Bevezetés

Tartalom

- **Adminisztratív információk**
- Félév áttekintése
- Esettanulmány
- Háromrétegű architektúra áttekintése

Elérhetőség

- Kovács Ferenc
  - I.B 151
  - kovacsf@aut.bme.hu
- Honlap
  - www.aut.bme.hu/portal/aaf
  - Regisztráció
    - Neptun kód
    - Név

} Ahogy a Neptunban szerepel

Oktatási Forma

- Hétfő
  - Előadás → IE 220
- Csütörtök
  - Időbeosztás függvényében
    - Előadás →IE220
    - Gyakorlat →Tanszéki laborokban

Oktatási anyagok

- Honlap
  - Előadás fóliák
  - Demók
  - Gyakorlatok anyagai
- Jegyzet
  - Készül

Számonkérések

- Gyakorlatok
  - 7 beadandó feladat
  - Legalább 5 feladatot be kell adni
  - A legjobb 4 pontszáma beszámít a vizsgajegybe
  - Legalább 10 pontot el kell érni az aláíráshoz
- ZH
  - 11. oktatási hét, április 20
  - 50%-ot el kell érni az aláíráshoz

Osztályzás

Megszerezhető pontok

Gyakorlat	20 pont
ZH	20 pont
Vizsga	60 pont
Összesen	**100 pont**

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
- Háromrétegű architektúra áttekintése

Cél

- Üzleti alkalmazások fejlesztése
  - Háromrétegű alkalmazások fejlesztése
    - Az egyes rétegek szerepe
    - Sajátosságok az egyes rétegekben
  - Integrációs feladatok
    - Adattárházak

Témakörök – Adatbázis

- Szerver platformok
  - Oracle Server 11g
  - MS SQL Server 2008
- Architektúrák
- Párhuzamos adathozzáférés
- Szerveroldali programozás
- Lekérdezés optimalizálás

Témakörök – Köztes réteg

- Adatelérési osztálykönyvtárak
- O/R leképzés
- LinQ

Témakörök – Megjelenítési réteg

- Adatkötés
- Globalizáció és lokalizáció
- ASP.Net
- AJAX
- WPF
- Mobil kliensek

Témakörök – Integráció

- Adatintegráció
  - Adattárházak
  - OLAP rendszerek

Gyakorlatok

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

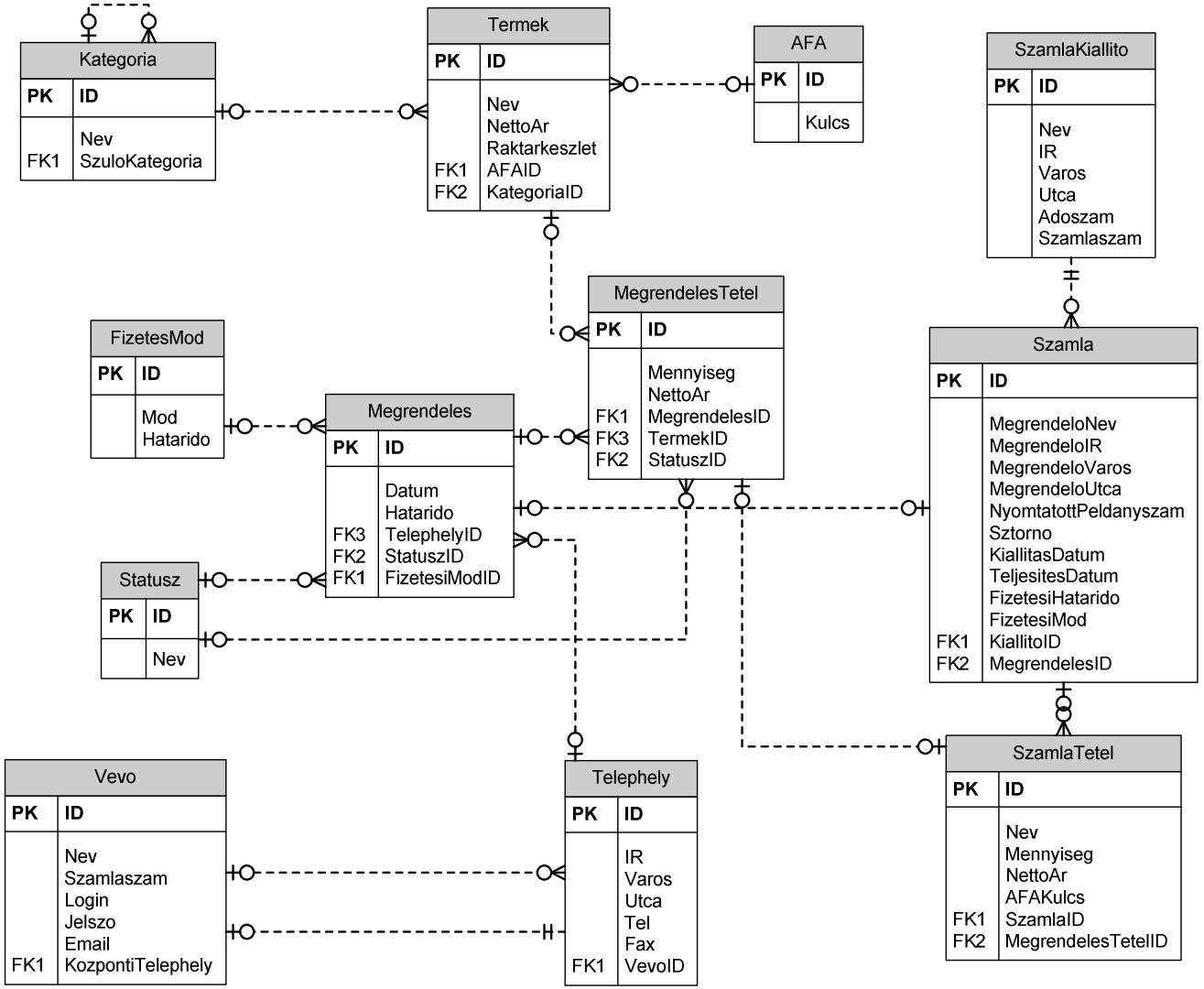
Tartalom

- Adminisztratív információk
- Félév áttekintése
- **Esettanulmány**
- Háromrétegű architektúra áttekintése

Egyszerűsített „CRM” rendszer

- Kereskedelmi cég
- Készletek nyilvántartása
- Vevők
- Megrendelések
- Számlázás

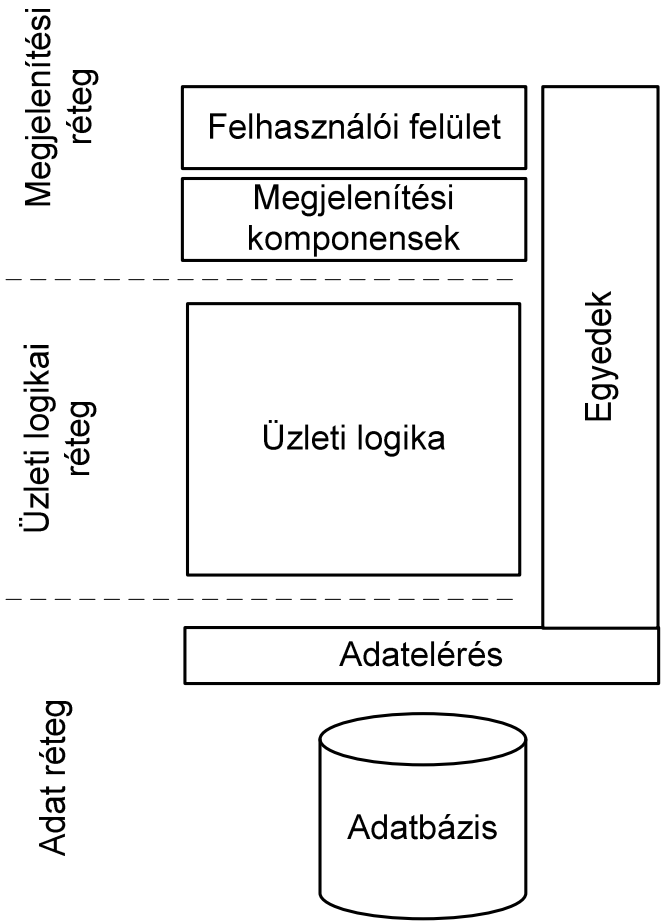
Adatmodell


Kategoria
PK ID
Nev
FK1 SzuloKategoria
Termek
PK ID
Nev
NettoAr
Raktarkeszlet
FK1 AFAID
FK2 KategoriaID
AFA
PK ID
Kulcs
SzamlaKiallito
PK ID
Nev
IR
Varos
Utca
Adoszam
Szamlaszam
MegrendelesTetel
PK ID
Mennyiseg
NettoAr
FK1 MegrendelesID
FK3 TermekID
FK2 StatuszID
FizetesMod
PK ID
Mod
Hatarido
Megrendeles
PK ID
Datum
Hatarido
FK3 TelephelyID
FK2 StatuszID
FK1 FizetesiModID
Szamla
PK ID
MegrendeloNev
MegrendeloIR
MegrendeloVaros
MegrendeloUtca
NyomtatottPeldanyszam
Sztorno
KiallitasDatum
TeljesitesDatum
FizetesiHatarido
FizetesiMod
FK1 KiallitoID
FK2 MegrendelesID
Statusz
PK ID
Nev
Vevo
PK ID
Nev
Szamlaszam
Login
Jelszo
Email
FK1 KozpontiTelephely
Telephely
PK ID
IR
Varos
Utca
Tel
Fax
FK1 VevoID
SzamlaTetel
PK ID
Nev
Mennyiseg
NettoAr
AFAKulcs
FK1 SzamlaID
FK2 MegrendelesTetelID

Tartalom

- Adminisztratív információk
- Félév áttekintése
- Esettanulmány
- **Háromrétegű architektúra áttekintése**

Háromrétegű architektúra

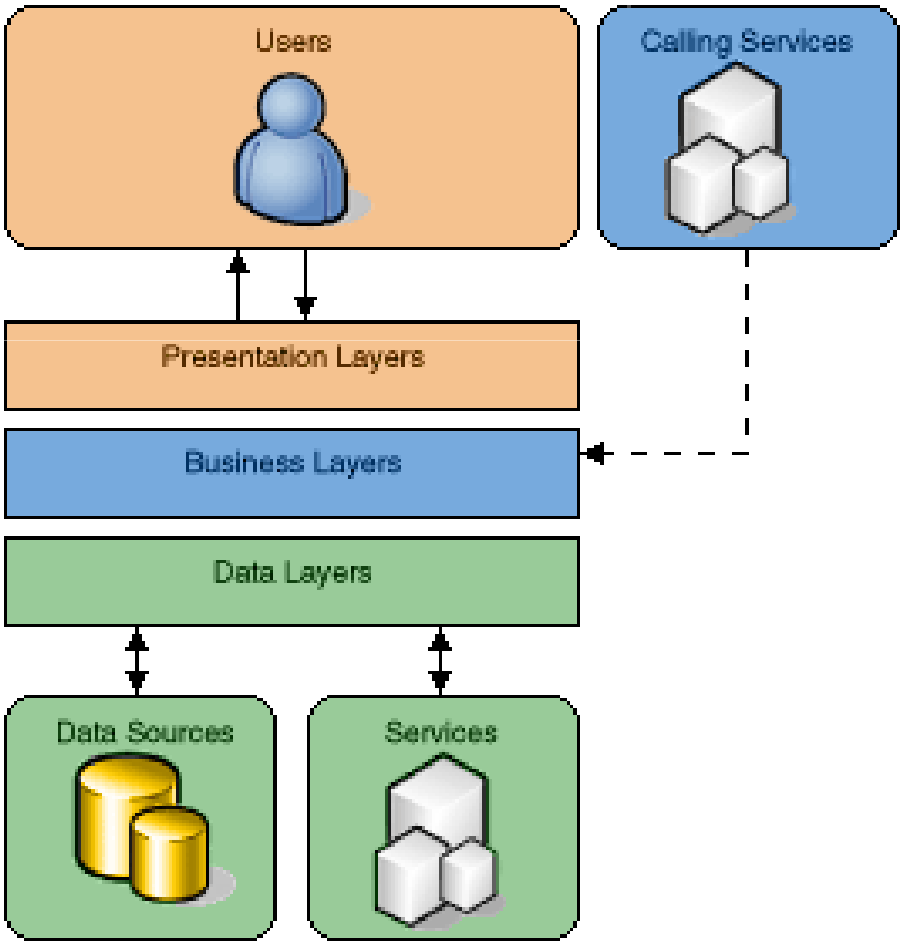


Rétegek szerepe

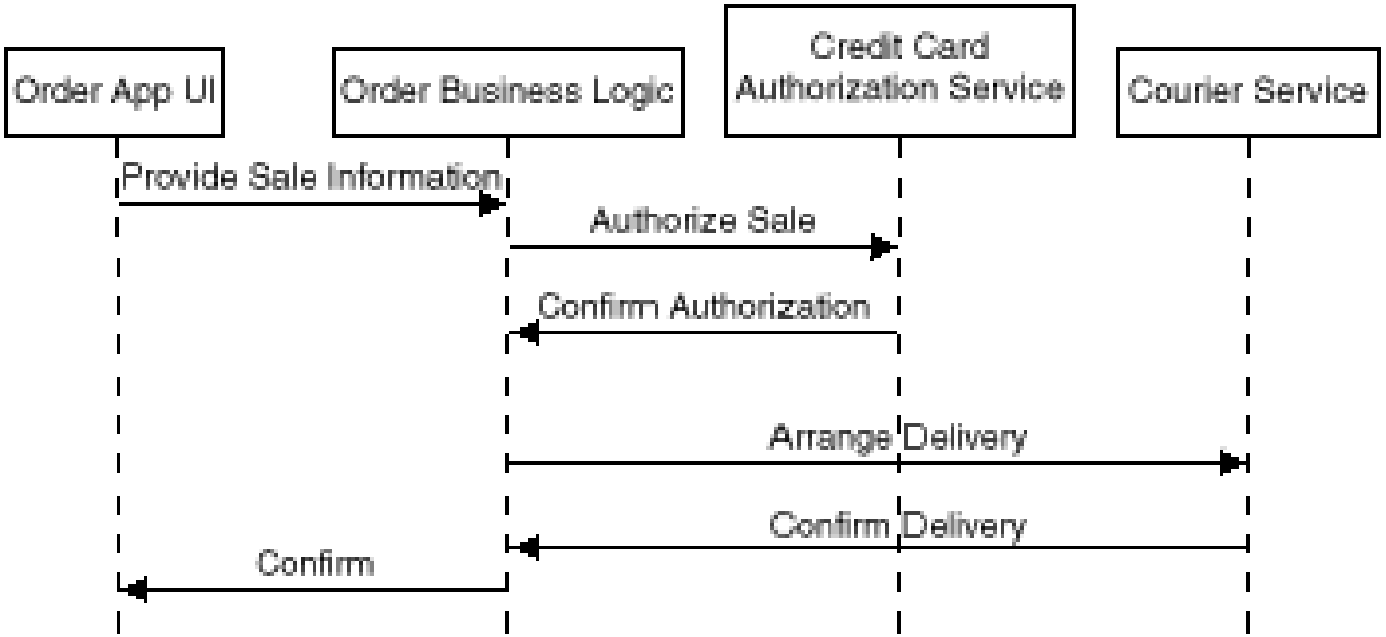
- Adatréteg
  - Perzisztens tárolás
  - Üzleti egyedek leképzése adatbázisba
- Egyedek
  - Rétegektől független egységes modell
- Üzleti logika
  - Tényleges üzleti folyamatok implementációja
- Megjelenítése réteg
  - Felhasználók ezen keresztül láthatják az adatokat
  - Üzleti tranzakciók indítása

Háromrétegű architektúra

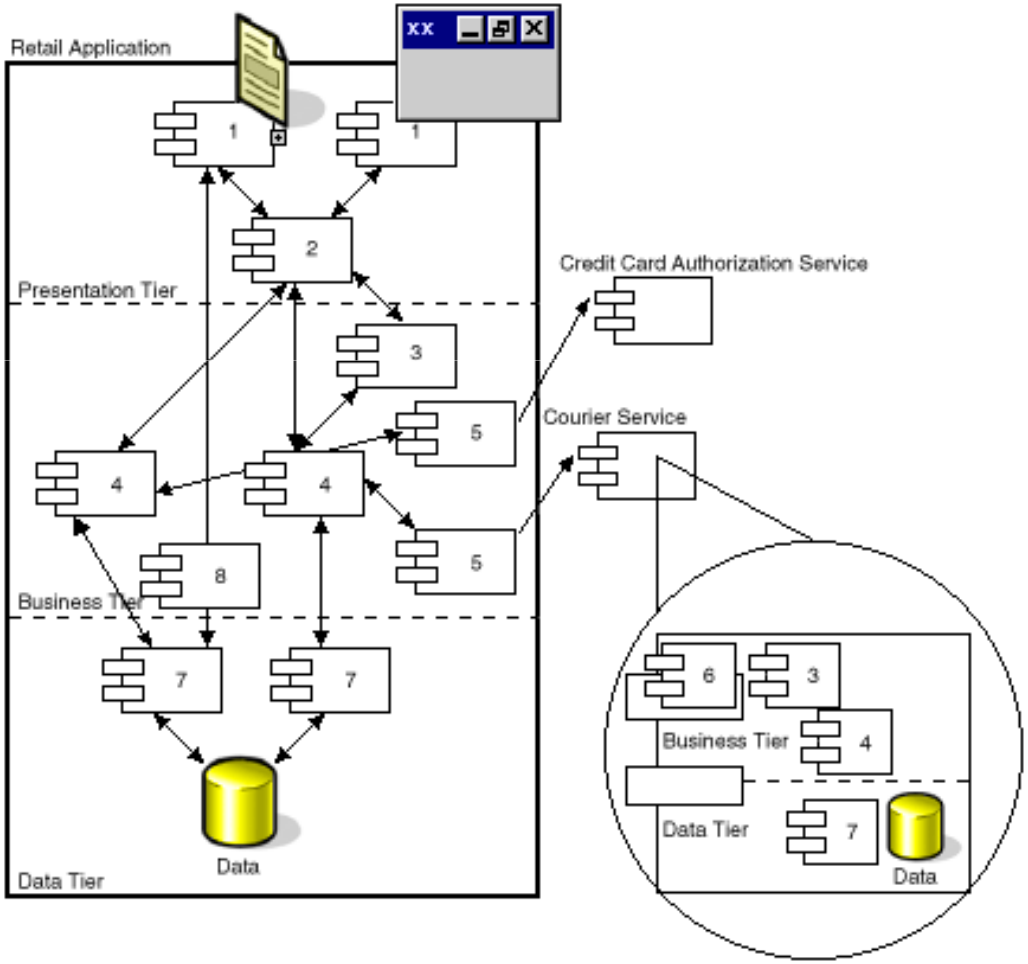
Háromrétegű architektúra



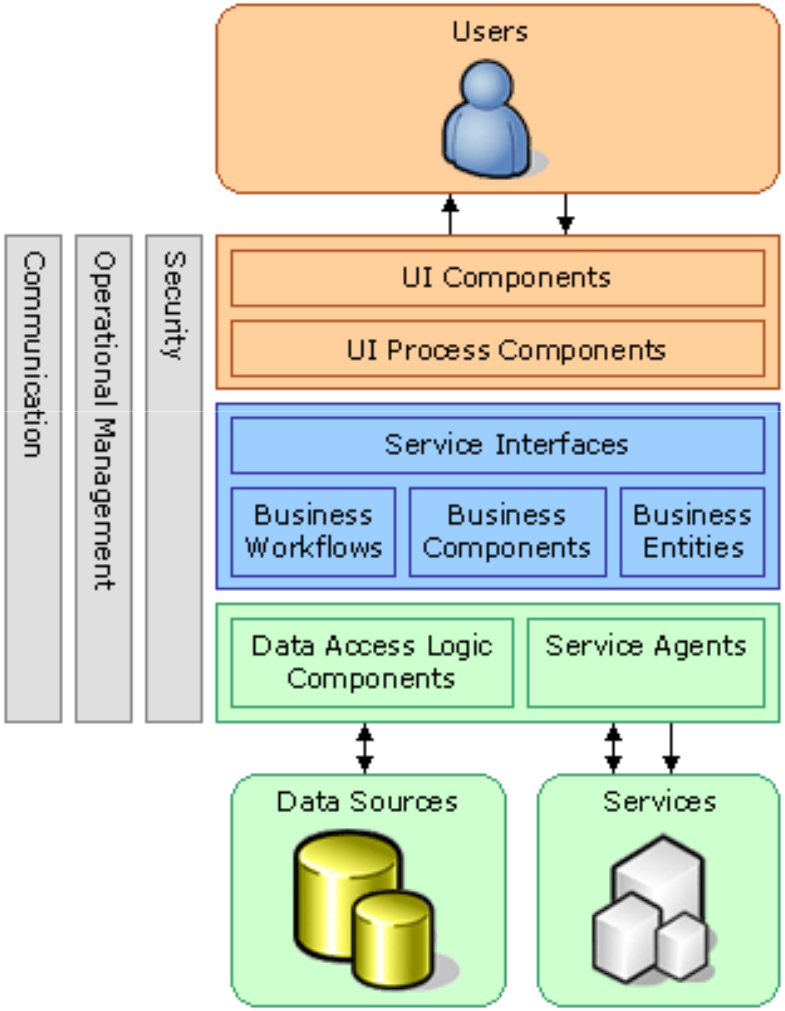
Egyszerű mintapélda



Egyszerű mintapélda



Részletes felépítés



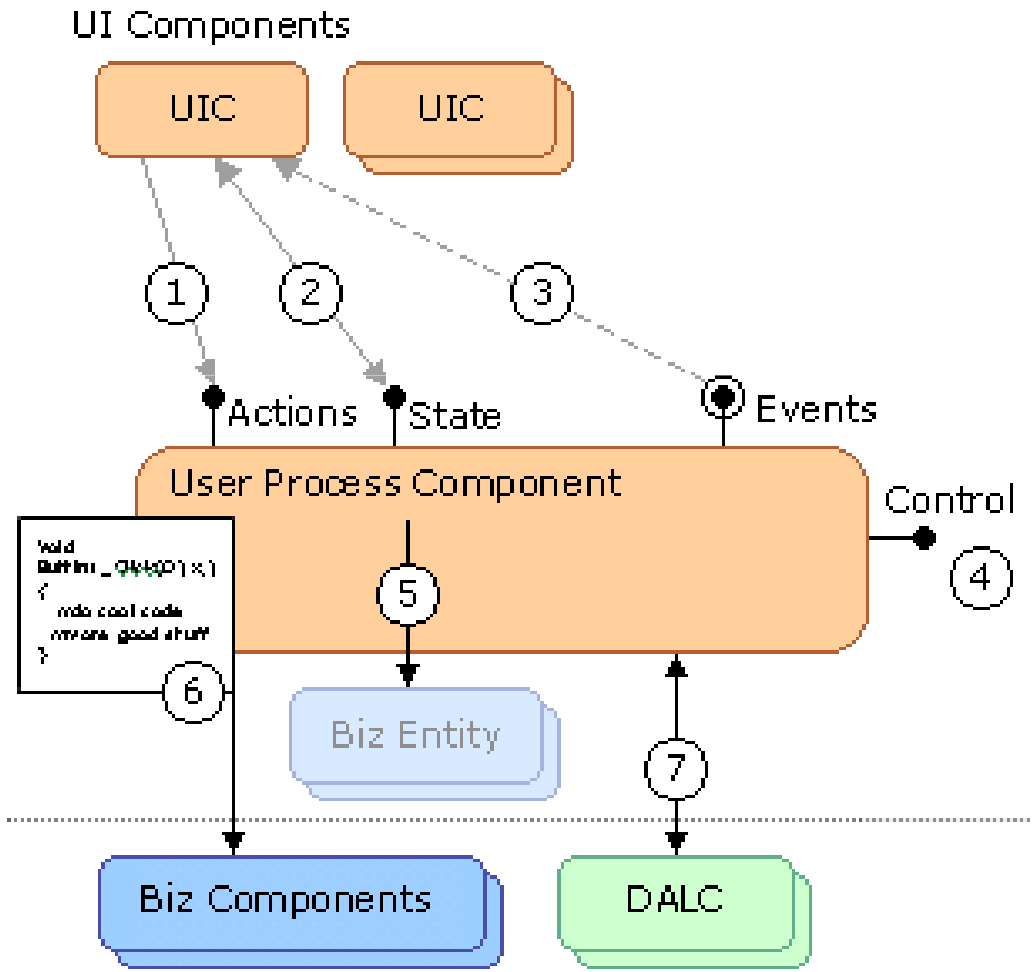
Megjelenítési réteg

- Tipikusan MVC architektúra
- Felület független a tranzakcióktól
  - Indítás, részvétel, szavazás

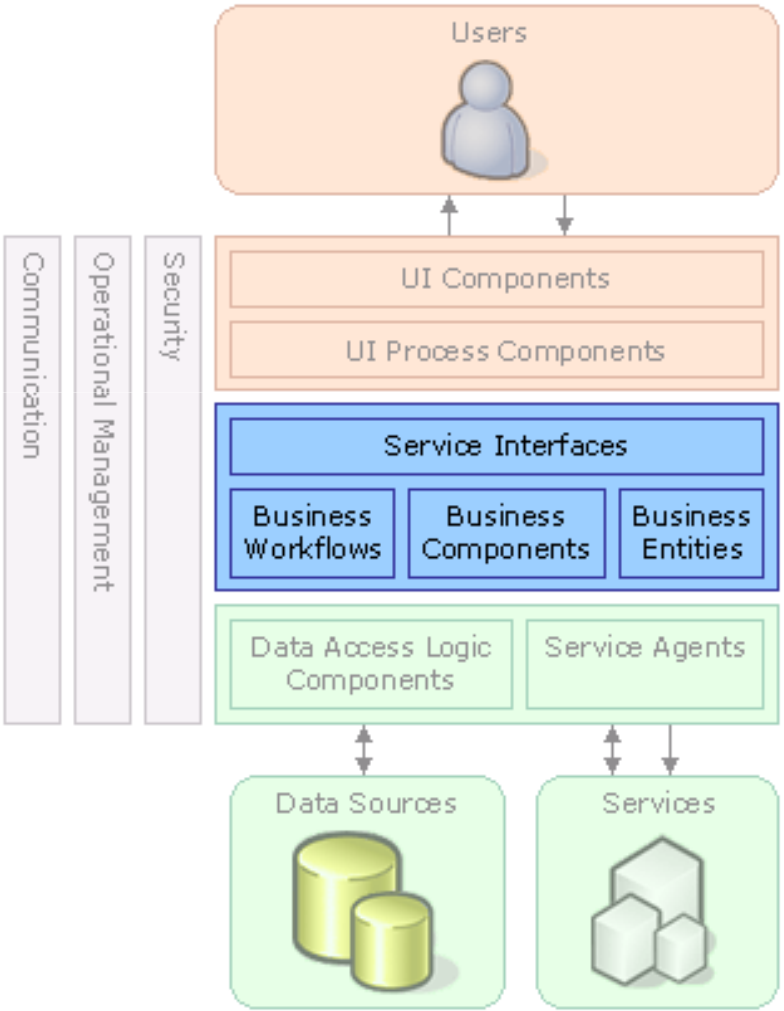
Megjelenítési réteg - feladatok

- 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

Megjelenítési réteg komponensei



Üzleti logikai réteg



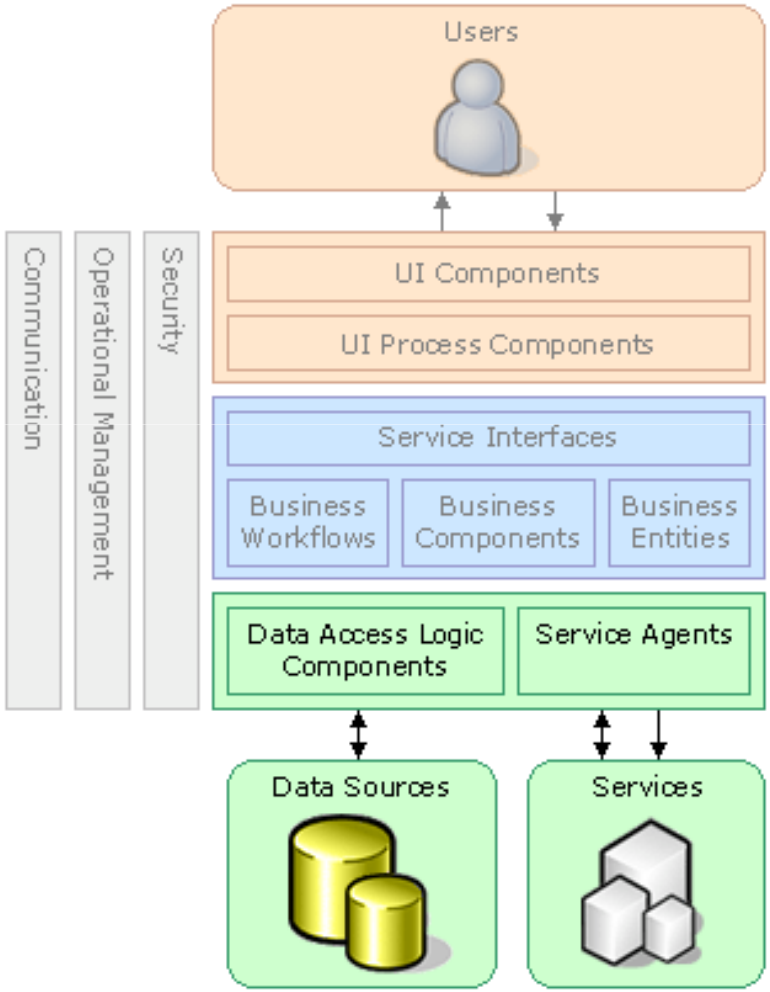
Üzleti logikai réteg komponensei – 1

- 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

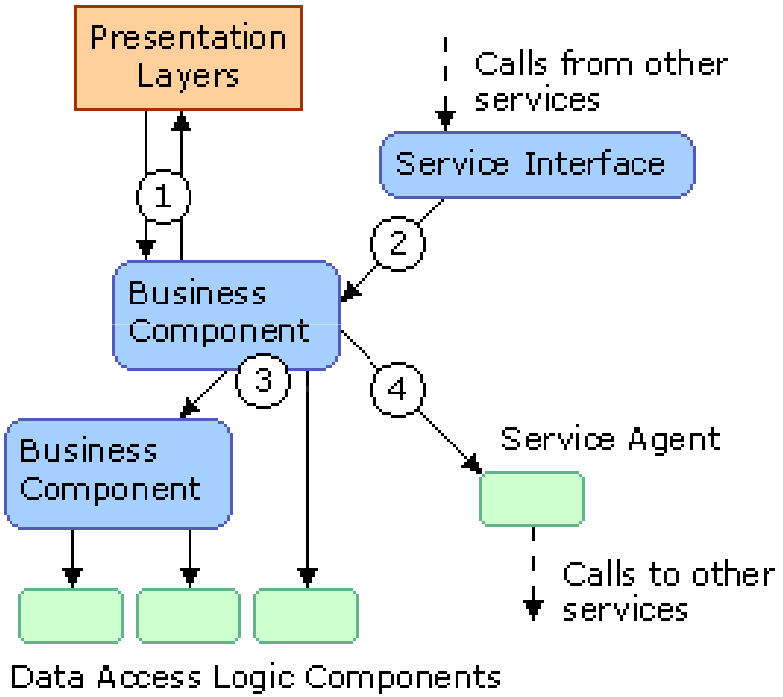
Adatréteg


Users
Communication
Operational Management
Security
UI Components
UI Process Components
Service Interfaces
Business Workflows
Business Components
Business Entities
Data Access Logic Components
Service Agents
Data Sources
Services

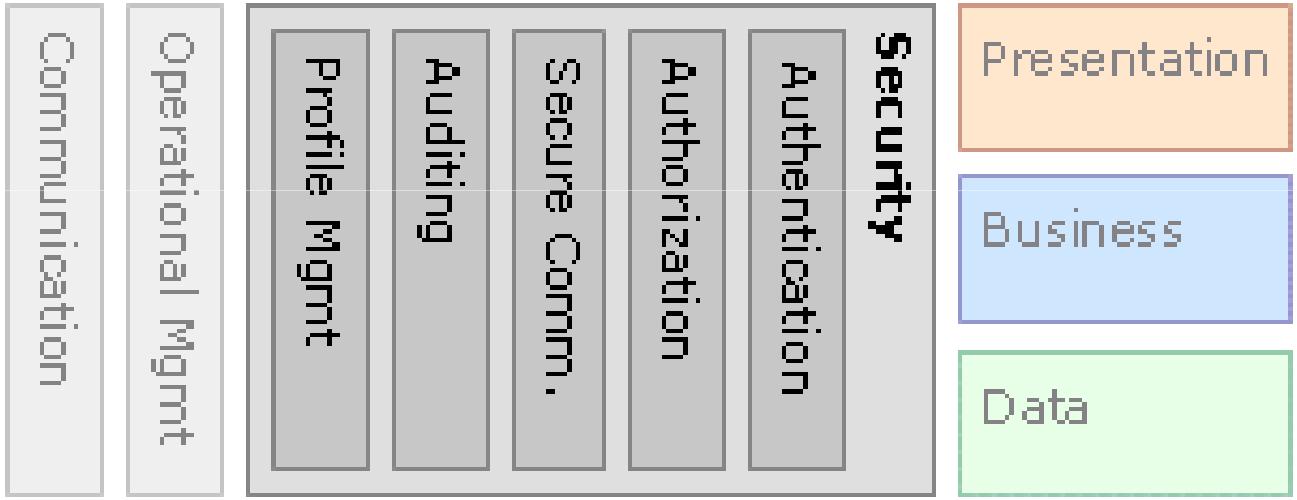
Adatréteg komponensei

- DAL
  - Adatbázis hozzáférés
  - Elemi adatszolgáltatások
- Service Agents
  - Szolgáltatások elérése
  - Külső szolgáltatások „adatbázisként” látszanak
  - Csomagolás

Adatréteg szerepe



Régszerkezet független szolgáltatások – 1



Biztonsági rendszer – 1

- 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 – 2

- 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

Régszerkezet független szolgáltatások – 2

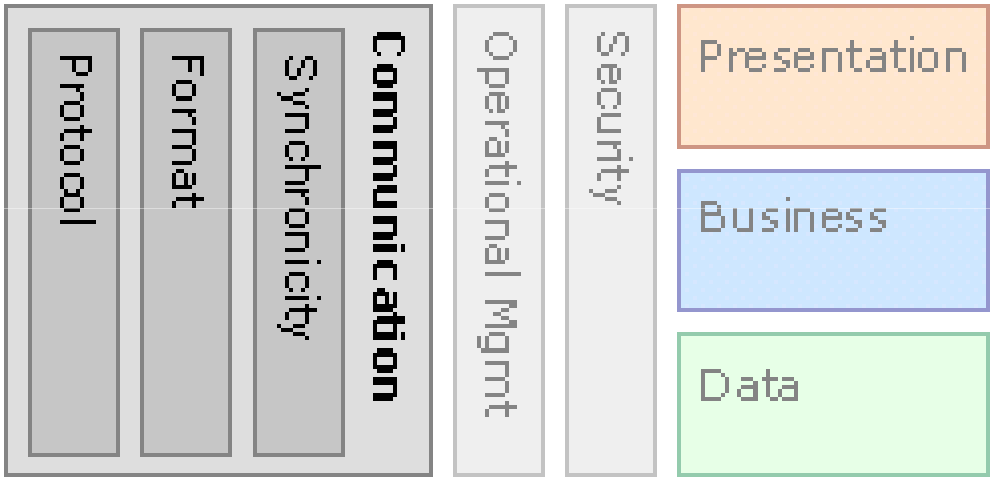
Üzemeltetési szolgáltatások – 1

- Kivételkezelés
  - Egységes szemlélet
  - Naplózás
- Monitorozás
  - Működés ellenőrzése
  - Technikai paraméterek
    - Válaszidő, átbocsájtó képesség …

Üzemeltetési szolgáltatások – 2

- 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, …

Régszerkezet független szolgáltatások – 3



Kommunikáció

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

Adatréteg alapjai

Tartalom

- **Adatbázis definíció**
- Relációs adatmodell
- Adatbázis architektúrák
- Biztonsági rendszer

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: úgy van rendezve, hogy könnyű legyen a tárolás, módosítás, illetve lekérdezés
- Összefüggő adatok: olyan adatok, mely lefedik egy felhasználó csoport érdeklődési területét

Meta Adatok – 1

Kovács Béla	12345678
Nagy Aladár	25898554
Kiss István	985332154

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

Meta Adatok – 2

Adat neve	Adat típusa	Adat mérete	Leírás
Név	Szöveg	30 karakter	Név (Vezeték, -Keresztnév)
Könyv	Szám	9 jegyű szám	Könyv ISBN száma

Tartalom

- Adatbázis definíció
- **Relációs adatmodell**
- Adatbázis architektúrák
- Biztonsági rendszer

A relációs adatmodell alapjai

- Matematikai alapja van
- Három alapvető komponens
  - *Tábla*: az adatok sorokba, és oszlopokba vannak szervezve
  - *Adatmanipuláció*: SQL, vagy relációs algebrai műveletek
  - *Adatintegritás*: bizonyos üzleti szabályok betartását ellenőrizhetjük

A relációs adatstruktúra

Tagok — **Fok**

Kardinalitás

Név	Születési dátum	Tagsági érvényes
Kiss Béla	1975-07-15	2002
Nagy Dezső	1984-12-28	2001

- Reláció: egy kétdimenziós tábla, amely megnevezett oszlopokból, és korlátlan számú sorból áll
- Attribútum: egy nevezett oszlop
- Rekord: egy sor

A relációk tulajdonságai

- Minden relációnak egyedi neve van
- Egy sor és oszlop kereszteződésében egyetlen érték szerepel
- Minden sor egyedi, nincs két egyforma sor
- Minden oszlopnak egyedi neve van a reláción belül
- Az oszlopok sorrendje lényegtelen
- A sorok sorrendje lényegtelen

Adatok integritása

- Tartomány integritás
- Entitás integritás
- Referenciális integritás
- Működési korlátozások

Tartomány integritás

- Oszlopban szereplő adatok ugyanabból a tartományból
- Adattípus
- Hossz
- Tartomány
- Példa: útlevél szám
  - 8 hosszúságú sztring, első két karakter betű, a többi szám

Entitás integritás

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

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

Relációs adatmodell a gyakorlatban

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

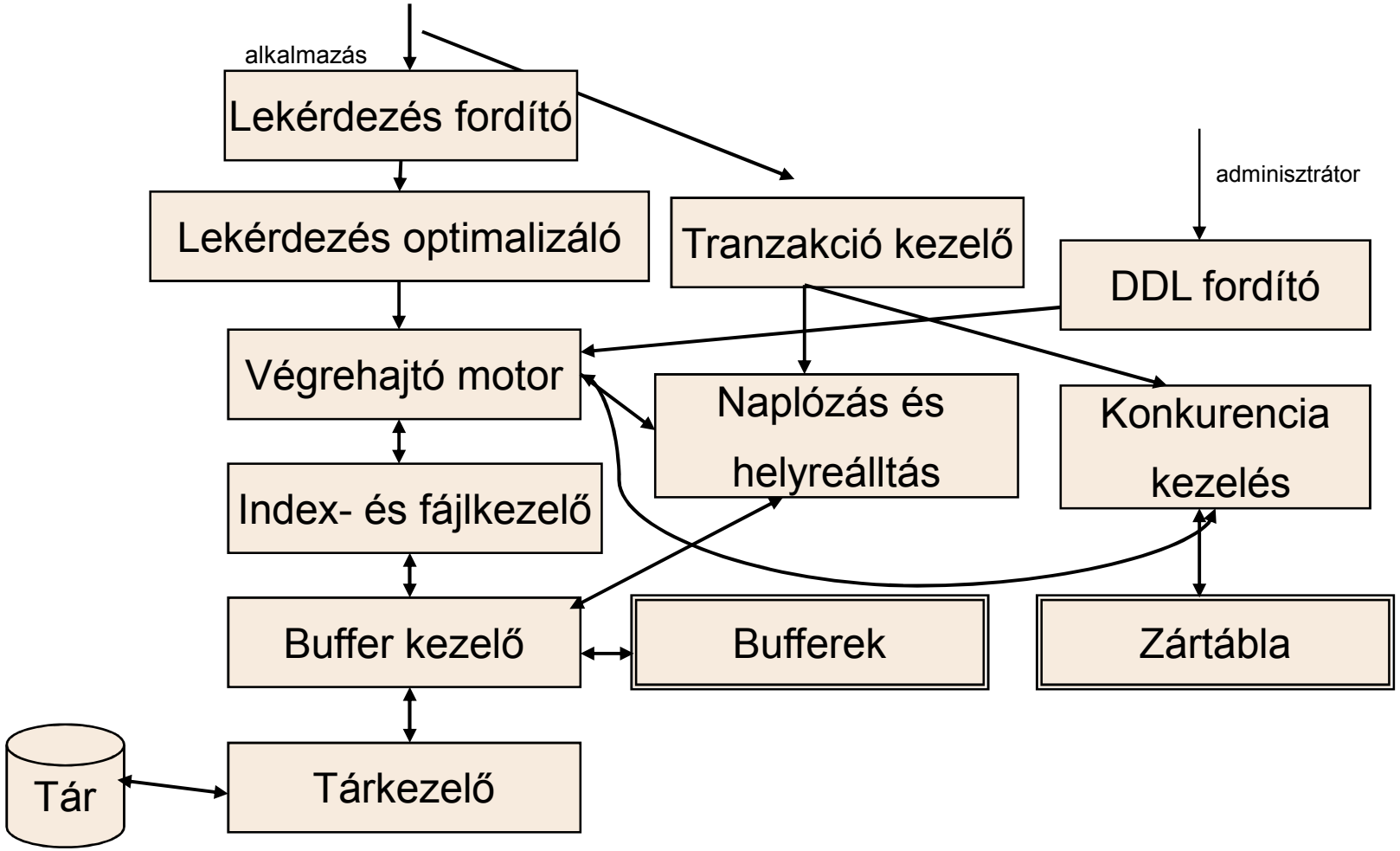
Adatmanipuláció

- Modell szinten: relációs algebra
- Gyakorlatban: SQL nyelv

Tartalom

- Adatbázis definíció
- Relációs adatmodell
- **Adatbázis architektúrák**
- Biztonsági rendszer

Adatbázis- kezelő rendszer elemei



Adatbázisok tárolása

- MS SQL Server
  - Adatfájl (.mdf)
  - Tranz akciós napló (.ldf)
  - Több felhasználói sémát tartalmazhat
- Oracle Server
  - Adatbázis, mint fogalom nem létezik
  - Felhasználói séma
  - Adatfájlok helyett táblahelyek

Felhasználói séma elemei

- Relációs adatmodellből adódó
  - Tábla
  - Megszorítások
- Szerver függő részek
  - Adattípusok (←→SQL szabvány)
  - Nemcsak relációs adatmodell elemek

Adattípusok – 1

MS SQL Server

- Szöveges típusok
  - Char(n)
  - Nchar(n)
  - Varchar(n)
  - Nvarchar(n)

  → Byte, Max 8000
  - Varchar(max)
  - Nvarchar(max)

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

Felhasználói séma elemek

MS SQL Server

- Tábla
- Nézet
- Index
- Programmodul
  - Eljárás
  - Függvény
  - Trigger

Oracle Server

- Tábla
- Nézet
- Index
- Adatbázis link
- Szekvencia
- Programmodul
  - Eljárás
  - Függvény
  - Trigger
  - Csomag
  - Type

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’)

ident_current(‘Statusz’)
```

Oracle Server

- Szekvencia használatával

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

create sequence Statusz_seq
start with 1 increment by 1

Insert into Statusz values
(Statusz_seq.nextval,‘Kész’)

Statusz_seq.currval
```

Tartalom

- Adatbázis definíció
- Relációs adatmodell
- Adatbázis architektúrák
- **Biztonsági rendszer**

Biztonsági rendszer feladatai

- Felhasználók azonosítása
- Adathozzáférés szabályozása
- Nyomkövetés (auditálás)

Felhasználók azonosítása

MS SQL Server

- Operációs rendszer
- Kevert üzemmód
  - OS
  - SQL Server

OS Admin DBA is lesz!

