Adatbázisok haladóknak

2012. ősz

Oszlopalapú adatbázis-kezelők

Tanulmány

Készítette: Barabás Ákos

1# BEVEZETÉS 3#

2# OSZLOPALAPÚ ADATTÁROLÁS 3#

2.1# Motiváció 3#

2.2# Az oszlopalapú szervezés [1][2] 5#

2.3# Sor- és oszlopalapú szervezés összehasonlítása [2][4] 6#

2.4# Materializáció [3][4] 7#

3# TELJESÍTMÉNY 10#

4# TÖMÖRÍTÉSI MÓDSZEREK [2][8][9][10] 14#

4.1# Futamhossz kódolás (run-length encoding) 14#

4.2# Szótáralapú kódolás 14#

4.3# Delta tömörítés 15#

4.4# Lempel-Ziv-Welch (LZW) tömörítés 15#

5# KIEGÉSZÍTŐ TECHNIKÁK 17#

5.1# Elosztott tárolás [1] 17#

5.2# MPP [12] 17#

5.3# Bitmap indexek [10] 18#

5.4# Sűrű csomagolás 19#

5.5# Párhuzamos töltés és lekérdezés [13] 19#

6# PIACI KITEKINTÉS 20#

7# FORRÁSOK 22#

1 Bevezetés

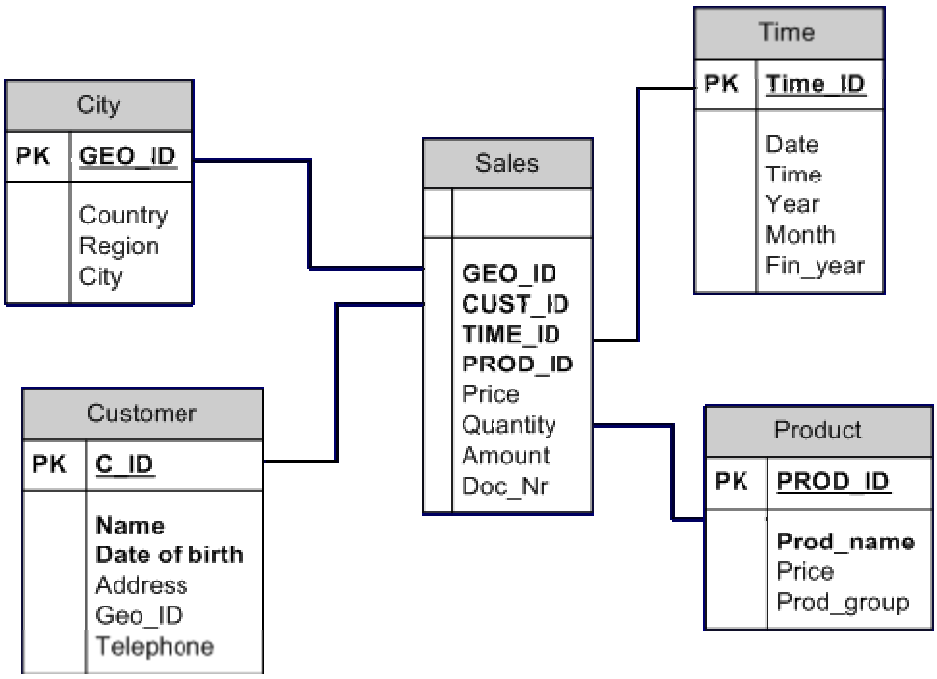
A tanulmány az ún. oszlopalapú (*column-based*, *column-oriented*) adatbázis-kezelők bemutatásának céljával jött létre. Ezek a rendszerek különleges fizikai szervezést használnak annak érdekében, hogy bizonyos műveleteket – jellemzően lekérdezéseket – a klasszikus fizikai szervezést használó rendszerekhez képest nagyságrendekkel hatékonyabban tudjanak elvégezni.
A tanulmány bemutatja az oszlopalapú szervezés alapmódszerét, összehasonlítja a soralapú (*row-based*, *row-oriented*) módszerrel, majd bemutatja a lekérdezések kiértékelésének lehetőségeit. Ezt követően szó esik ezen különleges rendszerek teljesítményéről, majd a körükben előszeretettel alkalmazott tömörítési módszerekről. Ezt követően bemutatásra kerül néhány olyan technika, melyeket az oszlopalapú adatbázis-kezelők számos esetben alkalmaznak, illetve rövid kitekintést teszünk a rendszerek szállítóinak piacára.

