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](#_Toc367457451)

[Kivonat 2](#_Toc367457452)

[Bevezető 2](#_Toc367457453)

[A backup készítésének fontossága 2](#_Toc367457454)

[Adatbázis fogalma 3](#_Toc367457455)

[Történelmi áttekintés 4](#_Toc367457456)

[Kifejezések: 5](#_Toc367457457)

[Kulcs 5](#_Toc367457458)

[Mi az FD (Funkcionális függség- Functional dependencies)? 5](#_Toc367457459)

[Mi a CFD (Kondicionális funkcionális függőség- Conditional functional dependencies)? 6](#_Toc367457460)

[Mi az AR? 7](#_Toc367457461)

[Szoftverfejlesztés közösen a Github-on 8](#_Toc367457462)

# Kivonat

A dolgozat egy backup rendszerről szól, mely örökérvényűnek elfogadott FD, CFD és AR összefüggések segítségével a backup készítés során az említett összefüggések érvényessége esetén csak a meghatározó oszlopok értékeit olvassa ki, a meghatározott oszlopok értékeit nem, mivel azok a meglévő, örök érvényűnek elfogadott összefüggésekből kiolvashatóak. Módszerünket olyan FD, CFD és AR összefüggésekre alkalmazzuk, melyekre létezik olyan garancia, hogy minden bejegyzés, melyekre ezek az összefüggések érvényesek most is és a jövőben is teljesítik az összefüggés által leírt logikai feltételt. Alkalmazásunkban megadtuk a lehetőséget arra, hogy a backup-ot az örökérvényűnek elfogadott összefüggések segítségével, vagy éppen anélkül a segítség nélkül futtassa a felhasználó annak érdekében, hogy a két módszert össze tudjuk hasonlítani sebesség, időigény és terhelési szint szerint is.

# Bevezető

## A backup készítésének fontossága

A biztonsági mentés fő célja az, hogy az állományainkat, rendszerünket, adatainkat sérülés, tönkremenetel, vírusfertőzés esetén vissza tudjuk állítani, ezzel a rendszer üzembiztonságát tudjuk növelni. Az adatmentés sok esetben kimerül a fájlok másolásában. Komolyabb mentési megoldás esetében is kérdés a mentés célja és a visszaállítási tesztek elvégzése. Az elkészült mentések esetén sokszor nincs kellő gondossággal kezelve az adathordozó és a rajta elhelyezkedő mentés. Ez óriási információ biztonsági kockázatot jelen. Manapság, a rosszindulatú számítógépes kártevőktől nyüzsgő mindennapokban nem mondhat le a biztonsági másolat készítéséről az, akinek fontosak a különféle, gépen tárolt állományai, fényképei, táblázatai, dokumentumai, videói. Ráadásul ma már a külső tárolóeszközök választéka is egyre színesebb, a nagy tárolókapacitású pendrive-októl kezdve a pehelysúlyú hordozható merevlemezeken át az írható optikai lemezekig, mindenki megtalálja a számára leginkább megfelelő háttértárat.  
Ma már több backup készítő szoftverrel is talákozhatunk, mint például a GFI Backup, Symantec NetBackup, Symantec Backup Exec, CommVault Simpana stb, a probléma az, hogy ez az adatmentési művelet napjainkban is nagyon lassú, leterheli szervereinket és mindez sok gondot okozhat. A mi módszerünkkel  backupkészítést olyan módon lehet felgyorsítani, hogy kevesebb adatot olvasunk ki és bizonyos érték meghatározó összefüggések adatmintájának segítségével kikövetkezteti a rendszer a nem kiolvasott adatot.

## 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) 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.

## 

## 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.

# Kifejezések:

## Kulcs

Azt az attribútumot vagy attribútum halmazt, melynek értékei egyértelműen azonosítják a relációt, a reláció kulcsának nevezzük. Ez a definíció séma (szerkezet) szintű, ami azt jelenti, hogy független a tábla aktuális tartalmától.

Kulcsok fajtái:

Egyszerű kulcs: a kulcs egyetlen attribútumból áll.

Összetett kulcs: a kulcsot kettő vagy több oszlop kombinációja alkotja, előfordulhat az is, hogy az összes oszlop szerepel a kulcsban.

## 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:

( Hely, Időpont ) -> Hőmérséklet

|  |  |
| --- | --- |
| Hely | Időpont |
| Marosvásárhely | 08:00 |
| Csíkszereda | 12:00 |
| Kolozsvár | 20:00 |

Ha tudjuk, hogy Marosvásárhely a helység és az időpont 08:00, akkor a két adatból biztosan következik a harmadik, vagyis a hőmérséklet.

Egy A és egy B oszlophalmaz esetén az isFD algoritmusal ellenőriztük le, hogy A meghatározza-e B-t és így FD-t alkotnak-e:

Pszeudokód:

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

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

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

CFD egy olyan értékmeghatározó összefüggés, melynek meghatározó oszlopainak az értékei meghatározzák a meghatározott oszlopok értékeit egy logikai függvény, más néven kondíció segíségével.

**Példa1:**

Nagy-Britanniában a cím meghatározza a postakódot:

(Nagy-Britannia= 'GB') => ((Cím) -> (Postakód))

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.

|  |  |  |
| --- | --- | --- |
| Ország | Cím | Postakód |
| Nagy-Britannia | 62 Hills Rd, Cambridge, CB2 1LA | AI-2640 |
| Nagy-Britannia | 4th Floor, Universal House, 88-94 | BIQQ 1ZZ |
| Románia | Nicolae Iorga utca, 1 szám | 540088 |

**Példa2:**

(Egészséges=’jó’) => ((Név, Kor) -> (Nem, Ihat-e alkoholt))

Ha teljesül az az alapfeltétel, hogy egészséges egyénről beszélünk, akkor fennáll a kondicionális funkcionális függőség.

|  |  |  |
| --- | --- | --- |
| Név | Egészségi állapot | Kor |
| Nagy László | jó | 18 |
| Kiss Gyöngyi | jó | 20 |
| Sándor Ferenc | gyenge | 20 |

Egy A és egy B oszlophalmaz esetén az isCFD algoritmusal ellenőriztük le, hogy A meghatározza-e B-t egy kondició segítségével és így CFD-t alkotnak-e:

Pszeudokód:

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

Ha teljesül az az alapfeltétel, hogy megbukott egyénről beszélünk, akkor fennáll csoportosan következik, hogy ugyan az a személy nem végez az idén, illetve nem ment át minden tantárgyból.

|  |  |  |
| --- | --- | --- |
| Név | Osztály | Megbukott |
| Péter Szilvia | VII. | Igaz |
| Nagy István | V. | Hamis |
| Simon Csaba | VII. | Igaz |

Egy A oszlophalmaz és egy kondíció esetén az isAR algoritmusal ellenőriztük le, hogy és AR-t alkotnak-e:

Pszeudokód:

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

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

# 

# Szoftverfejlesztés közösen a Github-on

A Git egy teljesen nyílt forráskódú verziókövető rendszer. 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.