Nagy méretű adatbázisok backup problémája

**Sapientia Erdélyi Magyar Tudományegyetem**

**Műszaki és Humántudományok Kar**

2013-2014

**Készítette:**

**Magyarosi Noémi és Balas Erika**

**Infomatika Szak**

# Tartalomjegyzék

Contents

[Tartalomjegyzék 1](#_Toc366857896)

[Bevezető 2](#_Toc366857897)

[Adatbázis fogalma 2](#_Toc366857898)

[Történelmi áttekintés 3](#_Toc366857899)

[Kivonat 4](#_Toc366857900)

[Mi az FD (Funkcionális függség- Functional dependencies)? 4](#_Toc366857901)

[Mi az CFD (Kondicionális funkcionális függőség- Conditional functional dependencies)? 5](#_Toc366857902)

[Mi az AR? 6](#_Toc366857903)

[Mi a backup? 6](#_Toc366857904)

[Szoftverfejlesztés közösen a Github-on 7](#_Toc366857905)

[Git történelem 7](#_Toc366857906)

# Bevezető

## Adatbázis fogalma

Adatbázison köznapi értelemben valamely rendezett, valamilyen szisztéma szerint tárolt adatokat értünk, melyek nem feltétlenül számítógépen kerülnek tárolásra. Képzeljük el, hogy egy céghez naponta átlagban 20 levél érkezik. A cég irattárosa kellő adattárolási tapasztalat híján a leveleket az irattár ajtajára vágott lyukon keresztül bedobja. Elképzelhető, hogy pár év eltelte után milyen reménytelen vállalkozás egy levelet megtalálni az irattárban. Ez az adathalmaz nem tekinthető adatbázisnak, ahhoz hogy adatbázis legyen nem elegendő a nagyszámú adat. Az adathalmaz csak akkor válik adatbázissá, ha az valamilyen rend szerint épül fel, mely lehetővé teszi az adatok értelmes kezelését. Természetesen ugyanazon adathalmazból többféle rendszerezés alapján alakíthatunk ki adatbázist. Például egy könyvtárban a könyveket rendezhetnénk a könyvek mérete vagy akár a szerző vagy szerzők testsúlya alapján. Ez már egy rendszert ad az adatok tárolásához. Íly módon minden könyv helye meghatározott. De bizonyára nehéz helyzetben lennénk, ha szerző és cím alapján próbálnánk meg előkeresni egy könyvet. Az adatok tárolásába bevitt rendszernek alkalmasnak kell lennie a leggyakrabban előforduló igények hatékony kielégítésére. Az adatbázisok mellé egy **adatbázis-kezelő rendszer (DBMS)** is járul, mely az adatbázis vagy adatbázisok üzemeltetését biztosítja. Hagyományos adatbázis esetén ez a kezelő személyzet intelligenciájának része, elektronikus adatbázisok esetén pedig valamilyen szoftver..

Az **adatbázis** azonos minőségű (jellemzőjű), többnyire strukturált [adatok](http://hu.wikipedia.org/wiki/Adat" \o "Adat) összessége, amelyet egy tárolására, lekérdezésére és szerkesztésére alkalmas szoftvereszköz kezel. Az adatbázisok célja adatok megbízható, hosszú távon tartós (idegen szóval: perzisztens) tárolása, és viszonylag gyors visszakereshetőségének biztosítása.

*Fenntartási költségek*

*Ma az adatbázisok üzemeltetése nagy összegekbe kerül. Az*[*IDC*](http://hu.wikipedia.org/w/index.php?title=IDC&action=edit&redlink=1)*piackutató cég adatai szerint csak az [Egyesült Államokban](http://hu.wikipedia.org/wiki/Amerikai_Egyes%C3%BClt_%C3%81llamok" \o "Amerikai Egyesült Államok) lévő 6500 nagy számítóközpont energiaigénye megegyezik*[*Utah*](http://hu.wikipedia.org/wiki/Utah)*államéval. Ehhez 30 nagy erőművet kell üzemben tartani, s az elfogyasztott energia több, mint az USA-ban működtetett több mint 300 millió tévékészüléké.*

## 

## Történelmi áttekintés

Azóta rendelkezünk adatbázisokkal, mióta írásban vagyunk képesek rögzíteni adatokat. Ez az ókorban történhetett akár kőtáblákra vagy papirusz tekercsekre. Az adatbázisok fejlettebb formái később a kartoték rendszerek lettek, melyek a számítógépek megjelenéséig az alapvető adatbázis rendszerek voltak. A számítástechnika hőskorában az 50-es 60-as években az adatok tárolása még lyukszalagon, lyukkártyán történt, az adatok közvetlenül nem voltak elérhetők a számítógép számára. A mágneses háttértárolók elterjedésével az adatok tárolása egyszerűbbé, elérésük hatékonyabbá vált. Ezekben az időkben még nem léteztek univerzális módszerek illetve rendszerek, melyek segítségével az adatbázisokkal kapcsolatos problémák nagy része általánosan megoldható lett volna.