2 Oszlopalapú adattárolás

Ebben a fejezetben az oszlopalapú tárolás alapjait, soralapú módszerhez képest vett előnyeit és hátrányait illetve a materializáció lehetőségeit mutatom be.

2.1 Motiváció

Az alábbiakban egy példát láthatunk arra, hogy miért érdemes a megszokott soralapú szervezéstől eltérő módszereket alkalmazni adatbázis-kezelőkben.
Az adatbázis-kezelők analitikus – lekérdezés-orientált – alkalmazásakor jellemző gyakorlat a dimenziós adatmodell használata. A dimenziós adatmodellben ún. tény- és dimenziós adatokat különböztetünk meg. A ténytáblákban jellemzően tranzakciók (pl. vásárlás, telefonhívás) adatai találhatók, melyeket a dimenziók (pl. áru, időpont, ügyfél) jellemeznek. A dimenziótáblák rekordjaira a ténytáblák idegen kulcsokkal hivatkoznak. Egy értékesítési adatokat modellező csillagséma látható az Ábra 1. ábrán. A ténytáblában az ügyél, idő, város és idő dimenziókra mutató idegen kulcsokon kívül az ár, a mennyiség, a végösszeg és a számlaszám található. Az ügyfeleket nevük, címük, városuk és telefonszámuk alapján jellemezzük. Megjegyzendő, hogy egy valós implementációban a dimenziók oszlopainak száma a példáéhoz képest sokkal nagyobb – több tucat vagy száz – is lehet.



Ábra 1 Csillagséma értékesítési adatokra

A ténytáblák jellemzően kevés oszlopot, de sok rekordot tartalmaznak, míg a dimenziótáblák ennek ellenkezői: nagyságrendekkel kevesebb soruk van, de oszlopaik száma akár a százat is átlépheti. Gondoljunk például egy mobilkommunikációs szolgáltató adatpiacának és/vagy –tárházának egy csillagsémájára, melyben a tényadatok a hívásokat jellemzik hívó és hívott fél, időpont, cellainformáció és számlacsomag dimenziók szerint! A ténytábla rekordjainak száma egy kb. másfél évtizetede a magyar piacon szereplő, nem piacvezető szolgáltatónál is tízmillárdos nagyságrendű lehet, azonban viszonylag kevés (2-10) dimenzió alapján jellemezzük a hívásokat. Ezzel szemben az előfizetők száma ugyanezen piaci körülmények közt néhány millió, de az egyes előfizetők jellemzésére számos szempontot megadhatunk az alapvető adatoktól (pl. vezeték- és keresztnév, lakóhely, életkor, családi állapot) az összetettebbekig (pl. hány családtagja előfizető még az adott szolgáltatónál, hányszor váltott számlacsomagot, milyen autója van). Egy ilyen sémán megfogalmazott lekérdezés lehet például a következő:

```
SELECT d.name, d.first_name, SUM(f.*), AVG(f.call_time)
FROM f_call AS f, d_customer AS d, d_time AS t
WHERE f.customer_id = d.id AND f.time_id = t.id
AND d.region = 'Borsod-Abauj-Zemplen'
AND t.year = 2011
GROUP BY d.name, d.first_name;.
```

A lekérdezés segítségével megtudhatjuk a BAZ megyei lakosok által 2011-ben kezdeményezett hívások számát és azok átlagos időtartamát.

Hagyományos, soralapú tárolást használó RDBMS használata esetén a ténytáblán kívül két dimenziótáblát is be kell olvasni a művelet elvégzéséhez, melyek összesen több mint száz oszlopot is tartalmazhatnak. Műveleteket azonban csak csak néhány oszlopon végzünk, így a beolvasott adatok nagy része feleslegesen foglalja az erőforrásokat (pl. I/O, memória).

Az ilyen és ehhez hasonló problémák megoldására született meg az oszlopalapú adattárolás technikája.

2.2 Az oszlopalapú szervezés [1][2]

A RDBMS-ekben az adatok fizikai szervezésének általánosan elterjedt módszere, hogy a rendszerek táblák sorait – a rekordokat – tárolják. A soralapú szervezés egy ezzel ellentétes megközelítést használ: az adatokat oszlopok szerint szervezi.

