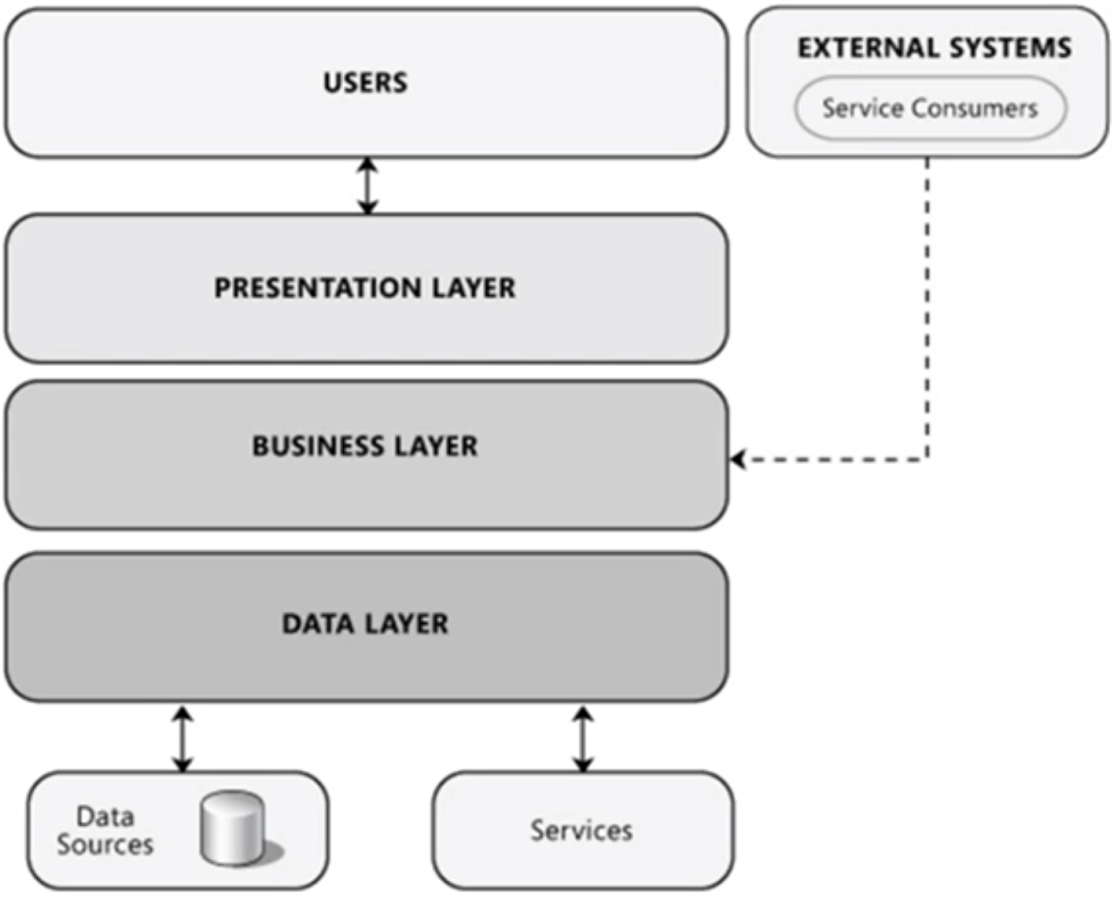
Adatvezérelt rendszerek - 1. előadás

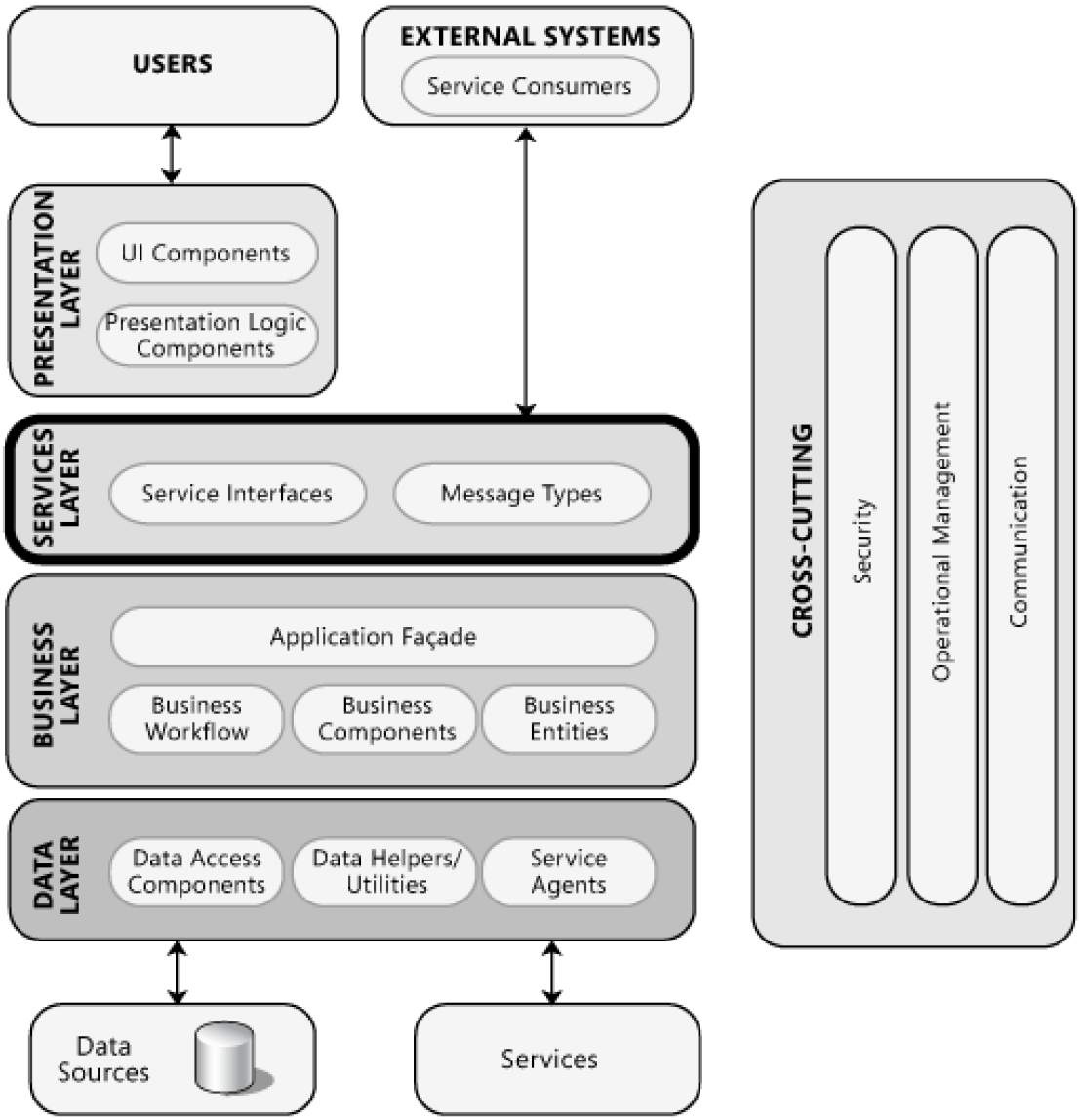
Bevezetés

- **céljuk**: Adat tárolása, és ezen adat elérhetővé tétele, adatot manipulálnak (a manipulációt befolyásolja maga az adat)
- **adatvezérelt** (data driven) -> pl: facebook, twitter, stackoverflow, neptun, booking.com, airbnb
- **folyamatvezérelt** (control driven) -> pl egy számítógépes játék
- félévben a példa → webshop
    - fő adat: egy termék, vásárlás, számla
    - megvannak a szabályok: mikor lehet rendelni, mikor megy át egy rendelés

