***Államvizsga tételsor “Integrált információs rendszerek” tárgyból***

Sunday, June 4, 2017

1:01 PM

Made by Forest, 2017.

Minden kérdés kidolgozva - de évfolyam függő hogy mely kérdések kellenek pontosan! Szóval dobj egy levelet a Tanárnak. (2015-16/1 évfolyamnak mind kell)

Tartalom:

1. Mind a 20 tételsorbeli kérdés kidolgozva
2. Vizsgakészüléshez tákolt jegyzet
3. Feladatok a pdf-ekből

**TÉTELSOR**

|  |
| --- |
| **1. Információs rendszerek: általános bemutatás, jellemzők, teljesítmény mértékek, architektúrák, teljesítmény növelésének módszerei** |
| **Információs rendszer:**  Általános, minden vállalaltnál felmerülő adattárolási és adatkezelési problémákat megoldó hardver-szoftver + humán elemekből álló együttes.  Személyek, folyamatok műszaki megoldások együttműkjödő összessége Tervezési minták, adatábrázolási szabályyok. **Célja,** hogy a vállalat számára a szkükséges külső és belső adatokat összegyűjtse, rendezett és biztonságosan tárolja, kiszolgálja, és döntésttámogasson.  Példa: Könyvelés, szállítmányozás, logisztikka, irodai alkalmazások. Rossz példa a lift, vagy oprendszer, telefonközpont stb.  **Integrált,** mert nem egyetlen monolitikus elem valósítja meg, hanem egy heterogén alkalmazás, amit össze kell "drótozni"  **Információs rendszerek tulajdonságai:**   * **Sok adat** * **Perzisztencia -** az adat védelme * **Sok felhasználó** - konkurensen is! * **Üzleti logika -** speciális business logic * **Sok emberi interakció** - űrlapszerkesztő felületek * **Biztonság -** hozzáférés, jogosultságok * **Együttműködés -** sok heterogén elem   **Információs rendszerek típusai**   1. **Tranzakció-feldolgozó** 2. **Vezetői információs** 3. **Döntéstámogatói**   Vagy:   1. Felsővezetői 2. Szakértői eszközök 3. Irodai tevékenységet kiegészítő   Vagy **Összekapcsolt emberek szerint**   1. **Business - to - employee B2E** : pl. ügyfélszolgálat, saját nyilvántartási rendszer, raktározás, számlázás 2. **Business - to - business B2B** : pl. vaáálalat alklamzásia között 3. **Business - to - customer B2C** : pl. vállalat és ügyfelei között, CRM rendszer   **Teljesítmény és jósági mértékek**   * **Átbocsátóképesség:** TPS * **Válaszidő** * Maximálsian kezelhető adatmennyiség * Maximum egyidejű felhasználók * Reakcióidő : kérés indítása és az első visszajelzés között * Késleleteés * **Little törvénye:** A rendszerben lévő tranzakciók (N) száma függ:   + A rendszerből kilépő tranzakciók számának (aktuális terhelés, P) és   + A tranzakciók rendszerben eltöltött idejének (aktuális válaszidő, t(P)) szorzatával   + **N = P \* t**   **Párhuzamosság:** elosztott rendszerekben elkerülhetetlen. Korlátokkal jár, de nagyobb lesz az átbocsátoóképesség, erőforrás kihasználás, skálázás. Hátránya, hogy bonyolultabb a rendszer, és csökkenhet a hatékonyság.   * **Költsége:** végrehajtott lépések száma, vagy felhasznált gépidő   + Pl:. Feldolgóz egységet tartalmazó rendszerben: k(n) = t \* Szumma kihasználtság. * **Hatékonyság**: legjobb egygépes vérgehajtás költsége / n számú gépen végrehajtott párhuzamos költés * Lényeg: a skálázás közben egyszercsak van egy csúcsérték, ami után új elemeket beállítva már csökken a hatékonyság. * Számítógép által létrehozott helyettesítő szöveg:   futási Idő (rnsl    1010    idó    Kormwriációs overtpad    O Haté konwág    Gprsułas    csom- ntck száma (    1224587 g 12    csanópontc* száma (db)   **Rendszerarchitektúra:** a rendszer alapvető felépítését bemutató koncepció. Részei a modulok, azok hogyxan kapcsolódnak egymással, megvalósulás, hardver és szoftver ami kell hózzá, leképezések, stb.  Vállalati struktúra jellemzői: Centralizáltság, Formalizáltság, Komplexitás    Egyéb fogalmak: Objektív / Szubjektív információ, Adat, vs. Információ vs. Tudás |
| **2. Központosított és elosztott rendszerek, vállalati rendszerek rétegszerkezete, hexagonális minta** |
| **Központosított rendszerek**  előnyei**:**   * **Biztonság, Adminisztráció, telepítés, üzemeltetés, karbantartás, fejleszőtgárda, támogatási lehetőgés, MEGBÍZHATÓSÁG, TELJESÍMÉNY**   hátrányai**:**   * **Rugalmatlanság, bottleneck, szűk keresztmetszet, kritikus hibapont a központ.**     **Elosztott rendszerek**  előnyei:   * Rugalmasság, Megbízhatóság, rendelkezésre állás, **teljesítmény,** helyi önállóság   hátrányai:   * Központosított előnyei, komplexitás, overhead.       **Modulok:**   * Független, jól definiált elemek, ember is átlátja * Sok jó architekturális tervezési minta van rá * Logikailag és fizikailag is szeparálunk, kombinálhatunk, stb. **rétegezhetőség**   **Rétegezhetőség:**   * Egymás fölé helyezett modulok, * **Modul és modul között interfész** * **Lentről szolgáltatásokat vesz igénybe** * **Felfelé szolgáltat** * **Előnyök:** rétegek könnyen cserélhetőek! * **Hátrány:** nem minden feladat szelhető rétegekre, + a réteges minta overhead.     **Rétegek szerinti típusok:**   1. **KÉTRÉTEGŰ**     1. Vastagkliens modell, fájl- vagy adatbázis szerver és kliens    2. Üzleti logika hova kerüljön?    3. **Számítógép által létrehozott helyettesítő szöveg:       Kliens        Prezentációs        réteg        Üzleti        logika        Hálózat i kapcso lat        Szerver        Adatelérési        réteg        Kliens        Prezentációs        réteg        Hálózati kapcsolat        Szerver        Üzl eti        logika        Adatelérési        réteg**        * **Több para is van vele.** 2. **HÁROMRÉTEGŰ**    1. **Adat - Üzleti logika - Prezentáció, Model-View-Control**    2. A kérdés az, hogy melyik réteg hova kerüljön a hálózati kapcsolaton?    3. **Számítógép által létrehozott helyettesítő szöveg:       elosztási nlinta neve        elosztott prezentáció        távoli prezentáció        elosztott feldolgozás        távoli adatelérés        elosztott adatelérés        alkalłnazási példa infornuiciós rendszerekben        A web alapú alkalmazások tipikus rétegszerkezete, a        prezentációs réteg egy része (a HTML oldalak generálása) a        szerveroldalon marad míg annak értelmezése és        megjelenítése a kliensoldali böngészÖ feladata.        Bankautomaták, egyes terminál szerverek, ahol a rajzoló        utasítások értelmezése a kliensoldal feladata.        Komponens alapú rendszerek, alkalmazásszerverek, Java        vagy más komponenseket telepitÖ web alkalmazások.        A klasszikus kétrétegú architektúra, ahol a szerver csak az        adatok tárolását végzi, tipikusan állományok, vagy relációs        adatbázis formąlában.        Az adatelérési réteg egy része a klasszikus vastagkliens        alkalmazásokban is megjelenhet a kliensoldalon (például        ODBC vagy nativ adatbázis driver), sót a mobil        alkalmazások kapcsolat nélküli működéséhez a teljes        adatelérésł réteget duplikálni kell a kliensen.        In egiegyzés        vékonykl i ens        arc h tektúra        vastagkl i ens        arch tektúra        okosklłens vagy        vastagkl i ens        arch tektúra**      1. **Számítógép által létrehozott helyettesítő szöveg:    Négyrétegú alkalmazások     • Prezentációs réteg kettéválik, a böngészőben lévő részt kliens rétegnek nevezik     Ötrétegú alkalmazások     • Extra beiktatott rétegek: terhelés elosztás, heterogenitás eltakarása, az elosztott     működés átlátszósága érdekében     Kliens     Prezentáció     (webb öng észö)     Hálózati kapcsolat     Szerver     Prezentáció     (webszerver)     zleti logika     alkalmazás szerver)     Adatelérés (DBMS     vagy fájl szerver)     zentációs     réteg     logika     lérés (DB     vagy fáj Iszerver     Kliensek     tációs     réteg     Hálózati kapcsolat     Szerve rek     átor.     réteg     Hálónti lat     ZI eti     logt ka     Hálózati kapcsolat     yitó     (data m apper)     Hálózati kap csolat     érés (D     gy fáj I szerver**   **Más megoldás még:**   1. **Peer-to-peer:**     1. Ugye torrent,    2. Erőforrás-pazaroló, és nehezen adminisztrálható 2. **Nem tiszta rétegszerkezet :** bizonyos rétegek átugorhatóak egyes esetekben, pl. windows és DirectX, vagy Java 3. **Számítógép által létrehozott helyettesítő szöveg:    Tiszta rétegszerkezet helyett hexagonális architektúra     Interfészek egy központi mag körül     Külsö szolgáltatásokat bevezető, és kifelé szolgáltató interfészek     Operatív     felület     1/0     Adminisztratív     alrendszer     felület     Központi     alkalmazás     logika     Operatív     felület     Adminisztratív Web service     felület     API     Központi alkalmazáslogika     Web service     API     Esemény-     napló     vo     alrendszer     Felhasználói     adatbázis     Esemény-     napló     elhasználói     jogosultsá     adatbázis** |
| **3. Kliens-szerver rendszerek evolúciója, vékony- és vastagkliens, okoskliens és mobilkliens alkalmazások. Felügyelt vastagkliens környezet** |
| **Kliens-szerver evolúció:**   1. **1960 mainframe** korszak: a számítógép nagyon drága, legyenek Master-slave jelleggel kliensek, de a mainframe aki számításokat végez 2. **1970 vége: miniszámítógépek :** szervezeti egységként több külön számítógép 3. **1980:** Személyi számítóép az ember szolgálatában 4. **1990:** Internet, és az összeköttetés 5. **2000+:** peer-to-pőeer, web2, SaaS, számítási felhők, stb.   **Klienstípusok**   * **Vastagkliens:**   + A klasszikus megoldás   + Kétrétegű, kliens-szerver architektúra   + Adatok egy szerveren, ezekre kapcsolódik a kliensnél létőz alkalmazápéldány   + Egy gépből álló rendszer, csak távoli adatkezelés   + **Előny,**  hogy specifikus, abllakok, szabánnyos alkalmazáskezelés, gyors reakció, helyi erőforrások kihasználása, kapcsolat nélküli működés   + **Hátány,** hogy nehézkes telepítés, konfigurálás és karbantartás, + a helyi gép… * **Vékonykliens:**   + 1990-es évektől, neten keresztül pl. böngészőben megjeleníítés   + Nincs háttértár, mozgó alkatrész, kioszkok,: klasszikus definíció egy nem teljes értékű számítógép   + Pl. gépen futó terminálszoftver vagy böngészőben futó kliens web alapon   + **Előny,** hogy nem kell telepíteni, platformfüggetlen, rugalmas, elenőrzött környerzet   + **Hátrány,** hogy válszodő van, kisebba szabadság, kapcsolatfüggőség, számítási teljesítmény korlátozva van. * **Böngészős vékonykliens:**   + Pl. egy Angular2 webite   + Akkora hülyeségek vannak írva a jegyzetbe, hogy le sem írom   **Vastagkliens -> Vékonykliens?**   * Vékony kliens rendszerek üzemeltetése olcsóbb. Vastag meg drágább, de jobb felhasználó élmény * Menedzselt vastag kliens környezetek első lépésként, távolról vezérelve * Költséges az új szoftverre állás, de megtérül * **Okoskliens:** vékony kliens és vastagkliens előnyeinek kombózása   + Lazán csatolt, szolgáltatásorientált architektúra   + Gazdag felhasználó könryezet, reaktív, kevés hely, könnyű telepítés + Offline is megy, biztonságos, frissít stb.   + Ezeknél csatolás kell, hogy szorosan, vagy lazán csatolt emelek legyenenk   + Megoldás: SOA alapelvek szerint.     - Ezek: önleíró interfészek, előre megbeszélt kommunikációval, állapotmentes szolgáltatásokon keresztül, esetleg címtáras vagy messaging szolg.   + "Weben keresztül érkező, vastaglkiensszerű viselkedéssel rendelkező alkalamzások, melyen meg igényelnek telepítést és saját frissítésüket auto hívják le".     - Jellemző jráuk hogy helyi erőforrásokat használknak, meg hálózati erőforrásokat, támogatják a mobil dolgokat, központilag menedzselhetők,     - Pl: .NET, Java appok, stb.   **Mobilkliens:**   * Ez egy speckó okoskliens kb. * Speckó modon a telefonra kifejlesztve * Gyakrolatilag vékonykliensek * Teljesítmény is variálódik   Számítógép által létrehozott helyettesítő szöveg: Klienstípusok összehasonlítása  vil aszi dó  kapcsolat nélküli  felha s ználói felület  kezelése  gazdag felhasználói  felület  telepítés, frissítés,  karbantartás  helyi hardver  eröforrások has ználata  együttmUk5dés más  helyi alkalmazás okkal  többszálúság  fejlesztés  számítás igényes  funkci ók a kliensben  különfél e képességü  eszközök támogatása  vastagkliens  megvalósítható  megszokott, szabványos  Igen, ewszerü a fejlesztés,  tesztelés és hibakeresés.  korlátlan  tgen  megszokott környezet, sok  eszköz, nagy tapasztalat,  bony Olult alkalmazások  tgen  vékon»zhens  I assú  egyszerú, de egyedi megoldások,  inkonziszt ens  Igen, de az összetett felhasználói felület  az alkalmazást egyetlen böngésző-  típushoz Vagy köti  korlátlan  Igen, de csak speciális komponenseken  keresztül (például COM).  Okoskliens mobil)  változó  megvalósítható  megszokott, szabványos  ewszerü a fejlesztés, tesztelés  és hibakeresés.  egyszerü  korl átlan  1 gen  mobil kliens  I assú  megvalósítható  A platform képességeihez igazodik.  korl átozott  1 gen  kevésbé kifinomult fejl esztÓ eszközök,  szimulátorok, Lnkompatibilis  kliens oldali platformok  Igen, de porlolni kell az alkalmazást a  különbözd környezet ekbe.  megszokott környezet, sok eszköz,  kevésbé kifinomult fejlesztö eszközök,  keretrendszerek, kommunikációra  inkompatibilis kliensoldali platformok  Igen, de a bonyolultabb alkalmazások  e 4'etI en böngészötipushoz kötödhetnek.  kih egyezve  lehet, de nem tipikus  Igen, de keretrendszer szükséges. |
| **4. Párhuzamos architektúrák: SMP, klaszterek, szuperszámítógépek, főbb tulajdonságok és felhasználási területek** |
| **Szerveroldali követelmények**   * Teljesítmény * Hibatűrés * Gazdaságos működés   Hogyan lehet a **teljesítményt növelni?**   * Teljesítménye a hardvernek nő * Jelterjedési és Hőtani krolátok vannak, meg gyártási korlátok * Jobb algoritmusok? ->elég kevés mozgástér * Párhuzmosítás: több mag, elosztott rendszer -> **na itt van még tartalék**     **Párhuzamos architektúra**   * **Sok réteg szoros együttműködése alakítja ki**   + Szorosan csatolt elemek   + Sok architektúra   + Klaszterek * **Szoftver: az architektúrák természetes szelekciója**   + Nem a jobbat választjuk ki, hanem amelyik elterljedtebb lesz.   + X86-os architektúra: ugyanazt az utasításkészletet futtatják, mint a 8086-osok   **Típusok**   1. **Nagy teljesítményű rendszerek**    1. **Számítógép által létrehozott helyettesítő szöveg:       Nagy teljesítményü számítási rendszerek        Szuperszámítógépek        SIMD        Elosztott feldolgozó rendszerek        Klaszterek Metaszámítási        rendszerek        — Toolkit-alapú        — Objektumorientált        Egyéb_        — Komponensalapú        — Peer to Peer        — Webalapú        Egyedi rendszerek        Közös        memórła        — Vektorvroc.        Elosztott        memórła        - MPP        —FKözös        memórła        - Közp. memó-        riavezérlö        - UMA (SMP)        — CC-NUMA        S-COMA        Elosztott        memórła        - NC-NUMA**    2. **Szuperszámítógép:** processzor- és memóriaszintű építőelemek, egysége rendszer    3. Elosztott rendszerek: elemei különálló gépek 2. **Több processzoros rendszerek**    1. Teljesítmény mérőszáma: fix v. lebegőpontos számítások helyett felhasználók által indított tranzakciók    2. Közös memóriás archtektúra, mert így nem kell új szoftver, szorosabb kapcsolás    3. **SMP, többmagos rendszerek:**        1. Közös címtéren osztoznaak a feldolgozóegységek - processzorok       2. Minden memória közös.       3. Nem hibatűrő, nem nagyon skálázható, csak baromi erős       4. Szálszintű és processzorszintű párhuzamosítás 3. **Elosztott rendszerek:**    1. Klasszikusan hálózaton keresztül együttműködő    2. Kívülről egyetlen rendszernek látszik    3. Információs rendszerekben általában komponens alapúak, DCOM, CORB, .NET, J2EE stb.    4. Webnszolgáltatásokon keresztül, estleg p2p    5. Külön is címezhetőek 4. **Klaszterek**    1. **Feladatvégrehajátst párhuzamosítanak**    2. **Köztük pedig kommunikáció üzenet alapú.**    3. Míg a sokprocesszorosnál a memóriában működnek együtt, itt egy kommunkációs csatorna van, és mindenki egész értékű szűmítógép    4. Klaszterek csomópontjai kifelé nem látszódnak , külön nem címezhetőek!    5. Klasztereknél mindig különálló, teljes értékű szűmítógépekből álló rendszer van, és úgy dolgoznak együtt.    6. Klaszterek I/O kapcsolása:       1. Rendszerek összekapcsolására tervezték       2. Lazább csatorlás, rendszerfüggetlen       3. Kimdenféle megoldások    7. Vezérlőszoftver:       1. Kell egy, aki airányít       2. Csomópontok, címtér, felhasználók, jogosultság, hibbakezelés, adminisztreáció stb.    8. Klaszter típusok       1. Cél szerint:          1. Teljesítmnyorientált -> olyan mint a szuperszámítógép, csak olcsóbb          2. Hibatűrés -> olcsó megoldás       2. Felépítés szerint          1. Közös háttértárral rendelkező          2. Közös háttértár nélkül (pl. webszerver)   Számítógép által létrehozott helyettesítő szöveg: Architektúrák összehasonlítása  Teljesítmény  Skálázhatóság  Hibatürés,  elérhetőség  Adminisztráció,  karbantartás  Licencek  Biztonság  Szoftverellátottság  Egységnyi számítási  teliesitménv költsége  Egyszerü SMP (és CC-  PC (viszonyítási alap)  Szuper-számítógépek  korlátozott  HW-kié ítés  kiváló  (redundáns  architektúra)  egységes,  egyszerü  speciális  (nagy biztonságú,  sokfelhasználós)  speciális  Klaszterek  nagy  (a kommunikáció  korlátoz)  folyamatos  kiváló  (a csomópontok egymást  helyettesítik)  mennyiségi problémák,  szoftverrel biztosított  egységesség  minden gépre  külön  speciális megoldások  szükségesek  speciális  (webszewer, DBMS)  egységnyi  ni ncs  egységny  egyszerü  hagyományos  1 (egységnyi)  NUMA  közepes  (közös memória és 1/0  korlátoz)  korlátozott  HW-kié ités  egységnyinél rosszabb  egységes,  egyszerű  hagyományos  (PC-s változtatás nélkül  futtatható |
