája

- Content controls
- HeaderedContentControls
- Items controls
- HeaderedItemsControls
- Range controls
- Text and link controls

## ➤ Elhelyezés

- Méretezés, pozicionálás
- Transzformáció
- Layout engine

...  
Framework  
Element

Panel

Canvas

DockPanel

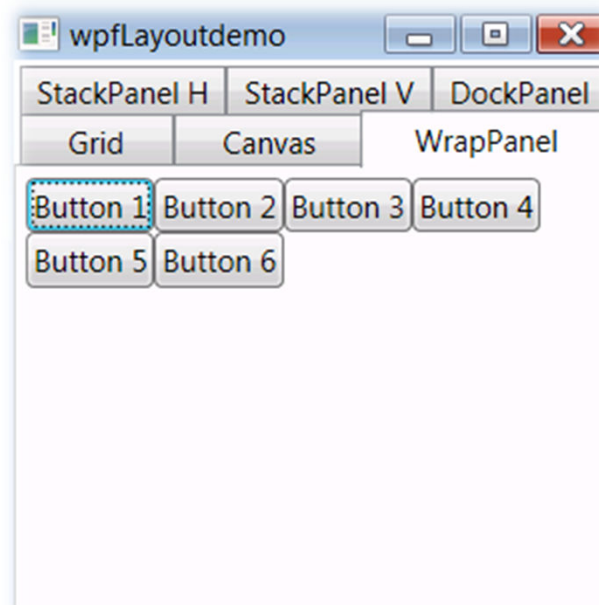
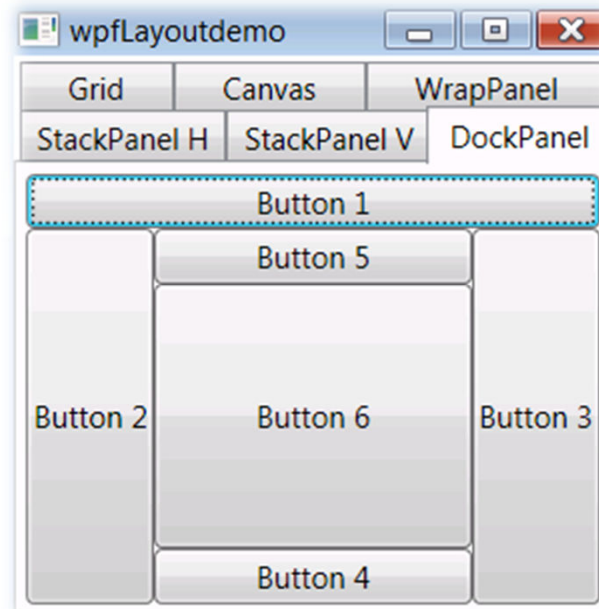
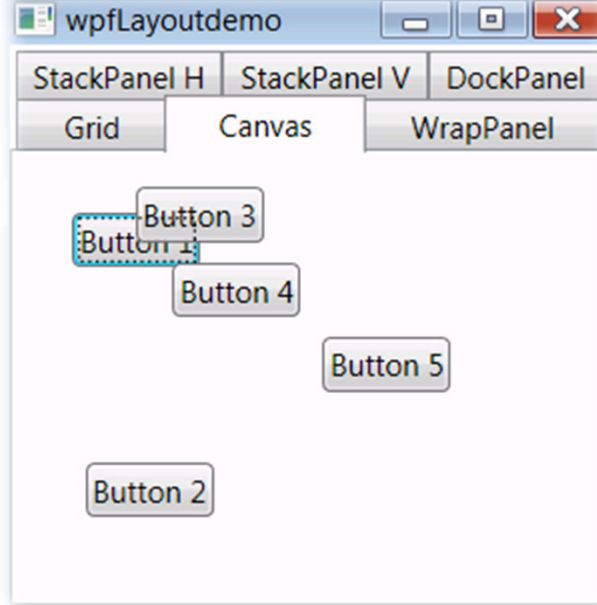
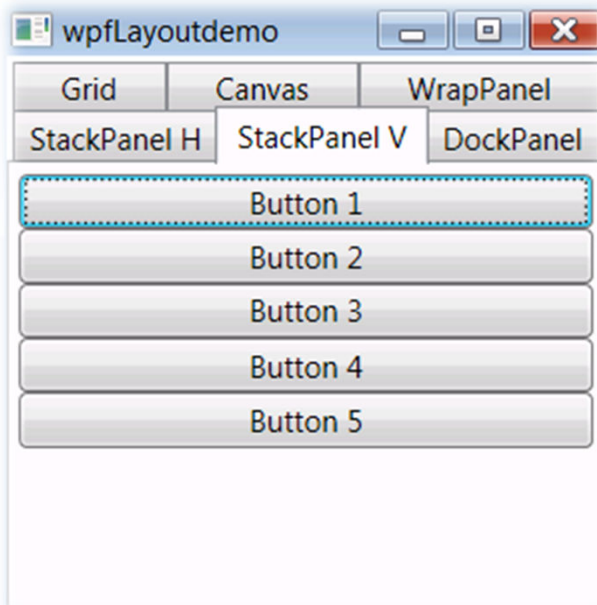
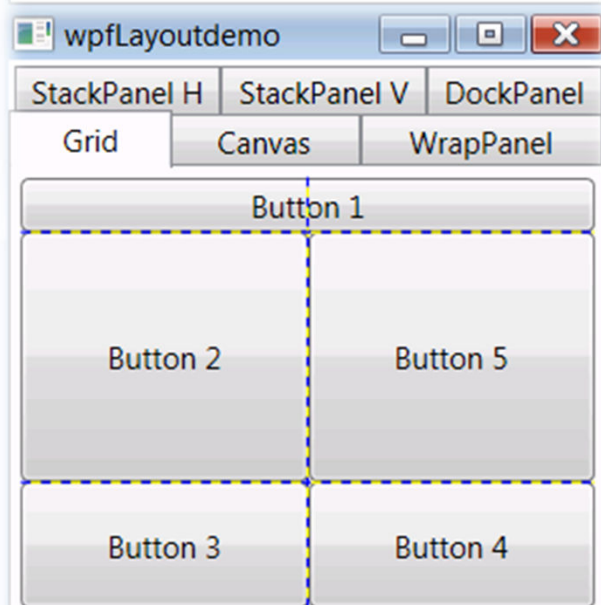
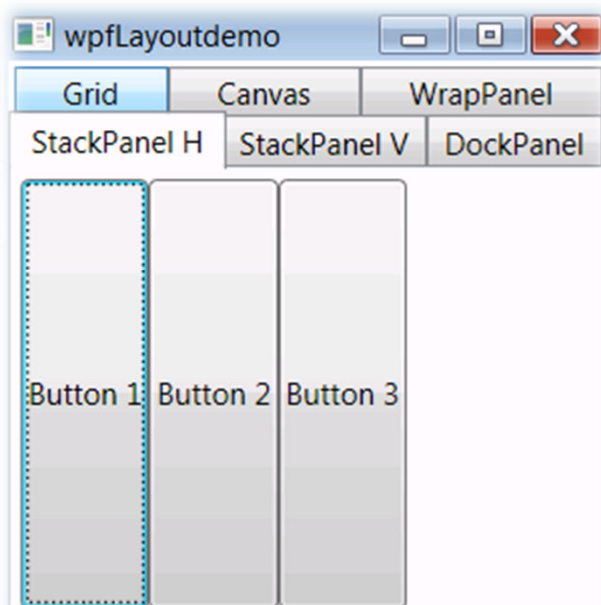
Grid