Az alábbiakban egy, a személy(ID, keresztnév, életkor, lakhely) sémára illeszkedő relációt láthatunk először sor-, majd oszlopalapú szervezéssel.

ID	Keresztnév	Életkor	Lakhely
1000	Klemens	42	Stuttgart
1001	Rajesh	29	Delhi
1002	Francesco	30	Rome
1003	Colin	51	Dublin

Soralapú szervezés:

1. blokk	1000	Klemens	42	Stuttgart
2. blokk	1001	Rajesh	29	Delhi
3. blokk	1002	Francesco	30	Rome
4. blokk	1003	Colin	51	Dublin

Oszlopalapú szervezés:

1. blokk	1000	1001	1002	1003
2. blokk	Klemens	Rajesh	Francesco	Colin
3. blokk	42	29	30	51
4. blokk	Stuttgart	Delhi	Rome	Dublin

Mivel az előbbi példa csak a szervezés alapelvét mutatta meg, egy lényeges elem kimaradt belőle: például hogyan állapítjuk meg, hogy egy adott névhez melyik életkor tartozik? Az

oszlopalapú RDBMS-ek különböző implementációi erre jellemzően rekord azonosítókat (*record ID*, *tuple ID*) alkalmaznak, melyek az oszlopok egyes elemeihez vannak rendelve. Ennek megfelelően az előbbi oszlopalapú szervezés példa a következőképpen egészíthető ki:

1. blokk	1	1000	2	1001	3	1002	4	1003
2. blokk	1	Klemens	2	Rajesh	3	Francesco	4	Colin
3. blokk	1	42	2	29	3	30	4	51
4. blokk	1	Stuttgart	2	Delhi	3	Rome	4	Dublin

Fontos megjegyezni, hogy az oszlopalapú adatbázis-kezelők is relációkat tárolnak, tehát az adatmodell nem különbözik a hagyományos RDBMS-ektől, mindössze a fizikai szervezésben tér el a sor- és oszlopalapú szervezés.

2.3 Sor- és oszlopalapú szervezés összehasonlítása [2][4]

Az eddigiekben már láthattunk rá egy példát, hogy milyen esetekben nyújtanak előnyöket az oszlopalapú RDBMS-ek soralapú társaikhoz képest – az alábbiakban az egyes szervezési módszerek előnyei és hátrányai kerülnek bemutatásra.

Ahogy a 3. fejezetben is látni fogjuk, a soralapú RDBMS-ek teljesítménye nagy adatmennyiségeken végzett lekérdezések, különösen nagy projektivitás és/vagy szelektivitás esetén marad el oszlopalapú társaikétól. Változatos lekérdezések esetén az indexek – különösen a rendelkezésre álló memória méretén túli – növekedése, illetve folyamatos karbantartása is nehézségeket okozhat a soralapú rendszerekben.

Az oszlopalapú rendszerek ennek megfelelően annál nagyobb előnyre tesznek szert, mennél a magasabb a lekérdezések projektivitása és/vagy szelektivitása (*megj.*: a szelektivitás hatása alacsonyabb, ld. 3. alfejezet). Az oszlopalapú RDBMS-ek ugyancsak előnyben vannak az aggregációs műveletek (oszlopfüggvények) végrehajtásakor. Mivel az aggregációk egy oszlopon végeznek műveleteket (ezért, melyeket az oszlopalapú rendszerek könnyedén beolvasnak, míg a soralapúaknak ehhez az egész tábla beolvasásra szükségük van.

Az oszlopalapú RDBMS-ek különleges fizikai szervezésüknek köszönhetően képesek több lekérdezést párhuzamosan végrehajtani, amennyiben azok szeparáltak, azaz a predikátumaik által érintett attribútumhalmazok diszjunktak.

Mivel az oszlopalapú rendszereken végzett lekérdezések végrehajtása esetén (erről bővebben ld. 2.4) jellemzően csak a lekérdezések által érintett oszlopok kerülnek beolvasásra a háttértárról, a soralapú szervezéshez képest a hardveres erőforrások kihasználása is hatékonyabb. Az

oszlopalapú rendszerek teljesítménye tovább javítható az oszlopok tömörítésével, melyről a 4. fejezetben lesz szó.

