Többrétegu alkalmazások

Készítette: Vastag Attila

2018

Bevezetés

Többfelhasználós rendszereknél különösen fontos a pontos tervezés, és kivitelezés. Az alkalmazás hibája egyszerre több ember munkáját akadályozza. Egy ilyen rendszernél gondolni kell a továbbfejlesztésre is, mert intenzív használat mellett potenciálisan új igények is felmerülnek.

Gondoljunk például egy pénzügyi szoftverre. Az adózási, számviteli szabályok állandóan változnak, ez gyakorlatilag folyamatosan üzemeltetési feladatokat generál. De ugyanilyen feladatok merülhetnek fel logisztikai, vagy statisztikai területen is.

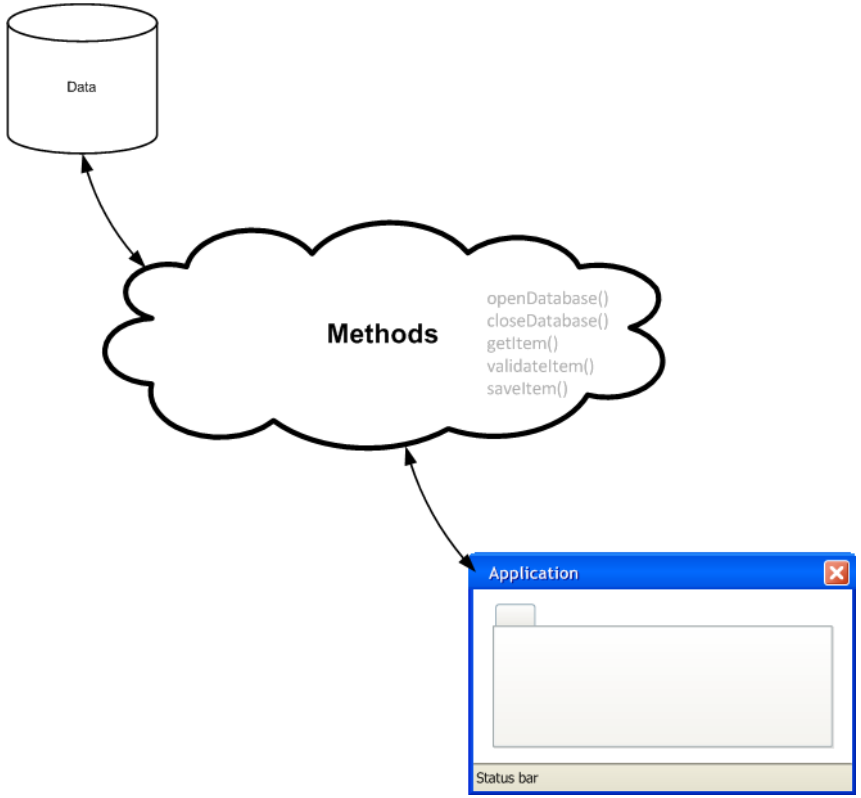
Cikkemben a többrétegű technológia előnyeit próbálom felvázolni az egyrétegű technikával szemben. Ez a technológia biztosítani tudja a rendszer integritását, átláthatóságát és a biztonságos továbbfejleszthetőségét, míg az egyrétegű technika komoly problémákat okozhat.

A többrétegű technológia különösen fontos, ha a rendszert egy fejlesztő csapat készíti. Ilyenkor jobban szeparálhatók a feladatok, illetve a rendezettség méginkább előtérbe kerül.

Az itt leírtak elsősorban a relációs adatbázisokra épülő vállalati rendszerekre vonatkoznak, de könnyedén átültethetők más környezetbe is.

Egyrétegű programozás

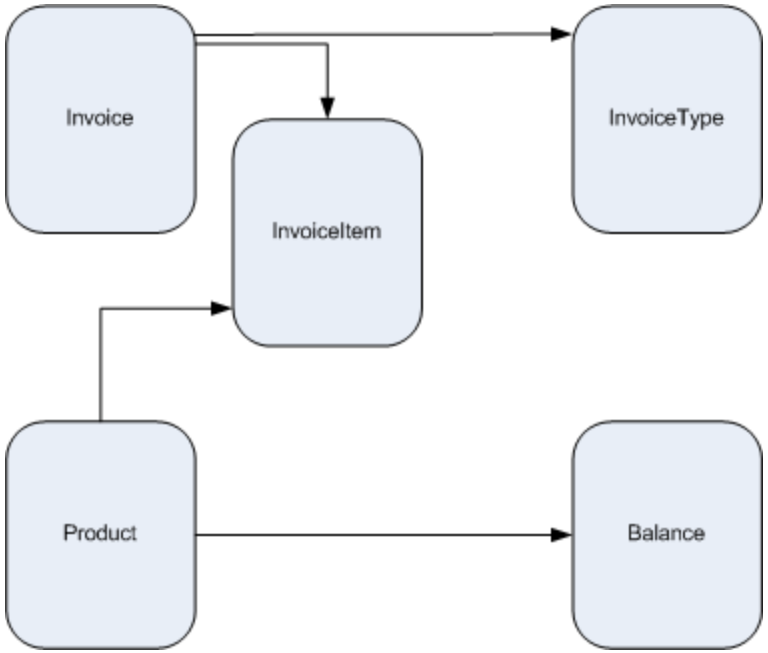
A BASIC, a Pascal, és más hasonló procedurális nyelvek idején még csak az egyrétegű technika volt ismert. A Pascal ugyan lehetővé tette a metódusok unit-okba rendezését, de a metódusok csak az induló alkalmazáson belül, annak rendszerfolyamatában (processz) futhattak, és akkor még nem beszéltünk arról, hogy sokáig a multitaszk is ismeretlen volt.



