Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Távközlési és Médiainformatikai Tanszék

Multidimenziós adatkezelés és bitmap indexelés

Szakdolgozat

Készítette
Szárnyas Gábor

Konzulens
dr. Gajdos Sándor, TMIT

Ipari konzulens
Balogh György, Kürt Zrt.

2011. december

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott *Szárnyas Gábor*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2011. december 9.

Szárnyas Gábor
hallgató

TANSZÉKVEZETŐ

SZAKDOLGOZAT FELADAT

Szárnyas Gábor
szigorló informatikus hallgató részére

Multidimenziós adatkezelés és bitmap indexelés

Az adatbázis-kezelés az 1990-es években igen nagy arányban relációs alapú termékekkel valósult meg, és egy tucat gyártó a felhasználók igen nagy százalékát szolgálta ki. 2005 után jelentek meg növekvő gyorsasággal azok a rendszerek, amelyek egy-egy speciális területen kiemelkedő tulajdonságok elérését célozták meg. A szakdolgozat keretében ilyen egyedi rendszerek létrehozásához is felhasználható egyes technológiák megismerése a cél.

1. A szakirodalom alapján foglalja össze
   a) a multidimenziós OLAP motorokkal
   b) a tömörített és a tömörítetlen bitmap indexekkel
   kapcsolatos fontosabb tudnivalókat (működési elv, sebesség, helyhasználat, karbantartás, ...)
2. Hasonlítsa össze a Jedox Palo, az icCube termékeket és a Count OLAP tanulmányt (a Kürt Zrt. saját fejlesztése) elsődlegesen a dokumentált funkcionalitásukon és alkalmazási korlátjaikon, másrészt pedig egy generált adathalmaz feldolgozásának tapasztalatain keresztül.
3. Végezzen méréseket tömörített és tömörítetlen bitmap index implementációkkal, és minősítse őket változatos szempontrendszer szerint.
4. Munkáját mintaszerűen dokumentálja és értékelje.

Tanszéki konzulens: Dr. Gajdos Sándor t. doc.
Külső konzulens: Balogh György, Kürt Zrt.

Budapest, 2011. szept. 26.

/ Dr. Henk Tamás /
tanszékvezető

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Távközlési és Médiainformatikai Tanszék
1117 Budapest, Magyar tudósok krt. 2.
Telefon +36 1 463 2448 Fax: +36 1 463 3107
E-mail: henk@tmit.bme.hu http://www.tmit.bme.hu

Kivonat

Napjaink hatalmas adathalmazainak feldolgozása komoly kihívást jelent a mérnökök az elemzők számára egyaránt. Az IBM szerint napi 2,5 exabájt (2.5×10^{18} bájt) adat keletkezik – a növekedés olyan sebességű, hogy az elmúlt két évben jött létre a ma tárolt adatok 90%-a. A vezető technológiai elemző és tanácsadó Gartner 2011. augusztusában felvette az új technológiák életciklusát ábrázoló hype cycle-re a „Big Data" (nagy adat) fogalmat.

Az analitikus adatbázisoknak gyakran közel valós időben kell eltárolniuk a beérkező adatokat, miközben a beérkező lekérdezéseket is gyorsan – néhány másodperc alatt – ki kell szolgálniuk. Így elengedhetetlen az adatok gyors és hatékony indexelése. A memória alapú adatbázis-kezelők (IMDB) az adatoknak a memóriában való tárolásával a lekérdezések gyorsítását egy drága, ugyanakkor rendkívül gyors tárolási réteggel támogatják.

Szakdolgozatom fő témája többdimenziós, memóriában tárolt adatbázisok tesztelése és gyors adatelérést biztosító indexstruktúrák készítése.

Dolgozatom első felében röviden ismertetem a dimenziós modellezés és az OLAP rendszerek fontosabb fogalmait. Megmutatom a tesztadatok mesterséges előállításának szükségességét és bemutatok egy parancssoros eszközt, amellyel tetszőleges méretű adathalmazok készíthetők. A generált adathalmazok felhasználásával bemutatok három egyedi OLAP rendszert, az icCube-ot, a Palot és a Kürt Zrt. Colap tanulmányát.

Dolgozatom második felében összefoglalom a többdimenziós indexelés és a bittérkép indexek fő problémáit. Részletesen tárgyalom a bittérkép tömörítés csúcsát képviselő algoritmusokat és megvalósítok kettőt. Részletes méréseket végzek a tömörített és a tömörítetlen reprezentációk összehasonlítására.

Zárásként összefoglalom az irodalomkutatás és az implementáció során nyert tapasztalataimat és körvonalazom az extrém méretű adathalmazok feldolgozásának jövőjét.

A dolgozat végén szójegyzék definiálja a felhasznált fogalmakat. A függelék tartalmazza az adathalmaz előállító eszköz nyelvtanát és a felhasznált függvénykönyvtárak és szoftverek telepítési útmutatóját.

Abstract

Processing huge volumes of data provides quite a challenge for engineers and analysts alike. According to IBM, 2.5 exabytes (2.5×10^{18} bytes) of data is created everyday—so much that the last two years account for 90% of all existing data. Leading information technology research and advisory company Gartner added "Big Data" to its hype cycle for emerging technologies in August 2011.

Analytic database systems often have to store incoming data streams near real time while also answering to queries in seconds. Therefore, quick and efficient indexing of data is a must. In-memory computing helps achieving these goals by providing an expensive but rapid storage layer.

In my thesis work, I focus on testing multidimensional in-memory analytic databases and creating index structures for swift data retrieval.

In the first half of the thesis work I briefly introduce the concept of dimensional modelling and OLAP systems. I show the necessity of using synthetical test data and present a command line tool for generating datasets of arbitrary size. The generated datasets are later used to review three unique OLAP systems, icCube, Palo and KÜRT Co.'s Colap study.

In the second half I discuss the core problems of multidimensional indexing and bitmap indices. I discuss the available state of the art algorithms for compressing bitmap indices and implement on-the-fly versions of two. I compared the compressed and uncompressed representations with detailed measurements.

In conclusion, I summarized my experiences gained in the research and implementation process and outlined the future of extreme data management.

A glossary clarifies the definitions and technologies used. The appendix contains the grammar for the data generation tool and the installation manual of the presented libraries and softwares.

Bevezető

A vendéglátóiparban tevékenykedő J. Lyons & Company céget 1887-ben alapították. A második világháború idején a cég Európa legnagyobb élelmiszergyártója és étteremlánca volt. 1947-ben a cég vezetői az Egyesült Államokba utaztak, ahol a Harvard Egyetemen megismerték a katonai célú számítások elvégzésére készült ENIAC-ot. Kiváló érzékkel rögtön felismerték a gép üzleti alkalmazásának lehetőségeit, és közben megtudták az is, hogy a nagy-britanniai Cambridge egyetem munkatársai egy hasonló gépen dolgoznak. A cég úgy döntött, hogy komoly összeggel támogatja a kutatást, így megépülhetett az EDVAC,[1] cserébe az egyetem segítséget nyújtott egy üzleti számítások elvégzésére alkalmas számítógép elkészítéséhez.

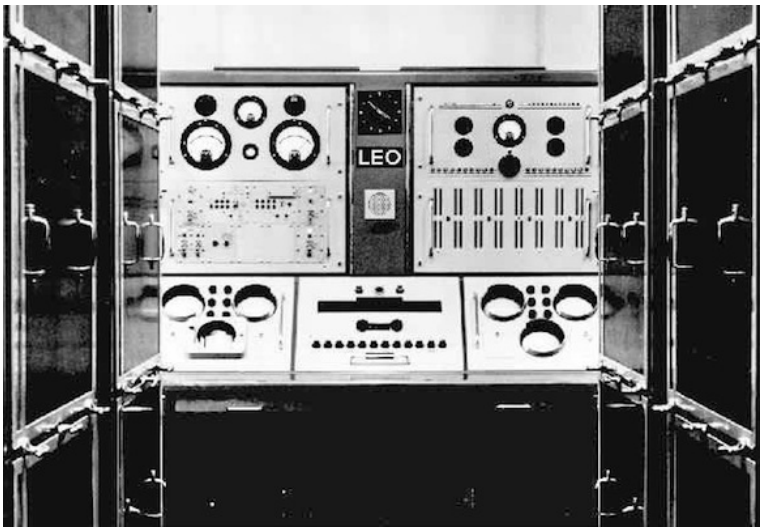
1951 őszére elkészült a Lyons Electronic Office I (LEO I), a világ első üzleti célokat szolgáló számítógépe. A gépet először logisztikai és ütemezési feladatokra használták. A LEO I a Lyons teaboltok napi tea- és aprósütemény forgalma alapján az éjszakai előállítási és szállítási terveket készítette el. Később további feladatokra kezdték alkalmazni, többek között bérszámfejtésre és leltározásra is. Képes volt egy alkalmazott fizetését 1,5 másodperc alatt kiszámítani – korábban ez egy gyakorlott ügyintézőnek is 8 percig tartott. A Lyons úttörő volt az *erőforrás-kihelyezés* (outsourcing) terén is, a gép kihasználatlan kapacitásait külső cégeknek értékesítették.

Az azóta eltelt több, mint fél évszázadban az informatika tudománya és technológiája hatalmas fejlődésen ment keresztül. Az adatbázis-kezelés és a mesterséges intelligencia lehetővé teszi olyan intelligens szoftverek készítését, amelyek akár felügyelet nélkül is képesek egy komplex rendszer – legyen az egy üzletlánc infrastruktúrája vagy egy nagy számítógépes hálózat – viselkedésének elemzésére és probléma esetén beavatkozásra.

Dolgozatom célja, hogy áttekintést nyújtson a többdimenziós adatkezelés alapfogalmairól és korlátairól, valamint bemutassa a többdimenziós lekérdezések egy lehetséges támogatását.

Az 1. fejezetben elemzem az általános adattárház archiktektúrát és az adatkocka fogalmat. A 2. fejezetben bemutatom a normalizált mátrixok mesterséges előállításának szükségességét, és ismertetem az erre a célra készített `mgen` eszközt. A 3. fejezetben áttekintem a többdimenziós adatokat kezelő piaci termékek egy részét, különös tekintettel a memóriában felépített adatstruktúrát használó megoldásokra. A 4. fejezetben felvázolom a többdimenziós lekérdezések tipikus problémáit és megoldásait, majd részletesen tárgyalom

[1] Az EDVAC (Electronic Discrete Variable Automatic Computer) volt a világ első bináris számrendszert és belső programvezérlést használó számítógépe.



1. ábra. *A LEO I (forrás: [The Register, 2011])*

a bittérképindexek használatának és tömörítésének problémáit. Az 5. fejezet tartalmazza az elvégzett mérések eredményeit. A munkát fogalomtár és függelék zárja.

1. fejezet

Adattárházak és OLAP rendszerek

Megfulladunk az információtól, miközben tudásra éhezünk.

(John Naisbitt)

A fejezetben röviden ismertetem az adattárházak létrejöttének okait. Elemzem az OLTP és az OLAP alkalmazások közötti különbséget és utóbbi típusait (ROLAP, MOLAP, HOLAP).

1.1. Hagyományos adattárházak

Az adatbázis-kezelő rendszerek egyik gyakori felosztása a rendszerek felhasználási módja szerint történik. A különböző felhasználási módok alapján megkülönböztetünk on-line tranzakció-feldolgozást (online transaction processing, OLTP) és on-line analitikus feldolgozást (online analytical processing, OLAP) végző rendszereket.

A tranzakciós rendszerek tipikusan *folyamat-orientált* szervezésűek, vagyis egy adott üzleti folyamatot (pl. termék eladása) támogatnak. A tranzakciós rendszereket jellemzően sok felhasználó használja egyidejűleg. Jellemző a beszúrások és a frissítések magas száma, miközben szigorú konzisztencia kritériumokat is teljesíteni kell. A tranzakciós rendszerek ezzel járó anomáliák (beszúrási, módosítási, törlési anomália [Gajdos, 2006]) elkerülése, az áteresztő képesség maximalizálása és a válaszidő minimalizálása érdekében normalizált adatstruktúrákon dolgoznak, egy lekérdezés általában csak néhány táblát és rekordot érint. A magas rendelkezésre állás kritikus, hiszen egy ilyen rendszer rövid leállása is súlyos anyagi és bizalmi veszteségekhez vezethet (gondoljunk egy banki tranzakciós rendszerre). A fenti előnyös tulajdonságokért cserébe a több táblát érintő lekérdező műveletek jellemzően lassúak, mert a nagyméretű táblák természetes illesztésének végrehajtása rendkívül erőforrásigényes.

Az analitikus rendszerek fő célja a döntéstámogatás, működésük jellemzően *témaorientált* (subject-oriented). A legtöbb rendszerben rövidebb leállások – bár nem kívánatosak – megengedhetők, nem feltétlenül járnak közvetlen veszteséggel. Ezeken a rendszereken jellemzően a vállalati hierarchia felsőbb szintjein dolgozó munkatársak futtatnak lekérdezéseket. A lekérdezések sokszor bonyolult, feltáró jellegű, így a tranzakciós rendszerekkel

szemben előre nem ismert, ún. „ad hoc” lekérdezések futtatására is fel kell készülni. Ezek jellemzően sok táblát érintenek és sokáig (órákig, napokig is) futhatnak. Az elemzések során gyakran lemondunk a finom granularitású adatok elemzéséről, helyette ún. aggregátumokon végezzük az elemzést.

A változatos adatforrásokból az adatok előkészítése és az aggregátumok előállítása az ETL (extract, transform, load) rövidítéssel fémjelzett betöltési folyamat felelőssége. A betöltési folyamat mindkét rendszerre komoly terhet ró, ezért tipikusan előre meghatározott időközönként (pl. éjjelente) a tranzakciós rendszer bezárása mellett történik – ez garantálja, hogy a betöltés során biztosan nem kerülnek új adatok a rendszerbe. Ez lehetetlenné teszi a legfrissebb adatok elemzését, közel egy napnyi holtidőt generálva a tranzakciós rendszer eseményei és az elemzések között.

1.2. Az OLAP története

Az Arbor Software-t 1991-ben alapították. Az alapítók felismerték, hogy a cégek a táblázatkezelő szoftvereket[1] nem csak megjelenítésre, hanem adattárolási, -elemzési és döntéstámogatási célokra is használták (ún. *spreadsheet-alapú döntéstámogatás* [Gajdos, 2011]). Gyakran több adatforrás adatait egyesítették a táblázatkezelőkbe, így az elemzések nem csak nehezen végrehajthatók, de nehezen ellenőrizhetők is voltak. Az Arbor Solutions terméke, az Essbase (Extended SpreadSheet dataBASE) olyan adatbázist kínált, amelyben kényelmesen elvégezhetők az adatelemzéshez szükséges műveletek [Roske, McMullen, 2008]. Az Arbor Software 1998-ban beolvadt a Hyperion Solutions cégbe, ami 2007 óta az Oracle tulajdona – az Essbase nevet azonban végig megtartották.

Korábban az adatbázisok jellemzően kétdimenziós struktúrákat definiáltak: minden táblához tartoztak mezők (attribútumok) és rekordok (sorok). Az Essbase elméletben nem korlátozta a dimenziók számát (csupán a rendelkezésre álló hardver teljesítménye jelent korlátot), ezzel egészen új távlatokat nyitva. A készítők Edgar F. Coddot kérték fel a termék elemzésére – a ’70-es években ő definiálta a 3NF, BCNF normálformákat, és alkotta meg a a relációs adatbázisokra vonatkozó 12 szabályt 1985-ben.[2] Coddot lenyűgözte az új technológia, és rögtön meglátta az abban rejlő potenciált.

A korábbi adatbázisok célja elsősorban az volt, hogy minél gyorsabban hajtsanak végre tranzakciós műveleteket. A háttértárak magas ára miatt szintén fontos szempont volt a tárhely minél hatékonyabb kihasználása. Ezeket Codd OLTP-nek nevezte.

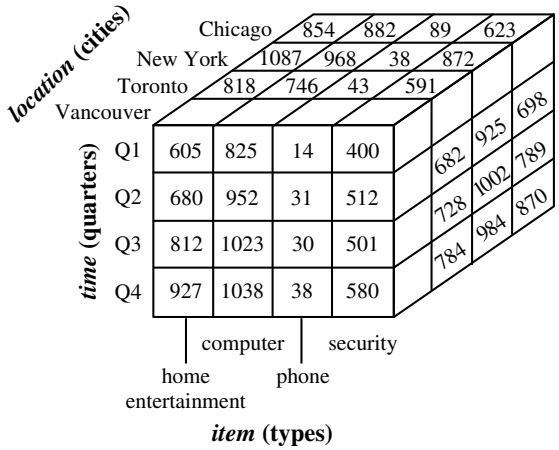
Az elsődlegesen elemzési célokat szolgáló adatbázisokra megalkotta az OLAP kifejezést, és definiált 12 szabályt, amelyeket egy „ideális” OLAP rendszernek teljesítenie kell.[3] Az azóta eltelt közel két évtizedben több másik cég is készített többdimenziós lekérdezéseket támogató adatbázis-rendszert, bár ezek száma jelentősen elmarad a „hagyományos” relációs adatbázis-kezelőktől.

[1] Ekkoriban még dúlt a Microsoft Excel és a Lotus 1–2–3 harca.

[2] `http://cims.clayton.edu/booth/ITDB%204201/Codd%20PDF.pdf`

[3] `http://www.olap.com/w/index.php/Codd's_Paper`

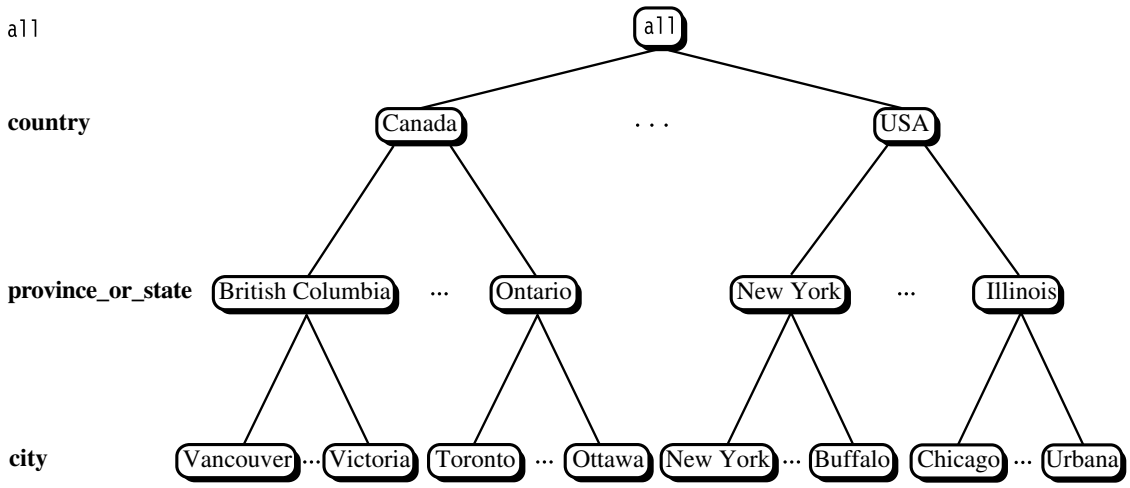
	location *= "Chicago"*				***location*** *= "New York"*				***location*** *= "Toronto"*				***location*** *= "Vancouver"*			
	item				***item***				***item***				***item***			
	home				*home*				*home*				*home*			
time	*ent.*	*comp.*	*phone*	*sec.*	*ent.*	*comp.*	*phone*	*sec.*	*ent.*	*comp.*	*phone*	*sec.*	*ent.*	*comp.*	*phone*	*sec.*
Q1	854	882	89	623	1087	968	38	872	818	746	43	591	605	825	14	400
Q2	943	890	64	698	1130	1024	41	925	894	769	52	682	680	952	31	512
Q3	1032	924	59	789	1034	1048	45	1002	940	795	58	728	812	1023	30	501
Q4	1129	992	63	870	1142	1091	54	984	978	864	59	784	927	1038	38	580



1.1. ábra. *Adatkocka reprezentáció (forrás: [Han, Kamber, 2006])*

1.3. Adatkocka modell

Az *adatkocka* az OLAP rendszerek működését szemléltető adatelemzési egység. Vizsgáljuk meg az adatkocka fogalmat egy egyszerű példán keresztül! Egy áruházban nyilvántartjuk, hogy melyik termékből hány darabot adtak el. Ez egyszerűen reprezentálható a népszerű táblázatkezelő szoftverekben egy olyan munkalapon (spreadsheet), ahol a vízszintes tengely a termékeket, a függőleges az eladási dátumokat tartalmazza. Hogyan valósítható meg az, ha több áruházról is szeretnénk rögzíteni ezeket az adatokat? Ezt nem tudjuk két dimenziós formában ábrázolni, kénytelenek vagyunk a munkalapot részekre osztani vagy új munkalapokat felvenni. Könnyen látható, hogy a táblázatos ábrázolás nem skálázódik jól, a dimenziók számának növekedésével – pl. tárolni kívánjuk az eladott termékek árát, a vevő adatait stb. – a tárolás és az elemzés egyre problémásabb lesz. Célszerű ezért a tárolt értékekre úgy gondolnunk, mintha azok egy többdimenziós térben vagy kockában lennének elhelyezve. A kocka dimenziói az egyes tengelyek megfelelői. Például három dimenzió esetén ábrázolva az x tengely a termék, az y a dátum, a z a hely dimenziónak felel meg. Az értékek a kocka belsejében, a dimenziók elemei által kijelölt pontokban helyezkednek el (1.1. ábra).



1.2. ábra. *A* location *(helyszín) dimenzió fogalmi hierarchiája (forrás: [Han, Kamber, 2006])*

1.3.1. Dimenziók

A dimenziók olyan entitások, amelyek mentén az adatot rendszerezzük és az elemzéseket futtatni kívánjuk. Minden dimenzión *fogalmi hierarchiát* (concept hierarchy) definiálunk. A fogalmi hierarchia olyan leképezések sorozata, amelyben az alacsony szintű, konkrétabb fogalmakat (pl. nap) magasabb szintű, általánosabb fogalmakra képezzük le (pl. hónap, év). Egy fogalmi hierarchia az 1.2. ábrán látható.

1.3.2. Cellák

A cellák a kocka dimenziók metszetében található pontok. Relációs adatbázist használó OLAP rendszerek (1.5.1.) esetén a cellák értékeit relációs táblákban tárolják, többdimenziós OLAP rendszerek (1.5.2.) esetén a memóriában lévő struktúrákban.

1.3.3. Adatkockák típusa

Az adatkockáknak két fő típusa ismert: a *hiperkocka* (hypercube) és a *multikocka* (multicube). Hiperkockák esetén egy dimenzió pontosan egy kocka része, multikockáknál viszont egy dimenzió több kockához is tartozhat, sőt lehetnek olyan dimenziók is, amelyek nem tartoznak egyik kockához sem. A továbbiakban mindig csak egy adatkockával fogok dolgozni, így a hiper- és a multikockák különbségei nem befolyásolják a munkát.

1.4. OLAP Műveletek

Az alábbiakban összefoglalom az adatkockán végzett leggyakoribb műveleteket [Gajdos, 2011]. A műveletek vizuális szemléltetése egy háromdimenziós adatkockán az 1.3. ábrán látható.

1.4.1. Aggregáció (roll up)

Aggregáció során valamelyik dimenzió mentén csökkentjük az adatok lekérdezésének granularitását, azaz kevésbé részletezetten nézzük azokat, pl. napok helyett heteket vizsgálunk.

1.4.2. Lefúrás (drill down)

A *lefúrás* az aggregáció ellentéte. Valamelyik dimenzió mentén növeljük az adatok lekérdezésének granularitását, azaz egyre részletezettebben nézzük azokat, pl. óra helyett perc bontásban. Aggregáció és lefúrás műveletek során a kocka dimenzióinak száma csak akkor változik, ha a dimenzióhierarchia gyökérelemébe aggegálunk vagy onnan indítjuk a fúrást.