A számítógépek fejlődésével együtt fejlődtek a programozói lehetőségek is. Az első számítógépeken csak a gépi kód (a bináris formában kiadott utasítások a mikroprocesszornak) állt rendelkezésre. Ezt első generációs programnyelvnek nevezzük. Ezt követték a második generációs (assembler) nyelvek, melyekben a gépi kód helyett úgynevezett mnemonikok és szimbólumok alkalmazhatók. Az első illetve második generációs programnyelvekben még nem észültek komoly adatbáziskezelő alkalmazások. Ezekre egyrészt a magas szintű nyelvek (3. generációs program nyelvek) COBOL, FORTRAN stb., másrészről a lemezes operációs rendszerek kialakulásáig kellett várni. Ekkor már komoly adatbázis alkalmazások születtek, melyek egyedi problémák megoldására voltak alkalmasak.

Az adatbázisok méretének és számának gyors növekedése következtében az egyedi alkalmazások létrehozása fárasztó és időrabló feladattá vált, ezért a programfejlesztők törekedtek az adatbáziskezelés általános formában történő megfogalmazására. Ennek eredményeként jöttek létre az adatbázis kezelő rendszerek. Az adatbázis kezelő rendszerek számos eszközt nyújtanak az interaktiv adatbevitel, menük létrehozása terén, melyek kialakítása a harmadik generációs nyelvekben sok sok oldal kód leírásával lenne csak lehetséges. A szabványos eszközök bevezetésével nem csak a programozói munka csökkent le, hanem az egységes felhasználói felület kialakítására késztetik a programozókat.

# Kivonat

A backup készítése egy időigényes művelet, mi ezt szeretnénk felgyorsítani olyan módon, hogy a megtanult és validált FD, CFD és AR összefüggések segítségével csak az adatok egy részét olvassuk ki (a meghatározó oszlopokat), abban az esetben, ha teljesül az összefüggés. Ezzel a szüréssel lehetséges például egy 3GB méretű adatbázis helyett 1GB-nyi adattal foglalkozni, ha kiszűrjük a CFD-ket.

## Mi az FD (Funkcionális függség- Functional dependencies)?

FD: olyan összefüggés, melyben a meghatározó oszlopok meghatározzák a meghatározott oszlopot minden esetben. Ha van egy T tábla és van egy Determinant oszlophalmaz, illetve egy Dependent oszlophalmaz, akkor Determinant -> Dependent.

Determinant = Determinant1, ..., Determinantm

Dependent = Dependent1, ..., Dependentm

Egy Determinant halmaz és egy Dependent halmaz FD-t alkot akkor és csakis akkor, ha:

((R1.Determinant1 = R2.Determinant1) ^ ... ^ (R1.Determinantm = R2.Determinantm)) => ((R1.Dependent1 = R2.Dependent1) ^ ... ^ (R1.Dependentn = R2.Dependentn)), ahol R1 és R2 bejegyzések a táblából.

**Példa:**

Egy vagy több adat konkrét értékéből más adatok egyértelműen következnek:

SZEMÉLYI\_SZÁM -> NÉV

AUTÓ\_RENDSZÁM -> TIPUS, TULAJDONOS

HELY, IDŐPONT -> HŐMÉRSÉKLET

Az FD ellenőrzésének algoritmusa:

boolean isFD(String table, AbstractList<String> determinantColumns, AbstractList<String> dependentColumns)

’’kibovul’’.................................................................................................................................

## Mi az CFD (Kondicionális funkcionális függőség- Conditional functional dependencies)?

A CFD egy olyan összefüggés, mely hasonlít egy FD-re, az egyedüli különbség az, hogy a funkcionális függőség csak egy feltétel esetén vizsgálandó.

**Példa:**

Nagy-Britanniában a cím meghatározza a postakódot:

(Country = 'GB') => ((Address) -> (PostCode))

Vagyis ha a táblában nem lennének olyan sorok, melyekre az alapfeltétel (esetünkben az, hogy az ország Nagy-Britannia) nem teljesül, akkor egy FD lenne. Innen az elnevezés, Conditional Functional Dependency, mely arra utal, hogy egy feltétel teljesülése esetén FD.