1. ábra: 1-rétegű *alkalmazások* felépítése

Egyrétegű alkalmazásnál a nincsenek szeparált rétegek. Az adatbázis-kezelést, az adatok értelmezését, feldolgozását a felhasználóval történő kapcsolattartást mind egy környezet végzi.

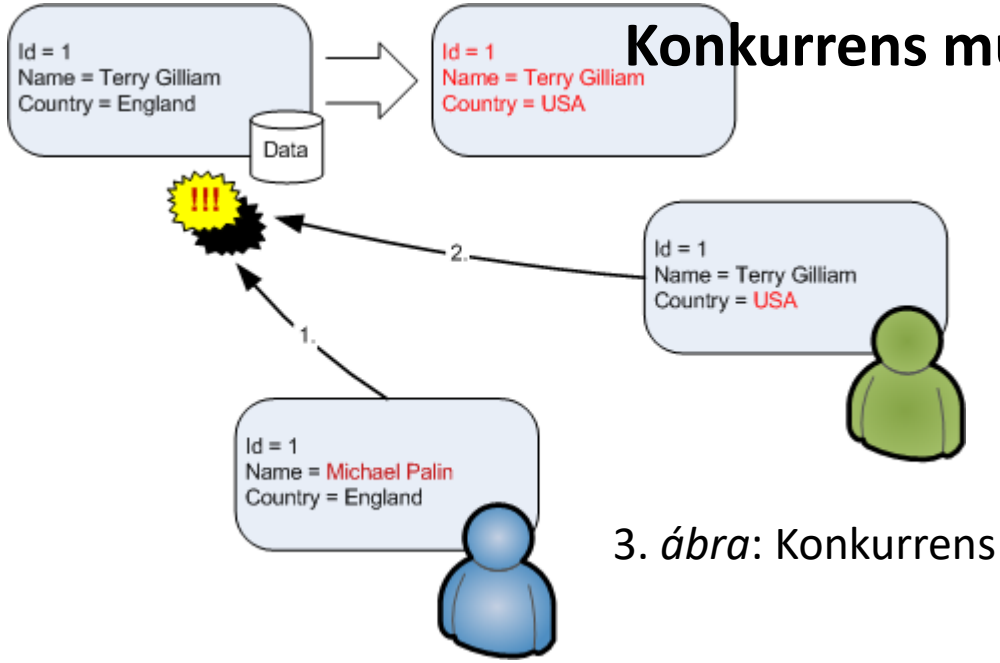
Vegyünk például egy tetszőleges adatbázist. Tegyük fel, hogy készítenünk kell egy rendszert, ami ezen az adatbázison alapul. Az adatbázis szerkezetéből adódóan több tábla kezelését kell megoldani, valamint szabályokat kell alkalmazni a feldolgozások során.



2. ábra: *Egy* adatbázis struktúra példa

Konkurrens műveletek

Mivel többfelhasználós rendszert készítünk, előfordulhat olyan helyzet, hogy két felhasználó konkurrens módon próbál módosítani ugyanazon a táblán. Ez inkonzisztens állapotot hozhat létre, mert amit az egyik felhasználó módosított el fog veszni, ha a másik felhasználó felülírja a saját adataival (az elavult adatokkal együtt) a bejegyzést.



3. *ábra*: Konkurrens módosítás

Milyen gyakran fordulhat ez elő?

Felmerülhet a kétely; *"Ilyen úgysem lesz soha, vagy ha igen, csak ritkán fog problémát okozni."*. Ez attól függ, milyen jellegű a feldolgozás, illetve hányan dolgoznak egyszerre a rendszerben. A komplex feldolgozások és a sok felhasználó mind növeli az ütközés esélyét.

Ha két felhasználó ugyanazt a bejegyzést nyitja meg módosításra – és itt vegyük figyelembe, hogy a felhasználói interakciók a rendszer szempontjából véletlenszerűen lassú műveletnek tekinthetők – az egyik módosítás biztosan el fog veszni.