StackPanel

WrapPanel

Virtualizing  
Panel

VirtualizingStack  
Panel



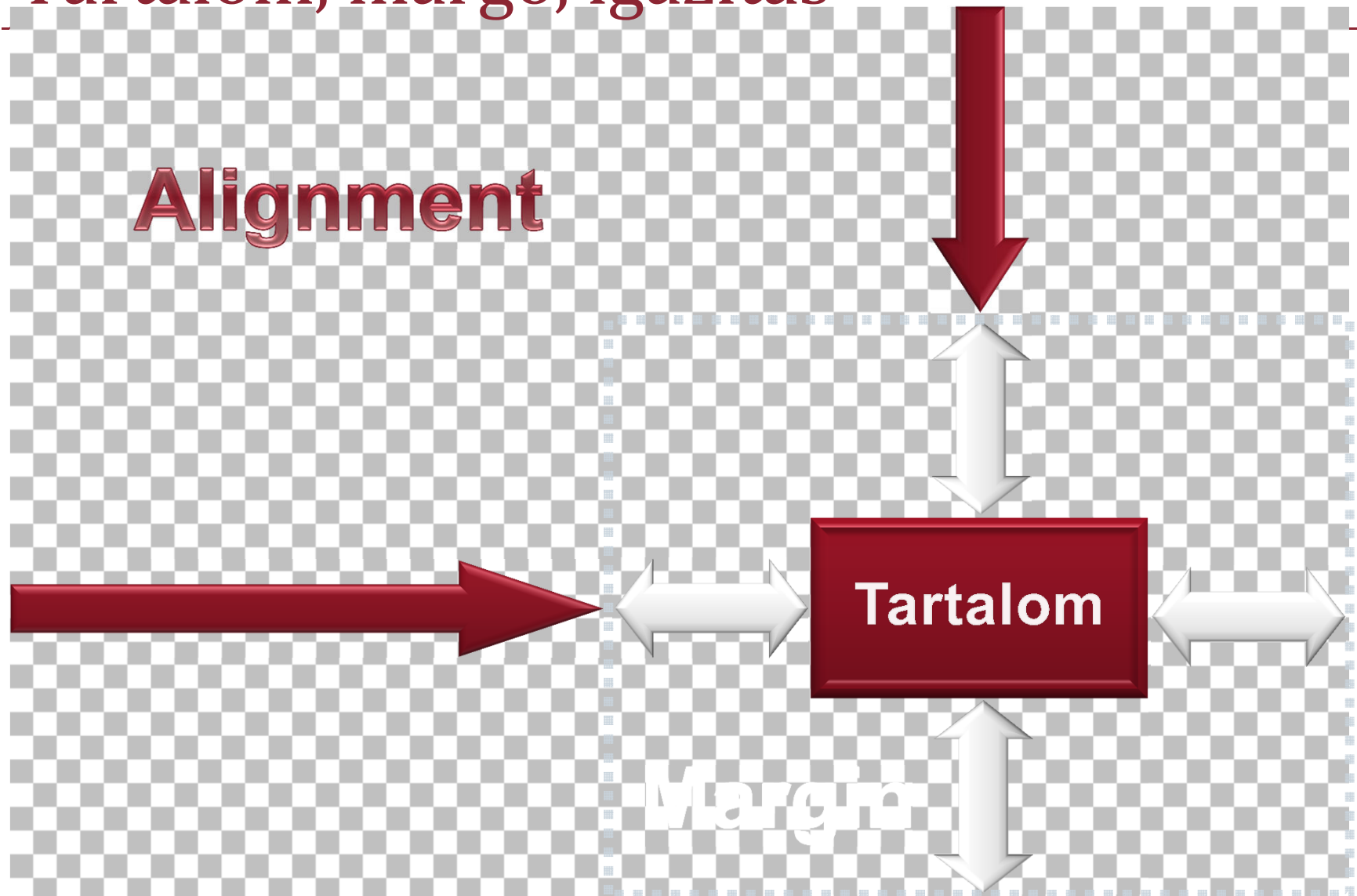


# Layout System – Framework szint

---

- A Panel gyerekeinek elrendezése
- Az elemek pozíciójának, méretének változása új layout számítást indít – lassú lehet
  - Aszinkron módon: a méret változás nem indukál szinkron layout számítást (2 queue: measure, arrange)
- Két fázisú rekurzív layout algoritmus
  - Measure – felmérés
  - Arrange – elrendezés
- Framework: Minden UIElement egy befoglaló téglalapban van

# Tartalom, margó, igazítás





# Panel

---

## ➤ Gyűjtő vezérlő

- Children tulajdonság
- ZIndex csatolt tulajdonság



# Canvas

---

## ➤ Fix X és Y koordináták

- Attached properties: Left, Right, Top, Bottom
- ClipToBounds = false by default
- Átrajzolhat más vezérlőket a canvason kívül



# DockPanel

---

- A vezérlő szélei mentén helyezi el a gyerek elemeket
  - Az elemek sorrendjében
- Dock attached property: Top, Bottom, Left, Right
- LastChildFill property



# StackPanel

---

- Horizontális és Vertikális elrendezés
  - Orientation property



# StackPanel

- Layout Transform esetén, ha 90 fokkal forgatjuk el, akkor kitölti a teret!