Mint láthatjuk, (változatos) lekérdezések esetén több előnnyel jár az oszlopalapú RDBMS-ek használata, azonban a írási és módosítási műveletek elvégzésében óriási hátrányt szenvednek. Soralapú szervezéssel sokkal könnyebben elvégezhetők ezek a műveletek. például egy telekommunikációs szolgáltatónál egy új ügyfél felvétele egy blokkművelettel is elvégezhető (az egyszerűség érdekében maradjunk a redundáns dimenziós adatmodellnél). Ezzel szemben egy oszlopalapú RDBMS-nek minden oszlopot fel kell olvasnia és a megfelelő rekordazonosító mellé felvenni az új értékeket – ezesetben – ha még el is fér egy oszlop egy blokkban – akár száz blokkműveletre is szükség lehet.

2.4 Materializáció [3][4]

Az oszlopalapú RDBMS-ek hagyományostól eltérő fizikai szervezése újfajta megközelítéseket igényel a lekérdezések kiértékelésénél is a rendszerekben rejlő lehetőségek minél mélyebb kiaknázásának érdekében.

Mivel az oszlopalapú adatbázis-kezelők felhasználói – sőt, akár fejlesztői – számára az adatok szervezése transzparens, illetve mivel egy

1000	1001	1002	1003

adathalmazzal vajmi keveset lehet kezdeni a

1000	Klemens	42	Stuttgart

helyett, ezért a lekérdezések végrehajtásakor a táblák „újraegyesítésének” (rekord azonosító mentén történő illesztés) a végrehajtására is szükség van – ezt a műveletet *materializáció*nak nevezzük.

A materializációnak az alábbi két típusa van:

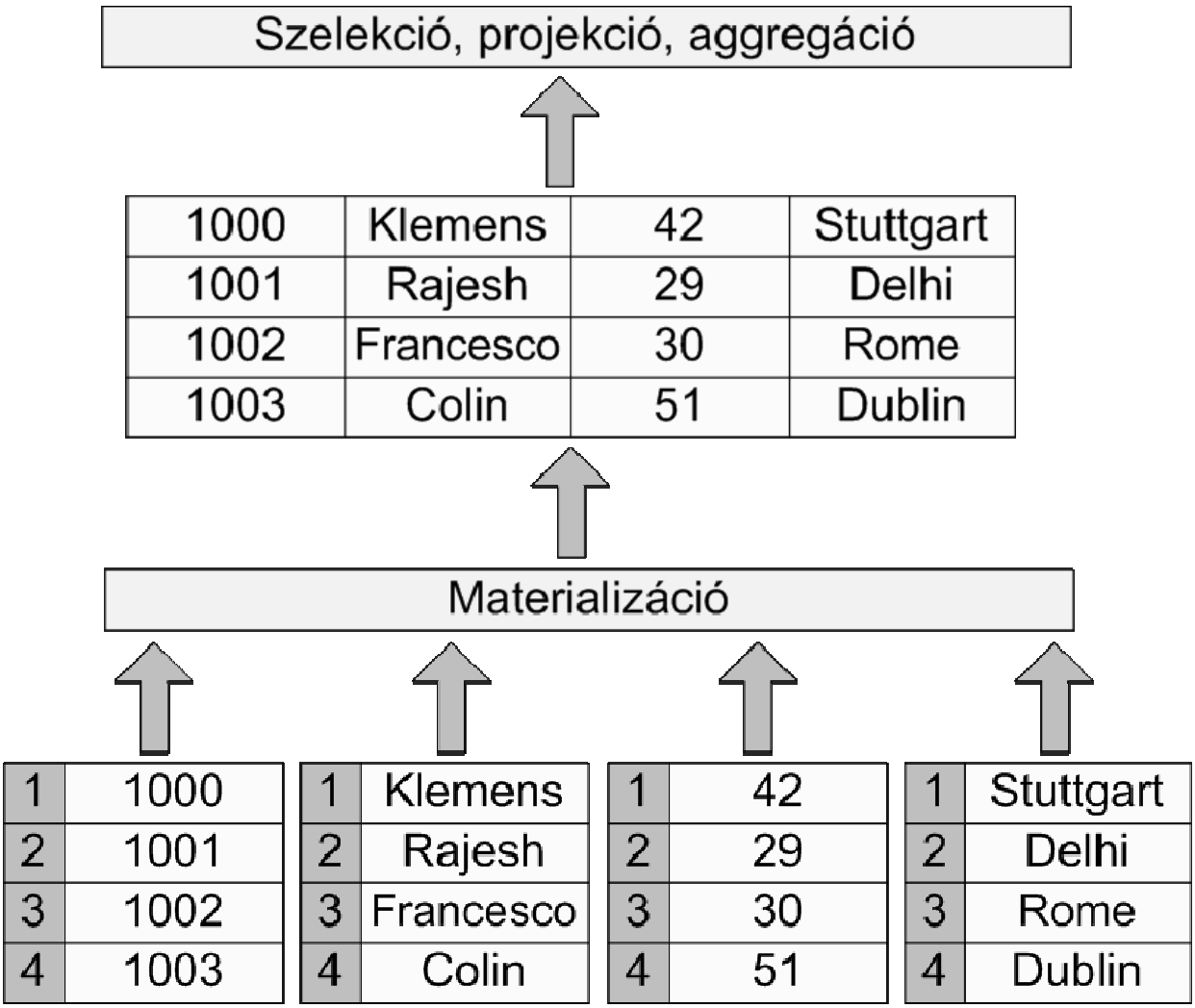
a) korai materializáció (*early materialization*)
b) kései materializáció (*late materialization*).