| **5. Gazdaságos üzemeltetés: szerver konszolidáció, virtualizálás típusai és módszerei, felhő megoldások** |
| **Gazdaságos üzemeltetés:**   * **Kettősség:**    + kötlségek csökkenjenek, lehetőleg minél kevesebb elromolható eszköz legyen, homogén rendszerek   + Viszont az eszközök mégis sokszínűek, rengeteg feladt van, és ez csak egyre növekszik tovább. * **Eszközök:**   + **Virtualizálás:** elfedi a modulok belső tulajdonságait, az építőelemk sokszínűségét   + **Konszolidáció:** információs rendszerek központosítására, hogy htékonyabb legyen üzemeletetni az elemeket. Gyakrolatilag egy vasra húzunk be dolgokat   + **Felhőalapú szolgáltatások:** vállalatzi határookon kívül mutató, máshonnan bérelt szolgáltatások -> leveszi a vállunkról a terheket.     **Virtualizálás**   * Sokféle termék -> legyünk függetlenek a beszálítóktól, legyenek -tlátszók a rendszerek,   + Virtualizálás: szoftver- és hardver erőforrások fizikai jellemzőeinek eltakarására szolgáló technika   + Eltakarja a rendszerek komplexitását, **több logikai részre bontja a fizikailag egységes rendszereket,** egységes interfészeket alkot, könnyebben kezelhető az eszköz, stb. * **Típusok**   + **Erőforrás-virtualizálás:** pl. virtuális processzormag     1. Virtuális memória, virtuális háttértaárak, virtuális szalagkönyvtárak(?) hálózat   + **Platformvirtualizálás** teljes virtuális hardver könyzeteket kialaktíása     1. Minden hardver-erőforrástra kiterjedő viretualizáció, teljes körű virtuális környezetet adunk     2. Egy számítógépen többféle oprendszer is működhet,     3. Pl. docker, vagy java alkalmazások mobilon     4. **Típusai:**        1. **Virtualizálás Alkalmazásszinten:**            1. tisztán szoftver alapon           2. Gazda-oprendszer szolgáltatásainak becsomagolása           3. Azonos architektúrát szolgálok ki           4. Pl. Wine, .NET, Java virtuális gép           5. Nem túl jó megoldás, általában a teljesítmény nem jó, meg sok az overhead, pl. Java esetén        2. **Virtualizálás Operációs rendszer szinten:**            1. A virtualizálás és az oprendszer összeolvad           2. Pl. Microsoft Virtual Server,           3. OS megsokszorozza magát izolált könyezetek formájában           4. Vannak módsítások hogy sneki ne barmolja szét pl. a kernelt           5. Kisebb redundancia        3. **Virtualizálás Emulációval:**            1. Adott fizikai eszköz teljes működését lemásoljuk           2. Nagyon lassú, de teljesen paltformfüggetlen tud lenni           3. VMWare és társai, virtual image-ek           4. Cross--platform, fejlesztések, Sandboxot is ad        4. **Virtualizálás Paravirtualizációval:**           1. Vendég oprendszert felkészítik a virtuális környezetben való futásra           2. Nagyobb a teljesítmény mint a tiszta virtualizáció, de az oprendszer speciális cserébe           3. Így egy réteget megspórolunk. Pl. User-Mode Linux UML, vagy        5. **Hardverből támogatott virtualizálás:**           1. Több processzort elérünk a vendég rendszerből, teljesíítmény práhuzamos futás           2. Olyan processorok, akik saját szinten támogatják az erőforrások partícionálását -> jobban csinálja mint az oprendszerek           3. Nincs hagyományos értelemben vett gazda-oprendszer, teljesen egymás mellett futtatott példányok           4. Virtuális gépek kezeléséről egy vékony "Hypervisor" réteg gondoskodik   Ez lehet **monolitikus:** nagyméretű, kontextusváltása lassabb, pl Vmware ESX, dirvert közösen opszjta meg  **Mikrokernel** - még vékonyabb, nincs benne pl., eszközmeghajtó, drivereket is a példányok adják   * + - * 1. **Beágyazott laptáblák: nested page tables :** shadow page, ??         2. **Megszakítás vezérlés:**   + **Munkakörnyezet-virtualizálás:** kliens gépek, helyfüggetlen munkakörnyezetek     1. **Terminálszoftverek:** mobil felhasználók, alkalmazások amelyek speckó hardvert használnak     2. **Távoli asztal:** egyszemélyes terminál szerver     3. **Távoli help:** betárcsázok anyuhoz     4. **Izolált környyezetek:** Sandbox, pl. Vmware stb.     **Konszolidáció**   * **Egy szolgáltatás - egy számítógép?**   + Bár a cuccok heterogének, külön-külön gépre kerülnek   + Közös gépen sok dolog miatt para lehet:     - Szolgáltatások zavarják egymást, fájlszinten, vagy erőforrásokban     - Biztonsági beállítások nem psecifikusak, megengedőbbek     - Nagyobb a singel point of fail para     - Harddver egosztás, stb. * Megoldás: a konszolidáció   + **Probléma:** a vállalat a kelleténél jóval több hardvert vásárol, és tart fenn. -> mivel minden szolgáltatás saját gépen fut. Merev, rugalmatlan, költséges.   + **Megoldás: konszolidáció**     - **Centralizálás:** meglévő szoftverek összevonása egy kisebb helyszínre     - **Fizikai könryezet egyesíts:** több különálló szerver helyett egy vas     - **Alkalamzások áthelyezése:** közös szerver vagy platform, ami logikus hogy összegűyjtheti a dolgokat     - **Adatintegráció:** különböző adatokat összegűyjtjük, és azonos formátumban, heterogén adatforrásokat megszűntetve egy közös DB-be zárjuk * **Eszközök a konszolidációhoz: virtualizálás**   + Pl. klasszikus virtuális gépek kialaktíása , partíciók   + Az egyes partíciók osztoznak a hardvereken, de nem tudnak róla   + Sok szerver -> kevés szerver   + Minden gépen egy oprendszer -> sokféle oprendszer   + Szoftver és haardver szorosan csatolt -> hardver és szoftver kevésbé szorosan csatolt   + Szolgáltatások egy gépen zavarhatják egymást -> egy virtuális gép, egy szoftver példány   + Kihasználtság nő * **Szerverkonszolidáció**   + Előnyei:     - Jobb erőforráskihasználtság, kevesebb géppel elintézhető ugyanaz a munka     - Jobb a redszermenedzsmet     - Viretualizálással ötvözve nagyon rugalmas     - Teljesítmény: ügyes csoportosítások csökkentik a holtidőt     - De vannak biztonsági kérdések     - C:\1083D865\0FF81E2A-E70C-4542-A8AC-7AF5E82FFF4B_elemei\image009.png     - Ha már egyszer virtualizálunk, akkor több egyéb módon is fel lehet használni a virutalizációt: felszabaudlló erőforrások * **Tipikus virtualizációs szolgáltatások:**   + Rendszerüzemeltetés-tmogatás, pillanatképek, riaasztások   + Egységes menedzsmentkörnyezet   + Egyséhges háttértár és fájlrendszer   + Rendszerüzemeltetés-támpgatás   + Dinamikus erőforrás kezelés     **Számítógépfelhők**   * **Hálózaton keresztül elérhető, igény szerint felhaználható IT erőforrás halmaz.** * Első ilyen klasszikus felhő: Amazon 2006, ma már mindenkinek van * Pl.: google webes alkalmazásai, amazon száímtási felhő, Microsoft Azuire, Vmware cloud,. Oracle cloud, stb. online tárhelyek * Felhasználók arányosan fizetnek a felhasznált erőforrásokért, közben másokkal megosztják azt, de ők virtuálisan el vannak tőlük választva * Szolgáltatás azonnal beszerezhető, és kis költséggel, jól skálázható, és nem nekünk kell menedszelni -> kiszerveztük az IT-t! * Hátrányok: nincs a kezünkben akkora hatalom a szolgáltatás felett, hálózati hozzáférés szügséges, jogi hercehurcák, átállás költség, stb. biztonság… * Felhőtípisok:   + **Infrastruktúra szogláltataás IaaS -** virtuááis hardverkönyezet ccsak   + **Platformszolgáltatás PaaS** - futtatókönryezetet is biztosítunk   + **Adatkezelőszolgáltatás SaaS & Alkalmazásszolgáltatás SaaS -** itt már konrkét szoftverek is stb.   + Magánfelhő: nem igazi felhő… vállalati kontroltt megtarthatpd     **Háttérrendszerek virtualizációja**   * **Teljes storage virtualizáció** * Pl. NAS, network attached Storage, fájlszinten hozzáférek a dolgokhoz, de nem tudom min vannak tárolva * Több ponton keresztül hozzáférhető a háttértár-hálózat * SAN:   + Legelterjedtebb   + Távoli diszkek lokálisnak látszódnaak az operációs rendszer számára -> szektor és blokkhozzáférés szintig!   + Virutális háttértár átirányítható másik géphez   + Egy tár egyszerre több géphez is csatolakozhat   **Raid rendszerek**   * **Redundant Array of Independent / Inexpensive Discs** * Hibatűrés megoldása redundanciával * **Típusok**   + **RAID1:**  csak sima tükrözés   + **RAID5:** +1 diszk, amin paritásokat tárolok, és ezt a paritásrekordot lemezenként mindig máshol helyezem el "csíkok"ban * **Ezért jó,** mert a lemezeket hiba esetén menet közben is lehet cserélni. |
| **6. Hibatűrő rendszerek alapfogalmai (MTTR, MTBF), tipikus hibaforrások és kezelésük. Hibatűrés egyszerű és több elemből álló, összetett rendszerekben. Egymásra épülő elérhetőségi szintek. Elérhetőség költsége és megtérülése. Kockázat kezelés.** |
| **Hibatűrés**   * Racionálisan kell rákölteni, a vállalat számára ez végtelen költség * A teljes rendszer megbízhatósága = a leggyengébb láncszem megbízhatósága * Több folyamatban is részt vesz egy rendszer -> többszörösen kell figyelembe venni * Számítógép által létrehozott helyettesítő szöveg:   Az elérhetőség annak az időnek a részaránya a teljes vizsgált időszakból,    amelyben egy adott eszköz (rendszer) a tőle elvárt funkciókat biztosítani képes.    • A meghibásodások között eltelt átlagos idó (Mean Time Beetween Failures, MTBF)    • A hibák kijavításához szükséges átlagos idó (Mean Time To Repair, MTTR)    hibamentes időszak    elérhetőség —    teljes vizsgált időszak    MTBF    MTBF + MTIR * Elérhetőség: %-ban, a "kilencesek" mértéke. Ötkilences : hetente 6 másodperc downtime. * Minden elemnek együtt kell nézni -> ha pl. 5 elem van, akkor az elérhetőségüket össze kell szorozni. (amúgy meg Poi eloszlás, na mind1…) * Az a hibatűrő rendszer, amelyben van jó sok tartalék hiba esetére. * **Hibatűrés javítása:**   + Vagy a hibagyakoriságot csökkentjük (MTBF nő)   + Vagy javítási időt csökkentjük (MTTR csökken)     **Hibagyakoriság csökkentése**   * Minőségi akatrészek * Üzemeltetséi körülmények * Redundáns esszközök * Előre felkészülünk bizonyos hibákra: szünetmentes táp, tűzfalak, rollback, felhasználó képzés, stb.   **Javítási idő csökkentése**   * Gyorsabban vesszük észre a hibát, riasztási módok * Gyorsabban szerelünk, mert van készleten cucc, jó elemzések * Helyreállítási stratégiák, hideg-meleg boot, stb.   **Elérhetőség szintjei**  Számítógép által létrehozott helyettesítő szöveg: Távoli redundáns telephelyek  Szinkronizált múködésú, hibatüró klaszterek  Rendszerelem-szintú redundancia, tartalékberendezések  Megbízható és visszaállítható adatmentések  Jól bevált elvek szerinti tervezett rendszer-adminisztráció  Költségek   1. Képzések, jelszavak, tűzfalak, vírusvédelem, érett szoftverek, karbantartott hardverek, dokumentációk, tesztelés, jó rendszerkialakítás, kritikus szituk kerülése 2. Biztonsági mentések, 3. RAID, SAN 4. Hálózatszervezés, DOS meg ilyenek ellen érzékeny, kell hozzá hálózati eszköz +1 megbhibásodhatószint 5. Adatközpontok, környezeti feltételek 6. +1 Kliensoldali védelem, vékonykliens vagy menedzselt vastagkliens, mentések a kliensnél is 7. Alkalmaáz és szolgáltatás tervezés, hibákkal eleve számolunk és nem fájnak 8. Tartalékok bevetése, számítógépek klaszterek 9. Replikáció, tükrözés, tehermentesítés 10. Katasztrófaterv: emberi élet és a vállalat túlélésének védelme 11. Minden egy nagy copy-paste a világ másik ponjtára akár     **Elérhetőség modellje** :   * Cél modellezni, hogy gazdaságilag reális hibatűrési szintet állítsunk be. Ezért modellezüük a rendszerben található kockeázatokat, mértékeket, az eredmény meg a megtérülés kiszámításával egy terv. * Az MTBF, MTTR mérőszámok jók, de a hibák eloszlása és egyéb tulajdonságai is fontosak. * Modellek csak az előre ismert hibajelenségekkel tudnak foglalkozni * Az egyes rendszerelemek hibái között lehetnek összefüggések * **Mi ellen mivel lehet védekezni?**  |  |  | | --- | --- | | **Hardverhiba** | Tartalék alkatrész, öndiagnosztika, redundáns alkatrészek, biztonsági mentés, jó minőségű eszközök | | **Szoftverhiba** | Ne legyél sóher, vegyél eredeti szoftvert támogatással, saját szoftver fejlesztése esetén legyen minőségi | | **Emberi gondatlanság** | Öldd meg | | **Emberi nemtudás** | Tanítsd meg és utána öld meg | | **Hálózat** | Hálózati védelem, szabályok, nyítl hálózat kerülése, fizikai kialakítás stb. | | **Áramszünet, kár, baleset** | Tűzoltó berendezés, biztosnági oh istenem mennyire kamu ez a tantárgy | | **Rosszindulatú támadások** | Zárd be az ajtót magad mögött |     **Megtérülési modell:**  megéri nekem egyáltalán ez?   * **E = p \* MTTR h**   + E = Effect, a hiba hatása   + P a hiba egy év alatti előforudlásainak száma, db.   + MTTR a hiba miatt keletkező kiesés a rendszeridőben   + H a hiba hatóköre, 0 és 1 között, mekkora részét kapcsolja le a rendszerünknek. * **R = K \* szumma E**    + R a kockázat, az összes hibatípushoz tartozó E érték órában mondjuk, és így már mréhető pénzben   + Általában órában adják meg az összes kockázatot, a diasorban a példa pl.: évente 3x4 óra karbantartás, a hálózat felén = 3 \* 4 \* 50% = 6 óra kiesés |
| **7. Tartalékképzés jelentősége és korlátai. Hibatűrő klaszterezés: szimmetrikus és aszimmetrikus átkapcsolás. Hideg- és melegtartalékok. Párhuzamosan működő csomópontok. Hálózati identitás átvétel és közös diszkek kezelési módszerei. Tudathasadás. Klasztervezérlő szoftverek szerepe.** |