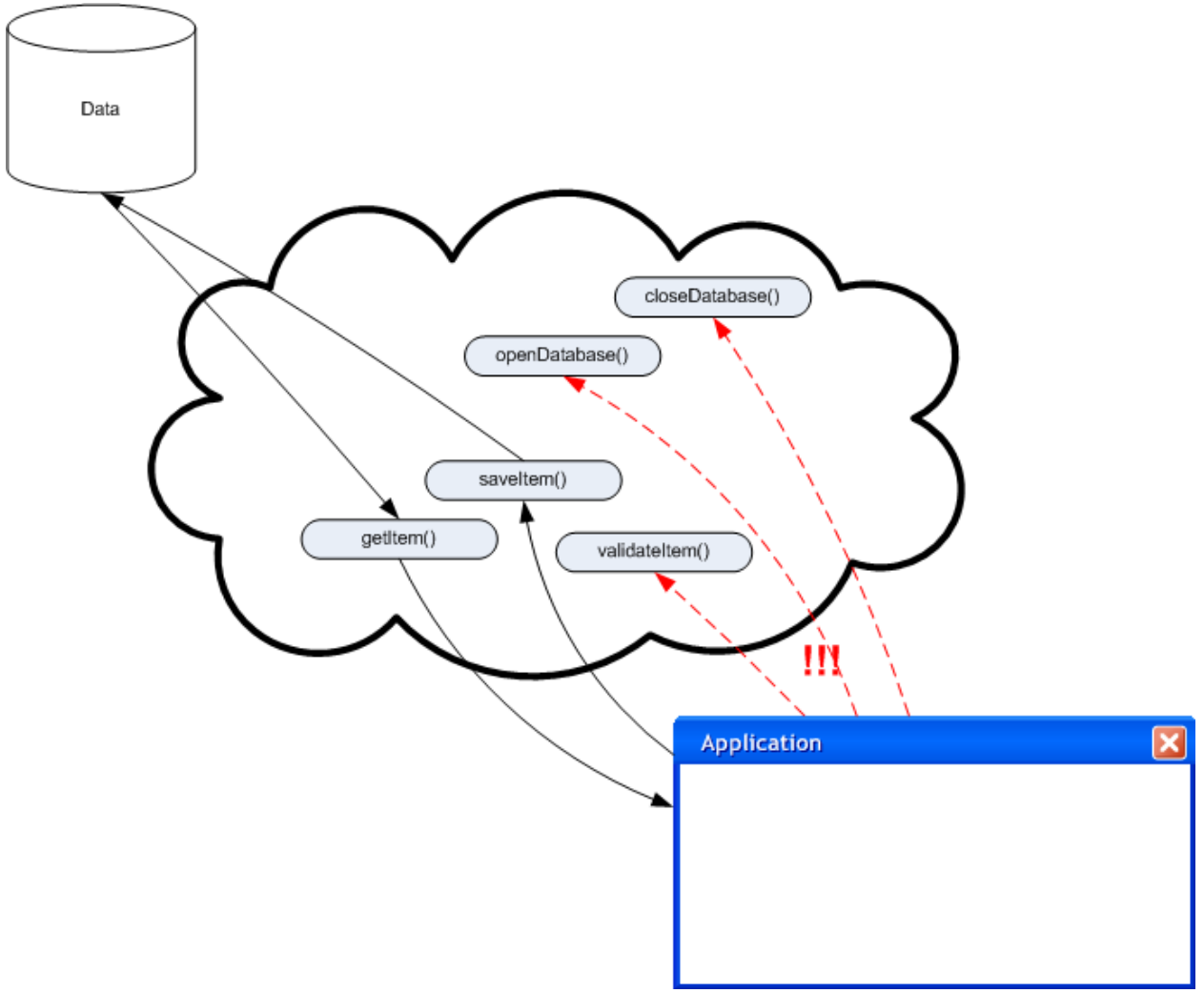
De az automatizált műveletekkel se sokkal jobb a helyzet. Ezeknél a műveleteknél nagy sebességgel tömeges feldolgozás történik, az érintett táblák tartalma folyamatosan változik. A feldolgozás futása alatt a bejegyzések módosítása hasonló helyzetet idéz elő, mint az előző példánál.

Ebből a két példából látható, hogy a konkurrens műveletvégzés potenciális veszélyt jelent egy ilyen rendszernél.

Mivel szem előtt tartjuk a kód-újrahasznosítás (code-reuse) szabályait, az egyes részfeladatokat külön metódusokba helyezzük. Ilyen metódusok például az alábbiak:

- openDatabase() megnyitja az adatbázis kapcsolatot
- closeDatabase() lezárja az adatbázis kapcsolatot
- getItem() visszaad egy elemet az adatbázisból
- validateItem() validálja az objektum adatait
- saveItem() elmenti a bejegyzést az adatbázisba

Ez a módszer azonban egy nagy hátrányt is rejt magában. Az egyrétegű alkalmazásoknál az egyes metódusok bárhonnan elérhetők, például a saveItem metódus egy kattintással elérhető a felhasználói felületről. Ez abból a szempontból aggályos, hogy a metódust meg tudjuk hívni anélkül, hogy a validateItem metódust meghívnánk.



4. *ábra* A metódusok megkerülhetősége

Skálázhatóság

Egy réteg esetén nem nagyon lehet skálázhatóságról beszélni. Az egyetlen réteg nem sok választási lehetőséget ad a felhasználó és az adatbázis közötti kapcsolat kialakításában. Ha az adatbázis műveletekért felelős komponensek leterhelődnek, az egész rendszer le fog lassulni. Ha a háttérben elindítunk egy feldolgozást, a felhasználói felület is belassulhat.

Egy súlyos rendszerhiba esetén a háttérben futó feldolgozás is megszakad, így nem csak a hibát okozó folyamat adatai vesznek el, hanem minden félbehagyott munka.

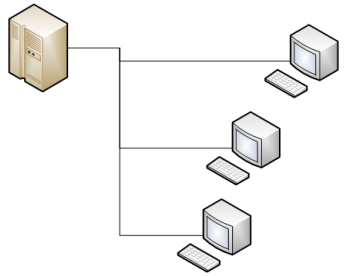
Környezeti hatások

Még ha azt feltételezzük, hogy a programunk “hibátlan” és soha nem fog lefagyni, akkor is számos hatás okozhatja az adatok sérülését. Az egyrétegű alkalmazás egyetlen gépen fut, és ez a gép rendszerint egy közönséges munkaállomás. A közönséges munkaállomások nincsenek áramszünet ellen védve, nincsen bennük drága hibajavító memória, nincs redundáns tápegységük, hibatűrő háttértáruk, és még lehetne sorolni.

Bármelyik egység meghibásodása az operációs rendszer lefagyásához, újraindulásához vezethet, ami a programunk szempontjából ugyanúgy káros.