A korai materializáció során első lépésben minden érintett tábla materializációja megtörténik (az lekérdezésben érintett oszlopokkal), majd ezeken a teljes táblákon kerülnek végrehajtásra a lekérdezések. Kései materializáció során minden tábla csak akkor kerül egyesítésre, amikor feltétlenül szükséges.

A korai illetve késői materializációra a 2.2 fejezetben ismertetett reláción mutatom be, melyen az alábbi lekérdezést (legöregebb nem stuttgarti lakos 1003-nál alacsonyabb azonosítóval és annak életkora) hajtjuk végre:

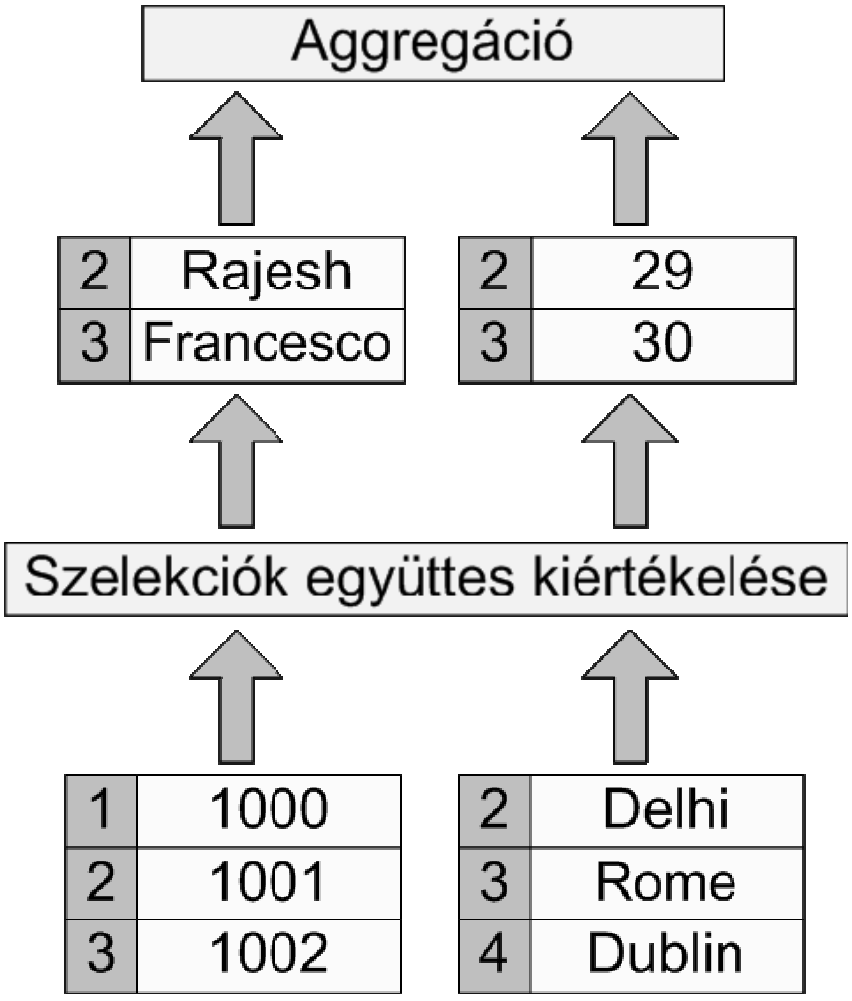
```
SELECT name, MAX(age)
FROM person
WHERE city <> 'Stuttgart'
AND id < 1003
GROUP BY name;.
```