No stretching at 80°



Stretching at 90°





# WrapPanel

---

## ➤ Szekvenciális elrendezés (FlowLayout)



# WrapPanel

- Orientation
- ItemHeight
- ItemWidth





# Grid

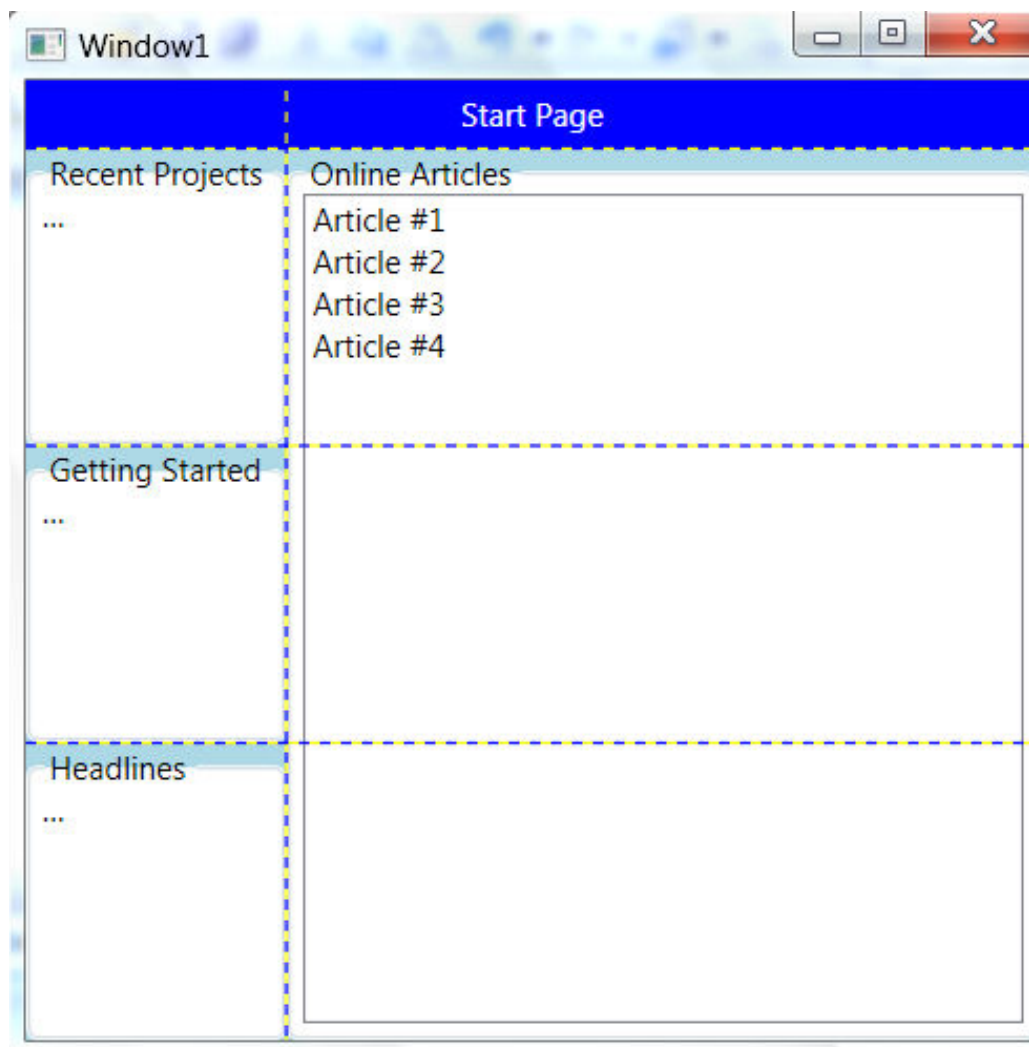
---

- Táblázat
- Cellák összevonása
- Automatikus méretezés
  - A benne lévő vezérlők alapján
- Elfoglalja a lehetséges helyet



# Grid

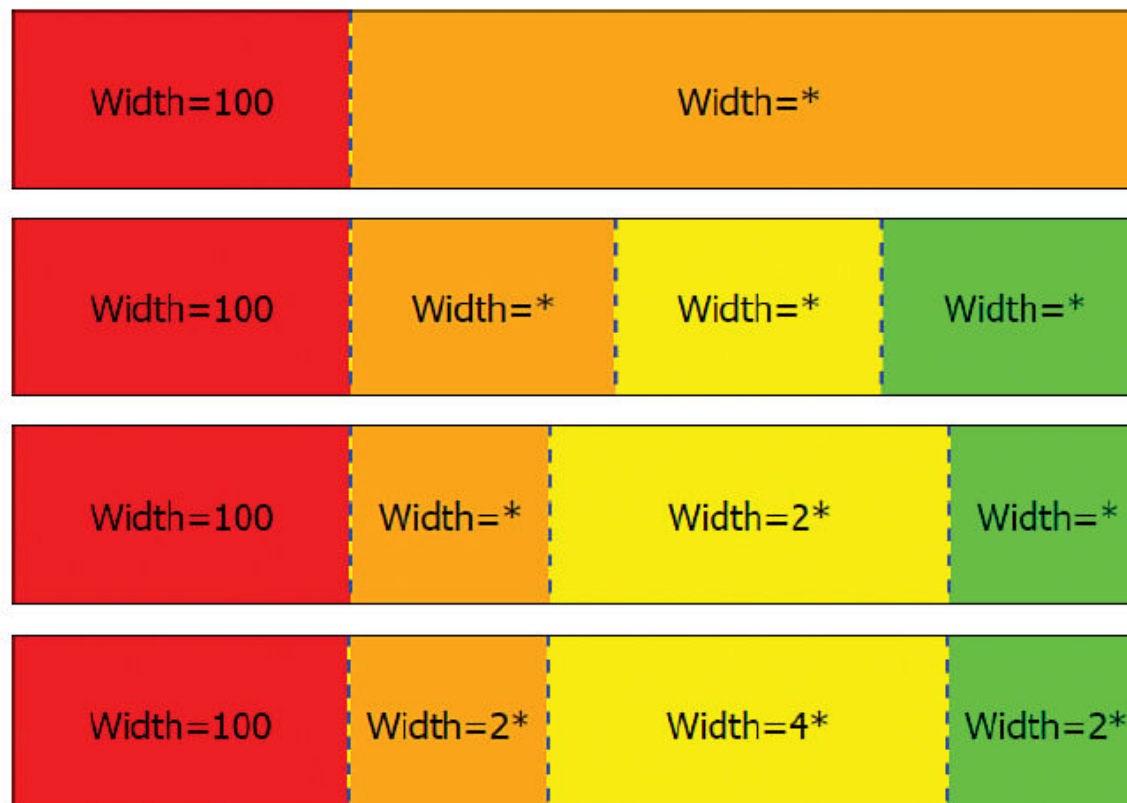
➤ `ShowGridLines="True"`





# Grid

- Cellák méretezése (GridLength):
  - Abszolút méretezés: `Height="100"`
  - Auto méretezés: `Height="Auto"`
  - Star méretezés: `Height="2*"`





# Grid

- Interaktív méretezés (GridSplitter)
- ResizeDirection, ResieBehavior (BasedOnAlignment / PreviousAndCurrent / CurrentAndNext / PreviousAndNext)
- HorizontalAlignment/VerticalAlignment:

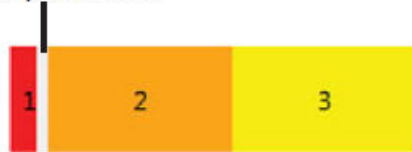
|                   |         | HorizontalAlignment               |                                    |                             |                                                                                                                                                |
|-------------------|---------|-----------------------------------|------------------------------------|-----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------|
|                   |         | Left                              | Right                              | Center                      | Stretch                                                                                                                                        |
| VerticalAlignment | Top     | Current cell and cell to the left | Current cell and cell to the right | Cells to the left and right | Current cell and cell above                                                                                                                    |
|                   | Bottom  | Current cell and cell to the left | Current cell and cell to the right | Cells to the left and right | Current cell and cell above                                                                                                                    |
|                   | Center  | Current cell and cell to the left | Current cell and cell to the right | Cells to the left and right | Cells above and below                                                                                                                          |
|                   | Stretch | Current cell and cell to the left | Current cell and cell to the right | Cells to the left and right | Cells to the left and right if GridSplitter is taller than it is wide, or cells to the top and bottom if GridSplitter is wider than it is tall |



# Grid

- Interaktív méretezés (GridSplitter)
- SharedSizeGroup:

GridSplitter



Default layout

GridSplitter



Layout after dragging  
GridSplitter to the right

GridSplitter



Default layout

GridSplitter



Layout after dragging  
GridSplitter to the right



# WPF – adatkezelés

---



# Agenda

---

1. WPF architektúra, GDI, DirectX, DWM
  - Többszálúság, property system
  - Grafikus primitívek, rajzolás alacsony szinten
  - Média/kép kezelés
2. Vezérlő hierarchia, XAML, layout
3. Adatkötés, sablonok, erőforrások, stílusok, triggerek, eseménykezelés
4. MVVM, 3D, dokumentum kezelés



# Agenda

---

- Erőforrások
- Adatkötés
- Sablonok
  - Adatsablonok, triggerek
  - Vezérlősablonok, stílusok
- Animáció
- Adat validáció
- Eseménykezelés



# Erőforrások

- Nem csak bináris erőforrások: szöveg, kép, stb.
- Hanem XAML-ben deklarált objektumhierarchiák
  - Alkalmazáshoz vagy valamelyik vezérlőhöz tartozik
  - Kulcs alapú azonosítás: 'x:Key' attribútum, elődeklarálás

```
<s:Double x:Key="betűméret"> 14 </...>
```

- Statikus hivatkozás: csak egyszer kérdezi le

```
<TextBlock FontSize="{StaticResource betűméret}">alma</...>
```

- Dinamikus hivatkozás: minden alkalommal lekérdezi
- Dekomponálás, erőforrásfájlok összefűzése

```
<ResourceDictionary.MergedDictionaries> ...
```



# Bináris erőforrás tárolása

---

- „Hagyományos” erőforrások (pl. bitmap)
  - XAML is így tárolódik (BAML)
- Fajtai
  - **Resource:** szerelvénybe ágyazott
    - Build Action = Resource
  - **Content:** szerelvényen kívül álló fájl de hozzá tartozó fájl
    - Build Action = Content
    - Relatív útvonallal lehet hivatkozni rá
  - **Site of Origin:** a szerelvénytől független fájl



# Bináris erőforrások elérése

---

## ➤ Resource:

- Egyszerű URI hivatkozással
- Kódból stream alapján: `getResourceStream`

## ➤ Content:

- Egyszerű URI hivatkozással
- Kódból stream alapján: `getContentStream`

## ➤ Site of Origin:

- Hagyományos URL hivatkozással
- Kódból stream alapján: `getRemoteStream`



# Logikai erőforrások deklarálása

## ➤ *FrameworkElement.Resource* - *ResourceDictionary*

■ Kulcs-objektum párok

## ➤ Alkalmazás vagy vezérlő szinten

## ➤ XAML erőforrás: tetszőleges objektum-fa

## ➤ Példa:

```
<Window xmlns="..." xmlns:x="..." Title="Simple
Window">
  <Window.Resources>
    <SolidColorBrush x:Key="backgroundBrush">
      Yellow</SolidColorBrush>
    <SolidColorBrush x:Key="borderBrush">
      Red</SolidColorBrush>
  </Window.Resources>
```



# Logikai erőforrások használata

## ➤ Erőforrás elérése

- `<StaticResource ResourceKey="backgroundBrush"/>`

- `<Button Background="{StaticResource backgroundBrush}" ... />`

## ➤ A logical treeben felfelé haladva valamennyi erőforrás elemét látjuk (alkalmazás szintűt is)

## ➤ *Dinamikus erőforrások*

- Előre mutató referencia hivatkozások
- Változásértesítés – téma csere, stb.