Oracle Server

- Saját
- Külső (címtár szolgáltató)

Adathozzáférés szabályozása

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

Demó

Oracle Enterprise manager

SQL Server Management Studio

Tranzakciók

Tartalom

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

Tranzakció fogalma

- A feldolgozás ***logikai egysége***, olyan feldolgozási műveletek sorozata, mely csak együttesen értelmes
- Alaptulajdonságok:
  - **A**tomicity
  - **C**onsistency
  - **I**solation
  - **D**urabilty

Atomitás biztosítása - 1

- Meg kell tudni jelölni a tranzakciós határokat
  - Begin Tranzaction
  - Commit/ Rollback
- MS SQL Server
  - **Auto commit:** Minden utatsítás önálló tranzakció
  - Explicit tranzakciók: Begin tran, több utasítás esetén
  - Implicit tranzakciók: Tranzakció vége jel után új tranzakció indul
  - DML és DDL utasítások is tranzakció része
    - Néhány kivétel van (Create Database, backup, restore …)

Atomitás biztosítása - 2

- Oracle Server
  - **Implicit tranzakciók:** Tranzakció vége jel után új tranzakció indul
    - Nincs külön tranzakció kezdés utasítás
  - Auto commit üzemmód
  - Csak DML utasításokat tartalmazhat

Konzisztencia

- A rendszert konzisztens állapotból konzisztens állapotba viszi
- Hiba esetén is képes konzisztens állapotba visszaállni
- Hiba
  - Soft crash →memória tartalom vész el
  - Hard crash →háttértároló sérül

Tranzakciós naplózás

Izoláció

- Tranzakciók hatása egymástól függetlenek
- Tranzakciók ütemezése
  - Mintha egymás után hajtódnának végre
  - Zárolás
  - Izolációs szintek

Tartós

- Hatása tartósan megmarad
- Nemcsak memóriában történik meg a módosítás
- Hiba
  - Soft crash
  - Hard crash

Tartalom

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

Cél

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

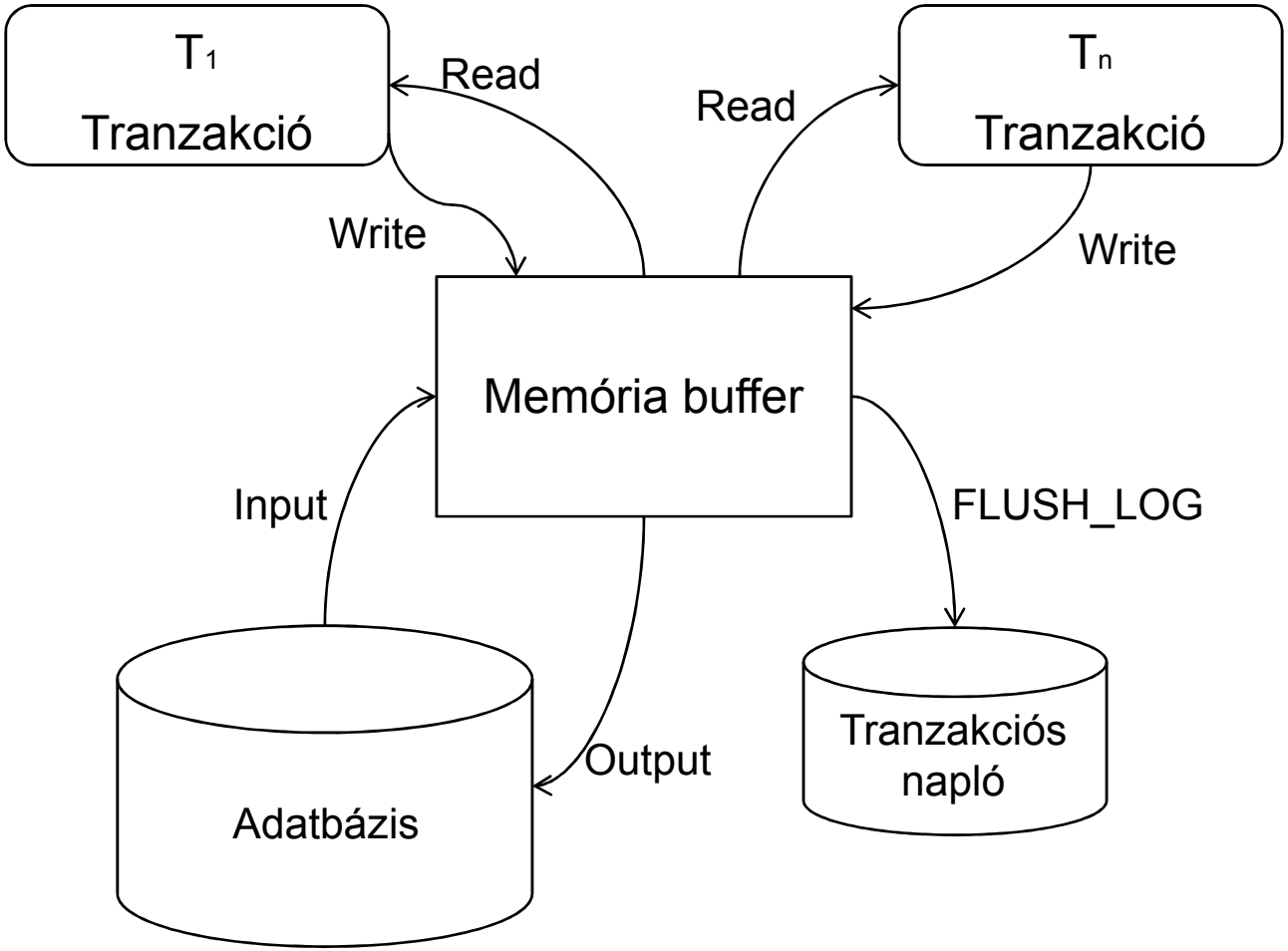
Háttérfolyamatok

- Érintett folyamatok
  - DB reader
  - DB Writer
  - Log Writer
- Minimális szinkronizáció
  - Független működés
  - Előnyösebb ütemezés

I/O modell - 1

- 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

I/O modell - 2


T1
Tranzakció
Read
Write
Tn
Tranzakció
Read
Write
Memória buffer
Input
FLUSH_LOG
Output
Adatbázis
Tranzakciós
napló

Mintapélda

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

Undo típusú tranzakciós naplózás

- 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 tranzakaciós napló nincs kiírva
  - Commit jelet csak az adatbázis írás után lehet kitenni a naplóba

Példa

	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

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

Redo típusú tranzakciós naplózás

- 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

Példa

	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,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	

Visszaállítás

- Tranzakciós napló feldolgozása az elejétől
- Commitált tranzakciók újbóli végrehajtása

Hátrányok

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

Undo/ Redo típusú tranzakciós naplózás

- Napló elemek:
  - Begin T1: T1 tranzakció kezdete
  - Commit T1: T1 tranzakció vége
  - T1,A,u,v: T1 az A értékét u-ról v-re megváltoztatta
- Szabályok
  - Adatelem nem írható át az adatbázisban amíg a rá vonatkozó naplóbejegyzés kiírásra nem került
  - Commit jele helye nem kötött

Példa

	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	

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

Checkpoint

- Tranzakciós napló csökkentése
- A commitált tranzakciók kiírása az adatbázisba
- Új elemek a naplóban
  - Begin Checkpoint <T1, … ,Tn>: Checkpoint kezdődött, T1,…Tn tranzakció aktív
  - End Checkpoint: Checkpoint folymat befejeződött
- T1,… ,Tn tranzakciók legelső bejegyzését megelőző részek eldobhatók

Tartalom

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

Ütemezés szükségessége

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

Elveszet módosítás

- Két párhuzamos tranzakció ugyan azt az elemet módosítja
- Csak a később commitált hatása marad meg

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

Nem megismételhető olvasás

- Lekérdezés eredménye függ attól, hogy mikor adták ki egy tranzakcióban
- Más tranzakció módosította az adatelemet

Fantom rekordok

- Kurzor megközelítés (rekordok halmaza)
  - 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

Helyes ütemezés

- 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 detektálás

Izolációs szintek

- Sorosítható ütemezés ritkán használt
- Túl szigorú
- SQL szabvány szerinti izolációs szintek:
  - Uncomitted 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

Demo

Serilaizable izolációs szint

Izolációs szintek a szerverekben

MS SQL Server

- Uncomitted read
- **Committed read**
- Repeatable read
- Serializable
- Snapshot

Oracle Server

- **Committed read**
- Serializable
- Read only

Implementációs különbségek

- Commited read
  - MS SQL Server: ha egy rekord commitált képe változik, akkor az a rekordot más nem olvashatja
  - Oracle Server: a rekord commitált képe mindenképp olvasható
  - Select utasítás
    - MS SQL Server: megosztott zárat helyez el
    - Oracle Server: nem helyez el zárat

Idegen kulcsok kezelése

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

Demo

SQL Server zárolása

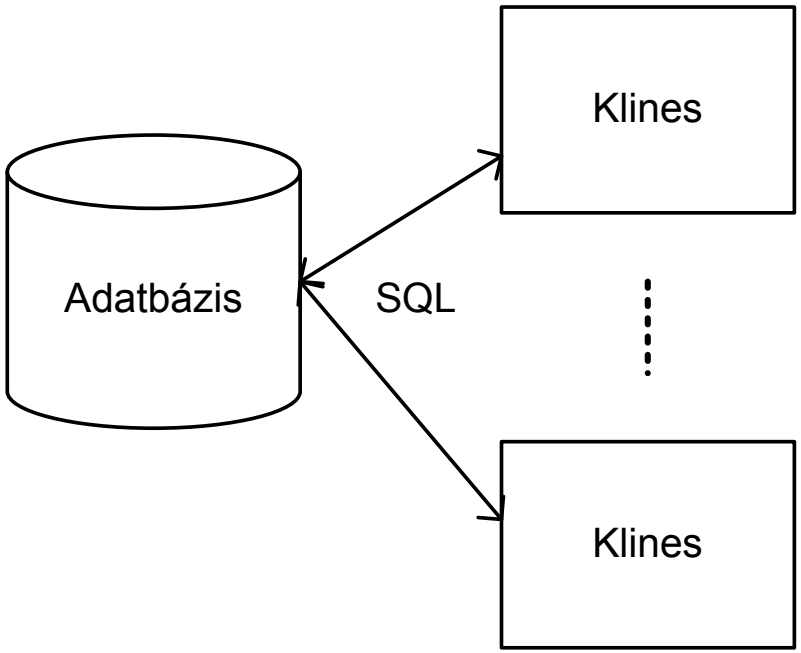
Külső kulcsok használata

Adatbázis programozás

Tartalom

- **Szerveroldali programozás szerepe**
- Oracle Server programozása
  - Tárolt eljárások
  - Triggerek
  - Csomagok
- MS SQL Server programozása
  - Tárolt eljárások
  - Triggerek

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
    - Hívásához 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

Módszerek

- Procedurális feldolgozás
  - Tárolt eljárás
  - Tárolt függvény
- Eseményvezérelt
  - Trigger

Az SQL nyelv és kiterjesztése

- SQL (Structured Query Language)
  - DDL (Data Definition Language)
  - DCL (Data Control Language)

  } Programmodulban általában nem
  - DML (Data Manipulation Language)
- Procedurális kiegészítés
  - Platformfüggő
    - Transact SQL (T-SQL) – MS SQL Server
    - PL/SQL – Oracle
  - Elemek
    - Változók
    - Ciklusok
    - Elágazások
    - Kivételek
    - Kurzorok

Tartalom

- Szerveroldali programozás szerepe
- **Oracle Server programozása**
  - **Tárolt eljárások**
  - Triggerek
  - Csomagok
- MS SQL Server programozása
  - Tárolt eljárások
  - Triggerek

Tárolt eljárások

- Belső
  - PL/SQL nyelv
  - Interpretált
  - Zárt környezet
    - Futtatás
    - Adatbázisba zárt
- Külső
  - Tetszőleges programozási nyelv
  - Adatbázisból kimutató feladatok

Tárolt eljárás létrehozása

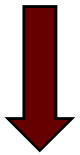
- Create Procedure utasítás
- Szerver letárolja
  - Eljárás szövegét (fordítás sikerétől függetlenül)
  - Végrehajtási tervek
  - Futtatáshoz szükséges Pcode
  - Státusz (Érvényes/ érvénytelen)
  - Hivatkozott objektumok

Fordítási folyamat

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

Tárolt eljárások érvényessége

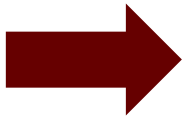
- Érvénytelenné válik
  - Hivatkozott objektum megváltozik
  - Jogosultság megváltozik



- 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)
- To_date(dátum, formátum)
- To_char(kif,formátum)
- Dátum kezelő függények
  - SYSDATE
  - Add_month()
  - Last_day()
  - Month_between
  - …
- String kezelő függvények
  - Ltrim
  - Rtrim
  - Lenght
  - Replace
  - …

Tartalom

- Szerveroldali programozás szerepe
- **Oracle Server programozása**
  - Tárolt eljárások
  - **Triggerek**
  - Csomagok
- MS SQL Server programozása
  - Tárolt eljárások
  - Triggerek

Triggerek használata

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

Események

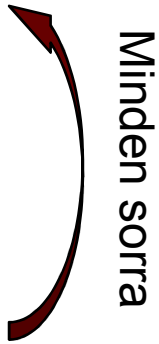
- DML esemény
  - Insert, update, delete
  - Táblához kötődik
- DDL trigger
  - Create, alter, drop, …
  - Sémához kötődik
- Rendszeresemény
  - Logon, logoff, SysError, …
- Instead of triggerek
  - Speciláis DML trigger
  - Nézetek adatmódosítása

DML trigger típusok

- 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)
   1. Sor módosítás előtti triggerek(sor szintű)
   2. Adatmódosítás, zárolás, konzisztencia kritériumok ellenőrzése
   3. Sor módosítás utáni triggerek(sor szintű)



4. 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 hatása

- 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**
- MS SQL Server programozása
  - Tárolt eljárások
  - Triggerek

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

Csomag elemei

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

Számos 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
- **MS SQL Server programozása**
  - **Tárolt eljárások**
  - Triggerek

Tárolt eljárás létrehozása

- Create procedure
  - Létrehozza a tárolt eljárást a sémában
  - Értelmezés, ellenőrzés
  - Eltárolja a kódot
- Első futtatáskor
  - Optimalizálás
  - Fordítás
  - Stored procedure cahce
    - Használaton kívüli tervek elöregednek

Minden SQL Server újraindításkor!

Tárolt eljárás újra fordítása

- Szükséges lehet
  - 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
- Bárhol használhatók ahol az érték
- Rekordhalmazzal is visszatérhetnek
  - Táblaként használhatók lekérdezésekben
  - Komplex logikával lehet rekordokat összeválogatni
- Csak érték kiszámítására használható
  - Adathatása nem lehet

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
- Dátum
  - Getdate()
  - DateAdd(datepart , number, date)
  - Month(date)
  - Year(date)
  - …
- String
  - Replace
  - Trim
  - Ltrim
  - Rtrim
  - Len
  - …

Tartalom

- Szerveroldali programozás szerepe
- Oracle Server programozása
  - Tárolt eljárások
  - Triggerek
  - Csomagok
- **MS SQL Server programozása**
  - Tárolt eljárások
  - **Triggerek**

Trigger típusok

- DML trigger
  - Insert, update, delete
  - Tábla
- DDL trigger
  - SQL Server 2005
  - Felhasználói séma
- Instead of trigger
  - Insert, update, delete
  - Nézet, tábla

DML triggerek

- Utasítás szintű trigger
- Adatmódosítás után hajtódik végre
- Adatmódosítás naplózása
  - Inserted tábla
  - Deleted tábla

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

Instead of triggerek

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

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
      - …

Adatbázis programozás – 2

Tartalom

- **Oracle Szerver programozása**
  - **PL/SQL nyelv elemei**
  - Hibakezelés
  - Kurzorok használata
  - Tárolt eljárások készítése
  - Triggerek készítése
- MS SQL Szerver programozása
  - TSQL nyelv elemei
  - Hibakezelés
  - Kurzorok használata
  - Tárolt eljárások készítése
  - Triggerek készítése

PL/SQL nyelv

- 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

IF utasítás

```
IF feltétel THEN
   <Utasítások1;>
ELSE
   <Utasítások2;>
END IF;
```

```
IF feltétel1 THEN
   <Utasítások1;>
ELSIF feltétel2 THEN
   <Utasítások2;>
ELSE
   <Utasítások3;>
END IF;
```

Ciklusok – 1

```
LOOP
  ...
[EXIT]
...
END LOOP;
```

```
LOOP
  ...
[EXIT WHEN feltétel]
...
END LOOP;
```

Ciklusok – 2

```
WHILE <feltétel> LOOP
…
END LOOP;
```

```
FOR ciklus_számláló IN [REVERSE] alsó_határ..felső_határ LOOP
…
END LOOP;
```

Demó

Egyszerű tárolt eljárás

Tartalom

- **Oracle Szerver programozása**
  - PL/SQL nyelv elemei
  - **Hibakezelés**
  - Kurzorok használata
  - Tárolt eljárások készítése
  - Triggerek készítése
- MS SQL Szerver programozása
  - TSQL nyelv elemei
  - Hibakezelés
  - Kurzorok használata
  - Tárolt eljárások készítése
  - Triggerek készítése

Hibakezelés

- Kivételek
  - Előre definiált hibák
  - Felhasználó által definiált hibák
- Hibaüzenet generálása
  - Feldolgozás megszakítása
  - RAISE_APPLICATION_ERROR tárolt eljárás

Kivételek

- Kivétel eldobása
  - Raise utasítás
    - Felhasználó által definiált hiba
    - EXCEPTION típusú változó
- Kivétel elkapása
  - Exception blokk
  - Minden PL/SQL blokhoz tartozhat Exception blokk

Exception blokk szerkezete

```
EXCEPTION

WHEN kivétel_név1 THEN
        utasítások_sorozata1;

WHEN kivétel_név2 THEN
        utasítások_sorozata2;

...

WHEN OTHERS THEN
        utasítások_sorozata3;

END;
```

Előre definiált kivételek

- 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_ALREDY_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

- **Oracle Szerver programozása**
  - PL/SQL nyelv elemei
  - Hibakezelés
  - **Kurzorok használata**
  - Tárolt eljárások készítése
  - Triggerek készítése
- MS SQL Szerver programozása
  - TSQL nyelv elemei
  - Hibakezelés
  - Kurzorok használata
  - Tárolt eljárások készítése
  - Triggerek készítése

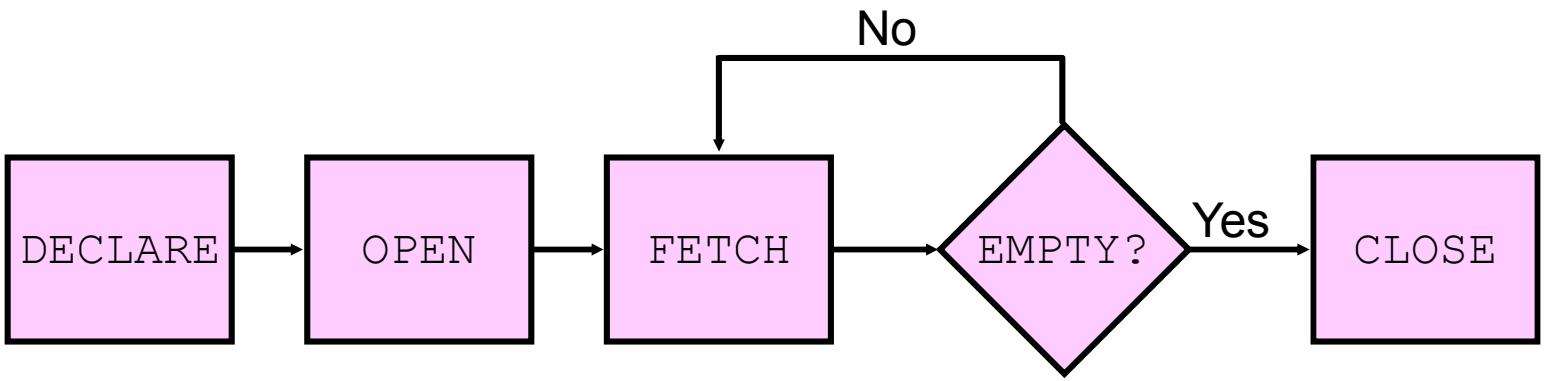
Kurzorok használata

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

Kurzorok deklarálása

- Változók után
- Lehet paraméterük
- CURSOR kurzor_név [(paraméter[, paraméter]...)] IS lekérdezés;

Kurzorok használata – 1



Kurzorok használata – 2

```
DECLARE
  CURSOR emp_cursor IS
   SELECT employee_id, last_name FROM employees
   WHERE  department_id =30;
  empno employees.employee_id%TYPE;
  lname employees.last_name%TYPE;
BEGIN
  OPEN emp_cursor;
  LOOP
    FETCH emp_cursor INTO empno, lname;
    EXIT WHEN emp_cursor%NOTFOUND;
    …
  END LOOP;
  CLOSE emp_cursor;
  …
END;
```

Kurzorok használata – 3

```
DECLARE
  CURSOR emp_cursor IS
   SELECT employee_id, last_name FROM employees
   WHERE  department_id =30;
  emp_record emp_cursor%ROWTYPE;
BEGIN
  OPEN emp_cursor;
  LOOP
    FETCH emp_cursor INTO emp_record;
    EXIT WHEN emp_cursor%NOTFOUND;
     …
  END LOOP;
  CLOSE emp_cursor;
  …
END;
```

Kurzorok használata – 4

```
DECLARE
  CURSOR emp_cursor (did integer) IS
   SELECT employee_id, last_name FROM employees
   WHERE  department_id = did;
BEGIN
  FOR rec in emp_cursor (30) LOOP
     …
  END LOOP;
    …
END;
```

Demó

Kurzorok

Tartalom

- **Oracle Szerver programozása**
  - PL/SQL nyelv elemei
  - Hibakezelés
  - Kurzorok használata
  - **Tárolt eljárások készítése**
  - Triggerek készítése
- MS SQL Szerver programozása
  - TSQL nyelv elemei
  - Hibakezelés
  - Kurzorok használata
  - Tárolt eljárások készítése
  - Triggerek készítése

Tárolt eljárások készítése

- Create or replace procedure utasítással
  - 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

- **Oracle Szerver programozása**
  - PL/SQL nyelv elemei
  - Hibakezelés
  - Kurzorok használata
  - Tárolt eljárások készítése
  - **Triggerek készítése**
- MS SQL Szerver programozása
  - TSQL nyelv elemei
  - 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

Tartalom

- **Oracle Szerver programozása**
  - PL/SQL nyelv elemei
  - Hibakezelés
  - Kurzorok használata
  - Tárolt eljárások készítése
  - Triggerek készítése
- **MS SQL Szerver programozása**
  - **TSQL nyelv elemei**
  - Hibakezelés
  - Kurzorok használata
  - Tárolt eljárások készítése
  - Triggerek készítése

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

Elágazó utasítás

- IF...ELSE

  IF boolean_kifejezés

  Utasítás_blokk

  [ELSE

  Utasítás_blokk]

- Utasítás blokk: BEGIN és END között

Cikluskezelés

- A WHILE ciklus

  WHILE boolean_kifejezés

  Utasítás blokk
- BREAK
- CONTINUE

Demó

Egyszerű tárolt eljárás

Tartalom

- **Oracle Szerver programozása**
  - PL/SQL nyelv elemei
  - Hibakezelés
  - Kurzorok használata
  - Tárolt eljárások készítése
  - Triggerek készítése
- **MS SQL Szerver programozása**
  - TSQL nyelv elemei
  - **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

Demó

Hibagenerálás

Tartalom

- **Oracle Szerver programozása**
  - PL/SQL nyelv elemei
  - Hibakezelés
  - Kurzorok használata
  - Tárolt eljárások készítése
  - Triggerek készítése
- **MS SQL Szerver programozása**
  - TSQL nyelv elemei
  - Hibakezelés
  - **Kurzorok használata**
  - Tárolt eljárások készítése
  - Triggerek készítése

Kurzorok kezelése

- 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. Deallokálás (DEALLOCATE)

Deklarálás

- Kétféle szabvány szerint lehetséges (SQL-92 és T-SQL)
- T-SQL:

```
DECLARE kurzornév CURSOR
[ FORWARD_ONLY | SCROLL ]
[ STATIC | KEYSET | DYNAMIC | FAST_FORWARD ]
[ READ_ONLY | SCROLL_LOCKS | OPTIMISTIC ]
FOR lekérdezés
[ FOR UPDATE [ OF oszlopnév [ ,...n ] ] ]
```

Megnyitás

- OPEN

  OPEN kurzornév

Kezelés

- FETCH

```
FETCH
    [ [ NEXT | PRIOR | FIRST | LAST
        | ABSOLUTE { n | @nvar }
        | RELATIVE { n | @nvar }
      ]
      FROM
    ]
cursor_name [ INTO @variable_name [ ,...n ] ]
```

Bezárás, elengedés

- 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

Kurzorváltozók, információ kurzorokról

DECLARE kurzor_változó_név CURSOR

- @@CURSOR_ROWS
  - A függvény hívásakor hány sort tartalmaz a kurzor
- @@FETCH_STATUS
  - A legutolsó FETCH utasítás státuszát adja vissza
    - 0 – sikeres FETCH
    - -1 – sikertelen FETCH
    - -2 – a lekért sor hiányzik

Demó

Kurzorok

Tartalom

- **Oracle Szerver programozása**
  - PL/SQL nyelv elemei
  - Hibakezelés
  - Kurzorok használata
  - Tárolt eljárások készítése
  - Triggerek készítése
- **MS SQL Szerver programozása**
  - TSQL nyelv elemei
  - Hibakezelés
  - Kurzorok használata
  - **Tárolt eljárások készítése**
  - Triggerek készítése

Tárolt eljárás létrehozása

- CREATE PROCEDURE utasítással egy önálló batchen belül

  CREATE PROC [EDURE] *eljárás_név*
  [{ *@paraméter adat_típus* }
  ] [,...*n*]

  AS *sql_utasítások* [...*n*]

Tárolt eljárások futtatása

- EXECUTE utasítással
- Paraméter átadás:
  - Név szerinti: @parameternev = ertek
  - Pozíció szerinti (az értékek sorrendje megegyezik a paraméterek sorrendjével)

Módosítás, törlés, infók eljárásokról

- Módosítani az ALTER PROCEDURE utasítással lehet, és meg kell adni az egész új eljárást.
- Törölni a DROP PROCEDURE utasítással lehet.
- Információ eljárásokról:
  - Sp_helptext → az eljárás kódját kapjuk vissza, ha nem titkosított
  - Sp_depends → az eljáráson belül hivatkozott objektumok listája
  - Sp_rename → az eljárást így lehet átnevezni

Tartalom

- **Oracle Szerver programozása**
  - PL/SQL nyelv elemei
  - Hibakezelés
  - Kurzorok használata
  - Tárolt eljárások készítése
  - Triggerek készítése
- **MS SQL Szerver programozása**
  - TSQL nyelv elemei
  - Hibakezelés
  - Kurzorok használata
  - Tárolt eljárások készítése
  - **Triggerek készítése**

DML trigger létrehozása

- CREATE TRIGGER utasítással

CREATE TRIGGER *trigger_név*
ON { *tábla* | *nézet* }
 FOR { [DELETE] [,] [INSERT]
 [,] [UPDATE] }
AS
sql_utasítás [...*n*]

Demó

Triggerek

1. ASP.Net alkalmazás készítése 1

1.1. Feladat leírása

ASP.Net alkalmazást készítünk, mely a menüként funkcionáló főoldalon kívül három oldalt fog tartalmazni:

- Áttekintő oldal egy vevők megrendeléseinek a követéséhez
- Fizetési mód rögzítésére szolgáló oldal konkurenciakezelés nélkül
- Fizetési mód rögzítésére szolgáló oldal optimista konkurenciakezeléssel (ezt a következő gyakorlaton fogjuk elvégezni)

1.2. Elvégzendő feladatok

1.2.1. Alapoldalak létrehozása

1. Hozzunk létre egy új C# Web Application projektet
2. Adjunk hozzá a projekthez három aspx oldalt (WebForm), mely a három megvalósítandó oldal keretéül fog szolgálni
3. Az oldalakat nevezzük el (adjuk meg a Title propertyt)
4. A főoldalra (Default.aspx) helyezzünk el három linket a három mely a három oldalra mutat.

1.2.2. Megrendelés követése

1. Helyezzünk el az oldalon egy Drop Down Listet. Ez fog szolgálni a vevők listájának megjelenítésére.
2. Adjuk meg a drop down listhez kapcsolódó adatforrást:
    a. SQL szerver adatforrás legyen, a connection stringet mentessük el a config fájlba.
    b. Válasszuk ki a Vevo táblát
    c. Válasszuk ki az Id és a Név oszlopot
3. Állítsuk be a listához tartozó következő tulajdonságokat:
    a. AutoPostBack=true, azaz ha változik a kiválasztás, akkor újra le kell generálni az oldalt.
    b. DataTextField= Nev, azaz a listában a név oszlop értékei szerepeljenek
    c. DataValueField=ID, azaz kiválasztáskor a kontroll a vevő azonosítóját adja vissza és ne a nevét
4. Helyezzünk el az oldalra egy DataGridView-t, ez fogja tartalmazni a kiválasztott vevő megrendeléseit (Megrendlés dátuma, határidő, státusz, város).
5. Állítsuk be a listához tartozó adatforrást:
    a. Új SQL Serveres adatforrás
    b. Használjuk az előzőekben elmentett kapcsolatot
    c. Mivel a megjelenítendő adatok nem egy táblából érkeznek ezért lekérdezés kell készítenünk
    d. Query Builder segítségével könnyedén el tudjuk készíteni a lekérdezést
        i. adjuk hozzá a lekérdezéshez a megrendelés, telephely és státusz táblákat
        ii. Megrendelés táblából válasszuk ki az ID, dátum, határidő oszlopokat
        iii. Státuszból a név oszlopot
        iv. Telephelyből a Város és VevoID oszlopokat
        v. A VevőID-t ne tegyük hozzá a kimenethez, értékére tegyünk egy szűrő feltételt (=@VevoID)

   e. Paraméter definíciós ablakon adjuk meg, hogy a VevoID paraméter értéke egy controlból fog érkezni, mégpedig a vevőket tartalmazó drop list boxból (ha nem állítunk be mást akkor a SelectedValue property-t köti hozzá, ami nekünk most pont megfelel)
6. A grid view állítsunk be néhány tulajdonságot:
   a. Engedélyezzük a rendezést, a kiválasztást és a lapozást
   b. Az ID oszlopot tüntessük el a listából (állítsuk át a rá vonatkozó Visible tulajdonságot)
   c. DataKeyNames property-t állítsuk be ID-ra, azaz sor kiválasztáskor vissza fogja adni a megrendelés azonosítót
   d. EmptyDataText tualjdonsága írjuk be egy hibaüzenetet, ez akkor jelenik, meg, ha nincs megjelenítendő elem
   e. PageSize-ot állítsuk át 5-re, ezzal tudjuk megadni a lapozás méretét
   f. Ha gondoljuk valmilyen stylust is kiválaszthatunk, hogy szebb legyen a megjelenítés
7. Most már a megrendeléshez tartozó megrendelés tételek megjelenítéséhez tartozó grid view elkészítése van hátra. Meg kell jeleníteni a termék nevét a rendelt mennyiséget, a nettó árat és a státuszt (itt is egyedi lekérdezést kell írni és a megrendelés azonosító a fenti grid view-ból jön)

1.2.3. Fizetési mód változtatása konkurenciakezelés nélkül

1. A konkurencia kezeléses feladat miatt adjunk a FizetesMod táblához egy verzió oszlopot, és a meglévő rekordoknál a verzio értékét állítsuk 1-re.
2. Helyezzünk az oldalra egy táblázatot, mely 1 sorból és két oszlopból áll.
3. A bal oldali cellába helyezzünk egy grid view-t.
4. A grid viewnak adjuk meg az adatforrását:
   a. Új SQL Server adatforrás
   b. Fizetés mód táblához kapcsolódik
   c. Az összes oszlopot válasszuk ki
   d. Advanced opciónál állítsuk be, hogy generáljon, insert, update és delete scriptet is
5. A grid view állítsuk be a következőket:
   a. Az ID oszlopot rejtsük el
   b. Engedélyezzük a rendezést a szerkesztést és a törlést
   c. Esetleg megadhatunk formázási stílust is
6. A grid view nem biztosítja számunkra új elem felvitelét, ehhez egy másik vezérlőelemre van szükség. A táblázat jobb oldali cellájába helyezzünk el egy details viewt
7. A Details view-nak állítsuk az adatforrását arra, amit a grid viewnál létrehoztunk.
8. A deatils viewn engedélyezzük az új rekordok felvitelét
9. A details viewnak állítsuk át a DefaultMode tulajdonságát insertre

Ezzel gyakorlatilag készen is van az adatok rögzítését támogató oldal. Próbáljuk ki, hogy mi történik, ha ketten módosítják ugyan azokat az adatokat!
Sok esetben ez a megoldás elégséges, de ha többen dolgoznak ugyan azon az adatokon, akkor előfordulhat, hogy egymás módosításait felül írják. Ezért ha ilyen előfordulhat és zavaró a munka során akkor ezt kezelni kell.

Lekérdezés optimalizálás

Tartalom

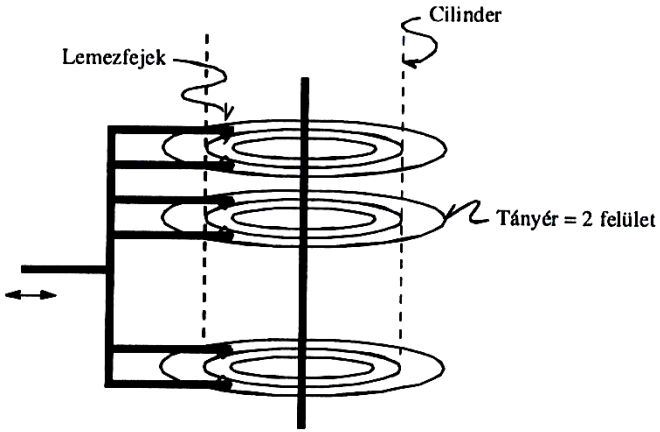
- **I/O költség**
- Lekérdezések feldolgozása
  - Logikai lekérdezési terv
  - Fizikai végrehajtási terv
- Oracle Server lekérdezés optimalizációja
- MS SQL Server lekérdezés optimalizációja
- Néhány jó tanács

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 cilinderre állása
- Rotációs késés: keresett blokk fej alá érkezése
- Adatátviteli idő



I/O költség csökkentése

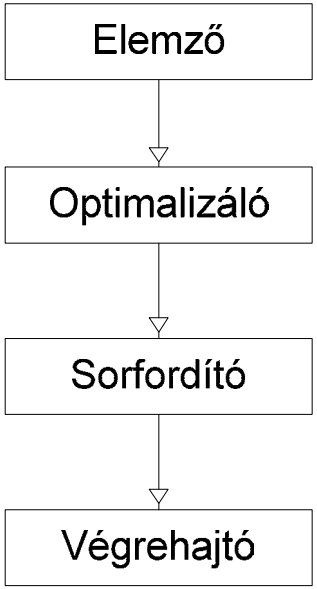
- Korai beolvasás
  - Nagyobb bufferek
  - Előre ismert a beolvasandó blokkok sorrendje
- Cilinder alapú szervezés
  - Egymást követő adatblokkok egy cilinderen
  - Rotációs késés csökkentése
- Több lemez használata (RAID)
  - Több fejszerelvény
  - Keresési idő csökkentése
- Lift algoritmus
  - Fejszerelvény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

Tartalom

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

Lekérdezés feldolgozó szerkezete

- Elemző
  - Lekérdezés fordítása
  - Logikai terv készítése
- Optimalizáló
  - Fizikai terv elkészítése
  - Táblák bejárása
  - Táblák összekapcsolása
- Sorfordító
  - Fizikai terv leképezése I/0 műveletekre
- Végrehajtó
  - Műveletek végrehajtása

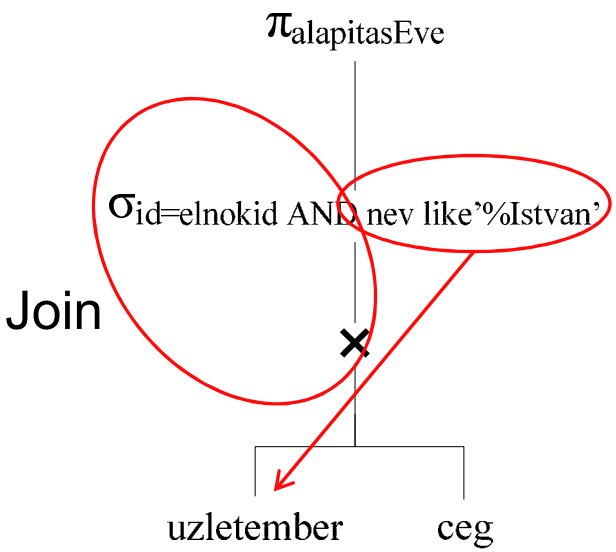


Logikai terv elemei

- Elemző fa (átalakítás után)
  - Relációk (levél elemek)
  - Műveletek (csomópontok)
  - Adatok áramlása (csak lentről fölfelé)
- Relációs algebrai műveletek
  - Descartes-szorzat (R×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 elemzőfája

```
select alapitasEve
from uzletember u, ceg c
where u.id = c.elnokid AND nev like '%István'
```

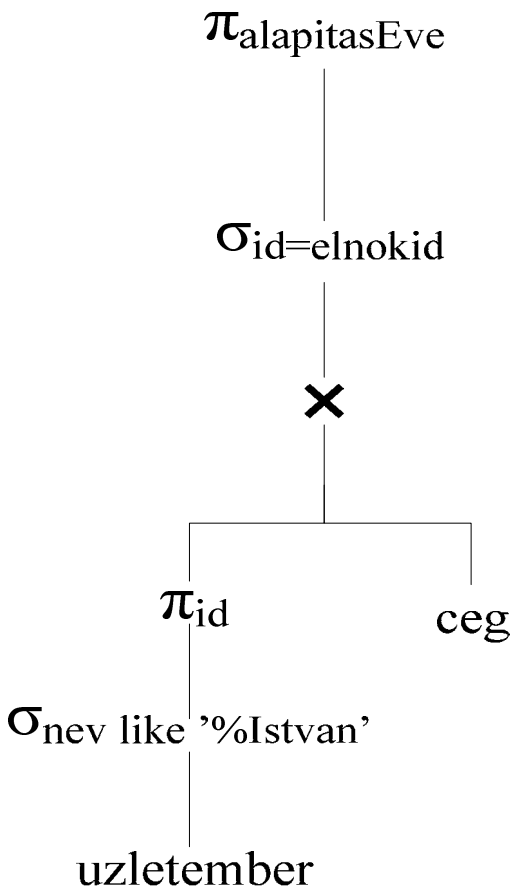


Összetett lekérdezések

- Allekérdezés
  - Nem korrelatív
    - Alkérdés független a fő kérdéstől
    - Külön elvégezhető
    - Tipikusan: in operátor
  - Korrelatív
    - Alkérdés függ a fő kérdéstől
    - A fő kérdés minden sorára le kell futtatni
    - Tipikusan: exsits operátor
- Nézetet tartalmaz
  - Általában nem optimalizálódik
  - Tárolt terv beillesztése

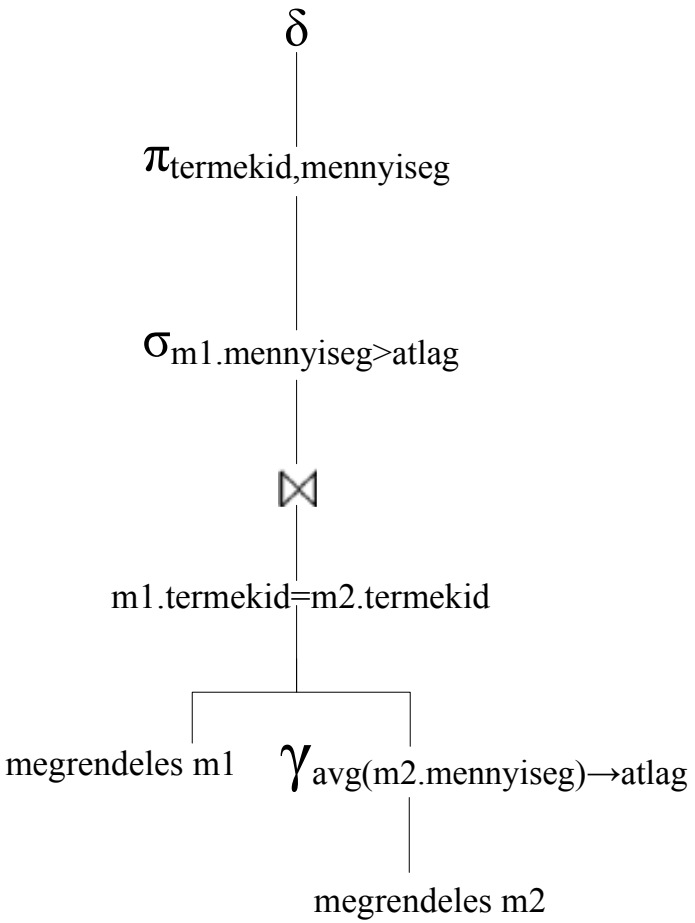
Nem korrelatív lekérdezés

```
select alapitasEve
from ceg c
where c.elnokid IN (select id
                    from uzletember u
                    where u.nev like '%István')
```



Korrelatív lekérdezése

```
select distinct termekid, mennyiseg
from megrendeles m1
where m1.mennyiseg > (
    select avg(m2.mennyiseg)
    from megrendeles m2
    where m2.termekid = m1.termekid);
```



Elemzőfa átalakítása – 1

- Optimális logikai terv kialakítása
- Fizikai végrehajtási terv keresési terének vágása
- Szabályok
  - 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

Elemző fa átalakítása – 2

- Kiválasztás
  - Felcserélési szabály: $\sigma_{F1}(\sigma_{F2}(R)) = \sigma_{F2}(\sigma_{F1}(R))$
  - Szétvágási szabály:
    - $\sigma_{F \text{ and } G}(R) = \sigma_F(\sigma_G(R))$
    - $\sigma_{F \text{ or } G}(R) = \sigma_F(R)$ UNION $\sigma_G(R)$ (Ha R-ben nincsenek ismétlődések)
- Összekapcsolás
  - $R \bowtie_F S = \sigma_F(R \times S)$
  - $R \bowtie S = S \bowtie R$
  - $(R \bowtie S) \bowtie U = R \bowtie (S \bowtie U)$.