1.4.3. Szeletelés (slice)

Szeleteléskor a kocka egyik dimenziója mentén rögzítjük a felvett értéket, így egy eggyel kisebb dimenziójú kockát kapunk. Egyes szerzők külön *szelekció* (selection, filter) műveletet definiálnak, ahol a dimenzió mentén olyan feltételre szűrünk, amit a dimenzió több eleme is kielégíthet [OLAP Council], míg mások a szeletelést általánosabban definiálják és a szelekció szinonimájaként használják [Han, Kamber, 2006].

1.4.4. Kockázás (dice)

Kockázás során a kocka egyes dimenzióin szelekciót végzünk, így egy azonos számú dimenziót tartalmazó részkockát kapunk.

Az utóbbi két művelet miatt a kocka dimenziói mentén történő navigálásra gyakran „slice and dice” néven hivatkoznak.

1.4.5. Forgatás (pivot)

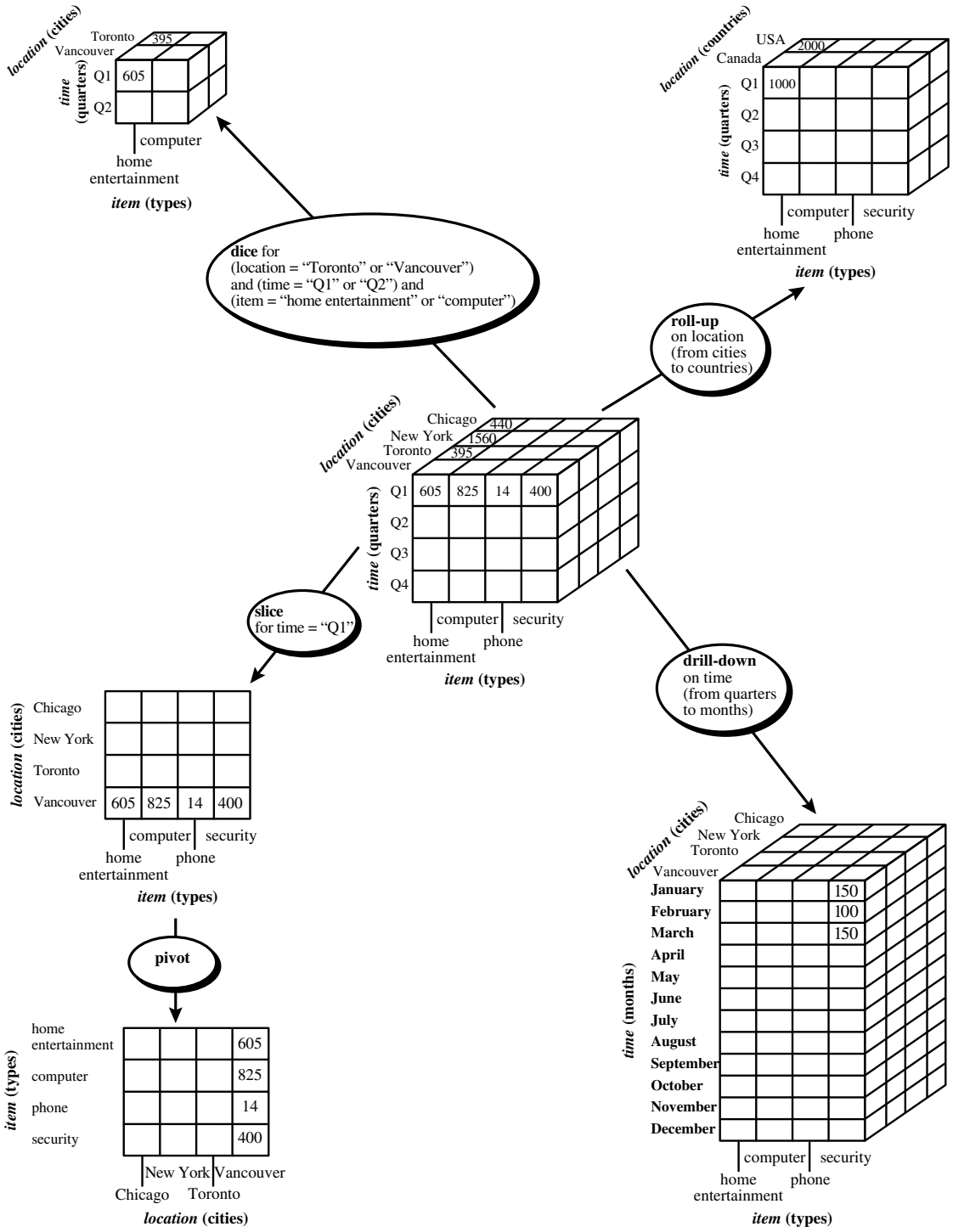
A *forgatás* művelet során megjelenített adathalmaz tengelyeit forgatjuk, a felhasználói elemzések támogatásának érdekében.

1.4.6. További műveletek

Egyes OLAP rendszerek további műveleteket is kínálnak. A *keresztülfúrás* (drill across) esetén a lekérdezéshez egynél több ténytáblát használunk, *részletezés* (drill through) esetén a kocka alsó szintjét összekötjük a forrásrendszerben hozzá tartozó relációs táblával.

1.5. OLAP rendszerek típusai

Az OLAP rendszerek egyik lehetséges csoportosítása a tárolási modell alapján történik. Három fő típust különböztetünk meg, egyedi előnyökkel, hátrányokkal és felhasználási esetekkel.



1.3. ábra. *Az OLAP műveletek szemléletesen (forrás: [Han, Kamber, 2006])*

1.5.1. ROLAP

A relációs OLAP (relational OLAP, ROLAP) rendszerek az adatot relációs adatbáziskezelő relációiban (tábláiban) tárolják. A „nyers" adatokat az ún. *ténytáblában* (fact table) tároljuk. A dimenziók menti navigálás a ténytáblával összekapcsolt *dimenziótáblák* (dimension tables) mentén lehetséges. A dimenziótáblák lehetnek *csillagséma* (star schema) vagy *hópehely séma* (snowflake séma) szervezésűek [Kimball, Ross, 2002].

A ROLAP rendszerek előnye, hogy a tárolt adatok lekérdezhetők SQL-en keresztül is.

1.5.2. MOLAP

A többdimenziós OLAP (multidimensional OLAP, MOLAP) rendszerek az adatokat ún. adatkockában tárolják, ezért gyakran nevezik ezeket adatkocka-rendszernek [Garcia, Ullman, Widom, 2001]. A MOLAP rendszerek gyakran olyan műveleteket is megvalósítanak, melyeket relációs lekérdezőnyelveken csak körülményesen és alacsony hatékonysággal tehetünk meg (lásd 1.4. alfejezet).

1.5.3. HOLAP

A HOLAP (Hybrid OLAP) rendszerek célja a ROLAP és a MOLAP architektúrák egyesítése a sebesség és a skálázhatóság javítása érdekében. Két fő megközelítés ismert: *horizontális partícionálás* (horizontal partitioning) esetén az adat egy részét multidimenziós struktúrákban, másik részét relációs adatbázisban tároljuk, míg *vertikális partícionálás* (vertical partitioning) esetén az aggregátumokat tároljuk MOLAP-ban és a részletes adatokat ROLAP-ban.

1.6. OLAP szabványok

1.6.1. MDX

Az MDX (multidimensional expressions) az OLAP adatbázisok lekérdező nyelve, amely nagyban hasonlít a relációs adatbázisok lekérdezésére használt SQL-re. Az MDX először a Microsoft 1997-es OLE DB for OLAP specifikációjában jelent meg. Idővel a szerver- és a kliensoldali alkalmazások készítői is beépítették termékeikbe, így az MDX mára a többdimenziós lekérdezések *de facto* szabványa lett, manapság (2011) a nagy OLAP rendszerek mindegyike támogatja.

1.6.2. XMLA

Az XMLA (XML for Analysis) az XML, SOAP és HTTP szabványokra épülő, OLAP rendszerek szabványos adatelérését biztosító formátum. Az XMLA szabványt a Microsoft és a Hyperion bocsátotta ki 2001 áprilisában. Később az SAS és több más cég is csatlakozott a támogatók közé. Az XMLA lekérdezőnyelve az MDXML, egy `<Statement>` XML elembe foglalt MDX lekérdezés.

1.7. Előkalkuláció

Az OLAP rendszerek létrejöttének egyik fő célja a lekérdezések minél gyorsabb futtatása. Ezért célszerű lehet az egyes hierarchiaszinteken és dimenziók mentén előre kiszámítani a kapott kockákat [DMKD97]. Például célszerű lehet minden hónapra, napra és órára kiszámítani a cellaértékek aggregátumait.

Az előkalkuláció fő problémája, hogy minden lehetséges részkocka előállítása esetén az igényelt tárhely mérete exponenciálisan nő. A *dimenzionalitás átka* (curse of dimensionality) néven ismert probléma oka, hogy n dimenzió esetén az előállítandó kockák száma a dimenziók összes lehetséges részhalmaza, vagyis 2^n. A bekövetkező *kombinatorikus robbanás* (combinatorial explosion) miatt erre gyakran úgy hivatkoznak, hogy a szükséges tárhely „felrobban," vagyis robbanásszerűen nő. A problémát fokozza, hogy a dimenziók fogalmi hierarchiával is rendelkezhetnek, ekkor az előállítható részkockák száma

$$\prod_{i=1}^{n} (L_i + 1)\,, \tag{1.1}$$

ahol a L_i az i. dimenzió szintjeinek számát jelenti (a $+1$ az összes elemet tartalmazó legfelső szint miatt szükséges).

A fentiek miatt a gyakorlatban csak akkor végeznek teljes előkalkulációt, ha a rendszer válaszideje kritikus, a dimenziók száma alacsony, és a kockák kellően kis méretűek [Kovács, 2004].

1.8. Visszaírás

Néha – például költségvetési tervek készítésekor – hasznos lehet ún. „mi lenne, ha ..." elemzések (what-if analysis) futtatása. Ilyenkor az aggregált mezők értékeit módosítjuk, és megfigyeljük, hogy hogyan változnak az alacsonyabb hierarchiaszinten lévő értékek. Az erre szolgáló művelet a *visszaírás* (writeback). A visszaírás nem feltétlenül egyenletesen osztja el az aggregált érték változását – pl. bizonyos területek költségvetése fix, ezeken nem lehet változtatni – ezt *súlyokkal* (weights) állíthatjuk be. A visszaírás MDX-ben az `UPDATE CUBE` utasítással végezhetjük el. A műveletre csak bizonyos elemzéseknél van szükség, ezért nem minden OLAP motor támogatja azt. A későbbiekben nem foglalkozom a visszaírással.

1.9. Összefoglalás

A fejezetben ismerettem a hagyományos adattárházak és az OLAP fogalom kialakulását. Ezután sorra vettem az OLAP adatkockákon végzett műveleteket és az OLAP rendszerek fő típusait, végül bemutattam a témakörben használt legfontosabb szabványokat.

2. fejezet

Számosság mátrixok előállítása

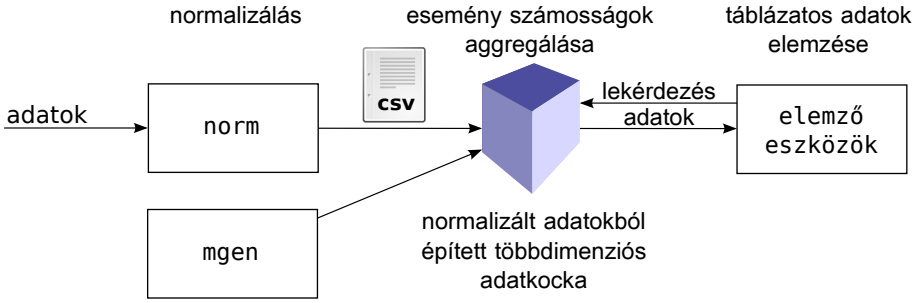
Vagy találunk ott utat, vagy építünk egyet.

(Hannibál)

Napjaink informatikai hálózatai korábban elképzelhetetlen komplexitásúra nőttek. A hálózatok viselkedésének és forgalmának elemzése mára külön tudomány, bonyolult statisztikai és gráfelméleti eszköztárral. Az elemzést támogató szoftverek fejlesztése sok tesztadatot igényel, ezért ezek előállítására saját alkalmazást fejlesztettem.

2.1. Az elemző rendszer architektúrája

Egy elemző rendszer tipikusan olyan nagymennyiségű adattal dolgozik, hogy nem célszerű az adatok eredeti formájában történő elemzése. Ezért az adatok ún. számosság normalizálás után kerülnek be az adatbázisba, ahol az elemző eszköz a lekérdezéseket futtatja. Egy ilyen architektúrája a 2.1. ábrán látható.



2.1. ábra. *Egy többdimenziós adatelemző rendszer architektúrája*

2.2. Az adatgeneráló eszközzel szemben támasztott követelmények

Egy informatikai hálózat forgalmának adataiból részletes információk nyerhetők ki a hálózat felépítéséről és felhasználóiról. Ez komoly biztonsági kockázatot jelent, ezért ezek bizalmas adatoknak minősülnek. Ezért a MOLAP motorok teszteléséhez és később az indexelés optimalizálásához felhasznált adatokat mesterségesen kellett előállítanom. Ehhez

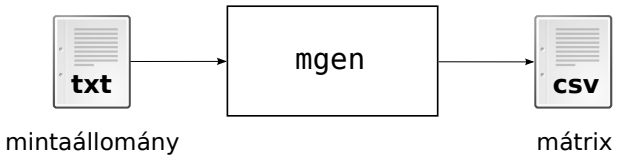
egy olyan adatgeneráló eszközre volt szükségem, mellyel hatékonyan állíthatunk elő tetszőleges méretű normalizált mátrixokat.

A többdimenziós adatokat gyakran meglepően egyszerű formátumban, vesszővel elválasztott szövegfájlokban (CSV, comma-separated values) tárolják. A fájl táblázatos formában tartalmazza az adatokat, ahol a táblázat attribútumai (oszlopai) az adatkocka egyes dimenzióinak felelnek meg.

A programban specifikálható a mátrix attribútumainak száma, az attribútumok kardinalitása, a felvett értékek eloszlása és a kapott sorok száma. Így lehetséges olyan mátrixok előállítása, amelyek a valódi hálózati forgalomhoz hasonló attribútumokkal és eloszlással rendelkeznek.

2.3. Az `mgen` adatgeneráló eszköz

A fenti követelmények kielégítésére készült az `mgen` parancssoros eszköz. A program működése a 2.2. ábrán látható. A program bemenete egy szöveges *mintaállomány*, amely tartalmazza az előállítandó mátrix paramétereit. Kimenete egy, a mintaállománynak megfelelő CSV fájl, ami tartalmazza az előállított mátrixot.



2.2. ábra. *Az mgen működése*

A program hívását a 2.1. felsorolás mutatja. A programot kötelezően négy paraméterrel kell meghívni: az első a mintaállomány, a második az előállítandó mátrix sorainak száma, a harmadik és a negyedik a kimeneti állományok.

```
$ ./mgen
mgen -- matrix generator
Usage: mgen <pattern_file> <number_of_lines>
<file_with_raw_datetimes> <file_with_formatted_datetimes>
$ ./mgen pattern.txt 100 tesztkocka.csv tesztkocka.cube
```

2.1. lista. *Az mgen eszköz működése*

2.4. Implementáció

A programot C++-ban implementáltam. Az implementálásához felhasználtam a Boost C++ Libraries függvénykönyvtárait, többek között a Spirit, a Tokenizer és a Random könyvtárakat. Az alábbiakban röviden összefoglalom az implementáció során felmerülő fő részfeladatokat.

2.4.1. Szövegelemző készítése

Az mgen bemeneti formátumának feldolgozásához szövegelemzőt készítettem a Boost.Spirit függvénykönyvtár felhasználásával.

A Boost.Spirit[1] könyvtár célja szövegelemzők és generátorok egyszerű és hatékony előállításának támogatása. A Spirit `boost::spirit::qi` moduljának segítségével *sablon metaprogramozás* (template metaprogramming) használatával készíthetünk ún. szövegelemzőt (parser). Más elemzőgenerátorokkal – pl. Flex–Bison, Lex–yacc – ellentétben a Spirit eszközei lehetővé teszik, hogy a nyelvtant tisztán a C++ nyelv eszközeivel készítsük el. A Qi lehetővé teszi, hogy az egyes szimbólumokhoz *szemantikus akciókat* (semantic action) társítsunk. Ezek olyan függvények, melyek akkor futnak le, ha a megadott szimbólum előfordul a szövegben.

Az mgen mintaállományainak nyelvtani szabályait a Qi modul eszközeivel készítettem el. Az mgen nyelvtani szabályai megtalálhatók az F.1. alfejezetben.

2.4.2. Dátumok előállítása

A 3. fejezetben bemutatott OLAP rendszerek közül az icCube és a Palo engedi a dátumformátum specifikációját, míg a Colap egy speciális formátumban várja azt. Ezért dátumokat kétféle formátumban állítottam elő: az ún. formázott dátumok `YYYY/MM/DD` és `mm:ss` alakúak,[2] a nyers dátumok egy speciális reprezentációban tárolódnak. Utóbbiban a dátum egy `YYYYMMDD` formátumú egész szám, az idő az éjféltől eltelt másodpercek száma.[3]

2.4.3. Véletlenszámok előállítása

Álvéletlenszámok generálása során a számítógép bonyolult algebrai műveletek sorozatával olyan számsorozatot generál, amely látszólag véletlennek tűnik. A valós adathalmazokban előforduló adatok gyakran jól közelíthetők valamilyen nevezetes eloszlással. Az mgen programban kétféle eloszlást valósítottam meg.

Diszkrét egyenletes eloszlás esetén a generált véletlenszám a megadott intervallumon felvehető egész értékek mindegyikét egyforma valószínűséggel veszi fel. Például a szabályos dobókockával dobott számok eloszlása egyenletes eloszlást követ az $1, 2, \ldots, 6$ számok halmazán. Az egyenletes eloszlású véletlenszámokat a Boost.Random függvénykönyvtár MT19937 generátorával állítom elő.[4]

A *Zipf-eloszlást* eredetileg az angol nyelvben előforduló szavak gyakoriságának jellemzésére vezették be. Zipf-eloszlás esetén az i. leggyakoribb érték gyakorisága $1/\sqrt{i}$-vel arányos. Később felfedezték, hogy sokféle típusú adat eloszlásánál fellelhető, Zipf-eloszlást követ például a Földön (anyanyelvként) beszélt nyelvek száma, Egyesült Államok államainak népessége és a földrengések előfordulása. Sok, számítógépes hálózatokkal kapcsolatos paraméter is Zipf-eloszlást követ [INFOCOM, 1999]. A Zipf-eloszlású véletlenszámokat egy

[1] `http://boost-spirit.com/home/`

[2] `Y`: év, `M`: hónap, `D`: hónap napja, `m`: perc, `s`: másodperc

[3] A dátum- és időformátumok közötti gyors konverziót támogató `Time.h` Balázs László munkája.

[4] Az MT19937-es egy ún. Mersenne twister. Nevét onnan kapta, hogy az előállított pszeudovéletlen számok periódusa $2^{19937} - 1$ hosszú, ami egy Mersenne-prím.

külső függvénykönyvtár segítségével állítottam elő [Christensen].

Egyes attribútumok, pl. a HTTP metódus és a HTTP státuszkód értékei egy előre becsülhető, de nem nevezetes eloszlással jellemezhetők, ekkor *súlyozott eloszlást* alkalmazok, előre definiált súlyokkal. A súlyok ismeretében az alábbi egyszerű algoritmussal generálható súlyozott eloszlás:

Adott az egyes értékek listája a hozzájuk tartozó súlyokkal $(w_1, w_2, \ldots, w_n)$ együtt. A megadott súlyoknak megfelelő eloszlású értékeket úgy generálhatunk, hogy előállítjuk a súlyok W összegét:

$$W = \sum_{i=1}^{N} w_i. \tag{2.1}$$

Előállítjuk a súlyoknak megfelelő intervallumokat:

$$I_{kmax} = \sum_{i=1}^{k} w_i \qquad (1 \leq k \leq N). \tag{2.2}$$

Ezután olyan R véletlenszámot generálunk, melyek a $(0, W)$ intervallumba esnek. A kapott R-nek megfelelő intervallum kijelöli a generált értéket.

A programhoz mellékeltem olyan konfigurációs állományokat, amelyek tartalmazzák a gyakori HTTP metódusok és státuszkódok eloszlásához tartozó súlyokat, így a valós adathalmazokból nyerhetőkhöz hasonló számosságmátrixok generálhatók [Agnosti, Nunzio, 2007].

Természetesen sok más eloszlás szerint is előállíthatunk adathalmazokat (normális eloszlás, exponenciális eloszlás stb.), azonban a gyakorlat azt mutatja, hogy az egyenletes és a Zipf-eloszlású adatokon már nagyon jól megfigyelhető a bittérkép tömörítő eljárások (4.5. alfejezet) hatékonysága [Wu, Stockinger, Shoshani, 2008].

2.5. Összefoglalás

A fejezetben ismertettem, hogy miért szükséges a mátrixok mesterséges előállítása. Bemutattam az `mgen` program implementációját és működését. A későbbiek során a méréseket a programmal előállított mátrixokon végeztem.

3. fejezet

OLAP rendszerek

A fejezet elején röviden ismertetem a kereskedelmi forgalomban kapható OLAP szervereket, majd részletesen összehasonlítok két MOLAP terméket, és bemutatom a Colap tanulmányt.

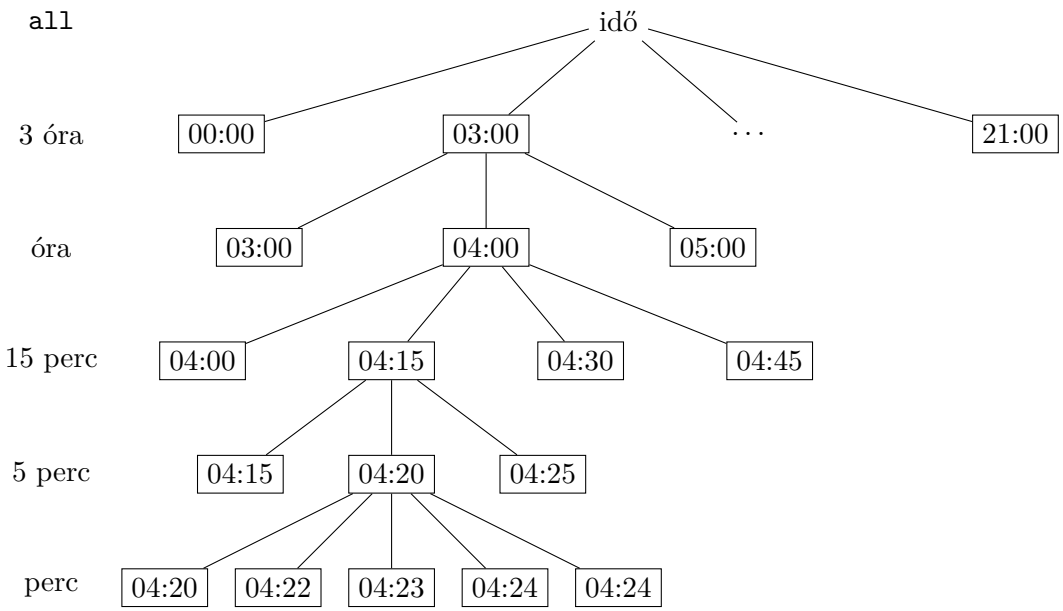
3.1. OLAP rendszerek összehasonlítása

Az OLAP rendszerek összehasonlítása többféle szempont szerint történhet. A megfelelő rendszer kiválasztásának fontos szempontjai a rendszer architektúrája (MOLAP, ROLAP, HOLAP), a támogatott platformok (Windows, Unix-szerű rendszerek) és a támogatott lekérdezőnyelvek (MDX, SQL). A leggyakoribb kereskedelmi OLAP rendszerek összehasonlítása a 3.1. táblázatban látható.

név	**gyártó**	**arch.**			**platf.**		**lek.**	
		MOLAP	**ROLAP**	**HOLAP**	**Windows**	***nix**	**MDX**	**SQL**
Essbase	Oracle	●	●	●	●	●	●	❍
icCube	Crazy Dev.	●	❍	❍	●	●	●	❍
Microsoft Analysis Services	Microsoft	●	●	●	●	❍	●	●
MicroStrategy Intelligence Serv.	MicroStrategy	●	●	●	●	●	●	●
Mondrian OLAP server	Pentaho	❍	●	❍	●	●	●	❍
Oracle Database OLAP Opt.	Oracle	●	●	●	●	●	●	●
Palo	Jedox	●	❍	❍	●	●	●	❍
SAS OLAP Server	SAS Institute	●	●	❍	●	●	●	❍
SAP BusinessObjects	SAP	●	❍	❍	●	●	●	❍
TM1	IBM	●	❍	❍	●	●	●	❍

3.1. táblázat. *Kereskedelmi OLAP szerverek összehasonlítása architektúra, platform és lekérdezőnyelvek szerint (források: F.2.)*
Jelmagyarázat: ● Az adott funkció elérhető. ❍ Az adott funkció nem érhető el.