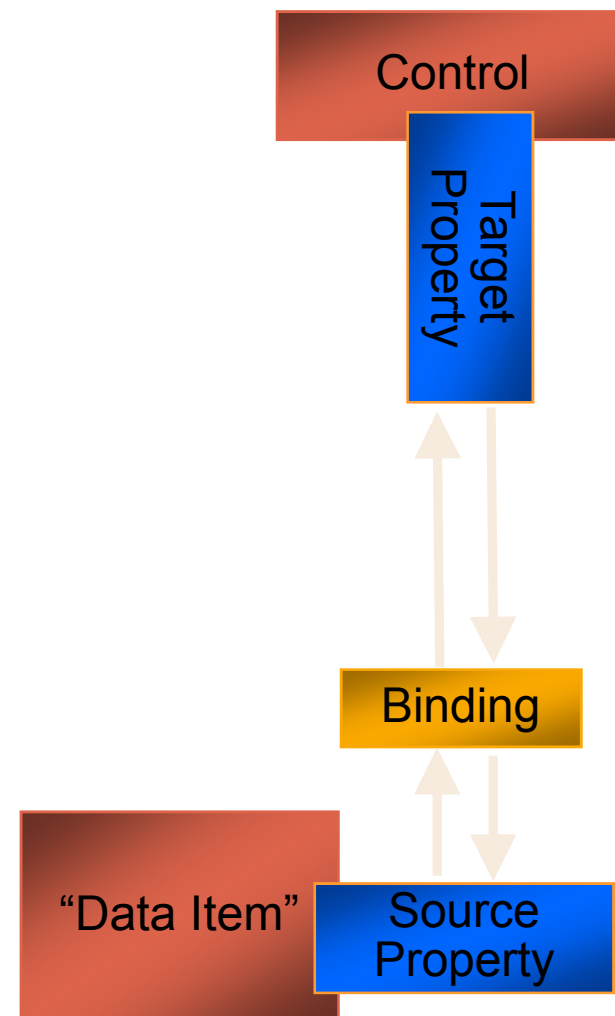
# Agenda

---

- Erőforrások
- Adatkötés
- Sablonok
  - Adatsablonok, triggerek
  - Vezérlősablonok, stílusok
- Animáció
- Adat validáció
- Eseménykezelés

# Adatkötés WPF-ben

- Adatkötés célja: egy vezérlő tulajdonságainak hozzákötése az adatforráshoz
- Target: dependency property
- Source: tetszőleges property





# Binding osztály

---

- Az adatkötés jellemzőit írja le
- *System.Windows.Data.Binding*
  - *Source* : forrás objektum, tetszőleges típus
  - *Path* : forrás property, elmaradhat
- Cél objektum : csak **DependencyObject**
- Cél property : csak **dependency property**
  - *FrameworkElement.SetBinding* vagy
  - *BindingOperations.SetBinding*
  - *Törlés*: *BindingOperations.ClearBinding*



## Binding . Source

---

- Alapértelmezett forrás a logikai fában a cél objektumhoz rendelt DataContext tulajdonság értéke
  - Ha a cél objektumnak nincs DataContextje explicit megadva, akkor a logikai fában a szülőtől örökli
  - A Source explicit beállítható másik objektumra
- Nem állítható együtt az ElementName és RelativeSource tulajdonságokkal



# Binding . ElementName

---

- Az adatkötés forrása lehet egy másik vezérlő
- Az ElementName tulajdonság kell hivatkozzon a forrás vezérlő nevére
- Nem állítható a Source, RelativeSource tulajdonságokkal együtt



# Binding . RelativeSource

---

- Binding . RelativeSource beállítása
  - Saját objektumon lévő propertyhez köt
  - Tipikusan sablonokban, stílusokban használják
- Típusai (RelativeSourceMode):
  - **Self**: az adat forrás megegyezik a cél objektummal
  - **TemplatedParent**: sablonoknál az adatforrás
  - **FindAncestor**: típus vagy mélység alapján keres szülő objektumot a logikai fában
  - **PreviousData**: gyűjtemény esetén az előző element jelenti, összehasonlításra jól használható



## Binding . Path

---

- Az adatforrás pontos elemét határozza meg:
- **PropertyName**: egyszerű tulajdonság név
- **A.B**: elérési útvonal a propertyken keresztül
- **(DockPanel.Dock)**: hivatkozás csatolt tulajdonságra
- **Items[0]**: indexelt elem
- **„/”**: gyűjtemény esetén az aktuális elem kiválasztása
- **„.”**: a teljes forrás objektum kiválasztása
- A fenti lehetőségek kombinálhatók (pl „Items/s[1]”)



## Binding . Mode

---

- Az adatkötés módját alapértelmezetten a célproperty adja meg vagy explicit beállítható
  - TextBox.Text, stb kétirányú, a többi tipikusan egyirányú
- **TwoWay**: kétirányú kötés, a forrás- és cél property változása frissíti a másikat
- **OneWay**: a cél property frissül amikor a forrás megváltozik
- **OneTime**: csak alkalmazás indulásakor vagy a DataContext változásakor kerül frissítésre
- **OneWayToSource**: a forrás property frissül a mikor a cél property megváltozik
- **Default**: alapértelmezett, a cél propertytől függ



## Binding . UpdateSourceTrigger

---

- Kétirányú kötésnél a frissítés módját alapértelmezetten a célproperty adja meg vagy explicit beállítható
  - TextBox.Text, pl LostFocusra frissít, általában PropertyChanged
- **PropertyChanged**: a cél property változásakor frissül a forrás property
- **LostFocus**: focus vesztésnél kerül frissítésre a forrás
- **Explicit**: nincs automatikus frissítés, UpdateSource metódust kell hívni a BindingExpression-ön
  - Minden FrameworkElementnek van ilyen metódusa
- **Default**: alapértelmezett, a cél propertytől függ