Háromrétegű architektúra



- **Presentation Layer** (*Megjelenési réteg*) → a UI, pl a holnap mint kezelőfelület
- **Business Layer** (*Üzleti logikai réteg*) → üzleti szabályok, pl csak oktató írhat be jegyet viszgaidőszakban
- **Data Layer** (*Adat réteg*) → perzisztens tárolás (adatbázisokban, de ebbe egyéb részek is beletartoznak)
- ezen rétegek sokszor külön gépen futnak, rétegek mentén szétbontva (layer - egy gépen fut, tier - több gépen fut)

- 

Data Layer - *Adat réteg*

- Perzisztenciáért felelnek → data source - adatforrás (pl relációs adatbázis, google drive)
- Akkor hívjuk ***Adat réteg***-nek, ha nem tartalmazza az adatbázist

Data Access Layer - *Adatelérési réteg*

- **Data Access Layer** (*Adatelérési réteg*) → Ha az adatbázist vagy adatforrást is beleértjük
- pl csatoljuk hozzá ezt a fájlt az email-hez
- szolgáltatásként nyújtja az adatok manipulálásának módját

Business Layer - *Üzletilogikai réteg*

- **Üzleti entitások** (*Business Entities*) → főbb elemek, amiket manipulálunk (megrendelés, termék, kurzus, vizsga)
- **Üzleti komponensek** (*Business Components*) → egyszerűbb műveletek (pl vizsgajegy beírása - ezzel manipuláljuk a vizsgajegyet)
- **Üzleti folyamtok** (*Business Workflow*) → több lépéses, folyamatlépés folyamat (pl rendelés véglegesítése - sok lépésből áll, entitások halmazát manipuláljuk)
- **Szolgáltatási interfész réteg** (*Services layer*) → üzleti logika funkcióit a felhasználói réteg felé szolgáltatásként biztosítja (külön réteg, mert helyzetfüggő a megvalósítási módja)

Presentation layer - *Megjelenítési réteg*

- **UI Components** (*Felhasználói felület komponensek*) → pl weblap, asztali vagy mobil alkalmazás
- **Presentation Logic Components** (*Megjelenési logikai komponensek*) → pl keresés, szűrés
- feladatok:
  - adatok értelmes megjelenítése
  - egyéb funkcióra lehetőségek: keresés, szűrés
  - lokalizáció (dátumok, pénzek, ezek felhasználó függően kerüljön kiírásra)
- a megjelenítési réteg nem nagyon gondolkozik, azt az üzletilogikai réteg csinálja

Cross-cutting - *Rétegfüggetlen alkalmazások*

- Minden rétegben megjelenő közös aspektusok:
  - biztonság → bejelentkezés (aki belép, mit is csinálhat)
  - operational management (üzemeltetés szolgáltatás) → naplózás, hibakeresés, audit naplózás (rögzítjük, hogy ki mit csinál a rendszerben), konfigurációkezelés, hibák naplózása
  - kommunikáció → rétegek sokszor külön gépen vannak, ezek közt kommunikálni kell
    - lehet szinkron és aszinkron is, adatbázissal általában szinkron, felhasználói felületről néha aszinkron

Összegzés

- backend → minden, ami a megjelenési réteg alatt van, a többi frontend
- sokszor kód szinten is megtörténik a szétválasztás

Tranzakciók

Konkurens adathozzáférés (az adatbázis tekintetében)

- **Def:** Egyazon adategységhez (pl egy tábla egy rekordja) egy időben <u>többen férnek hozzá</u>, és legalább <u>egyikük módosítja</u>.