Többrétegű programozás

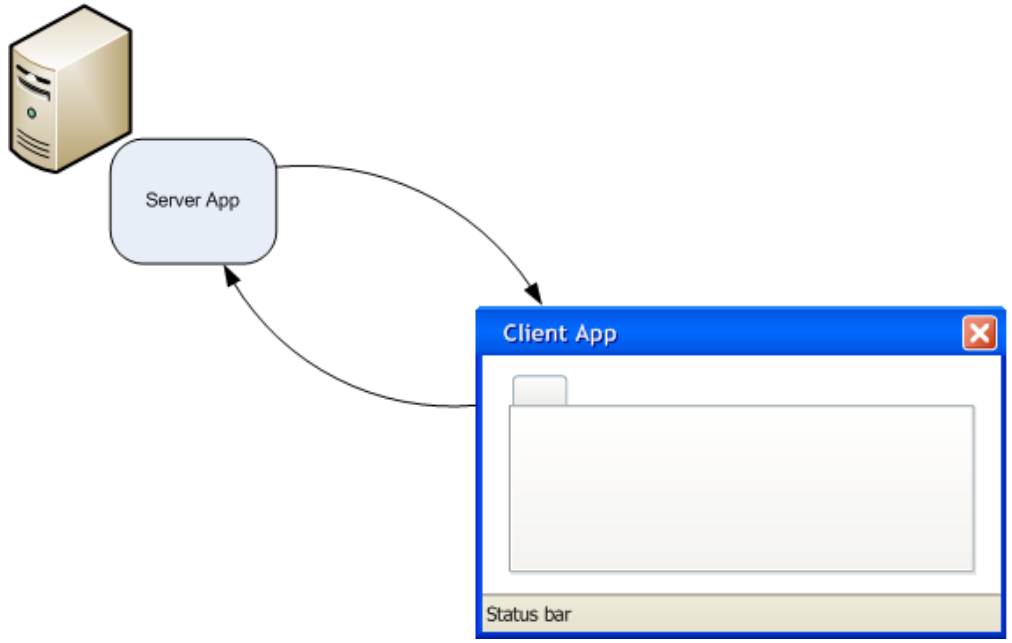
A hálózatos rendszereknél – amilyenek a többfelhasználós rendszerek – alapvető kialakítás a kliens-szerver architektúra.



5. ábra Kliens-szerver architektúra

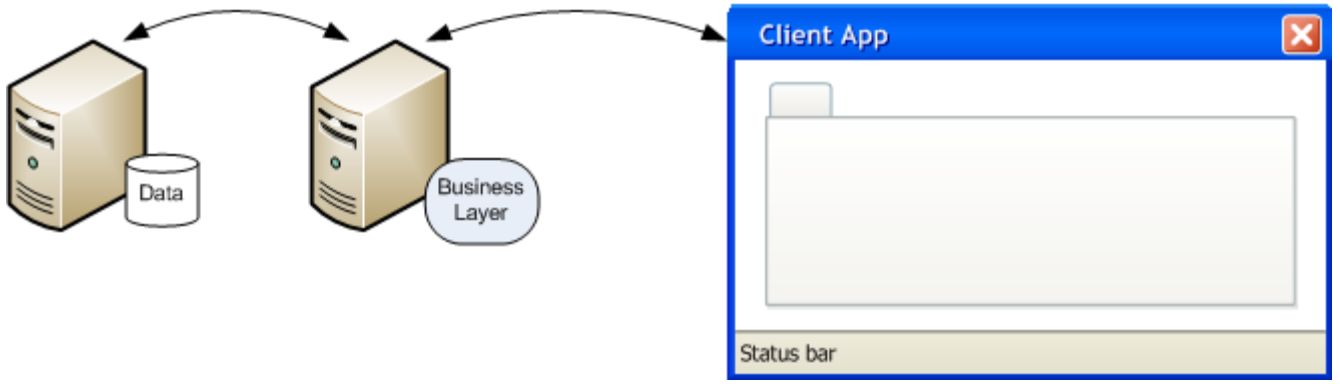
Hány réteg kell?

Azt, hogy hány réteget akarunk használni, mi döntjük el, de a tervezést érdemes jól megfontolni. A megfelelő kialakítás esetén – a kliens oldal felelős a megjelenítésért, a szerver oldal minden más feladatért. Bizonyos szempontból ide sorolhatók a WEB alapú technológiák is, mint az ASP, vagy a Tomcat, de ezeknél a rendszereknél általában csak két réteg lesz kialakítva; a kliens oldal és a szerver oldal.



6. ábra Kétrétegű alkalmazás

Elterjedtebb módszer viszont a 3-rétegű technológia, ahol külön rétegre kerül a megjelenítés, az üzleti logika és az adatbázis-kezelés.



7. ábra Háromrétegű alkalmazások felépítése

Ez a módszer sokkal rugalmasabb környezetet biztosít, különösen akkor, ha adattárolásra több alternatívát is szeretnénk alkalmazni.

Hátrányok

•A többrétegű programozásnak az egyik hátránya, hogy sokkal komplexebb a rendszer felépítése, így már a tervezéstől kezdve sokkal több időt igényel a fejlesztés.