$O(n!)$ sorrend

Elemző fa 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

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

Fizikai terv elemei

- Relációt beolvasó operátorok
  - Logikai terv levél eleminek beolvasása
- Relációs algebrai műveletet végrehajtó operátor
- Operátorok mérőszámai
  - Argumentumok száma
    - Hány reláción dolgozik
  - Menetek száma
    - Hányszor olvassa végig a relációkat

Fizikai tervek készítése

- Több fizikai terv készül
  - Tábla elérési módok
  - Join operátorok megvalósítási módjai
  - Join sorrend
- Költségbecslés
  - CPU idő
  - **I/O költség**
  - Memória használat

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

- Beolvasó operátorok megvalósítása
- Teljes átvizsgálás (Full table scan)
  - Nyalábolt esetben ~ *O(B)*
  - Nem nyalábolt tárolás ~ *O(R)*
- Index alapú átvizsgálás (Index scan)
  - σ operátor megvalósítása
    - Egyenlőség esetén $\approx O\left(\frac{B(R)}{V(R,a)}\right)$
  - Rendezés megvalósítása

Join operátorok megvalósítása

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

Nested Loops Join

- Egymásba ágyazott kettős for ciklus
- I/O költség
  - Nyalábolt esetben: *O(B(R)·B(S))*
  - Nem nyalábolt esetben: *O(T(R)·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))*

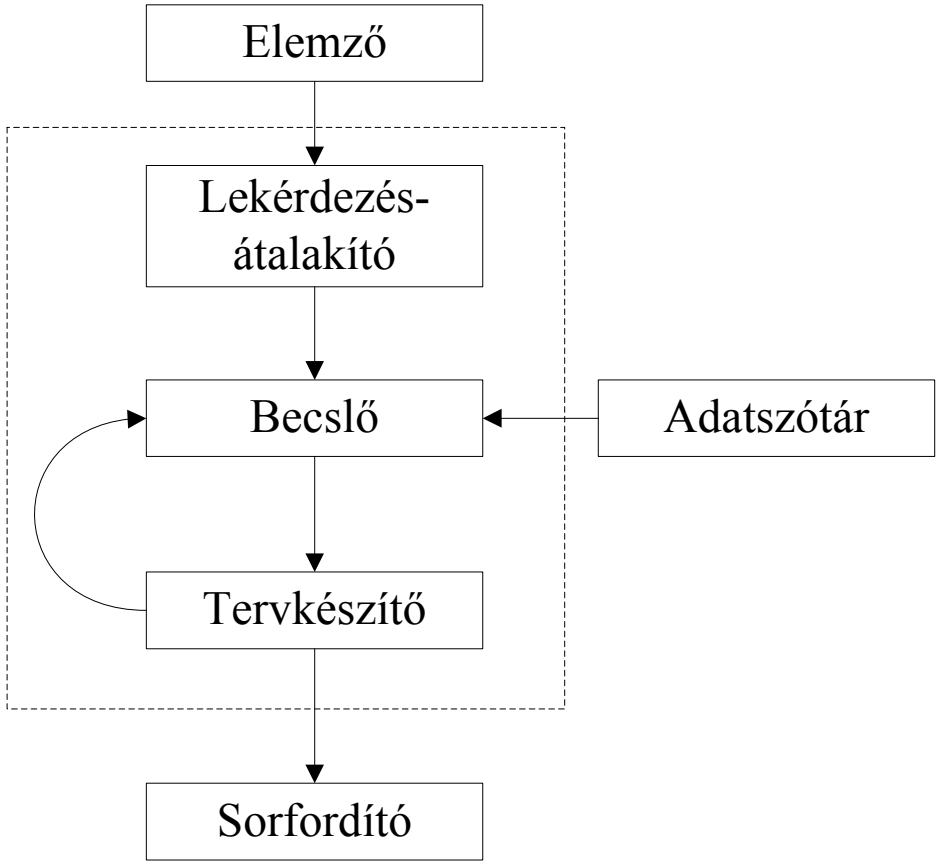
Sort merge join

- 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

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

Oracle Server lekérdezés optimalizálója



Optimalizáció 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

Lekérdezés átalakító szerepe

- 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 input adatai

- Default értékek használata
- Dinamikus mintavételezés
- Statisztikák
  - Adatszótárban tárolja
  - Hisztogramok az oszlop értékekre
  - Indexek telítettsége
  - Hiányában nagyon rossz terv születik

Táblaelérési 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

Táblaelérési módok – 2

- 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

Táblaelérési módok – 3

- 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

Táblaelérési módok – 4

- 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
- Bitmap index scan
  - Bitmap index alapú bool kifejezés kiértékelése

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 = 'married'		region = 'central'		region = 'west'						
0		0		0		0		0		0
1		1		0		1		1		1
1	AND	0	OR	1	=	1	AND	1	=	1
0		0		1		0		1		0
0		1		0		0		1		0
1		1		0		1		1		1

Összekapcsolási módok

- 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

Optimalizáló befolyásolása

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

Oracle Serever Indexei

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

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

SQL hint lehetőségek

- Preferált index használata
- Index használatának tiltása
- Join típus megadás
- Join sorrend megadása
- Táblaelérési mód megadása
- Párhuzamos végrehajtás
- Lekérdezés átírása
  - OR → Lekérdezések + Union all
- …

Tartalom

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

Optimalizálás

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

Keresési tér különböző mélységű vágása

Három fázisú optimalizáló

- 0. Fázis
  - Egyszerű átalakítások
  - Preferált hash join
  - Ha a költség < 0,2 → végrehajtás
- 1. Fázis
  - Kibővített átalakítások
  - Ha a költség < 1 → végrehajtás
- 2. Fázis
  - Párhuzamos végrehajtás vizsgálata

Táblaelérési módok – 1

- Table scan
  - Ha nincs semmilyen index
  - Táblára vonatkozó szűrési feltételt is kiértékeli
- Clustered index scan
  - Nyalábolt adatolvasás
  - Adatblokkok index szerint rendezve
  - Primary key mentén létrejön
  - Table scan helyett ezt preferálja

Táblaelérési 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

Optimalizáló befolyásolása

- Indexek deklarálásával
- 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

SQL Server 2008 sajátosságok 1

- Plan cache
  - Végrehajtási terv cache
  - Ha ugyan olyan struktúrájú lekérdezés jött
  - Statisztikák nem változtak
- Indexelt nézetek
  - Nézet eredményének tárolása
  - Index is rendelhető hozzá

SQL Server 2008 sajátosságok 2

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

Tartalom

- I/O költség
- Lekérdezések feldolgozása
  - 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
  - Kerüljük a select * -ot
  - Törekedjünk az egyszerűségre
  - „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

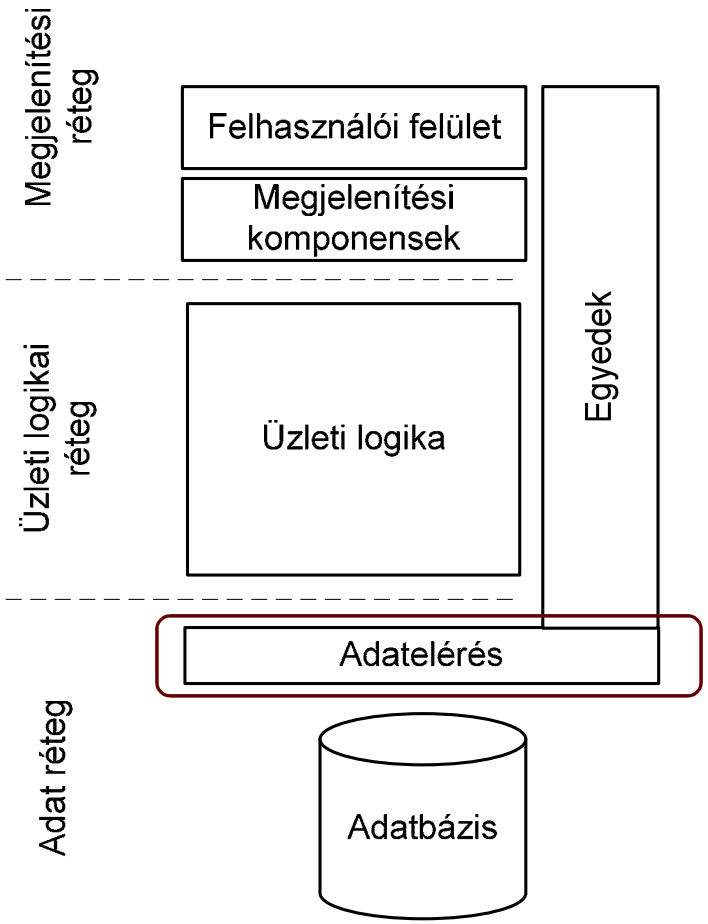
Demó

Végrehajtási tervek

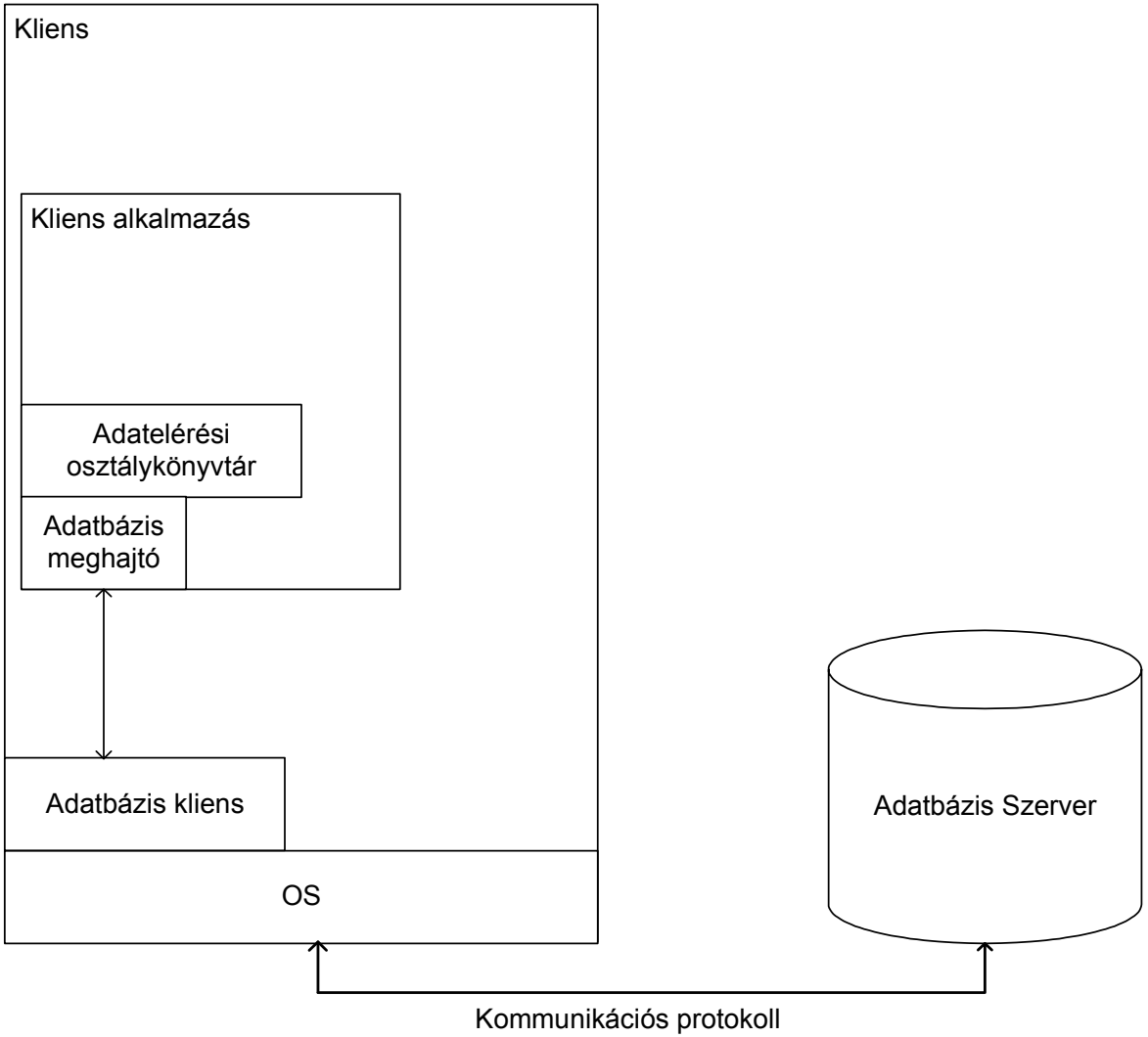
Indexek

Adatelérés – ADO.NET

Háromrétegű architektúra



Adatelérési könyvtár szerepe



Adatelérési könyvtár feladata

- Adatbázis használat absztrakció
- Tipikus elmei
  - Connection
    - Adatbázis meghajtók
  - Command
    - Paraméterek
  - ResultSet
  - Exceptions

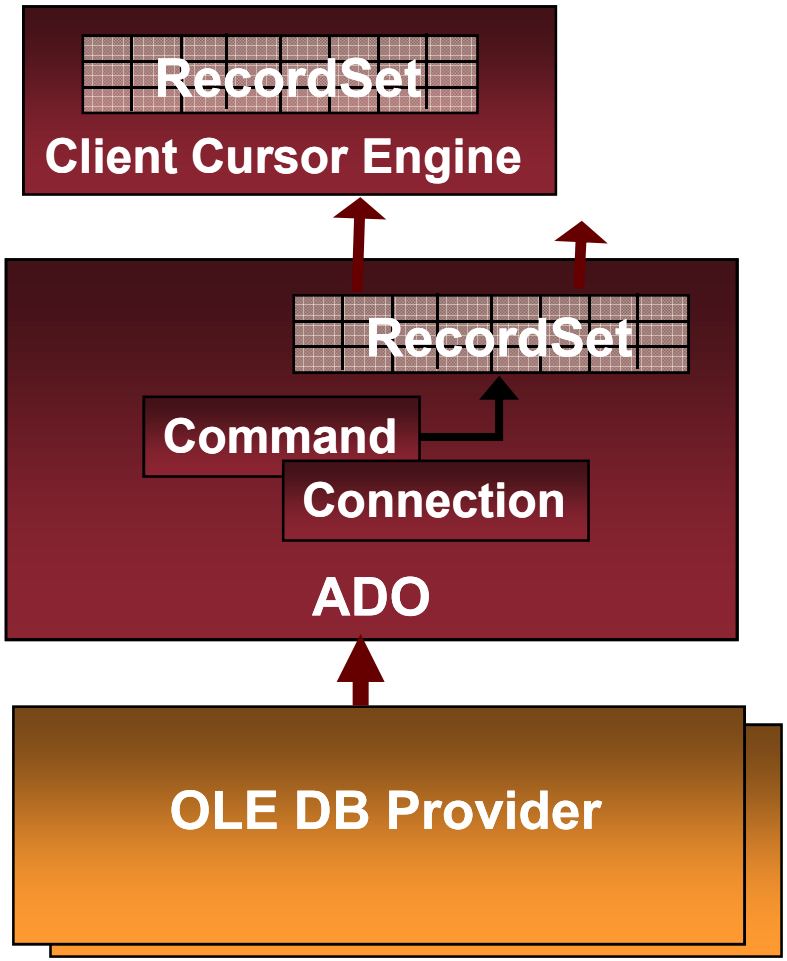
Az ADO.NET általában

- Relációs adatbázisok elérése
- Adatbázis független
- Adatbázis interfész független
- Interfészek és absztrakt osztályok
- Implementációk
  - Megvalósítják az alap funkciókat
  - Ki is terjeszthetik azokat
- Egységes, adatbázis független kódolás

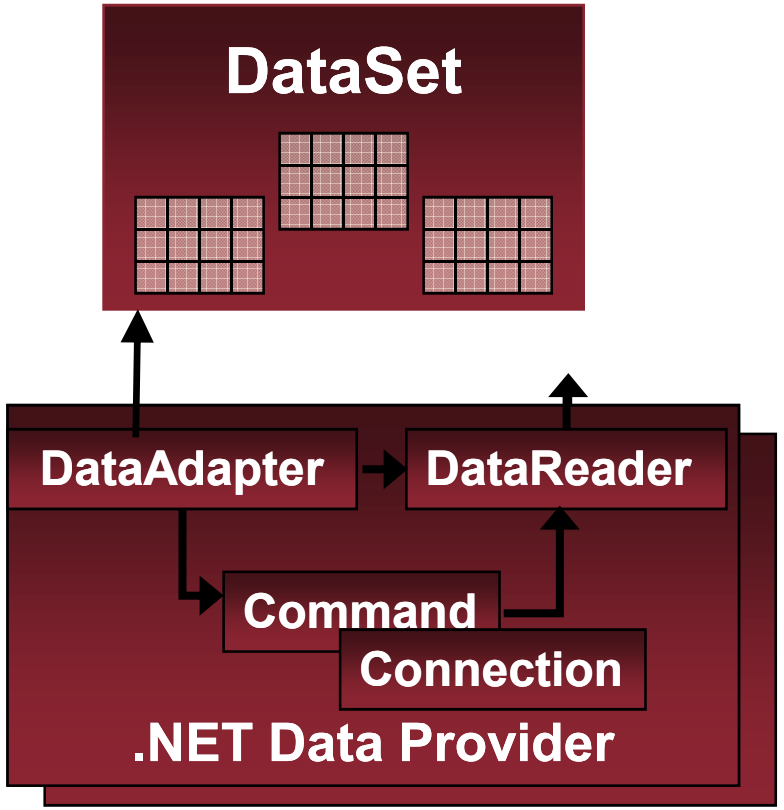
ADO.NET implementációk

- A .NET framework részeként
  - OleDb
  - Microsoft SQL Server
- Egyéb implementációk a Microsoft-tól
  - ODBC
  - Oracle
- Más cégek implementációi
  - PI: MySQL

Az ADO és ADO.NET összehasonlítása

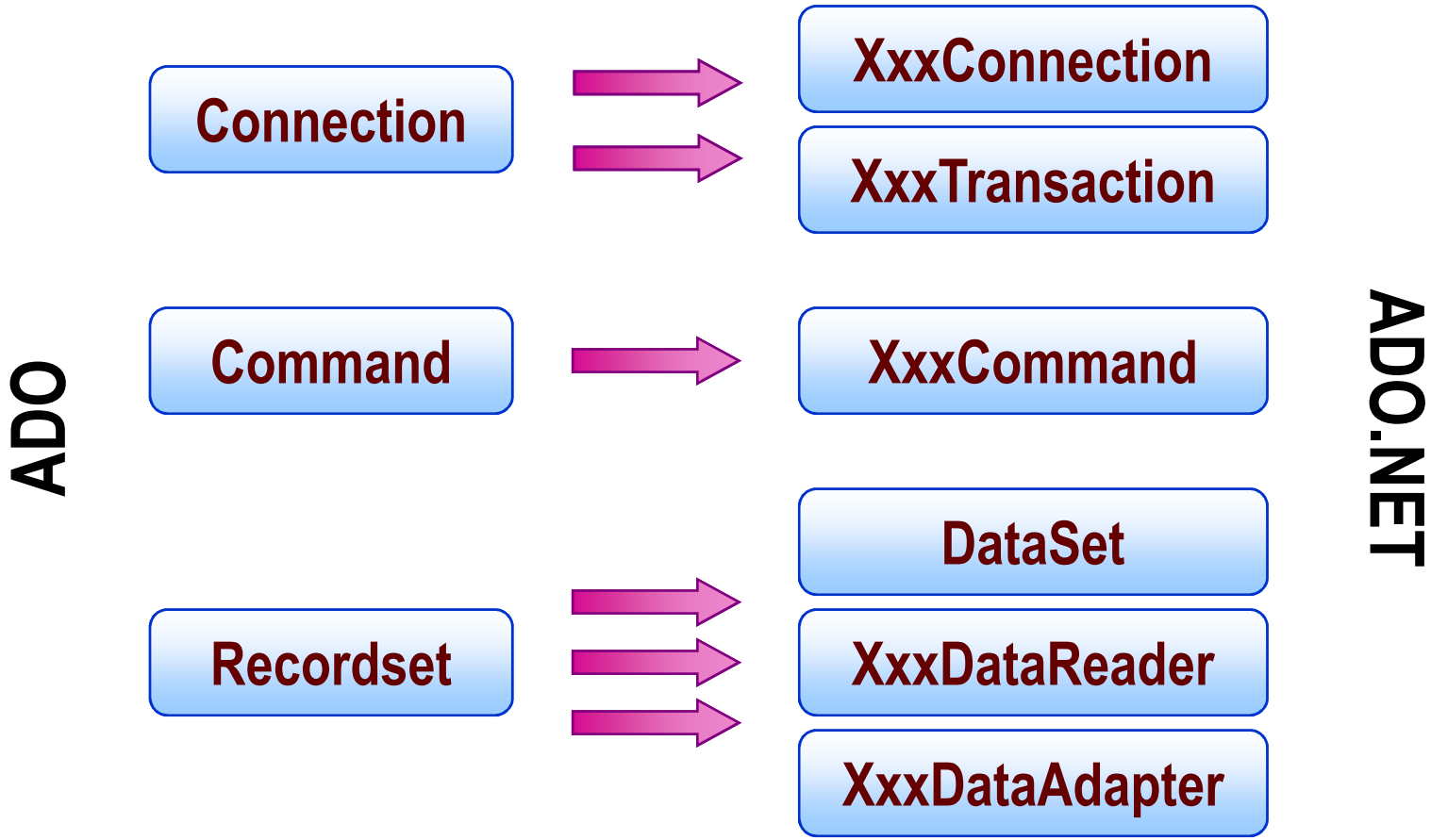


ADO



ADO.NET

Az ADO és ADO.NET összehasonlítása



Kapcsolati modellek

- Kapcsolat alapú modell
  - Folyamatos kapcsolat az adatbázis szerverrel
- Kapcsolat nélküli modell
  - Kapcsolat csak az adatmanipuláció idejére

A kapcsolat alapú modell

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

A kapcsolat nélküli modell

- Előnyök
  - Nem szükséges folyamatos hálózati kapcsolat
  - Skálázhatóság
- Hátrányok
  - Az adatok nem mindig a legfrissebbek
  - Ütközések lehetségesek

DataReader kontra DataSet

- DataReader
  - Egyirányú, csak olvasható
  - Ha az adatokat azonnal feldolgozzuk
- DataSet
  - Lokális adat cache
  - Ha az adatok közt navigálunk
  - Ha az adatokat módosítjuk is

Az 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

Conncetion String

- DB szervertől függ a szintaktika

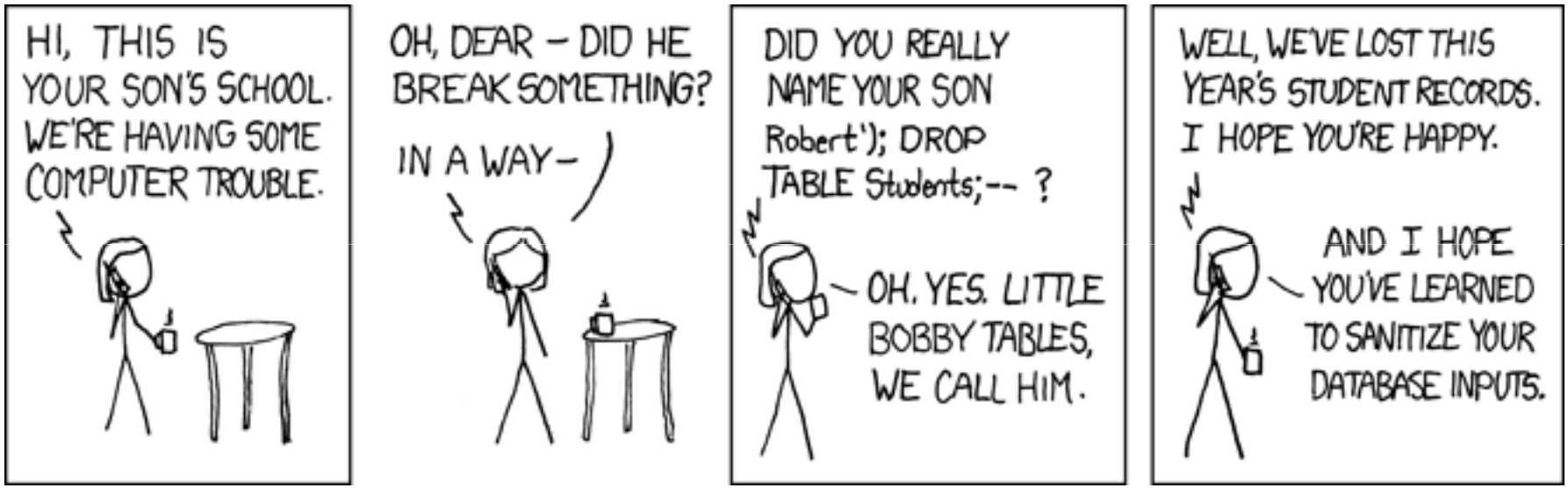
```
static string _connectionStringValue = "User
    ID=LOGIN;Password=PASSWD;Persist Security Info=false;Initial
    Catalog=AdventureWorks;Data Source=DATASOURCE;Packet
    Size=4096";
```

- http://www.connectionstrings.com/ ☺
- Connection string alapú támadások
  - All input is evil
  - Connection String Builder
    - DB platformonként
    - Paraméterek használata

Az 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

SQL Injection – 1



SQL Injection – 2

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

Demó

SQL Injection

A parancs végrehajtása

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

Adatbázis parancs – 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();
```

Gyakorlati tanácsok

- Paraméterek használata
  - Speciális karakterek nem okoznak gondot
  - Biztonságos
    - SQL Injection elleni védekezés
- Tárolt eljárások használata
  - Gyorsabb
  - Hálózati forgalom spórolható meg

Tranzakciók és az ADO.NET

- Tranzakció létrehozása
  - BeginTransaction metódus
- Parancsok tranzakcióhoz rendelése
  - Transaction tulajdonság
- Parancsok futtatása
- Tranzakció befejezése
  - CommitTransaction
  - RollbackTransaction

Az izolációs szint meghatározása

- ADO .NET
  - A BeginTransaction paramétere
  - Szerver függő izolációs szintek

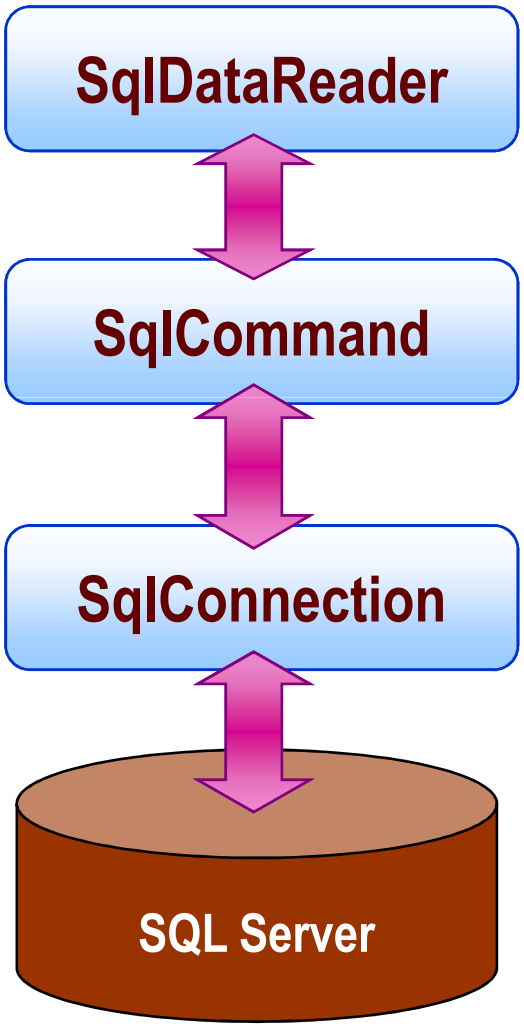
Hibakezelés – 1

- A 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

Hibakezelés – 2

```
using (SqlConnection connection = new
  SqlConnection(connectionString))
        {
            connection.Open();
            …
        }
```

Adatbázis kezelés DataReader-el



- A kapcsolat alapú modellben az adatok végig az adatbáziban maradnak
  1. Kapcsolat megnyitása
  2. Parancs futtatása
  3. Eredmény feldolgozása
  4. Reader lezárása
  5. Kapcsolat lezárása