A nagy gyártók termékeinek magas licenszelési költsége miatt két kisebb cég termékét, a Jedox Palot és az icCube-ot vizsgáltam meg részletesebben. A termékek bemutatásához



3.1. ábra. *Az idő hierarchia részlete*

mindkettővel elkészítettem egy egyszerű minta adatkockát az `mgen` eszközzel (2.3. alfejezet) generált adathalmazból. Az adathalmaz az F.1.1. mintaállomány alapján készült, így hat dimenziót tartalmaz: dátum, idő, forrás IP, cél IP, HTTP metódus, státuszkód. A cél olyan adatkockák elkészítése volt, amelyek minél egyszerűbbé teszik az adathalmaz szemléltetését, elemzését.

Az elkészített példákat XML állományokba exportáltam, ezek megtalálhatóak a mellékletben `TesztKocka.xml` és `TesztKocka.icc-schema` néven.

3.2. Idő dimenzió készítése

Az adathalmaz perces felbontású adatokat tartalmaz, amelyeket ötperces, negyedórás, órás és háromórás bontásban is szeretnék összesíteni. Ilyen opció egyik tesztelt MOLAP-ban sem szerepelt, ezért Perl nyelven készítettem egy szkriptet, amely egy olyan `idobelyegek.csv` állományt készített, amely tartalmazza egy nap perceihez tartozó nagyobb egységeket (3.1. ábra).

Fontos, hogy egy elem több hierarchiaszinten is megjelenthet, hiszen például a 20:15 időbélyeg a negyedórás, az ötperces és a perces bontásban is szerepel. Ezért a különböző szinten álló elemeket meg kell különböztetni, így ehhez az értékhez negyedórás felbontásban `[15m] 20:15`, az ötpercesben `[5m] 20:15`, a percesben `20:15` tartozik.

Az adatszerkezet nyilvánvalóan erősen redundáns, ugyanakkor csak $24 \times 60 = 1440$ sort tartalmaz, ezért tárhelyigénye a valódi adathalmazhoz képest elhanyagolható.[1] Egy takarékosabb tárolási módban – a MOLAP motorokban általában támogatott szülő-gyermek fa készítésével – kihasználhatnánk a fennálló szülő-gyermek hierarchiát. Ebben olyan kétatt-

[1]Könnyen látható, hogy az illeszkedő relációs séma függéshalmaza $F = \{minute \rightarrow minute5,\ minute5 \rightarrow minute15,\ minute15 \rightarrow hour,\ hour \rightarrow hour3\}$ lenne, az ezekből következő tranzitív függőségeket kellene megszüntetni a redundanciamentességhez.

```
hour3,hour,minute15,minute5,minute
...
[h3] 03:00,[h] 04:00,[m15] 04:15,[m5] 04:15,04:19
[h3] 03:00,[h] 04:00,[m15] 04:15,[m5] 04:20,04:20
[h3] 03:00,[h] 04:00,[m15] 04:15,[m5] 04:20,04:21
[h3] 03:00,[h] 04:00,[m15] 04:15,[m5] 04:20,04:22
[h3] 03:00,[h] 04:00,[m15] 04:15,[m5] 04:20,04:23
[h3] 03:00,[h] 04:00,[m15] 04:15,[m5] 04:20,04:24
[h3] 03:00,[h] 04:00,[m15] 04:15,[m5] 04:25,04:25
...
```

3.1. lista. *Az `idobelyegek.csv` egy részlete*

3.2. ábra. *A Palo logója*

ribútumú táblát kell definiálni, melynek első oszlopa a szülő, a másik a gyermek attribútumot tartalmazza. A megoldásomban a redundáns, egyszerűbb adatszerkezet használatát választottam.

3.3. Kimutatástábla

A többdimenziós adatstuktúrák szemléltetesen ábrázolhatók *kimutatástábla* (pivot table) segítségével, ahol a táblázat függőleges és vízszintes tengelyeit feleltetjük meg a többdimenziós adatszerkezet dimenzióinak. Ha egy tengelyen több dimenzió is található, a kimutatástábla adott tengely mentén a dimenziók elemeinek Descartes szorzatát jeleníti meg. Így ha egy tengelyre helyezzük el a dátum és az idő dimenziókat, a tengelyen minden lehetséges dátum–idő pár megjelenik, így az adott intervallum tetszőleges percéhez tartozni fog egy oszlop.

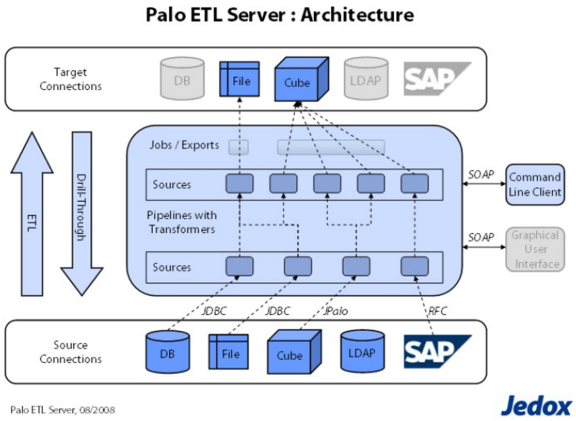
A kimutatástábla elforgatása a *forgatás* (pivot) OLAP művelettel végezhető el (1.4.5).

3.4. Palo

3.4.1. Áttekintés

A Palo OLAP Server a 2002-ben alapított német Jedox AG MOLAP adatbázisa, a Palo Suite (3.2. ábra) része.[2] A Palo Suite keretrendszer fő célja vállalatirányítási rendszerek kiszolgálása, ezért különös hangsúlyt fordít a felhasználható adatforrások sokszínűségére. Tölthetünk be adatot például relációs adatbázisokból, Microsoft Excel munkalapokból,

[2] `http://www.jedox.com/`



3.3. ábra. *A Palo architektúrája (forrás: [Palo Wiki])*

CSV fájlokból vagy más adattárházakból. A Palo Suite keretrendszer magában foglalja a Palo OLAP Server, a Palo ETL Server, a Palo Web, a Palo for Excel, a Palo Supervision Server és a Palo Client Libraries modulokat. A Palo OLAP Server elérhető zárt- és nyílt forráskódú változatban is. A teszteléshez a Premium Edition 30 napos próbaverzióját használtam.

3.4.2. Működés

A rendszer architektúrája a 3.3. ábrán látható. A betöltési folyamata az adatforrásokból kinyert adatokat *csővezetékeken* (pipelines) keresztül alakítja át és tölti be a célrendszerekbe, tipikusan egy adatkockába. A betöltési folyamatot később részletesen is bemutatom.

A Palo jól dokumentált, így bemutatása és a példa kocka elkészítése során komoly segítségemre voltak a rendszer kézikönyvei. A telepítést a *Setup Guide*, a kocka elkészítését a *Palo ETL Server*, az Excel integrációt a *3rd Party Software* útmutatók alapján végeztem [Jedox Palo Manuals].

3.4.3. Kocka készítése

A Palo telepítésének és indításának lépései az F.3.1. szakaszban olvashatók. Telepítés után a rendszert a Palo Weben, egy Google Web Toolkit (GWT) alapú webes felületeten keresztül érhetjük el.

A Paloban az adatkocka összeállítása az ETL folyamat része. Egy ETL projekt több elemből áll, ezeket az alábbiakban röviden ismertetem és bemutatom egy példa adatkocka konkrét megvalósítását. ETL projekt a webes felület *ETL Manager* menüpontja alatt hozható létre. A példa kocka elkészítéséhez létrehoztam a `TesztKocka` projektet.

A kocka tárolásához az *OLAP Manager* menüpont *Database Admin* menüjében létrehoztam a `KockaAdatbazis` adatbázist.

Kapcsolatok (connections)

Az ETL folyamat cél- és forrásrendszerét azonosítja. Adatforrásként használhatunk struktúrált szövegfájlokat, más Palo szervereket, LDAP szervereket, XML fájlokat és relációs adatbázisokat JDBC vagy ODBC meghajtókon keresztül.[3]

A példához feltöltöttem a `tesztkocka.csv`-t (*Upload file to ETL Server server*), és három kapcsolatot definiáltam.

`TesztKockaForras` *File* típusú kapcsolat, a fájl neve `tesztkocka.csv`, karakterkódolása UTF-8, elválasztó karaktere a vessző (`,`).

`IdobelyegForras` *File* típusú kapcsolat, a fájl neve `idobelyegek.csv`, beállításai az előzővel megegyezőek.

`CelSzerver` *Palo* típusú kapcsolat, a `localhost:7777` szerverre csatlakozik a `KockaAdatbazis` adatbázishoz az alapértelmezett felhasználónév–jelszó párossal.

Kinyerések (extracts)

A kinyerések segítségével meghatározhatjuk, hogy a forrásrendszerekből milyen adatokkal kívánunk foglalkozni, ill. előállíthatunk generált adathalmazokat az idő, dátum dimenziókhoz. Lehetséges más Palo szerverek kockáinak és dimenzióinak kinyerése is. A minta adathalmazban az idő és a dátum külön attribútumokként jelennek meg, ezért célszerű külön kezelni őket.[4]

A példa adatkockához az alábbi kinyeréseket definiáltam.

`DatumKinyeres` *Calendar* típusú kinyerés, gyökér csomópontja (root node) `All date`, nyelve `hu`.[5] A dátumhierarchia szintjei:

1. Évek: `years`, intervallum: 2011–2011, minta: `yyyy`.
2. Negyedévek: `quarters`, intervallum: 1–2, minta: `yyyy 'Q'Q`.
3. Hónapok: `months`, minta: `yyyy. MMM`.
4. Napok: `days`, minta: `yyyy/MM/dd`.

`IdobelyegKinyeres` *File* típusú kinyerés, forrása az `IdobelyegKinyeres` kapcsolat.

`TesztKockaKinyeres` *File* típusú kinyerés, forrása a `TesztKockaForras` kapcsolat.

[3] JDBC: Java Database Connectivity, ODBC: Open Database Connectivity. A JDBC Java, az ODBC C nyelven biztosít egységes programozási interfészt (application programming interface, API) a kliensek számára az adatbázishoz való hozzáféréshez.

[4] Ha mégis a Paloval készítettjük el az idő dimenziót, ne lepődjünk meg, ha figyelmeztetést kapunk, hogy duplikátum szerepel a generált adathalmazban. A figyelmeztetés oka, hogy a nyári időszámításról (daylight savings time, DST) az őszire történő óraátállítás miatt október utolsó szombatján két példány is létrejön a 02:00 időpontból.

[5] A nyelvet az ISO 639-1 szabvány szerinti rövidítésével kell megadni, ez megtalálható a `http://www.loc.gov/standards/iso639-2/php/English_list.php` oldalon.

régió	nettó ár	darabszám
fúró	1000	2
kalapács	500	4

$\Longleftrightarrow$

régió	mérték	#value
fúró	nettó ár	1000
fúró	darabszám	2
kalapács	nettó ár	500
kalapács	darabszám	4

3.2. táblázat. *Normalizált és denormalizált tábla*

Átalakítások (transforms)

Az átalakítások végzik a forrásrendszer(ek)ben található adatok megadott típusúra konvertálását. Az átalakítások bemenetei nem csak kinyerések, hanem átalakítások is lehetnek, így egy adathalmazon több átalakítás is végezhető.

A *táblaunió* (TableUnion) transzformációval táblák unióját készíthetjük el a relációalgebra $\cup$ operátorához hasonlóan. A *táblaillesztés* (TableJoin) transzformációval – akár különböző forrásból származó – adattáblák théta-illesztését végezhetjük el. Definiálhatjuk az illesztés típusát: a belső, bal- és jobboldali külső illesztés támogatott, a teljes külső illesztés viszont nem. Kötelező megadni a két illesztendő tábla forrását és az illesztés menti kulcsokat.

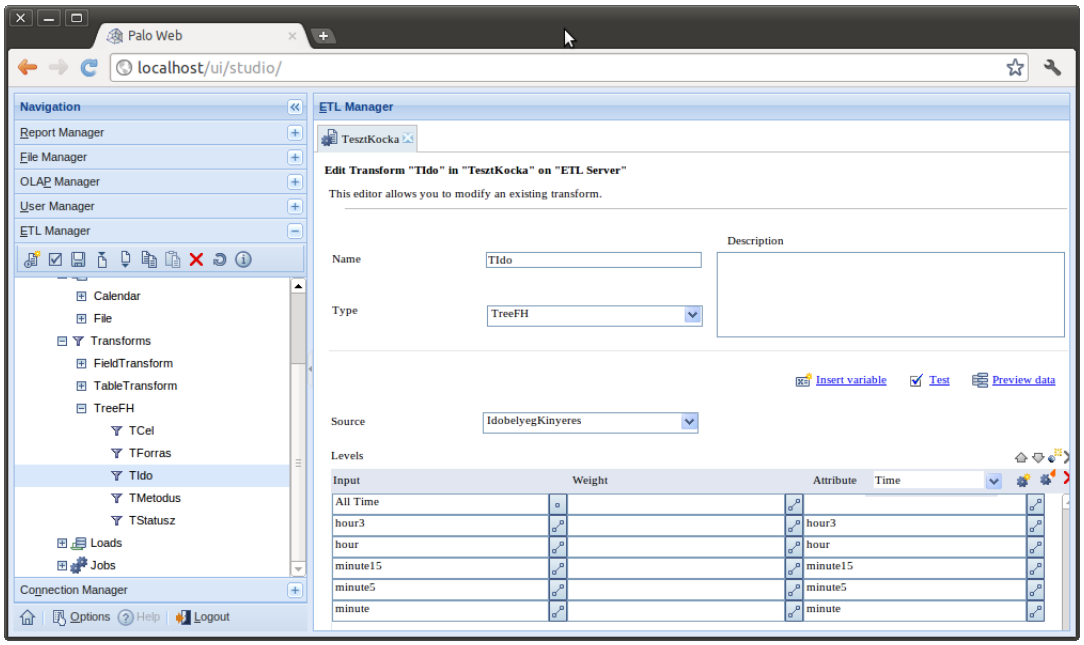
A *táblaátalakítás* (TableTransform) segítségével készíthetjük el az aggregációkat, ill. végezhetünk normalizációt és denormalizációt. Fontos, hogy utóbbiak nem egyeznek meg a relációs sémák tervezésénél alkalmazott normalizációfogalommal; itt denormalizálással új attribútumok definiálhatók, ha megadjuk, hogy melyik oszlopok tartalmazzák az attribútum–felvett érték párokat. A normalizációra egy példa a 3.2. táblázatban látható. Az adott *mértékhez* (measure) tartozó értéket alapértelmezésben a `#value` oszlop tárolja.

A *hierarchiafa* (tree full hierachy, TreeFH) transzformáció fát épít a megfelelő attribútumok értékeiből. A fának hierarchiákat is definiálhatunk úgy, hogy megadjuk, hogy az hierarchia egyes szintjeihez melyik oszlop tartozik (ez erősen redundáns adathalmazt feltételez, amelyet például a táblaillesztés átalakítással készíthetünk el). A kapott fából a megfelelő betöltéssel dimenzió készíthető. Hasonló hierarchiafa építhető a *szülő-gyermek fa* (parent-child tree, TreePC) transzformációval, ez két oszlopot vár: egyik a szülő, másik a gyermek elemet írja le.

`TDatum` *FieldTransform* típusú átalakítás. Forrása a `fh TesztKockaKinyeres`, a `date` attribútumot képzi le egy azonos nevű attribútumra a *DateFormat* típusú `DatumFormatum` függvényen keresztül. Utóbbi felelős a dátumformátum felismeréséért, amely a `date` mezőt értelmezi `yyyy/MM/dd` formátummal.

`TIdo` *TreeFH* transzformáció, forrása az `IdobelyegKinyeres`, szintjei:

1. Egész nap: `All time`.
2. Háromórás felbontás: `hour3`.
3. Órás felbontás: `hour`.
4. Negyedórás, ötperces, perces: az előzőekhez hasonlóan `minute15`, `minute5`, `minute`.



3.4. ábra. *Idő dimenziót előállító transzformáció készítése a Paloban.*

Fontos, hogy a legfelső, egész napot tartamazó szint típusa *konstans* (constant), míg a többié nem *referencia* (reference). A `TIdo` átalakítás előállítása a 3.4. ábrán látható.

`TForras`, `TCel`, `TMetodus`, `TStatusz` *TreeFH* transzformációk, egyszerű kétszintű hierarchia felépítésére: a hierarchia első szintje az összes elemet tartalmazza, a második pedig az atomi elemeket. Forrásuk a `FajlKinyeres`. Az `TIdo` átalakításhoz hasonlóan a legfelső szint itt is konstans típusú. Természetesen ennél bonyolultabb hierarchiák is elképzelhetők, a HTTP státuszkódok csoportosíthatók a kezdő számjegyük alapján, az IP címek tartományokra bonthatók stb.

`TAggregacio` *TableTransform* típusú átalakítás. Forrása a `TesztKockaKinyeres`. Egyetlen mértéket definiáltam: ez a `count`, aggregáció típusa *sum*, adattípusa numerikus.

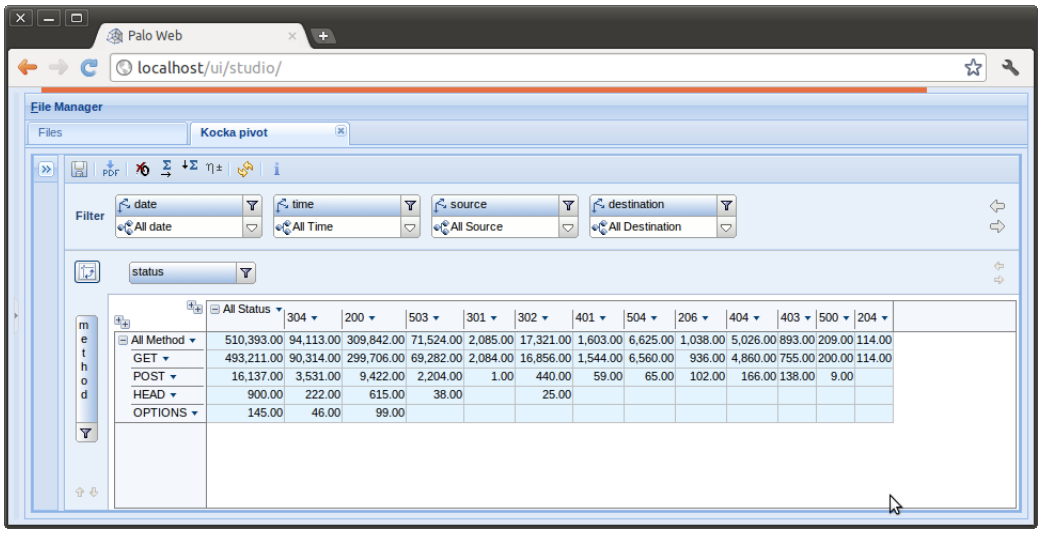
Betöltések (loads)

A betöltések végzik a kinyerésekből és az átalakításokból kapott adatok célrendszerbe töltését. A betöltés történhet kockába, dimenzióba, fájlba és relációs adatbázisba.

Példámban az alábbi betöltéseket definiáltam. A betöltés célja minden esetben a `CelSzerver` kapcsolat. A betöltés nem inkrementális, ezért *módja* (standard mode) *létrehozás* (create).

`KockaBetoltes` *Cube* típusú betöltés. A *forrás* (source) a `FajlKinyeres`, a *cél kocka neve* (target cube name) `Kocka`.

`DDatum`, `DIdo`, `DForras`, `DCel`, `DMetodus`, `DStatusz` *Dimension* típusú betöltések. Az egyes dimenziók forrása egy-egy átalakítás (`TDatum`, `TIdo`, ...), céljuk a betöltött adathalmaz megfelelő attribútumának neve (`date`, `time`, ...).



3.5. ábra. *Kimutatástábla a Palo weben*

Feladatok (jobs)

Külön definiálnunk kell az ETL folyamat részfeladatait, vagyis azt, hogy milyen betöltéseknek kell lefutni a dimenziók és az adatkockák létrehozásához.

A kocka betöltéséhez egy feladatot definiáltam, ami sorra betöltötte a `DDatum`, `DIdo`, `DForras`, `DCel`, `DMetodus`, `DStatusz` dimenziókat és a `Kocka` kockát.

Változók (variables)

Az egyes feladatok futása változókkal paraméterezhető. Gyakori eset, hogy egy nagy adathalmazból csak egy évnyi adatot kell betöltenünk, de nem ismerjük előre, hogy melyiket. Ilyenkor – ha nem akarjuk külön elkészíteni a szükséges kinyeréseket, átalakításokat, betöltéseket és feladatokat – definiálhatunk változókat, amelyeket a betöltési folyamat beállításai során felhasználhatunk (pl. szűrhetjük az adatokat a dátum mentén a `${ev}` változó értékére).

A példa ETL folyamathoz nem definiáltam változókat.

3.4.4. Kocka lekérdezése

Lekérdezés webes felületen

Az icCube-ban a kocka a *File Manager*-ben jeleníthető meg *New Palo Analyzer Report* létrehozásával, a `KockaAdatbazis` `Kocka` adatkockájának kiválasztásával. A megjelenő kimutatástábla tengelyeit és szűrési feltételeit beállítva a kocka tetszőleges nézetét vizsgálhatjuk, az egyes dimenziók elemei mentén végezhetünk lefúrásokat és felgörgetéseket.