# Binding . Converter

---

## ➤ Konvertálás: *Binding.Converter*

- Adatcsere előtt az adat transzformálható
- *IValueConverter*-t implementáló osztály
- **ValueConversionAttribute** a konverter osztály külön megjelölhető
- A konvertálás bemenő argumentumot, paramétert és kultúra információt kap
  - A paraméter a kötésnél adható meg
- Convert: az egyirányú adatkötéshez
- ConvertBack: a kétirányú adatkötéshez



# Binding egyébek

---

- **StringFormat:** Ha a cél property szöveg, akkor a forrás adatot ezzel a stringgel formázza
  - {0}-val kell hivatkozni az adatra
- **TargetNullValue:** Ezt az objektumot használja, amikor a forrás (illetve az elérési útban bármelyik elem) *null*.



## Binding: XML forrás

---

- **XPath**: ez adja meg a kifejezést ami az XML-ből kiválasztja a forrás részt. Az eredmény egy attribútum, csomópont vagy csomópontok halmaza.
- A forrás az **XmlDataProvider**-rel adható meg
  - Távoli fájlra is rá lehet irányítani
  - Az névtér kezelés külön figyelmet igényel



# Binding vs. BindingExpression

---

- **Binding:** sablon, ami az adatkötés tulajdonságait átrolja, megosztható több binding között
  - BindingOperations . Get/SetBinding(DO,DP,binding)
  - FrameworkElement . SetBinding( DP, binding )
- **BindingExpression:** egy kötés példány
  - BindingOperations . GetBindingExpression( DO, DP )
  - FrameworkElement . GetBindingExpression( DP )
  - UpdateTarget / UpdateSource
  - ValidateWithoutUpdate



# Adatforrás változás jelzés

---

## ➤ Property Changes

- `INotifyPropertyChanged`
- `PropertyChanged` event pattern
  - Szöveggel hivatkozott tulajdonság megváltozott

## ➤ Collection Changes

- `INotifyCollectionChanged`
  - Új elem, elem mozgatása/törlése/változása, reset
- `ObservableCollection<T> : INCC`



# Listás adatkötés

---

- **ItemsControl . ItemsSource** propertyra kell kötni
  - A ListBox, stb. az ItemsControlból származik
- Nézetek: **CollectionView** alaposztály
  - Aktuális elem nyilvántartása
  - Adatlista rendezése, szűrése és csoportosítása (!)
  - Minél gazdagabb forrás lista, annál optimálisabb
- XAML-ben: **CollectionViewSource**, becsomagolja a CV-t, paraméterezhetjük, tipikusan erőforrásként
  - Source property: a forrás lista, tipikusan adatkötéssel
- Master-Detail: az összes vezérlőt a CV-ra kötve automatikusan megkapják az aktuális elemet



# ObjectDataProvider

---

- XAML-ben jelenik meg, az adatforrást csomagolja be
  - Eszközökbarát (metaadat)
  - Létrehozás konstruktorral, paraméterezve
  - Létrehozás metódushívással, paraméterezve
  - Aszinkron létrehozás támogatása



# Adatkötés XAML-ben

---

## ➤ BindingMarkupExtension

- `ItemsSource="{Binding Source={StaticResource listingDataView}}"`
- `Text="{Binding Path=StartDate, Converter={StaticResource dateConverter}}",`
  - `<local:DateConverter x:Key="dateConverter"/>`



# Agenda

---

- Erőforrások
- Adatkötés
- Sablonok
  - Adatsablonok, triggerek
  - Vezérlősablonok, stílusok
- Animáció
- Adat validáció
- Eseménykezelés



# FrameworkTemplate ősosztály

---

- Logikai fa sablont tárol
- Lehetővé teszi a sablon alapú példányosítást
- A sablon határozza meg, hogy milyen vezérlőket kell a példányosítás során létrehozni és összerendelni
- Leszármazottjai:
  - DataTemplate
  - ControlTemplate
  - ItemsPanelTemplate



# Adat sablon (DataTemplate)

- Meghatározza az adat vizuális megjelenését
- Sablon választás
  - Vezérlő **ItemTemplate** tulajdonsága
  - **DataTemplateSelector** beállítása
  - A DataTemplate erőforrásban ki van töltve a **DataType**
- A sablonokat a **ContentPresenter** kezeli

```
<DataTemplate DataType="{x:Type local:Task}">
  <StackPanel>
    <TextBlock Text="{Binding Path=TaskName}" />
    <TextBlock Text="{Binding Path=Description}" />
    <TextBlock Text="{Binding Path=Priority}" />
  </StackPanel>
</DataTemplate>
```