| **Hibatűrés klaszterezéssel**   * Klaszter ugye több összekapcsolt, **önálló,** saját oprendszerrel rendelkező rendszer. * Legegyszerűbb hibatűrés, hogy ha kiesik egy csomópont, egy standby elem beugrik a helyére * Elég olcsó megoldás, és a klaszterezéssel már kaialkítottuk az infrostruktúrát hozzá, pl. hozzáférés-védelem, elosztott adattárolás, hibatűrő adatak   **Hibatűrő klaszterezések osztályozásai**   |  |  |  | | --- | --- | --- | | **Szempont** | **Típus** | **Leírás** | | Átkapcsolás iránya | Aszimmetrikus | Csak egy irányban lehet átkapcsolni, nincs visszaállás, vagy csak spec módon | | Átkapcsolás iránya | Szimmetrikus | Lehet oda-vissza. A modern klaszterek ilyenek. | | Átkapcsolás módja | Kézi | Lassú, szaki kell hozzá, nem kell hozzá speckó szoftver cserébe | | Átkapcsolás módja | Automatikus | Be kell konfigolni, de nem kell felkelteni a szakit éjfélkor. | | Működés módja | Hidegtartalék | Van a raktáran egy node… | | Működés módja | Melegtartalék közös háttértárral | Tartalék gép be van kapcsolva, figyeli a párját, ha az leáll, átkapcsol gyorsan | | Működés módja | Szinkronizált melegtartalék | Tartalék gép be van kapcsolva, és szinkron ifnókat kap az éles géptől, így még gyorsabban tudja átvenni a feladatát hiba esetén | | Működés módja | Eltérő szolgáltatásokat futtató tartalékok | Minden csomópont folyamatosan működik, de a fennmaradó erőforrásiakkal tartalékként is létenek ha kiesik valaki | | Működés módja | Párhuzamosan működő csomópontok | Minden csomópont megy, terheléseloszlást csinálnak egymás között, így konkrét tartalék nincsen, ha kiesik valaki, akkor a többiek elosztják a plusz terhelést |     **Átkapcsolás folyamata**   * Észreveszi a user hogy átkapcsoltunk? Átlátszóság * Mennyire sikerül gyorsan? * Mennyire kell felkelteni éjjel a szakikat hozzá?   Átkapcsolás során a szolgáltatáskörnyezet mozog. Ez ami a rendszer belülről kb. Tudja csinálni a Szaki, vagy a szoftver is.  Ilyeneket kell figyelni, hogy pl. a tartalék gépnek is ugyanazokhoz a hardverekhez legyen hozzáférése, amihez kell, vagy pl. IP és MAC címe változik-e?  Futnaak a szoftverek a gépekről, nekik is tisztában kell lenni azzal, hogy a gép mögülük kiesett? Vagy ezt harverrel is meg lehet oldani? CLOUD HM HM???   * Hálózati mizéria: lényeg, hogy az identitást átadjuk, vagy sem? Az fontos kérdés. * **Háttértárak, közösek, vagy tükrözve vannak, vagy valahogy előállítható?** * **Adatok hol vannak, hova mentsek, honnan olvassam ki?** * **Alkalmazások fel vannak készítve erre? Licencproblémák?**     **Konfigurációk**   * **Hidegtartalék** aktív-passzív mód: egyszerre csak az egyik gép megy. Mögöttük mondjuk ugyanaz a winchester fut   + Előnye, hogy nem kell külön klasztrezeő szoftver -> talán nem is klaszter igazán, olcsó védelem   + Hátrány, hogy kézzel kell átkapcsolni   + Dupla szerver * **Melegtartalék** mindkét csomópont m3gy, de csak egyik dolgozik   + Előnye, hogy nagy elérhetőséget biztosít, nemigazán van downtime   + Hátránya, hogy nagy az overhead, kb. felesleges kettő   + **közös háttértár:** hálókártyák közötti átkapcsolásra kell figyelni   + Porblémákat belül kell kezelni   + Valakinek figyelnie kell a csomópont életjeleit is   + Átvegyük az identitást? IP, MAC cím? Szívűs. Hpgy a hálókártyát is tudni kell konfigolni   + **Háttértár nélkül**: itt is egyszerre működnek, de csak az egyik nyújt szolgáltatást, a másik nem, és folyamatosan szinkronziáálják a merevlemezeket   + Hát ez ugye jó nagy overhead, de legalább véd a merevlemez hibája ellen * **Több aktív csomópontú konfiguráció**   + Amikor nem kettő van -> egyre jobbhibatűrés, egyre nagyobb pénzkidobás   + Tartalék csomópontokat mondjuk használjuk addig másra?     - Tesztelés, fejlesztés, másodlagos adatbázis, stb.     - Az a legjobb ha nem koptatjuk a célhardvert * **Aktív-aktív konfiguráció**: mindkét csomópont egyszerre működik, és egymás tartalékaik is egyszerre   + Viszont nem ugyanazt a szolgáltatást nyújtják   + Ezért kicsit jobb a kihasználtság   + Kevesebb az elérhetőség cserébe     **Klaszterek kettőnél több csomópontta:**   * **N-to-1 aszimmetrikus:** egy közös tartaléka van mindenkinek. Szívás a kábelezés, meg korlátos hányhoz tudod fizikailag odadugni, meg elég lájtos * **N-plus-1 szimmetrikus:** tartalék gép nem állandó, bárki lehet az. Menet közben derül ki, kit kinek a helyére teszünk be. * **Szolgáltatáskörnyezet-alapú tárkezelés:** a szerverek és a DB-k legyenek elválasztva, vagyis a lemezek legyenek külön menedzselve, RAID-ben, stb.   + **Gép-HDD nincs összekötve szorosan**, hanem hdd hálózatod van * **Párhuzamosan üzemelő csomópontok:** elég nehéz rá szoftvert írni, dobozos világ inkább vitualizációs dolgokra bukik   + Pl. webszerverfarmok mondjuk ilyenek, adatbázisok, SWARM technológiák * **Webszerverfarm:** az ugye ilyen mint az előző   + Egy webszerver nem hibatűrő, de átveszi szerepét más. A diszpécser aki a klaszterezést felügyeli   Amúgy a SAN-diszkek nagyon sokat segítenek mindegyik megoldásnál    **Tudathasadás:** párhuzamosan egymással versengő aktív csomópontok a diszk írásakor maradandó hibákat okoznak.   * Védekezhetsz ellene, ha van lock mechanizmus, vagy közös életjel detektálás * Kapcsoljuk le azt, aki meghibásodott, ha feléled ne barmoljon össze mindent     **Tervezési minták:**   * **Védekezés a ping-pong failover ellen:**   + Egy szolgáltatás nem indítható egy csomópontból sem, mindenki azt látja hogy a másikon nem fut, így folyamatosan próbálják átvenni egymás helyét sikertelenül.   + Megoldás, hogy ne induljon újra a hibáás csomópont magától * **Runaway failover**   + Még nem sikerült helyreállítani a gépet, de már adnám neki oda a kérést   + Ellenőrizzük ezt jobban!   + Legyenek szintek: kis hiba, nagy hiba, kritikus hiba, ennél már szaki kellejen aki visszakapcsol     **Mit csinálnak a klasztervezérlő szoftverek?**   * Monitorozzák a folyamatokat * Nézik ki él, ki nem * Hibahelyzetekben döntéseket hoznak * Terheléselosztás * Beavatkoznak autamitkusan:   + Ezt lehet gazdagon paraméterezni, újraindítani, stb. * Microsoft Cluster Server, Sun Cluster, |
| **8. Vállalati alkalmazásintegráció története, okai, előnyei és korlátai.** |
| **Vállalati rendszerek kialakulása:**   * Vállalati rendszer sokféle elemből épül fel: hardver, szoftver **és ember** * Mászaki sokféleség: kritikus rendszerek, spec célszofftverek, saját fejlesztésű szoftverek, DBMS * **Integráció okai:**   + **Cél:** kapcsolatrendszert alakítsunk ki a sját részterületet kiszolgáló alkalmazások között.   + Új alkalamzás gyorsan beilleszthető legyen, össszeolvasádok miatt beépüljünk az új rendszerekbe, lehessen továbbfejleszteni a rendszert, legyenek lazán csatoltak az eszközök, beszállítók felé szolgáltatások, interfészek, stb. * **Integráció előnyei**   + Más alrendszerekben található adatok és az ott implementált funkciók elérhetővé tétele   + Vállalati hatékonyság növekedjen: jobban dokumentált, gyorsabb, pontosabb és olcsóbb folyamatosa.   + Rugalmasabb az egész, átláthatóbb, mérhető, konfugárlható, új szolgáltatások előfeltételei is * **Integráció nehézségei:**   + Sok idő, és pénz. Sok meg sem valósul. Kapcsolatok fenntartása egy külön speciális réteg.   + **Integrációs spagetti:** túl sok és túl specifikus kapcsolat   + **Kockázatok:**     - Kis költségek -> korlátoznak, előre nem látható parák     - Állandóan változó rendszer     - Sokféle integráció szabvány létezik, melyik jó nekem?     - Szaktudás kell hozzá     - Informáióhiányos környezet, felfedezés     - Felelősségi körök, érdekellenségek a rendszeren belül     **Integráció folyamata:**   1. **Elemzés** - mik a követelmények? Üzleti folyaamatok? Rendszer felmérése. Mi a cél? Ez a legmunakigényesebb rész. Tudás elossztva van jelen Adatkatalógus, interfészkatalóguis, folyamatkatalógus stb. 2. **Koncepció -** Sikerkritériumok állítása költségterv, piaci elemzés, eszközök kiválasztása 3. **Tervezés - adatszótár,** üzenet és csereszabványok, architektúra megalkotása 4. **Megvalósítás -** Közvetlen kialakítás, kódolás, telepítés, tesztelés 5. **Bevezetés -** mindnki áttérjen az új rendszerre, próbaüzem     C:\1083D865\0FF81E2A-E70C-4542-A8AC-7AF5E82FFF4B_elemei\image012.png |
| **9. Integrációs szintek rétegek szerint. Integrációs szintek funkciók szerint (szállítás, irányítás konverzió, üzleti folyamatok).** |
| **Integrációs kapcsolatok:**   * **Multiplicitás szerint:**   + **Pont-pont:** egy dedikált csatona   + **Egy-több:**   + **Több-több** : pl. adatszórás * **Időzítrés:**    + **Szinkron:** gyors, egyszerű, pontos kép   + **Aszinkron:** nagyobb szabadság, hibatűrés, k.ésleltetések * **Indirekció:**   + **Közvetlen:** felek közveltenül ismerik és címzik egymást, szoros, de rugalmatlan, mneg gyors   + **Indirekt:** kommunikáció egy közvetítő segítségével, komplexitás * **Állapot-kezelés:**   + **Kapcsolatalapú:**   + **Állapotmentes:** üzenetek, skálázhatóság * **Nyugtázás:**   + **Kérdés-válasz:** handshaking, válasz iután kommunikálok csak   + **Nyugtázás nélküli:** nem várok választ, fire and forget * **Biztonság:** igény szerint     **Integrációs szintek**  C:\1083D865\0FF81E2A-E70C-4542-A8AC-7AF5E82FFF4B_elemei\image013.png   1. **Alkalmazásrétegben:** kommunikálnak az alkalmazások 2. **Adatrétegben:** közös adatok, importálással vagy közös dbms-sel, vagy XML, JSON stb. szinten    1. Előny, hogy keveset kell kódolni, olcsó, jól megvalósítható    2. Hátrány, hogy nem kívánatos, adattárolási formaa rögzítetté válik, duplikálódnak az üzleti logikai elemek 3. **Api szintű integráció**    1. Programozói interfészeket biztosítunk    2. Jól dokumentált, világos funkcionalitás    3. Alkalmazás kódjába bele kell túrni! Elk lell készíteni az api-t    4. Szaktudás kell hozzá,    5. API publikált funkciói többfélék lehetnek: üzleti folyamatok indítása, adathozzáférés, objektumok kiadása, stb.    6. Előny, hogy viselkedés szintű újrafelhasználás, meg obejktumorientált    7. Hátrány, hogy kódolni kell, sokat dolgozni, nem midnig sikerül objektum orientáltra 4. **Metódus szintű integráció**     1. Hasonló mint az API, az üzleti logikai rétegben működik    2. **Funkciók közössé tétele szabványos módon**        1. Mert az API lehet egyedi, pont-pont kapcsolat, ez még mélyebb    3. Egyszerű, szép objektumorientált kapcsolatrandszer    4. Beépéül egyik alkalmazás a másikba    5. Kétféle megvalósításaa van:       1. **Üzleti logikai központi tárolásával:** alkalmazásból kiszevezzük, egy közös futtatóhelyre tesszük. Az appok meg őrá mutatnak.       2. **Funkciók megosztásával az elosztott alklamazásokban** : minden alkalmazás a saját funkicóit maga publikálja, és egy infrastruktúra helyezi őket közös címtérbe    6. Szükség van hozzá objektum könyvtárakra, pl. dll    7. Kelle hozzá komponens keretrendszer, pl CORBA, EJB, J2EE, DCOM, .NET    8. **Előnye,** hogy vilgáos átllátható, magasan objektumorientált, kódújrafelhasználás    9. Hátrány, hogy drága, kell a keretrendszer, élő kódot kell módosítani, alkalmazásszerver is kell 5. **GUI szintű integráció**    1. Emberi felhasználásra készült GUI-kra kattint a szoftver…    2. Működik amúgy, sok helyen nincs is más lehetőség, pl. biztonság vagy legacy    3. Web világában olcsóbb ezt megvalósítani, mint egy rendes rendszert    4. A felhasználót egy programmal helyettesítjük, ami feldolgozza a képernyőt, és a kinyert dolgokat már ő kezeli    5. Ez ugye sok hátránnyal jár. De a teljesítménye javítható     **Rétegszerkezet:**   * **Üzenetszállítás:** kommunikációs protokoll szintű adattovábbítás * **Irányítás:** Integrációs központ kell hozzá. Szabályrendszer a címzések feloldására, * **Konverziós réteg:** integrációs központ kell ehhez is. Adatformátumok és üzenetek átalakítása az alkalmazások között.   **Integrációs központ :**   * Új köztponti elemet vezetünk be, akihez minden alkalmazás kapcsolódik. * **Feladatai:**   + Közös szabályrendszeren műkdöő adatátvitel biztosítása   + Alkalmazások egymáshoz való közvetlen ohzzáférésének elfedése * Lehetséges tervezési minták:   + Hub-and-spokes: passz   + Mediáció: brókerszerepet játszunk az alklamazások között, én vagyok a postáks   + Föderáció: szolgáltatásgazda az ingetrációs központ, lkalmazások egymást nem is látják   + Mediáció és föderáció kombinálva, kívülről ez, belülről amaz * **Előnye,** hogy kevesebb a kapcsolat, lehet aszinkron működni, rugalmas, meg csomó extraszolgáltatást be lehet vezetni * **Hátránya,** hogy lassítja a kommunikációt, extra rőforrás, plusz egy rendszer, meg bottleneck lesz     **Üzletifolyamat-kezelés**   * Következő témakörben leírva |
| **10. Alkalmazások tömeges integrációjának módszerei (empirikus, pragmatikus), spagetti, EAI, monolitikus vállalatirányítási rendszerek** |
| **Pragmatikus integráció:**   * **Egy formális szabálygyűjtemény az integráció helyes megvalósításához** * Üzleti logikai alapfunkciók és ezek szétválasztása, újrafelhasználás. * **Szabványos formai modellek,** metamodellek, transzformációk. * Pltaform független absztratk metamodell -> automatikus transzformmációk sorozatával létrejön a konkrét alkalmazás kód. * Folyamat -> protokollszekvencia -> osztálydiagram -> implementációs váz, külön szakterületi nyelv. * Nem annyira elterjed, magától születő kód, transzformációs lánc, komoly szaktudást igényel. * **Model Driven Architecture:**   + UML a folyamatok leírására, meg különféle specifikus modellek, XML, MoF, CWM stb. * Nagyon elegáns, de nehéz karbantartani. Csak akkor van értelme, ha az egész lánc megvalósíult   **Emprikus integráció**   * A pragrmatikus elegáns, de nagyon bonyolult. * **Az integráció is csak egy újabb rendszerfejlesztés, amihez megvan a szaktudás** * Jól megtervezett közös működés, tapasztalati együttműködés, ügyes mérnöki munka -> de kell hozzá hogy megfelelő minták mentén valósuljanak meg a folyamatok. * **Tervezési minták**   + **Spateggi probléma:**      - Számos interfész, és túl sok struktúrálatlan kapcsolat. Minden új eszköz, alkalmazás n-1 új kapcsolatot hoz be, és n-t növeli egyel…     - Problem: nehéz nagyon hozzáadni új elemet     - Megoldások       * Csak egy maradhat! Monolitikus alkalmazássá fejlesszük amit csináltunk. Válallatirányítási rendszert vezessünk be, és abba csatlakozzanak be az új elemek       * Esetleg közös kommunikációs szabány, megfelelő middleware, integrácós központok       * Huszárvágás: legyen az egész alkalmazás moduláris, és csak 2 interfészt kell beiktatni akkor   + **EAI: Enterprise Architecture Integration: Middlewaare**     - Köztes rendszer egy közös kommunikációs alrendszer legyen, ami központi kommunikációs postás gyakrolatilag     - Egységes adattovábbítás a palatformok között     - **Megvalósítás:** elosztott legyen, vagy integrációs központ? Rétegrendszer?   **Üzleti folyamatok:**   * Ha már van egy kommunikációs centrum, akkor használjuk ki hogy van.   + Naplózást építünk rá, monitorozzuk őket * Ötlet: válasszuk szét az üzleti logikát egy állandó és egy gyakran változó részre.   + Állandó: vállalati alkalmazásokban lesz   + Változó: **üzletifolyamat-alapú vezérlés veszi át (BPM, Business Process Management)**, az alkalamzáson kívül valósítjuk meg * **BPMS: Business Process Management System rendszerek**   + Üzleti folyamatok szétválasztása **pragmatikus** fejlesztéi modellek segítségével   + Válalalti **szolálgatásorinetált architektúra (SOA) épül vele**   + Segítségével gyakran változó üzleti logika könnyen cserélhető, egy másik szakterületi nyelv segítségével irányítható   + **Funkciók:**     - Folyamatok automatikus vezérlése, monitorozása     - Statisztikák és jelentlséek, teljesítményértékelés, ügyfélezelés, stb.   + **Megvalósítás**     - Automatizált folymatokat felmérjük     - Ezek kapcsolatait felmérjük     - Folyamatokaat implementálun BPEL Business Process Exectuin Langeuge nyelven.       * Ez sima XML szabány, nyítl, Microsoft + IBM       * Egyszerűbb rpogramozási nyevl, változókezelés, adatműveletek, külső interakció, stb.       * Van hozzá általában grafikus tervező eszköz       * Külső moldukat hív meg, kódot futtat le, stb.   + **Három rétegű architektúra**      - Prezentáció - üzleti logika -adatelérés itt is     **Integrációs rendszerek tervezése**   * Úgy kéne csinálni, hogy ne kelljen hoszárvágást csinálni -> sok módszer   + Ami biztos: kell egy integrációs központ   + Ami biztos: kell az alkalmazások között adapter, kommmunikációs szabvány     - Mondjuk XML vagy SOAP * **Közvetlen pont-pont kapcsolat:** minden réteg között van kapcsolat, alkalmazás->szállítás->alkalmazás * **Szolgáltatás szintű kapcsolat:** alkalmazás -> konverzió -> iránmyítás -> szállítás ->… -> alkalmazás   + Bullshitbullshit és kis bullshit |