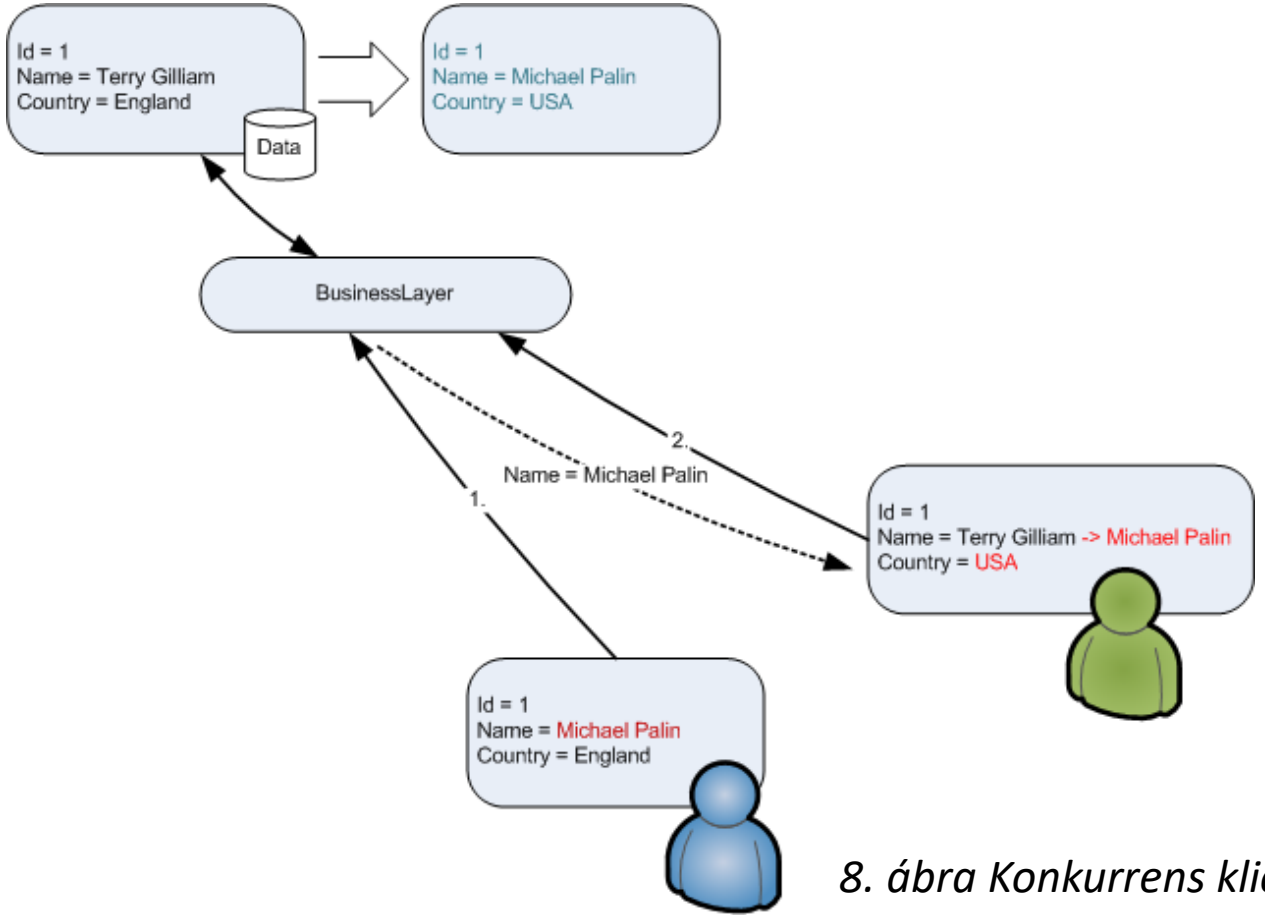
•Mivel a feladatok elsősorban a szerverre (szerverekre) vannak terhelve, ha a szerver meghibásodik, vagy lefagy, az összes kapcsolódó kliens munkája megszakad, emiatt sokkal körültekintőbben kell a rendszert konfigurálni.

Előnyök

- Konkurrens műveletek
- Megkerülhetőség
- Skálázhatóság
- Alternatív megjelenítés
- Adatbázis réteg cseréje
- Adatbiztonság
- Interaktív kommunikáció

Vegyük sorra az előnyöket az egyrétegű alkalmazással szemben.

A szerver oldali összefogott felügyelet lehetővé teszi, hogy a szerver menedzselhesse a műveletek sorrendiségét, vagy a konkurrens kliensek szinkronizálását.



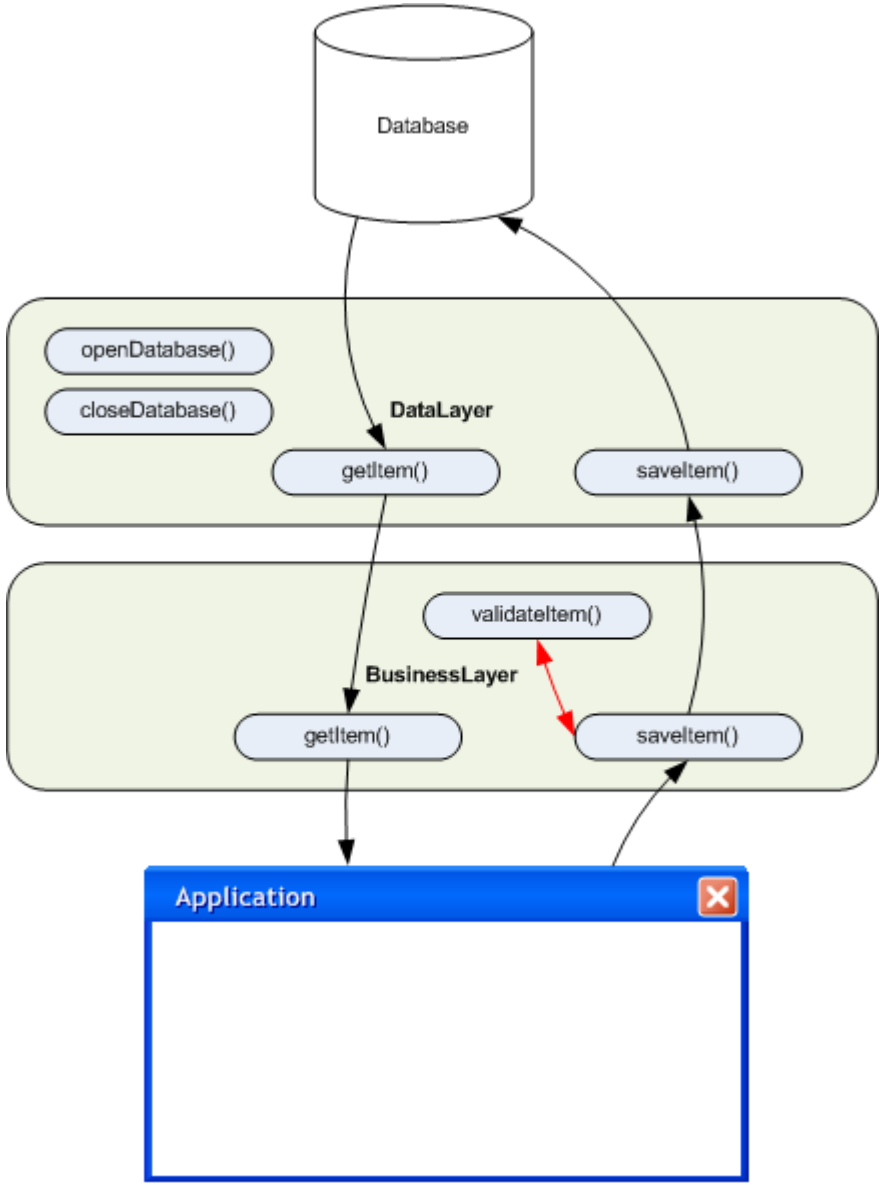
8. ábra Konkurrens kliensek értesítése a változásról