# Adat triggererek (DataTrigger)

---

- Tulajdonságok megváltoztatása a forrás adatnak megfelelően
  - Vizuális *megjelenés* deklaratív leírása
  - **Binding**: a vizsgálandó adatot választja ki
  - **Value**: az érték, amire a trigger tüzel
  - **Setters**: beállítandó tulajdonságok és értékük
- **Setter** osztály: a tulajdonságok beállítását végzi
  - **Property**: a beállítandó tulajdonság
  - **Value**: az érték
  - **TargetName**: a vezérlő, amit át kell állítani
- Általában a stílus vagy adat sablon része
- **MultiDataTrigger**: több feltétel is megadható(AND)



# Trigger példa

```
<DataTemplate.Triggers>  
  <DataTrigger Binding="{Binding Path=TaskType}">  
    <DataTrigger.Value>  
      <local:TaskType>Home</local:TaskType>  
    </DataTrigger.Value>  
    <Setter TargetName="border"  
      Property="BorderBrush" Value="Yellow"/>  
  </DataTrigger>  
</DataTemplate.Triggers>
```



# Agenda

---

- Erőforrások
- Adatkötés
- Sablonok
  - Adatsablonok, triggerek
  - Vezérlősablonok, stílusok
- Adat validáció
- Eseménykezelés



# Vezérlő sablonok

---

- Egy WPF vezérlő kódja független a megjelenéstől
- A vezérlő vizuális megjelenését és vizuális működését a ControlTemplate határozza meg
- A template-ben jelzni kell a tartalom helyét
  - ContentControlnál a ContentPresenter osztály
    - Button esetén ide kerül a tartalom (Content)
    - TabItem esetén ez jelenítette meg szöveg helyett a szűrőt
  - ItemsControl-nál az ItemsPresenter vagy a Panel (IsItemsHost=true) ad helyet az elemeknek (például ListBoxItem-ek)



# Vezérlő sablon példa

```
<ItemsControl.Template>  
  <ControlTemplate TargetType="ItemsControl">  
    <Border BorderBrush="Aqua" BorderThickness="1"  
CornerRadius="15">  
      <ItemsPresenter/>  
    </Border>  
  </ControlTemplate>  
</ItemsControl.Template>  
  
<ItemsControl.ItemsPanel>  
  <ItemsPanelTemplate>  
    <WrapPanel />  
  </ItemsPanelTemplate>  
</ItemsControl.ItemsPanel>
```



# Stílusok

- Tetszőleges tulajdonság beállítása FrameworkElement-ből származó osztályokon

```
<Style TargetType="ListBoxItem">  
    <Setter Property="Opacity" Value="0.5" /> ...
```

- Egy vezérlőhöz egyszerre csak egy stílus tartozhat
  - De a stílusok származhatnak egymásból
- Tipikusan erőforrásként definiáljuk a stílusokat
  - TargetType megadása esetén automatikus
  - Ha az erőforrás kulcs is meg van adva, akkor explicit
- Minden beépített vezérlőnek van saját stílusa, az határozza meg a kinézetét is a Template-tel



# Agenda

---

- Erőforrások
- Adatkötés
- Sablonok
  - Adatsablonok, triggerek
  - Vezérlősablonok, stílusok
- Adat validáció
- Eseménykezelés



# Adat validáció

---

- A Binding osztály ValidationRules gyűjteményébe kerülhetnek szabályok
  - **ExceptionValidationRule**: ha a forrás objektum beállítása kivételt dob, akkor az adat hibás
    - Rövidebben: **ValidatesOnExceptions** = true
  - **DataErrorValidationRule**: ha az objektum implementálja az **IDataErrorInfo** interfészt
    - Rövidebben: **ValidatesOnDataErrors** = true
  - Saját validációs szabály is készíthető
- A validációs hibákat a (**Validation.Errors**) csatolt tulajdonsággal lehet lekérdezni



# Az IDataErrorInfo interfész

---

- **String Error:** általános hibalírás az objektumhoz
- **String Item( string propertyName ):** tulajdonság specifikusan lehet megadni a hibaszöveget



# Hiba megjelenítés

- **Validation . ErrorTemplate** csatolt tulajdonsággal megadható a hibás adathoz tartozó vezérlő sablonja
  - **AdornedElementPlaceholder**: vezérlő sablon esetén a meglévő elemet lehet elhelyezni az új logikai fában

```
<ControlTemplate x:Key="validationTemplate">  
  <DockPanel>  
    <TextBlock Foreground="Red" FontSize="20">!</TextBlock>  
    <AdornedElementPlaceholder/>  
  </DockPanel>  
</ControlTemplate>
```



# Agenda

---

- Erőforrások
- Adatkötés
- Sablonok
  - Adatsablonok, triggerek
  - Vezérlősablonok, stílusok
- Adat validáció
- Eseménykezelés



# Eseménykezelés

- Hasonlóan delegate alapú megközelítés
- Rugalmasabb mint a Win32+WinForms
  - Nem téglalap alakú vezérlők hierarchiája
  - A vezérlő is hasonló UIElementekből épül fel
  - Az események a teljes vezérlőfát bejárják.
    - Kétszer
    - Hasonlóan a DHTML eseménykezeléséhez!
- Alagút események (tunneling)
  - Preview, Handled tulajdonság
- Buborék események (bubbling)