| **11. Middeware: definíció, middleware típusok és tulajdonságaik (RPC, üzenet alapú, tranzakciós, objektum orientált), típusok összehasonlítása** |
| **FONTOS: ezek az anyagrészek már nem szerepelnek a diákban. Könyv kéne… kéne. Szóval innentől wikipedia.**    **Middleware:**  A *köztes szoftver* (angolul *middleware*) általánosan véve egy olyan [számítógépes](https://hu.wikipedia.org/wiki/Sz%C3%A1m%C3%ADt%C3%B3g%C3%A9p) [szoftver](https://hu.wikipedia.org/wiki/Szoftver), amely az [operációs rendszerek](https://hu.wikipedia.org/wiki/Oper%C3%A1ci%C3%B3s_rendszer) mögötti, azok számára nem elérhető szoftveralkalmazásokat biztosítja. A köztes szoftvert akár *szoftverragasztónak* is nevezhetnénk.[[1]](https://hu.wikipedia.org/wiki/K%C3%B6ztes_szoftver#cite_note-MW-00-1) Ezáltal a köztes szoftver nem része egyértelműen az operációs rendszernek, nem [adatkezelő rendszer](https://hu.wikipedia.org/w/index.php?title=Adatkezel%C5%91_rendszer&action=edit&redlink=1), valamint nem része a [szoftveralkalmazásoknak](https://hu.wikipedia.org/wiki/Alkalmaz%C3%A1s_(sz%C3%A1m%C3%ADt%C3%A1stechnika)) sem. A köztes szoftver megkönnyíti a [szoftverfejlesztők](https://hu.wikipedia.org/wiki/Szoftverfejleszt%C5%91) dolgát a kommunikációs és az [input/output](https://hu.wikipedia.org/w/index.php?title=Input/output&action=edit&redlink=1) feladatok végrehajtásában, így a saját alkalmazásuk sajátos céljára tudnak összpontosítani.    **Middleware típusok:**   * **Tranzakciós rendszerek - ????** * **DBMS rendszerek:**    + A **database management system** (**DBMS**) is a [computer software](https://en.wikipedia.org/wiki/Computer_software) application that interacts with the user, other applications, and the database itself to capture and analyze data. A general-purpose DBMS is designed to allow the definition, creation, querying, update, and administration of databases. Well-known DBMSs include [MySQL](https://en.wikipedia.org/wiki/MySQL), [PostgreSQL](https://en.wikipedia.org/wiki/PostgreSQL), [MongoDB](https://en.wikipedia.org/wiki/MongoDB), [MariaDB](https://en.wikipedia.org/wiki/MariaDB), [Microsoft SQL Server](https://en.wikipedia.org/wiki/Microsoft_SQL_Server), [Oracle](https://en.wikipedia.org/wiki/Oracle_Database), [Sybase](https://en.wikipedia.org/wiki/Sybase), [SAP HANA](https://en.wikipedia.org/wiki/SAP_HANA), [MemSQL](https://en.wikipedia.org/wiki/MemSQL) and [IBM DB2](https://en.wikipedia.org/wiki/IBM_DB2). A database is not generally [portable](https://en.wikipedia.org/wiki/Software_portability) across different DBMSs, but different DBMS can interoperate by using [standards](https://en.wikipedia.org/wiki/Technical_standard) such as [SQL](https://en.wikipedia.org/wiki/SQL) and [ODBC](https://en.wikipedia.org/wiki/ODBC) or [JDBC](https://en.wikipedia.org/wiki/JDBC) to allow a single application to work with more than one DBMS. Database management systems are often classified according to the [database model](https://en.wikipedia.org/wiki/Database_model) that they support; the most popular database systems since the 1980s have all supported the [relational model](https://en.wikipedia.org/wiki/Relational_model) as represented by the [SQL](https://en.wikipedia.org/wiki/SQL) language.[[*disputed*](https://en.wikipedia.org/wiki/Wikipedia:Disputed_statement)*–*[*discuss*](https://en.wikipedia.org/wiki/Talk:Database#All_SQL.3F)]Sometimes a DBMS is loosely referred to as a "database". * **RPC:**    + A **Remote Procedure Call** (röviden: RPC, "távoli eljáráshívás") olyan, [processzek](https://hu.wikipedia.org/w/index.php?title=Processz&action=edit&redlink=1) közti kommunikáció adott [programban](https://hu.wikipedia.org/wiki/Sz%C3%A1m%C3%ADt%C3%B3g%C3%A9pes_program), mely szerint egy [függvény](https://hu.wikipedia.org/wiki/F%C3%BCggv%C3%A9ny_(programoz%C3%A1s)) vagy procedúra egy másik [címtartományban](https://hu.wikipedia.org/w/index.php?title=C%C3%ADmtartom%C3%A1ny&action=edit&redlink=1) fut (általában másik számítógépen) anélkül, hogy a programozó definiálná ezen kommunikáció működésének részleteit. Tehát a programozó megírja a kódot, és nem kell törődnie azzal, hogy a kód a lokális vagy egy távoli számítógépen fog-e futni.   + **RPC üzenetküldés:**      - A kliens kezdeményez i ->üzenetet küld a szervernek, az hajtson végre egy procedurát egy szerveren.     - A szerver visszaüzeni a választ a kliensnek.     - Amíg a szerver vár az üzenetekre, kliens bokkolva vár. * **Komponens alapú rendszerek**   + Alulról felfelé történő fejlesztés   + A rendszert kisebb alrendszerekre bontjuk, mindegyik önálló alegység   + Az elkézült rendszert pedig közös rendszerbe integráljuk   + Az elkészült alrendszerek illeszkednek a proejktünkhöz, * **MOM:**   + Message Oriented Middleware   Az RPC hibáinak orvoslása találták ki. Egy tipikusan **üzenetsor** menedzselő szoftver, ami pont-pont információtovábbítást tesz lehetővé. A **perzisztens** üzenet aszinkron módon jut el a címzetthez, ezért a küldő **nem blokkolódik**. Az üzenetsor **laza csatolás**t eredményez, az alkalmazások egymástól függetlenül működnek.  **Előnye**: hogy az üzenetek kicsik, továbbításuk olcsó (séma/adat, egy rekord kell), jó teljesítményű (üzenetek prioritásai, terhelésmegosztás, thread pooling), megbízható (üzenet nem vész el, ha megszakad a hálózati kapcsolat). **Megvalósítások**: MSMQ, MQSeries.  A MOM termékek nem csak kizárólag üzenettovábbításra használhatók; általában tartalmaznak üzenetfordító szolgáltatásokat, biztonsági modult, illesztő adaptereket többféle programhoz, hibajavítást, hálózati erőforrás-allokációt, routolási költségbecslést, üzenet priorizálási és igénylési, illetve hatékony debuggolási eszközöket.  Ellentétben az ORB-bal és az RPC termékekkel, a MOM többnyire nem feltételezi, hogy a rendszer megbízható hálózati réteggel rendelkezik. Ha az átviteli réteg megbízhatatlan, megpróbálja a problémákat az adott szinten helyrehozni, amelyen kommunikál. Ez például WAN vagy Internet alapú kommunikáció esetén szokott előfordulni. Két különböző típusú MOM alakult ki:  • sorban állásos (message queuing),  • üzenet átadásos (message passing). |
| **12. Integrációs központ. Szerepe, jelentősége, előnyök, hátrányok. Megvalósított integrációs funkciók. Egyéb szolgáltatások (audit, biztonság, perzisztencia, API)** |
| ???? |
| **13. Adatbázis alapú integráció: előnyök, hátrányok, tranzakciók jelentősége, DBMS szerepe az integrációban, adatbázis alapú szoftver architektúrák (3 v. több réteg). Elosztott adatbázisok, Replikáció, hibatűrő fürtözés** |
| **Adatmegosztás- Data Sharing:**   * Ez volt az első verziója az elosztott rendszereknek * **File alapú adatmegosztás:**   + Legelterjedtebb, legtöbb hátránnyal   + Nem valós idejű, napi-havi adatcsere, backup. Megbízhatatlan, szinkronziálni kell.   + Közös állományformtáum kell. Lockolási problémák, stb. Külön gépen futás -> rémálom a diszk menedzselés * **Adatbázis alapú megosztás:**    + Közös DB   + Ez már erre van felkészítve, elég jó   + SQL főleg, sqlite mobilokon * **Tranzakciók:** acid meg ilyesmi, majd a következő dián.   **Elosztott adatbázis:** fizikailag megosztott, de logikailag egységes adatbázis.   * Kommunikációs költésgek csökkennek, mindenki a hozzá közelebb állóval kommunikál * DB belül szinkronziál * Egy-egy csomópont kiesik: többi átveszi a helyét. * Cserébe nehéz és sebezhető a DB, overhead a belső kommunikáicó * Redundanciával mi legyen? Nagyon nehéz precízen anélkül tartani   **Replikáció:** db-k másolása, amikor közben szinkron tartjuk őket.  **Fürtözés**  A fürtözés (clustering) lehetővé teszi, hogy egy hálózatban a funkciókhoz egyetlen számítógép helyett egy számítógépcsoport segítségével férhetünk hozzá, ezt a csoportot hívják "fürtnek" (cluster). A "fürtözés" tulajdonképpen a terheléselosztás egyik formája, segítségével lehetővé válik, hogy egy hálózatból ilyen csomópontokat egyszerűbben lehessen eltávolítani, vagy hozzádani - mindezt anélkül, hogy a rendszer működésében zavar keletkezne -, mert például automatikus migrációval, adatmentéssel migrálódnak a fizikailag sérült szerverekről az adatok egy másikra. Sokféle megoldás és megvalósítás létezik, de mindegyik célja az ún. rendelkezésre állási idő közelítése a 100% felé. Ideális esetben a fürtözött rendszerek létrehozása egy jól megtervezett és átgondolt terheléselosztási terv részét képezik, ahol különféle tervek és szabályok mentén osztják el (pl. párhuzamos feldolgozás) a beérkező kérelmeket a fürtök között. |
| **14. tranzakció­­s middleware: tranzakciók fogalma és tulajdonságai (ACID), tranzakciós rendszerek célja és tulajdonságai, összehasonlítása az RPC-vel** |
| **Tranzakció:**   * Atomi folyamatr, mely minden lépése meg kell hogy valósuljon, vagy egyik sem. Ezért visszapörgethetőnek kell lennie, és igazolnunk kell hogy ha megtörtén.   **ACID:**   * **Atomicity:** minden utasítás egységes: ha valami hiba történik, akkor nincs köztes állapot, vissza kell tértnünk konzisztens állapotba. Egy esemény atomi, nem osztható tovább, vagy megtörténik minden, vagy semmi. * **Consisteny:** konzisztens állapotok követik egymást * **Isolation:** párhuzamosan futhatnak tranzakciók, de nincs hatásuk egymásra-db szinten sem. Zárolási mechanizmusok * **Durability** : tranzakció halezárult, eredménye nem veszhet el.       **Ez kurvafontos összefoglaló a tárgy teljesen zavaros felépítéséhez képest:**  **Integrációs rendszerek, azaz middleware architektúra alaptípusok fejlődés szerint:**  C:\1083D865\0FF81E2A-E70C-4542-A8AC-7AF5E82FFF4B_elemei\image014.png   1. **Socket:** TCP socket, valós idejű adattovábbítás    * **Általában bináris adattovábbítás, TCP vagy UDP protokoll felett**    * **Állandó csatorna, ez a fapados megoldás** 2. **RPC - Remote Process Call** - **távoli eljáráshívás**     * **Klasszikus szerver-kliens programozás**    * **Ami alatta socket alapon működik, elrejtődik**    * **Csatorna már nem csak adatotovábbításra jó, API hívásokat támogat**    * **Platform független** 3. **ORB - Object Request Broker -**     * **Objektumorientáltság paradigmáit követi**    * **Távoli objektumok szabványosan kommunikálnak egymással**    * **Corba vagy Java-RMI**    * **Marshallign-unmarshalling, paraméterek és adatok szerializációja. Mostmár a csatorna elkódol komplexebb dolgokat**    * **Első megjelenése a nyelvfüggelten service-interface-eknek.**    * **Pl. COM, asszem** 4. **Üzenetkezelő rendszerek**    * **ORB-val párhuzamosan alakulnak ki**    * **Aszinkron**    * **Háttérben még mindig ccoket alapú**    * **Üzenetsorok megjelenése**    * **Nagy előnye a laza kapcsolás**    * **Garantált üzenetküldés, köztes üzenetküldések** 5. **Webservices -**     * **XML-en alapuló kommunikáció**    * **Egyesíti az RPC ORB üzenetküldés előnyeit**    * **WebServiceDefinitionLanguage alkalmazása WSDL**       + **Emiatt lesz nyelv és platformfüggetlen**    * **Universal Description, Discovery and Integration UDDI**       + **Önleíró interfészek**    * **Simple Object Access Protocoll SOAP**       + **Klasszikus szabványos üzenetküldő nyelv** 6. **Enterprise Service Bus ESB**    * **Teljesen heterogén rendszerek szorosabban csatolt, specifikus és jó összeközése**    * **Protokoll és üzenet transzformáció,**    * **Message routing**    * **Minden kapcsolatra személyre szabható hogy hogyan valósítjuk meg**    * **Pl. JavaEE** |
| **15. Üzenet alapú middleware: üzenet sorok, közzétevő-előfizető (publish-subcriber) architektúra, RPC és üzenet alapú megoldások összehasonlítása** |
| **Üzenet alapú middelware:**   * Lásd, fentebbi felsorolás. * RPC és ORB amik erre épülnek. * Üzenet lehet szinkron vagy aszinkron. Marshallign, unmarshalling itt is van   **MOM - Message Oriented Middleware**   * A alkalmazás létrehozza az üzenetet * Elküldi azt a MOM-nak * Az üzenetkezelő átküldi az üzenetet * B alkalmazás olvas a sorból, majd feldolgozza az eseményt * C:\1083D865\0FF81E2A-E70C-4542-A8AC-7AF5E82FFF4B_elemei\image015.png   **Publish-subscriber architektúra:**   * C:\1083D865\0FF81E2A-E70C-4542-A8AC-7AF5E82FFF4B_elemei\image016.png * Feliratkozók mindegyike megkapja az ünzetetet * Üzenet fejlécében minden infó benne van * Pl. CORBA vagy Java Messaging Service * **Vs. RPC:**   + **RPC:** Remote Process Call, távoli eljárás hívás - szerver kliens megvalósulás     - Csatorna távoli eljárások végrehajtására     - Interfészleírásra szükség van, csatorna dedikált,   + Itt nincs szükség csatornára   + Aszinkron több szolgáltatással is lhete kommunikálni |
| **16. Vállalat irányítási rendszerek. Célok, eszközök, tulajdonságok. Testreszabás, tesztelés.** |
| **Vállalatirányítási rendszer:**  Akkor lehet egy vállalat irányítási rendszeréről beszélni, ha meghatározott a vállalat stratégiája (politikája és céljai), majd a vállalat vezetése a vállalat összes tevékenységét folyamatokba rendezve, ezek működésére – valamely vezérlő elv(ek) figyelembe vételével – meghatározta a folyamatok végrehajtási szabályait (a belső játékszabályokat). Ekkor a vállalat - a stratégiának és céljainak megvalósítása érdekében - tervezetten és szabályozottan működik, ahol ebbe a szabályozott működésbe beleértettük a folyamatos javítás, és az elvégzett tsevékenységek illetve eredmények szükséges igazolásának a rendszerét is.    **Típusok:**   * **Minőségirányítási rendszer:** cég számára cél a működésének átláthatósága, stabilitása nagyon fontos. ISO 9001-es szabvány. A cél hogy kiszolgáljuk a usereket. * **Környezetközpontú irányítási rendszer:** vállalat a környezetszennyezést akarja csökkenteni… kajak faszom. * **Információbiztonsági irányítási rendszer:** működéés és a működőképesség folyamatosa fenntartása és biztosítása a különböző vesélyekkel szemben. Első sorban informatikai értelemben. * **Egészségvédelmi rendszer….**     A **vállalatirányítási (*corporate governance*) információs rendszer** – a szakirodalomban egyre inkább ERP-ként emlegetett [információs rendszer](https://hu.wikipedia.org/w/index.php?title=Inform%C3%A1ci%C3%B3s_rendszer&action=edit&redlink=1) – a [vállalat](https://hu.wikipedia.org/wiki/V%C3%A1llalat) környezetére, belső működésére és a vállalat–környezet tranzakcióira vonatkozó információk koordinált és folyamatos beszerzését, feldolgozását, tárolását és szolgáltatását végző személyek, tevékenységek, valamint a funkciók ellátását lehetővé tevő hardver- és szoftvereszközök összessége.  Szigorúan véve az ERP elnevezés egy szűkebb kategóriára vonatkozik: a vállalati erőforrás-tervező (*Enterprise Resource Planning*) rövidítése. A gyakorlatban azonban az egész rendszer elnevezésére is meghonosodott az ERP rövidítés.    **Az információs rendszer fő összetevői:**   1. az [ember](https://hu.wikipedia.org/wiki/Ember) mint döntéselőkészítő és döntéshozó; 2. a külső és belső [információ](https://hu.wikipedia.org/wiki/Inform%C3%A1ci%C3%B3); 3. a külső és belső [hardver](https://hu.wikipedia.org/wiki/Hardver) és [szoftver](https://hu.wikipedia.org/wiki/Szoftver) elemek, szervezeti megoldások (azaz az [orgver](https://hu.wikipedia.org/w/index.php?title=Orgver&action=edit&redlink=1)). |
| **17. Döntéstámogató rendszerek. Adattárházak jellemző tulajdonságai, felépítése. Extraction-Transformation-Load folyamatok. Adatvizualizáció alapjai.** |