Ha két kliens ugyanazt a bejegyzést kívánja módosítani, a “vesztes” klienst – vagyis amelyik egy pillanattal később akarja a bejegyzést menteni – a szerver értesíteni tudja, hogy ütközés keletkezett, újra kell kezdeni a műveletet.

Megkerülhetőség

Mivel az egyes rétegek jól elkülönült folyamatokban futnak, nincs lehetőség arra, hogy az egyes rétegek átlépjék a közbenső rétegeket. A kliens réteg nem is ismeri az adatbázis-réteget, így csak az üzleti rétegen keresztül tud műveletet végezni.

Ez a megoldás biztosítani tudja a sorrendiséget, de természetesen a rétegen belül továbbra is nekünk kell gondoskodni a helyes kódolásról.



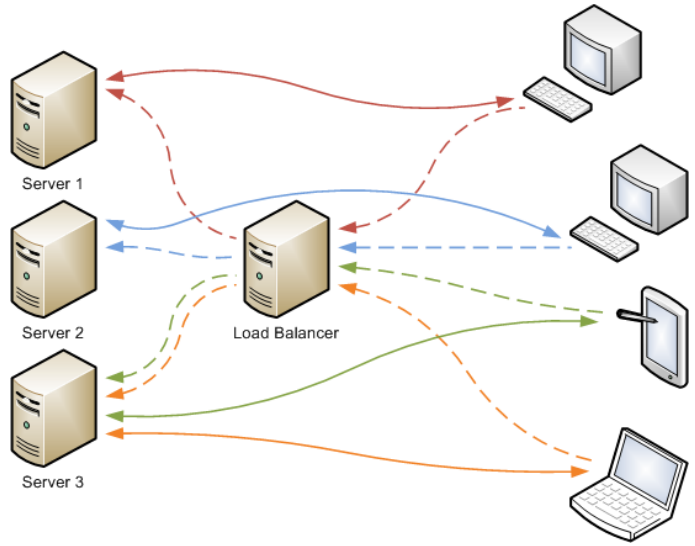
9. ábra Az ellenőrzés nem kerülhető meg

Skálázhatóság

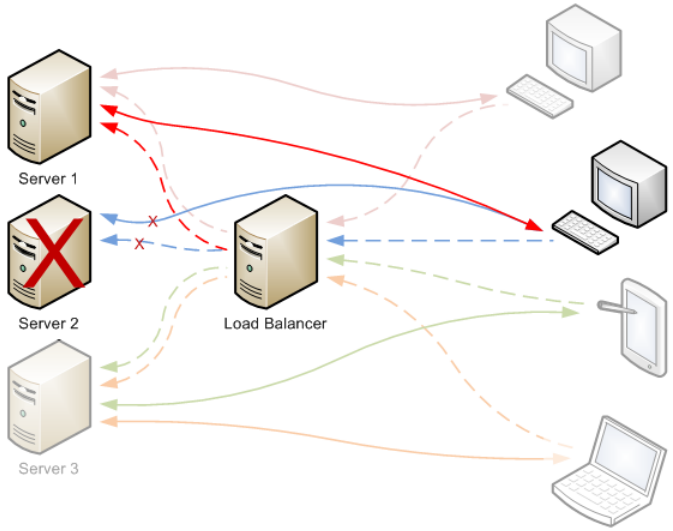
Ha a rendszert jól alakítottuk ki, meg tudjuk oldani, hogy a kliensek ne csak egy szerverhez tudjanak csatlakozni, hanem akár többhöz is. A szerverek közti választást egy terhelés elosztó (LoadBalancer) funkcióval lehet megoldani, ami automatikusan a legkevésbbé terhelt szervert adja vissza.

A skálázhatóság alapja, hogy több szerver is képes ellátni ugyanazt a feladatot. Ezek a szerverek működhetnek párhuzamosan, vagy hot-swap üzemmódban, amikor az egyik kiesése esetén egy másik veszi át annak szerepét.