```
using (connection)
 {
    SqlCommand command = new SqlCommand(
     "SELECT CategoryID, CategoryName FROM Categories;“,   connection);
    connection.Open();
 try {
      SqlDataReader reader = command.ExecuteReader();
      if (reader.HasRows)
     {
        while (reader.Read())
        {
        …
       }
   }
 finally{  reader.Close(); }
}
```

Adatbázis kezelés DataSettel

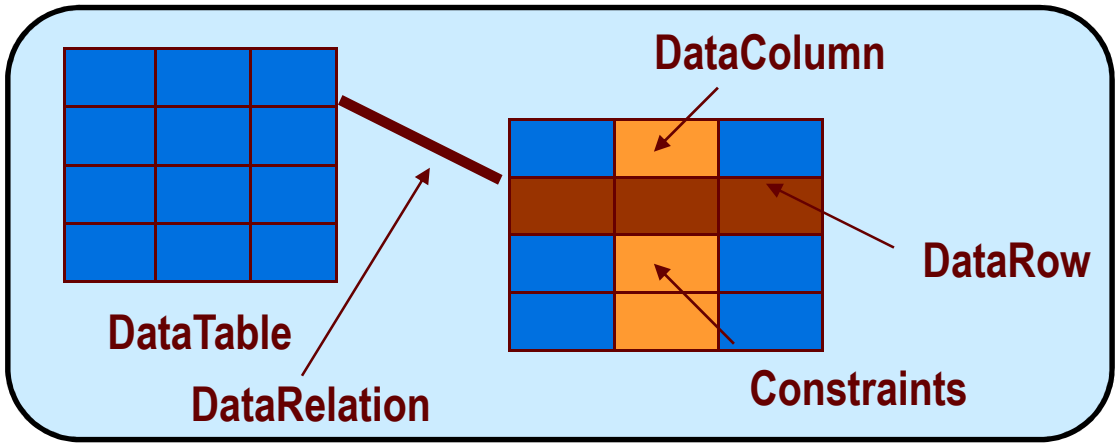
- Adatbázis a memóriában
- Tartalma
  - Táblák, mezők, sorok
  - Nézetek, kényszerek
  - Kapcsolatok (szülő-gyermek)
- Lehetőségei
  - Feltöltése (alapállapot)
  - Visszatöltése (minden, ami változott)
  - Nagyon erős Xml sorosítási támogatás

A DataSethez kapcsolódó osztályok

- System.Data névtér (adatbázis független)
  - DataRow
  - DataColumn
  - DataTable
  - DataView
  - DataAdapter
  - DataSet

A DataSethez kapcsolódó osztályok

- Gyűjtemények
  - Tables
  - Relations
- Séma: XSD alapján vagy kódból



A DataSet egyéb lehetőségei – 1

- Kényszerek
  - Egyediség biztosítása (UniqueConstraint)
  - Elsődleges kulcs
  - Külső kulcs (ForeignKeyConstraint)
- Kapcsolatok (DataRelation)
  - Közös értékű mező alapján
  - Elsődleges kulcs – Külső kulcs
  - Szülő gyermek kapcsolatok

A DataSet egyéb lehetőségei – 2

- Számított értékű mezők
  - Felhasználhatóak
    - Más mezők értékei
    - Konstansok
    - Matematikai műveletek
    - Oszlopfüggvények (A kapcsolatokban)
  - DataColumn osztály
    - Expression tulajdonság

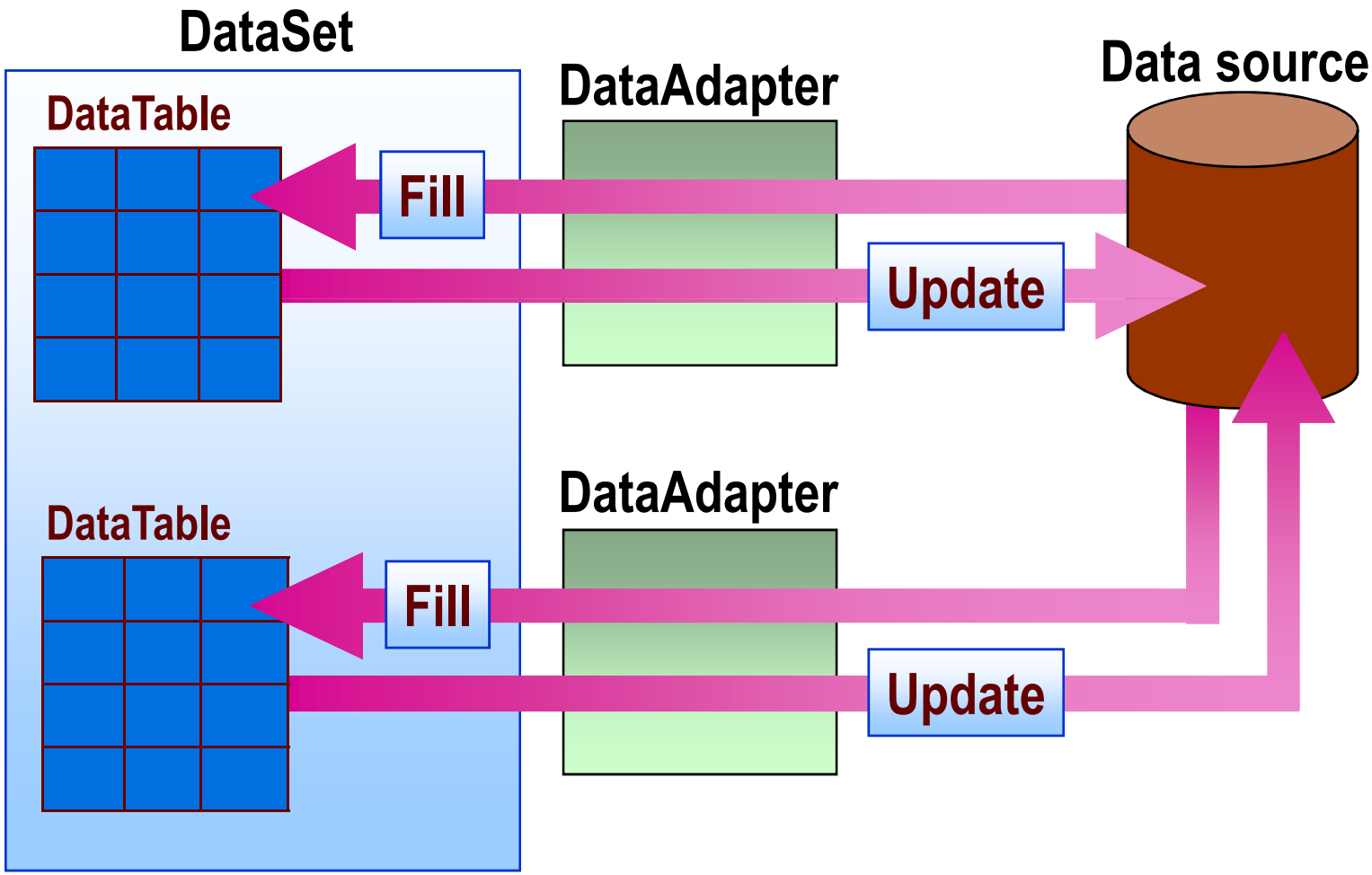
Típusos DataSet

- Definiált struktúrájú DataSet
- Struktúra leírása: XML séma
- Generált osztályok
  - Az előzőekből öröklődnek
  - Entitásra jellemző tulajdonságok, metódusok
- Előnyei
  - Gyorsabb fejlesztés
  - Típusosság fordítási időben

DataAdapter

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

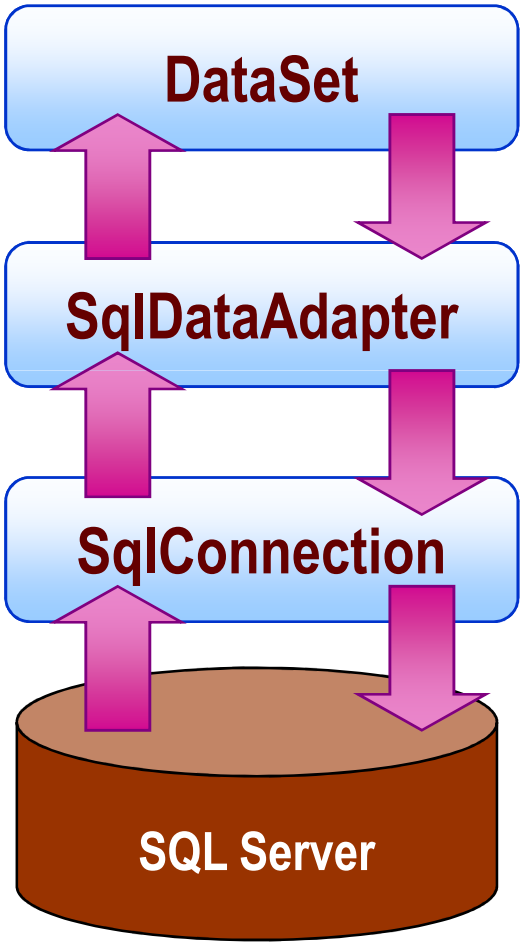
DataAdapter



DataView

- Nézet egy tábla fölött
- DataTable változásait követi →adatkötés
- Lehetőségei
  - Szűrés
  - Sorrendezés
  - Lekérdezés
  - Módosítás engedélyezése/tiltása
  - Beszúrás engedélyezése/tiltása
  - Törlés engedélyezése/tiltása

Adatbázis kezelés DataSet-el - Összegző ábra



- A kapcsolat nélküli modellben az adatok a kliensnél vannak feldolgozás közben
  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

Állapot követés

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

Verzió követés

- 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["SupplierID", DataRowVersion.Original]

Változáskövetés

- DataTable, DataSet és DataAdapter szinten támogatott
  - GetChangeSet()
  - Merge()
  - Load()
- Csak óvatosan, a feldolgozás táblaszinten történik
  - Néha kézzel kell ellenőrizni a táblák frissítési sorrendjét
  - A Merge idejére DataSet szinten kikapcsolhatjuk a konzisztencia ellenőrzését

Ütközések kezelése

- Probléma
  - Több felhasználó ugyan azt módosítja
- Pesszimista ütközés kezelés
  - Zárak használata
  - Kapcsolat alapú modell esetén
- Optimista ütközés kezelés
  - Ütközések feloldása
  - Kapcsolat nélküli modell esetén

Ütközésvizsgálat

- Pesszimista ütközésvizsgálat:
  - Adatbázis szintű zárak, pl. tranzakcióval
- Optimista ütközésvizsgálat:
  - Tartalom alapján döntünk
    - TimeStamp
    - Teljes tartalomvizsgálat
  - DataAdapteren állítjuk be
    - Különböző SQL kódot generál

FEGURM probléma

- Egy egyszerű adatmódosítás 6 lépést igényelne, ha el akarjuk kerülni az adatvesztést
- Fill / Edit / Get Changes / Update / Refresh / Merge
- Megoldás: DataSet.Load()
  - FillOption
    - OverwriteChanges (mint a Fill)
      - Az „Original” és a „Current” verziót módosítja
    - PreserveChanges (alapértelmezett):
      - Az „Original” verziót módosítja
    - Upsert
      - A „Current” verziót módosítja

Azonosítók

- Probléma: Az elsődleges kulcsok szerver oldalon generálódnak, átmenetileg mégis szükség lehet rájuk kliens oldalon
- Megoldás
  - Ideiglenes azonosítók generálása
    - Autoincrement, seed=-1, step=-1
  - Ha más kliens oldali hivatkozások is vannak az új kulcsra, akkor ez kevés
    - Egyedi szerver oldali megoldás

Pesszimista lock a BLL-ben

- Alternatívák
  - Tranzakciókezelés
  - Egyedi implementáció
    - Kis lockmanagger osztály, bejegyzi, hogy melyik ID-t módosítják
    - A nyilvántartáshoz elég egy Hashtable
      - Kulcs: tábla + ID
    - További problémák:
      - Gyerekek zárolása
      - Shared/explicit zárolás
      - Szerver farm, cluster

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
  - Az ExecuteReader szerver oldali erőforrásokat foglal, ezért ha nincs feltétlenül szükségünk a szolgáltatásaira, ne használjuk.
  - INSERT / UPDATE / DELETE műveletek gyorsabbak, ha ExecuteNonQuery-vel futtatjuk őket
  - Egy adatmező lekéréséhez használjuk az ExecuteScalar függvényt

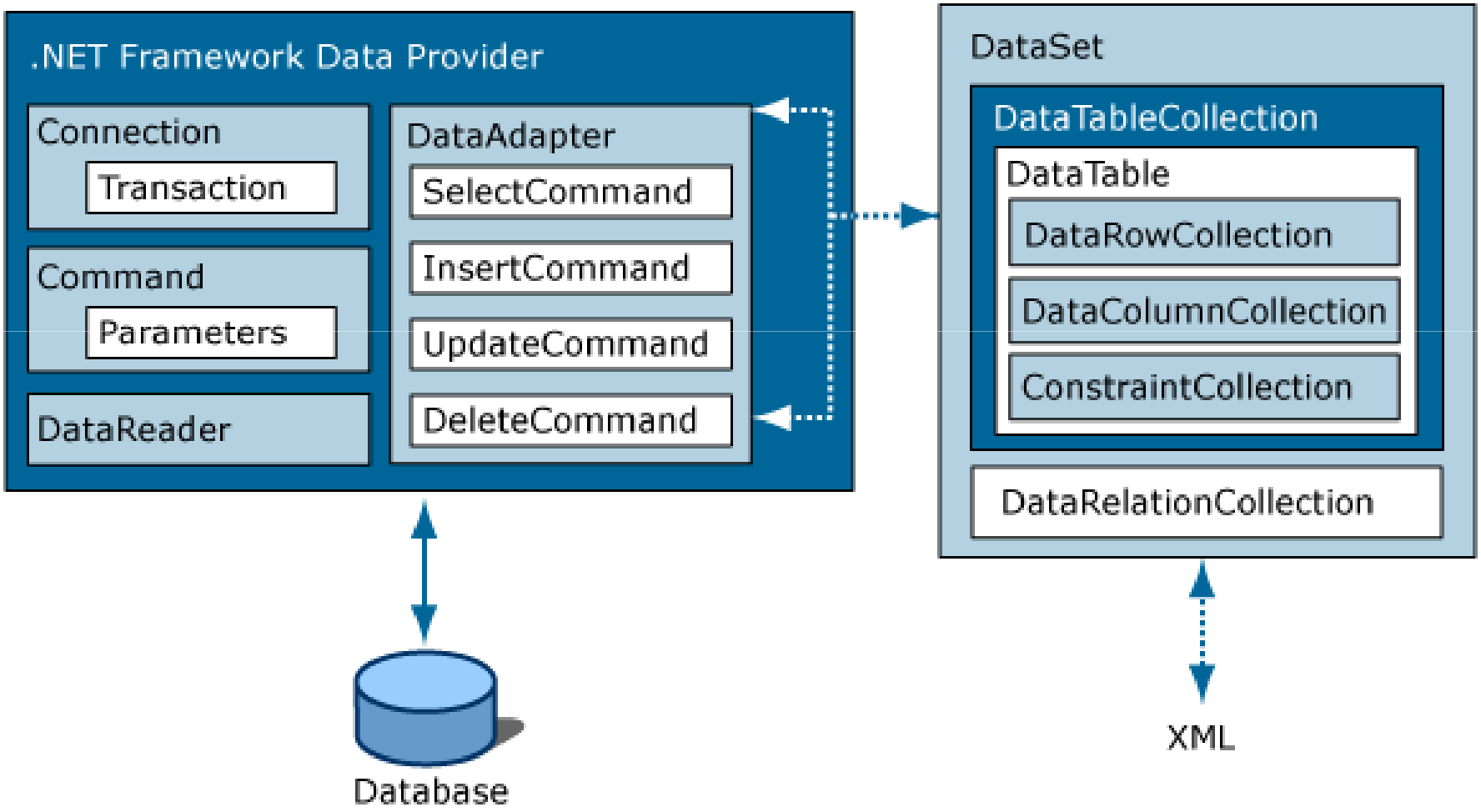
ADO.NET optimalizálás – 3

- 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

ADO.NET optimalizálás – 4

- DataProvider választás
  - A 100%-osan .Net alapú az ideális
    - Sajnos kevés ilyen van ☹
  - Kevert megoldások (ODBC kerülendő)
- DataSet / DataReader
  - DataReader: hatékonyabb, célirányosabb, szerveroldali erőforrásokat foglal, kis memóriaigény
  - DataSet: kapcsolat nélküli munkához, többrétegű architektúrához, nagyon nagy kliens oldali overhead, elsősorban memóriában

Összefoglalás

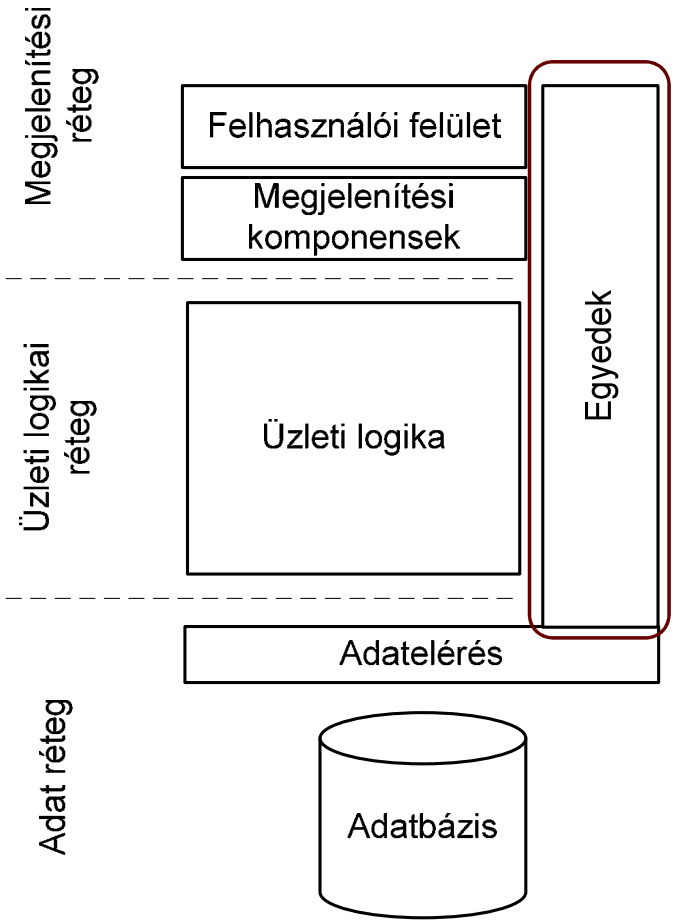


Demó

ADO.NET

O/R Mapping

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

ORM feladata

- Object Relational Mapping
  - Üzleti objektumok leképezése relációs adatmodellre
  - Adattárolás és üzleti folyamatok összekötése
- Problémák
  - Eltérő koncepciók
  - Öröklődés
  - Shadow információk
  - …

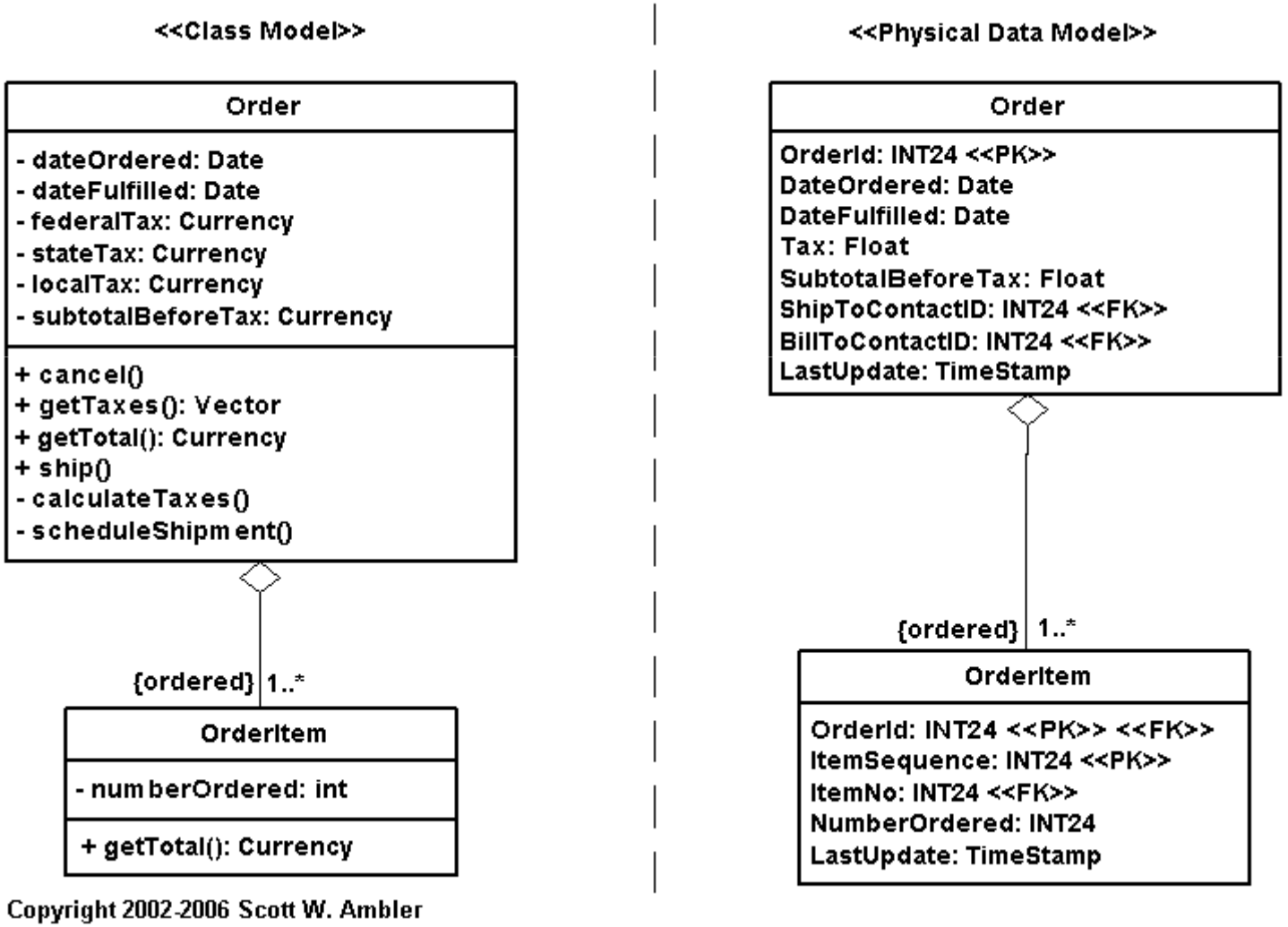
Irodalom

- Felhasznált irodalom, ábrák, példák:
  - http://www.agiledata.org/essays/mappingObjects.html

Egyszerű elgondolás

- Egyszerű megoldás
  - Objektum → Tábla (?)
  - Adattag → Oszlop(?)
- Probléma
  - Nem minden attribútum perzisztens
    - Ideiglenes kalkulálódik
  - Vannak attribútumok, melyek önmagukban objektumok
  - Adattípusok leképezése

Egyszerű mintapélda – 1



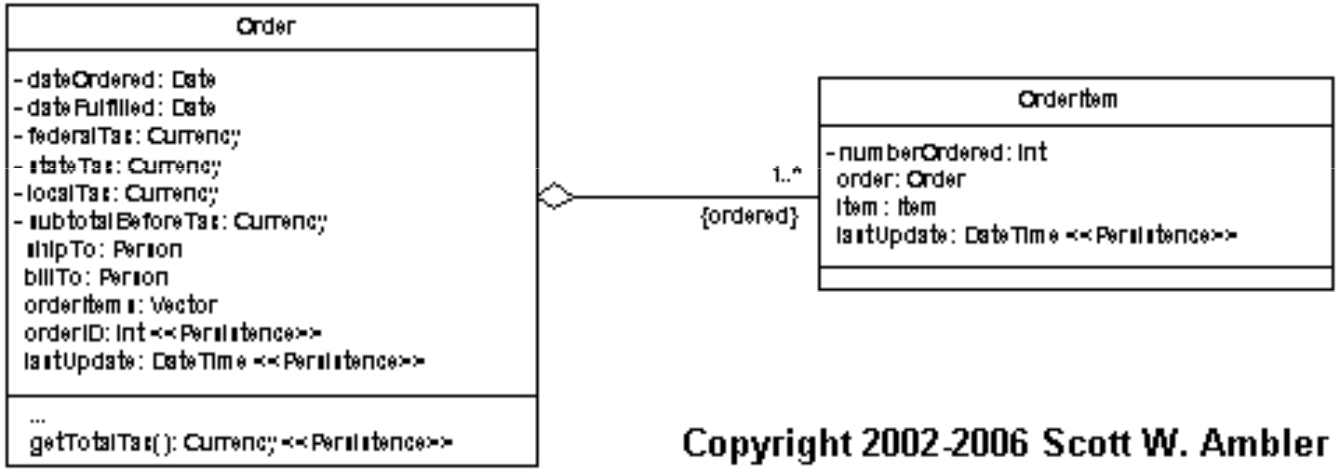
Egyszerű mintapélda – 2

- 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 Infromation

- Szükségesek az objektumnak a perzisztencia implementálásához
  - Kulcsok
  - Időbélyegek (optimista konkurencia kezelés)
  - Új/ már korábban mentett objektum
    - Insert/ Update
  - Nem fontos az üzleti objektumba helyezni
    - EJB/ JDO
    - Kezelni kell valahogy

Shadow information beépítése



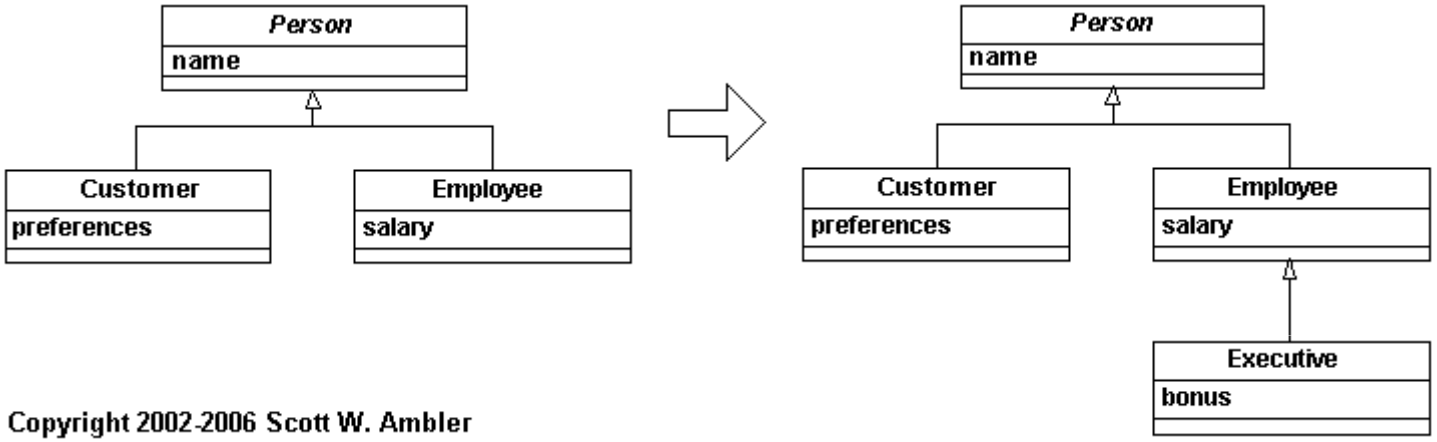
Meta információk összerendelése

Property	Column
Order.orderID	Order.OrderID
Order.dateOrdered	Order.DateOrdered
Order.dateFulfilled	Order.DateFulfilled
Order.getTotalTax()	Order.Tax
Order.subtotalBeforeTax	Order.SubtotalBeforeTax
Order.shipTo.personID	Order.ShipToContactID
Order.billTo.personID	Order.BillToContactID
Order.lastUpdate	Order.LastUpdate
OrderItem.ordered	OrderItem.OrderID
Order.orderItems.position(orderItem)	OrderItem.ItemSequence
OrderItem.item.number	OrderItem.ItemNo
OrderItem.numberOrdered	OrderItem.NumberOrdered
OrderItem.lastUpdate	OrderItem.LastUpdate

Öröklődés modellezése

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

Mintapélda

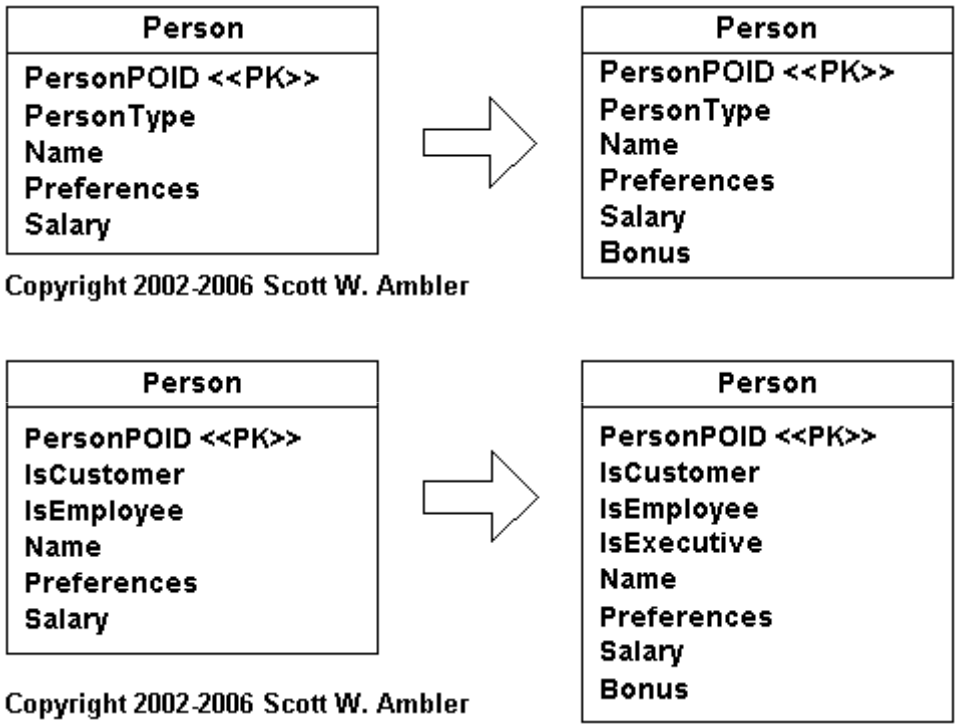


- Absztrakt Person osztály
- CR miatt új osztályt kell implementálni
  - Executive

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



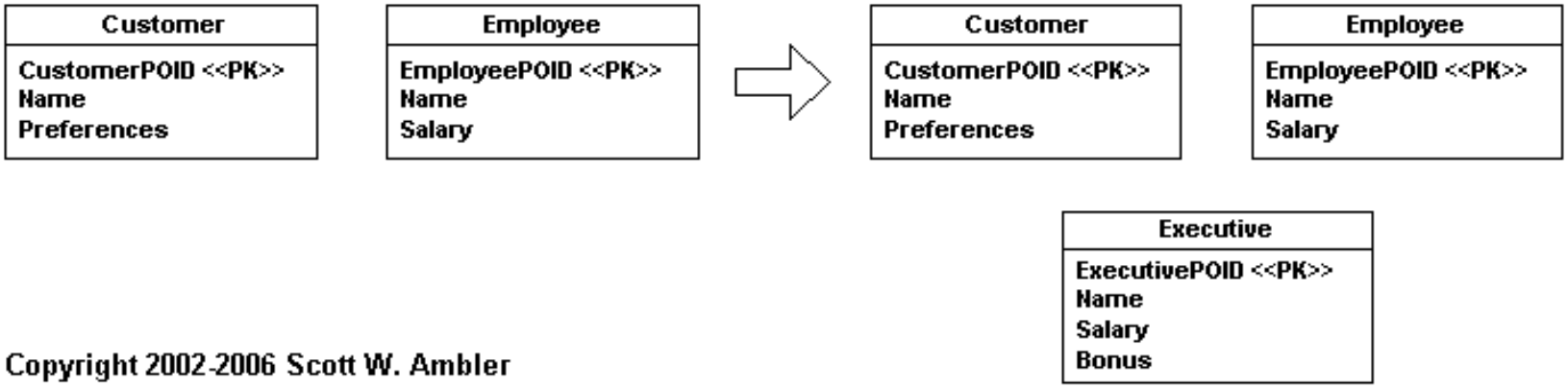
Egy táblába történő leképezés – 3

- 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 leképzése táblába – 2



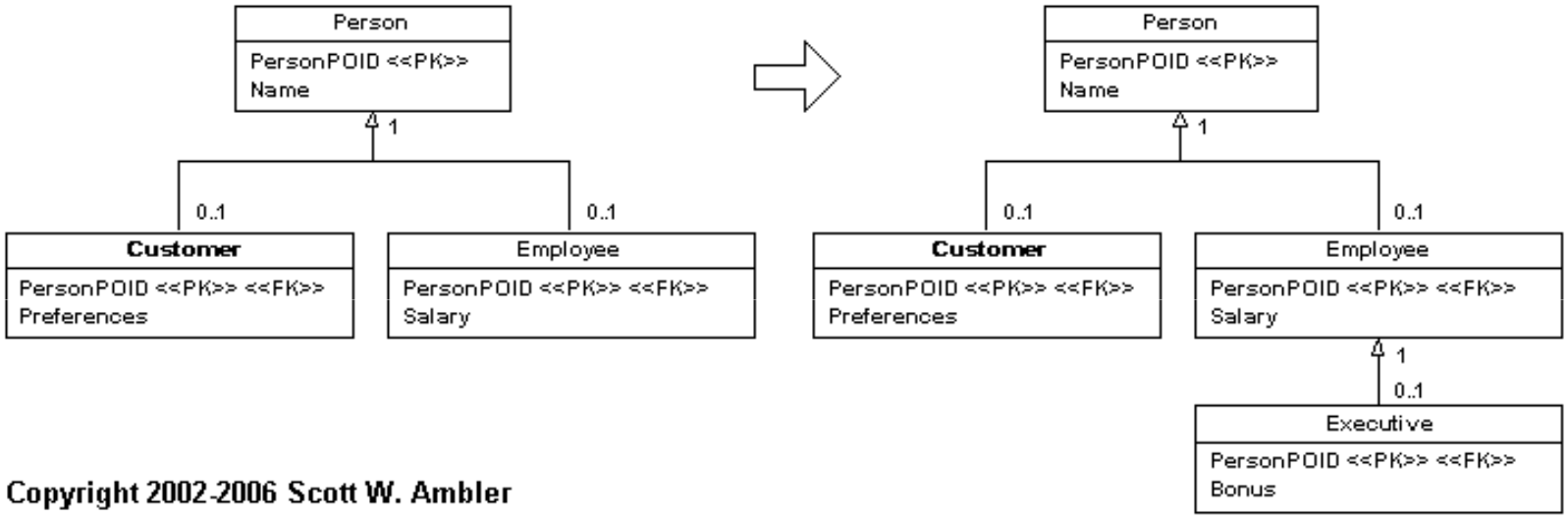
Valós osztályok leképzése táblába – 3

- 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



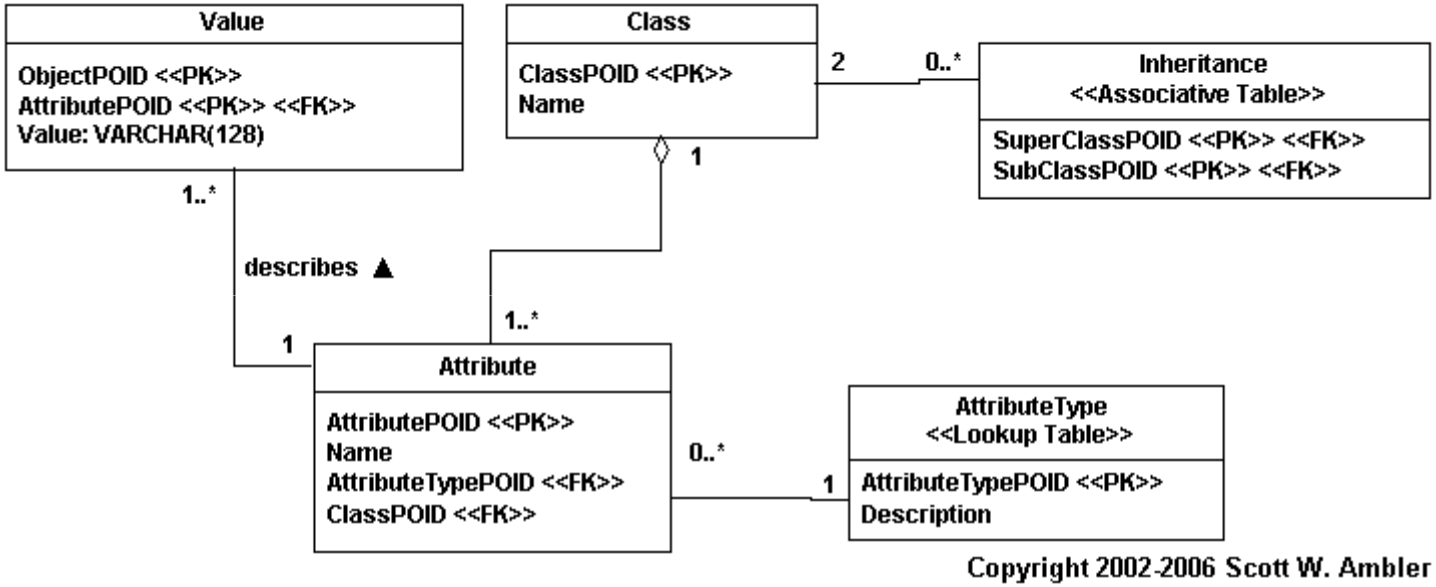
Összes osztály leképezése táblába – 3

- 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 struktúrába – 2



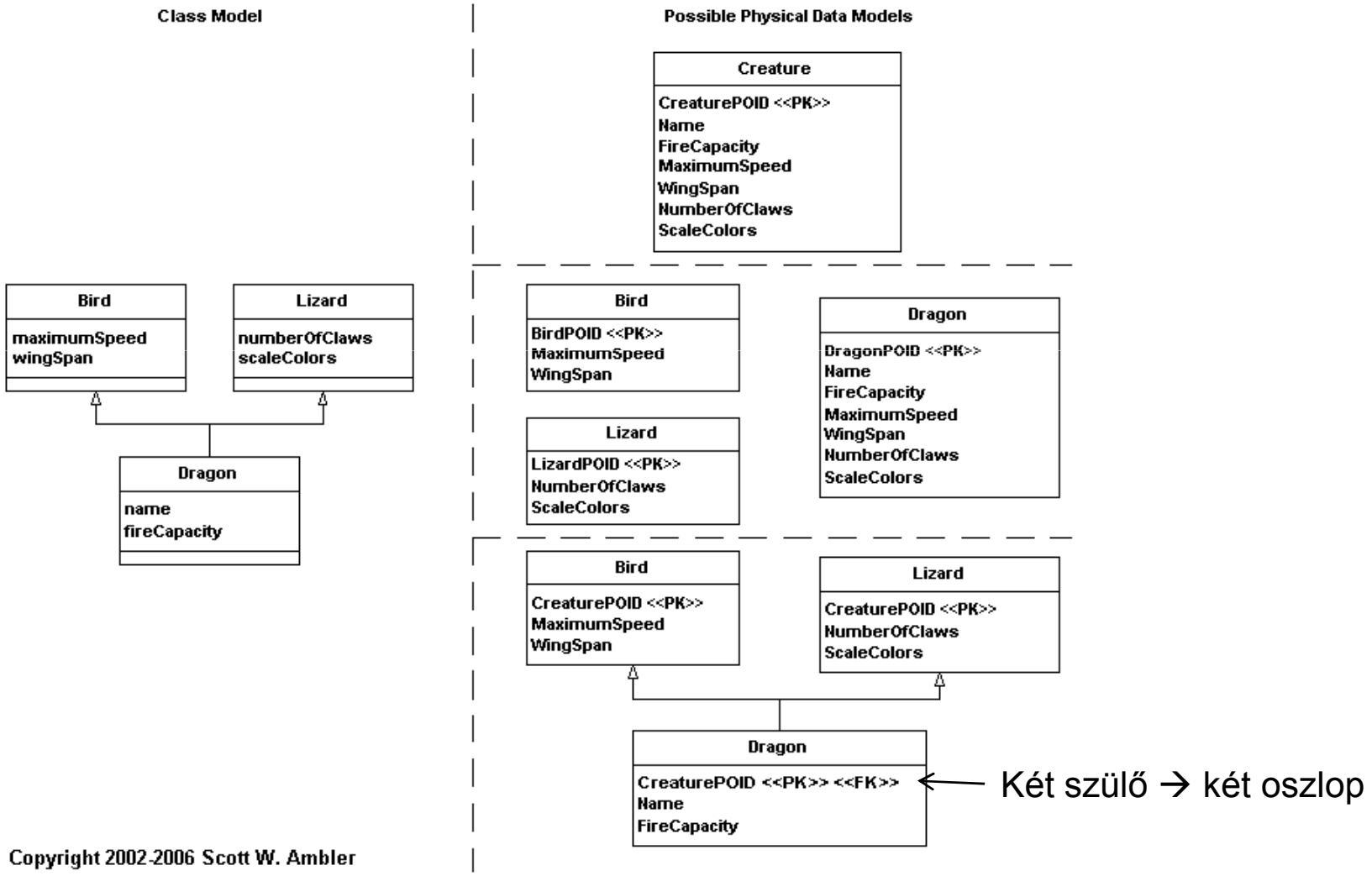
Leképezés általános struktúrába – 3

- Előnyök
  - Perzisztencia kezelő keretrendszerekhez illeszkedik
  - Flexiblis
  - „Bármi” 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



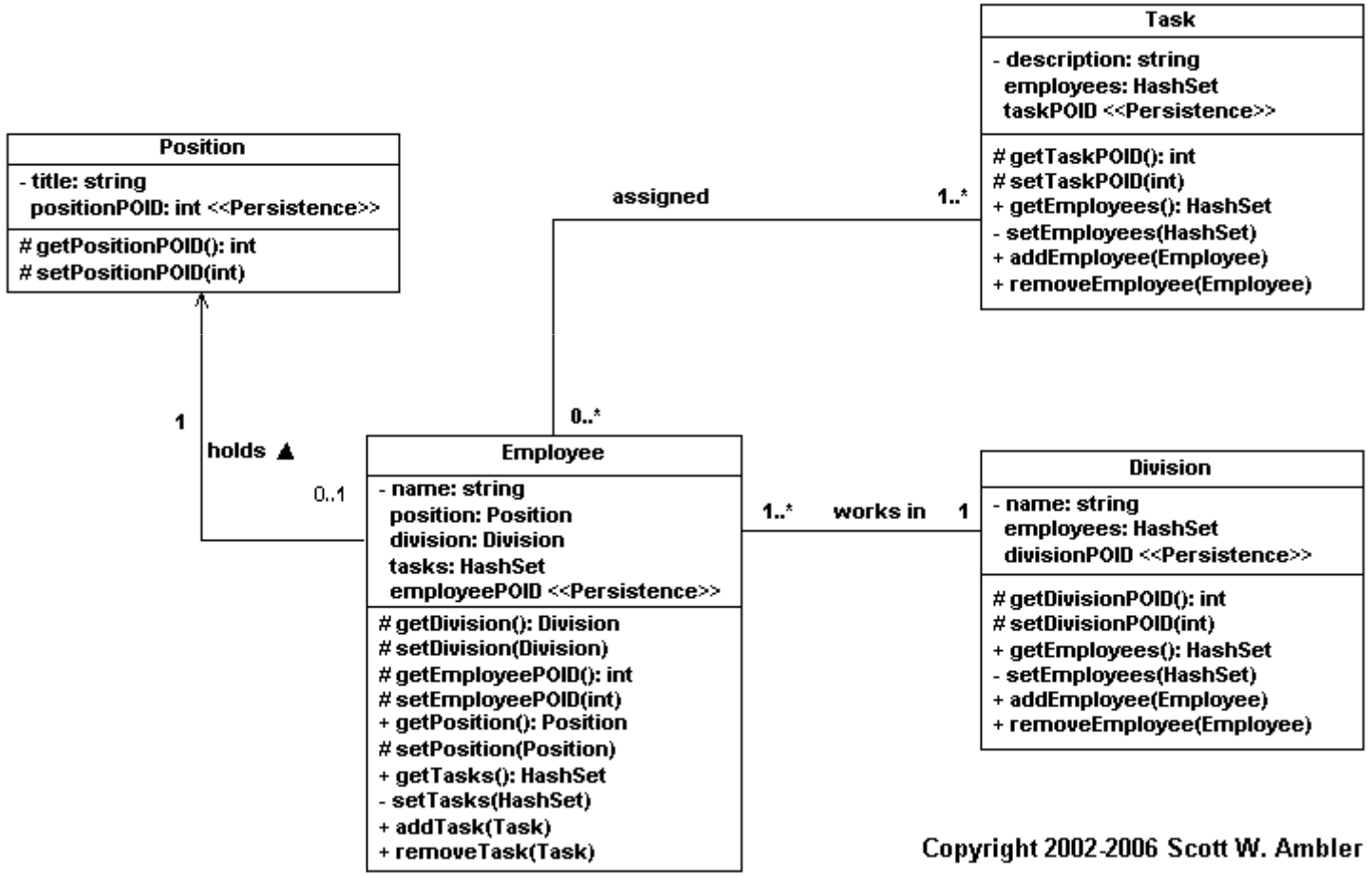
Objektum kapcsolatok leképzése – 1

- Kapcsolatok
  - Asszociáció
  - Aggregáció
  - Kompozíció

  } →Referenciális integritás
- Típusok
  - Egy – Egy
  - Egy – Több
  - Több – Több
- Irány
  - Egy irányú
  - Két irányú

  } →Nem képezhető le

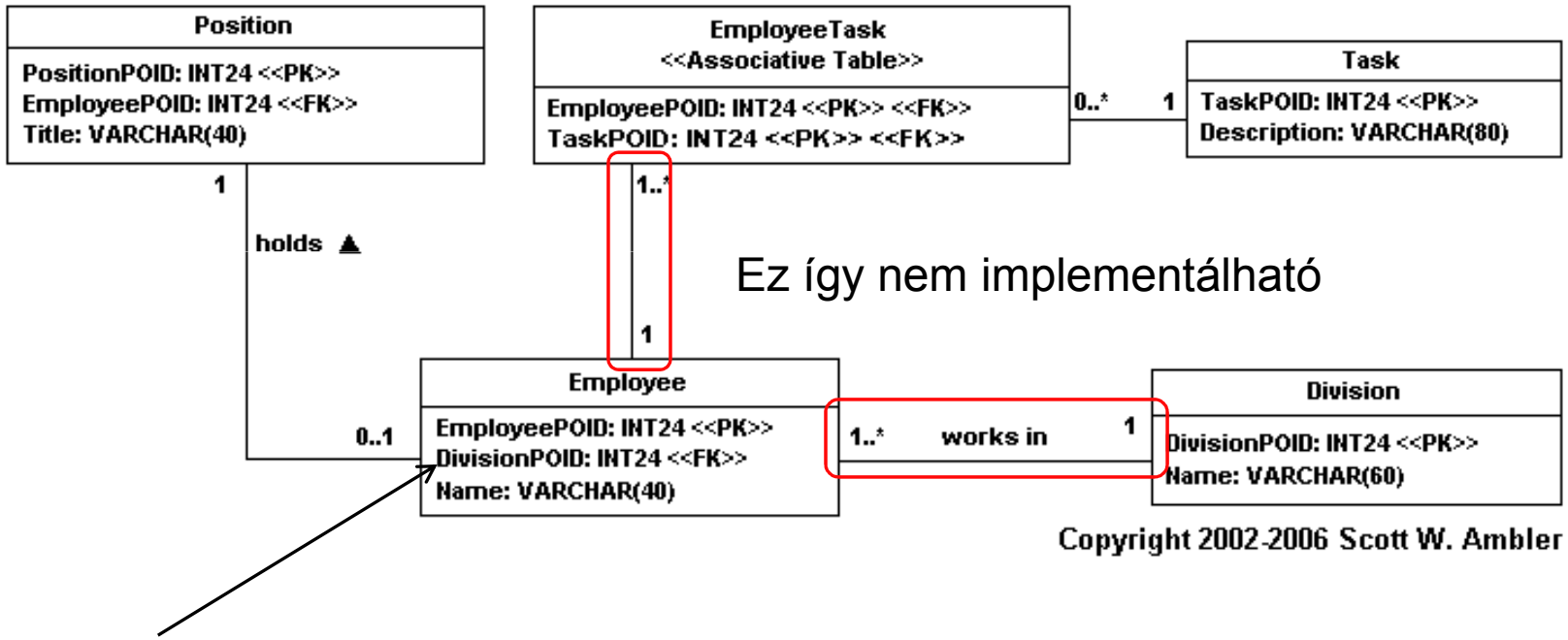
Objektum kapcsolatok leképzése – 2



Objektum kapcsolatok leképzése – 3

- 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

Objektum kapcsolatok leképzése – 4

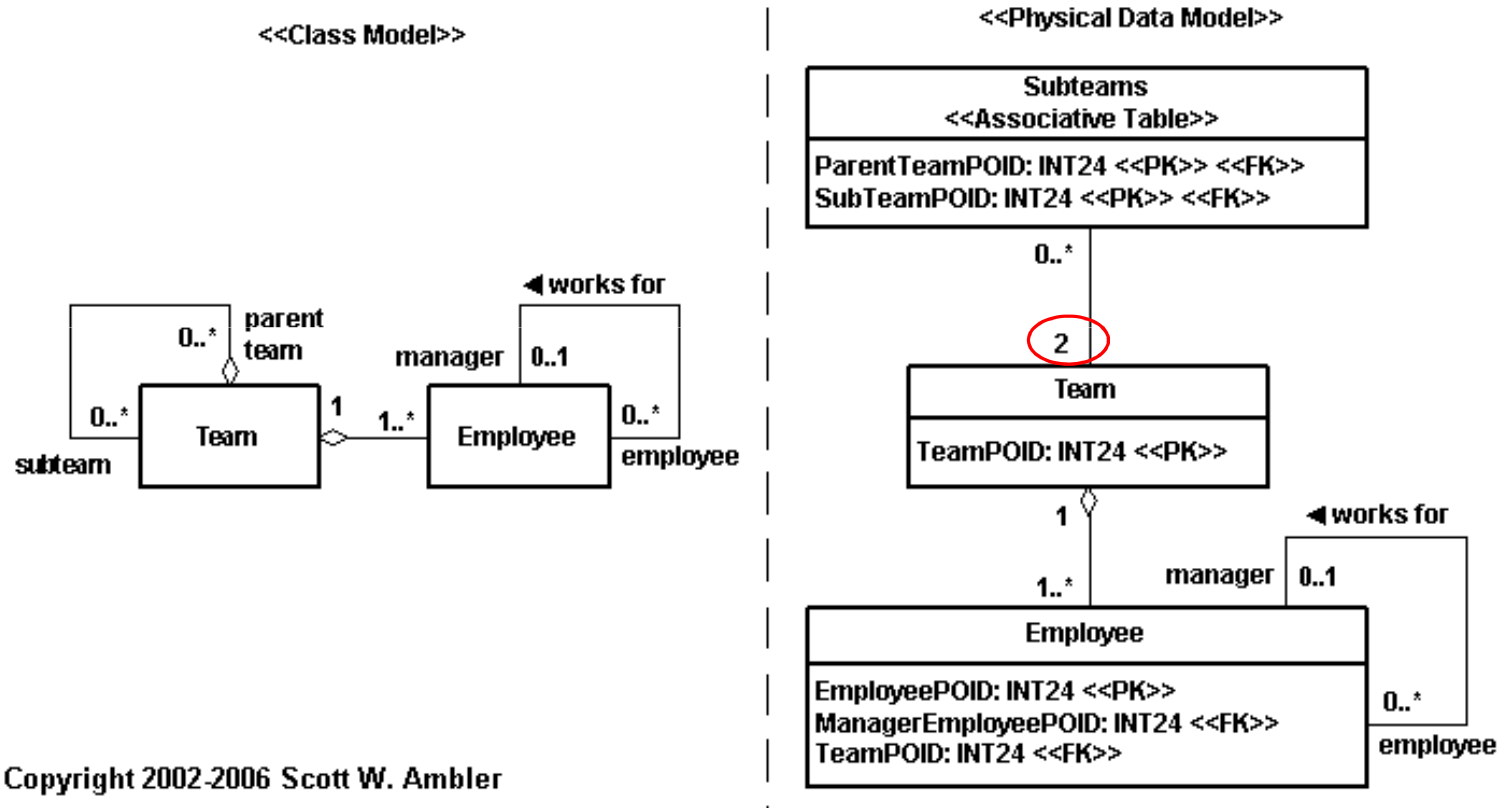


Egy – több kapcsolat is lehetne az adatréteg alapján

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>>
parent
team
0..*
works for
manager
0..1
0..*
Team
1
1..*
Employee
0..*
subteam
employee
Copyright 2002-2006 Scott W. Ambler
<<Physical Data Model>>
Subteams
<<Associative Table>>
ParentTeamPOID: INT24 <<PK>> <<FK>>
SubTeamPOID: INT24 <<PK>> <<FK>>
0..*
2
Team
TeamPOID: INT24 <<PK>>
1
1..*
works for
manager
0..1
Employee
EmployeePOID: INT24 <<PK>>
ManagerEmployeePOID: INT24 <<FK>>
TeamPOID: INT24 <<FK>>
0..*
employee

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
- PL:
  - Következő számla sorszám
- Osztály szintű konstansok kezelése is hasonló

Osztály szintű tulajdonságok – 2

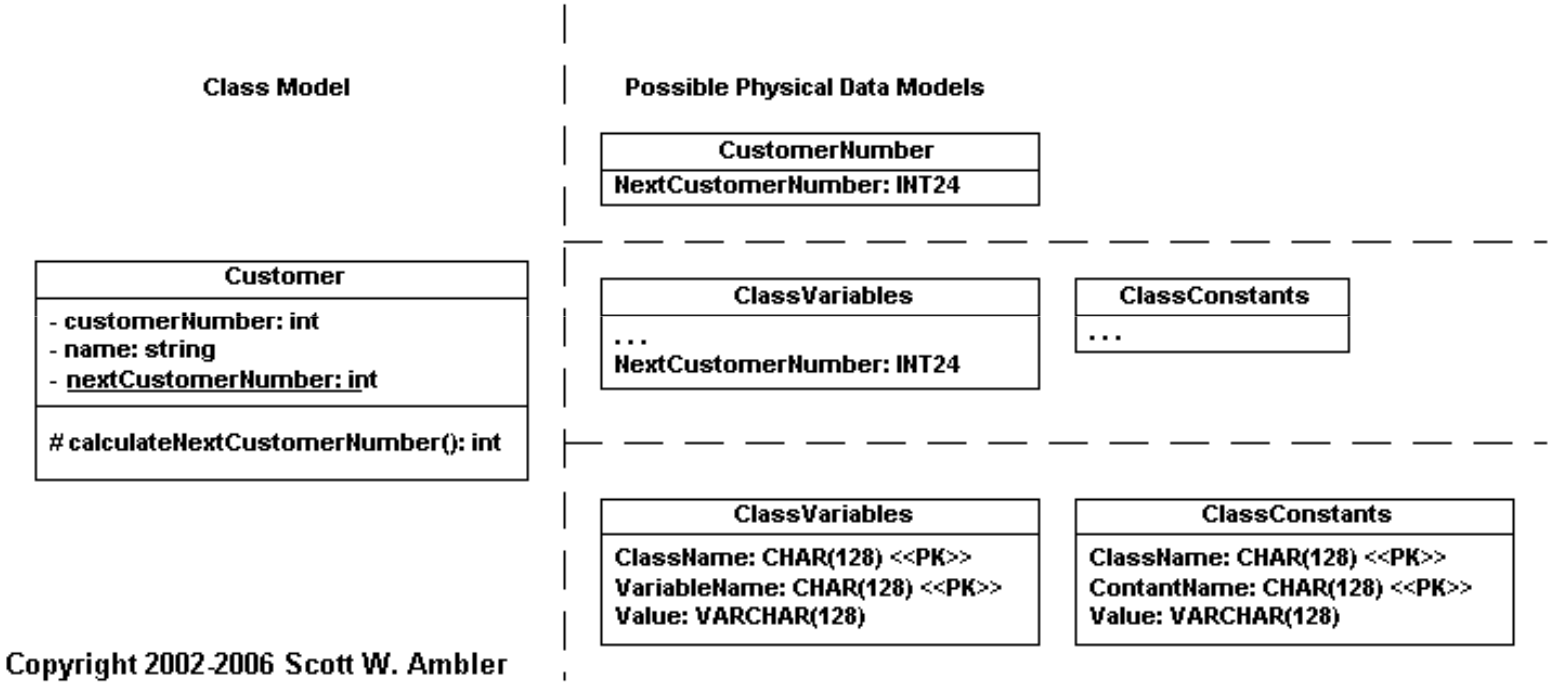
- Minden tulajdonságnak külön tábla
  - Gyors
  - Sok kicsi tábla → áttekinthetetlen adatmodell
- Minden tulajdonság ugyanabban a táblában különböző oszlopokban
  - Gyors
  - Egyszerű (egy tábla az összesnek)
  - Konkurencia (mivel sor alapú és nem oszlop)
- Osztályonként egy tábla az értékek különböző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



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

Demó

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();
```