| **Döntéstámogató rendszerek:**   * **Decision Support System** * **Interaktív szoftverek, melyek csoportok, közösségekk hatkonyab működését vagy üzleti folyamatok előrejelzését követését tszik lehetővé.** * Típusai:   + Adatorientált vagy adatvezérelt   + Kommunikáció orinetált   + Dokumentumorientált   + Tudásorientált   + Szimuláció oriente.t * Informatikában általában adatorientált döntéstámogatásról beszélünk. * Elsődleges módszer az a**adatok elemzése, hozzáférése, belőlük információ kinyerése, prezentálása.** * Adatbányászat, szövegbányászat… struktúrált adatszerekezetek * A tudát értelmezése üzleti döntés * Fókusz lehet a múltra, jövőre, stratégiára, stb. * Hat szintjük van:   + Adatbázis   + Adattárház -> sok adatbázis   + Adatkinyerés, tisztítás -> lényeges dolgok kinyerése   + Adatbányászat -> algoritmus   + Riportkészítés, vizualizáció -> megmutatni az infókat   + Kognitív érzékelés -> emberi döntések     Az OLAP is egy döntéstámogató rendszer ->  **Extraction-Transformation-Load:**  **ETL:** <https://en.wikipedia.org/wiki/Extract,_transform,_load>  In [computing](https://en.wikipedia.org/wiki/Computing), **Extract, Transform and Load** (**ETL**) refers to a process in [database](https://en.wikipedia.org/wiki/Database) usage and especially in [data warehousing](https://en.wikipedia.org/wiki/Data_warehouse) that:   * [Extracts](https://en.wikipedia.org/wiki/Data_extraction) data from homogeneous or heterogeneous data sources * [Transforms](https://en.wikipedia.org/wiki/Data_transformation) the data for storing it in the proper format or structure for the purposes of querying and analysis * [Loads](https://en.wikipedia.org/wiki/Data_loading) it into the final target (database, more specifically, [operational data store](https://en.wikipedia.org/wiki/Operational_data_store), [data mart](https://en.wikipedia.org/wiki/Data_mart), or [data warehouse](https://en.wikipedia.org/wiki/Data_warehouse))   The typical real-life ETL cycle consists of the following execution steps:   1. Cycle initiation 2. Build [reference data](https://en.wikipedia.org/wiki/Reference_data) 3. Extract (from sources) 4. [Validate](https://en.wikipedia.org/wiki/Data_validation) 5. Transform ([clean](https://en.wikipedia.org/wiki/Data_cleaning), apply [business rules](https://en.wikipedia.org/wiki/Business_rule), check for [data integrity](https://en.wikipedia.org/wiki/Data_integrity), create [aggregates](https://en.wikipedia.org/wiki/Aggregate_(data_warehouse)) or disaggregates) 6. Stage (load into [staging](https://en.wikipedia.org/wiki/Staging_(data)) tables, if used) 7. [Audit reports](https://en.wikipedia.org/wiki/Audit_report) (for example, on compliance with business rules. Also, in case of failure, helps to diagnose/repair) 8. Publish (to target tables) 9. [Archive](https://en.wikipedia.org/wiki/Archiving)     **Adatvizualizáció dióhélyban:**   * Hogyan jelenítjük meg az adatokat. Adat vs. Design -> ezt értik meg a userek, meg néha mi is.   Az adatvizualizációval foglalkozó szakember legfontosabb aranyszabálya az, hogy meg kell találni az arányt a precíz adatmennyiség és a design között. Ha túl sok az illusztráció, akkor abban elveszhet az információ, ha viszont túl sok az adat, az olvasó nehezen értelmezi, amit lát, és éppen az illusztráció célja veszik el. (Wiederkehr, 2011)   * Grafikon, táblázat, többdimenziós táblázat, Excel |
| **18. Két és háromrétegű adattárház architektúra: kétrétegű adattárház építése, adattisztítás, adat-transzformáció, indexelés, adatpiacok.** |
| **Adattárház:**  Az **adattárház** egy szervezet történeti adatainak fő tárhelye. Az adatait a szervezetben található más informatikai rendszerektől - adattárház terminológia szerint a *forrásrendszerektől* - veszi át, a célja pedig, hogy olyan rendszereket lásson el adatokkal, mint a [döntéstámogató rendszerek](https://hu.wikipedia.org/wiki/D%C3%B6nt%C3%A9st%C3%A1mogat%C3%B3_rendszer), a [vezetői információs rendszerek](https://hu.wikipedia.org/w/index.php?title=Vezet%C5%91i_inform%C3%A1ci%C3%B3s_rendszer&action=edit&redlink=1) vagy [adatbányász](https://hu.wikipedia.org/wiki/Adatb%C3%A1ny%C3%A1szat) szoftverek. Az adattárház a vezetők számára optimalizált formában, a vezetői információigényeknek megfelelő aggregáltságú és szervezettségű adatokat tartalmaz, illetve lehetővé teszi ezek sokoldalú lekérdezését, magas szintű analitikus műveletek végzését.  Az adattárházak létét elsősorban az indokolja, hogy egy szervezet (vállalat, hivatal) napi működését támogató informatikai rendszerek az egyes tranzakciók (adatbevitel, adatmódosítás módosítás) egyszerű és gyors végrehajtására vannak optimalizálva, így kevéssé alkalmasak a sok adatot összevontan kezelő elemzői/vezetői lekérdezések kiszolgálására. Az adattárházak alapkoncepciója, hogy a megváltozott és az új adatokat adott gyakorisággal (jellemzően naponta) *egyszer* átveszik a forrásrendszerektől, majd minden további elemzői/vezetői lekérdezést, illetve rendszert már az adattárház szolgál ki, ezzel mentesítve az alaprendszert az ilyen erőforrásigényes és nehezen előrejelezhető terhelésektől. Az adattárház adattárolása (jó esetben) éppen az ilyen sok adatot érintő lekérdezések kiszolgálására van optimalizálva, így az elemzői/vezetői riportok is gyorsabban futnak.  **Kulcsszó: historikus!**  **Adatpiac: vállalat kisebb csoporjta számára készült, konkrét feladatot ellátó, kisebb adattároló és anaizáló egységet jelent, mely önmagában is adattárház funkciókat lát el.**  **Vagy**  **Adattárház**: az összegzett és megtisztított adatok közös, témaorientált, rendezett és tartós tárolóhelye. Üzleti intelligencia: teljes folyamat az adattárházon át(ábra).  **Adatpiac**: Tranzakciós rendszerekből nyert, összegzett és megtiszttott adatok közös, témaorientált, rendszerezett és tartós tárolóhelye, amely csak egy speciális felhasználói csoport kiszolgálását célozza meg.    C:\1083D865\0FF81E2A-E70C-4542-A8AC-7AF5E82FFF4B_elemei\image017.png    **Adattárház építése:**   1. **Big Bang módszer:** felmérik a különböző igényeket, majd felépítik az adattárházat és az adatpiacot is. Top-down hozzáállás 2. **Inkrementális:** igény jelentkezése során felépítenek néhány adatpiacot. Aztán ezeket lehet folyton hozzáadni, újra megcsinálni. Integrálod, amikor kell, addig születhetnek az adattárházak.     **Adatranszformáció:** adatok kinyerése után transzformálódnak, saját mértékegységre, nyelvre, formátumra. Stb.  **Adattisztítás:** felesleges adatok eltörélése, vagy hibásak kiszűrése,stb.   * **Eszközei:**   + **Zajszűrés :** hibás adatok   + **Klaszterezés:** csoportok kialakítása   + **Osztályozás:** osztályokba sorolás   + **Regresszió, normalizálás, szótárak, ismétlődések szűrése, hiánykezelés, törlés, modellezés, átlagolás.**   **Indexelés:** A relációs tárolás hatékonyságát, a lekérdezések válaszidejét nagyban javíthatja megfelelő indexstruktúrák alkalmazása. Kutatás tárgyát képezi az adattárházak struktúráival hatékony módon működő indexelési módszerek kialakítása. Két már bevált és hasznos ilyen jellegű index-típus az úgynevezett bitmap index, valamint a join index. Az indexelés témaköre nagyban összefügg az előző pontban tárgyalt aggregátum-kezelés témakörrel: min  dkét témakör olyan, adatbázisban járulékosan tárolt, redundáns adatok kezelésével, kialakításával foglalkozik, melynek célja a lekérdezések válaszideinek csökkentése. |
| **19. OLAP rendszerek célja, alapvető működése. Multidimenzionális adatmodell, OLAP architektúra: dimenziós és tényadatok, csillag- és hópehely séma, OLAP kocka felépítése, alapvető OLAP-kocka műveletek** |
| **OLAP**: **Online Analytical Processing** egy tisztított és előfeldolgozott adatokra épülő döntéstámogató rendszer, amelynek fő célja a többdimenziós adatokra vonatkozó statisztikai jellegű kérdések gyors megválaszolása és a válaszok szemléletes megjelenítése. Codd­szabályok az OLAP definiálásához:   1. . Multi­dimenzionális nézet 2. Transzparencia (itt most technikai részletek ismerete nélküli könnyű elérhetőség, tehát :áttekinthetőség értelemben) 3. Elérhetőségek (jogosultságok) beállíthatósága 4. Állandó riportozási (lekérdezési) teljesítmény 5. Kliens­szerver architektúra 6. Általános dimenzió fogalom 7. Dinamikus ritka­mátrix kezelés (ez a multidimenzionális modell tárolására vonatkozik, megvalósításra megkötés) 8. Több konkurens felhasználó támogatása 9. Korlátozás nélküli dimenzióműveletek 10. Intuitív adatkezelés (a végfelhasználó számára) 11. Rugalmas riportozás (vagyis beszámoló­készítés, lekérdezés) 12. Korlátlan dimenziószám és aggregációs szint szám   **FASMI**:   * Gyors(Fast): A rendszer interaktív működésű, vagyis rövid időn belül választ kell adnia. * Elemzési képesség(Analysis): Az OLAP rendszer lehetőséget teremt változatos, dinamikusan összeállított üzleti és statisztikai elemzések elvégzésére. * Többfelhasználós környezet(Shared): Az adatokat több felhasználó párhuzamosan használhatja (biztonság) * Multidimenzionális(Multidimensional): többdimenziós adatábrázolás és adatkezelés * Információ(Information): az OLAP rendszer tartalmazza az összes közvetlen és származtatott adatot a komplex számítások elvégzéséhez.     **Dimenzió:** szintek hiearchiájaáának együttese egy kockában, földrajzi hely pl. tehát egy tengely.  **Dimenziótábla:** csillag séma dewfinícióhoz tartozik. Egy dimenziótábla attribútumai az olap kocka dimenzióit jelentik.  **Tényadat:** konkrét számok a különböző dimenziók metszetében.  **Csillag séma:** úgy mutatjuk az olap kockát, hogy középen vannak a kulcsok, abban van az összes tényadat, és ahhoz kapcsolódnak a dimenziók körbe. Ezekből lehet a kockát építeni  C:\1083D865\0FF81E2A-E70C-4542-A8AC-7AF5E82FFF4B_elemei\image018.png   * **Előnye,** hogy egyszerű, intuitív, kevés tábla olvasása kell hozzá, **hátránya** hogy nehézkes a naagy aggregációk elkészíteése.   **Hópehely séma**   * Hasonló, mint a csillag, **csak itt normalizáljuk a dimenziótáblákat** -> azaz megszűntetjük a redundanciát benne. * C:\1083D865\0FF81E2A-E70C-4542-A8AC-7AF5E82FFF4B_elemei\image019.png     <http://scs.web.elte.hu/Work/DW/adattarhazak.htm>    **A multidimenzionális adatmodell prezentálása és tárolása**  Az adatkocka a multidimenzionális adatok tárolási modelljétől független adatelemzési egysége, ROLAP­, MOLAP­, HOLAP­tárolás OLAP kocka: a kocka oldalai  mentén tüntetjuük fel a dimenziókat, míg a dimenziók által meghatározott metszéspontokban található cellák a tényadatokat tartalmazzák.   * **ROLAP(Relational OLAP)**: a multidimenzionális adatmodellt relációs adatbázisra képezik le csillag és hópehelysémák használatával → relációs jó, mert a fejlesztők már ismerik, sok eszköz van rá, skálázható… * **MOLAP(Multidimensional OLAP)**: az adatok tárolására multidimenzionális adatmodell. Ez a modell az egyes elemeket többdimenziós vektorban tárolja, ahol azok közvetlen indexeléssel hozzáférhetők. * **HOLAP(Hybrid OLAP**): ötvözik a relációs megközelítés skálázhatóságát a multidimenzionális tárolás gyorsabb adatelérési és számítási képességeivel. ROLAP vs MOLAP tárhelykezelés: ritka adatok kezelésénlé MOLAP ugyanannyi multidimenziós adat tárolásához akár 2­10­szer kevesebb helyet igényel, mint a ROLAP.     **Műveletek az OLAP-kockával**   * szeletelés(slice): a hiperkocka egyik dimenzióját rögzítjük, és e rögzített dimenzió értéke mentén hajtunk végre elemzést. * részkocka­kiválasztás(dice): az eredeti hiperkocka dimenzióit csak a kiválasztott értékek mentén vizsgáljuk. (Tehát a kocka egy kisebb részét vizsgáljuk csak.) * Lefúrás(drill­down): megnöveli az egyik dimenzió felbontását (éves→negyedéves) * Felgörgetés(roll­up): egy adott dimenzió mentén egy magasabb hierarchiaszintre emelkedünk. (hónap→negyedév) * elforgatás(pivot): a koordinátatengelyek sorrendjét vagy azok irányát felcseréljük megjelenítési okok miatt. (A vizsgált adatok térbeli elforgatása szemléletesebb eredménymegjelenítés érdekében.) * további műveletek: átfúrás(drill­across) OLAP kockák közötti elemzés; keresztülfúrás (drill­through): a kocka felépítéséhez használt operatív adatokhoz tudunk hozzáférni.     **Előkalkuláció:** válaszidő javulása miatt, előre kiszámolok dolgokat. A teljes előkalkuláció nagyon sok idő lenne, de van jó határvonal. Részleges előkalkuláció a gyakran használt aggregátumok előzőetes meghatározása. Azokra az adatokra, amiket statisztikailag gyakran használunk    **Az adatbázis-robbanás**: Az a szívás az OLAP-palk, hogy az előkalkuláció és dimenziók hozzáadásával aránytalanul nagyot növekszik az adatbázis. Fontos, hogy ne találjunk ki hülyeséget előkalkulációnak és dimenzióknak, és nem lesz gond. |
| **20. Adatbányászati módszerek és algoritmusok áttekintése. Klaszterezés, osztályozás, társítási szabályok, szekvencia bányászat.** |
| **Adatbányászat:** Az adatbányászat nagy adathalmazokon végzett önálló tudásfeltárás. Felhasználási területek: kereskedelem, pénzügy, távközlés, orvostudomány adatbányászat: egy interdiszciplináris tudományterület, amely a nagy adathalmazokból történő tudásfeltárással foglalkozik, vagyis az adatokban rejlő újszerű, nem triviálisan előállítható összefüggések és minták automatikus kinyerését célozza.    **Adatbányászat folyamata:**   1. adatok, dokumentumok begyűjtése 2. Adatok, dokumentumok megtisztítása, előkészítése 3. Adatok feldolgozása adatbányászati módszerekkel 4. Kinyert eredmények értelmezése, szabályok hatásosságának eldöntése 5. Prezentálás, értelmezés     **Adatbányászat módszerei:** <https://www.aut.bme.hu/Pages/Research/DataMining/Algorithm>   * **Osztályozás:** előre elkészített csoportok, akikhez vannak jól elkülönülő szabályok vagy konkrét példák akiken be tud tanulni az algoritmus. Ezek után új elemeket már a megtanított algortimus be tud sorolni.   + **Felügyelt tanulási módszertan**   + Pl. Döntési fák, Regresszió, Neurális hálózatok, Bayes-i osztályozók, legközelebbi szomszédok * **Klaszterezés:** előre nincsenek csoportok, hanem menet közben határozzuk meg őket.   + **Felügyelet néélküli tanulási modszertan**   + Pl. partícionáló algoritmusok, sűrűségalapú algoritmusok, hierarchikus fa szerkezet kialakítása stb. * **Társítási feladatok:**   + Asszociációs szabályokat alakítunk ki, bizonyos elemzések mentén. Mintákat keresünk első sorban,   + Pl. a vásárlók kosarában mely item-ek jelennek meg, és köztük milyen kapcsolat mutatható ki.   + Általában különböző bejárási szélességi és mélységi algoritmusok * **Szekvencia vagy mintabányászat:**   + Gyakran felőforduló minták megtalálása a cél. Ez a minta lehet egyszerű elemhalmaz, vagy komplex szöveg   + Pl. a Szövegbányászat ilyen szekvenciabányászat lehet.   + Pl. vásárlási módszerek   VAGY  **Data mining involves six common classes of tasks:**[**[4]**](https://en.wikipedia.org/wiki/Data_mining#cite_note-Fayyad-4)   * [Anomaly detection](https://en.wikipedia.org/wiki/Anomaly_detection) (outlier/change/deviation detection) – The identification of unusual data records, that might be interesting or data errors that require further investigation. * [Association rule learning](https://en.wikipedia.org/wiki/Association_rule_learning) (dependency modelling) – Searches for relationships between variables. For example, a supermarket might gather data on customer purchasing habits. Using association rule learning, the supermarket can determine which products are frequently bought together and use this information for marketing purposes. This is sometimes referred to as market basket analysis. * [Clustering](https://en.wikipedia.org/wiki/Cluster_analysis) – is the task of discovering groups and structures in the data that are in some way or another "similar", without using known structures in the data. * [Classification](https://en.wikipedia.org/wiki/Statistical_classification) – is the task of generalizing known structure to apply to new data. For example, an e-mail program might attempt to classify an e-mail as "legitimate" or as "spam". * [Regression](https://en.wikipedia.org/wiki/Regression_analysis) – attempts to find a function which models the data with the least error that is,for estimating the relationships among data or datasets. * [Summarization](https://en.wikipedia.org/wiki/Automatic_summarization) – providing a more compact representation of the data set, including visualization and report generation. |