A webes felületen MDX lekérdezéseket nem adhatunk meg, de amennyiben telepítve van a Palo ODBO Provider, XMLA formátú lekérdezéseket futtathatunk a `http://<PaloHost>:4242/xmla` címen. Az MDX-en túl a Palo egy saját HTTP API felületet is nyújt, amely a *lekérdezési karakterlánc* (query string) feldolgozása után pontosvesszővel tagolt CSV szövegfájlt ad vissza. Az API dokumentációja elérhető a `http://<PaloHost>:`

3.6. ábra. *Az icCube logója*

`7777/api` címen. A 3.2. felsorolásban a `curl` parancssoros eszközzel futtattam HTTP lekérdezéseket. A bejelentkezés után kapott *munkamenet azonosítóval* (session id) indítottam lekéréseket, amelyek sorra visszaadták a szerveren lévő adatbázisokat, az azokban tárolt adatkockát és dimenziókat, majd kijelentkeztem a munkamenetből.

```
$ # md5(admin) = 21232f297a57a5a743894a0e4a801fc3
$ curl "http://localhost:7777/server/login?user=admin&
        password=21232f297a57a5a743894a0e4a801fc3"
SHUe;3600;
$ curl "http://localhost:7777/server/databases?sid=SHUe"
7;"KockaAdatbazis";25;26;2;0;1868401782;
$ curl "http://localhost:7777/database/cubes?sid=SHUe&database=7"
162;"Kocka";6;133,135,139,137,141,143;414983790;8300;2;0;1868401782;
$ curl "http://localhost:7777/database/dimensions?sid=SHUe&
        database=7"
11;"datum";181;0;1;0;0;12;10;11;0;
133;"date";191;4;5;4;0;134;150;151;2;
135;"time";1857;5;6;5;0;136;152;153;2;
137;"destination";3;1;2;1;0;138;154;155;2;
139;"source";6;1;2;1;0;140;156;157;2;
141;"method";5;1;2;1;0;142;158;159;2;
143;"status";13;1;2;1;0;144;160;161;2;
$ curl "http://localhost:7777/server/logout?sid=SHUe"
1;
```

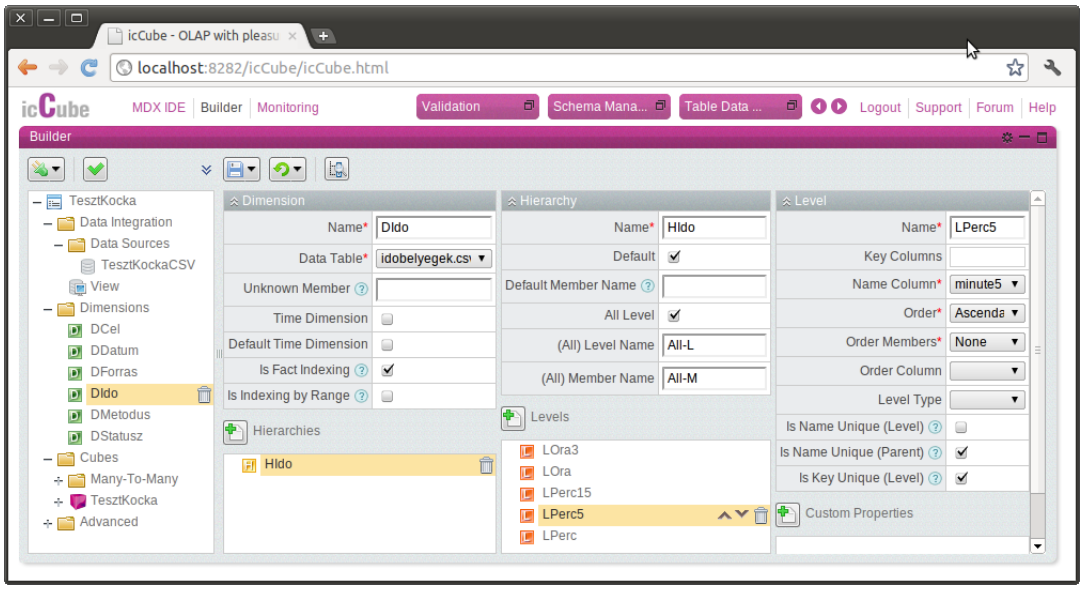
3.2. lista. *Palo adatkocka lekérdezése HTTP API-n*

3.5. icCube

3.5.1. Áttekintés

Az icCube (3.6. ábra) a svájci CRAZY Development terméke.[6] Az icCube a cég saját fejlesztésű *Pegasus* MOLAP motorját használja a többdimenziós lekérdezések hatékony kiszolgálására. Az adatkocka lehetséges adatforrásai: CSV fájlok, Excel munkalapok, relációs adatbázisok és adatfolyamok (pl. HTTP protokollon keresztül) [icCube Documentation]. Dolgozatomban a 2011. október 17-én megjelent 2.0 ingyenes Community Editiont használtam, amely legfeljebb 512 MB memóriát használ. Az Enterprise kiadás olyan pluszfunkciókat tartalmaz, mint a visszaírás, többnyelvű felhasználói felület, korlátlan memóriakihasználás, lekérdezések futtatása több processzormagon és betöltött kockák perzisztálása.

[6] `http://www.crazy-development.com/`, `http://www.iccube.com/`



3.7. ábra. *Dimenziók létrehozása icCube-ban*

3.5.2. Működés

Az icCube telepítésének és indításának lépései az F.3.2. szakaszban olvashatók. Az icCube a Palohoz hasonlóan szintén GWT-alapú webes kezelőfelülettel rendelkezik. Az icCube a betöltés során nem végez előkalkulációt [icCube OLAP Server].

A Palohoz hasonlóan az icCube is képes a dátum és az idő dimenzió automatikus előállítására az *idő varázsló* (Time Wizard) funkcióval. Sajnos a legmagasabb felbontás is csak félóránkénti részletességet tesz lehetővé.

3.5.3. Kocka készítése

Az adatkocka tartalma a *Builder* modulban található *Schema Manager* menü *Create Schema* menüpontjával állítható össze. A példában a `TesztKocka` sémán dolgoztam. Az alábbiakban összefoglalom az icCube sémakészítési folyamatának lehetőségeit és a példakocka előállításakor végzett lépéseket.

Adatintegráció, adatforrások (data integration, data sources)

Az icCube többféle adatforrást támogat: használhatunk Excel (.xls) és CSV fájlokat, JDBC-n keresztül lekérdezhető relációs adatbázisokat és HTTP szervereket. Gyors prototipizálásra ajánlott az *In-Memory* opció használata, amellyel kisebb méretű CSV formátumú adathalmazokat tárolhatunk a szerver memóriájában.

A példában a CSV típusú `TesztKockaForras` adatforrást hoztam létre. Hozzáadtam a `tesztkocka.csv` és a `idobelyegek.csv` adattáblákat, és beállítottam az egyes attribútumokhoz a megfelelő típusokat. A *dátumkonverzió mintája* (date converter pattern) `yyyy/MM/dd`.

Dimenziók (dimensions)

A Palohoz hasonlóan definiálhatunk dimenziókat a teljes hierarchia (Multi-Levels) és a szülő-gyermek kapcsolatok (Parent/Child) megadásával. Egy dimenzión belül több *hierarchiát* (hierarchy), egy hierarchián belül több *szintet* (level) is definiálhatunk. Az egyes szinteken meg kell határoznunk azt a *névattribútumot* (name attribute), amely a kockában megjelenik. Ha a névattribútum önmagában nem elég a dimenzió és a ténytábla összekapcsolásához, specifikálhatunk *kulcsattribútumokat* (key columns) is.

Dátum és idő dimenziókat készíthetünk a korábban már említett idő varázsló segítségével. A Path Hierarchy funkcióval *elérési utakból* (path) definiálhatunk hierarchiákat, így például Java/C# osztályok elérési útjait alakíthatjuk hierarchikus struktúrákká. Készíthetünk olyan *statisztikai* (Statistical/Utility) dimenziókat, amelyek nem kapcsolódnak a ténytáblához. Így olyan konverziókat írhatunk, amelyeket később több attribútumon is futtathatunk.

Az alábbiakban felsorolom a létrehozott dimenziókat. Azt az elnevezési konvenciót használtam, hogy a dimenziók nevei D, a hierarchiák nevei H, a szintek nevei L betűvel kezdődnek.

`DDatum` Az idő varázsló állítja elő. A `HDatum` hierarchia szintjei és típusai: `LEv` (year), `LNegyedev` (quarter), `LHonap` (month), `LNap` (day).

`DIdo` Többszintű hierarchia, forrása az `idobelyegek.csv` tábla.

`DForras` Többszintű hierarchia, forrása a `tesztkocka.csv` tábla. Egyetlen hierarchiája a `HForras`, amelynek egyetlen szintje az `LForras`. A névattribútum `source`, a kulcsattribútumok halmaza ezzel megegyezik, ezért üres.

`DCel, DMetodus, DStatusz` A `DForras`-hoz hasonlóan a `destination`, `method`, `status` attribútumok megfelelői.

Kockák (cubes)

A kockákhoz felvehetők *tények* (facts). Beállításukhoz szükséges a forrás tábla neve, és a dimenzió → attribútum leképzések megadása. Készíthetők továbbá *több-több dimenziók* (many-to-many dimension), ahol egy adott hierarchia elemei több-több kapcsolaton keresztül vannak összekötve.

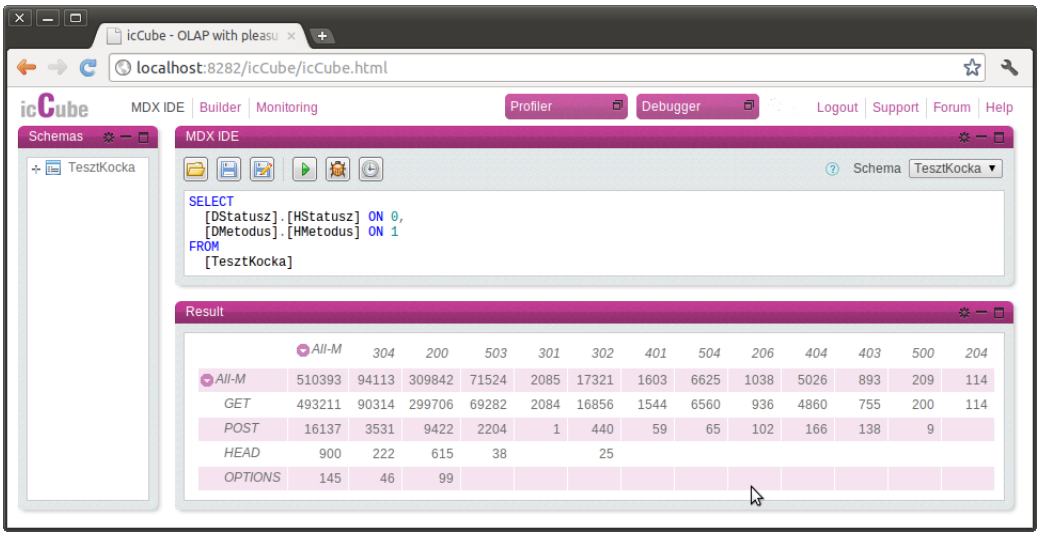
Létrehoztam a `TesztKocka` kockát és hozzáadtam a `count` tényadatot. Az *attribútumleképzéseket* az alábbiak szerint állítottam be:

`DDatum` `LNap` → `date`

`DIdo` `minute` → `time`

`DForras` `source` → `source`

`DCel, DMetodus, DStatusz` A `DForras`-hoz hasonlóan.



3.8. ábra. *MDX lekérdezés végrehajtása icCube-ban*

Haladó (advanced)

A haladó funkciók között MDX kifejezésekkel definiálhatunk *számított értékeket* (calculated member), *függvényeket* (functions) és *halmazokat* (sets). A példa során nem alkalmaztam egyik típust sem.

3.5.4. Kocka lekérdezése

Az MDX IDE felületen MDX lekérdezéseket fogalmazhatunk meg, az eredményeket kimutatástáblán tekinthetjük meg.

```
SELECT
  [DStatusz].[HStatusz] ON 0,
  [DMetodus].[HMetodus] ON 1
FROM
  [TesztKocka]
```

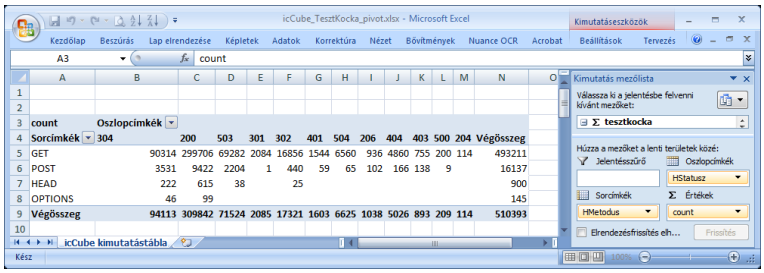
3.3. lista. *MDX lekérdezés a* `Teszkocka` *adatkockán*

Közvetlenül futtathatunk XMLA lekérdezéseket a `http://<icCubehost>:8282/icCube/xmla` címen. A `curl` parancssoros eszközzel különböző protokollokra (pl. FTP, HTTP, SMPT stb.) küldhetünk kéréseket és megfigyelhetjük a generált választ. Egy példa XMLA lekérdezés található az F.4. alfejezetben.

3.5.5. Kimutatástábla készítése Excelben

A Palohoz hasonlóan az icCube-ból is készíthető kimutatástábla Excelben. Ez a 2007-es verzióban a Beszúrás | Kimutatástábla (Insert | Pivot Table) menüpontra kattintva érhető el. Az icCube a *Microsoft SQL Server Analysis Services*-szel kompatibilis felületet nyújt a meg kell adnunk a külső adatforrás kapcsolatát, amely alapértelmezésben a `http://<icCubeHost>:8282/icCube/xmla` címen érhető el a megfelelő autentikációval.

A kapott kimutatástábla a 3.9. ábrán látható.



3.9. ábra. *Kimutatástábla készítése Microsoft Excelben*

3.6. Colap

3.6.1. Áttekintés

A Colap (Count OLAP) tanulmány a Kürt Zrt. saját fejlesztése. A piaci OLAP eszközök széles funkcionalitást nyújtanak, cserébe konfigurálásuk összetett, a betöltés sebessége meglehetősen lassú. A Colap tanulmány célja annak vizsgálata, hogy egy egyszerű, csupán néhány alapvető funkciót megvalósító MOLAP motor milyen sebességet tud elérni a piaci termékekhez viszonyítva. A Colap C++ nyelven készült.

3.6.2. Működés

A Colap az alábbi megkötések mellett használható.

- Csak egyetlen adatkockán dolgozunk.
- Az adathalmaz első két dimenziója a dátum és az idő, amelyek a 2.4.2. szakaszban ismertetett formátumban kódoltak.
- Egyetlen tényadatot kívánunk aggregálni, az *esemény számosságot* (count).
- A program nem nyújt szabványos felhasználói felületet.

3.6.3. Kocka készítése

A megadott CSV formátumú fájl alapján a Colap automatikusan elkészíti a szükséges dimenziókat és felépíti az adatkockát. A kocka elkészítése a 3.4. felsorolásban látható.

```
# a parancsfajl.txt tartalma
LOAD AS cube ~/kockak/tesztkocka
$ time ./colap < parancsfajl.txt
loaded.
```

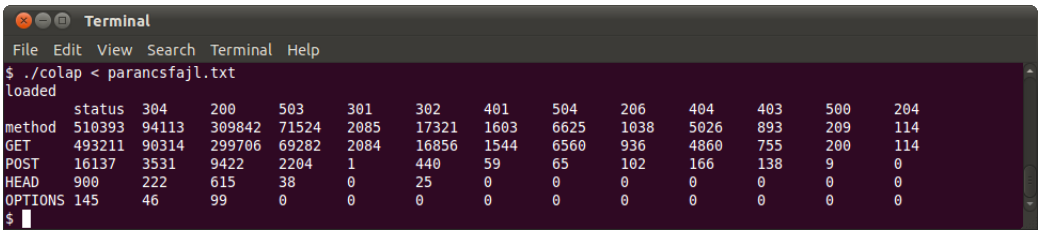
3.4. lista. *Kocka készítése a Colap programban*

3.6.4. Kocka lekérdezése

A lekérdezéseket egy, az MDX-hez hasonló nyelven fogalmazhatjuk meg. Egy példa lekérdezés a 3.5. felsorolásban látható. A Colap `.cube` kiterjesztésű fájlokat dolgoz fel, ezért az importálandó `tesztkocka.cube` fájl kiterjesztését nem kell megadni.

```
LOAD AS TesztKocka "/cubes/tesztkocka"
SELECT { method.@".*"} ON 0, { status.@".*" } ON 1 FROM TesztKocka
```

3.5. lista. *Lekérdezés parancsfájlja a Colap programban*



3.10. ábra. *Lekérdezés a Colapban*

A Colap nem szolgáltat külön eszközt a lekérdezések idejének mérésére, ezért ezt a `time` Unix paranccsal mértem. A `time` parancs által szolgáltatott értékek közül a *valós* (real) időt vettem figyelembe – ezt érzékeli ugyanis a felhasználó, valamint a többi program kimenetéből is ez az érték számítható ki.

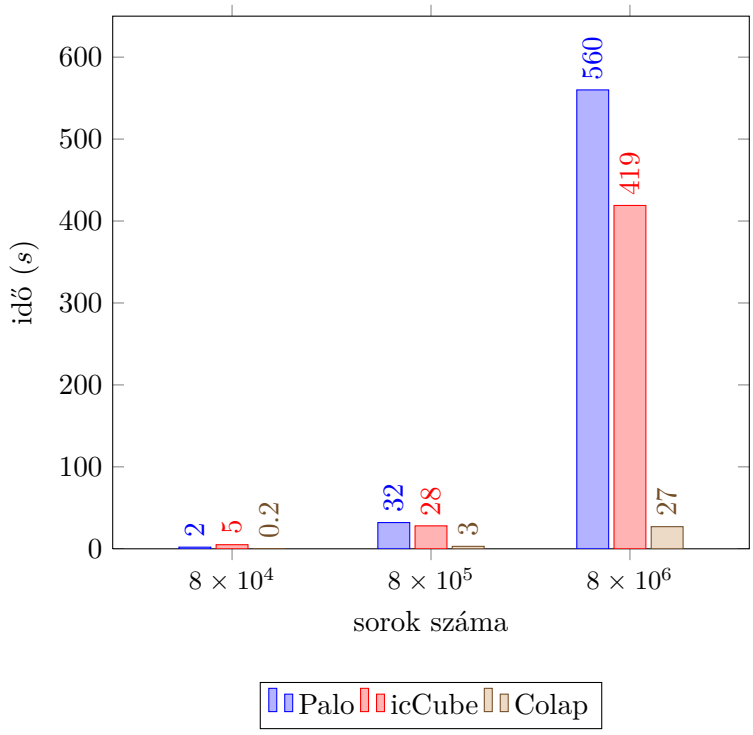
3.7. Betöltési idők

A teszthez az `mgen` eszközzel generáltam 80 ezer, 800 ezer és 8 millió soros adathalmazokat, majd betöltöttem azokat a fenti OLAP alkalmazásokba. A mérési eredmények a 3.11. ábrán láthatók. Az icCube és a Palo hasonlóan teljesítenek, a Colap több mint egy nagyságrenddel gyorsabb náluk. Fontos tulajdonság továbbá, hogy a Colap közel lineárisan skálázódik, míg a többi programnál enyhe exponenciális jelleg figyelhető meg.

A lekérdezési idők mindhárom alkalmazásnál rendkívül gyorsak voltak (a kért adatok kevesbb, mint 1 másodperc alatt megjelentek), ezért erre nem végeztem külön méréseket.

3.8. Összefoglalás

A fejezetben bemutattam a piacon elérhető OLAP megoldásokat, valamint részletesen ismertettem a két MOLAP rendszert, a Palot és az icCube-ot. Ezek gazdagon konfigurálhatók és változatos forrás- és célrendszereket kezelnek, betöltési folyamatuk meglehetősen lassú. A könnyűsúlyú, erősen specializált Colap megoldás egyszerű adatkockák készítését rendkívül hatékonyan végzi el.



3.11. ábra. *A Palo, az icCube és a Colap betöltési idői különböző méretű adathalmazokon*

4. fejezet

Bittérkép indexek

– Itt van – mutatott egy pontra a kapitány a szigettenger térképén. – Mint látja, a térképbe belerajzoltam az újonnan keletkezett szigeteket.
(Jules Verne – Nemo kapitány)

A fejezetben áttekintem az indexstruktúrák és a bittérkép indexek felhasználását. Ismertetem utóbbiak tömörítésének aktuális kutatási eredményeit, majd bemutatom az ezek alapján elkészített két saját implementációmat.

4.1. Motiváció

Ha egy nagy adathalmazon kívánunk lekérdezéseket futtatni, azok futási ideje az adatok mennyiségével egyre nő, ezzel romlik a válaszidő, így a felhasználó munkájának folytonossága. Ezért a gyakorlatban gyakran valamilyen segédstruktúrát rendelnek az adathalmazhoz, amely lerövidíti az egyes adatrekordok megtalálásához szükséges időt. Többféle megközelítés ismert: a hashelés, a sűrű index, a ritka index (B*-fa) egyedi előnyökkel és hátrányokkal rendelkeznek, de közös jellemzőjük, hogy eredeti formájukban csak egydimenziós lekérdezéseket támogatnak hatékonyan [Gajdos, 2006].

4.2. Többdimenziós lekérdezések támogatása

Gyakran felmerülő igény – hagyományos relációs adatbázisok mellett például térinformatikai rendszerekben és OLAP kockákban –, hogy az adathalmaz több kulcs szerint is kereshető legyen. A korábban említett alapvető indexstruktúrák *egydimenziósak*, azaz csak egyetlen kulcs szerinti keresést támogatnak. Még ha egy keresési kulcs állhat több attribútumból is (*többkulcsos indexek*), az összes lehetséges lekérdezést támogató indexek száma az attribútumok függvényében rendkívül gyorsan nő. Belátható, hogy n attribútum esetén a szükséges indexek száma n elem $\frac{n}{2}$-ed osztályú ismétlés nélküli kombinációja, vagyis:

$$\mathbf{C}_n^{\left[\frac{n}{2}\right]} \equiv \binom{n}{\left[\frac{n}{2}\right]} \equiv \frac{n!}{(n-\left[\frac{n}{2}\right])!\,\left[\frac{n}{2}\right]!}. \tag{4.1}$$

Ennyi index szükséges ahhoz, hogy minden lehetséges k-elemű attribútumhalmazhoz találjunk olyan indexet, amelynek első k attribútuma megegyezik az attribútumhalmazzal[1] [Timoshenko]. Ez már 10 attribútum esetén is $\binom{10}{5} = 252$ indexet jelent, 20 attribútum esetén pedig $\binom{20}{10} = 184756$ indexre van szükség, amelyek karbantartása komoly adminisztrációs terhet ró a rendszerre, tárolásuk pedig nagy helyet igényel. Ha tehát a felhasználók olyan lekérdezéseket futtatnak, amelyek sok oszlopra tartalmaznak szelekciót, nem tudjuk egydimenziós indexekkel hatékonyan támogatni azokat.

A fentiek miatt a *többdimenziós lekérdezések* támogatására külön indextípusokat alkalmaznak. A hash szervezés ötletét használja fel a partícionált hash szervezés, míg a *kd*-fák (k dimenziós keresőfák) és az R-fák (*régiófák*) a B-fák általánosítása [Garcia, Ullman, Widom, 2001]. Szintén általánosan elterjedt és hatékony megoldás a később részletesen tárgyalt bittérkép indexek alkalmazása.

4.3. Bittérkép indexek

A *bittérkép indexek* (bitmap index) először a Nucleus cég adatbázis-kezelő rendszerében jelentek meg. Bár a cég csődbe ment, az ötlet rohamosan terjedt, manapság a nagy adatbázis-kezelő rendszerek (pl. Oracle Database, Sybase IQ, IBM Informix) többsége támogatja ezt az indexelési módot [Wu, Stockinger, Shoshani, 2008]. A bittérkép indexekhez köthető első publikáció 1987-ben jelent meg, szerzője Patrick E. O'Neil. A bittérkép indexek témája azóta is intenzíven kutatott terület.

A bittérképek használata olyan adathalmaz esetén célszerű, amelybe új adatok csak úgy kerülhetnek, ha a korábbiakat változatlanul hagyva szúrjuk be azokat (ún. read only, append only adathalmazok). Ilyenek például a tranzakciós rendszerekből adattárházba kötegelt módon áttöltött, valamint a tudományos kísérletek során keletkező adathalmazok.[2]

A fa alapú indexektől eltérő megközelítést alkalmaznak a *bittérkép indexek* (bitmap index) [Gajdos, 2011]. A bittérkép indexek olyan bitvektorok halmazai, amelyek meghatározzák, hogy az adott értéket az adattábla mely sorain veszi fel. Pontosabban egy n sorból álló adattábla esetén az A attribútumhoz tartozó bittérkép index n hosszú bitvektorok olyan halmaza, amelyben minden A-n felvett értékhez tartozik egy bitvektor. Adott v értékhez tartozó bitvektor az i. helyen 1-et vesz fel, ha az i. rekordban az attribútum értéke v, különben nullát [Garcia, Ullman, Widom, 2001].