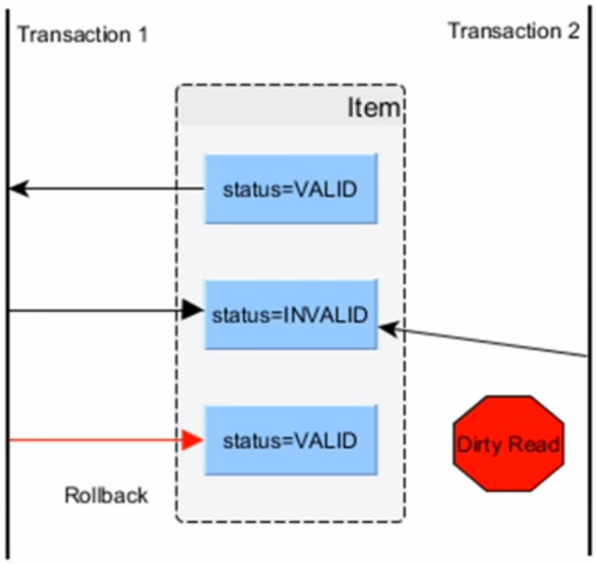
Tranzakció fogalma

- A feldolgozás logikai egysége, olyan műveletek sorozata, melyek csak együttesen értelmesek
- Pl: rendelés véglegesítése → kosárban levő mind a 15 termékre raktárkészletet megnézni, csökkenteni az ottani számokat, elmenteni az adatbázisba, véglegesíteni
- Alaptulajdonságok:
  - **Atomicity** (*Atomi*) → oszthatatlanság, a műveletek sorozatát egyben csináljuk végig, nem lehetnek részeredmények (vagy végig csinálja, vagy semmit nem csinál vele → atomi, oszthatatlan)
  - **Consistency** (*Konzisztencia*) → konzisztens állapotból konzisztensbe megy (közben érinthet inkonzisztenst)
  - **Isolation** (*Izoláció*) → úgy tud végig menni a műveletek sorozatán, mintha a rendszerben egyedül lenne (a rendszer biztosítja, hogy az átlapolódásból ne legyen baj)
    - T1(A = 12, C = A+2), T2(A = 15). Ha T1 közben T2 végrehajtódik, T1 értéke nem az elvárt lesz

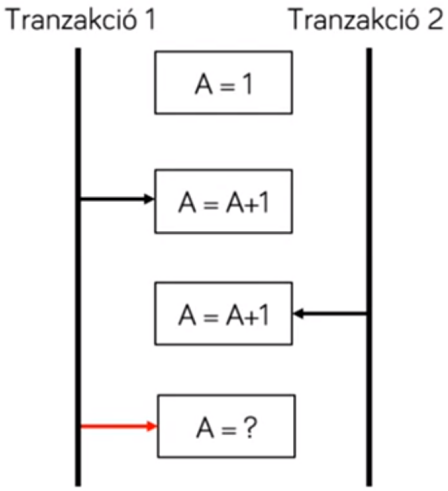
- **Durability** (*Tartósság*) → tranzakció végén disk-re kiírva van az adat (nem csak memóriában)

Izolációs alapproblémák

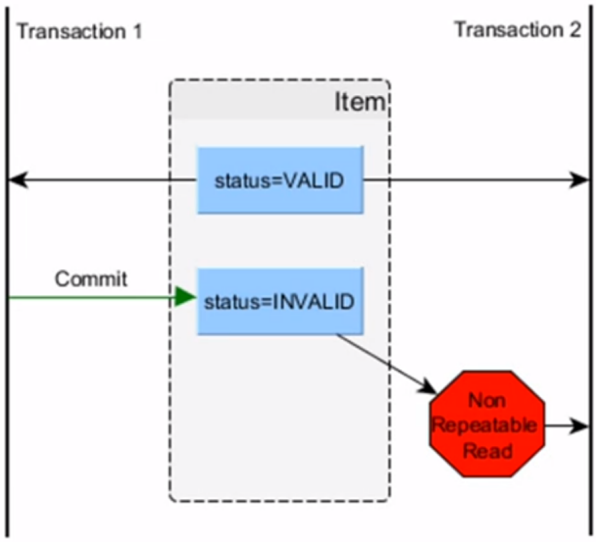
- sok párhuzamos tranzakció
- izoláció: "úgy kell végrehajtani, mintha egymás után történnének és nem párhuzamosan"
  - de párhuzamosan futnak a tranzakciók
- 4 alap probléma:
  - **piszkos olvasás** (*dirty read*) → T1 elkezd futni, de abortál, így T2 egy nem commitált tranzakció által módosított értéket olvasott ki