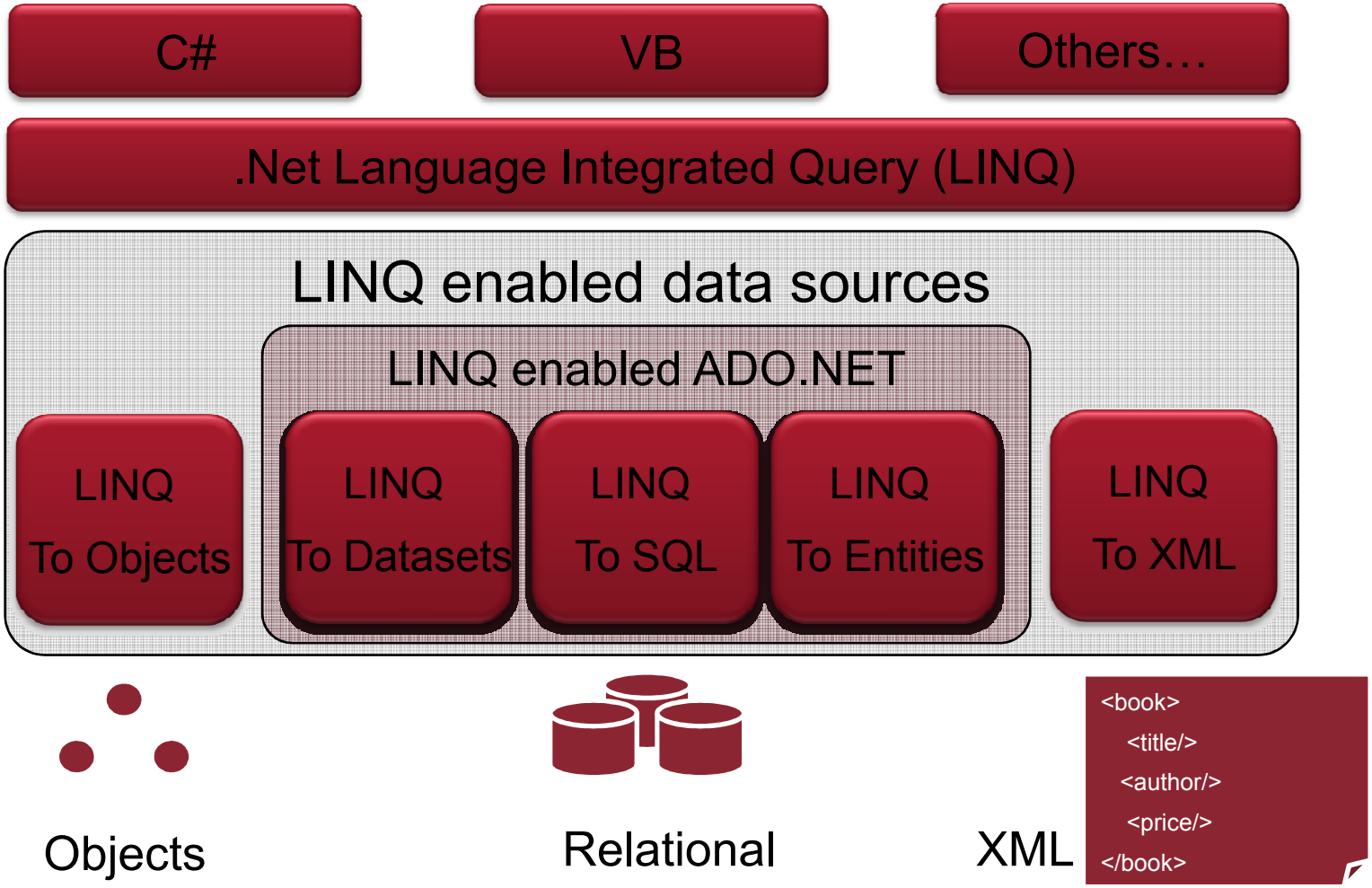
Előnyök

- 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

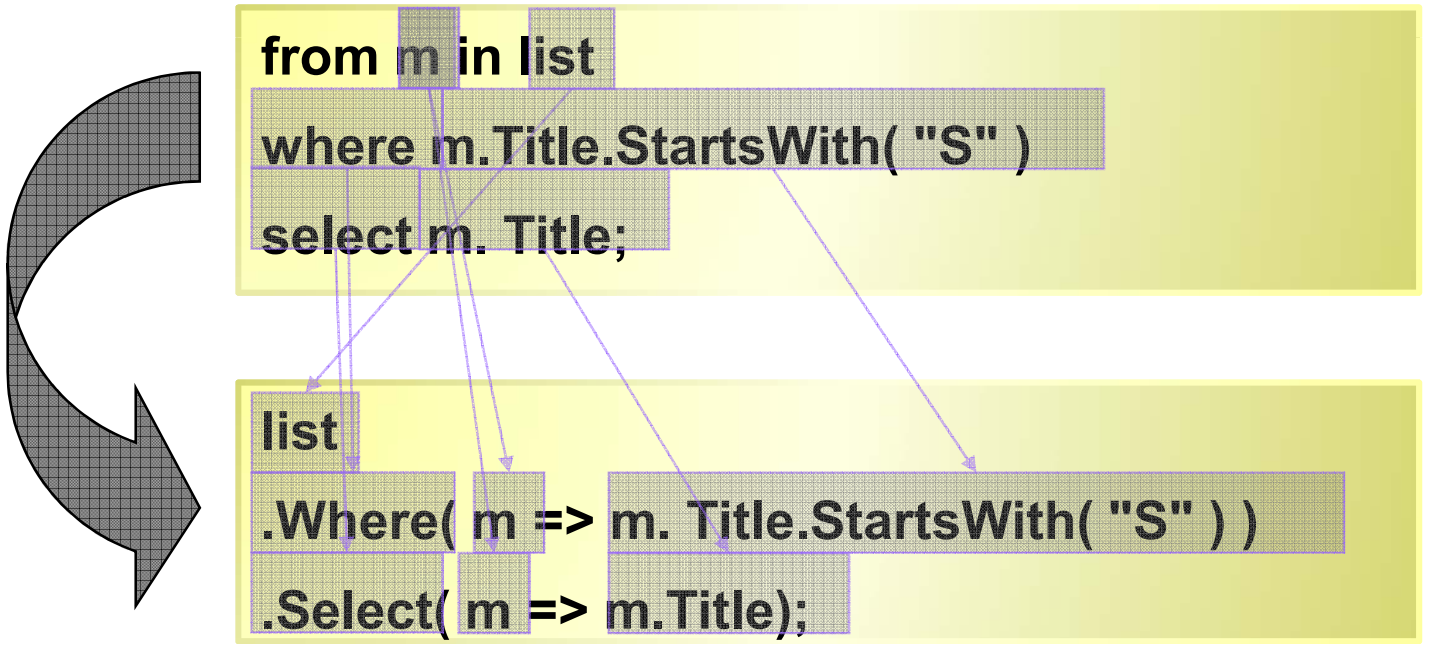


LinQ to Objects használata

- 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

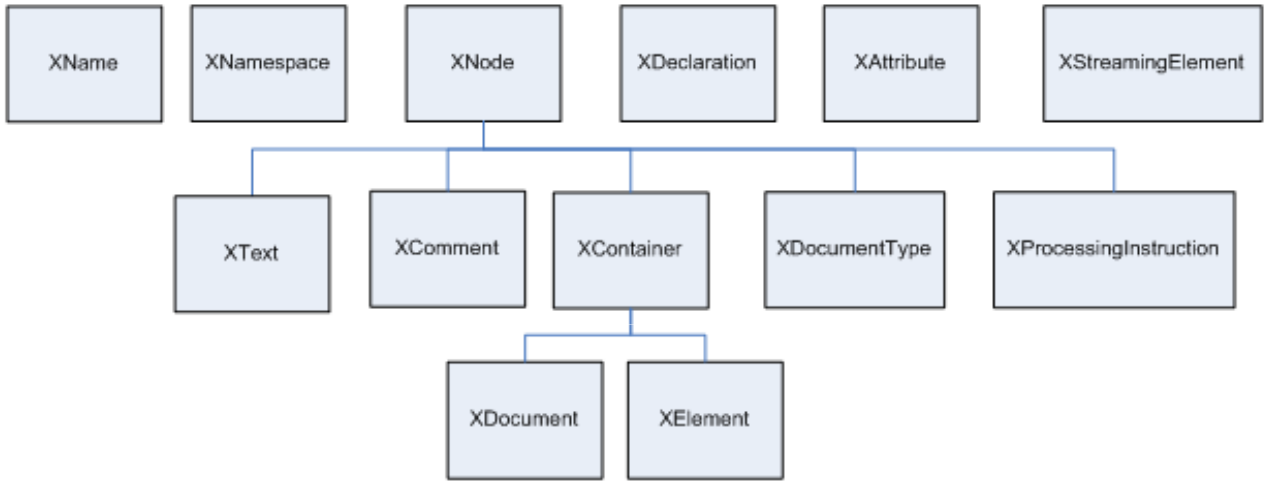
- XML mindenhol megjelent
  - Konfigurációs állományok
  - Dokumentumok
  - Platformfüggetlen kommunikáció
  - Hierarchikus adatok
  - Web szolgáltatások
  - …

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>2009-05-21 15:12:30</pubDate>
      <guid>11e7e86b-1ad6-465a-8f74-cf00be28aaba</guid>
    </item>
  </channel>
</rss>
```

Xml lekérdezés C#

```
// Lekérdezés összeállítása 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
```

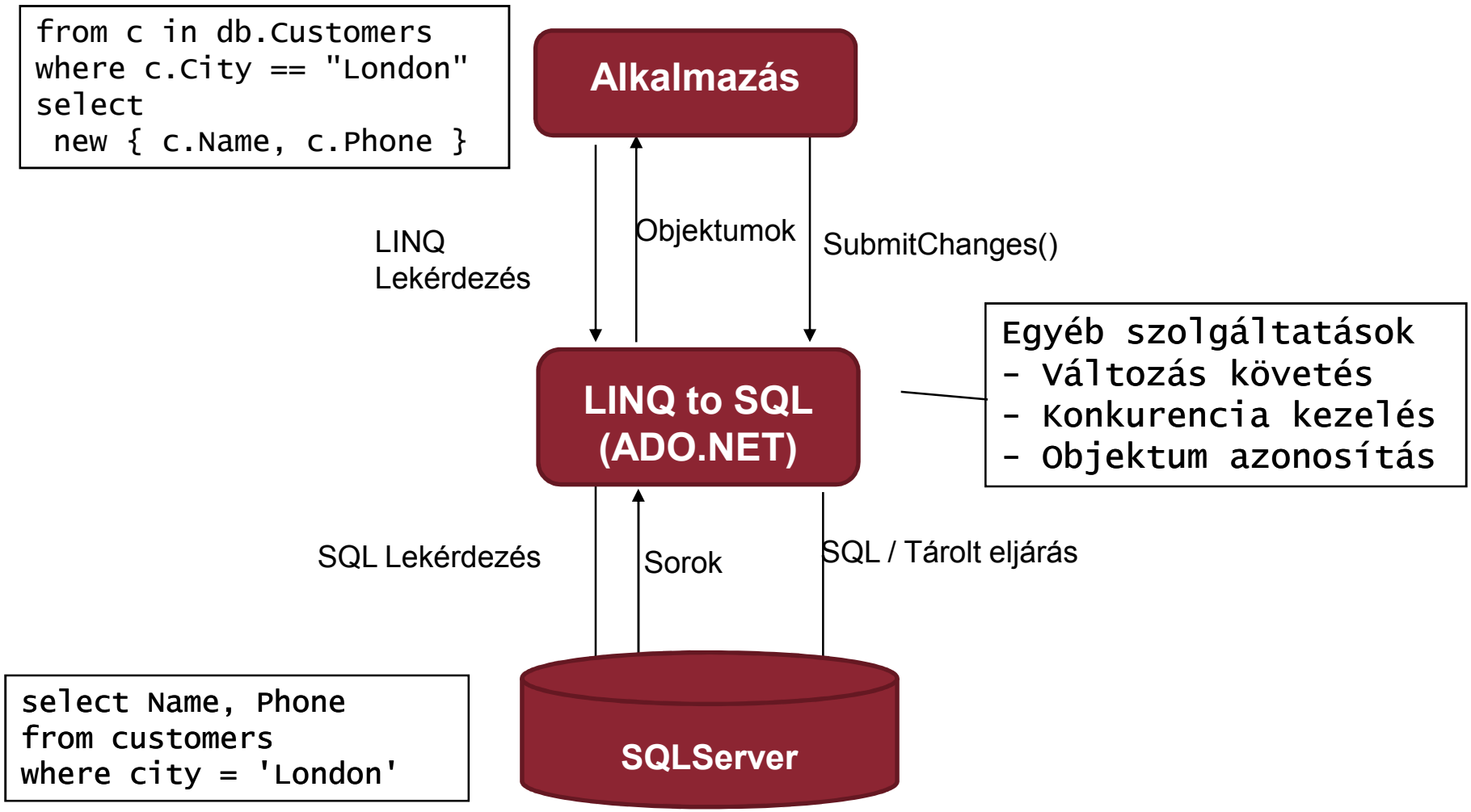
Demó

Avagy ki szerelmesebb Rómeó vagy Júlia?

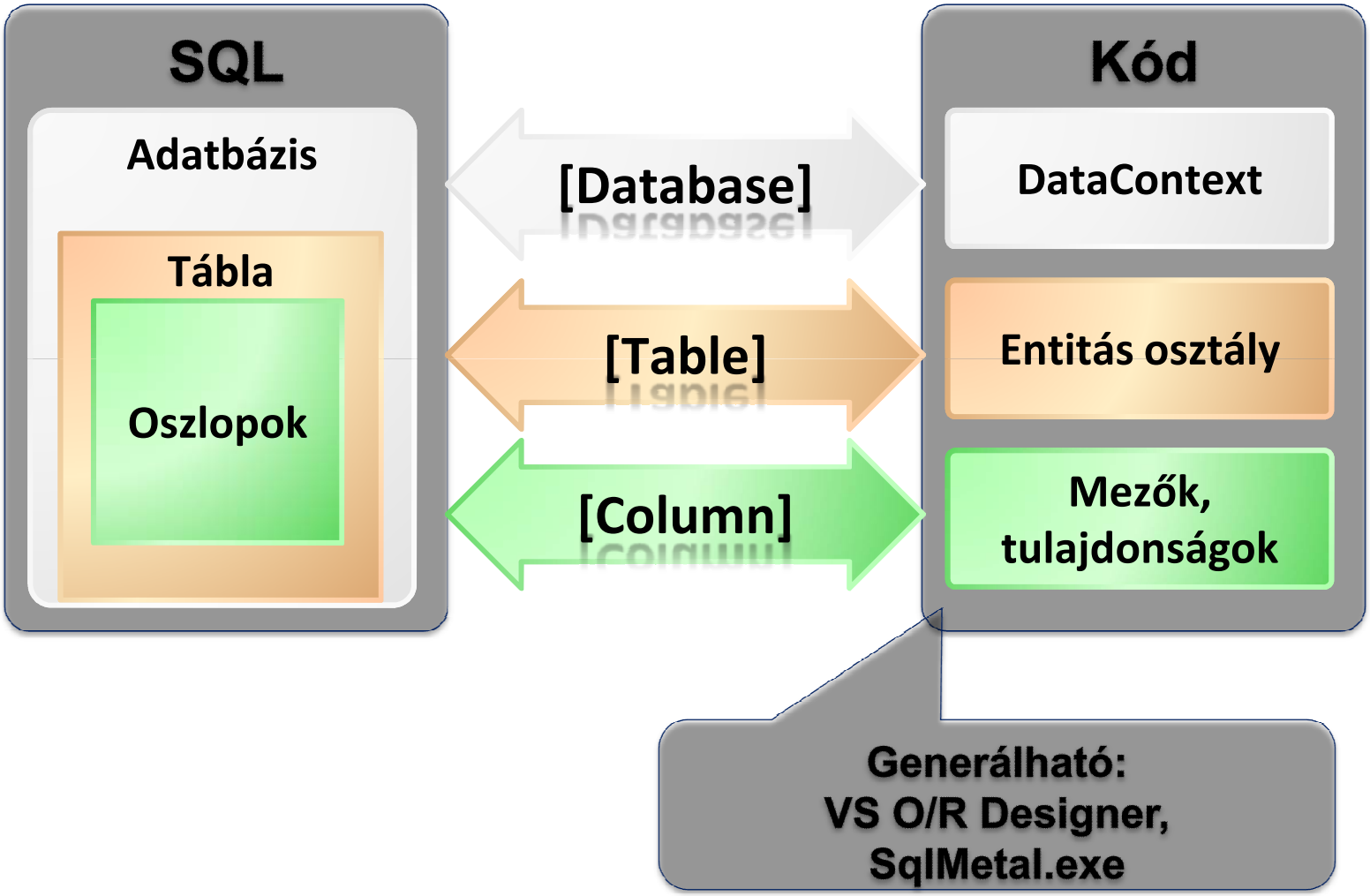
LinQ to SQL

- Relációs adatbázisok és OO világ összekötése
- Egyfajta ORM
- Csak MS SQL Szerverrel működik
- Más platformok
  - Elvileg van LinQ to Dataset (de nincs)
  - Entity Framework (még nincs kész)

LINQ to SQL Architektúra



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

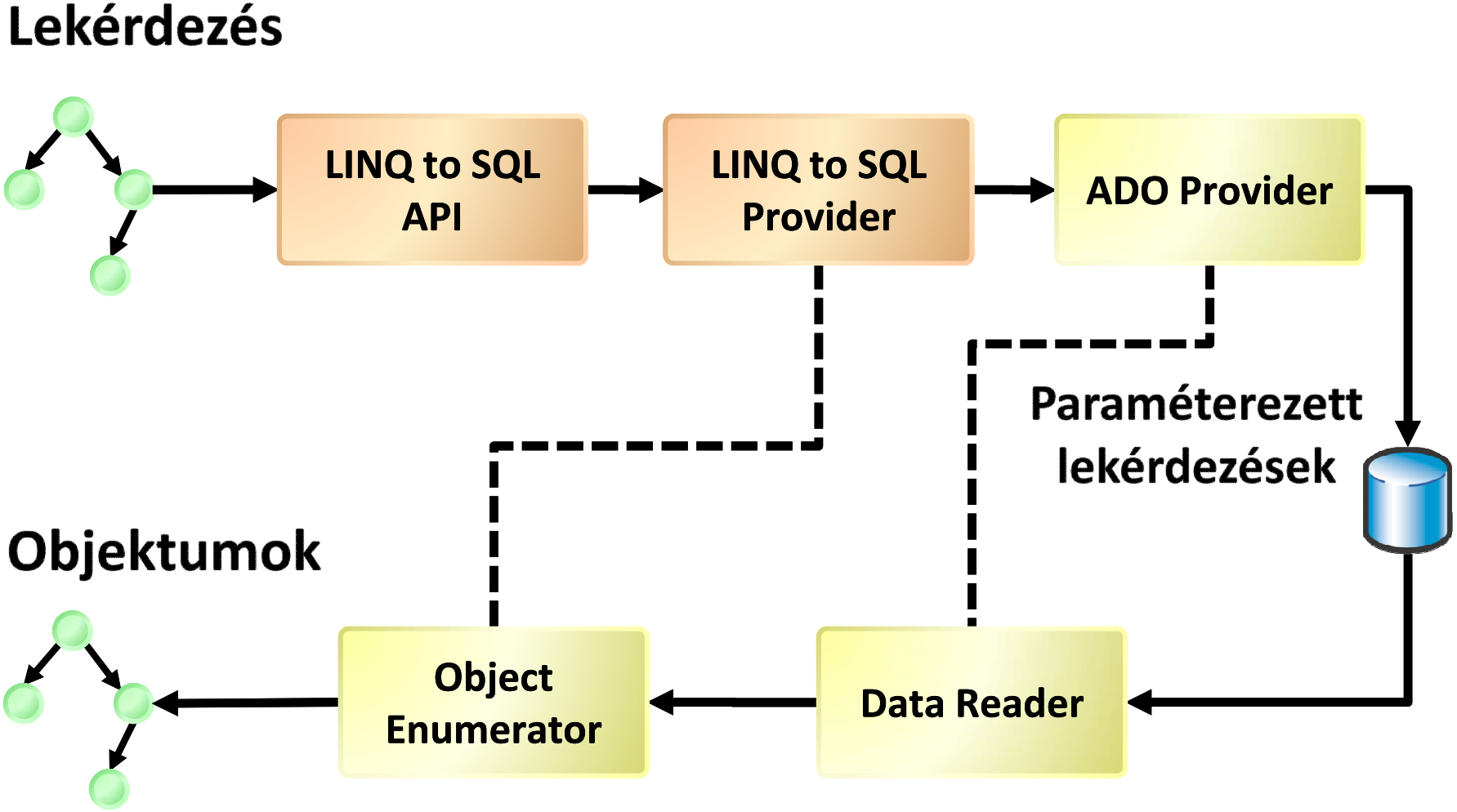
```
public partial class Northwind : DataContext
{
    public Table<Customer> Customers;
    public Table<Order> Orders;
    public Northwind (string connection):
  base(connection) {}
}
...
Northwind ndc = new Northwind();
var q2 = from c in ndc.Customers
         where c.Orders.Count > 15
         select c;
```

Lekérdezések

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

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

Lekérdezések végrehajtása



Vetítés

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

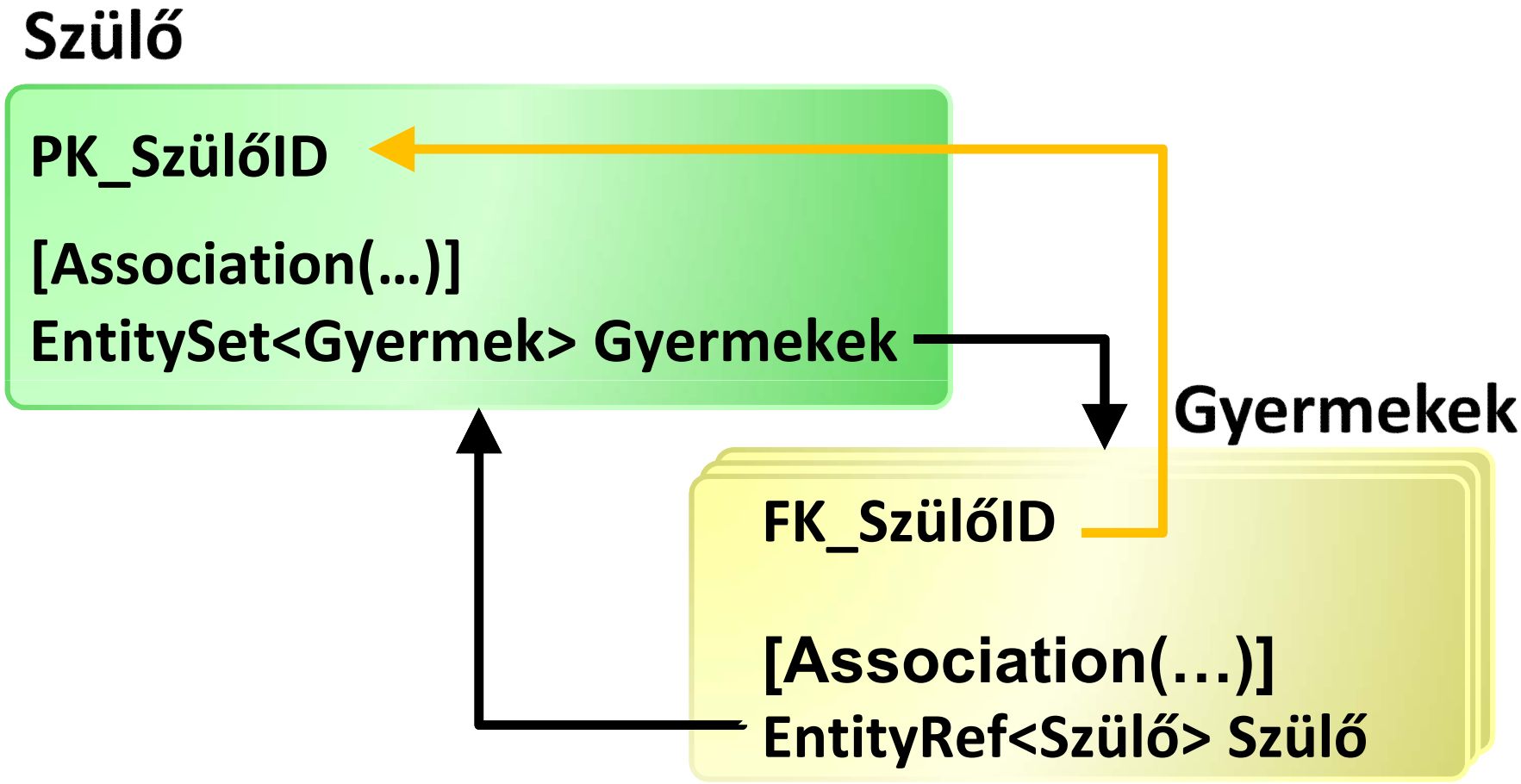
```
var q = from c in dc.Categories
        where c.Products.Count > 5
        select new {
                     Name = c.CategoryName,
                     Count = c.Products.Count
                   };
```

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



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

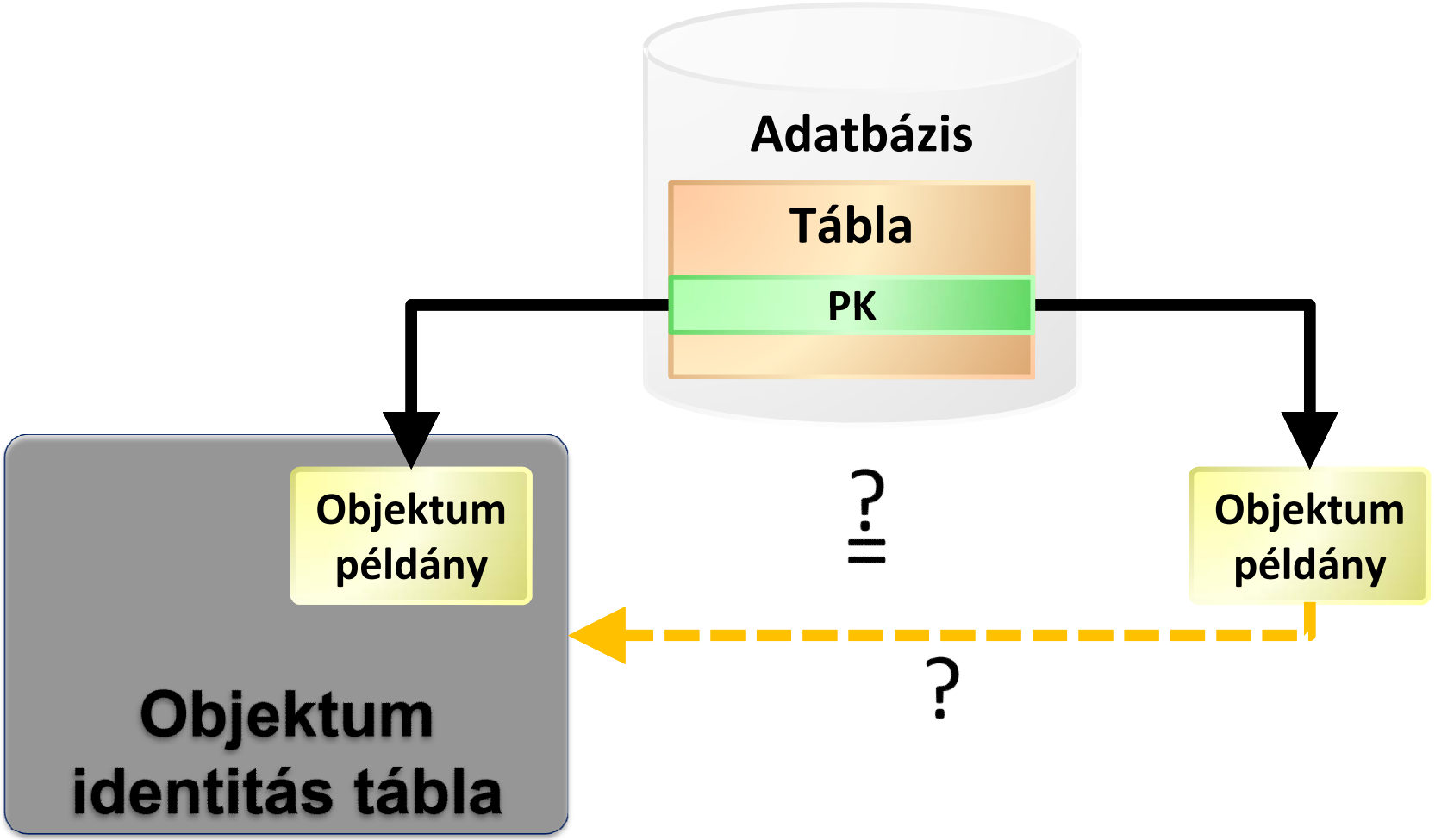
```
var q =     from s in db.Suppliers
      join c in db.Customers on s.City equals c.City
  into custs
      orderby custs.Count() descending
      where custs.Count() > 0
      select  new
            {
                  Supplier = s,
                  Customers = custs
            };
```

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

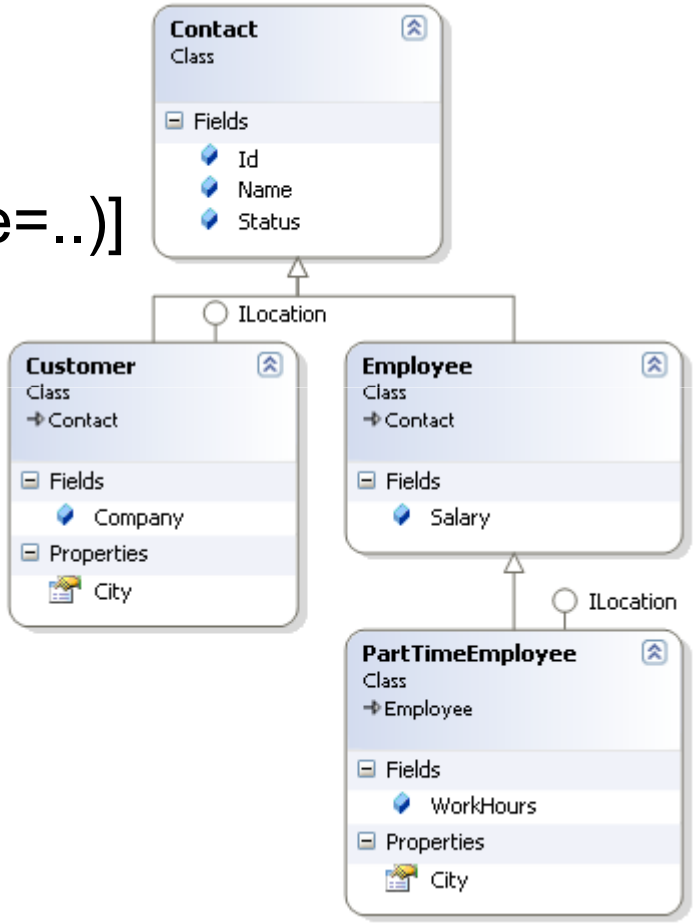


Entitások több rétegben

- 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, OverwiteCurrentValues

Tranzakció

- Automatikusan read commited
  - 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();
```

Érdekesség – 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

Windows Presentation Foundation

Tartalom

- Bevezetés
- Architektúra
- XAML
- WPF belülről
- WPF vezérlők
- Méretezés, pozícionálás, transzformálás
- Panelek
- Adatkötés
- Erőforrások
- Style, Control template

Bevezetés

- Microsoft .NET Framework technológia
- 3.0 verzióban jelenik meg (jelenleg 3.5-nél tartunk)
- Esemény vezérelt megjelenítési réteg
- A Windows Forms utódjának tekinthető
  - vs. deklaratív gondolkodásmód
  - Néhány esetben vissza kell nyúlnunk a Windows Forms-hoz

Bevezetés - elvek

- Teljeskörű integráció
  - 2D, 3D, speech, audiovizuális vezérlők
- Felbontás függetlenség
  - „eszközfüggetlen pixel”
- Hardveres gyorsítás
  - Direct 3D
  - Szoftveres render pipeline fallback
  - Windows Vista

Bevezetés – elvek (2)

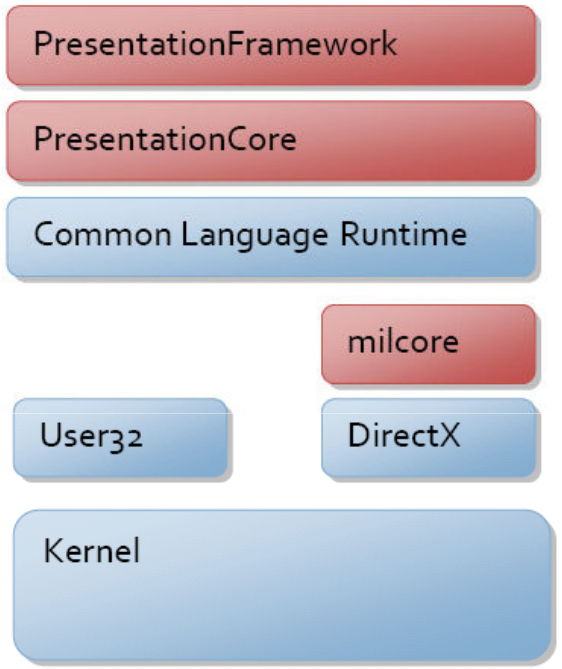
- Deklaratív szemlélet
  - vs Windows Forms
  - XAML
  - Adatkötés általánossá válik
- Testreszabhatóság
  - Egymásba ágyazható vezérlők
  - Templatek, skinek stb
- Egyszerű telepítés
  - Browser, ClickOnce, Windows Installer

Tartalom

- Bevezetés
- ***Architektúra***
- XAML
- WPF belülről
- WPF vezérlők
- Méretezés, pozícionálás, transzformálás
- Panelek
- Adatkötés
- Erőforrások
- Style, Control template

Architektúra - felépítés

- WPF
  - PresentationFramework
  - PresentationCore
  - Milcore
    - Integráció DirectX-szel
    - Összes képernyőművelet
    - Natív =>gyors (követelmény)



Architektúra – osztályok

- System.Windows.DependencyObject
  - Propertyk dominanciája (vs metódus/esemény)
  - Változásértesítés
    - *Dependency Property*
    - vs. *INotifyPropertyChange*
    - *Automatikus validáció*
  - Függőségek deklarálása (adatkötés)
  - *Attached Property*
    - Pl. *Grid – Row, Column*

Architektúra – osztályok (2)

- System.Windows.UIElement
  - Layout kezelés
    - vs Windows Forms – *Dock*, *Anchor*
    - Measure, Arrange fázis
  - Esemény kezelés
    - „*Előnézeti*” fázis (tunneling)
    - „*Tényleges*” fázis (bubbling)
  - *CommandBinding*
    - *Command end points*
    - *ICommand*

Architektúra – osztályok (3)

- System.Windows.FrameworkElement
  - Propertyk tartalmazott elemek egységes kezeléséhez
    - *HorizontalAlignment, VerticalAlignment*
    - *Margin*
    - *…*
  - Animáció készítés
    - *Storyboard*
  - *Adatkötés (DataTemplate)*

Architektúra – osztályok (4)

- System.Windows.FrameworkElement (2)
  - Stílusok
    - Property-k kötése közös halmazhoz
    - Explicit, implicit
- System.Windows.Controls.Control
  - Templatezhető
    - ControlTemplate
    - Script szerűen felüldefiniálható a Visual Tree
  - Három részből áll:
    - Adatmodell (property-k)
    - Interakciós modell (command, event)
    - Megjelenítési modell (template)

Tartalom

- Bevezetés
- Architektúra
- ***XAML***
- WPF belülről
- WPF vezérlők
- Méretezés, pozícionálás, transzformálás
- Panelek
- Adatkötés
- Erőforrások
- Style, Control template

XAML

- eXtensible Application Markup Language
- XML alapú (deklaratív) nyelv
- .NET objektumok példányosítása
- Specifikáció: kulcsszavak, compiler direktívák
  - Framework nélkül nem használható! (vs. HTML, SVG)

XAML – Alapok

```
<Button
xmlns="http://schemas.microsoft.com/winfx/
2006/xaml/presentation" Content="OK"/>
```

- Böngészőben futtatható ! (.xaml)
- C#-ban ugyanez:

```
System.Windows.Controls.Button b =
new System.Windows.Controls.Button();
b.Content = "OK";
```

XAML – Alapok (2)

- Példányosítás az osztálynak megfelelő taggel
- Propertyk elérése attribútumokon keresztül
- Események elérése szintén attribútummal

```
<Button
xmlns="http://schemas.microsoft.com/winfx/
2006/xaml/presentation"
Content="OK" Click="button_Click"/>
```

- Ehhez már szükséges codebehind fájl

XAML – Alapok (3)

- Csak a default konstruktor érhető így el
- Először az eventek majd a propertyk inicializálódnak
  - Esetleges események property használatkor már „éljenek”

XAML – Névterek

- XML névtér ~ .NET névtér
- `xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"`
  - **System.Windows, System.Windows.Control, … (17 db)**
- root névtér: `xmlns=`
  - Gyökér és gyermek elemek
- További névterek:
  - Egyedi prefix kell
  - `xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"`
    - `x:Key`

XAML – Property element syntax

- Attribute element syntax
  - Értékadás attribútumon keresztül
  - Összetett objektummal nem működik
- Property element syntax

```
<Button>
  <Buttton.ContextMenu>
   <ContextMenu>
         <MenuItem Header="1"> First item </MenuItem>
         <MenuItem Header="2"> First item </MenuItem>
   <ContextMenu>
 <Buttton.ContextMenu>
</Button>
```

  - `<Buttton.ContextMenu>` : property és nem példányosítás

XAML – Típus konverterek

- Példa:
  - `<Rectangle Height="40" Width="40" Fill="Black"/>`
- „Black” : string ⇔ *System.Windows.Media.Brush* ?
- Megoldás : típus konverterek
- A XAML feldolgozó automatikusan keres egy regisztrált konvertert
  - *BrushConverter, ColorConverter*

XAML – Típus konverterek (2)

- Mi is készíthetünk
  - *TypeConverter* leszármazott
  - *TypeConverterAttribute* a regisztráláshoz
- *BrushConverter* nélkül

```
<Button.Background>
  <SolidColorBrush Color="White"/>
</Button.Background>
```

- *ColorConverter* nélkül
  - `<Color A="255" R="255" G="255" B="255"/>`

XAML – Típus konverterek (3)

- *ByteConverter* nélkül már nem megy
- C#-ban is használható

```
System.Windows.Controls.Button b =
    new System.Windows.Controls.Button();

b.Background = (Brush)System.ComponentModel.
    TypeDescriptor.GetConverter(typeof(Brush)).
    ConvertFromInvariantString("White");
```

XAML – Markup extension

- Stringet alakít át .NET objektummá
  - Hasonlít a típus konverterre
- Általában futás időben
  - Néha hatékonysági okokból fordítás közben
- Explicit kell meghívni (vs. Konverter)
- { } jelek között lévő attribútum érték
  - vs. „{}{Így már nem markup extension}”

XAML – Markup extension (2)

- Példa

```
<Button
Background="{x:Null}"
Height="{x:Static SystemParameters.IconHeight}"
Content="{Binding Path=Height,
RelativeSource={RelativeSource Self}}"/>
```

  - *null*, statikus és dinamikus kötés

- *Null**Extension**, Static**Extension**, Binding*
  - *Általában Extension*-re végződik
  - Tag-ként is használhatók
  - Path – már létező objektum propertyjei

XAML – Objektumok gyermekei

- Háromféle gyerek elem (nem XML!)
  - Content property
  - Kollekció elemei
  - Konvertálható tartalom
- Nem: Property element syntax

```
<Button>
 <Buttton.ContextMenu>
   <ContextMenu>
         <MenuItem Header="1"> First item </MenuItem>
         <MenuItem Header="2"> First item </MenuItem>
   <ContextMenu>
 <Buttton.ContextMenu>
</Button>
```

XAML – Objektumok gyermekei (2)

- Content property
  - *ContentControl*
  - Az XML tag gyermekeleme lesz az értéke
  - Általában Content a neve
- Példa:

```
<Button>
 <Rectangle Height="40" Width="40" Fill="Black"/>
</Button>
```

(hivatkozhatunk hosszan is : `<Button.Content>`)

XAML – Objektumok gyermekei (3)

- Kollekció elemei
  - Listák (minden, ami *IList*)

```
<ListBox>
  <ListBox.Items>
    <ListBoxItem Content="Item 1"/>
    <ListBoxItem Content="Item 2"/>
  </ListBox.Items>
</ListBox>
```

  - `<ListBox.Items>` el is hagyható (*Content* property)

XAML – Objektumok gyermekei (4)

- Szótárak (minden, ami *IDictionary*)

```
<ResourceDictionary>
  <Color x:Key="1" A="1" R="1" G="1" B="1"/>
  <Color x:Key="2" A="0" R="0" G="0" B="0"/>
</ResourceDictionary>
```

- Kulcs : `x:Key`
  - Nem property, speciális attribútum
  - A másodlagos névtérben definiált

XAML – Objektumok gyermekei (5)

- Konvertálható tartalom
  - Példa:

```
<SolidColorBrush Color="White"/>
```

ami alapján:

```
<SolidColorBrush>White</SolidColorBrush>
```

    - Fontos: `Color` nem Content property!
    - De: létezik *string* => `SolidColorBrush` converter
      - White, white, #FFFFFF
  - Érdekesség:

```
<Brush>White</Brush>
```

    - ! `Brush` absztrakt, de létezik konverter (*SolidColorBrush*)

XAML – Feldolgozás

- Betöltés és feldolgozás futási időben

```
Window window = null;
using (FileStream fs =  new FileStream(
 "MyWindow.xaml", FileMode.Open, FileAccess.Read))
{
  window = (Window)XamlReader.Load(fs);
}
```

  - A root elemre kapunk referenciát
  - A gyermek elemek név szerint érhetők el:
    - `window.FindName("okButton");`
    - **`FindName : FrameworkElement`** metódusa
    - Név: **`x:Key`** (**`x`** : a szokásos névtér)

XAML – Feldolgozás (2)

- Codebehind fájl

```
<Window xmlns="..." xmlns:x="..."
        x:Class="MyNamespace.MyWindow">
...
</Window>
```

  - Összerendelés `x:Class` kulcsszó segítségével
  - C#, VB : Parciális osztályok, *InitializeComponent*
  - C++ : öröklléssel
    - `x:SubClass`

XAML – Feldolgozás (3)

- BAML
  - Binary Application Markup Language
  - XAML bináris reprezentációja
  - Optimális (vs. XML)
  - Beágyazott erőforrás
  - Nincs köze az IL kódhoz, nincs benne procedurális kód

XAML – Feldolgozás (4)

- g.cs (g.vb) fájlok
  - g = generált
  - Ragasztó (glue) kódot tartalmazza
  - *x:Class* – szal megadott osztály implementációja
    - BAML betöltése
    - Változó – erőforrásobjektum összerendelés
    - XAML-ben leírt eseményfeliratkozások
    - *InitializeComponent*( )

XAML – Egyebek

- Tetszőleges .NET osztály elérése

```
<Window
  xmlns:sys="clr-namespace:System;assembly=mscorlib"
  ...
```

  - `clr-namespace` prefix
  - Mostantól a `sys` xml-névtéren keresztül érhető el
  - assembly elhagyható (aktuális assembly)

XAML – Egyebek (2)

- Procedurális kód XAML-ben
  - **KERÜLJÜK !**
  - a .g.x fájlba másolódik

```
<Window xmlns="..." xmlns:x="...">
  <Button Click="button_Click">OK</Button>
  <x:Code>
   <![CDATA[
   void button_Click( object sender,
                      RoutedEventArgs e) {
     this.Close();
   }
   ]]>
  </x:Code>
</Window>
```

Tartalom

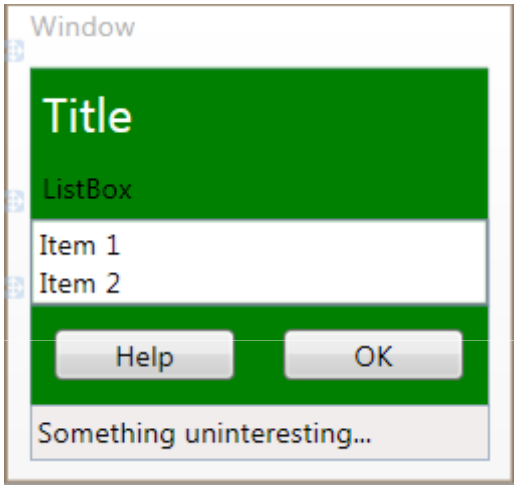
- Bevezetés
- Architektúra
- XAML
- ***WPF belülről***
- WPF vezérlők
- Méretezés, pozícionálás, transzformálás
- Panelek
- Adatkötés
- Erőforrások
- Style, Control template