A kialakítás előnye, hogy a kliens szempontjából lényegtelen, hogy melyik szerverhez csatlakozik. Ha egy szerver kiesik, akkor nem omlik össze a teljes rendszer, csak azok a kliensek szakadnak le a hálózatról, amelyek ehhez a szerverhez csatlakoztak. Ráadásul a kliens azonnal újra tud csatlakozni egy másik szerverhez, és szerencsés esetben a felhasználó ebből semmit sem vesz észre.



10. ábra Terheléselosztás több szerver közt



11. ábra Egy szerver kiesése csak részleges hibát okoz

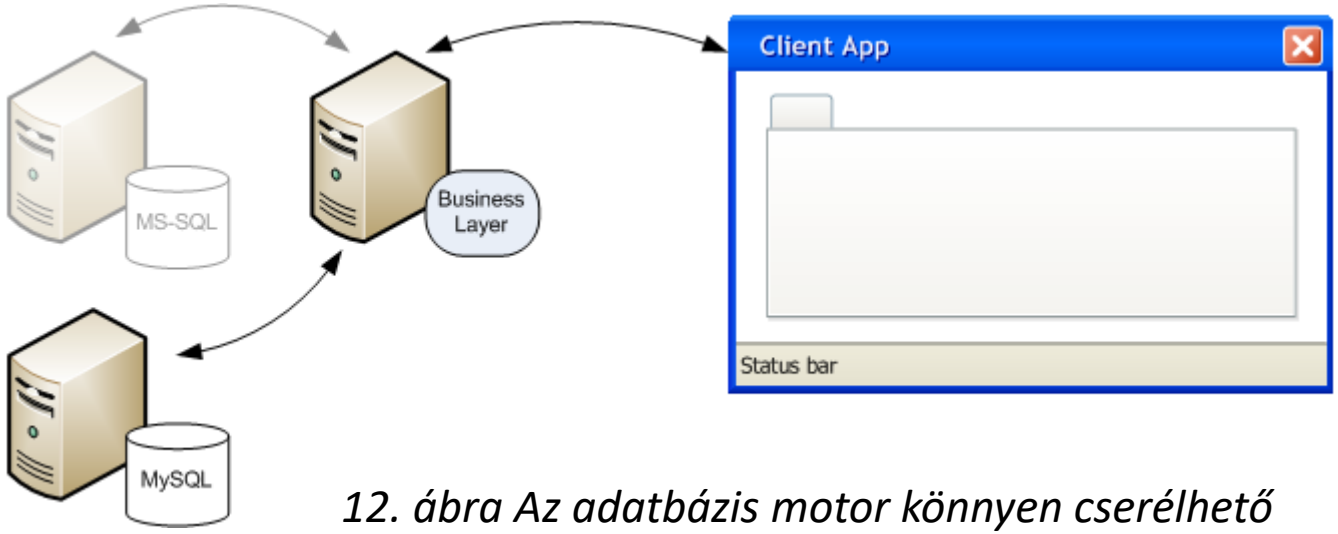
Alternatív megjelenítés

A felhasználói igények igen változatosak lehetnek. Előfordulhat, hogy egy nagyvállalaton belül több különböző csoport használja ugyanazt a rendszert más-más környezetből. Például az irodában dolgozó munkatársak Windows munkaállomásokat használnak, ezért ők elvárják, hogy a felhasználói felület részletgazdag és mutatós legyen. Más a helyzet a terepen dolgozó ügynökökkel, akiknél nem a megjelenés, hanem a hatékonyság a fontosabb, vagy ők böngészőből futtatható felületet szeretnének, amit akár az ügyfél bármelyik gépéről el tudnak indítani.

Jól látható, hogy ez két teljesen különböző felhasználási mód. A háromrétegű alkalmazás egyik előnye, hogy a felhasználói felület minden további nélkül lecserélhető, mivel az üzleti logika a szerveren található.

Adatbázis réteg cseréje

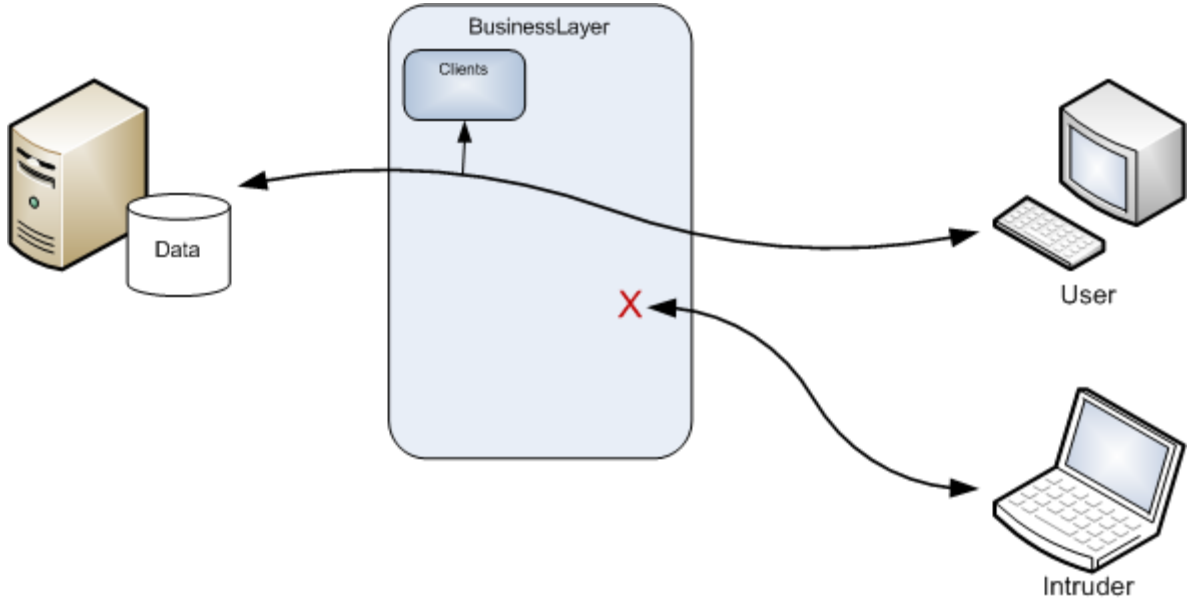
Előfordulhat, hogy a rendszer alatt cserélni kell az adatbázist.