Hagyományosan bittérkép indexeket alacsony kardinalitású attribútumokra építenek. Például személyek tárolása esetén tipikusan ilyen attribútumok: a személy neme, családi állapot, régió, végzettség típusa stb. A bittérkép indexek egyik legfontosabb előnye, hogy segítségükkel gyorsan megkaphatók a lekérdezés eredményrekordjainak indexei. Ha egy lekérdezés több bittérkép indexet is használ, a kiértékelés során azok logikai műveletekkel

[1] Pl. $n = 3$ esetén $\binom{3}{2} = 3$, egy jó indexhalmaz a $\{(3,2,1),(1,3),(2,1)\}$. Ekkor a $(3,2,1)$ indexhalmaz felhasználható a $(3,2,1)$, $(3,2)$, (3), $(1,3)$, (1), $(2,1)$, (2) attribútumhalmazokon végzett szelekciók hatékony elvégzéséhez, vagyis az összes lehetséges ($2^3 - 1 = 7$-féle) attribútumhalmaz lekérdezése gyorsítható ezzel a halmazzal.

[2] Az európai CERN kutatóintézet hadronütköztetőjéhez, valamint az amerikai Brookhaven National Laboratory ionütköztetőjéhez kapcsolt adatbázisok is használnak bittérkép indexeket [Duellman, 2007] [Wu, Otoo, Shoshani, 2005]. Lásd még 4.6.3.

név	**nem**	**családi állapot**	**lakhely típus**	**lakóhely**	**fizetés**
Aladár	férfi	özvegy	családi ház	vidék	3243
Brigitta	nő	házas	panel	főváros	2231
Cecília	nő	házas	panel	vidék	4992
Dezső	férfi	egyedülálló	sorház	vidék	2714
Erzsébet	nő	elvált	családi ház	vidék	2265
Frigyes	férfi	egyedülálló	sorház	főváros	2188

4.1. táblázat. *Sok alacsony kardinalitású attribútumot tartalmazó tábla*

név	**nem**		**családi állapot**					**lak. típ.**			**lakóh.**		**fizetés**
	férfi	nő	házas	elvált	egyedülálló	özvegy	élettársi	családi ház	sorház	panel	főváros	vidék	
Aladár	1	0	0	0	0	1	0	1	0	0	0	1	3243
Brigitta	0	1	1	0	0	0	0	0	0	1	1	0	2231
Cecília	0	1	1	0	0	0	0	0	0	1	0	1	4992
Dezső	1	0	0	0	1	0	0	0	1	0	0	1	2714
Erzsébet	0	1	0	1	0	0	0	1	0	0	0	1	2265
Frigyes	1	0	0	0	1	0	0	0	1	0	1	0	2188

4.2. táblázat. *Az előző tábla alacsony kardinalitású oszlopaihoz tartozó bittérképek*

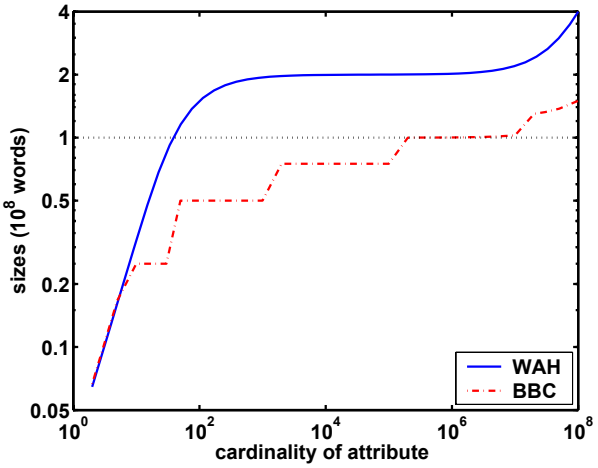
(és, vagy, kizáró vagy, ekvivalencia stb.) összeművelhetők.

Például adott a 4.1. táblában egy SZEMÉLY sémára illeszkedő reláció. A nem, családi állapot, lakhely típusa, lakóhely attribútumok alacsony kardinalitásuk miatt kiváló jelöltek bittérkép index készítésére, a kapott reláció a 4.2. táblázatban látható.

Látható, hogy a kapott bittérkép segítségével hatékonyan megválaszolhatók olyan kérdések, mint például „kik a vidéken élő, egyedülálló férfiak.” Az eredményhalmaz kiszámításához egyszerűen vesszük a megfelelő attribútumokhoz tartozó bittérképeket és logikai *és* művelettel összeműveljük őket (4.3. táblázat). A kapott eredmény alapján látható, hogy csak a 4. rekordban található személy, Dezső felel meg a megadott kritériumoknak. A lekérdezések során gyakran van szükség más logikai műveletek felhasználására is (pl. a *vagy* művelettel egyszerűen képezhetjük az eredményhalmazok unióját).

```
  100101
  101110
& 000101
--------
  000100
```

4.3. táblázat. *Logikai „és” művelet bittérképeken*



4.1. ábra. *Bittérkép indexek mérete a kardinalitás függvényében (forrás: [Stockinger, Wu, Shoshani, 2002]). A BBC és a WAH elterjedt bittérkép tömörítési algoritmusok.*

4.4. A bittérkép indexek előnyei és hátrányai

Elterjedt vélekedés, hogy bittérkép indexek használata csak alacsony kardinalitású attribútumok esetén célszerű. Ezzel szemben a bittérképek sok egyéb előnyös tulajdonsággal is rendelkeznek [Oracle, 2005]:

- A bittérkép lehetővé teszik a logikai műveletet tartalmazó lekérdezések hatékony futtatását.
- Relációs adatbázisokban a bittérkép indexek lehetővé teszik a *NULL* értéket tartalmazó sorok megszámolását, míg a fa alapú indexek általában nem indexelik a keresési kulcson *NULL*-t tartalmazó sorokat.
- n attribútum esetén elég n bittérkép indexet definiálni ahhoz, hogy többdimenziós lekérdezéseket futtathassunk az attribútumok tetszőleges halmazán.

A bittérkép indexek jellemző hátrányai:

- Új sorok beszúrása esetén bővítésük nehézkes.
- Magas kardinalitás esetén tárolásuk költséges lehet.

Utóbbi pont nem jelent áthidalhatatlan problémát, ugyanis megfelelő tömörítési eljárás alkalmazásával a bittérkép indexek teljes mérete – tehát egy adott attribútumhoz tartozó összes bittérkép méretének összege – magas kardinalitás esetén sem lesz kezelhetetlenül nagy (4.1. ábra). Ha ugyanis az attribútum kardinalitása nagyon magas, az egyes bittérképek ritkák lesznek, ezért kevés szóval kódolhatók, a bittérkép mérete tehát csak lassan nő.

4.5. Bittérkép reprezentációs algoritmusok

4.5.1. Literal reprezentáció

A *literal* (angolul „szó szerinti") algoritmus a bittérkép minden bitjéhez a memória egy bitjét rendeli hozzá. A literal reprezentáció tehát nem tömörít a bittérképen. Éppen ezért a tárolás és bitműveletek algoritmusának implementálása nagyon egyszerű. Literal reprezentációt kevés számú, kis méretű bittérkép esetén, ill. gyors prototipizáláshoz célszerű használni.

4.5.2. Általános célú tömörítő algoritmusok

Sok hatékony tömörítés biztosító általános célú tömörítőalgoritmus létezik, például Lempel és Ziv algoritmusának különféle változatai (LZ77, LZ78, LZW) és a ZIP/gzip tömörítés során alkalmazott, az LZ77 és a Huffmann kombinációját alkalmazó Deflate-algoritmus. Bár az algoritmusok módosítás nélkül használhatók lennének bittérképek tömörítésére is, mégsem használják őket, ugyanis fontos, hogy a bittérképek *kitömörítés nélkül* hatékonyan összeműveljenek.

4.5.3. Byte-aligned Bitmap Compression

A Byte-aligned Bitmap Compression algoritmust Gennady Antoshenkov publikálta 1994-ben. A BBC-vel tömörített bittérképek általában több helyet foglalnak az általános célú algoritmussal tömörítetteknél, de gyorsabban végezhetők rajtuk műveletek [Johnson, 1999]. A BBC az Oracle Corporation szabadalma.[3]

4.5.4. Word-Aligned Hybrid (WAH)

A WAH tömörítés további kompromisszumot hoz a tárhely–időkomplexitás mentén: a WAH algoritmussal tömörített bittérképek általában nagyobbak a BBC-vel tömörítetteknél, ugyanakkor gyorsabb műveletvégzést biztosítanak [Stockinger, Wu, Shoshani, 2002]. A BBC-vel ellentétben a WAH tömörítés bájtok helyett szavakat használ, mert ez jobban illeszkedik a modern számítógépek architektúrájához.

A WAH tömörítés során kétféle szót használ. A *fill* (kitöltő) szavak hosszú, több szón keresztül tartó homogén 0, ill. 1 sorozatokat írnak le, míg a *literál* (literal, szó szerinti) szavak egy-egy szó bitjeit tartalmazzák módosítás nélkül. A szavak típusát a legfelső bitjük (most significant bit, MSB) alapján állapítjuk meg. A fill szavak MSB-je 1, míg a literáloké 0. A fill szavak második bitje 1, ha az adott sorozat csupa 1-esből; 0, ha csupa 0-ból áll. Az egyes típusok fejlécét a 4.4. táblázat tartalmazza.

A WAH eljárás rendkívül egyszerű: a hosszú, csupa 0-t tartalmazó sorozatokat 0-fillre, a csupa 1-et tartozókat 1-fillre cseréljük. A hosszú sorozatok mérete mindig az egy literálban tárolható bitek számának többszöröse, azaz 32 bites szavak esetén 31, 64 bites szavak esetén 63 többszöröse lehet. A fillben a fill típusát és az annak megfelelő szavak számát tároljuk.

[3] `http://www.google.com/patents/US6205442`

literál	`0.......`
0-fill	`10......`
1-fill	`11......`

4.4. táblázat. *A WAH algoritmussal szótípusainak fejléce*

128 biten	`1*1,20*0,3*1,79*0,25*1`			
31-bites csop.	`1*1,20*0,3*1,7*0`	`62*0`	`10*0,21*1`	`4*1`
tömörítetlen	40000380	00000000 00000000	001FFFFF	0000000F
WAH	40000380	80000002	001FFFFF	0000000F

4.5. táblázat. *WAH algoritmussal tömörített bittérkép*

A ma elterjedt processzorokon az egyik legköltségesebb művelet a *feltételes elágazó utasítások* (conditional branching instructions, cbi) végrehajtása, ezért az implementáció során tördekedtem arra, hogy ezek számát alacsonyan tartsam.

4.5.5. Position List Word-Aligned Hybrid (PLWAH)

A PLWAH tömörítési eljárással kapcsolatos első publikáció 2010-ben jelent meg [Deliège, Pedersen, 2010]. A PLWAH tömörítés ötlete az, hogy WAH tömörítéssel még 32 bites szavak esetén is csak ritkán használjuk ki a fill szavak felső bitjeit. Ezért célszerű lehet ide egy *pozíciólistát* (position list, PL) elhelyezni, amelyben elfér az a „néhány" bit, ami eltér a sorozat többi elemétől. Nyilvánvaló, hogy w hosszú szavak esetén egy bit pozíciójának tárolásához

$$\lceil \log_2 (w-1) \rceil \tag{4.2}$$

bit szükséges, tehát 32 bites szavak esetén 5, míg 64 bites szavak esetén 6 hosszú egy pozíció a listában.

Az eredeti tömörítési eljárás négy lépésből áll, melyeket egy példán keresztül mutatok be. Egy 175 bit hosszú bittérképet kívánunk tömöríteni 32 bites architektúrán, tehát a szóhossz $w = 32$.

1. A bittérképet $w-1$ hosszú csoportokra tördeljük. $175 = 5 \times 31 + 20$, így öt 31 bit és egy 20 bit hosszú csoportot kapunk. Utóbbit kiegészítjük 11 nullával, hogy a többivel egységesen kezelhessük, így hat 31 bites csoportot kapunk.

2. Azokat a csoportokat, amelyekben csak nullák vagy csak egyesek szerepelnek, *homogén csoportnak* (homogeneous group) nevezzük. Ezek a csoportok *egyesíthetők*

0 0000000000 0000000000 000000000 0		
0 0000000000 0000000010 000000000 0		
0 0000000000 0000000000 000000000 0	→	1 0 1010000000 0000000000 0000000001
0 0000000000 0000000000 000000000 0		1 0 0100000000 0000000000 0000000010
0 0000001000 0000000000 000000000 0		0 0 0000000000 0000001000 0000000000
0 0000000000 0000001000 000000000 0		

4.6. táblázat. *PLWAH32 tömörítés (forrás: [Deliège, Pedersen, 2010])*

literál	0<literal.....>
0-fill	10<p_2><p_1>...
1-fill	11<p_2><p_1>...

4.7. táblázat. *PLWAH32 szavak fejlécei*

(merge). Példánkban az 1., 3. és 4. csoport egyesíthető.

3. A szavakat a legnagyobb helyiértékükön (most significant bit, MSB) kiegészítjük egy bittel. Az egyessel kezdődő szavak az ún. fill, a nullával kezdődök a literal szavak. A fill szavak a 2. legmagasabb helyiértéken álló bitjük alapján lehetnek 0-fillek és 1-fillek. Az egyesíthető csoportokból fill szavakat képzünk.
   Példánkban az 1., 3., és 4., szavak fillek, 2., 5., 6. szavak literálok.

4. Meghatározzuk azokat a literálokat, melyek közvetlenül egy „hasonló" fill után következnek, és ha lehetséges, egyesítjük őket. Hasonlóság alatt azt értjük, hogy ha az előző szó pozíciólista nélküli 0-fill, akkor az adott szó csupán annyi 1-est tartalmaz, amennyi belefér a pozíciólistájába, ill. fordítva: az előző szó pozíciólista nélküli 1-fill és az aktuális szó csupán néhány 0-t tartalmaz.

A WAH-hoz hasonlóan tehát a PLWAH tömörítésben is *literálokat* és *filleket* használunk, az egyetlen különbség, hogy az utóbbiak felső bitjeiből pozíciólistát alkotunk. Az általában megszokott konvencióval ellentétben a pozíciólistában a biteket 1-től sorszámozzuk. Erre azért van szükség, hogy könnyen ellenőrizhessük, hogy az adott pozíció üres-e – bitmaszkolás után megvizsgáljuk, hogy az adott pozíción van-e 1-es bit a szóban.

A PLWAH tömörítés szavainak fejlécei a 4.7. táblázatban láthatók.

Problémák a PLWAH tömörítéssel

A PLWAH tömörített bittérképek elvégzése nem csak számításigényesebb a WAH-hoz képest, hanem új problémákat is felvet. A pozíciólisták által elfoglalt bitek miatt ugyanis a 4.7. táblázat alapján készült PLWAH fill csak 20 fill bitet tartalmazhat, így mérete legfeljebb $(2^{20} - 1) \times 32505825$ hosszú lehet – ez pedig nagyon sok gyakorlati esetben kevésnek bizonyulhat. A probléma legegyszerűbb megoldása, hogy az ennél hosszabb fill sorozatokat több kisebb fill sorozatra tördeljük, pl. egy 100 millió hosszú 0-fill sor három hosszú 32505825 és egy 2482525 hosszú 0-fillből áll. A probléma természetesen hosszú 1-es sorozatok esetén is fennállhat, ekkor hasonlóan járunk el.

A problémára általánosabb megoldást ad a két szó szélességű *adaptív számlálók* (adaptive counter) létrehozása, bővebben l. [Deliège, Pedersen, 2010].

4.5.6. Bitműveletek tömörített bittérképeken

A bittérkép tömörítő eljárásokkal szemben támasztott egyik legfontosabb követelmény, hogy képesek legyenek a bittérképek tömörített reprezentációban történő összeművelésére. A 4.8. táblázatban látható a A és B WAH32 tömörítésű szavakból bitenkénti „és"

tömörítetlen					
A	40000380	00000000	00000000	001FFFFF	0000000F
B	7FFFFFFF	7FFFFFFF	7C0001E0	3FE00000	00000003
C	40000380	00000000	00000000	00000000	00000003
tömörített					
A	40000380	80000002		001FFFFF	0000000F
B	C0000002		7C0001E0	3FE00000	00000003
C	40000380	80000003			00000003

4.8. táblázat. *Bitenkénti és művelet WAH bittérképeken, $C = A$ és B (forrás: [Wu, Otoo, Shoshani, 2005])*

művelettel kiszámítható C bittérkép előállítása. A logikai műveletek kiszámítása jelentősen felgyorsítható egy egyszerű ötlettel. Például „és” művelet esetén ahol az egyik bittérképben 0-fill található, ott az eredmény biztosan 0-fillt fog tartamazni, ahol az egyik bittérképben 1-fill található, ott az eredmény másik bittérkép aktuális szavának értéke (4.8. táblázat). Tényleges bitenkénti „és” műveletet csak akkor kell végrehajtanunk, ha ugyanabban a pozícióban mindkét bittérképben literál szavak találhatók.

A WAH és a PLWAH tömörített bittérképeken végezhető műveleteketről bővebben [Wu, Otoo, Shoshani, 2005] és [Deliège, Pedersen, 2010] ír.

4.5.7. További tömörítési eljárások

A bittérkép tömörítési eljárások továbbra is élénken kutatott téma. A korábbi eljárások finomhangolása mellett új tömörítési eljárások készültek, pl. a COMPressed Adaptive indeX format (COMPAX) [Fusco, Stoecklin, Vlachos, 2010] és a COmpressed 'N' Composable Integer SEt (CONCISE) [Colantonio, Pietro, 2010].

4.6. A `bitmapper` keretrendszer

A bittérképek tömörítésének teszteléshez és méréséhez elkészítettem a `bitmapper` keretrendszert. A keretrendszer öt különböző implementációt képes használni a bittérkép indexek tárolására. Az implementációkhoz készített interfész – csomagolóosztályokon keresztül[4] – az alábbi műveleteket nyújtja:

`setBit(i)` Az i. bitet 1-esre állítja. i mindig nagyobb, mint a bittérkép aktuális hossza.

`createIterator()` Iterátort készít, amely sorban visszaadja a bittérképben szereplő 1-esek indexét.

4.6.1. Szabványos sablonkönyvtár vektor tárolója

A C++ szabvány fontos része a szabványos sablonkönyvtár (Standard Template Library, STL), amely többek között olyan előre elkészített tárolókat tartalmaz, melyekkel hatékonyan, típushelyesen tárolhatunk adatokat. A könyvtár többféle tárolási stratégiát is

[4] wrapper tervezési minta

támogat: verem, lista, prioritásos sor, vektor stb. Bittérképek tárolására a statikus méretű `bitset` és a dinamikus méretű `vector<bool>` osztályok használhatók. Esetünkben a tárolni kívánt adathalmaz mérete előre nem ismert, ezért erre a feladatra a `bitset` nem alkalmas.

A szabvány rögzíti, hogy a helykihasználás javítása érdekében a `vector<bool>` tárolókat úgy kell megvalósítani, hogy a bittérkép elemei valóban csak egy-egy bitet foglaljanak el. Az egyes bitek egyenként állíthatók be a `vector<bool>::reference` osztályon keresztül[5] [C++ Standard, 2005].

4.6.2. Boost C++ Libraries

A 2. fejezetben is használt Boost C++ Libraries bittérképek tárolására hatékony eszközt nyújt a `dynamic_bitset` formájában.[6] A szabványos sablonkönyvtárban definiált `vector<bool>` osztályhoz képest a `dynamic_bitset` eltérő allokációs stratégiát követ, amivel a struktúra gyorsabban épülhet fel. A `dynamic_bitset` a bitműveletek támogatását is hatékonyabban végzi, a szabványos vektorral ellentétben nem bitenként, hanem szavanként végzi el.

4.6.3. FastBit

A FastBit[7] egy nyílt forráskódú függvénykönyvtár, melyet eredetileg a Brookhaven National Laboratory-ban folyó kísérlet, a STAR[8] (Solenoidal Tracker at RHIC) adatrögzítésének és -feldolgozásának támogatására fejlesztettek ki. A FastBit a WAH tömörítési eljárást implementálja, valamint gazdag függvénykönyvtárával egy oszlopalapú adatbázis-kezelő funkcionalitását is képes nyújtani.

A FastBit telepítési útmutatója megtalálható az F.5.2. szakaszban.

4.6.4. Saját implementációk: `AOWAH32Bitmap` és `AOPL32Bitmap`

Követelmények

A korábban ismertetett WAH és PLWAH algoritmusok – a publikációkban szereplő formában – általános bittérkép tömörítési problémákra kínáltak megoldást. A Colap rendszer működéséhez szükséges bittérkép-tömörítő eljárás több speciális megkötéssel rendelkezett.

1. Az eljárásnak képesnek kell lenni folyamatosan érkező új bitek beszúrására (on-the-fly tömörítés).
2. A bittérképnek csak a végére szúrunk be új egyest.
3. Az előző értelmében ugyanarra a pozícióra nem szúrunk be egyest.
4. A bittérképek jellemzően ritkák, azaz kevés egyest tartalmaznak.

[5] Fontos megjegyezni, hogy a `deque<bool>` típus esetén a szabvány nem rögzíti ezt.
[6] `http://www.boost.org/doc/libs/1_48_0/libs/dynamic_bitset/dynamic_bitset.html`
[7] `https://sdm.lbl.gov/fastbit/`
[8] `http://www.star.bnl.gov/`

`<...>1`	`...1000`
`& <...>0`	`& ...0111`
`<...>0`	`...0000`

4.9. táblázat. *Legalsó bit törlése*

5. Nem szükséges a bittérkép hosszának eltárolása.

Az 1. feltétel miatt a beszúrást a tömörített reprezentáción kell elvégezni. Ez általános esetben meglehetősen bonyolult feladat lenne, szerencsére a 2., 4 és 5. pontok miatt több egyszerűsítést is alkalmazhattam. A bittérképeket 32 bites szavakon tároltam, ezért a típusok neve `AOWAH32Bitmap` (append-only WAH 32 bit) és `AOPL32Bitmap` (append-only PLWAH 32 bit).

Az iterátorok működése

Az objektum-orientált programozásban gyakori *iterátor tervezési mintában* (iterator pattern) egy ún. *iterátorral* lépkedhetünk végig egy tároló elemein. Többek között a C++ szabványos függvénykönyvtár generikus tárolóit – `map`, `vector` stb. – is jellemzően iterátorokon keresztül érjük el. A bittérképindex implementációkhoz olyan iterátorokat készítettem, amelyek sorban visszaadják az 1 értékű bitek indexeit. Például a `00010011` bittérkép esetén – ha a sorszámozást 0-tól kezdjük – az iterátor értékei sorra 3, 6 és 7 lesznek.

Az iterátorokat a FastBit iterátorának mintájára készítettem el. A FastBit iterátorai úgy működnek, hogy az egy szóban található indexeket egy lépésben kiszámítják és tárolják. A megközelítés azért szerencsés, mert az összes index előre kiszámítása esetén tárolásuk költséges lenne, ha pedig egyesével számolnánk ki őket, az iterátor léptetései között meg kell tartanunk az adott szó feldolgozottsági állapotát, ami a lekérdezés idejét növeli. Így az 1-es bitek sorozatának lekérdezéséhez nincs más dolgunk, mint addig hívni ezt a függvényt, amíg az indexek el nem fogynak, és közben folyamatosan feldolgozni a kapott indextömböket.

Szó 1 értékű bitjeinek meghatározása

Egy szó 1 értékű bitjei számának meghatározására több optimalizációs lehetőség is kínálkozik. A problémát megoldó naiv algoritmus elve, hogy megvizsgáljuk a legalsó bitet, ha ez 1, növeljük a számlálót, majd jobbra léptetjük a szót, amíg az tartalmaz egyeseket. Ekkor a legrosszabb eset futásideje a szó hosszával lesz arányos.