WPF belülről – Logical Tree

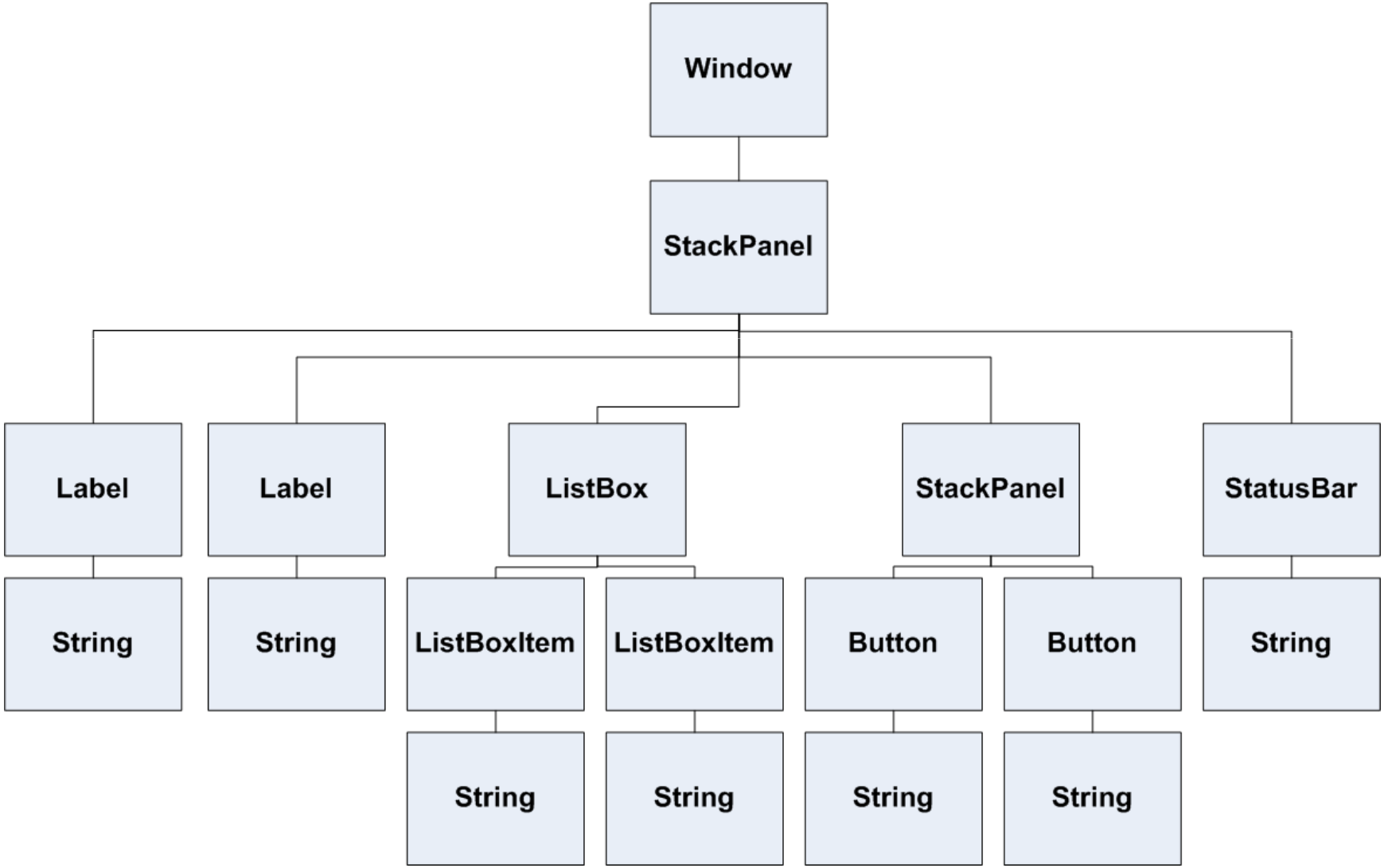
- Hierarchikus felépítésű felületek, vezérlők
- => Logical Tree
  - Nem csak XAML használatkor
  - Esemény áramlás iránya
  - Property érték propagálás
  - *System.Windows.LogicalTreeHelper*

WPF belülről – Logical Tree (2)

```
<Window xmlns="" Title="Window" Background="Green">
  <StackPanel>
    <Label FontSize="20"
      Foreground="White">Title</Label>
    <Label>ListBox</Label>
    <ListBox>
          <ListBoxItem>Item 1</ListBoxItem>
          <ListBoxItem>Item 2</ListBoxItem>
    </ListBox>
    <StackPanel Orientation="Horizontal"
       HorizontalAlignment="Center">
      <Button MinWidth="75" Margin="10">Help</Button>
      <Button MinWidth="75" Margin="10">OK</Button>
    </StackPanel>
    <StatusBar>Something uninteresting...</StatusBar>
  </StackPanel>
</Window>
```



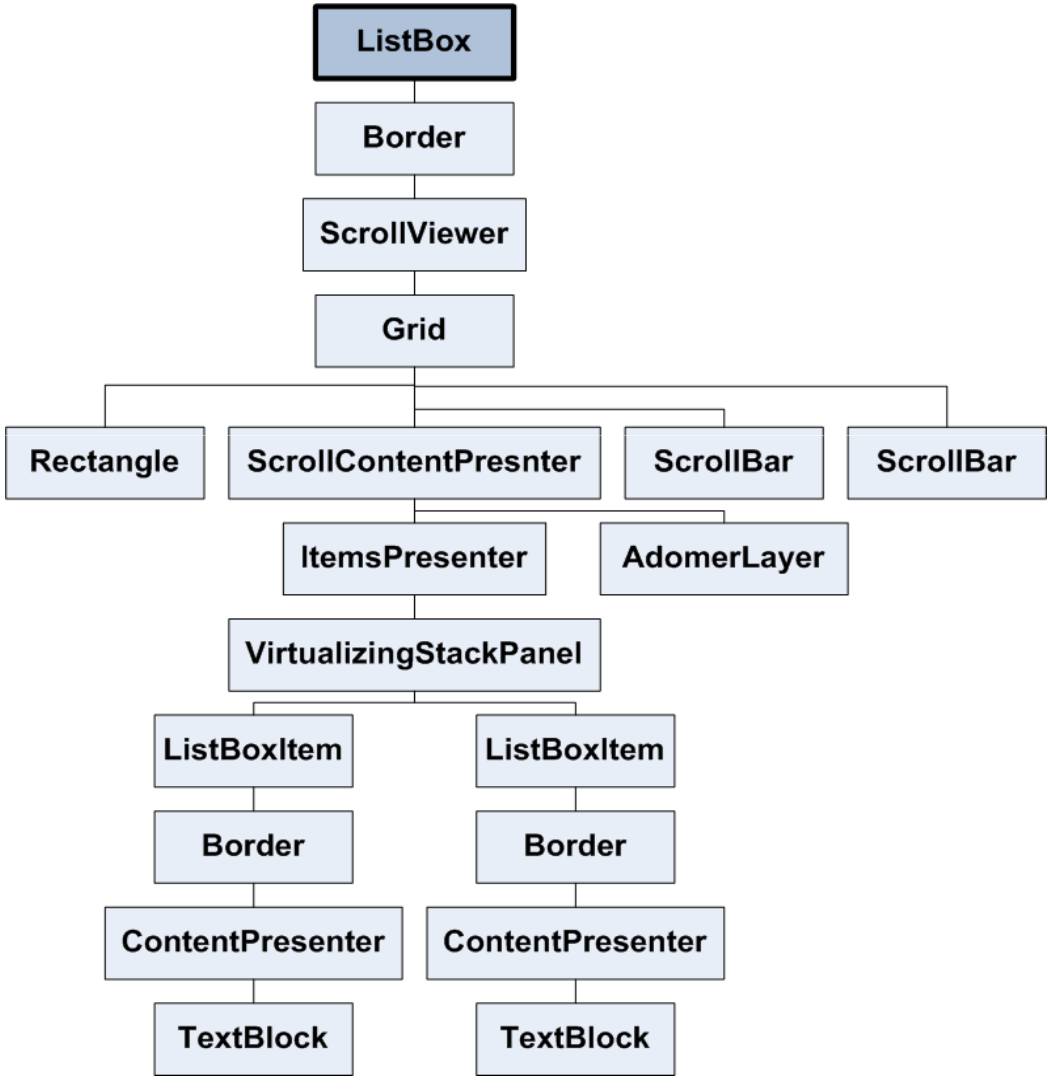
WPF belülről – Logical Tree (3)



WPF belülről – Visual Tree

- Csomópontok helyett az elemi *megjeleníthető* objektumok
- pl ListBox => ScrollBar, Border …
- csak *System.Windows.Media.Visual* és *System.Windows.Media.Visual3D*
- *System.Windows.Media.VisualTreeHelper*

WPF belülről – Visual Tree (2)



WPF … – Dependency Property

- *System.Windows.DependencyObject*
- *System.Windows.DependencyProperty*
- Változás értesítés automatikus
- Property érték öröklés (értékek áramlása)

WPF … - Dependency Property (2)

```
public class Button : ButtonBase
{
  public static readonly DependencyProperty IsDefaultProperty;
    static Button() {
      Button.IsDefaultProperty = DependencyProperty.Register(
       "IsDefault", typeof(bool), typeof(Button),
       new FrameworkPropertyMetadata(false,
         new PropertyChangedCallback(OnIsDefaultChanged)));
      …
    }
    public bool IsDefault {
      get { return (bool)GetValue(Button.IsDefaultProperty);}
      set { SetValue(Button.IsDefaultProperty, value);}
    }
    private static void OnIsDefaultChanged(
        DependencyObject o,
        DependencyPropertyChangedEventArgs e) { … }
    …
}
```

WPF … - Dependency Property (3)

- Konvenció:

```
public static readonly DependencyProperty
   xxxxProperty;
```

- *DependencyProperty.Register()*
- Wrapper property és callback opcionális
- XAML : wrapper property szükséges, de a fordító kioptimalizálja (!)
- *DependencyObject*
  - *GetValue( ), SetValue( )*

WPF … - Dependency Property (4)

- Változás értesítés
  - *PropertyChangedCallback*
  - *ValidateValueCallback*
  - *CoerceValueCallback*
  - XAML : Property triggerek

```
<Trigger Property="IsMouseOver" Value="True">
  <Setter Property="Foreground" Value="Blue"/>
</Trigger>
```

WPF … - Dependency Property (5)

- Property érték öröklés
  - Property érték áramlás *lefelé* az objektum hierarchián
  - *DependencyProperty.Register*
    - *FrameworkPropertyMetadataOptions.Inherits*
  - Újbóli beállítással megszakítható

WPF … - Attached Property

- Egyfajta *DependencyProperty*
- Nem csak a regisztráló osztály használhatja
- *DependencyProperty.RegisterAttached*

```
<StackPanel TextElement.FontSize="30"
  TextElement.FontStyle="Italic">
  <Button>Cancel</Button>
  <Button>OK</Button>
</StackPanel>
```

(öröklés is)

WPF … - Attached Property (2)

- De (!) *Button* nem *TextElement* leszármazott
  - Viszont a property ugyanaz:

```
Control.FontSizeProperty =
  TextElement.FontSizeProperty.AddOwner(
  typeof(Control), new FrameworkPropertyMetadata(
   SystemFonts.MessageFontSize,
   FrameworkPropertyMetadataOptions.Inherits));
```

WPF … - Attached Property (2)

- XAML : szükséges a megfelelő Get és Set metódus

```
public static void SetFontSize(DependencyObject element,
  double value) {
    element.SetValue(TextElement.FontSizeProperty, value);
}
public static double GetFontSize(DependencyObject element) {
  return
  (double)element.GetValue(TextElement.FontSizeProperty);
}
```

- Általánosan használt:
  - Pl. konténerek (Grid, Canvas, DockPanel …)

WPF belülről – Routed event

- A tartalmazási hierarchia mentén továbbítható események
- Pl. *Button* : *Click*
  - *TextBlock/ButtonChrome*
- Routing stratégia
  - Bubbling, tunneling, direct
- *UIElement*
  - *AddHandler, RemoveHandler, RaiseEvent*

WPF belülről – Routed event (2)

```
public class Button : ButtonBase
{
  public static readonly RoutedEvent ClickEvent;
  static Button(){
    Button.ClickEvent =
      EventManager.RegisterRoutedEvent("Click",
        RoutingStrategy.Bubble,
        typeof(RoutedEventHandler),typeof(Button));
        …
  }
  public event RoutedEventHandler Click {
    add { AddHandler(Button.ClickEvent, value); }
    remove { RemoveHandler(Button.ClickEvent, value);
  }
}
```

WPF belülről – Routed event (3)

- *RoutedEventHandler* delegate
  - *object sender*
  - *RoutedEventArgs*
    - *Source* (Logical tree)
    - *OriginalSource* (Visual tree)
    - *Handled*
      - *AddHandler, handledEventsToo = true*

WPF belülről – Attached event

- ~ Attached Property

```
<Window xmlns="... " xmlns:x="..." x:Class="xxxx"
ListBox.SelectionChanged="ListBox_SelCh">
```

- => az ablakban található valamennyi listbox eseményét megkapja

Tartalom

- Bevezetés
- Architektúra
- XAML
- WPF belülről
- ***WPF vezérlők***
- Méretezés, pozícionálás, transzformálás
- Panelek
- Adatkötés
- Erőforrások
- Style, Control template

WPF vezérlők

- Content Control
- Items Control
- Range Control
- Text Control

WPF vezérlők – Content control

- *System.Windows.Controls.ContentControl*
- Másik elemet tartalmazhatnak
- *object Content* property
- Max. 1 közvetlen gyerek
- Gombok, egyszerű konténerek, konténerek fejléccel

WPF vezérlők – Content Control (2)

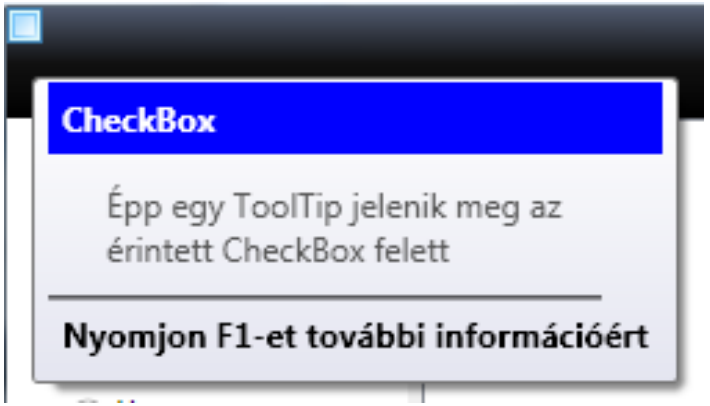
- Gombok
  - *Button*
  - *ToggleButton*
    - *IsChecked*
  - *CheckBox*
    - Testreszabott *ToggleButton*
  - *RadioButton*
    - *GroupName*, közös logical root

WPF vezérlők – Content Control (3)

- Egyszerű konténerek
  - *Label*
    - Tetszőleges tartalom
  - *Frame*
    - „elszigeteli” a tartalmát (pl öröklődő propertyk)
    - Tetszőleges HTML/XAML tartalom
    - Win32 API alapú renderelés
  - *ToolTip*
    - Nem állhat közvetlenül a tartalmazási fában
    - *FrameworkElement.ToolTip* property

WPF vezérlők – Content Control (4)

```
<CheckBox>
  <CheckBox.ToolTip>
    <StackPanel>
      <Label FontWeight="Bold" Background="Blue"
        Foreground="White" Content="CheckBox"/>
      <TextBlock Padding="10" TextWrapping="WrapWithOverflow"
        Width="200" Text="Épp egy ToolTip jelenik meg az
  érintett CheckBox felett"/>
      <Line Stroke="Black" StrokeThickness="1" X2="200"/>
      <StackPanel Orientation="Horizontal">
        <Label FontWeight="Bold" Content="Nyomjon F1-et
  további információért"/>
      </StackPanel>
    </StackPanel>
  </CheckBox.ToolTip>
</CheckBox>
```



WPF vezérlők – Content Control (5)

- Konténerek fejléccel
  - *GroupBox*
    - *Header*-ben bármi lehet

```
<GroupBox Header="Számok">
  <StackPanel>
    <CheckBox>Egy</CheckBox>
    <CheckBox>Kettő</CheckBox>
    <CheckBox>Három</CheckBox>
  </StackPanel>
</GroupBox>
```

  - *Expander*
    - ~ *GroupBox*, összecsukható

WPF vezérlők – Items Control

- Tetszőleges számú tartalmazott elem
- *System.Windows.Controls.ItemsControl*
- *Items* property
- Renderelés
  - *UIElement* : szokás szerint
  - Minden más : *TextBlock*-ként (vagy *DataTemplate*)
- *ListBox, ComboBox, ListView, TreeView, Menu, ToolBar, StatusBar*

WPF vezérlők – Range Control

- *System.Windows.Controls.RangeBase*
- Számértéket jelenítenek meg
- *ProgressBar, Slider*
- *Value, Minimum, Maximum*

WPF vezérlők – Text Control

- *TextBox*
- *RichTextBox*
- *PasswordBox*

Tartalom

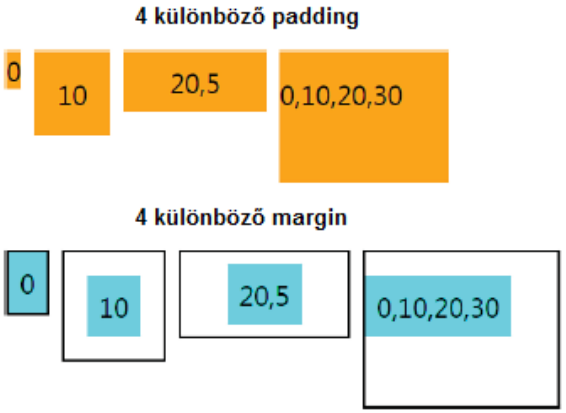
- Bevezetés
- Architektúra
- XAML
- WPF belülről
- WPF vezérlők
- ***Méretezés, pozícionálás, transzformálás***
- Panelek
- Adatkötés
- Erőforrások
- Style, Control template

Méretezés … - Méretezés

- Elrendezés változáskor (layout, pl ablak átméretezés)
- A szülő lekérdezi az igényelt helyet a gyerektől
  - *MinWidth, MinHeight,* tartalmazott elemek
- *Width, Height*
  - Explicit méret megadás (min/max-on belül)
  - double.NaN (Auto)

Méretezés … - Méretezés (2)

- *RenderSize, ActualWidth, ActualHeight*
  - (!) aszinkron (kiv. *LayoutUpdated)*
- *Margin (Thickness)*
  - Extra hely a control körvonalán kívül
- *Padding (Thickness)*
  - Extra hely a control körvonalán belül
  - *Tickness* : 1, 2, 4 érték



Méretezés … - Méretezés (3)

- Visibility
  - *UIElement*
  - *System.Windows.Visibility* enum
    - *Visible, Hidden, Collapsed*

Méretezés … - Pozícionálás

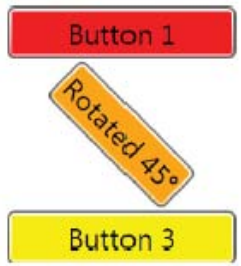
- Alignment
  - Mi legyen az extra hellyel?
    - A gyermek dönt róla
  - *FrameworkElement.HorizontalAlignment, FrameworkElement.VerticalAlignment*

Méretezés … - Pozícionálás (2)

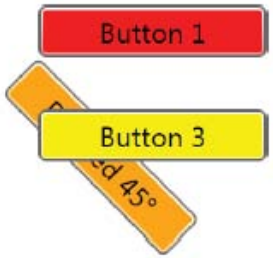
- Content Alignment
  - Hogyan helyezkedjen el a gyermek?
    - vs. Alignment (az extra hellyel mit kezdjen)

Méretezés … - Transzformációk

- *FrameworkElement*
- *ActualWidth, ActualHeight* változatlan
- *LayoutTransform*
  - Elrendezés meghatározása előtt
- *RenderTransform*
  - Elrendezés után, renderelés előtt



Forgatás LayoutTransformmal

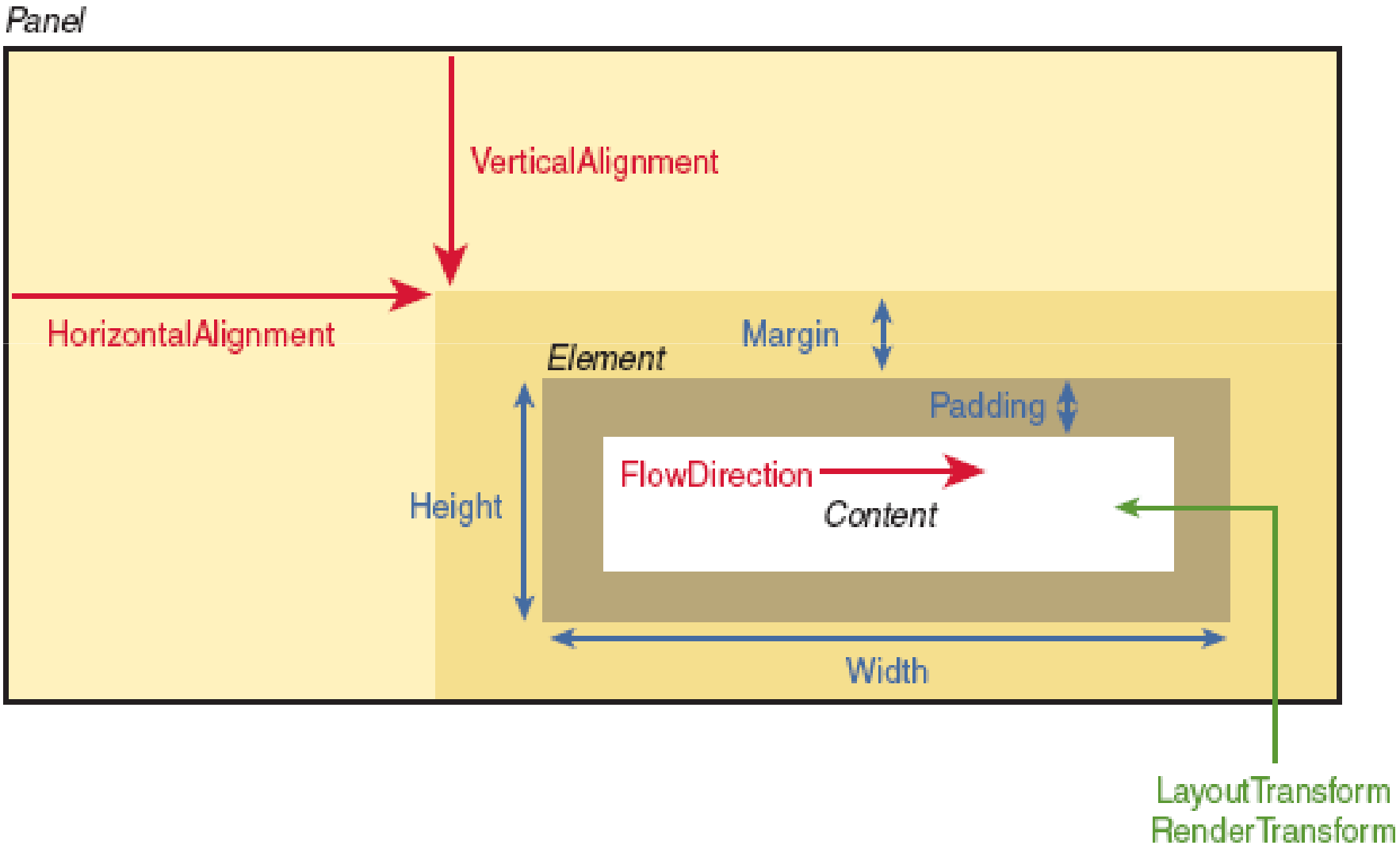


Forgatás RenderTransformmal

Méretezés … - Transzformációk (2)

- Típusok
  - *TranslateTransform* (eltolás)
  - *RotateTransform* (forgatás)
  - *ScaleTransform* (skálázás)
  - *SkewTransform* (döntés)
  - *MatrixTransform* (tetszőleges)
  - *TransformationGroup* (kompozit)

Méretezés … - Összefoglaló



Tartalom

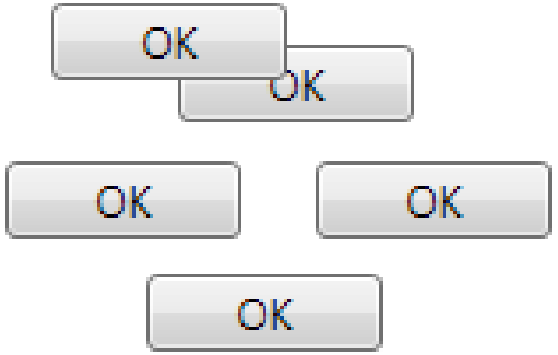
- Bevezetés
- Architektúra
- XAML
- WPF belülről
- WPF vezérlők
- Méretezés, pozícionálás, transzformálás
- ***Panelek***
- Adatkötés
- Erőforrások
- Style, Control template

Panelek

- Layout logika
- *System.Windows.Controls.Panel*
- *Children (UIElementCollection)*

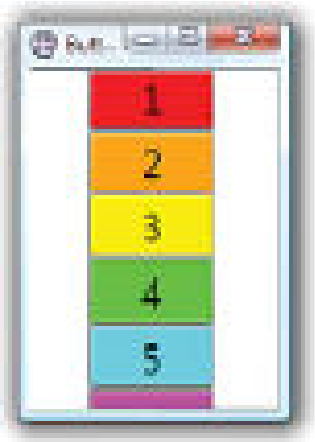
Panelek – Canvas

- Abszolút pozícionálás
- *Left, Top, Right, Bottom* attached propertyk (vs. WinForms)
  - Egy dimenzió mentén csak egy használható
- *Zindex* attached property
  - Takarás
  - Azonos érték esetén a sorrend dönt



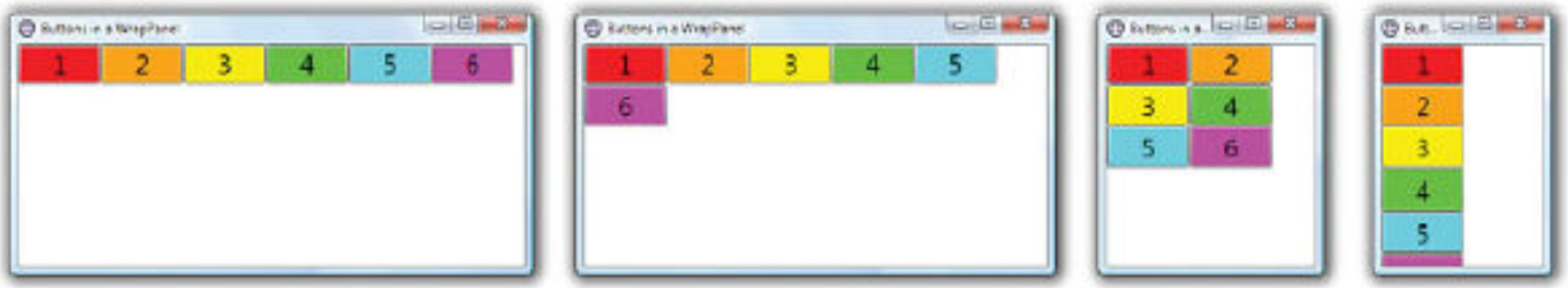
Panelek – StackPanel

- Függőleges vagy vízszintes, folytonos kitöltés
- *StackPanel.Orientation*
  - Default: *Vertical*
- *StackPanel.FlowDirection*
  - Default: *LeftToRight*



Panelek – WrapPanel

- ~*StackPanel*
  - Egy sorba (oszlopba több elem is kerülhet)
  - Kitöltés iránya az *Orientation* propertyvel állítható



Panelek – DockPanel

- Gyerekeket a konténer egyes oldalihoz dokkolhatjuk
- *DockPanel.Dock*
  - *Left, Top, Right, Bottom, Stretch*
- Dokkolás a felvétel sorrendjében

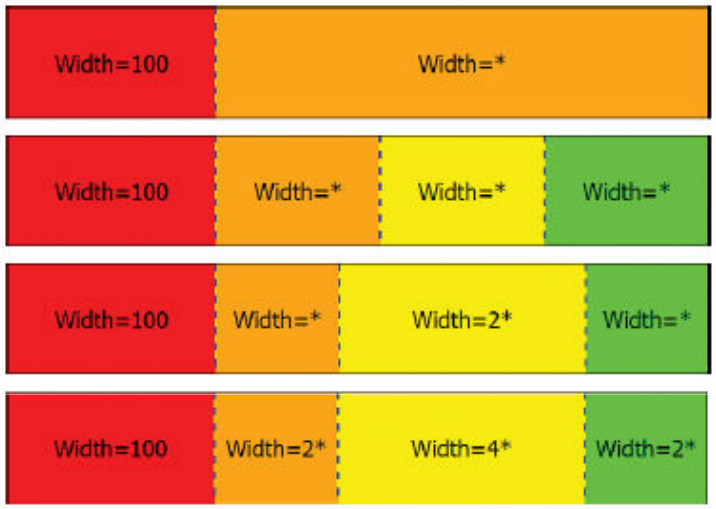
Panelek – Grid

- Táblázatos megjelenítés
- *RowDefinitions (RowDefinition)*
  - *Height (GridLength)*
- *ColumnDefinitions (ColumnDefinition)*
  - *Width (GridLength)*
- *Gird.Row, Grid.Column*

Panelek – Grid (2)

- *GridLength*
  - Méret típus
  - Abszolút
    - Eszközfüggetlen pixel
  - Auto
    - double.NaN (amennyire szükség van)
  - Arányos (* méretezés)
    - „n*” forma
    - n_i / ∑ n_i * Width

Panelek – Grid (3)



- Utolsó sor:

```
<Grid.ColumnDefinitions>
  <ColumnDefinition Width="100"/>
  <ColumnDefinition Width="2*"/>
  <ColumnDefinition Width="4*"/>
  <ColumnDefinition Width="2*"/>
</Grid.ColumnDefinitions>
```

Tartalom

- Bevezetés
- Architektúra
- XAML
- WPF belülről
- WPF vezérlők
- Méretezés, pozícionálás, transzformálás
- Panelek
- ***Adatkötés***
- Erőforrások
- Style, Control template

Adatkötés – Binding

- Két property összekötése
  - Egymástól való függés
- *System.Windows.Data.Binding*
  - *Source* : forrás objektum
  - *Path* : forrás property, elmaradhat
- Cél objektum : *DependencyObject*
- Cél property : dependency property
  - *FrameworkElement.SetBinding vagy*
  - *BindingOperations.SetBinding*

Adatkötés – Binding (2)

- Pl. *TreeView*:

```
Binding binding = new Binding();
binding.Source = treeView;
binding.Path =
  new PropertyPath("SelectedItem.Header");
textBlock.SetBinding(TextBlock.TextProperty,
  binding);
```

- Törlés
  - BindingOperations.ClearBinding
  - Explicit propertyállítás

Adatkötés – Binding (3)

- Forrás property nem feltétlen dependency
  - Lemondunk a változáskezelésről
  - *INotifyPropertyChange* implementálása
  - *XXXXChanged* esemény implementálása
    - *(XXXX* = property neve)
- *Binding.Converter*
  - Adatcsere előtt az adat transzformálható
  - *IValueConverter*-t implementáló osztály

Adatkötés – XAML

- Korábbi példa:
  - `<TextBlock Text="{Binding ElementName=treeView,Path=SelectedItem.Header}"/>`
  - *Markup extension*
  - *ElementName*
    - *FrameworkElement, FrameworkContentElement*
  - *Source (ElementName* helyett*)*
  - *RelativeSource*

Adatkötés – XAML (2)

- *RelativeSource*
  - Forrás és célobjektum megegyezik
    - `{Binding RelativeSource={RelativeSource Self}}`
  - ControlTemplate esetén
    - `{Binding RelativeSource={RelativeSource TemplatedParent}}`
  - Legközelebbi, adott típusú szülő
    - `{Binding RelativeSource={RelativeSource FindAncestor, AncestorType={x:Type desiredType}}}`

Adatkötés – XAML (3)

- N. adott típusú szülő
  - `{Binding RelativeSource={RelativeSource FindAncestor, AncestorLevel=n, AncestorType={x:Type desiredType}}}`
- Megelőző elem kollekcióban
  - `{Binding RelativeSource={RelativeSource PreviousData}}`

➢ *FrameworkElement.DataContext*

- Alapértelmezett forrás
- Logical treeben lefelé rekurzívan érvényes

Adatkötés – Kötés kollekcióhoz

- *ListBox* feltöltése
  - `<ListBox ItemsSource="{Binding RelaticeSource={RelativeSource TemplatedParent}, Path=Photos}"/>`
  - *ItemsSource* property való adatkötéshez

    (Items manuális feltöltéshez, egyszerre csak egy)
  - Ha *Photos* nem *UIElement*-ek listája, akkor a *ToString()* értékük fog megjelenni

Adatkötés – kötés kollekcióhoz (2)

- *DisplayMemberPath*
  - Kiválasztható a megjelenítendő property
  - ```
    <ListBox DisplayMemberPath="Name"
    ItemsSource="{Binding RelaticeSource={RelativeSource
    TemplatedParent}, Path=Photos}"/>
    ```
- *DataTemplate*
  - Testreszabott Visual Tree készítése
  - *ItemsControl.ItemTemplate*
    - De pl. *ContentControl.ContentTemplate*
    ```

Adatkötés – kötés kollekcióhoz (3)

- DataTemplate példa:

```
<ListBox ItemsSource="...">
  <ListBox.ItemTemplate>
    <DataTemplate>
      <Image Source="{Binding Path=FullPath}"
        Height="35"/>
    </DataTemplate>
  </ListBox.ItemTemplate>
</ListBox>
```

Tartalom

- Bevezetés
- Architektúra
- XAML
- WPF belülről
- WPF vezérlők
- Méretezés, pozícionálás, transzformálás
- Panelek
- Adatkötés
- ***Erőforrások***
- Style, Control template

Erőforrások

- Az alkalmazás nem kód-alapú részei (bitmap, font, szöveg)
- Bináris erőforrás
- Logikai erőforrás

Erőforrások – Bináris erőforrás

- „Hagyományos” erőforrások (pl. bitmap)
  - XAML is így tárolódik
- Fajtái
  - Assemblybe ágyazott
    - Build Action = Resource
  - Assemblyn kívül, fordítási időben ismert
    - Build Action = Content
    - Relatív útvonal ismert ezáltal
  - Fordítási időben nem ismert

Erőforrások – Bináris erőforrás (2)

- XAML
  - `<Image Height="21" Source="previous.gif"/>`
  - Relatív elérési út elég, ha
    - Build Action = Resource v. Content vagy
    - a XAML nem fordított (runtime beolvasott)
  - Egyébként
    - Abszolút elérési út, vagy
    - Speciális URI
      - `pack://siteOfOrigin:,,,/A/B/previous.jpg`
      - `pack://application:,,,/MyDll;Component/logo.jpg`
        - `pack://application:,,,/` csak proc. kódban kell

Erőforrások – Bináris erőforrás (3)

- C#

```
Image image = new Image();
image.Source = new BitmapImage(new
  Uri("pack://application:,,,/logo.jpg"));
```

- Tetszőleges erőforrás elérhető
  - *Application.GetResourceStream(…)*

Erőforrások – Logikai erőforrás

- *FrameworkElement.Resource, FrameworkContentElement.Resource*
  - *ResourceDictionary*
    - Kulcs-objektum párok
- Példa:

```
<Window xmlns="…" xmlns:x="…" Title="Simple Window">
  <Window.Resources>
    <SolidColorBrush x:Key="backgroundBrush">
      Yellow</SolidColorBrush>
    <SolidColorBrush x:Key="borderBrush">
      Red</SolidColorBrush>
  </Window.Resources>
```

Erőforrások – Logikai erőforrás (2)

- Erőforrás elérése
  - `<StaticResource ResourceKey="backgroundBrush"/>`
  - `<Button Background="{StaticResource backgroundBrush}" … />`
- A logical treeben felfelé haladva valamennyi erőforrás elemeit látjuk (alkalmazás szintűt is)
- *DynamicResourceExtension*
  - változásértesítés

Tartalom

- Bevezetés
- Architektúra
- XAML
- WPF belülről
- WPF vezérlők
- Méretezés, pozícionálás, transzformálás
- Panelek
- Adatkötés
- Erőforrások
- ***Style, Control template***

Style, Ctrl. Template - Style

- *System.Windows.Style*
- Property-érték párok, „kötegelt” beállítás
  - *Setter* (egy pár)
    - *Property, Value*
  - *TargetType*
    - Típus szűrés (csak pontos egyezés)
    - *x:Key* nélkül az erőforrás hatókörén belül minden egyező típusú objektumre

Style, Ctrl. Template – Style (2)

```
<StackPanel Orientation="Horizontal">
  <StackPanel.Resources>
    <Style x:Key="buttonStyle">
      <Setter Property="Button.FontSize" Value="22"/>
      <Setter Property="Button.Background" Value="Purple"/>
      <Setter Property="Button.Foreground" Value="White"/>
      <Setter Property="Button.Height" Value="50"/>
      <Setter Property="Button.Width" Value="50"/>
      <Setter Property="Button.RenderTransform">
        <Setter.Value>
          <RotateTransform Angle="10"/>
        </Setter.Value>
      </Setter>
    </Style>
  </StackPanel.Resources>
  <Button Style="{StaticResource buttonStyle}">1</Button>
  <Button Style="{StaticResource buttonStyle}">2</Button>
  <Button Style="{StaticResource buttonStyle}">3</Button>
</StackPanel>
```

Style, Ctrl. Template – Style (3)

- Triggerek
  - Property érték-változás figyelése, propertyk beállítása
  - Módosítások visszavonása

```
<Style x:Key="buttonStyle">
  <Style.Triggers>
    <Trigger Property="IsMouseOver" Value="True">
      <Setter Property="RenderTransform">
        <Setter.Value>
          <RotateTransform Angle="10"/>
        </Setter.Value>
      </Setter>
    </Trigger>
  </Style.Triggers>
```

Style, C. T. – Control Template

- Control visual tree-jének kicserélése
- *Control.Template*
- Típus alapú szűrés (~ style)
- Triggerek (~ style)

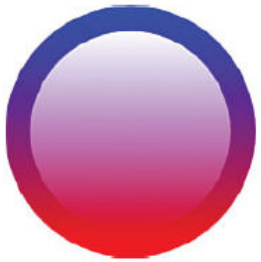
Style, C. T. – Control Template (2)

```
<Grid>
 <Grid.Resources>
  <ControlTemplate x:Key="buttonTemplate">
   <Grid>
    <Ellipse Width="100" Height="100">
     <Ellipse.Fill>
      <LinearGradientBrush StartPoint="0,0" EndPoint="0,1">
       <GradientStop Offset="0" Color="Blue"/>
       <GradientStop Offset="1" Color="Red"/>
      </LinearGradientBrush>
     </Ellipse.Fill>
    </Ellipse>
    <Ellipse Width="80" Height="80">
     <Ellipse.Fill>
      <LinearGradientBrush StartPoint="0,0" EndPoint="0,1">
       <GradientStop Offset="0" Color="White"/>
       <GradientStop Offset="1" Color="Transparent"/>
      </LinearGradientBrush>
     </Ellipse.Fill>
    </Ellipse>
```

Style. C.T – Control Template (3)