12. ábra Az adatbázis motor könnyen cserélhető

A háromrétegű modell azért előnyös, mert az üzleti logika érintése nélkül az adatbázis réteg bármikor kicserélhető, így “egy mozdulattal” térhetünk át mondjuk Oracle-ről MS-SQL-re, vagy PostGreSQL-re. Mivel az adatbázis rétegen csak egyszerű lekérdezések és műveletek vannak, azok átírása egy másfajta technológiára relatív rövid idő alatt elvégezhető.

A szétválasztott rétegeknek van egy biztonsági előnye is. Mivel a bizalmas információk a szerver oldalon maradnak, az egyes munkaállomások csak egy jól kontrollált környezeten keresztül férhetnek hozzá. Ha jó a biztonsági stratégia, akkor minimálisra csökkenthető a visszaélések, támadások száma.



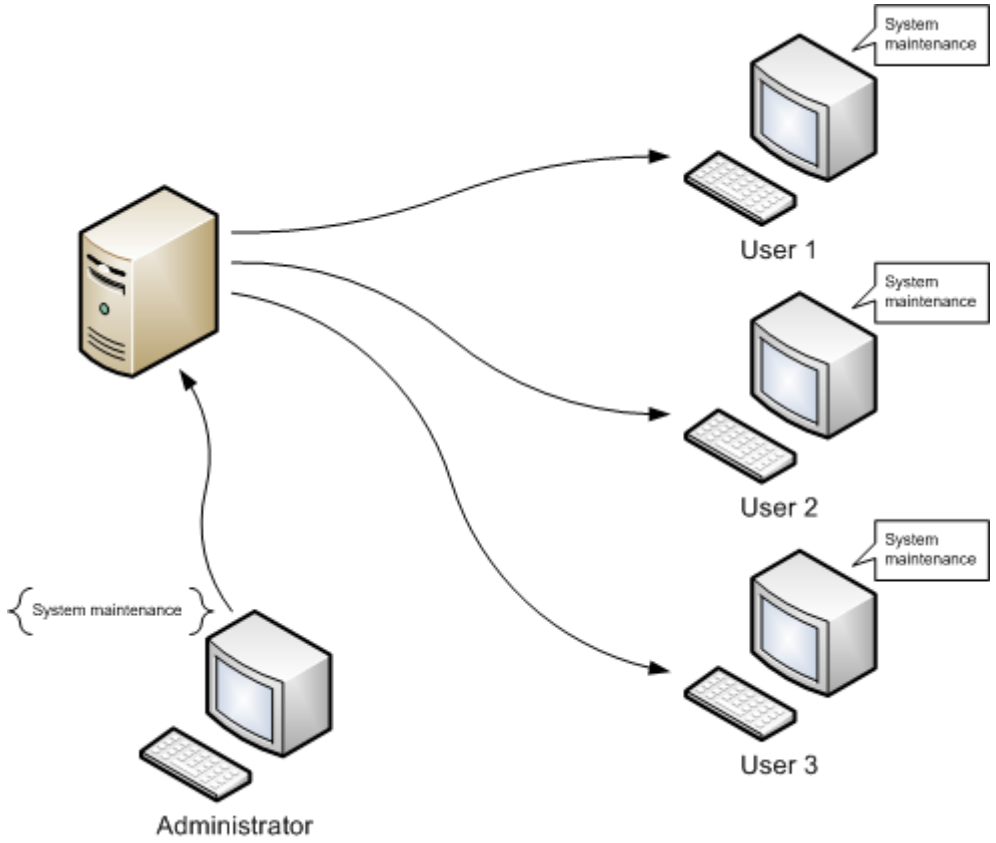
13. ábra A hozzáférés csak engedéllyel lehetséges

Interaktív kommunikáció

A vékonyréteg technológia miatt a munkaállomásnak elve úgy kell kommunikálnia a szerverrel, hogy a szerver üzeneteire bármikor tudjon reagálni. Ez a funkció felhasználható az interaktív kommunikációra is, például két felhasználó között.

De így üzenhet a rendszergazda is a klienseknek, hogy ha például le kell állítani a rendszert karbantartás miatt.

Az interaktivitás felhasználható arra is, hogy a szerver adatokat gyűjtsön a munkaállomásról, vagy az esetleges kliens oldali hibákról.



14. ábra A munkaállomások könnyen értesíthetők

Összefoglaló

Az eddig leírtak alapján azt hiszem, jól látható, hogy mennyi előnnyel jár a többrétegű alkalmazás-fejlesztés. Bár összetett feladat a tervezés és a kivitelezés, de sokkal korszerűbb és dinamikusabb rendszert tudunk készíteni ezzel a módszerrel.

A mai világban egy alkalmazás gyakorlatilag soha nem készül el teljesen, mivel folyamatosan keletkeznek új igények. A többrétegű technológia sokkal több lehetőséget ad és – mivel már az elején alapos volt a tervezés – könnyebb a bővítés is.