Még hasznos linkek:

* Adattárházakról: <http://scs.web.elte.hu/Work/DW/adattarhazak.htm>
* Tárgyoldal és néhány jegyzet: <https://www.aut.bme.hu/Course/VIAUM209>
* Wikioldal: (van a hiányzó két témakörből egy közepesen outdate-ed pdf rajat) <https://wiki.sch.bme.hu/Integr%C3%A1lt_inform%C3%A1ci%C3%B3s_rendszerek>

**VIZSGAKÉSZÜLÉSHEZ ÍRT JEGYZETEM**

Tippek vizsgához:

* tipuspéldák
* tegezd a tanárt
* utolsó előadásokból csak a könyvben van jegyzet

**Kérdések a wikiről**

**Tudathasadás:**

Klaszterek témakör. (Kalszter: hálózattal összekapcsol, önálló (saját operációs rendszerrel rendelkező) elemekből álló, egységes erőforrásnap látszó rendszer.

Tudathasadás az, amikor csomópontok aktívan versenyeznek egymással, és párhuzamos diszk írással maradandó hibákat okozhatnak.

Elkerülni ezt úgy lehet, ha

* Kerüljük a kialakulását (amikor a csomópontok egymást tévesen hibásnak érzékelik)
  + Redundáns életjelhálózatok (tilos a csatolás, pl. közös többportos hálózati kártya, közös életjel hub, külön hub-ok, de azonos szünetmentes tápegységbe dugva)
  + A hasznos működésre képtelen csomópont visszalépése a szolgáltatástól (minden külső hálózati kapcsolata megszakad, nem tud írni a közös lemezre, mert a másik lefoglalja)
  + Kézi rendszerműködés-felfüggesztés, és főleg az ezutáni újraindítás tiltása
* A károkozás elkerülése
  + A partner tápellátásának lekapcsolása
  + Párhuzamos írás elleni védelem a diszkvezérlő szintjén
  + Felügyelt/óvatos visszakapcsolódás

**Log shipping:** “Log shipping is the process of automating the backup of a database and transaction log files on a primary (production) database server, and then restoring them onto a standby server. This technique is supported by Microsoft SQL Server, 4D Server, MySQL and PostgreSQL. Similar to replication, the primary purpose of log shipping is to increase database availability by maintaining a backup server that can replace production server quickly.

Compared to database replication, log shipping does not provide as much in terms of reporting capabilities, but also backs up system tables along with data tables, and locks standby server from users' modifications. A replicated server can be modified (e.g. views) and therefore is not suitable for failover purposes.” *(source:* [*https://en.wikipedia.org/wiki/Log\_shipping*](https://en.wikipedia.org/wiki/Log_shipping)*)*

**MOM:**

Message Oriented Middleware

Az RPC hibáinak orvoslása találták ki. Egy tipikusan **üzenetsor** menedzselő szoftver, ami pont-pont információtovábbítást tesz lehetővé. A **perzisztens** üzenet aszinkron módon jut el a címzetthez, ezért a küldő **nem blokkolódik**. Az üzenetsor **laza csatolás**t eredményez, az alkalmazások egymástól függetlenül működnek.

**Előnye**: hogy az üzenetek kicsik, továbbításuk olcsó (séma/adat, egy rekord kell), jó teljesítményű (üzenetek prioritásai, terhelésmegosztás, thread pooling), megbízható (üzenet nem vész el, ha megszakad a hálózati kapcsolat). **Megvalósítások**: MSMQ, MQSeries.

A MOM termékek nem csak kizárólag üzenettovábbításra használhatók; általában tartalmaznak üzenetfordító szolgáltatásokat, biztonsági modult, illesztő adaptereket többféle programhoz, hibajavítást, hálózati erőforrás-allokációt, routolási költségbecslést, üzenet priorizálási és igénylési, illetve hatékony debuggolási eszközöket.

Ellentétben az ORB-bal és az RPC termékekkel, a MOM többnyire nem feltételezi, hogy a rendszer megbízható hálózati réteggel rendelkezik. Ha az átviteli réteg megbízhatatlan, megpróbálja a problémákat az adott szinten helyrehozni, amelyen kommunikál. Ez például WAN vagy Internet alapú kommunikáció esetén szokott előfordulni. Két különböző típusú MOM alakult ki:

• sorban állásos (message queuing),

• üzenet átadásos (message passing).

**ERP:Enterprise Resource Planning (ERP)**

A vállalatirányítási (corporate governance) információs rendszer – a szakirodalomban egyre inkább ERP-ként emlegetett információs rendszer – a vállalat környezetére, belső működésére és a vállalat–környezet tranzakcióira vonatkozó információk koordinált és folyamatos beszerzését, feldolgozását, tárolását és szolgáltatását végző személyek, tevékenységek, valamint a funkciók ellátását lehetővé tevő hardver- és szoftvereszközök összessége.

Szigorúan véve az ERP elnevezés egy szűkebb kategóriára vonatkozik: az erőforrás-tervező alrendszer (Enterprise Resource Planning) rövidítése. A gyakorlatban azonban az egész rendszer elnevezésére is meghonosodott az ERP rövidítés.

Az információs rendszer fő összetevői:

1. az ember, mint döntéselőkészítő és döntéshozó;
2. a külső és belső információ;
3. a külső és belső hardver és szoftver elemek, szervezeti megoldások (azaz az orgver).

(wikipedia)

vállalatirányítási szoftver bevezetése: ilyenkor a vállalat megvesz egy dobozos terméket pl.: SAP, amely konkrét megoldásokat tartalmaz bizonyos vállalati egységekhez, és biztosítja az ezek közötti kapcsolatot is. A „best practice”-ek problémája: nem illeszkednek pontosan a konkrét vállalathoz, ezért mindig elengedhetetlen a konfiguráció, rendszerre szabás, amely sok időt vesz igénybe.

**Előnyök**:

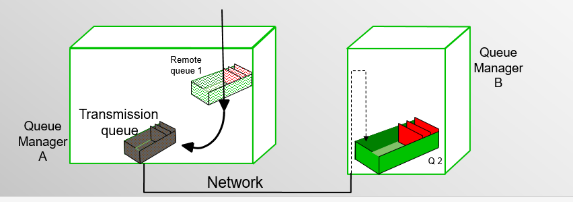
* a szakterület „best practice”-einek átvétele a vállalat életébe
* szabványos megoldások
* támogatja a későbbi változtatásokat az integrálás és az egyes alkalmazások életében

**Hátrányok:**

* drága
* lassú, fokozatos áttérés az új rendszerre, ez nagy vállalatok életében akár 1-2 év is lehet
* best practice nem feltétlenül illeszkedik a folyamatokhoz, akár a vállalati folyamatokon is alakítani kell
* csak nagy vállalatok esetében éri meg

**Transmission Queue**

IBM WebSphere MQ architektúránál volt említve. Egy Queue tipus (no shit),



* Csak definíciós információk (másik queue manager neve, queue neve, transmission queueu)
* egy másik queue manager local queue-jára mutat
* csk írható

Hogyan is mûködnek a queuek ? Két alapvetõ queue fajta van

- lokális (local) - saját queue manager

- távoli (remote) - távoli queue manager

Egy alkalmazás nem is tudja, hogy olyan queueba írja az adatokat amelyek a saját queue managerhez tartoznak vagy egy távoli queueba. Üzenetek olvasása nem lehetséges csak lokális queuekból.

Emlékezzünk vissza az aszinkron üzenettovábbításra illetve az osztott tranzakció-kezelésre. Az író alkalmazás számára a feladat (tranzakció) akkor fejezõdik be, amikor az üzenetet sikeresen beírta a queueba (Q1 vagy a Transmission Queue). Ha távoli queue a címzett, a transmission queueból az MQ gondoskodik arról, hogy az adatok átkerüljenek a címzett queue manager megfelelõ queuejába.

**GUI alkalmazásintegráció:**

A prezentációs réteg általában emberi felhasználásra készül, integrációra kevéssé alkalmas, akkor miért használjuk:

* Bizonyítottan működik (eddig is ezen keresztül kommunikáltak az alkalmazással, igaz, hogy emberi közvetítéssel)
* Néhol nincs is más lehetőség, egy mainframe rendszer pl. technikai okokból és szaktudás hiányában nem is érhető el másképpen, de lehet, hogy egy legacy alkalmazás semmilyen dokumentált felülettel sem rendelkezik
* Az új alkalmazások között a web világban tipikus, hogy csak az alkalmazások felhasználó felülete érintkezik egymással
  + Portál technológiák a különböző forrásból származó alkalmazások és adatok egységes keretbe fogására (Web Service For Remote Portlets, JAVA Portlet Specification)
  + Az alkalmazások hozzák az adatokat, és a portletek illesztik be ezeket a portál egységes környezetébe

Lényege, hogy a felhasználót egy programmal megszemélyesítjük

* Billentyű és egér eseményeket küldünk a forrás alkalmazásnak
* A képernyőképekből olvassuk ki az eredményt

A képernyőképek feldolgozása során

* Használhatunk szabványos eszközöket (IBM HLLAPI: egy terminálemuláció mainframe-ek eléréséhez, OLE Automation technológia Windows környezetben)
* A kinyert adatokat szöveg folyammá (screens as data) vagy objektumokká (screens as objects) alakítva adjuk át a fogadó alkalmazásnak

**Hogyan alakítjuk ki a kapcsolatot?**

* A forrás alkalmazás adatszerkezetének feltérképezése (logikai objektum modell)
* Képernyő katalógusok készítése (melyik adathoz melyik képernyőn keresztül férhetünk hozzá)
* Egy adatelem nem feltétlenül rendelhető össze egy képernyőképpel (egy képernyőn több adattábla összefűzésével is születhet, illetve az azonos rekordhoz tartozó információk több képre is szét lehetnek szórva)
* A kiolvasás történhet
  + Statikus leképzéssel (képernyő koordináták), ez egyszerű, de a hibakezelés nehéz, a különböző verziókban a képernyőkép változhat
  + Dinamikus leképzéssel, feldolgozzuk a képernyőt és pl. az előtte álló fejléc után keressük az adatot, illetve a hibaüzeneteket is felismerhetjük

A GUI nem gépi kommunikációra tervezték

* Használata lassú, körülményes
* Rosszul skálázható, erőforrás igényes, de
* Maximálisan figyelembe veszi az üzleti és biztonsági szabályokat, ezért gyakran az adatréteg szintű integrációnak is alternatívája lehet

A teljesítménye javítható

* Párhuzamosítás, (a mainframe rendszereket éppen sok felhasználó párhuzamos kiszolgálására tervezték)
* Cache-ek létrehozása, a válaszokat megjegyezzük, és nem fordulunk minden alkalommal az eredeti rendszerhez
* Speciális adatkinyerő képernyők használata, ha a szaktudás megvan, a sávszélesség így is növelhető

**Előnyök**:

* Mindig működik
* Egyszerű (de nagyon munkaigényes) megvalósítás
* Az üzleti és biztonsági funkciók újrahasznosulnak, konzisztens
* Kapcsolattartás éppen olyan jól és egyszerűen működhet, mint a többi rétegben (sebesség trükkök, objektum csomagolás)

**Hátrányok:**

* A UI nem erre lett tervezve
* Alapvetően lassú, erőforrás-igényes működés
* A képernyő-leképzések előállítása költséges és hosszú idő
* Nehéz a hibakezelés megvalósítása

**Adatpiac:**

Adattárház: az összegzett és megtisztított adatok közös, témaorientált, rendezett és tartós tárolóhelye. Üzleti intelligencia: teljes folyamat az adattárházon át(ábra).

Adatpiac: Tranzakciós rendszerekből nyert, összegzett és megtiszttott adatok közös, témaorientált, rendszerezett és tartós tárolóhelye, amely csak egy speciális felhasználói csoport kiszolgálását célozza meg.

**ETL:**

In [computing](https://en.wikipedia.org/wiki/Computing), **Extract, Transform and Load** (**ETL**) refers to a process in [database](https://en.wikipedia.org/wiki/Database) usage and especially in [data warehousing](https://en.wikipedia.org/wiki/Data_warehouse) that:

* [Extracts](https://en.wikipedia.org/wiki/Data_extraction) data from homogeneous or heterogeneous data sources
* [Transforms](https://en.wikipedia.org/wiki/Data_transformation) the data for storing it in the proper format or structure for the purposes of querying and analysis
* [Loads](https://en.wikipedia.org/wiki/Data_loading) it into the final target (database, more specifically, [operational data store](https://en.wikipedia.org/wiki/Operational_data_store), [data mart](https://en.wikipedia.org/wiki/Data_mart), or [data warehouse](https://en.wikipedia.org/wiki/Data_warehouse))

The typical real-life ETL cycle consists of the following execution steps:

1. Cycle initiation
2. Build [reference data](https://en.wikipedia.org/wiki/Reference_data)
3. Extract (from sources)
4. [Validate](https://en.wikipedia.org/wiki/Data_validation)
5. Transform ([clean](https://en.wikipedia.org/wiki/Data_cleaning), apply [business rules](https://en.wikipedia.org/wiki/Business_rule), check for [data integrity](https://en.wikipedia.org/wiki/Data_integrity), create [aggregates](https://en.wikipedia.org/wiki/Aggregate_(data_warehouse)) or disaggregates)
6. Stage (load into [staging](https://en.wikipedia.org/wiki/Staging_(data)) tables, if used)
7. [Audit reports](https://en.wikipedia.org/wiki/Audit_report) (for example, on compliance with business rules. Also, in case of failure, helps to diagnose/repair)
8. Publish (to target tables)
9. [Archive](https://en.wikipedia.org/wiki/Archiving)

(VIKI)

Még esetleg itt: <http://www.iit.uni-miskolc.hu/iitweb/export/sites/default/department/labs/iit-szolgaltatasok/www-db/Tantargyak/OLAP_DM_MSc/ora_07.pdf> van anyag.

**Horizontális/vertikális partícionáló algoritmusok**

Hát ezt tudja tököm,

The partitioning can be done by either building separate smaller databases (each with its own [tables](https://en.wikipedia.org/wiki/Table_(database)), [indices](https://en.wikipedia.org/wiki/Index_(database)), and [transaction](https://en.wikipedia.org/wiki/Database_transaction) [logs](https://en.wikipedia.org/wiki/Database_log)), or by splitting selected elements, for example just one table.

**Horizontal partitioning** (also see [***shard***](https://en.wikipedia.org/wiki/Shard_(database_architecture))) involves putting different rows into different tables. Perhaps customers with [ZIP codes](https://en.wikipedia.org/wiki/ZIP_code) less than 50000 are stored in CustomersEast, while customers with ZIP codes greater than or equal to 50000 are stored in CustomersWest. The two partition tables are then CustomersEast and CustomersWest, while a [view](https://en.wikipedia.org/wiki/View_(database)) with a union might be created over both of them to provide a complete view of all customers.

**Vertical partitioning** involves creating tables with fewer columns and using additional tables to store the remaining columns.[[1]](https://en.wikipedia.org/wiki/Partition_(database)#cite_note-1) [Normalization](https://en.wikipedia.org/wiki/Database_normalization) also involves this splitting of columns across tables, but vertical partitioning goes beyond that and partitions columns even when already normalized. Different physical storage might be used to realize vertical partitioning as well; storing infrequently used or very wide[[*further explanation needed*](https://en.wikipedia.org/wiki/Wikipedia:Please_clarify)] columns on a different device, for example, is a method of vertical partitioning. Done explicitly or implicitly, this type of partitioning is called "row splitting" (the row is split by its columns). A common form of vertical partitioning is to split dynamic data (slow to find) from static data (fast to find) in a table where the dynamic data is not used as often as the static. Creating a view across the two newly created tables restores the original table with a performance penalty, however performance will increase when accessing the static data e.g. for statistical analysis.

**Tranzakció:**

Na, legalább 100x szerepel a jegyzetben ez a szó, de definíció az nincs hozzá, szóval vikiről:

**Tranzakciókezelő rendszerek:**

jah, nyugi, ez sincs kifejtve, viki:

In [database systems](https://en.wikipedia.org/wiki/Database_system), **atomicity** (or **atomicness**; from [Greek](https://en.wikipedia.org/wiki/Greek_language) *a-tomos*, *undividable*) is one of the [ACID](https://en.wikipedia.org/wiki/ACID) [transaction](https://en.wikipedia.org/wiki/Database_transaction) properties. In an **atomic transaction**, a series of database operations either *all* occur, or*nothing* occurs. The series of operations cannot be divided apart and executed partially from each other, which makes the series of operations "indivisible", hence the name. A guarantee of atomicity prevents updates to the database occurring only partially, which can cause greater problems than rejecting the whole series outright. In other words, atomicity means *indivisibility* and *irreducibility*.[[1]](https://en.wikipedia.org/wiki/Atomicity_(database_systems)#cite_note-1) As a consequence, the transaction cannot be observed to be in progress by another database client. At one moment in time, it has not yet happened, and at the next it has already occurred in whole (or nothing happened if the transaction was cancelled in progress).

A **transaction** symbolizes a unit of work performed within a [database management system](https://en.wikipedia.org/wiki/Database_management_system) (or similar system) against a database, and treated in a coherent and reliable way independent of other transactions. A transaction generally represents any change in database. Transactions in a database environment have two main purposes:

1. To provide reliable units of work that allow correct recovery from failures and keep a database consistent even in cases of system failure, when execution stops (completely or partially) and many operations upon a database remain uncompleted, with unclear status.
2. To provide isolation between programs accessing a database concurrently. If this isolation is not provided, the programs' outcomes are possibly erroneous.