```
    </Grid>
   </ControlTemplate>
  </Grid.Resources>
  <Button Template="{StaticResource buttonTemplate}"
   Content="OK"/>
</Grid>
```

- Eredmény:

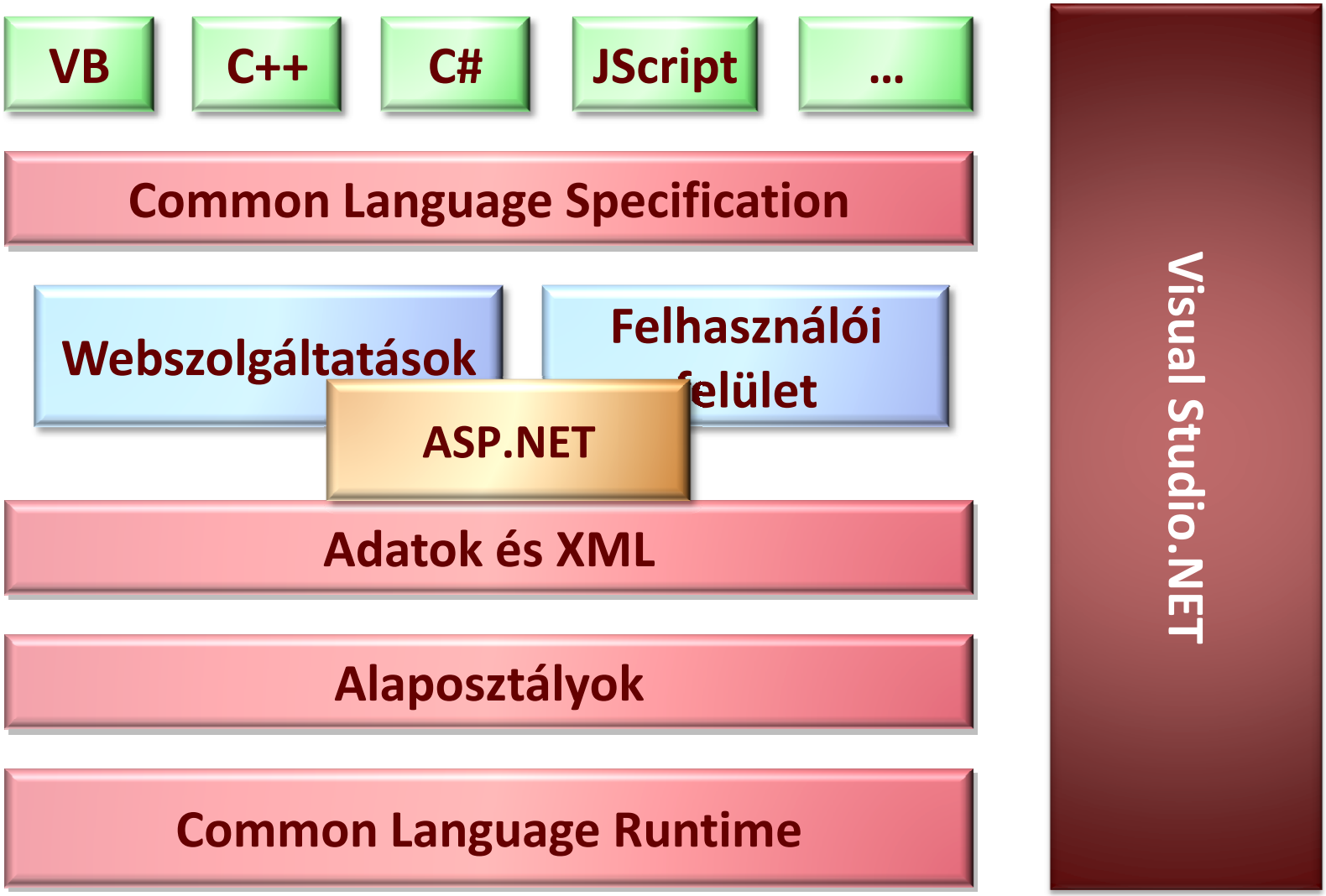


ASP.NET alapok

Miről lesz szó

- ASP.NET architektúrája
- ASP.NET a gyakorlatban
  - Webes űrlapok
  - Adatkötés
  - Konfiguráció, állapotmegőrzés
  - Moduláris felépítés

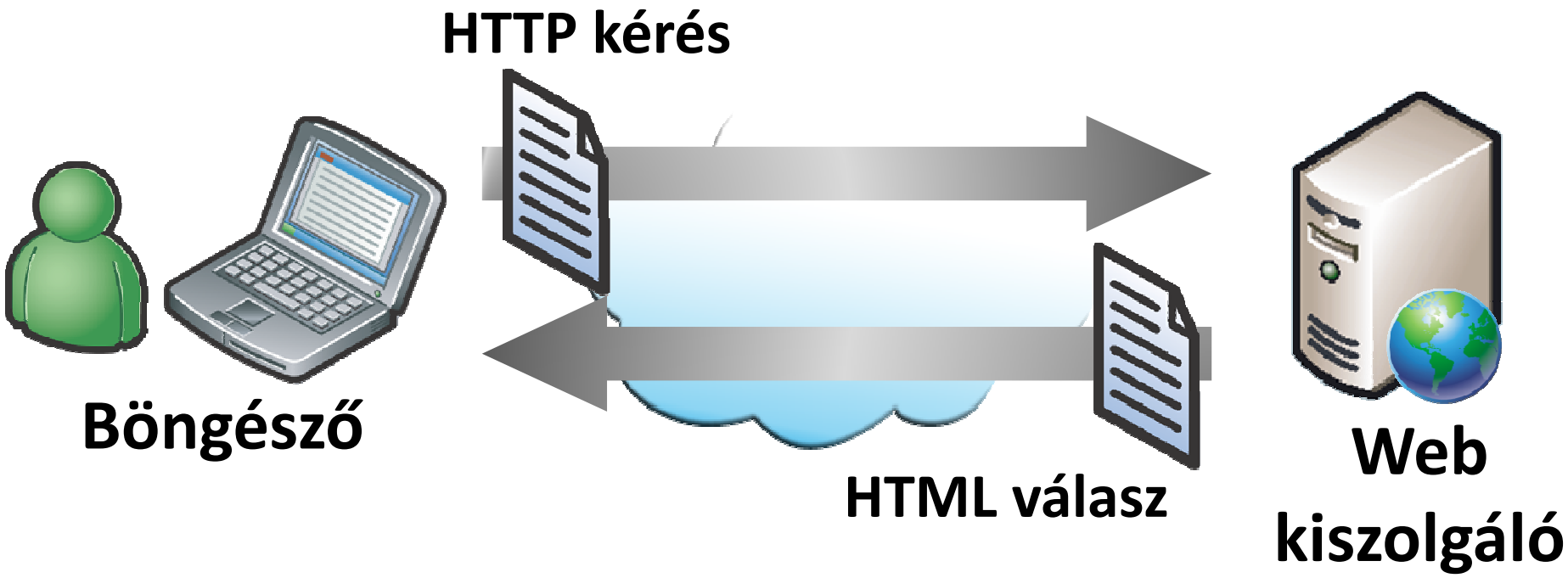
.NET Keretrendszer



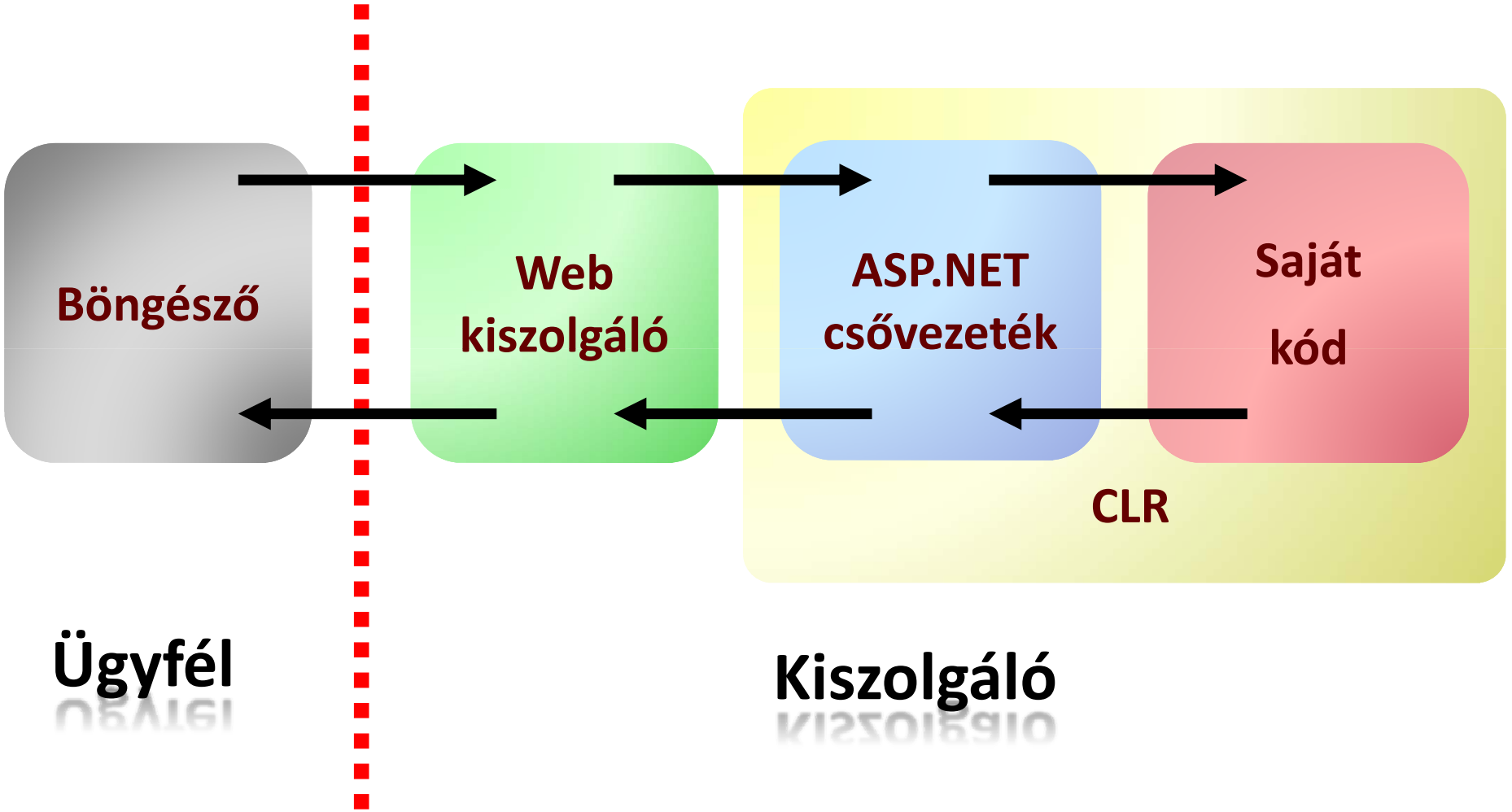
ASP.NET

- Összeállítja, telepíti és futtatja a webes alkalmazásokat
- A .NET Framework része
  - CLR által **felügyelt**, **natív** kódú végrehajtás
  - Egységes programozási paradigma
  - C#, Visual Basic .NET, Jscript, …
- Cél: „Visual Basicet a webre!”
  - Új kontroll- és esemény alapú modell

Kérések feldolgozása



Architektúra

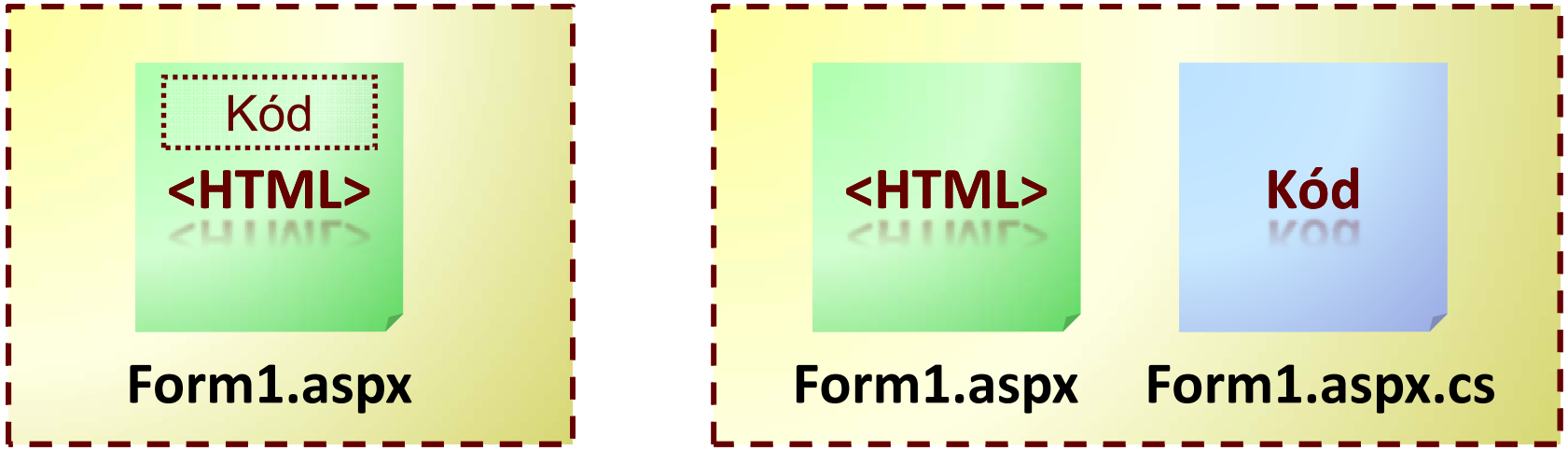


Miről lesz szó

- ASP.NET architektúrája
- **ASP.NET a gyakorlatban**
  - Webes űrlapok
  - Adatkötés
  - Konfiguráció, állapotmegőrzés
  - Moduláris felépítés

Webes űrlapok (webforms)

- Cél: vezérlő- és eseményalapú fejlesztés
- A webform összefogja
  - HTML, direktívák, kontrollok, statikus szövegek
  - Funkcionalitást megvalósító kód
- Elkülönülhet a kód és a megjelenítés



Szerver oldali vezérlők

- Felületelemek megjelenéséért és viselkedéséért felelnek
- Feldolgozzák a kliens oldalról érkező adatokat
  - Elérhetővé teszik a Forms kollekcióban érkező adatokat
  - Eseményeket generálnak a szerver oldalon
- Generálják a kliensnek küldendő HTML kódot
  - Több böngésző típust is támogathatnak (App_Browsers)

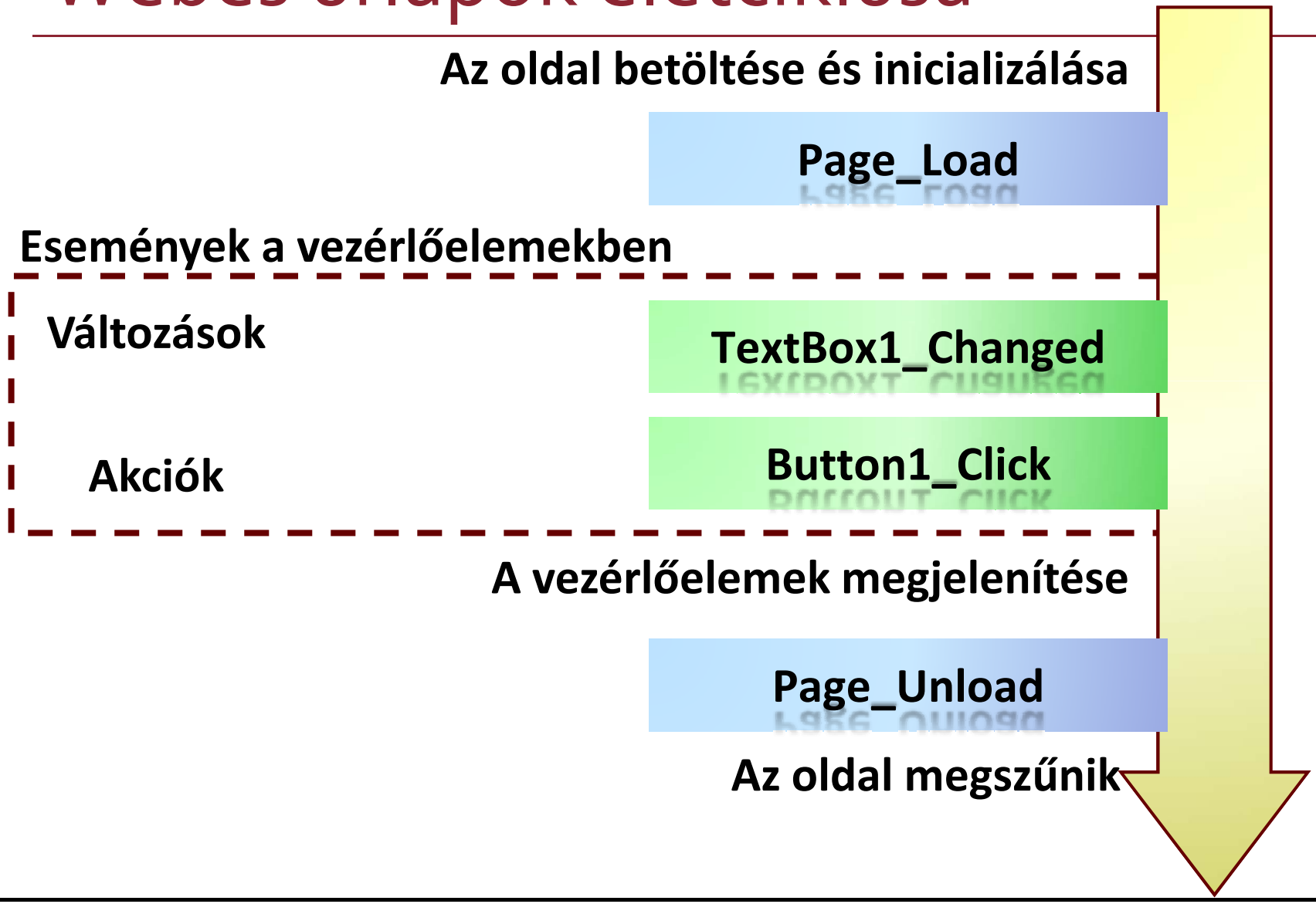
Vezérlőelemek szerepe

- Deklaratív létrehozás: **runat=server**
- Tulajdonságokon keresztül (deklaratív módon) testre szabható a megjelenésük és működésük
- Állapotmentesek a szerver oldalon
  - Egy vezérlőelem állapota („ViewState”) együtt utazik a lappal
- Az ID attribútummal maghatározott névvel lehet az oldalhoz tartozó kódból az elemet elérni
  - A vezérlő egy .NET objektum

Szerver oldali vezérlők típusai

- HTML Controls
  - Egy HTML tag-et tesz elérhetővé a kódból
  - System.Web.UI.HtmlControls névtérben
  - Kicsit gyorsabbak, mint a WebControlok
- Web Controls
  - Absztraktabb osztályok
  - System.Web.UI.WebControls névtérben
  - Összetett HTML kódot generálnak
  - Bonyolultabb funkcionalitás
- Mobile Controls

Webes űrlapok életciklusa



Adatkötés

- Kontrollok majdnem minden tulajdonságához
- Tervezési- vagy futási időben
- Kétfajta adatkötés
  - Egyszerű: TextBox, Button, Label
  - Listás: DataGrid, Repeater, DataList
- Egyirányú illetve kétirányú

Egyszerű adatkötés

- Adatforrás
  - Lapon lévő objektum vagy annak tulajdonsága
  - Metódushívás eredménye
  - Másik vezérlőelem
- Deklaratív kötés

```
<asp:Label ID=„lblText" runat="server" Text=”<%# custId %>” />
```

- DataBind() meghívása

```
protected void Page_Load( object sender, EventArgs e )
{
        lblText.DataBind();
}
```

Listás adatkötés

- IEnumerable típusú adatforráshoz
  - ArrayList, Generikus lista, DataReader, DataRow…
- Deklaratív kötés
- Kötés kódból

```
myDataGrid.Datasource = ds;
```

- DataBind() meghívása
  - A Page.DataBind() az összes tartalmazott vezérlőre meghívja

Adatkötés

Webalkalmazások konfigurálása

- XML formátumú
- Minden könyvtárban lehet
- A beállítások öröklődnek
  - Ős: machine.config és web.config
  - Nem minden beállítás adható meg minden szinten
  - Tartalmazhat standard és egyedi beállításokat

Session state

- Sok erőforrást (memóriát) igényelhet a kiszolgáló oldalon
- Nem lehet megosztani a felhasználók között
- Amennyiben nincs feltétlen rá szükség, érdemes kerülni a használatát

```
<%@ Page EnableSessionState="ReadOnly | False" %>
```

Session objektum tárolása

- In-proc (alapértelmezett)
  - A leggyorsabb hozzáférést teszi lehetővé
  - Nem kell az objektumokat szerializálni / deszerializálni
  - A limit a kiszolgáló gép memóriája
  - Webfarmok esetén korlátozottan használható
- State service
  - Külön, out-of-process állapotszerver (Windows Service)
  - Szükséges szerializálni/deszerializálni
  - Alkalmazható webfarmok esetén is
  - A mérethatár az egy folyamat maximális méretével egyenlő (kb. 2GB)
  - Valamivel lassabb, mint a belső tárolás
- SQL szerver
  - Ha nagyon nagy adattömeget kell tárolni
  - A leglassabb az összes közül

Session használat

- Elem hozzáadása
- Elem elkérése
- Session-höz tartozó események

Validator vezérlők

- Felhasználó által megadott adatok ellenőrzése
- A BaseValidator osztályból származnak
- Kliens és szerver oldali ellenőrzés:
  - Kliens: gyors visszajelzés
  - Szerver: a kliens oldali ellenőrzés kikerülése ellen
- Lekérdezendő az eredmény szerver oldalon (Page.IsValid)
- Beállítható, hogy mely elemeket ellenőrizzék
- Megadható a hibaüzenet
- Az üres mező definíció szerint érvényes
  - Kivéve, ha RequiredFieldValidator is van rajta

Beépített validatorok

- RequiredFieldValidator
  - A mező nem lehet üres, vagy default értékű
- CompareValidator
  - Konstanssal, vagy másik beviteli mező értékével hasonlítja össze
- RangeValidator
  - Két konstans érték közé kell esnie az adatnak
  - Szöveg, szám, dátum típus
- RegularExpressionValidator
  - Reguláris kifejezés adható meg
- CustomValidator
  - Saját vezérlőelem, akár kliens oldali szkripttel
- ValidationSummary
  - Listázza az oldalon lévő hibákat
  - Üzenetablakot is megjeleníthet kliens oldalon

Validátorok alkalmazása

- RequiredField validátor
- Validation Summary

Újrafelhasználás

- Cél: egy komplex, felhasználói felülettel rendelkező komponenst újrafelhasználni
- User Control
  - Újrafelhasználható oldal-részlet
  - Általában több szerver oldali vezérlőből áll össze
- Custom Control
  - Összetett vezérlőelem
  - Felelős a saját HTML kimenetének előállításáért

User Control

- Újrafelhasználható modul, oldal részlet
- Konvertálás ASPX oldalból
  - .ASPX → .ASCX
  - @Page direktíva → @Control
  - <HTML>, <HEAD>, <BODY> elemek törlése
  - Code-behind: Page helyett a UserControl osztályból kell származtatni

```
<%@ Register TagPrefix="uc1" TagName="Login"
             Src="login.ascx" %>
<uc1:Login id="login1" runat="server"/>
```

Összefoglalás

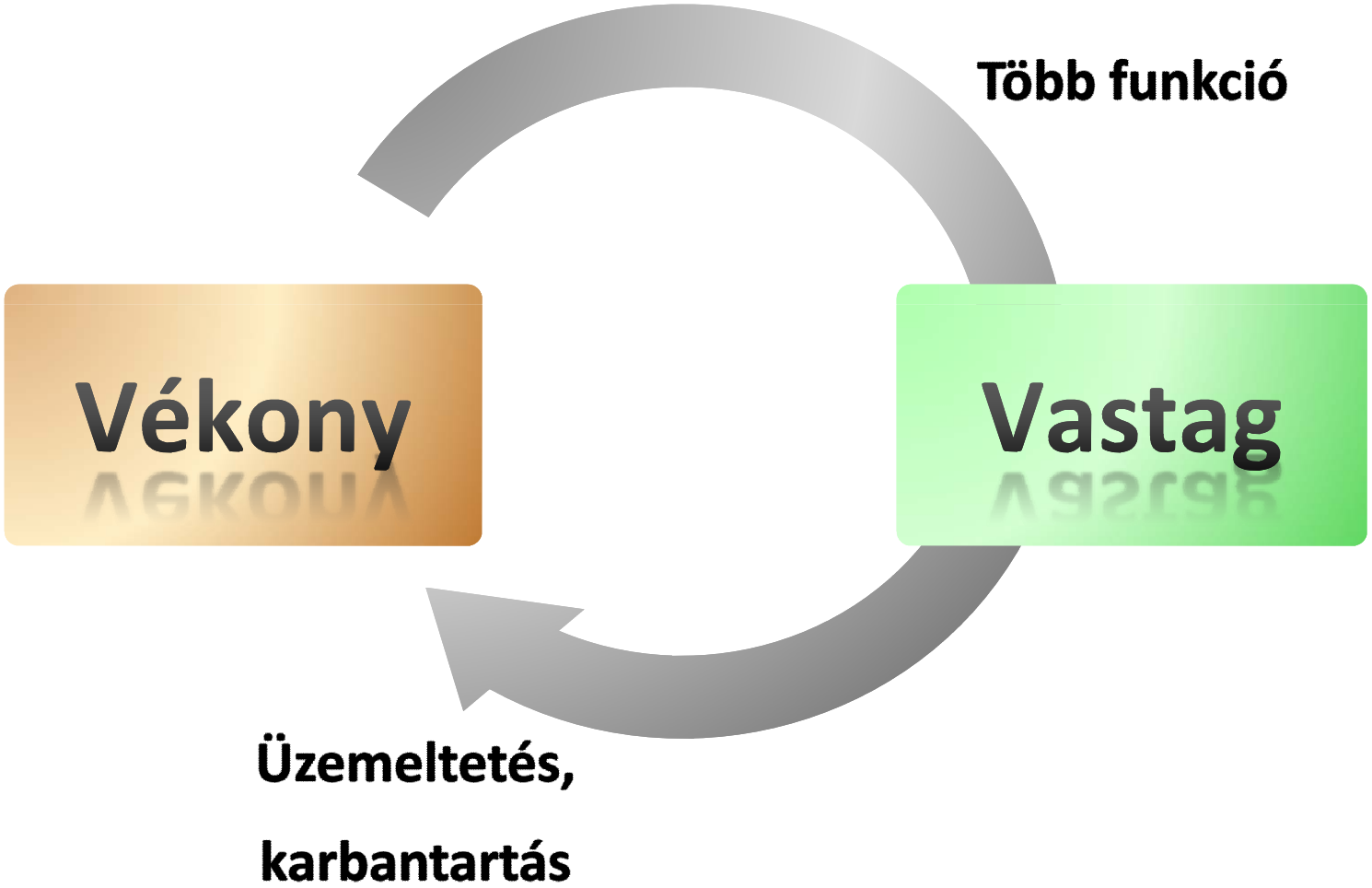
- Kényelmes, esemény alapú fejlesztés
- Nagyfokú VS.NET támogatás
- Fontos, hogy HTTP van a háttérben

Aszinkronitás a web világában

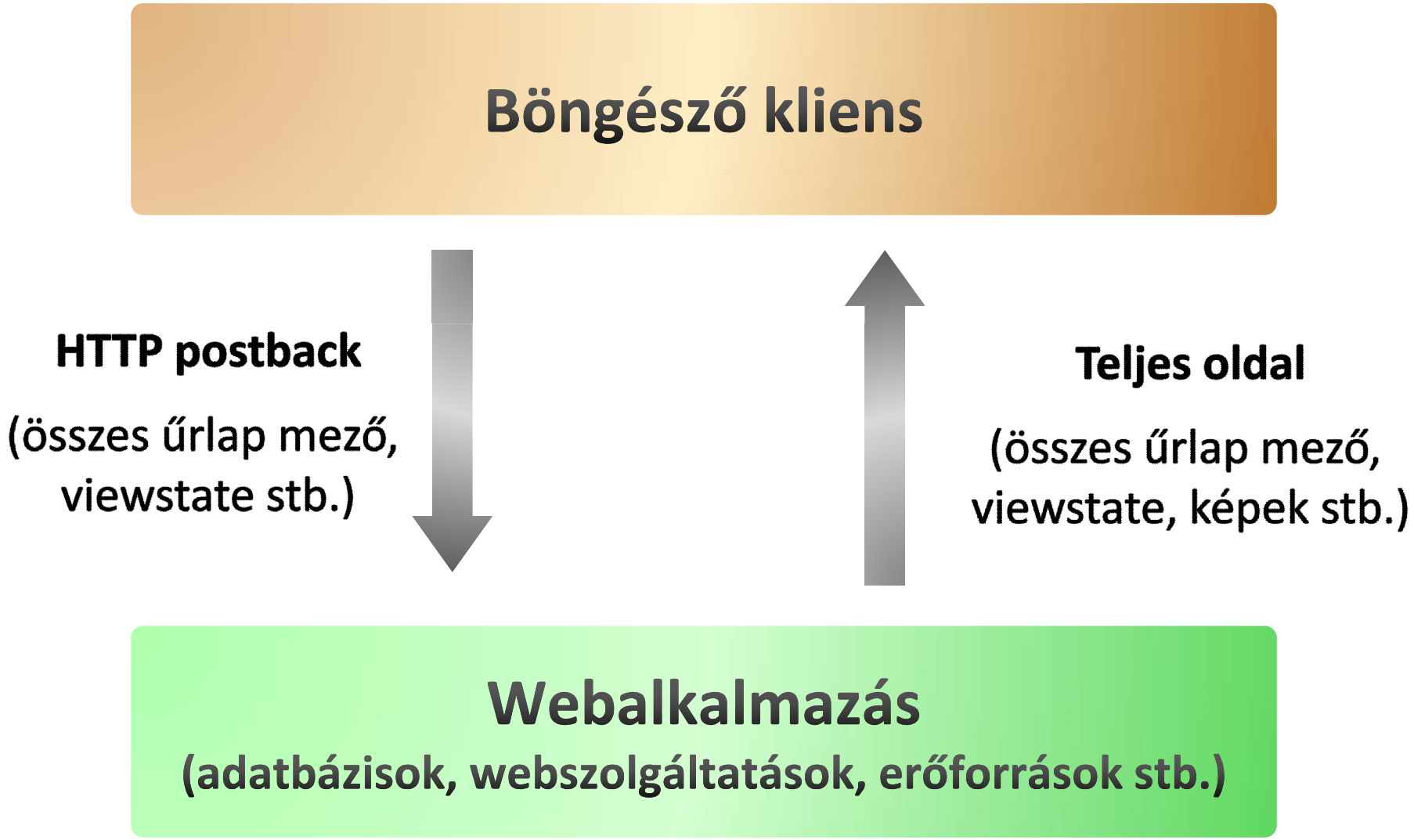
Miről lesz szó

- Szinkron és aszinkron végrehajtás
- Mi az AJAX
- UpdatePanel és a részleges oldalfrissítés
  - Hogyan kerül a kódba JavaScript?
  - Mire jó az UpdatePanel?
  - Hogyan fut le egy részleges oldalfrissítés?

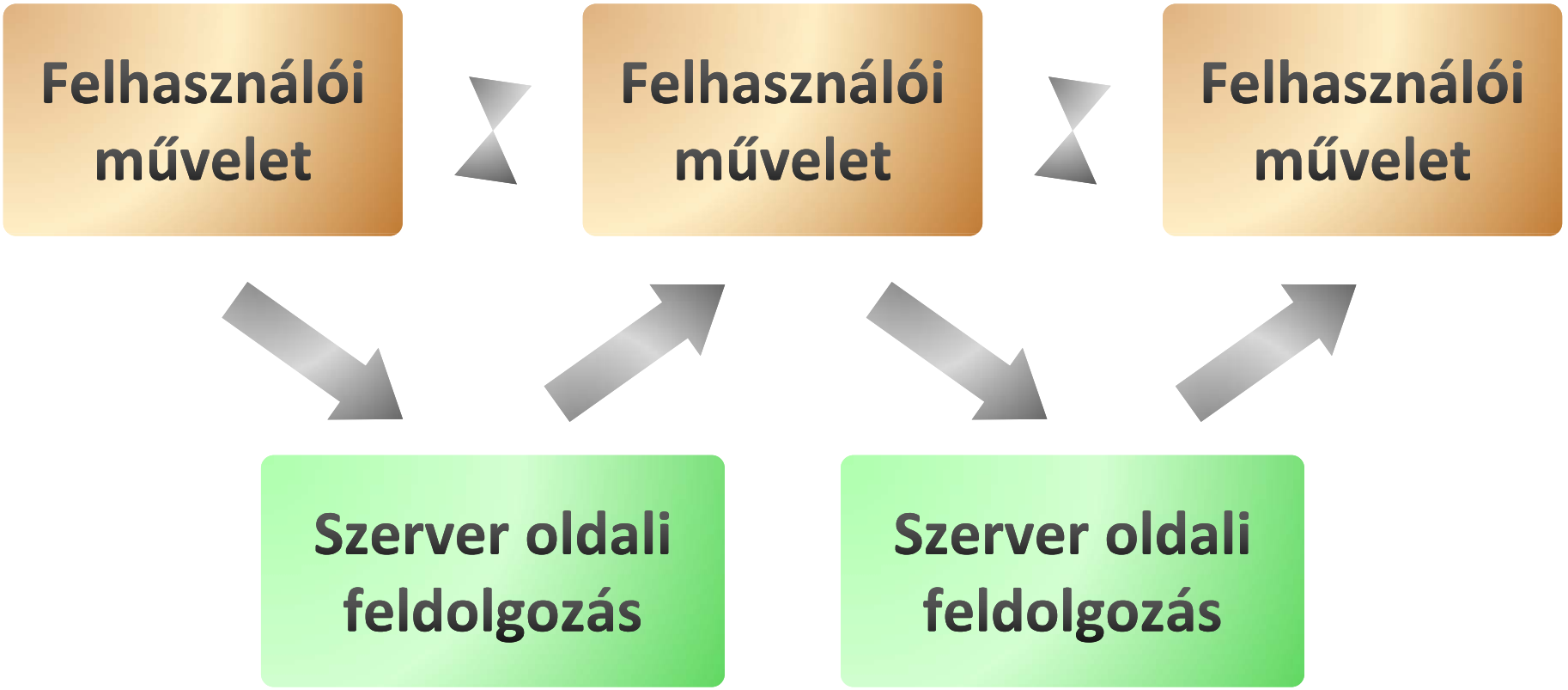
A felhasználói felület fejlődése



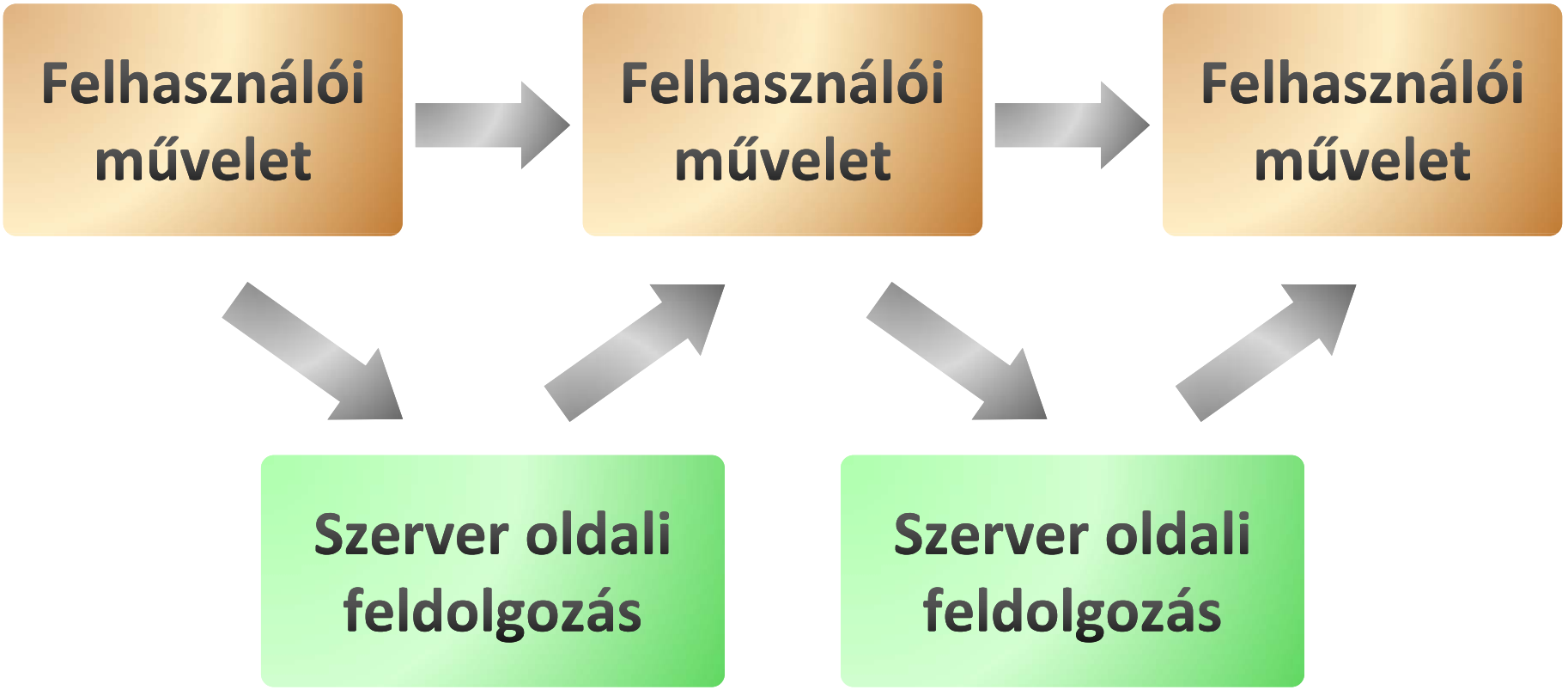
Hagyományos kommunikáció



Szinkron végrehajtás



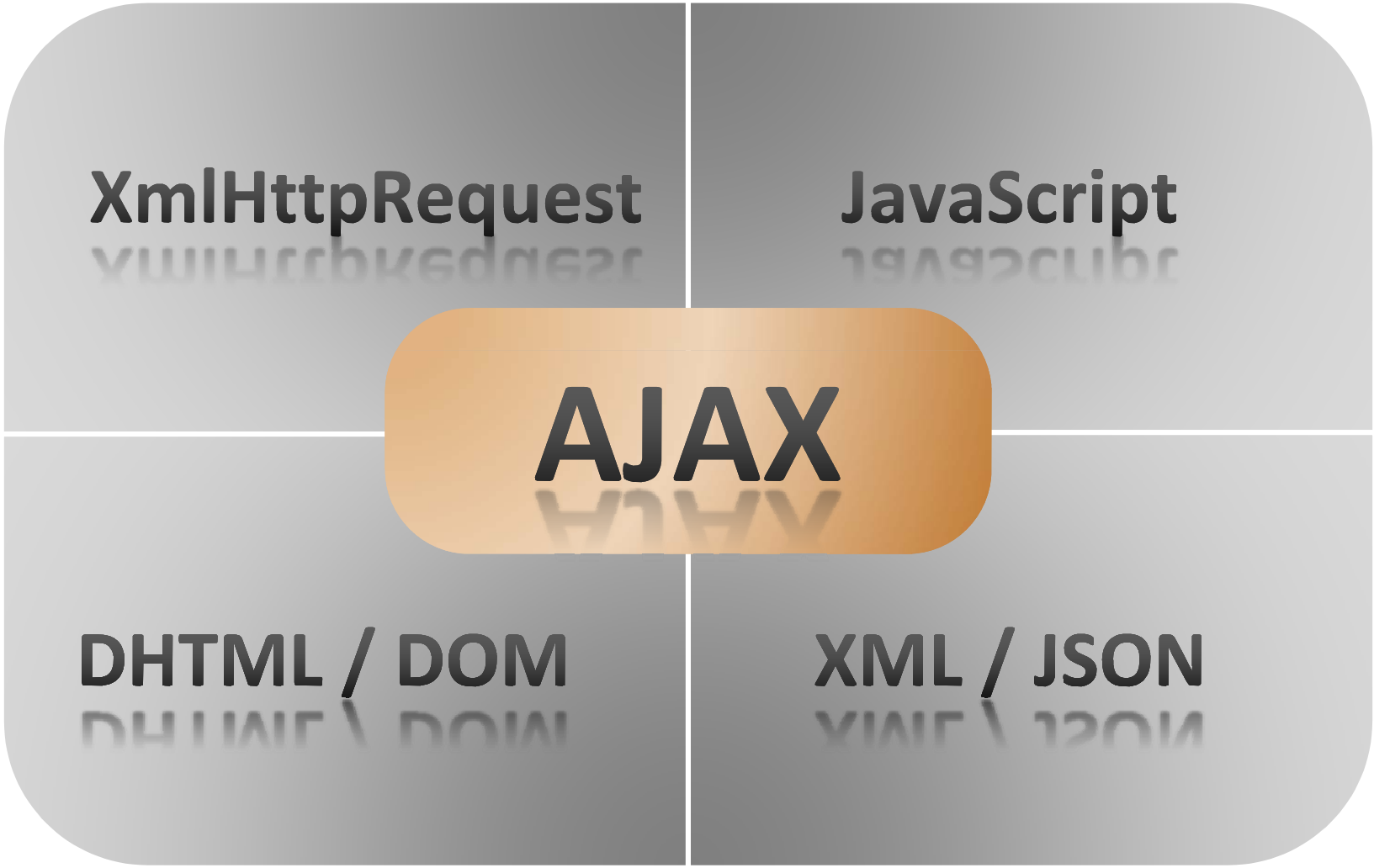
Aszinkron végrehajtás



Miről lesz szó

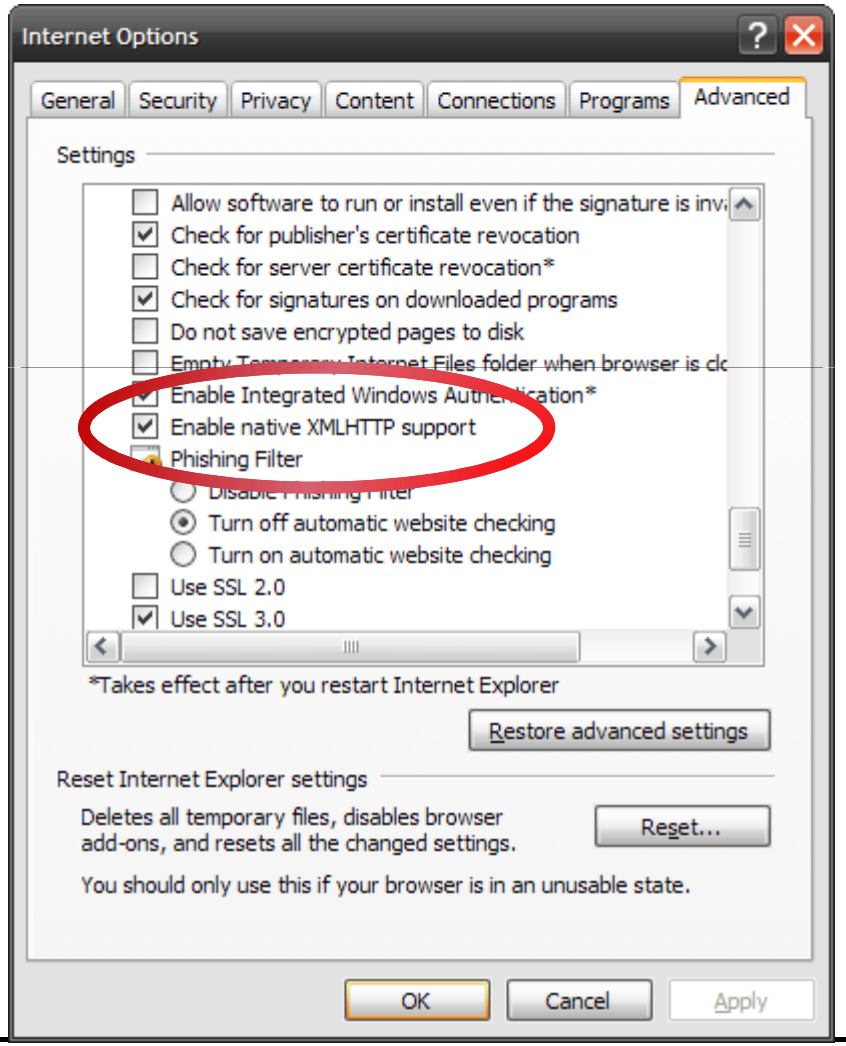
- Szinkron és aszinkron végrehajtás
- **Mi az AJAX**
- UpdatePanel és a részleges oldalfrissítés
  - Hogyan kerül a kódba JavaScript?
  - Mire jó az UpdatePanel?
  - Hogyan fut le egy részleges oldalfrissítés?

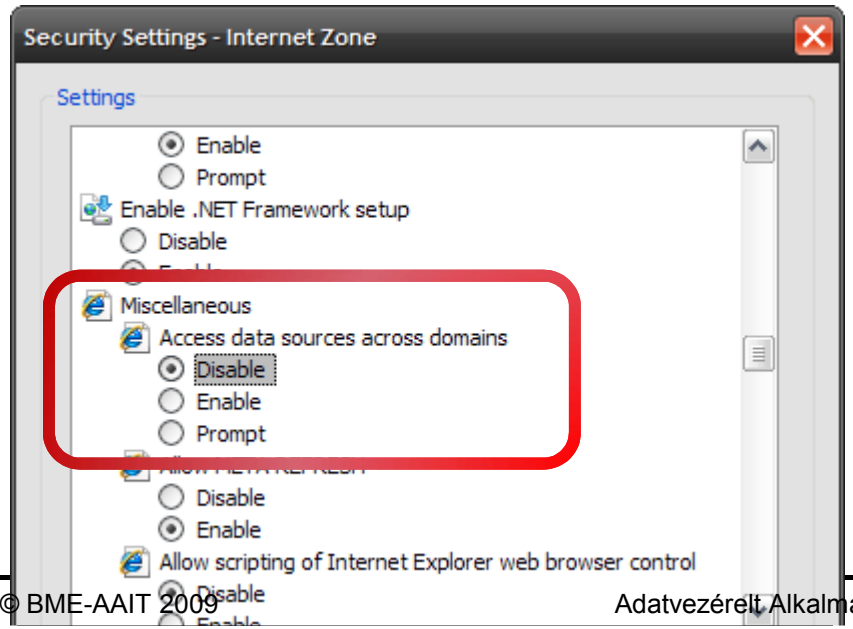
Asynchronous JavaScript and XML



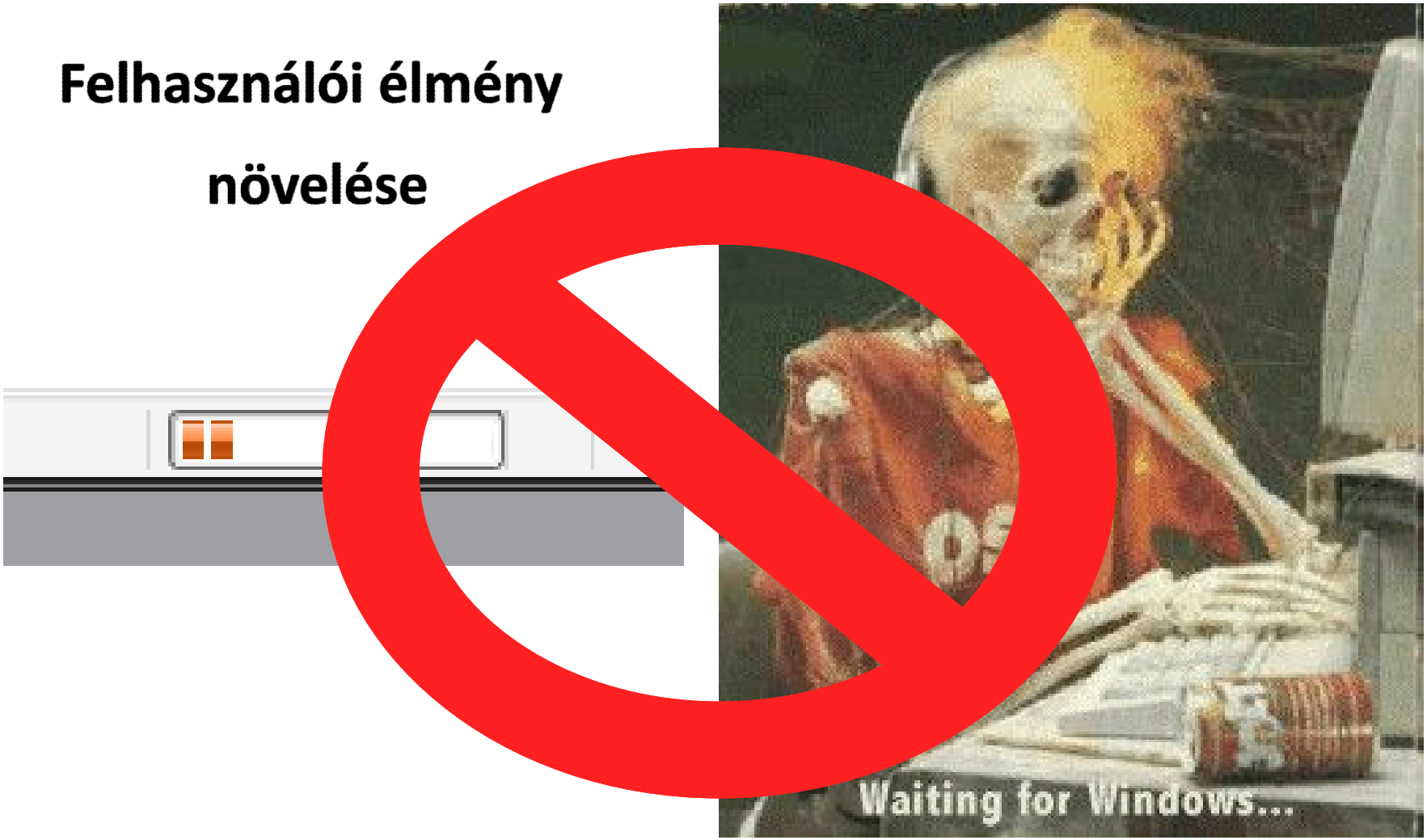
XmlHttpRequest

- Böngészőfüggő
  - ActiveX: IE 5.x, 6.x
  - Natív: Mozilla, Safari, IE 7
- „Same origin” szabály





Az AJAX célja



AJAX pro és kontra

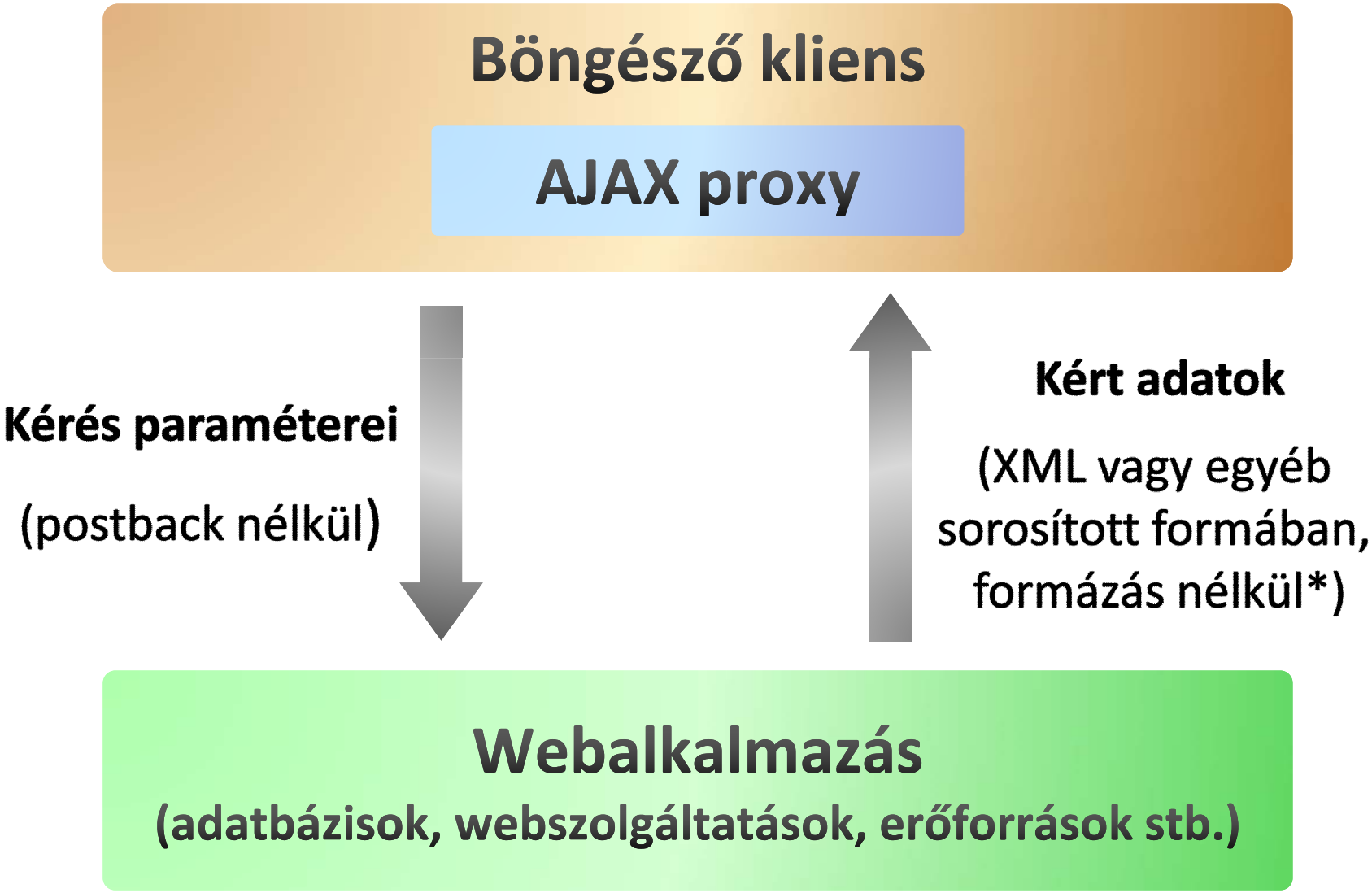
Előnyök

- 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

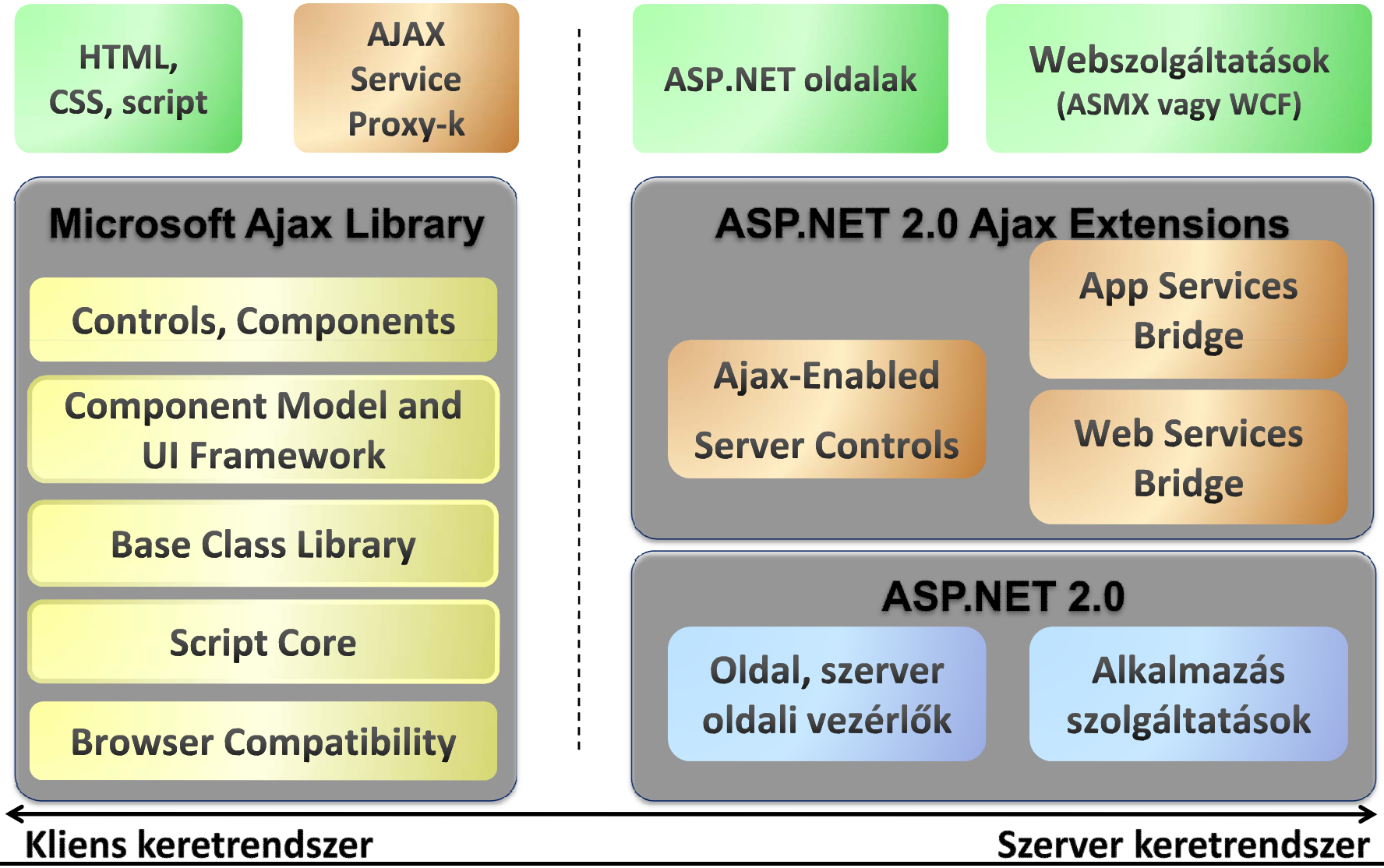
AJAX jellegű kommunikáció



ASP.NET AJAX céljai

- Tipikus problémák megoldása
  - Kliens oldali kódolás nélkül
  - Böngészőfüggetlen kódolás megvalósítása
- Illeszkedés az ASP.NET modellbe
  - Integrálódás és kiterjesztés szerver oldalon
  - ASP.NET szolgáltatások elérése
  - Inkrementális funkció bővítés
- Produktivitás++

ASP.NET AJAX architektúra



Az AJAX válasz az új kihívásokra

- Új felhasználói igények: Web 2.0
- Web 2.0: AJAX + Webszolgáltatások
- AJAX: XmlHttp + DHTML + XML + JavaScript
- ASP.NET AJAX: ASP.NET 2.0++

Miről lesz szó

- Szinkron és aszinkron végrehajtás
- Mi az AJAX
- **UpdatePanel és a részleges oldalfrissítés**
  - Hogyan kerül a kódba JavaScript?
  - Mire jó az UpdatePanel?
  - Hogyan fut le egy részleges oldalfrissítés?

Hogyan írjuk a JavaScriptet?

- A folyamatos postback-ek elkerülésére JavaScriptet használunk
  - Nehéz megírni
  - Erősen böngészőfüggő
  - Hibakeresés időigényes
- Megoldás: Nem mi írjuk
  - Készen kapjuk az AJAX vezérlők által használt JavaScriptet
  - ScriptManager kezeli

ScriptManager

- Minden oldalon elejére szükséges egy példány
- Menedzseli a böngészőnek leküldendő kliens oldali szkripteket
  - Microsoft AJAX Library
  - Saját szkriptek
- Szükséges a szerver oldali vezérlők működéséhez

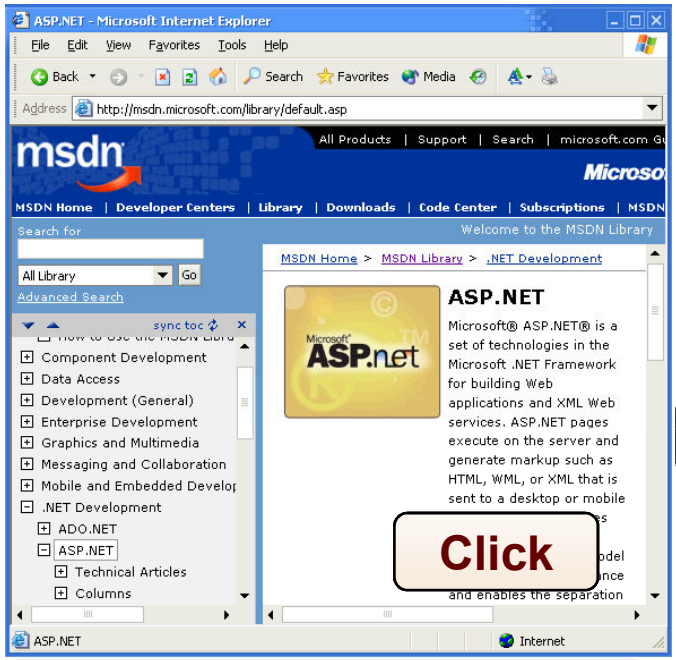
ScriptManager