Egy egyszerű ötlettel elérhető, hogy a futásidő az 1-es bitek számával legyen arányos (a 4.1. felsorolás). Az eljárás ötlete, hogy egy szót bitenként *és* műveletbe hozva a nála 1-gyel kisebb értékű szóval a szó legalsó 1 értékű bitje törlődik. A 4.9. táblázatban látható, hogy hogyan törlődik a legalsó 1-es bit a művelet során. A bal oldali táblázatban a legalsó 1-es bit egyben a szó legalsó bitje, míg a jobb oldali általánosabb esetet mutat. A művelet során a `<...>` szimbólummal jelzett bitek értéke nem változik, mert a kivonás során ezek nem változnak meg, és a bitenkénti *és* idempotens művelet (`A & A == A`).

```
unsigned int v; // a v szo bitjeinek szamat keressuk
unsigned int c; // a bitek szama a c valtozoba kerul
for (c = 0; v; c++)
{
  v &= v - 1;
}
```

4.1. lista. *Bitek számlálása (forrás: [Anderson])*

Az eljárást általában Brian Kernighannek tulajdonítják, mert Kernighan és Dennis Ritchie C programozási nyelvről szóló népszerű könyvének 1988-ban megjelent második kiadásában szerepelt. Valójában az eljárást már korábban is felfedezték, többek között Peter Wegner 1960-ban és tőle függetlenül Derrick Lehmer 1964-ban. A problémára természetesen számos egyéb hatékony megoldás is létezik [Anderson].

4.6.5. `AOWAH32Bitmap`

Az `AOWAH32Bitmap` az „append only WAH 32 bit” eljárás rövidítése. Az AOWAH32 a WAH tömörítési eljárásnak megfelelően végzi a bitek tömörítését és a biteken történő iterálást.

Az F.6.1. szakaszban található az 1. algoritmus új bit beszúrását, a 2. algoritmus a következő szó kiolvasását mutatja be részletesen.

4.6.6. `AOPL32Bitmap`

Az `AOPL32Bitmap` név az „append only PLWAH 32 bit” rövidítése. Az AOPL32 olyan 32 hosszú PLWAH szavakat használ, melyek pozíciólistája 2 eltérő bit megcímzésére is alkalmas, így az AOPL fill szavainak felépítése: 2 bit fejléc + 10 bit pozíciólista + 20 bit számláló. Az F.6.2. szakaszban található a 3. algoritmus új bit beszúrását, a 6. algoritmus a következő szó kiolvasását mutatja be részletesen.

4.6.7. További optimalizációk

A bittérképek tömörítése jelentősen javítható, ha a tömörítést rendezetett adathalmazon végezzük [Lemire, 2009] [Lemire, Kaser, 2011]. A kiugróan magas kardinalitások kezelésére is több ötlet született. Egy jó kompromisszum például az értékek tartományokra bontása és annak megfelelő *ládákban* (bin) tárolása és a ládák indexelése [Wu, Stockinger, Shoshani, 2008]. Ezekkel a módszerekkel dolgozatomban nem foglalkozom.

4.7. Összefoglalás

Ismertettem a többdimenziós indexelés problémáit és az arra adott egyik lehetséges megoldást, a bittérkép indexet. Bemutattam a bittérkép indexek tömörítésének okait és algoritmusait, és elkészítettem a saját implementációmat tömörítetlen reprezentációkra és két modern tömörítési megközelítésre.

5. fejezet

Mérési eredmények

A fejezetben ismertetem a memóriahasználat mérésének tipikus problémáit. Bemutatom a bevált megoldások korlátait, majd kiválasztom a feladatra legmegfelelőbbet. Ennek felhasználásával elvégzem a mérést, és elemzem a kapott mérési eredményeket.

5.1. Mérési módszerek

A Linux és a többi Unix-szerű operációs rendszer gazdag beépített eszköztárral rendelkezik az egyes folyamatok memóriafogyasztásának mérésére. A *Unix-szerű* (Unix-like) operációs rendszerek közös jellemzője, hogy hasonlóak a Unix rendszerhez, de nem feltétlenül felelnek meg a Unix specifikációnak (Single UNIX Specification).

5.1.1. `ps`, `/proc`, `(h)top`

A `ps` (process state) parancssoros eszközzel pillanatképet kaphatunk az éppen futtatott folyamatokról. A memóriahasználat szempontjából fontos értékek a teljes felhasznált virtuális memória mennyiségét mutató VSZ (virtual set size) és a fizikai memóriában tárolt memória mennyiségét jelző RSS (resident set size).

A `ps` eszközzel kapható információk a `/proc` könyvtárban a folyamat azonosítójának (process ID, pid) megfelelő könyvtárban található szöveges állományokból is kinyerhetők, valamint megfigyelhetők grafikus (System Monitor) és parancssoros (top, htop) rendszermonitorozó alkalmazásokkal. Az 5.1. felsorolásban megfigyelhető a `bitmapper` eszköz memóriafoglalásának a felsorolt eszközökkel történő lekérdezése. Látható, hogy az eszközök konzisztens értékeket adnak, csupán azok mértékegysége tér el (423624 kB = 413,7 MB, 586380 kB = 572,6 MB).

A Linux programok jelentős része *megosztott könyvtárakat* (shared libraries) használ. Bár az operációs rendszer a memóriába csak egy példányt tölt be, a könyvtárat egyszerre több folyamat is használhatja, a foglalt memória mértéke pedig minden folyamatban megjelenik. Nem triviális probléma a megosztott memóriaterületek vizsgálata, vagyis az, hogy az adott és más alkalmazások által is használt memória méretét melyik alkalmazáshoz számítjuk (mindegyikhez, egyikhez sem, ahhoz, amelyik írta stb.). Az operációs rendszer alapértelmezés szerint több memóriát foglal le a kelleténél (overcommit), ezért a kapott

5.1. ábra. *A Valgrind logója*

érték valószínűleg több lesz a program által valóban igényeltnél. Önmagában a fenti eszközök egyikével sem követhető a memóriafoglalás időbeli változása. A felsoroltak miatt a beépített eszközök nem megfelelőek a memóriafelhasználás precíz méréséhez.

```
$ ps -C bitmapper -o pid,rss,vsz
  PID   RSS    VSZ
19778 423624 586380
$ top | grep bitmapper
19778 gszarnya  20   0  572m 413m 2528 R  101 11.0
1:46.80 bitmapper
$ cat /proc/19778/status | grep VmRSS
VmRSS:    423624 kB
$ cat /proc/19778/status | grep VmSize
VmSize:   586380 kB
```

5.1. lista. *Folyamat vizsgálata parancssoros eszközökkel*

5.1.2. Valgrind

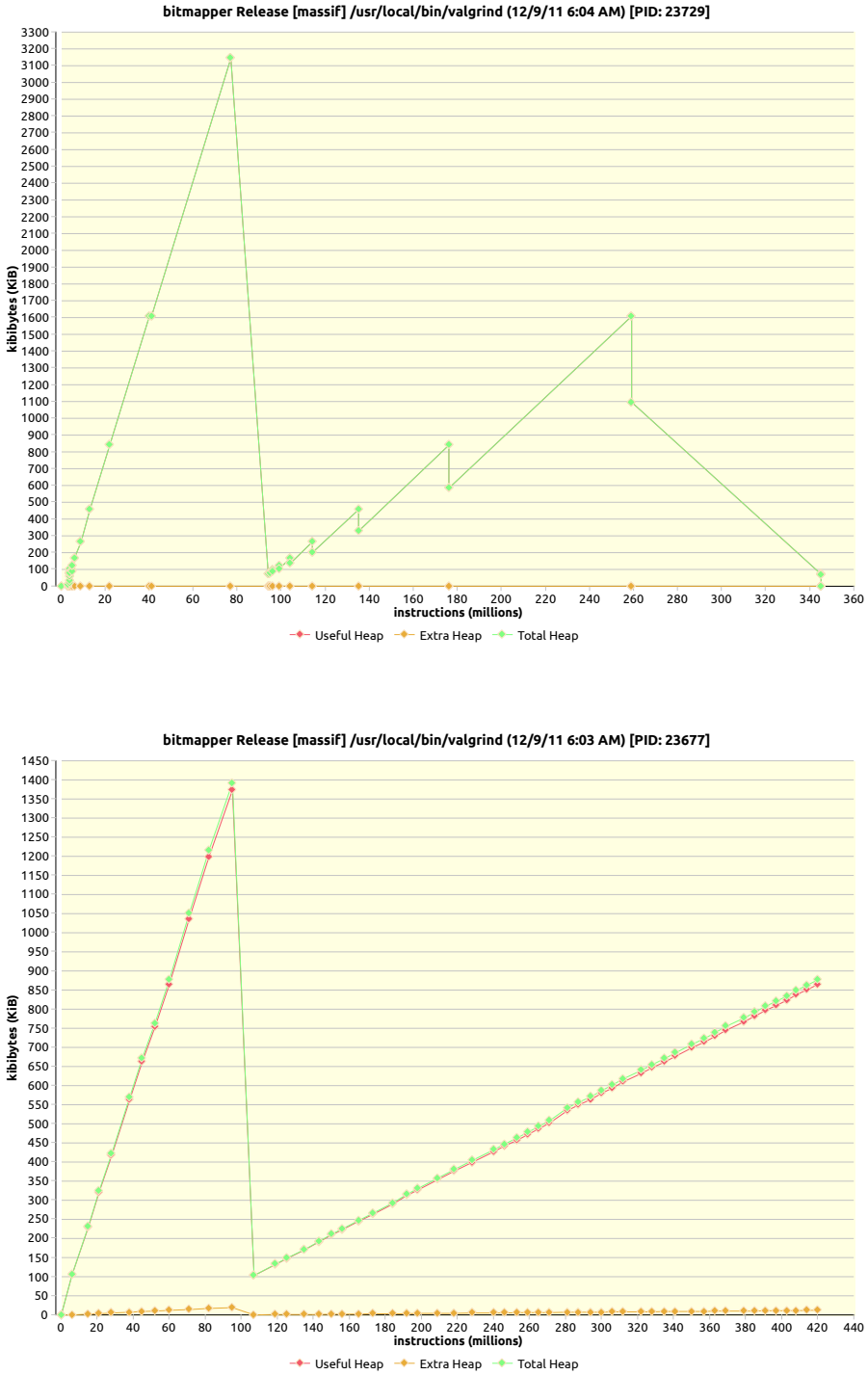
A Valgrind keretrendszer[1] (5.1. ábra) lehetővé teszi programok futásidejű elemzését. A Valgrind több eszközt is tartalmaz, a legnépszerűbb a hibás memóriakezelés és memóriaszivárgások felderítésében segítséget nyújtó Memcheck (memory error detector) , míg a Massif (heap profiler) a dinamikus memóriafoglalás vizsgálatát teszi lehetővé. A Valgrind működési elve, hogy az alkalmazást egy virtuális processzoron futtatja, így nem csak a lefoglalt memóriaterületekről kap pontos képet, hanem a program által kiadott processzor utasítások számát is rögzíti.

A `valgrind` parancssoros eszközhöz elérhető egy Eclipse plug-in, amellyel a kapott mérési eredmények vizualizálhatók és exportálhatók. A mérések során a Massif eszköz segítségével kapott adatokat használtam fel.

5.2. STL tároló kiválasztása

A memóriafelhasználás és így a mérések szempontjából is fontos a tömörített szavakat tároló struktúra kiválasztása. A C++ szabványos függvénykönyvtár tárolói közül az `std::vector` és az `std::deque` alkalmasak egyre növekvő adathalmaz hatékony tárolásá-

[1] `http://www.valgrind.org/`



5.2. ábra. *Mérési eredmények* `std::vector`*-ral (fenn) és* `std::deque`*-val (lenn) megvalósított AOWAH32 és AOPL32 bittérképeken*

ra. Mindkét tároló hasonló interfészt nyújt, de a belső megvalósításuk különböző: míg a `vector` tárolási módja hasonlít egy tömbre, a `deque` az adatot nem feltétlenül összefüggően tárolja. Az AOWAH32 és az AOPL32 implementációkat mindkét tárolóval elkészítettem, a kapott eredmények az 5.2. ábrán láthatók. A mérés igazolta, hogy azonos adatmennyiség esetén a `deque` tárolók valóban hatékonyabban használják ki a rendelkezésre álló memóriát – körülbelül fele akkora memóriaterületetet igényelnek –, kb. 20–25%-kal hosszabb betöltési időért cserébe. A későbbi mérések során a `deque` tárolóval megvalósított implementációkat használtam.

5.3. Adathalmazok

5.3.1. Ritka bittérkép

Generáltam egy olyan bittérképet, amelynek pontosan minden 100. bitje 1 értékű, így nagyon jól tömöríthető, továbbá jól szemlélteti a WAH és a PLWAH kódolási sémák közötti különbségeket. A bittérkép 400 millió bájt hosszú és 4 millió egyest tartalmaz.

5.3.2. Véletlen generált adathalmaz

A mérések megismételhetősége érdekében a pszeudovéletlen generátort mindig ugyanazzal a kiindulóértékkel (random seed) inicializáltam. Az előállított véletlenszámsorozatban a bitek közötti távolság az $1, 2, \ldots, 100$ számhalmazból kerül ki egyenletes eloszlás szerint. A bittérkép 6 millió darab egyes bitet tartalmaz.

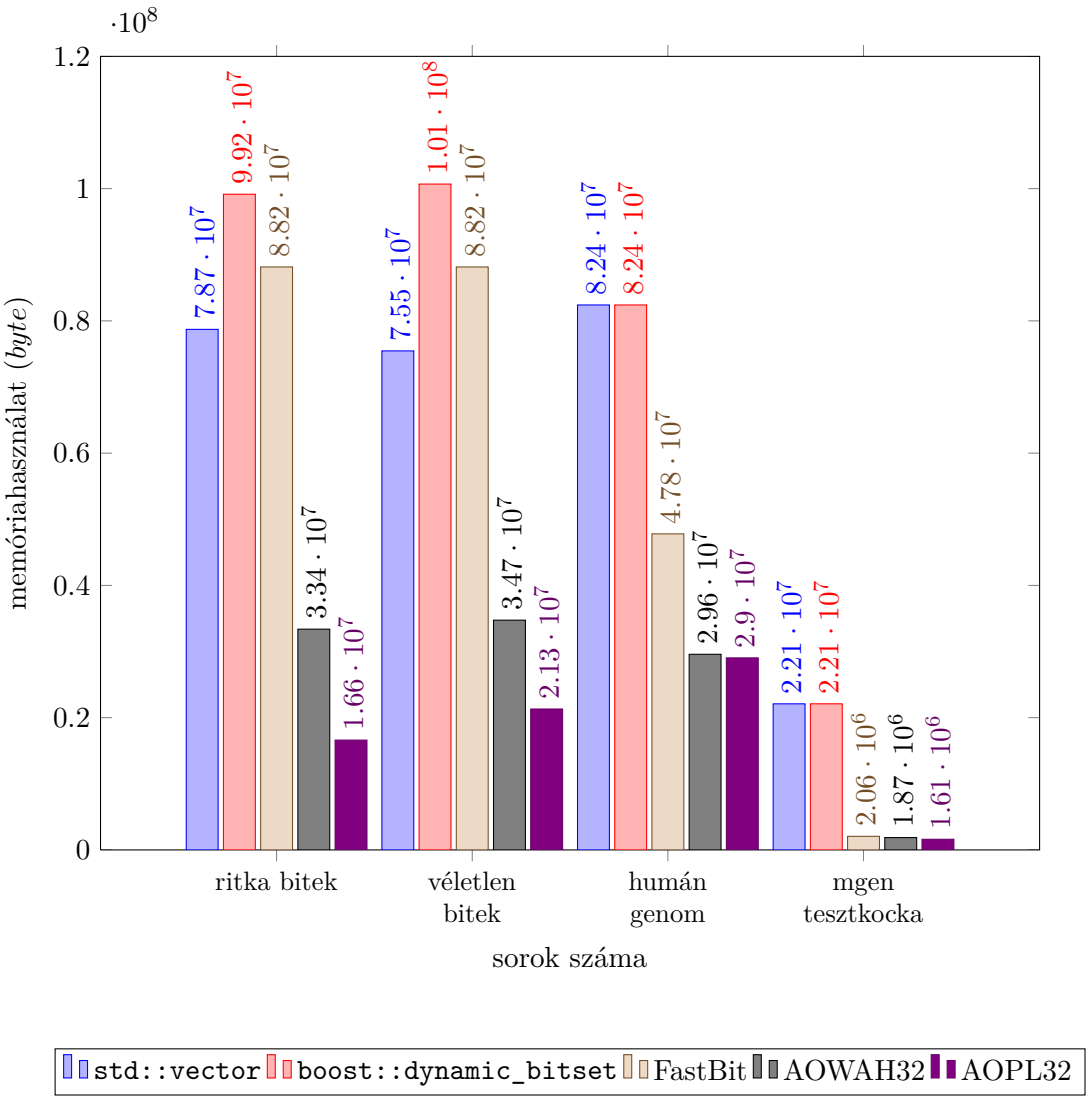
5.3.3. Humán Genom Projekt adathalmaza

A mesterségesen előállított adatok mellett gyakran tesztelnek természetben előforduló adathalmazokra, ezek ugyanis sok szempontból véletlenszerűbbek a pszeudovéletlen generátorokból kapottaknál. Az egyik tesztállományom a Humán Genom Projektben meghatározott, az egyes kromoszómákhoz tartozó ismert nukleotidszekvenciákat tartalmazza. Ezt kedvező tulajdonságai miatt gyakran használják nagy adathalmazokat feldolgozó algoritmusok – nem csak adatbázisok, hanem például *névutó fa* (suffix tree) építés – méréséhez [Mansour et al., 2011]. A géntérképet tartalmazó állomány szabadon letölthető.[2] A mérésekhez a géntérkép állomány utolsó 1 millió sorából képzett 49 megabájtos állományt használtam fel.

5.3.4. `mgen`-nel generált adathalmaz

A `bitmapper` képes az `mgen` által generált fájlokat beolvasásni és bittérkép indexeket építeni az adathalmaz összes attribútumára.

[2] `http://webhome.cs.uvic.ca/~thomo/HG18.fasta.tar.gz`



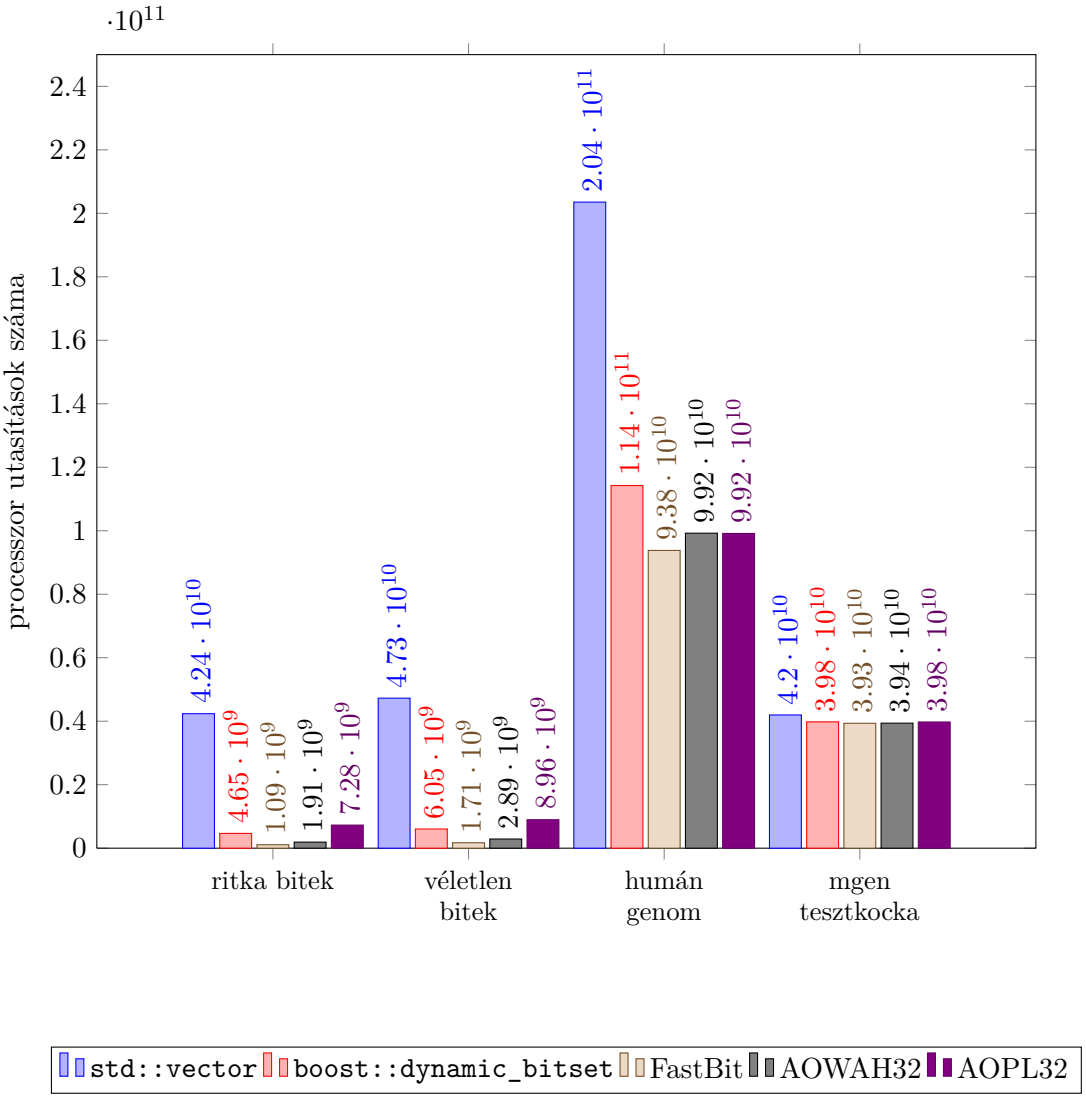
5.3. ábra. *Bittérképek memóriafelhasználása*

5.4. Mérési eredmények

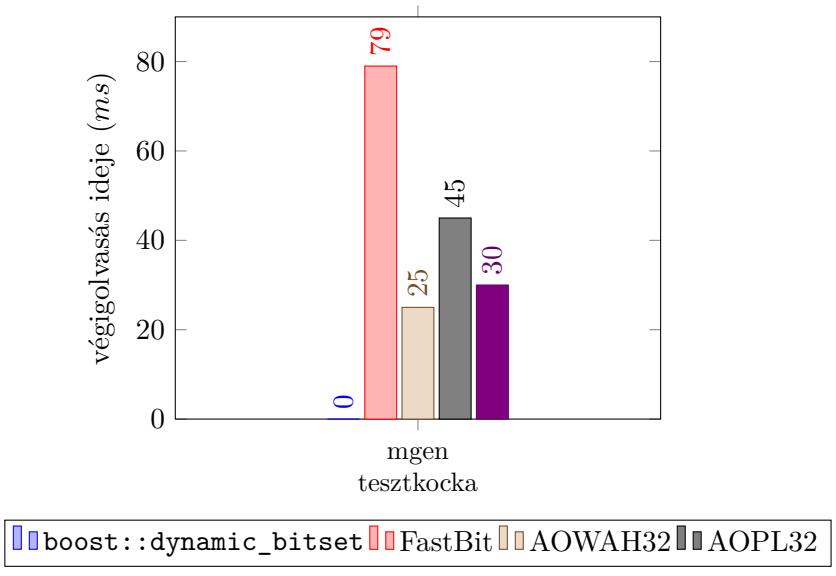
A méréseket Eclipse CDT-ben végeztem Vagrind Massif profilozóval. A profilozóból kapott szövegállomány `mem_heap_B` és `time` értékeinek maximumát, ezek jelzik ugyanis a program által felhasznált memória maximumát és a program futási idejét.

Az 5.3. ábrán az egyes tömörítési eljárások memóriaigénye, az 5.4. ábrán az egyes tömörítési eljárások betöltési ideje látható.

Az 5.5. ábrán a már felépített bittérképek végigolvasásának ideje látható, vagyis az, hogy mennyi ideig tart végigiterálni az 1-es biteken. A diagramon nem ábrázoltam az `std::vector`-hoz tartozó 10091 ms-es értéket, ez ugyanis olyan magas volt, hogy értéke nem összehasonlítható a többivel.