A database transaction, by definition, must be [atomic](https://en.wikipedia.org/wiki/Atomicity_(database_systems)), [consistent](https://en.wikipedia.org/wiki/Consistency_(database_systems)), [isolated](https://en.wikipedia.org/wiki/Isolation_(database_systems)) and [durable](https://en.wikipedia.org/wiki/Durability_(database_systems)).[[1]](https://en.wikipedia.org/wiki/Database_transaction#cite_note-1) Database practitioners often refer to these properties of database transactions using the acronym [ACID](https://en.wikipedia.org/wiki/ACID).

**Runaway failover:**

Ping-pong failover

* Ha egy szolgáltatás nem indítható egyik csomóponton sem, de mindegyik látja, hogy a másikon nem megy, így folyamatosan próbálják egymás helyét átvenni sikertelenül)
* A hibás állapotból nem induljon a csomópont magától újra

Runaway failover

* Korai szolgáltatás átadás, a tartalék megpróbálja visszaadni a szolgáltatás az éppen induló eredeti csomópontnak, anélkül, hogy ott a hibát javították volna
* Az átadást kössük engedélyhez (emberi beavatkozással jelezzük, ha egy csomópont kész a klaszterbe való visszalépésre)

**Felhasználói felületi integráció:**

= GUI integráció

**tranzakciós rendszerek**

* Middleware tipus
* tömeges alkalmazásintegrációnál le van írva a tartalom közé, de hogy a dián mi akar ez lenni…

**Dead letter queue**

A nem továbbítható üzenetek tárolására szolgál

Mindíg lokális queue és csak egy van belőle

A Queue Manager kezeli és az MQDLH prefix-el egészíti ki, amely infomációt ad arról, hogy az üzenet miért került a DLQ-ba

Tervezési megfontolások:

* a DLQ-ban levő üzenetek perzisztenciája
* egy program amely feldolgozza és üríti a DLQ-t

a Queue tipusok egyébként:

**Alias Queue**

•Referencia egy lokális vagy távoli queue-ra

•Egy queue-nak több alias neve is lehet

•Használata:

–különböző default attributumok

–különböző security

–különböző használat (put / get)

–elválasztja a logikai queue nevet a fizikai névtől

**Model és Dynamic Queue**

•A modell queue megnyitásakor létrejön egy dinamikus queue

•A dinamikus queue kétféle lehet:

–temporary - bezáráskor meg is szűnik

–permanent - explicit módon törölni kell

•A neve a megnyitáskor megadott minta alapján a Queue Manager által generálódik

**Transmission Queue**

•A lokális queue egyik specializált használata

•Minimum egy kell minden távoli Queue Manager felé

•Default neve a távoli Queue Manager neve

•Az alkalmazások direkt módon nem használhatják

•A üzenetek egy speciális kiegészítő header-rel kerülnek bele (MQXQH)

•A Message Channel Agent használja az üzenetek továbbítására

**Initiatin Queue**

•A lokális queue egy speciális használata

•A Triggerelés lehetővé teszi az alkalmazások automatikus indítását

–A vezérlő üzenetek a Queue Manager által generálódnak amikor a trigger-esemény bekövetkezik:

–First

–Depth

–Every

–Priority

•Speciáli üzenet header QMTM

**Repy-toQueue**

•Reply illetve Report tipusú üzenetek kezelésére szolgál

•A neve az MQMD ReplyToQ mezőjéből származik

•A név két részből áll (A Queue Manager név-feloldó eljárása is kitöltheti)

–a lokális queue neve

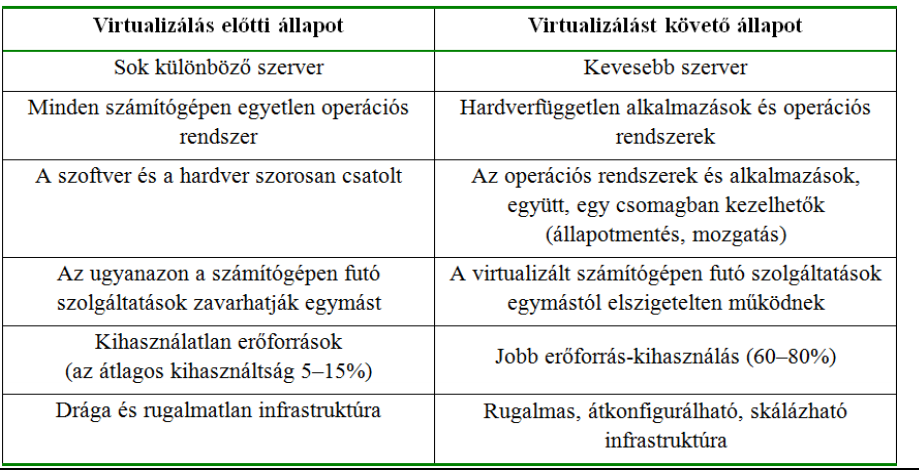
–a Queue Manager neve

•Elterjedt tervezési konvenció:

–a reply Correlation ID-je megegyezik a request Message ID-jével

**A hardweres virtualizálás előnyei**

A rendszerek közötti különbségek elsimítása: a virtualizálás elfedi a modulok belső tulajdonságait, illetve a bennük található építőelemek sokszínűségét.



A processzor a saját szintjén támogatja az erőforrások particionálását

**Adattisztítás**

A zaj, a redundancia és az inkonzisztencia eltávolítása, hiányzó adatok kezelése

Eszközei: zajszűrés: klaszterezés, osztályozás, regresszióa, szótárak, ismétlődések szűrése, hiánykezelés: törlés, modellezés, klaszterezés, átlagolás

A hiányzó értékek kezelése:

A hiányos rekordok eldobása vagy korlátozott felhasználása. Hiány pótlása: helyettesítés alapértelmezett, átlagos, tipikus vagy legvalószínűbb értékkel. Helyettesítés más hasonló rekordok adatai alapján vagy csoportokba sorolás után a csoport reprezentatív értékével 2

Zaj-, redundancia- és inkonzisztenciaszűrés:

Duplikátumok helyettesítése egy elemmel, egymásnak és a kényszereknek ellentmondó rekordok eltávolítása. Szélsőséges értékek kiszűrése. Adatsimítás kosarazással, klaszterezéssel, regresszióval. Szöveges attribútumok zajszűrése klaszterezéssel és szótárakkal A kényszereket be nem tartó, üzleti logikának ellentmondó adatok eltávolítása. Adatsimítás: kosarazás, klaszterezés, modellillesztés, szövegre szótáralapú is ; az adathatlmaz lokális tulajdonságai alapján becsüljük a zajt, majd új, reprezentatívabb értékkel helyettesítjük a zajjal terhelt adatokat. Kosarazás: a zajosnak vélt attribútum szerint rendezzük az adatokat, az eredményt azonos méretű csoportokra bontjuk, ezután az egy kosárba került elemek adott attribútumát egy, a kosárra jellemző értékkel helyettesítjük. Klaszterezés: kosarazás, csak nem rögzítettek a kosarak mérete. Modellillesztés: az adatsorunkat egy vagy több másik változó függvényében elemezzük, és az attribútumértékek helyettísére a szabály alapján leképzett modellt használjuk. Szövegre működik az előbbiek mindegyike, plusz Szótáralapú módszer: explicit módon felsoroljuk a leggyakoribb helyes és hibás formákat. Az attribútumok transzformálása Szótárak, algoritmikus transzformációk, attribútum­összevonások és ­szétválasztások, normalizálás, diszkretizálás

**Diszkrtizálás**

Kvantálás azonos méretű és azonos elemszámot tartalmazó értékintervallumokkal. Hisztogramok Kvantálás: kategóriákat alakítunk ki, ezekhez alsó és felső határokat rendelünk az elemek besorolásához. (cm helyett alacsony, magas..). Hisztogram: az adattartományok, illetve a hozzá tartozó elemek relatív gyakoriságának szemléltetése.

Ezt nem tudja felfogni az én halandó szövegértő agyam. azt jelenti, hogy az értékekből folytonos mértékekből szakaszokat alakítunk ki, és azokat adjuk meg (pl. kerekítünk egész számra), esetleg túl sok diszkrét érték esetén kisebb csoportokat hozunk létre (maradéktalan osztás 10-zel pl.)

**horizontális és vertikális particionált nézetek,**

és mikor használjuk őket? (elosztott adatbázisok, adatintegráció, szinkron eset -> elosztott nézetek, ZH utáni első óra)

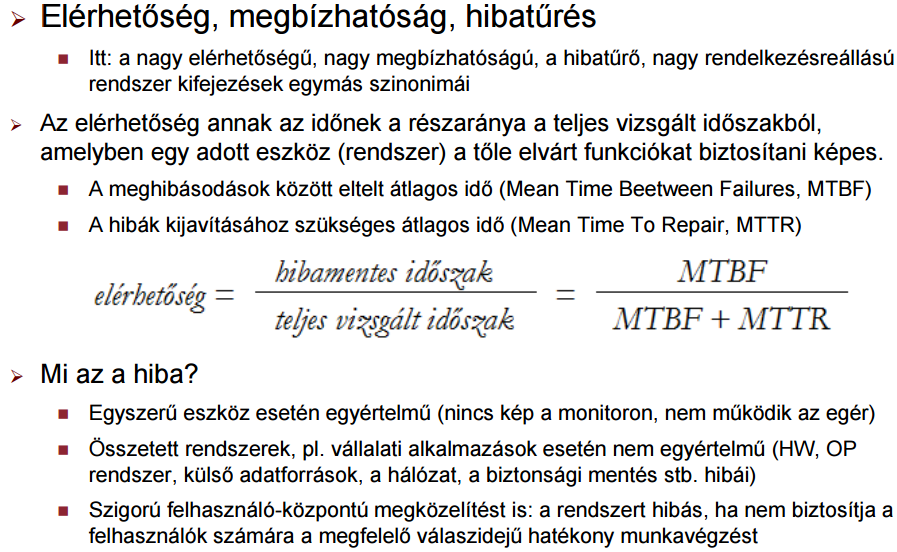
**mediációs réteg**

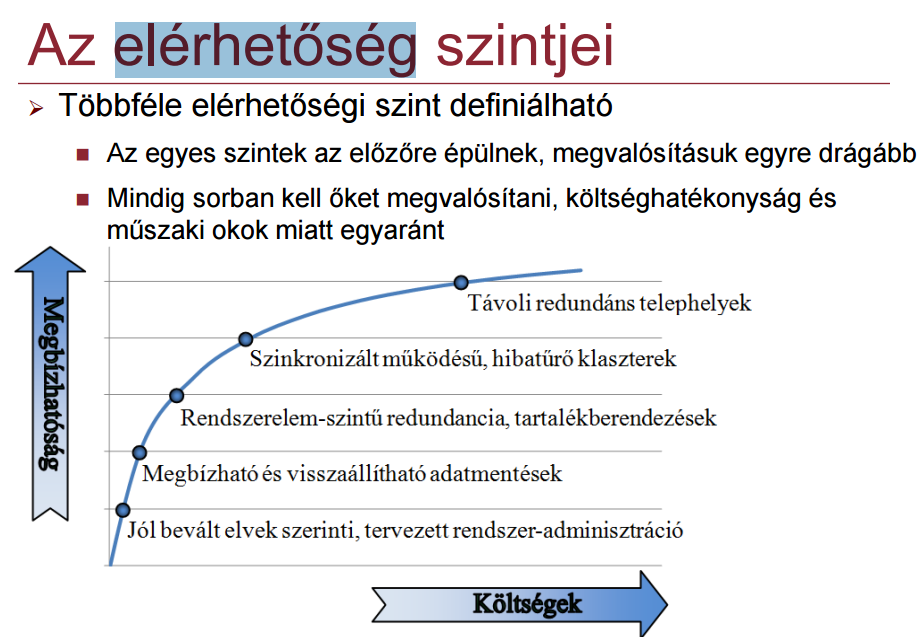
Mi az öt rétegre történő kiterjesztés lényege?

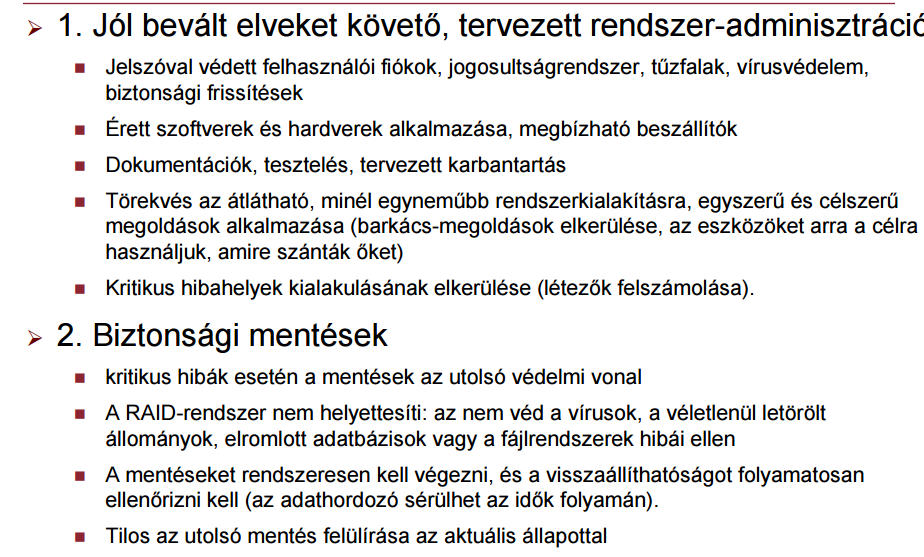
Két új réteg kerül a rendszerbe: egy üzleti logika és a prezentáció közé

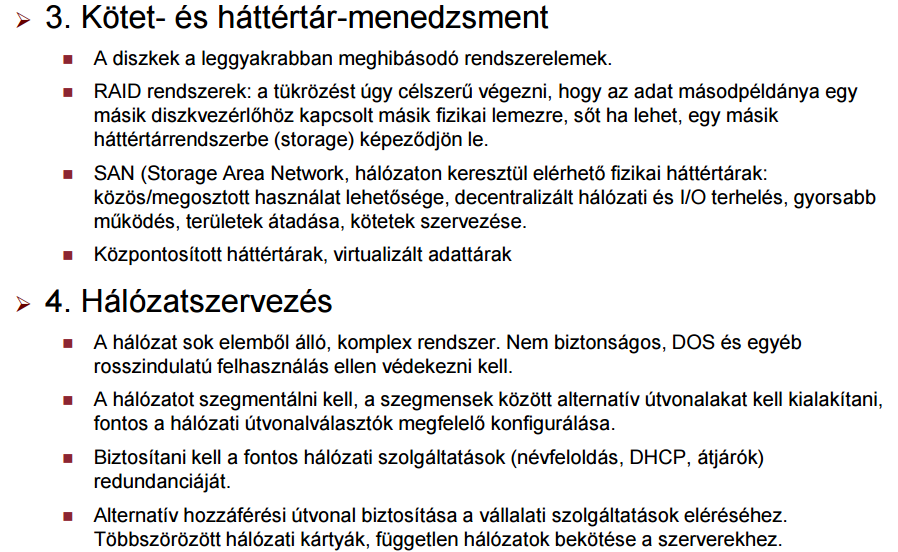
(mediator), a másik az adatelérési réteg és üzleti logika közé (data mapper).

**elérhetőség szintjei**

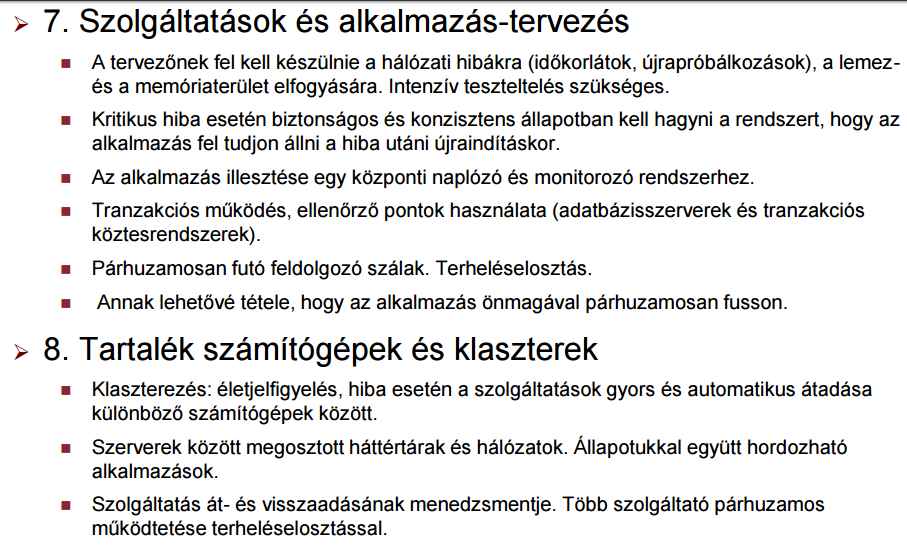


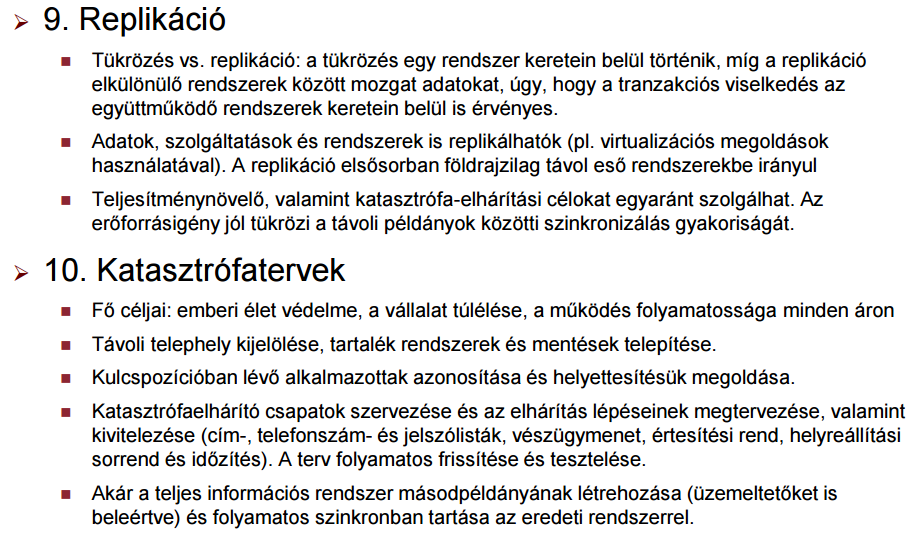












**ROLAP, MOLAP, HOLAP**

A multidimenzionális adatmodell prezentálása és tárolása Az adatkocka a multidimenzionális adatok tárolási modelljétől független adatelemzési egysége, ROLAP­, MOLAP­, HOLAP­tárolás

OLAP kocka: a kocka oldalai mentén tüntetjuük fel a dimenziókat, míg a dimenziók által meghatározott metszéspontokban található cellák a tényadatokat tartalmazzák.