```
<asp:ScriptManager ID="…" Runat="server"
    EnablePartialRendering = "true | false"
    AsyncPostBackTimeout = "seconds"
    AsyncPostBackErrorMessage = "message"
    AllowCustomErrors = "true | false"
    OnAsyncPostBackError = "handler"
    ScriptMode="Auto | Inherit | Debug | Release"
    ScriptPath = "path">
    <Scripts>
        <!– Szkript referenciák megadása -->
    </Scripts>
    <Services>
        <!– Webszolgáltatás referenciák megadása -->
    </Services>
</asp:ScriptManager>
```

Részleges oldalfrissítés

- Teljes oldal frissítése helyett csak a kijelölt régiók frissülnek
  - Alapértelmezés szerint engedélyezve van.
  - Ha letiltjuk vagy a böngésző nem támogatja, akkor teljes frissítést hajt végre
- Előre elkészített böngésző független kliens oldali szkriptek
- Minden szerver oldali esemény lefut, de csak az UpdatePanel tartalma kerül vissza a kliensnek

Részleges renderelés



Click

Form Submit

Form Data +

Custom Header

PageRequestManager

Partial Rendering Response

Init

Load State

Process Postback

Load

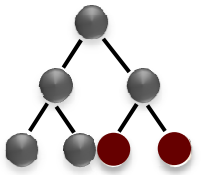
Postback Events

Save State

PreRender

Render

Unload



PageRequestManager

- Böngészőben frissítés
- Kliens oldali életciklus
- Több aszinkron művelet prioritása
- Státusz üzenet a futó kérés esetén
- Egyedi hibaüzenet
- Request / Response objektumok elérése

Mi az UpdatePanel?

- Egységbe foglalja az aszinkron módon frissítésre kerülő részeket
  - Automatikusan aszinkron callback-ké alakítja a postback-et
  - Automatikusan frissíti a tartalmat a callback eredménye alapján
  - Nem kell hozzá ismerni se a JavaScript-et se az XMLHttpRequestet

UpdatePanel

```
<asp:ScriptManager ID=”…” runat=”server” EnablePartialRendering=”true” />
…

<asp:UpdatePanel ID = ”…” Runat = ”server”
   UpdateMode = ”Always | Conditional” ChildrenAsTriggers = ”true | false” >

   <Triggers>
      <!-- Itt lehet Triggereket definiálni -->
   </Triggers>

   <ContentTemplate>
      <!-- Tartalmat itt kell definiálni -->
   </ContentTemplate>

</asp:UpdatePanel>
```

Triggerek

- AsyncPostBackTrigger
  - Postback → aszinkron callback
  - UpdatePanelen kívüli eseményre frissíteni a panel tartalmát
- PostBackTrigger
  - UpdatePanelen belül postback legyen

UpdatePanel.Update()

- A panel tartalmát kódból frissítjük
  - Csak ha az UpdateMode=”Conditional”
- Tipikus példa
  - Több UpdatePanel láncszerű frissítése

```
protected void OnClick ( object sender, EventArgs e )
{
        // Click esemény elsütése az UpdatePanelben
        // Másik UpdatePanel frissítése
        OtherUpdatePanel.Update()
}
```

UpdateProgress

- Az UpdatePanel kiegészítő eleme
  - Jelzi a felhasználónak, hogy aszinkron frissítés folyik a háttérben
- Automatikusan megjelenik, amikor a frissítés elkezdődik
  - Vagy adott idővel később (DisplayAfter)

UpdateProgress

```
<asp:UpdateProgress ID = ”…” Runat = ”server”
        DisplayAfter = ”milliseconds”
        DinamicLayout = ”true | false”
        AssociatedUpdatePanelID = ”UpdatePanelID” >
        <ProgressTemplate>
                <!-- Itt kell megadni az UI-t -->
        </ProgressTemplate>
</asp:UpdateProgress>
```

```
<asp:UpdateProgress ID = ”…” Runat = ”server”
        DisplayAfter = ”500”
        <ProgressTemplate>
                <asp:Image ID=”…” runat=”server”
                        Url=”~/Images/SpinningClock.gif” />
        </ProgressTemplate>
</asp:UpdateProgress>
```

Kliens oldali események

- Kliens oldalon kiváltott események
  - initializeRequest
  - beginRequest
  - pageLoading
  - pageLoaded
  - endRequest

```
Sys.Application.add_load(ApplicationLoadHandler);
function ApplicationLoadHandler(sender, args)
{
    Sys.WebForms.PageRequestManager.
        getInstance().add_initializeRequest(CheckStatus);
}
```

Frissítés megszakítása

```
// PageRequestManager példány elkérése
var prm = Sys.WebForms.PageRequestManager.getInstance();

// Ha aszinkron kérés van folyamatban,
// és a Cancel gombra kattintottunk, akkor abort
if (prm.get_isInAsyncPostBack()
    && args.get_postBackElement().id == 'CancelRefresh')
{
    prm.abortPostBack();
  }
}
```

Összefoglalás

- Új alkalmazás létrehozásánál azonnal használható
- Megoldja az állandó postback-elést
- Nem kell hozzá ismerni a JavaScript-et
- Egyszerű
- Testreszabható

Egyszerű AJAX oldal készítése

Egyszerű demo

- Részleges frissítés
- UpdatePanel
- Trigger

Globalizáció és lokalizáció

Fogalmak

- Globalizáció
  - Globalization (g11n), internationalization (i18n)
  - Nincs kultúrafüggő rész
- Lokalizáció
  - Localization (L10n)
  - Honosítás, adaptálás
- „Glokalizáció”
  - „Glocalization”

Kultúra információk

- Nyelv
- Ország
- Számformátum
- Dátumformátum
- Időformátum
- Időzóna
- Naptár
- Pénzösszegek formátuma
- Mértékrendszer
- ABC
- Karakterek típusa
- Szöveg iránya
- Gyakori szövegek formátuma
- Papírméret
- Képek és színek

Sok szabvány

- ISO 639
  - 639-1: 2 karakteres, 639-2: 3 karakteres nyelv kód
  - 639-3: 639-2 + kiegészítések, 2009.
- ISO 3166-1:
  - alpha-2: 2 karakteres, alpha-3: 3 karakteres terület kód
- ISO 4217: 3 karakteres pénznem kód
- RFC 1766:
  - <2-3 kisbetűs nyelv (ISO 639)>-<2 nagybetűs ország (ISO 3166)>
- National Language Support (NLS) identifier: locale identifier (LCID)

Hol számít

- Felhasználói felület
  - Kultúrafüggő
  - Honosított UI, kimenet formázása, bemenet feldolgozása
- Kommunikáció
  - Kultúrafüggetlen
- Adatkezelés, tárolás
  - Több nyelvi változat tárolása
  - Rendezés, szűrés

Mit ír ki? Hol a hiba?

```
static void Main( string[] args )
{
  string path = @"file://c:\windows";
  Console.WriteLine( IsFileUrl( path ) );
}


// Meghatározza, hogy fájl vagy web URL-e.
public static bool IsFileUrl( string path )
{
  // String.Compare(strA, indexA, strB, indexB, length, ignoreCase)
  return ( String.Compare( path, 0, "FILE", 0, 4, true ) == 0 );
}
```

Turkish-I probléma

- Sztringek összehasonlítása kultúrafüggő
- Török (tr-TR) azeri (az-Latn-AZ):

English vs. Turkish Case Mappings

Language	Letter	Lowercase Map	Uppercase Map
English	i	i	I
Turkish	dotted i	i	İ
Turkish	dotless ı	ı	I

```
return ( String.Compare( path, 0, "FILE", 0, 4, true,
                         CultureInfo.InvariantCulture ) == 0 );
```

Kultúra információk .NET-ben

- Kultúra definíció:
  - Invariáns (invariant)
    - Kultúra független (angol)
  - Semleges (neutral)
    - Nyelvhez tartozik, országhoz nem (pl. "en")
  - Specifikus (specific)
    - Nyelv- és országfüggő (pl. "en-GB")

Kultúra azonosítása

- System.Globalization.CultureInfo
  - CultureInfo.InvariantCulture
- RFC 1766:
  - <2-3 kisbetűs nyelv (ISO 639)>-<2 nagybetűs ország (ISO 3166)>
  - pl. en-US, de-AT, hu-HU.
- National Language Support (NLS) identifier, locale identifier (LCID).
  - English, United States: 0x0409 (1033);
  - Hungarian, Hungary: 0x040e (1038).

Kultúra felhasználása

- Current Culture
  - Dátumok, számok stb. formázására
  - CultureInfo.CurrentCulture = Thread.CurrentThread.CurrentCulture
- Current UI Culture
  - Erőforrás fájlok kezelésére
  - CultureInfo.CurrentUICulture = Thread.CurrentThread.CurrentUICulture
- Célszerű együtt állítani

Formázás

- Kimenet formázása
  - IFormattable.ToString(string format, IFormatProvider formatProvider)
  - format:
    - Standard szimbólumok („C” – currency)
    - Egyedi szimbólumok („yyyy-MM-dd”)
  - IFormatProvider:
    - CultureInfo
- Bemenet feldolgozása
  - Parse metódusok
  - IFormatProvider megadása célszerű

Erőforrás fájlok

- Előnyök:
  - Alkalmazástól függetlenül fordítható
  - Új erőforrás fájl → új nyelv (?)
- Hátrányok:
  - Lassít
  - Elválik a szöveg a környezettől → félrefordítás
  - Hosszabb üzenet több rekordokban → félrefordítás
  - Kontrollok és szövegek mérete, pozíciója változik
  - Nem tárolható minden (pl. rendezési sorrend)
  - Fájlok szinkronban tartása

Erőforrások kezelése .NET-ben

- .resx fájlok – XML
- Elnevezési konvenció:
  - akarmi.resx – alapértelmezett szerelvény locale
  - akarmi.en.resx, akarmi.hu.resx stb.
- Feldolgozás:
  - ResourceManager
  - ResourceReader, ResourceWriter – alacsony szinten
- Visual Studio támogatás
  - Erőforrás szerkesztő
  - Automatikus fordítás: Resources osztály

Windows Forms

- Visual Studio:
  - Localizable → Language tulajdonság
  - Automatikus erőforrás készítés
  - Fordításkor szatellit szerelvények

ASP.NET

- Culture és UICulture megadható
  - @Page vagy web.config
  - Állítható kódból is
- Lokális erőforrások: minden oldalhoz
  - App_LocalResources mappa
  - meta:resourcekey attribútum
- Globális erőforrások: teljes alkalmazáshoz
  - $Resources resource expression
  - GetGlobalResourceObject() metódus

Kódolás (encoding)

- Karakter
  - Absztrakt fogalom, fizikailag bináris forma
- Műveletek:
  - Encoding: karakter → bájtok
  - Decoding: bájtok → karakter
- Kódlap (codepage)
  - Kapcsolat a karakter és a bitsorrend között

Kódolási típusok

- ASCII: 7 bit
  - Csak a kódtábla alsó felét tudja ábrázolni
- UTF-7 (Unicode Transformation Format)
  - 7 bitre kódol, azonban egy karaktert akár két bájton is ábrázolhat
- UTF-8: változó hosszúságon kódol
  - ASCII kompatibilis
  - 0xEFBBBF prefix → "ï»¿"
- UTF-16: két bájt (surrogate pairs) →
  Byte-order mark (BOM):
  - 0xFFFE: little endian (kisebb bájt előbb)
  - 0xFEFF: big endian (nagyobb bájt előbb)

Kódolás .NET-ben

- String: UTF-16
- System.Text.Encoding
  - ASCIIEncoding
  - UnicodeEncoding (UTF-16)
  - UTF7Encoding
  - UTF8Encoding
  - GetByteCount(), GetBytes(), GetString()
- System.Convert
  - ToBase64String(), ToBase64CharArray()
  - FromBase64String(), FromBase64CharArray()

Megjelenítési réteg

Mobil kliens

Az előadás felépítése

- A mobil kliens
- A Python nyelv alapjai
- Prototípus-fejlesztés Python nyelven mobil eszközökre

A mobil kliens - Áttekintés

- Mit tud a mobil?
- Mobil platformok...
- ...Fejlesztői szemmel

A mobil kliens felhasználói szemmel

Mobil telefonok, smartphone-ok, PDA-k

- Vannak
  - Gyakorlatilag mindenkinél
- Sokféle gyártó, sokféle felület
  - A saját mobiljával mindenki elboldogul

A mobil kliens felhasználói szemmel

Elvárások

- Lehessen vele „telefonálni”

A mobil kliens felhasználói szemmel

Elvárások

- Lehessen vele „telefonálni”
  - Sokat
  - Sokszor
  - Kényelmesen
  - Ha mégsem lehet, valami akkor is legyen

A mobil kliens felhasználói szemmel

Elvárások

- Lehessen vele „telefonálni”
  - Sokat
    - Fejlett energiagazdálkodás kód szintjén is ügyelni kell rá
  - Sokszor
    - Magas rendelkezésre állás
    - Erőforrásgazdálkodás

A mobil kliens felhasználói szemmel

Elvárások

- Lehessen vele „telefonálni”
  - Kényelmesen
    - PIM funkciók (telefonkönyv, naptár, notesz, todo)
    - Alkalmazások magas fokú integráltsága
    - Specifikus UI
  - Ha mégsem lehet, valami akkor is működjön
    - Bármikor megszakadhat a kapcsolat

A mobil kliens fejlesztői szemmel

- Tudnak
  - Megközelítőleg 10 évvel ezelőtti PC-k műszaki színvonala
    - 100-400 MHz RISC processzor (többnyire ARM)
      - Gyors működés, gyakran multitask
    - 4-30 MByte operatív memória
    - 30-200-8000 MByte háttértár

A mobil kliens fejlesztői szemmel

- Tudnak
  - Sokféle, vezeték nélküli adatkapcsolat
    - GSM (CSD, GPRS, EDGE, 3G, stb.)
    - BT, IrDA
    - WLAN
  - Elfogadható prezentációs képességek
    - Színes, QVGA-körüli felbontású kijelző
      - Akár HW 3D gyorsítással
    - Sztereo hang

A mobil kliens fejlesztői szemmel

- Kicsik
  - Memória
    - Néhány 10 MB
    - Sok párhuzamosan futó alkalmazás
    - VM nincs
  - Processzor
    - Néhány 100 MHz, RISC architektúra
  - Akkumulátor
    - Az egyik leglassabban fejlődő technológia

Mobil platformok

Elterjedt mobil platformok

- Böngésző
- Java Micro Edition
- Symbian OS
- Windows Mobile
- Linux

Kevésbé elterjedt mobil platformok

- Blackberry
- Brew
- iPhone
- Android

Mobil platformok

Böngésző

- Gyakorlatilag minden mobil készülékben van
  - Létezik „3rd-party” is: Opera Mobile / Mini
- Általában jól használható, az adott készülék kezelési sajátosságaihoz illeszkedő UI logika
- HTML, XHTML, WML támogatás a szokásos

Mobil platformok

Böngésző

- Dinamikus viselkedéshez
  - JavaScript gyakran
  - Java (appletekhez) ritkán
  - FlashLite (Adobe Flash mobilra) újabban
    - Symbian
    - Windows Mobile
    - Opera Mobile (jelenleg csak erre a kettőre van ☺)

Mobil platformok

Java Micro Edition (JavaME, nemrég J2ME)

- Ez is gyakorlatilag mindenhol van
- Applet-szerű alkalmazások
  MIDlet (Mobil Information Device)
  - UI, számítási kapacitás, kommunikáció, lokális adattárolás, alkalmazás-életciklus
  - Mobil eszközökre optimalizált
  - Opcionális JSR-ekkel bővül

Mobil platformok

Symbian OS

- Talán a legrégebbi
- A legelterjedtebb SmartPhone platform (80% feletti részesedés)
- Önálló operációs rendszer
  - Általános OS szolgáltatások
  - SmartPhone-specifikus szolgáltatások
    - PIM (Personal Information Management)
      - RDBMS
    - DFRD-k (Device Family Reference Design)

Mobil platformok

Symbian OS

- Sokféleképpen megszólítható
  - C++ (natív API, C++-ban készült az OS)
  - JavaME (régebben PersonalJava)
  - FlashLite
  - Python – szkriptnyelv
  - HTML/XHTML/WML + JavaScript

Mobil platformok

Windows Mobile (PocketPC)

- Windows, kicsiben
  - Régebben Windows CE + testreszabott Windows API
  - Manapság .Net CF
- Önálló OS, SmartPhone-ok és PDA-k számára
  - Általános OS szolgáltatások
  - SmartPhone-specifikus szolgáltatások
    - PIM funkciók, Windows look'n'feel
  - SQL server Embedded Edition

Mobil platformok

Linux

- Nem kifejezetten elterjedt
- Azon belül is sokfélék
  - Néhány telefonon, jelenleg zárt platformként
    - Van SmartPhone, ennek a nyílttá tételét ígérik
    - De van kifejezetten olcsó, belépő szintű eszköz is (ennek nem)
- Maemo: teljes OS, touchscreenes PDA-kra
  - GNOME alapok különféle módosításokkal (GTK+ widgetek bővített-módosított változata Hildon néven)
  - Matchbox WM
  - Projektek néhány szisztematikus változtatás után fordíthatóak rá
- OpenMOKO
- Trolltech Qtopia Greenphone

Mobil platformok

További fontos mobil platformok:

- BlackBerry: elsősorban a push-email szolgáltatásáról híres, de ettől még teljes mobil platform, üzleti felhasználók körében népszerű
- Brew: a Qualcomm mobil platformja, elterjedt lehetne (~50 partner köztük olyanok, mint Toshiba, Sharp, Samsung, LG, Lenovo, Kyocera, Hitachi, Benq, Casio), Európában mégsem az
- iPhone: MAC OS X, jelenleg még zárt platform
- Android: a Google egyelőre csak SDK formájában létező platformja

A fejlesztő lehetőségei

Böngésző

+:

- Vékony kliens
- Mindenhol van
- Könnyű rá fejleszteni, különösen, ha nem cél az igényesség

A fejlesztő lehetőségei

Böngésző

-:

- Elhanyagolható számítási teljesítmény
- Korlátozott adatfeldolgozási lehetőségek
- Szinte nem létező lokális adattárolási lehetőség
- Korlátozott kommunikációs lehetőségek
- Sokféleség
  - Egy adott eszközön is többféle böngésző lehet, többféle képességekkel, és persze hibákkal
  - HTML a közös pont, DHTML (JavaScript), FlashLite esetleges

A fejlesztő lehetőségei

JavaME

+:

- Szinte mindenhol van
- Elfogadható számítási és adatfeldolgozási teljesítmény
- Már van valamennyi lokális háttértár
- Sokféle kommunikációs protokoll elérhető
- Szebb és használhatóbb felület alkotható benne a böngészőhöz képest
- Elég könnyű elsajátítani, illetve fejleszteni rá

A fejlesztő lehetőségei

JavaME

-:

- A MIDP alkalmazások nem jól integrálódnak az eszközön megszokott felületbe
  - Nem úgy néz ki
  - Mindig lassabb valamennyivel
- Különféle, készülék-specifikus korlátozások
  - Háttértár, operatív memória
- JSR-ek vagy vannak, vagy nincsenek. Pl. Web-Service elérésnél ez nem mindegy (socket persze mindig van)

A fejlesztő lehetőségei

Symbian

+:

- Az eszköz teljes kapacitása kihasználható (memória, háttértár, kommunikáció, számítási teljesítmény)
- Natív alkalmazás (C++) esetén az adott készülékre jellemző look'n'feel 100%-ig elérhető
- Bármi mást (JavaME, Python, weblap) egyszerű(bb) fejleszteni

A fejlesztő lehetőségei

Symbian

-:

- Natív alkalmazás készítése nehéz
  - Összetett API
  - Rémes dokumentáció
  - Különös fejlesztői eszközök (toolchain, IDE-k)
- A nem natív alkalmazások itt sem illeszkednek jól a felülethez

A fejlesztő lehetőségei

Windows Mobile

+:

- Az eszköz teljes kapacitása kihasználható (memória, háttértár, kommunikáció, számítási teljesítmény)
- Az adott készülékre jellemző look'n'feel gyakorlatilag 100%-ig elérhető
  - Ebben azért annak is szerepe van, hogy nem túl sokfélék az ilyen eszközök
- Kiemelkedő fejlesztői támogatás
  - MSDN
  - .Net-es nyelvek és JavaME

A fejlesztő lehetőségei

Windows Mobile

-:

- Kevésbé elterjedt smartphone-platform
  - Vannak, akik például elvből utálják

A fejlesztő lehetőségei

Linux

+:

- Az eszköz teljes kapacitása kihasználható (memória, háttértár, kommunikáció, számítási teljesítmény)
- Az adott készülékre jellemző look'n'feel gyakorlatilag 100%-ig elérhető (többnyire C/C++, ritkán Java API)

A fejlesztő lehetőségei

Linux

-:

- Kevéssé elterjedt platform, még a Windows Mobile-hoz képest is
- Gyakran csak lelkesedés-alapú fejlesztői támogatás
  - Maemo/OpenMOKO/Qtopia nyitott platformok
  - A többit viszont előbb fel kell törni

Összegzés

Mobil telefonokra lehet szoftvert fejleszteni, nem is kizárólag csak vékony klienst

- Elterjedtségükhöz nem férhet kétség
- A prezentációs rétegben szükséges tulajdonságok (számítási teljesítmény, UI) rendelkezésre állnak
  - Adatbevitelre csak mérsékelten alkalmasak
- Hordozhatóságuk alkalmazástól függően kulcsfontosságú tulajdonság lehet

Elosztott adatbázisok

Definíció

- Logikailag egyetlen adatbázis, amely több számítógépen van tárolva (több helyen), és hálózattal vannak összekötve.
- Központilag adminisztrált
- A helyi példányok külön-külön rugalmasak
- Nagy területet is befoghatnak

Miért van szükség elosztott adatbázisokra?

- A vállalatok szervezése is elosztott
  - helyi adatok
  - közös adatok
- Adatátvitel költsége, és megbízhatósága
  - nagymennyiségű tranzakció
  - megbízhatatlan kapcsolat
  - adatok közel a felhasználóhoz

Az elosztott adatbáziskezelők céljai

- Helyfüggetlen
  - a felhasználó nem tudja, hogy fizikailag hol az adat
  - táblák különböző helyekről összekapcsolhatók
- Helyi autonómia
  - ha nincs háló, akkor működjön magában
  - saját biztonsági rendszer, naplózás, felépülés, teljes hozzáférés a helyi adatokhoz

Az elosztott adatbáziskezelők előnyei 1.

- Nagyobb megbízhatóság, és elérhetőség
  - alacsonyabb szinten tovább működhet, ha egy csomópont kiesik
  - függ az adatelosztástól
- Helyi irányítás
  - jobb helyi adatintegritás
  - távoli adatok elérhetők, ha szükséges
  - hardver a helyi feldolgozási igényekhez igazodhat

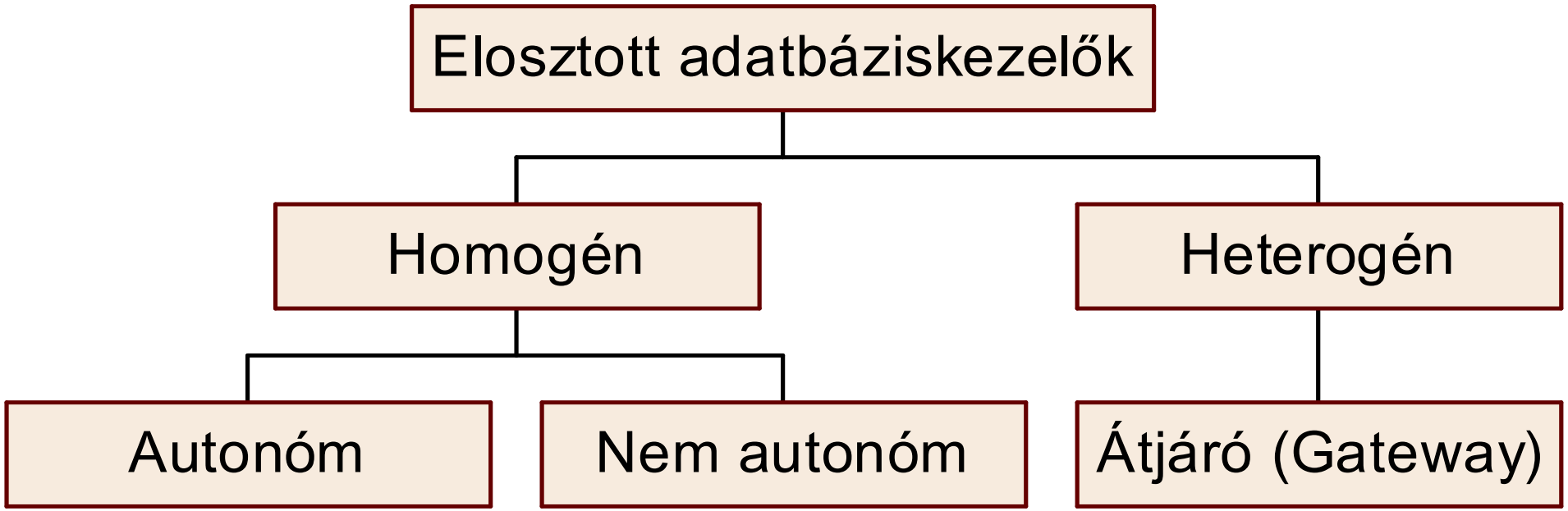
Az elosztott adatbáziskezelők előnyei 2.

- Moduláris növekedés
  - új hely vagy csoport beállítása - helyi számítógép
  - kisebb hatás a már létező felhasználókra
- Kisebb kommunikációs költségek
- Gyorsabb válaszidő

Az elosztott adatbáziskezelők hátrányai

- Szoftver költség, és komplexitás
- Megnövekedett feldolgozási feladat
  - különböző csomópontok koordinálása
- Adatintegritás biztosítása nehezebb
- Lassú válaszidő
  - ha rossz elosztás

Elosztott adatbázisok típusai



Szinkron és aszinkron rendszerek

- Szinkron elosztott adatbáziskezelők
  - minden példány naprakész
  - lassú
- Aszinkron elosztott adatbáziskezelők
  - helyi adatok elérhetők önmagukban
  - átmeneti inkonzisztenica elfogadható
  - módosítások szinkronizálása bizonyos időközönként

Alkalmazott módszerek

- Replikáció
- Vízszintes particionálás
- Függőleges particionálás

Vízszintes particionálás

- Egyes sorok egyik helyen
- Más sorok másik helyen
- Tranzakciók abban a fiókban, ahol a számla van

Számlaszám	Név	Fiók	Egyenleg
200	Kovács	Belváros	100,000
342	Harczi	Belváros	2,221,457
231	Hotek	Belváros	321,321

111	Litvai	Újpest	211,121
321	Lató	Újpest	11,321

Vízszintes particionálás előnyei és hátrányai

- Előnyök
  - hatékonyság: adat közel a felhasználóhoz
  - helyi optimalizáció
  - biztonság: csak azok az adatok, amelyek érdekesek egy adott helyen
  - lekérdezés eredménye egyszerű unió
- Hátrányok
  - inkonzisztens elérési idő
  - hiba esetén nem térhetünk át egy másik csomópontra

Függőleges particionálás

- Oszlopok szerint osztjuk fel a relációt
- Két részleg
  - mérnöki
  - gyártás

Azonosító	Név	Ár	Terv-szám	Mennyiség
P11	Skoda hengerfej	12,000	T123	10
P32	Skoda dugattyú	14,110	T221	5
P12	Skoda első lámpa	7,320	T11	3

Függőleges particionálás jellemzői

- Előnyök
  - ugyanaz, mint vizszintesnél
  - de a lekérdezés eredménye: join
- Hátrányok
  - ugyanaz, mint vízszintesnél

Replikáció

- Előnyök
  - megbízhatóság
  - gyors válasz
  - elosztott lekérdezések nincsenek
  - nem kell koordináció más csomóponttal, csak szinkronizációkor
  - szinkronizáció történhet akkor, amikor kevés a forgalom a hálón

Replikáció

- Hátrányok
  - Tárolási szükséglet
  - Komplexitás, adatmódosítások ára
- Általában csak olvasható adatbázisok esetén használják
  - katalógus, telefonkönyv, menetrend
- Nem jó tranzakciófeldolgozó rendszerekben
  - repülőgép helyfoglalás

Pillanatkép-replikáció

- Tábla adatainak másolása
  - Szűrés megadható
- Teljes másolás
- Differenciális másolás

SQL Server Replikáció

- Publisher
  - Adat „tulajdonosa”
  - Article-t készít minden módosításról, ezeket juttatja el a Distributorhoz
- Subscriber
  - Replikáció előfizetője
  - Módosításokat megkapja a Distributortól
- Distributor
  - Article-k összegyűjtése
  - Article-k eljuttatása az előfizetőkhöz

Replikáció ütemezése

- Push
  - Disztributor kezdeményezi
- Pull
  - Subscirber kezdeményezi
- „Real-time” replikáció
  - Disztibutor kezdeményezi
  - Ha érkezik új article → replikációs folyamat indítása

Módosítható replikáció

- Adatmódosítás az előfizetőnél
- Elosztott tranzakcióban módosítás a kiadónál is
- Módosítás hatására article keletkezik
  - Többi előfizető replikáción keresztül értesül a változásról

Mikor használjunk replikációt?

- Megengedhető a késleltetett módosítás
- DBMS lehetőségei
  - ha nem lehet több csomóponton módosítani, akkor csak ez van
- Teljesítmény - akkor szinkronizáljunk, amikor nincs sok helyi munka
- Heterogén rendszerek - adattranszformáció
- Kommunikáció - nincs szükség egy kitüntetett kapcsolatra

Elosztott tranzakciók

- Kétfázisú commit
- Első fázis: felkészülés
  - el tudod végezni?
- Ha mindenki igen, akkor második fázis végrehajtás
  - végezd el