Az CFD ellenőrzésének algoritmusa:

boolean isCFD(String table, AbstractList<String> determinantColumns, AbstractList<String> dependentColumns, String condition)

’’kibovul’’.................................................................................................................................

## Mi az AR?

Az AR egy csoportosítási szabály, mely a következő képpen néz ki:

Condition => ((Dependent1 = a1) ^ ... ^ (Dependentn = an)), ahol (a1, ..., an) konstans értékek.

**Példa:**

f=>((A1=a1) és ... (An=an)) teljesülése esetén egy AR-ünk van.

Megj.: A=A1,....,An oszlopok

a=a1,..,an konstans értékek

Feltétel: megbukott=igaz

A1=végez az idén és A2=mindenből átment

a1=hamis és a2=hamis

Az AR ellenőrzésének algoritmusa:

boolean isAr(String table, String condition, AbstractList<String> dependentColumns)

’’kibovul’’.................................................................................................................................

## Mi a backup?

Fontos adatoknak, programoknak [biztonság](http://www.mimi.hu/informatika/biztonsag.html)i okokból készített másolata.  
A többnyire külön [tároló](http://www.mimi.hu/informatika/tarolo.html)ban őrzött adatok az eredeti meghibásodása, vagy elvesztése esetén használhatók.

Backuprol adatok pelda…………………………………………………………………………………………………………..

# Szoftverfejlesztés közösen a Github-on

A Git egy teljesen nyílt forráskódú verziókövető rendszer,amelyről az alábbiakban lesz bővebben szó. Mivel mi csapatban dolgoztunk elkerülhetetlen volt, hogy valamilyen verziókövető rendszert használjunk.

Github (https://github.com) segítségével oldottuk meg ezt, hogy még könnyebb és hatékonyabb tudjon müködni a csapatban való fejlesztés.

## Git történelem

2005. áprilisában a [Larry McVoy](http://wiki.hup.hu/index.php/Larry_McVoy) cége a [BitMover](http://wiki.hup.hu/index.php/BitMover" \o "BitMover) bejelentette, hogy visszavonja a szabadon (in beer) használható [Bitkeeper](http://wiki.hup.hu/index.php/Bitkeeper" \o "Bitkeeper)-t. A Bitkeeper helyett a nem sokkal korábban bejelentett nyílt forrású Bitkeeper kliens használható a jövőben. Sajnos a nyílt Bitkeeper kliens funkciójában messze elmarad a korábbi Bitkeeper-től. A sajtóbejelentésben az áll, hogy azoknak a fejlesztőknek akiknek szükségük van a kibővített funkcionalitásra, meg kell vásárolniuk a kereskedelmi verziót.

A bejelentés hatására [Linus](http://wiki.hup.hu/index.php/Linus) úgy döntött, hogy a jövőben nem fogja a [BK](http://wiki.hup.hu/index.php/BK)-t a Linux kernel fejlesztéséhez használni és alternatívát keres. Kérdéses volt, hogy melyik verziókezelő rendszerre esik majd Linus választása. Az [LKML](http://wiki.hup.hu/index.php/LKML)-en gyakran lehet hallani a [monotone](http://wiki.hup.hu/index.php/Monotone)-t emlegetni. Linus is tesztelte, de egy baja volt vele: lassúnak találta. Ezért Linus egy kicsit hackelt és a munka eredményeként egy új eszköz bukkant fel. Linus pár nap alatt összeütött egy patch backup management system-et, amelyet a jövőben a [Linux](http://wiki.hup.hu/index.php/Linux) [kernel](http://wiki.hup.hu/index.php/Kernel) fejlesztéséhez fog majd felhasználni. Az eszköz a **git** nevet kapta.

2005. júliusának végén Linus [átadta](http://www.hup.hu/modules.php?name=News&file=article&sid=9327" \o "http://www.hup.hu/modules.php?name=News&file=article&sid=9327) a git karbantartását Junio C Hamano-nak. Petr Baudi készített egy git honlapot, ahonnan letölthető az eszköz, de mellette tutorial-t, man oldalt, és minden egyéb segítséget megtalál az érdeklődő ahhoz, hogy felállítsa a saját git-jét.

A (2.6) Linux fejlesztése ma már teljes egészében a git segítségével zajlik.