● ROLAP(Relational OLAP): a multidimenzionális adatmodellt relációs adatbázisra képezik le csillag és hópehelysémák használatával → relációs jó, mert a fejlesztők már ismerik, sok eszköz van rá, skálázható…

● MOLAP(Multidimensional OLAP): az adatok tárolására multidimenzionális adatmodell. Ez a modell az egyes elemeket többdimenziós vektorban tárolja, ahol azok közvetlen indexeléssel hozzáférhetők.

● HOLAP(Hybrid OLAP): ötvözik a relációs megközelítés skálázhatóságát a multidimenzionális tárolás gyorsabb adatelérési és számítási képességeivel.

ROLAP vs MOLAP tárhelykezelés: ritka adatok kezelésénlé MOLAP ugyanannyi multidimenziós adat tárolásához akár 2­10­szer kevesebb helyet igényel, mint a ROLAP.

**Rendszer-architektúra**

**Operációs rendszer szintű virtualizáció**

**osztályozás (Adatbányászat)**

Modellezési feladat, besorolási kategória meghatározása tanítópéldák alapján

Minőségi paraméterek Módszer jósága(pontossága): osztályozási hiba becslésével mérhető. Túltanulás: ha az osztályozó algoritmus a tanítóhalmaz speciális összefüggéseit és mintáit tanulja meg ahelyett, hogy ezek általánosításával építené fel a modellt. Osztályozás megalapozottsága: az egyes osztályokhoz tartozó tanítóminták száma jellemzi Teljesítményparaméterek: betanulási idő, futási idő, skálázhatóság. Robosztusság. Eredmények értelmezhetősége.

Osztályozási módszerek

● döntési fák: bonyolult osztályozási döntéseket egyszerűbb döntések sorozataként kezelik

● neurális hálózatok: lineáris szeparációra képes univerzális függvényapproximátorok, amelyek statisztikai modellezőeszközként is felhasználhatók.

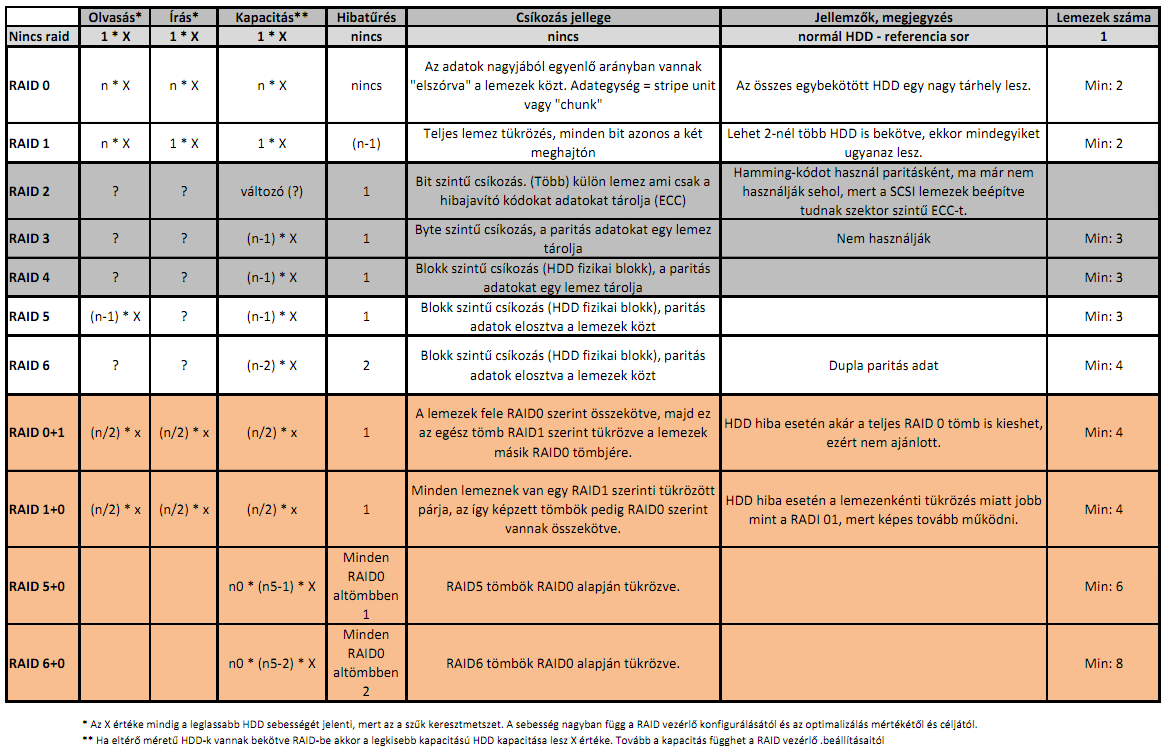
● bayes­i osztályozóhálók: Két alapelve ○ Bayes­tétel: ismeretlen feltételes valószínűségek meghatározása imertekkel ○ maximum likelihood elv: alapján a legnagyobb valószínűségű osztály mellett döntünk

● legközelebbi szomszédok módszere: hasonló adatok egymáshoz közel helyezkednek el az attribútumtérben.

● lineáris regresszió: valós értékkészletű változókra alkalmazható statisztikai módszer

**skálázhatóság**

**RAID összehasonlítás:**

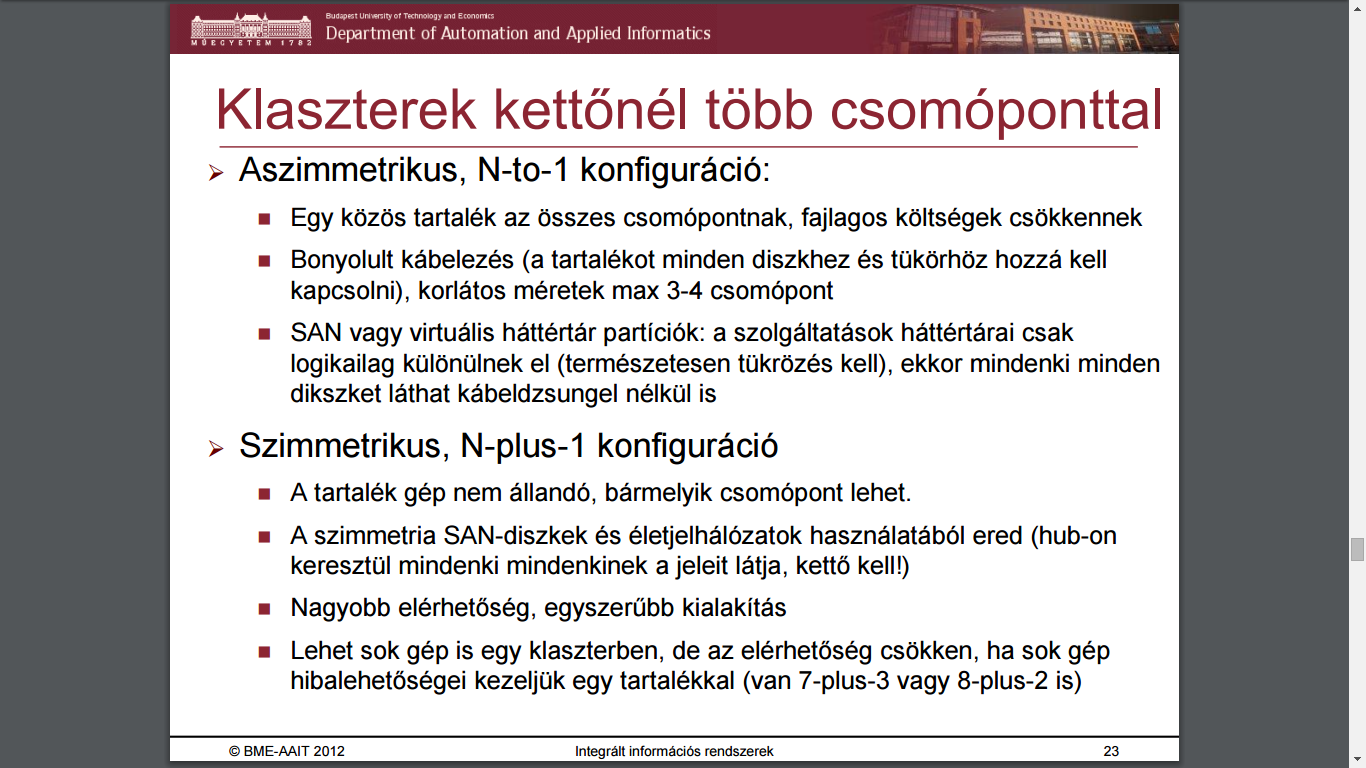


**Legutóbbi vizsga kérdései:**

**hasonlítsa össze az n-to-1 és n-plus-n konfigurációkat clusterekben!**

The most common size for an HA cluster is a two-node cluster, since that is the minimum required to provide redundancy, but many clusters consist of many more, sometimes dozens of nodes. Such configurations can sometimes be categorized into one of the following models:

* Active/active — Traffic intended for the failed node is either passed onto an existing node or load balanced across the remaining nodes. This is usually only possible when the nodes use a homogeneous software configuration.
* Active/passive — Provides a fully redundant instance of each node, which is only brought online when its associated primary node fails.[1] This configuration typically requires the most extra hardware.
* N+1 — Provides a single extra node that is brought online to take over the role of the node that has failed. In the case of heterogeneous software configuration on each primary node, the extra node must be universally capable of assuming any of the roles of the primary nodes it is responsible for. This normally refers to clusters that have multiple services running simultaneously; in the single service case, this degenerates to active/passive.
* N+M — In cases where a single cluster is managing many services, having only one dedicated failover node might not offer sufficient redundancy. In such cases, more than one (M) standby servers are included and available. The number of standby servers is a tradeoff between cost and reliability requirements.
* N-to-1 — Allows the failover standby node to become the active one temporarily, until the original node can be restored or brought back online, at which point the services or instances must be failed-back to it in order to restore high availability.
* N-to-N — A combination of active/active and N+M clusters, N to N clusters redistribute the services, instances or connections from the failed node among the remaining active nodes, thus eliminating (as with active/active) the need for a 'standby' node, but introducing a need for extra capacity on all active nodes.



**FELADATOK A WEBOLDALRA FELTÖLTÖTT PDF-EKBŐL**

Számítógép által létrehozott helyettesítő szöveg:
Egy levelező szerver 1 millió regisztrált felhasználóval rendelkezik. A 
legforgalmasabb órában a felhasználók 15%-a küld levelet, átlagosan 
2,4 darabot. A levelek továbbítása párhuzamosan történik, de egy-egy 
levél továbbításához a szervernek 230 ms idóre van szüksége. Milyen 
méretúre kell választani a leveleket ideiglenesen tároló (spool) fájlt, ha 
azt a legforgalmasabb óra terhelésének duplájára szeretnénk méretezni, 
és egy levél átlagosan 21 kB hosszúságú? 
A kritikus óra. 1 000 000*0, 15*2,4 360 000 darab, azaz 
az aktuális átbocsátóképesség: 360 000 db/óra 100 db/s. 
• Little törvénye alapján: 100 db/s*O,23 s 23 db levél van átlagosan 
a rendszerben 
Az egyes levelek 21 kB hosszúak, a spool fájl átlagos mérete. 
23 db * 21 kB 483 kB adódik, de az elóírt a méretezés ennek 
duplája 966 kB. 

Számítógép által létrehozott helyettesítő szöveg:
Egy webszerver csúcsterhelés idején egyszerre 50 darab kérés 
párhuzamos kiszolgálásával foglalkozik. A szerver óránként maximálisan 
72 000 kérés kiszolgálására képes. 
Mekkora a szerver válaszideje? 
Little törvénye értelmében, ha a rendszerben 50 kérés tartózkodik, és a szerver 
átbocsátóképessége 72 000 kérés/óra (20 kérés/s), akkor a rendszer válaszideie 
50 db/20 db/s 2 5 s. 
Hány regisztrált felhasználót képes kiszolgálni a rendszer, ha egyszerre 
maximum 10%-uk lépett be, és egy felhasználó percenként átlagosan 3 
lekérdezést végez? 
A belépett felhasználók óránként x *O, 1*3*60 x* 18 db tranzakciót generálnak, ahol x 
a regisztrált felhasználók száma. 
Ha a kihasználják a maximális átbocsátóképességet, akkor 
x 72 db/óra/ 18 db/óra 4 000 lehet. 

Számítógép által létrehozott helyettesítő szöveg:
Egy tranzakciófeldolgozó rendszer 3 darab, külön számítógépen futtatott, 
egymást adott sorrendben követő feldolgozási fázisból áll. A 
kezdeményező a kéréseket tetszőleges súrúséggel indíthatja, de a 
feldolgozók a kéréseket sorban egymás után hajtják végre. 
Az adat útja a K kezdeményezótól: K—+l 
minden adatátvitel hálózati kapcsolaton keresztül történik, és 50 ms-ig tart 
Az egyes fázisok feldolgozási ideje rendre 70, 160, 110 ms. 
Hogyan változik a rendszer minimális válaszideje és az 
átbocsátóképessége, ha a leglassabb második fázis feldolgozóegységét 
megkettózzük, és a páros elé egy 10 ms késleltetésú terhelés 
elosztófázist iktatunk be? 

Számítógép által létrehozott helyettesítő szöveg:
eredeti rendszer 
kliens 
(70 ms) 
( 160 ms) 
(110 ms) 
módosított rendszer 
kliens 
(70 ms) 
160 ms) 
160 ms) 
elosztó 
(IO ms) 
(110 ms) 
Minimális válaszidó: 50 ms + 10 ms 60 ms-mal nó a beiktatott 
terheléselosztó kommunikációs és feldolgozási késletetése miatt. 
Átbocsátó képesség. 
• Eredetileg a szúk keresztmetszetet a 2. feldolgozó, ezért az átbocsátóképessége 
1/160ms- 1/0,16 s- 626 TPS 
• A módosított rendszer szúk keresztmetszete a 3. feldolgozó lesz, ezért az 
átbocsátóképesség 1/110 ms 1/0, 11 s 9,09 TPS 
(a második fokozat: 160 ms alatt két feladat 2/0, 16 12,52 TPS) 
A módosítás az átbocsátóképességet 1.452-szörösére. vagyis 45.2%-kal növeli. 

**Melyik három alapvető réteg vesz részt az információs rendszerek felépítésében?**

* Adatelérési, üzleti logikai és prezentációs réteg. (Model, Control, View)

**Mi az öt réteglre tejlesztés lényege?**

* Két új réteg kerül be: egy az üzleti logikai és prenzetááció közé - mediator - a másik az adatelérési réteg és üzleti logika közé - data mapper.
* Feladata a két új rétegnek az elosztottság fedezése

Számítógép által létrehozott helyettesítő szöveg:
Egy folyamatosan múködó online pénzügyi rendszert a felhasználók 
hetente átlagosan 30 percig nem tudnak használni a különféle leállítást 
igénylő karbantartási munkák és a felmerüló hibák miatt. Mekkora a 
rendszer elérhetősége? 
Egy hét 168 óra, ez alatt a rendszer átlagosan 167,5 órát működik, és 0,5 
óráig elérhetetlen. A rendszer elérhetósége 167,5 / (167,5 + 0,5): 99 70% 
a MTBF / (MTBF + MTTR) képletből számolva. 
Egy vállalat az adattárházában 500 darab 100 GB kapacitású 
merevlemezes egységet működtet RAID-rendszerbe kötve. Az 
egységek adatlapja alapján a várható hibamentes élettartam (MTBF) 
1 6,2 év. Egyenletes hibaeloszlást feltételezve hetente átlagosan hány 
egység cseréjére kell számítania a vállalatnak? 
Az egyenletes hibaeloszlás várható értéke akkor 16,2 év, ha ez alatt az 
egységeknek a fele hibásodik meg. A 250 hiba ebben az idótattományban 
egyenletesen oszlik el, ezért egy hétre átlagosan 250/16,2/52 0 297 
darab hiba esik azaz körülbelül havonta egy egységet kell kicserélni. 

Számítógép által létrehozott helyettesítő szöveg:
Egy e-mail küldö rendszer 99,7%-os elérhetőséggel működik, és minden 
egyes óra kiesése 20 000 Ft-ba kerül. A rendszer elérhetősége a 
becslések szerint 1 millió Ft-os költséggel 99,9%-ra javítható. Megtérül-e 
egy éven belül a változtatás? 
• A változtatás egy éven belül (0,999-0,997)*365*24 17,52 óra állásidót lenne 
képes megtakarítani, amelynek költsége 350 400 Ft, ez jóval kisebb, mint a 
védelem kialakításához szükséges 1 millió Ft-os összeg, így nem térül meg. 
Egy vállalat raktárnyilvántartó szoftvere 98%-os elérhetőséggel működik. 
A rendszer kiesése a becslések szerint óránként 50 000 Ft kárt okoz. A 
rendszer készítője felajánlja, hogy a rendszerkiesések idejét különféle 
változtatásokkal a felére csökkenti. Mennyit költhet a vállalat erre a 
módosításra, úgy hogy a befektetés fél év alatt megtérüljön? 
• A kiesési idó felezése az elérhetőséget 98%-ról 99%-ra változtatja. Fél év alatt 
ez a módosítás a vállalatnak (0,99-0,98) *365/2*24 43,8 óra állásidót takarít 
meg, ezért a megadott feltételek mellett a vállalat maximum 43,8 * 50 000 2 19 
millió Ft-ot költhet erre a beruházásra. 