A lekérdezés korai materializáció esetén a Ábra 2. ábrán látható módon kerül kiértékelésre.



Ábra 2 Korai materializáció

Késői materializáció esetén (ld. Ábra 3) a lekérdezés feldolgozása a következőképpen történik: előbb az érintett oszlopokon az egyes szelekciók kerülnek elvégzésre, majd ezek együttes kiértékelése. Ezt követően a projekciók hajtódnak végre, majd az aggregációk. Így a relációknak csak a lekérdezés által érintett oszlopai kerülnek beolvasásra, és csak abban az esetben, amikor arra szükség van. Így például a neveket és az életkorokat csak a szelekciók végrehajtása után olvassuk be. A végrehajtás során jellemzően sor azonosítókkal térnek vissza az egyes lépések – éppen ezért a példában is így láthatók az oszlopok (megj.: a feldolgozás tovább javítható pl. bitmap indexek használatával).



Ábra 3 Késői materializáció

Bár az előbb láthatott példáinkban a blokkműveletek számát tekintve nem sok különbség van a két materializációs módszer között, belátható, hogy nagyobb oszlopszámú táblák, illetve magas projektivitás és/vagy szelektivitás esetén a késői materializáció sokkal előnyösebb. Illesztéseket

igénylő lekérdezések esetén ez közel sem ilyen egyértelmű, azonban néhány, a hatékonyságot növelő módszer (pl. *invisible join* [2]) alkalmazásával a késői materializáció alkalmazása ugyancsak üdvözítőbb.