5.4. ábra. *Bittérképek betöltési ideje*



5.5. ábra. *Bittérképek végigolvasásának ideje*

5.5. A mérési eredmények értelmezése

A tömörítési algoritmusokon végzett mérések nem okoztak komoly meglepetéseket, inkább csak néhány érdekességre mutattak rá. Jól megfigyelhető az az alapelv, hogy a gyorsan futó algoritmusok – FastBit, AOWAH32 – általában gyengébb tömörítést biztosítanak.

A tömörítetlen implementációk közül a `boost::dynamic_bitset` eltérő allokációs stratégiája miatt jelentősen gyorsabb az `std::vector`-nál, cserébe gyakran több memóriát foglal a kelleténél. A FastBit nagyon gyorsan szolgálta ki a lekérdezéseket, ugyanakkor a PLWAH algoritmusnál gyakran több memóriát foglalt.

Szintén fontos megfigyelés, hogy a rendezetetlen, többségében egyenletes eloszlást tartalmazó tesztkocka szinte alig tömöríthető. Az adathalmaz rendezettségének és tömöríthetőségének kapcsolata intenzíven kutatott terület [Lemire, 2009] [Lemire, Kaser, 2011], a konkrét implementációkra gyakorolt hatásának vizsgálata további méréseket igényel.

5.6. Összefoglalás

A tesztelés és a mérés igazolta, hogy az elkészített tömörítési algoritmusok megfelelően működnek, az egyes eljárások a várt paramétereket nyújtják a tárhelyigény és futási idő szempontjából. A FastBit kiforrott technológiája nagyon gyorsan fut, míg a lassabb AOPL32-vel nagyon erős tömörítés érhető el.

6. fejezet

Összefoglalás

Minden igazságot szabályként használva fel más igazságok felfedezésére nemcsak számos problémát oldottam meg, amelyeket azelőtt igen nehezeknek tartottam, hanem a vége felé úgy tetszett, hogy még azokra vonatkozóan is, amelyeket nem oldottam meg, meg tudom határozni, mi módon és milyen pontig oldhatók meg.

(René Descartes)

Dolgozatomban bemutattam a többdimenziós adatmodellezés problémáit és fogalmait. Készítettem egy olyan parancssoros eszköz, amellyel egyszerűen generálhatók tetszőleges méretű többdimenziós adathalmazok. Az így elkészített adathalmazokból különböző multidimenziós OLAP szoftverek használatával bemutattam egy-egy adatkocka elkészítésének és lekérdezésének lépéseit, valamint méréseket készítettem eltérő méretű adatkockák betöltési idejéről.

Áttekintettem a többdimenziós lekérdezések problémáját és egy gyakran alkamazott megoldást, a bittérkép indexek készítését. Ismertettem a bittérkép tömörítési eljárások friss kutatási eredményeit, majd implementáltam két eljárás, a WAH és a PLWAH változatait. Megismertem a memóriafelhasználás operációs rendszeren keresztül történő mérésének problémáit, kiválasztottam a mérésre alkalmas szoftvert és kialakítottam egy könnyen kezelhető, precíz mérési környezetet.

Az implementációk alapos tesztelése után különféle adathalmazokkal méréseket végeztem. Összehasonlítottam a saját és mások által késztített tömörítetlen és tömörített reprezentációkat. A mérés során kapott eredmények megerősítettek abban, hogy speciális igényekre valóban érdemes lehet specifikus algoritmusokat fejleszteni, amelyek így nagyon hatékonyan oldhatják meg a feladatot.

A téma több folytatási lehetőséget is kínál. Az elkészült `mgen` eszköz bővíthető további funkciókkal, például újabb eloszlások generálásával, generált adathalmaz rendezésével változatos kulcsok szerint, csomósodó adatértékek előállításával. A `bitmapper` keretrendszer további fejlesztése lehetővé tenné a bittérkép tömörítési algoritmusok automatikus

tesztelését, memória- és időigényének pontos meghatározását.

A megvalósított egyedi algoritmusok további optimalizáció, valamint további funkciókkal történő kiegészítés (pl. bitenkéni művelet elvégzése) után integrálhatók a Colap alkalmazásba és összehasonlíthatók egymással valamint a korábban használt bittérkép reprezentációk paramétereivel.

Összefoglalva, a dolgozat során elmélyedtem a többdimenziós adatkezelésben, megismertem és megvalósítottam ezt támogató algoritmusokat, amelyek a mérések alapján rendkívül hatékony tömörítést biztosítanak vertikális skálázódás mellett.

7. fejezet

Kitekintés

Az adathalmazok növekedésének üteme feltehetően a közeljövőben sem lassul, ezért továbbra is szükséges új, innovatív technológiák fejlesztése, amelyek képesek a hatalmas adatmennyiségek kezelésére. Ebben várhatóan kiemelt szerep jut az elosztott rendszereknek.

A többmagos processzorok egyre nagyobb térnyerésével komoly kihívást jelent a szekvenciális algoritmusok párhuzamosítása, ill. az algoritmusok elosztottan történő futtatása. Bár a szűk keresztmetszetet általában a processzor sebessége jelenti, a dolgozatban implementált algoritmusok nem skálázódnak jól ilyen szempontból: csak egy számítógépen, egy processzormag használatával képesek működni. Megfelelő horizontális skálázódás megvalósításával potenciálisan jelentős teljesítménynövekedés lenne elérhető.

A Gartner technológiai tanácsadó cég 2011. augusztusában felvette a nagy adathalmazok kezelését és feldolgozását takaró „Big Data” fogalmat a felemelkedőben lévő új technológiák közé. A cég elemzői figyelmeztetnek, hogy nem csak az adatok mennyisége, hanem azok minősége és a megfelelő hozzáférés biztosítása hozzáférés is fontos, különben hiába fordítunk erőforrásokat a nagy adatmennyiségek tárolására és elemzésére.

Biztos vagyok benne, hogy a nagy adathalmazok feldolgozása a következő évtizedekben is kimeríthetetlen forrása lesz számos érdekes informatikai problémának, amelyek megoldásai pénzügyi, természettudományos és szórakoztatóipari alkalmazásokban is felhasználhatók lesznek.

Köszönetnyilvánítás

Köszönöm konzulensemnek, Dr. Gajdos Sándornak, hogy megismertetett az adatbázisok és az adattárházak világával, szakdolgozatom során pedig végig konstruktív javaslatokkal látott el.

Köszönöm a Kürt Zrt.-nek a kiváló munkakörülményeket és munkatársaimnak, hogy gyakornokként is érdekes problémákon dolgozhattam velük. Külön köszönöm Balogh Györgynek és Balázs Lászlónak, hogy munkám során elláttak jótanácsaikkal, és mindig készségesen segítettek a problémák leküzdésében.

Végül köszönöm szüleimnek, hogy tanulmányaim során végig támogattak. Nélkülük ez a dolgozat nem jöhetett volna létre.

Budapest, 2011. december

Ábrák jegyzéke

Táblázatok jegyzéke

Fogalomtár

aggregáció (felgörgetés, roll up) Valamelyik dimenzió mentén csökkentjük az adatok lekérdezésének granularitását, azaz kevésbé részletezetten nézzük azokat. Pl. napok helyett heteket vizsgálunk. Lásd még *lefúrás*.

B*-fa (B*-tree) Olyan fa szervezésű index, amelynek minden levele és csomópontja blokkméretű, és a gyökértől a levelekig vezető út mindig ugyanolyan hosszú – ezért nevezik *kiegyenlített* (balanced) fának.

Backus–Naur jelölés (BNF, Backus–Naur form) Környezetfüggetlen nyelvtanok leírására használható jelölési rendszer.

bittérképindex (bitmap index) Egy attribútumhoz tartozó bittérképindex bitvektorok olyan gyűjteménye, amelyben minden előforduló értékhez tartozik egy bitvektor. Adott értékhez tartozó v bitvektor az i. helyen értéke 1, ha az i. rekordban az attribútum értéke v, különben 0.

comma-separated values (CSV) Szöveges fájlformátum, amiben az egyes attribútumokat vesszővel (`,`) választják el. Általában a kiterjesztése `.csv`, ritkábban `.txt`.

dimenzió (dimension) Az adatokat tartalmazó többdimenziós tér egy tengelye; olyan tulajdonság, amely alapján a tényadatokat csoportosítani tudjuk.

drill down Lásd *lefúrás*.

decision support system (DSS) Döntéstámogató rendszer.

Extensible Markup Language (XML) A World Wide Web Consortium (W3C) által ajánlott általános szabványosított leírónyelv. Gépek és emberek számára is olvasható.

extract-transform-load (ETL) A hagyományos adattárház-architektúrában olyan folyamatok sorozata, amik során az operatív adatforrásokból átkerül az adattárházba. Lépései: az adatok kinyerése a forrásrendszerből (extract), átalakítása (transform), betöltése és indexelése (load). A hagyományos esetben az ETL folyamatba tartozik az adatok tisztítása is.

folyamat (process) Egy program futó példánya.

Google Web Toolkit (GWT) Olyan nyílt forráskódú eszközkészlet, ami lehetővé teszi összetett JavaScript front-end alkalmazások készítését Java nyelven.

granularitás (szemcsésség, granularity) Az adattárházban tárolt adatok részletességének szintje.

hierarchikus kapcsolat (hierarchical relationship) Olyan kapcsolat, melyen keresztül az adat magasabb szintre aggregálható szigorúan több-egy kapcsolatok sorozatán keresztül.

hybrid OLAP (HOLAP) Olyan OLAP rendszer, ahol az adatok egy részét multdimenziós, egy részét relációs struktúrákban tárolják. Ezzel jó kompromisszum alakítható ki a MOLAP és a ROLAP megoldások előnyei és hátrányai között.

index Az indexelt szervezés során az adatokhoz egy speciális struktúrát rendelünk hozzá, amelynek célja az adathalmazon futtatott lekérdezések gyorsítása. Hagyományos adatbázis-kezelők esetén az adatrekordok tipikusan a háttértáron helyekedik el. Az egyre gyakoribb in-memory adatbázisok adataikat a memóriában (is) tárolják és indexelik. Lásd még B*-fa, bitmap index.

kardinalitás (cardinality) Egy (tipikusan relációs) tábla adott oszlopában szereplő egyedi értékek száma. Pl. alacsony kardinalitású attribútumok: nem, családi állapot; magas kardinalitású attribútumok: dátum, név.

kimutatástábla Többdimenziós adatstruktúrák megjelenítésére alkalmas táblázatos elrendezés.

lapozófájl A virtuális memóriakezelés során a processzor nagyobb memóriaterületet címezhet meg, mint amennyi a számítógépben operatív memóriaként rendelkezésre áll. A fennmaradó területet a háttértáron, a lapozófájlban tárolják. Linux rendszerek esetén a lapozófájlt külön cserepartíción tárolják.

lefúrás (mélyre ásás, drill down) Valamelyik dimenzió mentén növeljük az adatok lekérdezésének granularitását, azaz egyre részletezettebben nézzük azokat. Pl. órás helyett perces bontásban vizsgáljuk azokat. Lásd még *aggregáció*.

Linux Szűkebb értelemben a nyílt forráskódú Linux rendszermag, fejlesztését Linus Torvalds 1991-ben kezdte meg. Gyakran az előbbire épülő Unix-szerű GNU/Linux operációs rendszert is Linuxnak nevezik. Jelenleg több mint háromszáz aktívan fejlesztett disztribúciója létezik.

multidimensional OLAP (MOLAP) Olyan OLAP rendszer, ahol az adatokat multdimenziós struktúrában tárolják.

multidimensional expressions (MDX) OLAP adatbázisok lekérdezésére szolgáló SQL-hez hasonló nyelv. Az MDX-et a multidimenziós lekérdezések de facto szabványa, a nagy cégek OLAP rendszerei szinte kivétel nélkül támogatják.

on-line analitikus feldolgozás (online analytic processing, OLAP) Az OLAP szigorú értelemben egy megközelítés a multidimenziós lekérdezések minél gyorsabb

megválaszolására. Jellemzően sok adatrekordot érintő, komplex lekérdezéseket szolgálnak ki elemzési, döntéstámogatási céllal. Tágabb értelemben a multidimenziós lekérdezéseket kiszolgáló rendszereket is OLAP-nak nevezik. Lásd még *ROLAP*, *MOLAP*, *HOLAP*, vö. *OLTP*

on-line tranzakciós feldolgozás (online transaction processing, OLTP) Tranzakciókezelő rendszer. Tömegesen beérkező, kevés rekordot érintő kérések jellemzik, leállása kritikus az üzletfolytonosság szempontjából. Pl. áruház adatbázisa, egy légitársaság helyfoglaló rendszere stb. A tranzakciókezelő rendszerek az áteresztő képesség maximalizálása és a válaszidő minimalizálása érdekében jellemzően erősen normalizált adatstruktúrákon dolgoznak. *OLAP*.

pivoting Adatkocka elforgatása.

pivot table Lásd *kimutatástábla*.

relational database managament system (RDBMS) Relációs adatmodellre épülő adatbázis-kezelő rendszer.

relational OLAP (ROLAP) Olyan OLAP rendszer, ahol az adatokat relációs adatbázis-kezelőben tárolják

roll up Lásd *aggregáció*.

slice and dice Lásd *szeletelés*.

swap Lásd *lapozófájl*.

szeletelés (slice and dice) Részkocka kiválasztása.

több-egy kapcsolat Egy kapcsolat több-egy, ha az első halmaz minden eleméhez a második halmaznak legfeljebb egy eleme kapcsolódik.

XML Lásd *Extensible Markup Language*.

Unix Többfeladatos (multitasking), többfelhasználós (multi-user) operációs rendszer. 1969-es kiadása óta széleskörben alkalmazzák, beágyazott rendszereken, mobil eszközökön és szervereken egyaránt. Lásd még *Linux*.

Irodalomjegyzék

[Agnosti, Nunzio, 2007] Maristella Agosti, Giorgio Maria Di Nunzio, *Web Log Mining: A study of user sessions*, Proceedings of the 10th DELOS Thematic Workshop on Personalized Access Profile Management and Context Awareness in Digital Libraries PersDLm, `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.117.4538&rep=rep1&type=pdf`, hozzáférés ideje: 2011. november 2.

[Anderson] Sean Eron Anderson, *Bit Twiddling Hacks*, `http://graphics.stanford.edu/~seander/bithacks.html`, hozzáférés ideje: 2011. november 30.

[C++ Standard, 2005] Working Draft, Standard for Programming Language C++, `http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2005/n1905.pdf`, 572. oldal

[Christensen] Ken Christensen, *Christensen Tools Page*, `http://www.cse.usf.edu/~christen/tools/toolpage.html`, hozzáférés ideje: 2011. november 4.

[Colantonio, Pietro, 2010] Alessandro Colantonio, Roberto Di Pietro, *CONCISE: Compressed 'n' Composable Integer Set*, Computing Research Repository, 2011, `http://arxiv.org/pdf/1004.0403v1`, hozzáférés ideje: 2011. november 20.

[Deliège, Pedersen, 2010] François Deliège, Torben Bach Pedersen, *Position list word aligned hybrid: optimizing space and performance for compressed bitmaps*, Proceedings of the 13th International Conference on Extending Database Technology, EDBT '10, Lausanne, `http://doi.acm.org/10.1145/1739041.1739071`, hozzáférés ideje: 2011. november 2.

[DMKD97] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, H. Pirahesh, *Data cube: A relational aggregation operator generalizing group-by, cross-tab and sub-totals*, Data Mining and Knowledge Discovery, 1:29–54, 1997, `http://research.microsoft.com/pubs/69578/tr-95-22.pdf`, hozzáférés ideje: 2011. szeptember 12.

[Duellman, 2007] *Databases for the Large Hadron Collider at CERN*, Stanford Linear Accelerator Center, XLDB Workshop, 2007. október 25., `http://www-conf.slac.stanford.edu/xldb07/xldb_lhc.pdf`, hozzáférés ideje: 2011. november 10.

[Fusco, Stoecklin, Vlachos, 2010] Francesco Fusco, Marc Ph. Stoecklin, Michail Vlachos, *NET-FLi: on-the-fly compression, archiving and indexing of streaming network*

traffic, Proceeding of the VLDB Endowment, Vol. 3, No. 2 (2010): p. 1382–1393, `http://dl.acm.org/citation.cfm?id=1921011`, hozzáférés ideje: 2011. október 17.

[Gajdos, 2006] Gajdos Sándor, *Adatbázisok*, Műegyetemi Kiadó, Budapest, 2006.

[Gajdos, 2011] Gajdos Sándor, *Adattárházak és alkalmazásaik*, órai jegyzet, 2011.

[Garcia, Ullman, Widom, 2001] Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom, *Adatbázisrendszerek megvalósítása*, Panem Könyvkiadó, Budapest, 2001.

[Han, Kamber, 2006] Jiawei Han, Micheline Kamber, *Data Mining: Concepts and Techniques*, 2nd Edition, Morgan Kaufmann, 2006.

[icCube Documentation] *icCube Documentation*, `http://www.iccube.com/support/documentation`, hozzáférés ideje: 2011. október 31.

[icCube OLAP Server] *icCube OLAP Server*, `http://www.iccube.com/products/iccube/olap-server` hozzáférés ideje: 2011. november 20.

[INFOCOM, 1999] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, Scott Shenker, *Web caching and Zipf-like distributions: evidence and implications*, IEEE INFOCOM, 1999, `http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=749260`, hozzáférés ideje: 2011. november 20.

[Jedox Palo Manuals] *Jedox Palo Manuals*, `http://www.jedox.com/en/community/mypalo/palo-manuals.html`, hozzáférés ideje: 2011. október 31.

[Johnson, 1999] Theodote Johnson, *Performance Measurements of Compressed Bitmap Indices*, Proceedings of 25th International Conference on Very Large Data Bases, Edinburgh, Morgan Kaufmann, pp. 278–89., 1999.

[Kimball, Ross, 2002] Ralph Kimball, Margy Ross, *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*, John Wiley & Sons Inc., New York, 2nd edition, 2002.

[Kovács, 2004] Kovács Ferenc, *Fejezetek az adatbázis-kezelő rendszerekből*, BME, jegyzet, 2004, `http://avalon.aut.bme.hu/~kovacsf/adatbazis/eloadas/jegyzet.pdf`, hozzáférés ideje: 2011. szeptember 12.

[Lemire, 2009] Daniel Lemire, *All About Bitmap Indexes... And Sorting Them*, Université du Québec à Montréal, 2009, `http://lemire.me/talks/uqamtalk.pdf`, hozzáférés ideje: 2011. október 15.

[Lemire, Kaser, 2011] Daniel Lemire, Owen Kaser, *Reordering columns for smaller indexes*, Inf. Sci., Vol. 12, No. 181 (2011): p. 2550–2570 2011. június, `http://dl.acm.org/citation.cfm?id=1963761`, hozzáférés ideje: 2011. október 17.

[Mansour et al., 2011] Essam Mansour, Amin Allam, Spiros Skiadopoulos, Panos Kalnis, *ERA: Efficient Serial and Parallel Suffix Tree Construction for Very Long Strings*, Computing Research Repository, 2011, http://arxiv.org/abs/1109.6884

[OLAP Council] OLAP Council, *OLAP and OLAP Server Definitions*, 1995, http://www.olapcouncil.org/research/glossaryly.htm, hozzáférés ideje: 2011. október 20.

[Oracle, 2005] Vivek Sharma, *Bitmap Index vs. B-tree Index: Which and When?*, 2005, http://www.oracle.com/technetwork/articles/sharma-indexes-093638.html, hozzáférés ideje: 2011. november 10.

[Palo Wiki] *Palo ETL Server – Architecture*, http://www.jedox.com/wikipalo/de/index.php/Kategorie:Palo_ETL_Server, hozzáférés ideje: 2011. szeptember 30.

[Roske, McMullen, 2008] Edward Roske, Tracy McMullen, *Look Smarter Than You Are with Essbase System 9*, interRel Press, Arlington, 1st edition, 2008, ISBN: 9781435713512 http://books.google.com/books/about/Look_Smarter_Than_You_Are_with_Essbase_S.html?id=UILYg7b-kxsC, hozzáférés ideje: 2011. november 3.

[Stockinger, Wu, Shoshani, 2002] Kurt Stockinger, Kesheng Wu, Arie Shoshani, *Strategies for Processing ad hoc Queries on Large Data Warehouses*, Proceedings of the 5th ACM international workshop on Data Warehousing and OLAP, 72–79. old., McLean, USA, 2002, http://dl.acm.org/citation.cfm?id=583901, hozzáférés ideje: 2011. november 25.

[The Register, 2011] Anna Leach, *World's first biz computer was British – and sold teacakes*, The Register, 2011. november 11, http://www.theregister.co.uk/2011/11/11/sixtieth_anniversary_of_leo/, hozzáférés ideje: 2011. november 14.

[Timoshenko] Egor A. Timoshenko, *On minimal collections of indexes*, http://explainextended.com/files/index-en.pdf, hozzáférés ideje: 2011. november 10.

[Wu, Otoo, Shoshani, 2005] Kesheng Wu, Ekow J. Otoo, Arie Shoshani, *Optimizing Bitmap Indices With Efficient Compression*, ACM Transactions on Database Systems, 2005, http://dl.acm.org/citation.cfm?id=1132864, hozzáférés ideje: 2011. november 1.

[Wu, Stockinger, Shoshani, 2008] Kesheng Wu, Kurt Stockinger, Arie Shoshani, *Breaking the Curse of Cardinality on Bitmap Indexes*, Proceedings of the 20th international conference on Scientific and Statistical Database Management, 348–365. old., Springer-Verlang, Berlin, 2008, http://dl.acm.org/citation.cfm?id=1425174, hozzáférés ideje: 2011. november 25.

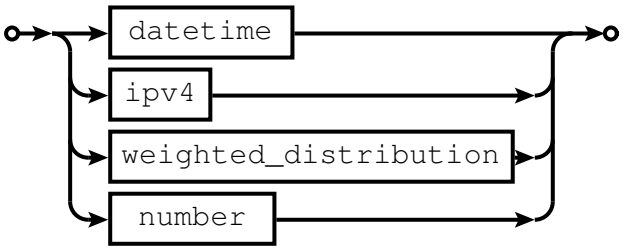
Függelék

F.1. Az mgen eszköz nyelvtana

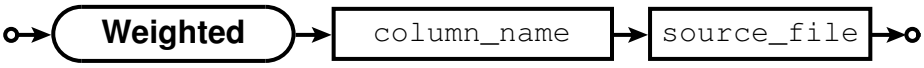
Az alábbiakban *szintaxis-diagramokkal* (syntax diagram vagy railroad diagram) ábrázolom az `mgen` eszköz formális nyelvtanának szabályait. A szintaxis diagramok a Backus–Naur formával (BNF) ekvivalens kifejezőerővel bírnak, de azoknál szemléletesebbek az emberi olvasó számára. A kifejezés a *linepattern* mintára illeszkedik.

F.1.1. ábra. *quoted_string*

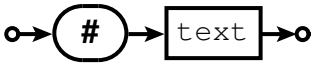
F.1.2. ábra. *ipv4*



F.1.3. ábra. *column_descriptor*

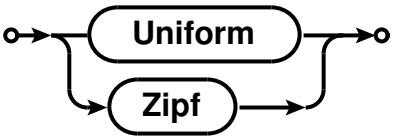


F.1.4. ábra. *weighted*



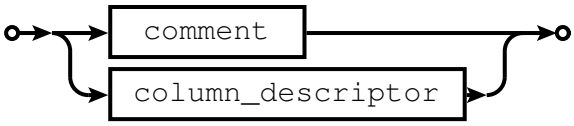
F.1.5. ábra. *comment*

A fenti nyelvtani szabályoknak megfelel az F.1.1. mintaállomány.