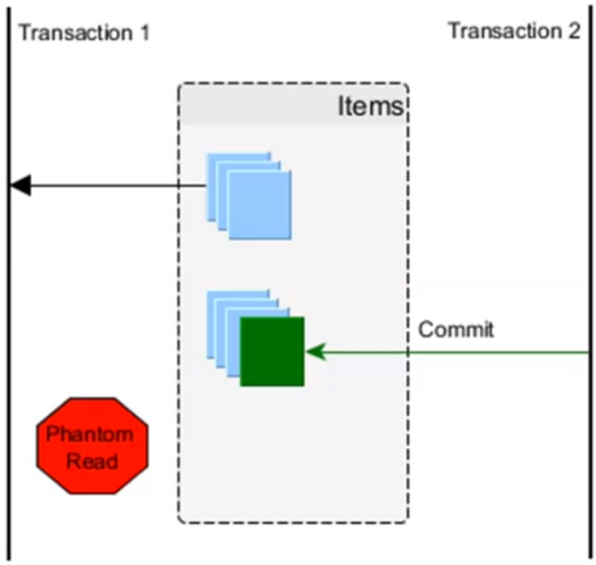
  - **elveszett módosítás** (*lost update*) → T1 és T2 is módosította, majd T1 olvasná, és nem azt látta amire ő írta át



  - **nem megismételhető olvasás** (*non-repeatable read*) → időben 2x kiadva ugyanazt a lekérdezést, más eredményt kap T2



- **fantom rekordok** (*phantom read*) → rekordhalmazoknál kerül elő: T2 olyan rekordot módosít, ami benne van T1 által épp olvasott rekordhalmazban



- **Fontos**: nem csak relációs adatbázisokban, hanem minden többfelhasználós, elosztott rendszerekben megjelenhetnek!

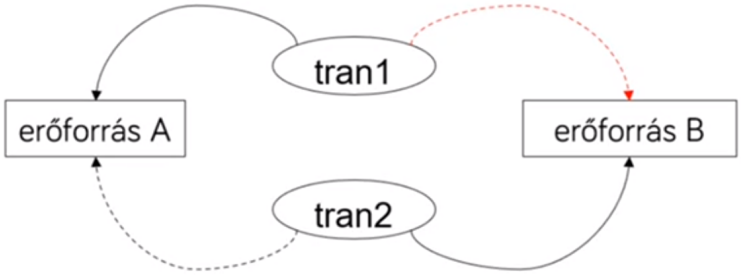
Izolációs szintek SQL szabvány szerint → ezeket a rendszer garantálja, de nekem kell jeleznem!

- **Read uncommitted** → mind a 4 probléma előfordulhat
- **Read committed** → nincs piszkos olvasás (ez az alap szint)
- **Repeatable read** → nincs piszkos olvasás, se nem megismételhető olvasás
- **Serializable** → egyik probléma se fordulhat elő

Tranzakciók ütemezése (megoldás az izolációs alapproblémákra)

- Csak olyan műveletek engedhetők meg, melyek nem sértik a helyes ütemezést
- Ha sérülne a helyes ütemezés, akkor a tranzakció vár
- probléma: nem tudjuk előre, hogy melyik tranzakció mit akar csinálni (csak futás közben látjuk)
- Olyan ütemezés megengedett, mely konfliksekvivalens egy soros ütemezéssel
  - azaz van egy olyan átrendezése a tranzakcióknak, ahol soros ütemezés teljesül

Ütemezés biztosítása

- **Kétfázisú zárolás (2PL)** → ha egy tranzakció hozzá akar férni egy erőforráshoz, arra zárat rak, majd ha elvégezte a dolgát (commit), leveszi a zárat.
  - probléma: **holtopont** (*deadlock*)
  - 

  - Ha **Serializable**-t használunk, akkor gyakran fordul elő holtpont → csökken a hatékonyság