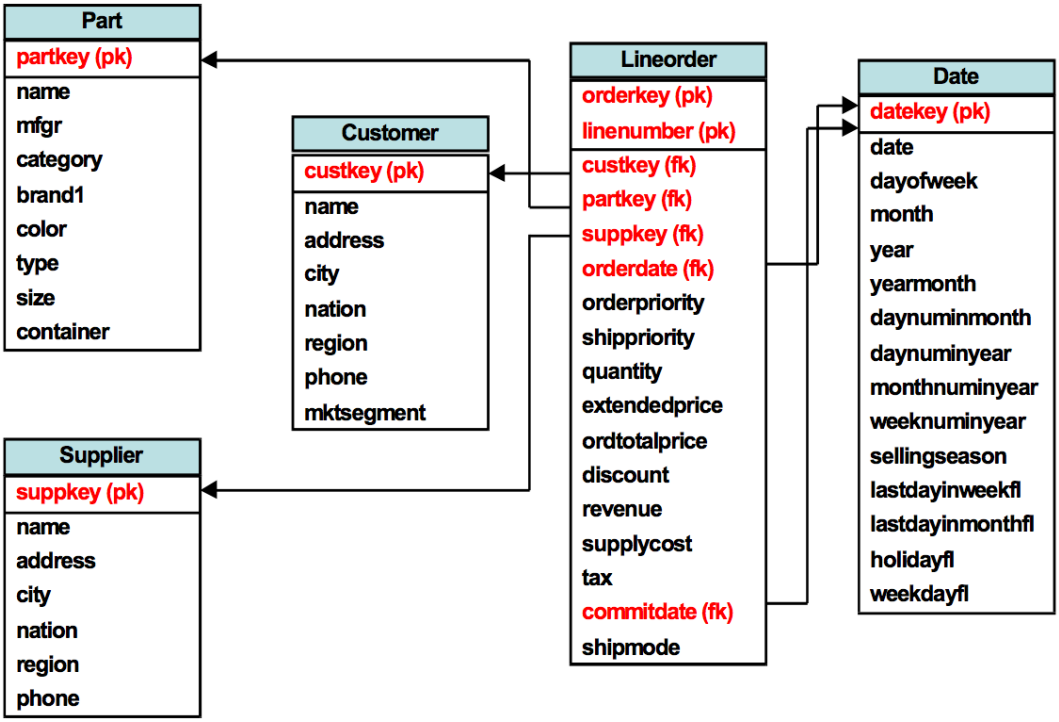
3 Teljesítmény

Az eddigiekben láthattuk, hogy az oszlopalapú RDBMS-ek bizonyos alkalmazási területeken sokkal hatékonyabbak mint soralapú társaik. Az alábbiakban számszerűen is láthajuk, hogy a különleges szervezési módszer milyen előnyökkel jár.

A TPC (Transaction Processing Performance Council) nonprofit szervezet az 1980-as évek óta foglalkozik adatbázis-kezelők objektív teljesítménymérésének (*benchmark*) kidolgozásával és alkalmazásával. Jelenleg három benchmarkjuk van, ezek közül a TPC-C és a TPC-E tranzakciós, a TPC-H pedig analitikus rendszerek teljesítményét méri [5]. A benchmarkok alapvetően adatbázis sémák definícióiból és a rajtuk értelmezett műveletekből állnak.

A következőkben bemutatandó mérési eredményeket a TPC-H egy egyszerűsített változatán készítették Michael Stonebreaker és társai 22[6]. A mérések során egy négymagos Opteron szerveren futó Vertica oszlopalapú adatbázis-kezelőt és egy közelről meg nem nevezett soralapú RDBMS teljesítményét hasonlították össze (*megj*.: az adatbázis-kezelők gyártói rendszerint szigorú licensz szerződéseikben azt is kikötik, hogy rendszerük más adatbázis-kezelőkkel való összehasonlítása nem publikálható).

A benchmark a Ábra 4. ábrán látható csillagsémára épült, mely rendelések adatait tartalmazza idő, szállító, megrendelő, valamint áru dimenziók szerint.



Ábra 4 TPC-H benchmark csillagséma

Stonebreaker-ék a következő, az adatbázisokon végrehajtott lekérdezéseket ismertették részletesen (összesen 12 lekérdezésből áll a benchmark):

1. Q1 (1. lekérdezés): Adott évben az 1-3%-os kedvezménnyel értékesített, 25 darabnál kisebb vásárlásokból származó bevételek

```
SELECT SUM
(lo_extendedprice*lo_discount) AS revenue
FROM lineorder, dwdate
 WHERE lo_orderdate = d_datekey
AND d_year = 1993
AND lo_discount between
1 and 3
 AND lo_quantity < 25;
```

2. Q5 (5. lekérdezés): Éves bevételek egy adott gyártó termékeire egy adott földrajzi régióból származó szállítóktól

```
SELECT SUM (lo_revenue), d_year, p_brand1
FROM lineorder, dwdate, part, supplier
WHERE lo_orderdate = d_datekey
AND lo_partkey = p_partkey
AND lo_suppkey = s_suppkey
AND p_brand1 between 'MFGR#2221' and
'MFGR#2228' AND s_region = 'ASIA'
GROUP BY d_year, p_brand1 ORDER BY d_year, p_brand1;
```

3. Q8 (8.lekérdezés): Adott években azon ügyfelektől származó bevétel, melyek földrajzi régiója megegyezik az áru szállítójának régiójával, szállító és ügyfél városa szerint csoportosítva, évek és bevételek szerint rendezve

```
SELECT c_city, s_city, d_year, sum(lo_revenue) as revenue
FROM customer, lineorder, supplier, dwdate
WHERE lo_custkey = c_custkey
AND lo_suppkey = s_suppkey
AND lo_orderdate = d_datekey
AND c_nation = 'UNITED STATES'
AND s_nation = 'UNITED STATES'
AND d_year between
1992 and 1997 GROUP BY c_city, s_city, d_year
ORDER BY d_year asc, revenue desc;
```