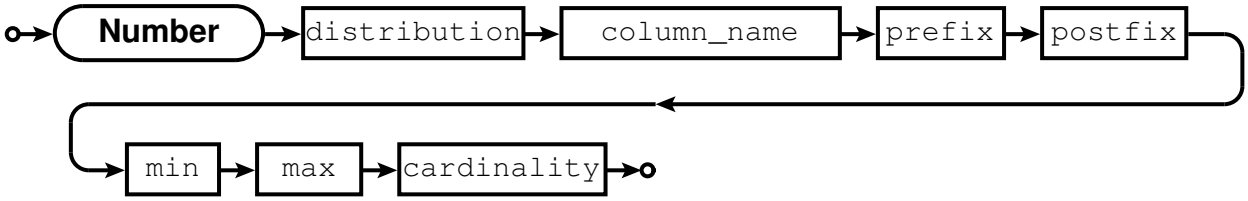


F.1.6. ábra. *distribution*

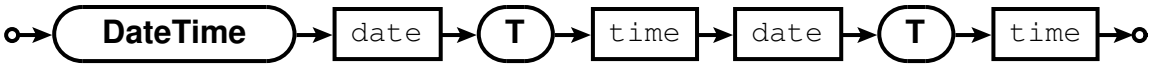
F.1.7. ábra. *column_name*



F.1.8. ábra. *linepattern*



F.1.9. ábra. *number*



F.1.10. ábra. *datetime*

```
# datum es ido
DateTime 20110311T1200 20110312T1200
# forras es cel IP cimek
IPv4 Zipf "source" 5
IPv4 Zipf "destination" 2
# HTTP metodus es statusz kodok
Weighted "method" "http_method.csv"
Weighted "status" "http_status.csv"
# esemenyek szamossaga
Number Uniform "count" "" "" 1 100 0
```

F.1.1. lista. *Egy mgen mintaállomány*

F.2. Kereskedelmi OLAP rendszerek tulajdonságai – források

A 3.1. táblázat elkészítéséhez az alábbi forrásokat használtam fel.

Oracle Essbase

Architektúra: `http://docs.oracle.com/cd/E10530_01/doc/epm.931/html_esb_dbag/frameset.htm?hybrid.htm`

Platform: `http://www.oracle.com/technetwork/middleware/essbase/`

```
$ mgen pattern01.txt 5
date,time,source,destination,method,status,count
2011/03/12,02:06,173.99.231.109,148.33.66.182,GET,200,67
2011/03/11,21:41,98.55.137.170,219.206.24.254,GET,200,100
2011/03/12,09:19,173.99.231.109,148.33.66.182,GET,200,26
2011/03/11,13:57,173.99.231.109,219.206.24.254,GET,200,82
2011/03/11,17:24,3.114.170.149,148.33.66.182,GET,200,87
```

F.1.2. lista. *Generált mátrix*

downloads/index.html

Lekérdezőnyelvek: http://docs.oracle.com/cd/E10530_01/doc/epm.931/esb_93130_readme.pdf

Crazy Development icCube

Platform, lekérdezőnyelvek: http://www.iccube.com/products/iccube/olap-server

Architektúra: http://www.iccube.com/support/forum/4-iccube-products/191-iccube-architecture

Microsoft Analysis Services

Architektúra: http://msdn.microsoft.com/en-us/library/aa155455(v=office.10).aspx

Platform: http://technet.microsoft.com/en-us/library/bb838707(office.12).aspx

Lekérdezőnyelvek: http://msdn.microsoft.com/en-us/library/bb500184.aspx

MicroStrategy Intelligence Server

Architektúra: http://www.microstrategy.com/bi-applications/5Styles/olap_analysis.asp

Platform: http://www.microstrategy.com/Software/Products/Intelligence_server/

Lekérdezőnyelvek: http://www.microstrategy.com/producthelp/manuals/9.2.1/en/MDXCubeReporting.pdf

Pentaho Mondrian

Architektúra: http://mondrian.pentaho.com/documentation/architecture.php

Platform: http://www.pentaho.com/download/

Lekérdezőnyelvek: http://mondrian.pentaho.com/documentation/mdx.php

Oracle OLAP Option

Architektúra: http://www.oracle.com/technetwork/database/options/olap/index.html

Platform: http://docs.oracle.com/cd/B28359_01/olap.111/b28124/start.htm

Lekérdezőnyelvek: http://www.oracle.com/technetwork/database/options/olap/olap-option-database-11g-171752.pdf

Jedox Palo

Architektúra: http://www.jedox.com/assets/files/downloads/pdf/pm/PM-Palo2-5-en.pdf
Platform: http://www.jedox.com/en/downloads/Palo-Downloads-current-version/further-palo-downloads.html
Lekérdezőnyelvek: http://www.jedox.com/en/about-jedox/press-ordner/press-archive/archive/Palo-Open-Source-OLAP-Server-now-supports-MDX-and-Excel-Pivot-Tables.html

SAS OLAP Server

Architektúra: http://support.sas.com/resources/papers/proceedings09/103-2009.pdf
Platformok: http://www.sas.com/technologies/dw/storage/mddb/#section=5
Lekérdezőnyelvek: http://support.sas.com/documentation/onlinedoc/91pdf/sasdoc_913/olap_mdx_9317.pdf

SAP BusinessObjects

Architektúra: http://download.sap.com/KIO0010/download.epd?context=80B3E6FF9435073A2FA25B8F0CA0E2078A773B8925FEA4E8A165095561E7BB6E4C8EE0ACD980E250815D91BFE4431038FFCE099F0C541817
Platform: http://www.sdn.sap.com/irj/sdn/linux
Lekérdezőnyelvek: http://www.sdn.sap.com/irj/scn/go/portal/prtroot/docs/library/uuid/b0320722-741c-2c10-afab-93b5c0fc7e96?QuickLink=events&overridelayout=true

IBM TM1

Architektúra: ftp://public.dhe.ibm.com/software/data/sw-library/cognos/ch/de/IBM_Cognos_8_20091012.pdf
Platform: https://www-304.ibm.com/support/docview.wss?uid=swg27020888
Lekérdezőnyelvek: http://publib.boulder.ibm.com/infocenter/c8bi/v8r4m0/index.jsp?topic=/com.ibm.swg.im.cognos.ug_cr_rptstd.8.4.0.doc/ug_cr_rptstd_id12174cr_rptstd_wrkdat_working_with_sql_m.html

F.3. A Palo és az icCube telepítése

Az alábbiakban röviden összefoglalom a dolgozatban bemutatott kereskedelmi MOLAP motorok telepítését. A telepítést az Ubuntu Linux 11.04 (kódnév: Natty Narwhal) 64 bites verzióján végeztem.

F.3.1. Palo

A Jedox Palo 64 bites verziója ingyenesen letölthető a http://www.jedox.com/en/downloads/Palo-Downloads-current-version/palo-downloads.html címről. A telepítés menete az F.3.1 felsorolásban látható.

```
$ tar -xvf Palo_Suite_3_2.tar
$ sudo ./install.sh
$ cp molap.lic /opt/jedox/ps/Data
$ sudo cp fopper.lic /opt/jedox/ps/tomcat/webapps/fopper/WEB-INF/
$ sudo cp svs.lic /opt/jedox/ps/svs-Linux-i386/
$ sudo cp etlpremium.lic /opt/jedox/ps/tomcat/webapps/etlserver/
$ cd /opt/jedox/ps
$ sudo ./startserver.sh
```

F.3.1. lista. *A Palo telepítése Linuxon*

Indítás után – ha nem változtattuk meg telepítéskor az alapértelmezett portot – a szerver a `http://localhost:80/` címen érhető el. Az automatikusan létrehozott felhasználók bejelentkezési adatai (felhasználónév / jelszó): admin / admin, john / john. Az ETL Manager indításakor az alábbi hibaüzenet jelenik meg: *ETL Login service is not available.*

Az ETL Manager automatikus indításához a `startserver.sh`-ban kell megszüntetni a `./tomcat/bin/startup.sh` sort kommentezését. Az ETL Manager külön is elindítható, ekkor a `/opt/jedox/ps/tomcat/bin` könyvtárban kell kiadni az F.3.2. parancsot.

```
$ cd /opt/jedox/ps/tomcat/bin/
$ sudo ./startup.sh
```

F.3.2. lista. *A Palo ETL Manager indítása*

Ha ezután sem működik az ETL Manager, a webes felületen ki- és be kell jelentkezni. Ha továbbra is hibaüzenetet kapunk, törölni kell a böngésző gyorsítótárából a sütiket.[1]

F.3.2. icCube

Az icCube OLAP Server Community Edition verziója letölthető a `http://www.iccube.com/download` címről.

```
$ unzip icCube.zip
$ cd bin
$ ./icCube.sh
```

F.3.3. lista. *Az icCube telepítése és indítása*

Ezután az icCube webes felülete elérhető a `http://localhost:8282/icCube/icCube.html` címen. A bejelentkezés az admin / admin felhasználónév–jelszó párral lehetséges.

F.4. XMLA lekérdezés icCube-bal

F.5. Függvénykönyvtárak telepítése

F.5.1. Boost C++ Libraries

A Boost C++ Libraries egyszerűen telepíthető az `libboost-dev` csomaggal.

[1] A legtöbb böngészőben ezt a CTRL + SHIFT + DELETE billentyűk lenyomásával tehetjük meg a leggyorsabban.

```
<soap:Envelope
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header/>
  <soap:Body>
    <Execute xmlns="urn:schemas-microsoft-com:xml-analysis">
      <Command>
        <Statement>
          SELECT
            [DStatusz].[HStatusz] ON 0,
            [DMetodus].[HMetodus] ON 1
          FROM [TesztKocka]
        </Statement>
      </Command>
      <Properties>
        <PropertyList>
            <Catalog>TesztKocka</Catalog>
            <Content>Data</Content>
        </PropertyList>
      </Properties>
    </Execute>
  </soap:Body>
</soap:Envelope>
```

F.4.1. lista. *A* `request.xmla` *fájl tartalma*

```
$ cat request.xmla | curl -X POST -H 'Content-type: text/xml'
  -d @- http://admin:admin@localhost:8282/icCube/xmla
<?xml version="1.0" ?>
<soap:Envelope
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 ...
 <soap:Body>
    <ExecuteResponse xmlns="urn:schemas-microsoft-com:xml-analysis">
      <return>
      ...
        <CellData>
          <Cell CellOrdinal="0">
            <Value xsi:type="xsd:long">
              510393
            </Value>
            <FmtValue>
              510393
            </FmtValue>
          </Cell>
        </CellData>
      ...
      </return>
    </ExecuteResponse>
  </soap:Body>
</soap:Envelope>
```

F.4.2. lista. *XMLA lekérdezés a* `Teszkocka` *adatkockán és a válasz néhány sora tördelve*

```
$ sudo apt-get install libboost-dev
```

F.5.1. lista. *A Boost telepítése*

F.5.2. FastBit

A FastBit letölthető a `http://crd-legacy.lbl.gov/~kewu/fastbit/index.html` oldalról. A telepítéshez szükséges utasításokat az F.5.2 felsorolás tartalmazza.

```
$ ./configure
$ make
$ make check
$ sudo make install
$ sudo ldconfig
```

F.5.2. lista. *A FastBit telepítése*

A függvénykönyvtár használatához be kell állítani, hogy a linker beépítse a FastBit könyvtárat. Ezt Eclipse CDT fejlesztőkörnyezetben a projekt tulajdonságai alatt a C/C++ Build | Settings | GCC C++ Linker | Libraries alatt kell hozzáadni a `fastbit` könyvtárat.

F.6. Bittérkép tömörítő algoritmusok pszeudokódja

F.6.1. AOWAH32

F.6.2. AOPL32

Algoritmus 1 AOWAH32 bittérképben bit 1-esre állítása

```
1: procedure SetBit(W tömörített bittérkép, i bitindex)
2:     szóPoz ← i div 31
3:     szóTávolság ← szóPoz − szavakSzáma
4:     if szóTávolság ≥ 1 then
5:         if W_n = 0x7FFFFFFF then                          ▷ W_n csupa 1 literál
6:             if n ≥ 2 és W_{n−1}&0xC0000000 = 0xC0000000 then   ▷ W_{n−1} szó 1-fill
7:                 W_{n−1} ← W_{n−1} + 1                      ▷ W_{n−1}-et bővítjük
8:                 n ← n − 1                                  ▷ az utolsó szót töröljük
9:             else
10:                W_n ← 0xC0000001          ▷ az utolsó szót 1 hosszú 1-fillre cseréljük
11:            end if
12:        else
13:            if W_n = 0x00000000 then   ▷ W_n csupa 0 literál (csak az első futáskor)
14:                n ← n − 1                                  ▷ az utolsó szót töröljük
15:                szóTávolság ← szóTávolság + 1
16:            end if
17:        end if
18:        if szóTávolság = 1 then                ▷ az új szó szomszédos az utolsóval
19:            n ← n + 1                                  ▷ egy új literált szúrunk be
20:            W_n ← 0x00000000
21:        else                                                ▷ szóTávolság > 1
22:            W_n ← 0x00000000 + szóTávolság − 1              ▷ új fillt szúrunk be
23:            n ← n + 1
24:            W_n ← 0x00000000                            ▷ új literált szúrunk be
25:        end if
26:    end if
27:    bitPoz ← i mod 31
28:    szavakSzáma ← i div 31
29:    W_n ← W_n|(0x40000000 >> bitPoz)                ▷ új bit beszúrása a literálba
30: end procedure
```

Algoritmus 2 AOWAH32 bittérkép következő szavának kiolvasása

```
1: procedure NextWord(W tömörített bittérkép, i szóindex)
2:     indexekSzáma ← 0
3:     if W_i&0x80000000 ≠ 0 then                                   ▷ W_i fill
4:         if i > hossz(W) then
5:             return -1
6:         end if
7:         fill ← true
8:         if W_i&0x40000000 ≠ 0 then                               ▷ W_i 1-fill
9:             indexek[1] ← blokkokSzáma × 31          ▷ a csupa 1-es sorozat kezdete
10:            indexekSzáma ← (W_i&0x3FFFFFFF) × 31
11:            blokkokSzáma ← blokkokSzáma + W_i&0x3FFFFFFF
12:            i ← i + 1
13:            return 0
14:        else                                                     ▷ W_i 0-fill
15:            blokkokSzáma ← blokkokSzáma + W_i&0x3FFFFFFF
16:            i ← i + 1                                ▷ ugrunk a következő szóra
17:        end if
18:    end if
19:    if W_i&0x80000000 = 0 then                                   ▷ W_i literál
20:        fill ← false
21:        v ← W_i
22:        v ← v << 1                                   ▷ a szó MSB-jét töröljük
23:        p ← 0                                ▷ az aktuális pozíció kezdetben 0
24:        while v ≠ 0 do
25:            if v&0x80000000 ≠ 0 then                             ▷ v MSB-je 1
26:                indexek[indexekSzáma] ← blokkokSzáma × 31 + p
27:                indexekSzáma ← indexekSzáma + 1
28:            end if
29:            p ← p + 1
30:            v ← v << 1                       ▷ a szót az MSB-je felé shifteljük
31:        end while
32:        i ← i + 1
33:        blokkokSzáma ← blokkokSzáma + 1
34:        return 0
35:    end if
36: end procedure
```

Algoritmus 3 AOPL32 bittérképben bit 1-esre állítása

```
1: procedure SetBit(W tömörített bittérkép, i bitindex)
2:     bitPoz ← i mod 31                                   ▷ a beszúrt bit hányadik a szóban
3:     szóPoz ← i div 31          ▷ a beszúrt bit hányadik szóban lenne tömörítetlenül
4:     szóTávolság ← szóPoz − utolsóSzóPoz
5:     if szóTávolság = 0 then
6:         W_i = W_i|0x40000000 >> bitPoz                  ▷ beszúrjuk az utolsó szóba
7:     else                                                ▷ szóTávolság > 0
8:         c ← CountBits(W_i)                              ▷ az utolsó szóban hány bit van
9:         if c ≤ 2 vagy c ≥ 29 then                       ▷ ha kellően homogén, fillé alakítjuk
10:            W ← ConvertToFill(W, i)
11:        end if
12:        if szóTávolság = 1 then                         ▷ a beszúrt szó szomszédos az utolsóval
13:            i ← i + 1          ▷ beszúrunk egy literált, amely az új bitet tartalmazza
14:            W_{i+1} ← 0x40000000 >> bitPoz
15:        else                                            ▷ szóTávolság > 1
16:            if W_i&0xC0000000 = 0x80000000 és W_i&0xC00FFFFF = W_i then  ▷ ha az
    utolsó szó 0-fill üres pozíciólistával
17:                fillSzám ← W_i&0x000FFFFF + szóTávolság − 1          ▷ bővítjük
18:            else
19:                fillSzám ← szóTávolság − 1
20:                i ← i + 1                                ▷ új fillt szúrunk be
21:                W_i ← 0x80000000
22:            end if
23:            while fillSzám > 0x000FFFFF do      ▷ amíg a fillek száma túlcsordulást
    okozna
24:                W_i ← 0x80000000|0x000FFFFF                  ▷ több fillre bontjuk
25:                i ← i + 1
26:                W_i ← 0x00000000
27:                fillSzám ← fillSzám − 0x000FFFFF
28:            end while
29:            W_i ← 0x80000000|fillSzám          ▷ az utolsó fill blokkot fillé alakítjuk
30:            i ← i + 1          ▷ beszúrunk egy literált, amely az új bitet tartalmazza
31:            W_i ← 0x40000000 >> bitPoz
32:        end if
33:        utolsóSzóPoz ← i div 31
34:    end if
35: end procedure
```

Algoritmus 4 Adott szó bitjeiek megszámolása

```
1: procedure CountBits(W tömörített bittérkép, i szóindex)
2:     v ← W_i
3:     c ← 0
4:     while v ≠ 0 do
5:         v ← v&(v − 1)
6:         c ← c + 1
7:     end while
8:     return c
9: end procedure
```

Algoritmus 5 AOPL32 bittérkép utolsó szavának fillé konvertálása

1: **procedure** ConvertToFill(W tömörített bittérkép, i bitindex, c bitek száma)
2: $v \leftarrow W_i << 1$ ▷ MSB eltávolítása
3: **if** $c \leq 2$ **then** ▷ 0, 1 vagy 2 *nulla* bit van → 0-fill
4: $W_i \leftarrow$ `0x80000001` ▷ 1 hosszú 0-fill
5: **end if**
6: **if** $c \geq 29$ **then** ▷ 0, 2 vagy 2 *egy* bit van → 1-fill
7: $W_i \leftarrow$ `0x40000001` ▷ 1 hosszú 1-fill
8: $v \leftarrow \neg v$ ▷ v-t negáljuk, így az 1-eseket keresve a 0-kat találjuk meg
9: **end if**
10: $p_1 \leftarrow 0$ ▷ a pozíciólista 1. eleme
11: $p_2 \leftarrow 0$ ▷ a pozíciólista 2. eleme
12: $n \leftarrow 0$ ▷ a pozíciólista bitjeinek száma
13: **for** $p = 0$ to 30 **do** ▷ eltérő bitek keresése
14: **if** v&`0x80000000` $\neq 0$ **then** ▷ v MSB-je 1
15: $n \leftarrow n + 1$
16: **if** $n = 1$ **then**
17: $p_1 \leftarrow p$
18: **else**
19: **if** $n = 2$ **then**
20: $p_2 \leftarrow p$
21: **end if**
22: **end if**
23: **end if**
24: $v \leftarrow v << 1$ ▷ v-t balra léptetjük
25: **end for**
26: **if** $n \geq 1$ **then** ▷ ha vannak a pozíciólistában elemek, beszúrjuk őket
27: $W_i = W_i|(p1 + 1) << 20$
28: **if** $n \geq 2$ **then**
29: $W_i = W_i|(p1 + 2) << 25$
30: **end if**
31: **end if**
32: **if** $i \geq 2$ **then** ▷ megvizsgáljuk, hogy egyesíthető-e a lista utolsó két eleme
33: **if** W_{i-1}&$MSB \neq 0$ és W_{i-1}&`0xC00FFFFF` $= W_{i-1}$ **then** ▷ W_{i-1} tiszta fill
34: **if** (W_{i-1}&`0xC0000000` = `0xC0000000` és W_i&`0xC0000000` = `0xC0000000`) vagy (W_{i-1}&`0xC0000000` = `0x80000000` és W_i&`0xC0000000` = `0x80000000`) **then** ▷ W_{i-1}, W_i azonos típusú fillek
35: **if** W_{i-1}&`0xC0000000` = `0xC0000000` és W_{i-1}&`0x000FFFFF` = `0x000FFFFF` **then**
36: $W_i \leftarrow$ `0xC0000001` ▷ túlcsordulás új 1-fillt nyitunk W_i-ben
37: **else**
38: $W_{i-1} \leftarrow W_{i-1} + 1$ ▷ W_{i-1} eggyel hosszabb
39: $W_{i-1} \leftarrow W_{i-1}|(W_i$& `0xC00FFFFF`) ▷ W_i pozíciólistája W_{i-1}-be
40: $i \leftarrow i - 1$ ▷ W_i-t töröljük
41: **end if**
42: **end if**
43: **end if**
44: **end if**
45: **return** W
46: **end procedure**

Algoritmus 6 AOPL32 tömörített szó olvasása

```
1: procedure NextWord(W tömörített bittérkép, i szóindex)
2:     if i > hossz(W) then
3:         return -1
4:     end if
5:     indexekSzáma ← 0                                   ▷ az indexek száma kezdetben 0
6:     while indexekSzáma = 0 és W_i&0x80000000 ≠ 0 do    ▷ amíg W_i nemüres fill
7:         fill ← true
8:         if W_i&0xC0000000 = 0x80000000 then                          ▷ W_i 0-fill
9:             while W_i&0x000FFFFF = 0x000FFFFF do                     ▷ túlcsorduló fill
10:                blokkokSzáma ← blokkokSzáma + 0x000FFFFF             ▷ szomszédos 0-fill
   szavak átlépése
11:                i ← i + 1
12:            end while
13:            falseFill ← true
14:            blokkokSzáma ← blokkokSzáma + W_i&0x000FFFFF − 1         ▷ blokkok
   számának növelése
15:            if W_i&0x01F00000 ≠ 0 then                   ▷ a pozíciólista nem üres
16:                (indexekSzáma, indexek[2, 3]) ← GetPositions(W, i)
17:            else                                                     ▷ a pozíciólista üres
18:                i ← i + 1
19:            end if
20:            blokkokSzáma ← blokkokSzáma + 1
21:        else                                                         ▷ W_i 1-fill
22:            falseFill ← false
23:            nullákSzáma ← 0                                  ▷ a 0-k száma kezdetben 0
24:            indexek[1] ← W_i × 31
25:            blokkokSzáma ← blokkokSzáma + W_i&0x000FFFFF − 1
26:            if W_i&0x01F00000 ≠ 0 then                   ▷ a pozíciólista nem üres
27:                (nullákSzáma, nullák[1, 2]) ← GetPositions(W, i)
28:            end if
29:            blokkokSzáma ← blokkokSzáma + 1
30:        end if
31:    end while
32:    if W_i&0x80000000 = 0 then                                       ▷ W_i literál
33:        fill ← false
34:        v ← W_i
35:        v ← v << 1                                          ▷ v MSB-jét töröljük
36:        i ← 0
37:        while W_i ≠ 0 do
38:            if v&0x8000000 ≠ 0 then                                  ▷ v MSB-je 1
39:                indexek[indexekSzáma] = blokkokSzáma × 31 + i)
40:                v ← v << 1
41:            end if
42:            blokkokSzáma ← blokkokSzáma + 1
43:        end while
44:        indexek ← Indexek
45:    end if
46:    i ← i + 1
47:    return 0
48: end procedure
```

Algoritmus 7 AOPL32 tömörített fill szó pozíciólistának kinyerése

1: **procedure** GetPositions(W tömörített bittérkép, i szóindex)
2: $poz[1] \leftarrow blokkokSzáma \times 31 + (W_i\&\texttt{0x01F00000}) >> 20 - 1$
3: $pozSzáma \leftarrow 1$
4: **if** $W_i\&\texttt{0x3E000000} \neq 0$ **then** ▷ a pozíciólista 2. eleme nem üres
5: $poz[2] \leftarrow blokkokSzáma \times 31 + (W_i\&\texttt{0x3E000000}) >> 25 - 1$
6: $pozSzáma \leftarrow 2$
7: **end if**
8: **return** $(pozSzáma, poz)$
9: **end procedure**