Az Táblázat 1. táblázatban látható, hogy a Q1, Q5 és Q8 lekérdezések milyen sebességgel hajtódtak végre az egyes adatbázis-kezelőkön, illetve mekkora helyet igényelt az adatbázis tárolása a háttértáron. A csak egy aggregált értéket eredményül adó, illetve néhány erős szelekciós feltételt és egy illesztést tartalmazó Q1 lekérdezés végrehajtása majdnem tízszer annyi időt igényelt a soralapú RDBMS-en az oszlopalapúhoz viszonyítva. A Q5 esetében, ahol három illesztést kellett végrehajtani, kb. azonos mértékű szelekcióval mint a Q1-nél, az oszlopalapú adatbázis-kezelő előnye csökkent ugyan, de még így is tetemes (kb. 8,5-szeres). A Q8 esetében már a soralapú rendszer már nem nagyságrendekkel, hanem csak egy másodperccel lassabb – ez az eredmény azonban az oszlopalapú RDBMS harmadik legjobb eredménye úgy, hogy egyik lekérdezést sem hajtotta végre hamarabb mint oszlopalapú társa. Fontos megemlíteni, hogy a [6] publikációban egyéb, nem részletezett mérések több száz százalékos előnyt is mutattak az oszlopalapú RDBMS-ek javára.

Metrika/DB	Soralapú RDBMS	Oszlopalapú RDBMS
Q1 (sec)	32.0	3.4
Q5 (sec)	26.1	3.2
Q8 (sec)	4.1	3.2
Q1-Q12 átlag (sec)	13.6	1.8
Tárterület (GB)	60.3	36.3

Táblázat 1 Benchmark eredmények [6]

A mérésekből kiderül továbbá, hogy az oszlopalapú RDBMS kb. hétszer gyorsabb soralapú társánál, miközben 40%-kal kevesebb diszkterületet igényel az adatok tárolására.

Stonebreaker-ék mérései tehát egyértelműen az oszlopalapú rendszerek fölényét mutatták. A képet azonban árnyalja a TPC honlapján [7] található összehasonlítás, ahol komplett hardver/szoftver csomagok (*appliance*-ek) teljesítményének összevetése található meg a TPC-H benchmark alapján, az adatbázisok mérete szerinti (100 GB-30 TB) csoportosításban. Mivel nem azonos hardveren, nem azonos operációs rendszeren történtek az itt látható mérések, illetve egyes neves szállítók (pl. Teradata, Vertica) rendszerei nem találhatók meg a listán, ezért fenntartásokkal lehet csak kezelni az itt található adatokat. Mindezek mellett elmondható, hogy kisebb (1 TB vagy az alatti) adatbázis méretek esetén a soralapú rendszerek (elsősorban EXASol és VectorWise) az oszlopalapúaknál jobban teljesítenek. Érdekesség, hogy a Sybase IQ, az oszlopalapú adatbázis-kezelők úttörője a listák végén – például a Microsoft MS SQL Server 2005 és az Oracle 11g mögött – található annak ellenére, hogy közel sem a leggyengébb szerveren, illetve legelavultabb operációs rendszeren futott a tesztek során.

4 Tömörítési módszerek [2][8][9][10]

Egyes szakértők szerint [2] az oszlopok adatai átlagosan háromszor-négyszer jobban tömöríthetők mint a soroké. Ezt a lehetőséget kihasználva alkalmaznak az oszloporientált adatbázis-kezelőkben különböző, az alábbiakban a teljesség igénye nélkül ismertetendő tömörítési technikákat.

A jobb tömörítési lehetőségek oka az adatok doménjeiben keresendők. Vegyük példaként a már jól ismert relációs sémánkat és egészítsük ki a "nem" és a "foglalkozás" oszlopokkal és egy új sorral!

ID	Keresztnév	Életkor	Lakhely	Nem	Foglalkozás
1000	Klemens	42	Stuttgart	Férfi	informatikus
1001	Rajesh	29	Delhi	Férfi	informatikus
1002	Francesco	30	Rome	Férfi	labdarúgó
1003	Colin	51	Dublin	Férfi	kocsmáros
1004	Mária	46	Budapest	Nő	informatikus

Ha a személy(ID, keresztnév, életkor, lakhely, nem, foglalkozás) sémára illeszkedő fenti reláció egy sorát szeretnénk tömöríteni, nehezen tudnánk azt hatékonyan megtenni. Ezzel szemben a "nem" oszlop könnyedén tömöríthető egy biten is.

4.1 Futamhossz kódolás (run-length encoding)

A futamhossz-kódolás egy széles körben alkalmazott, egyszerű tömörítési módszer. Lényege, hogy egy adott értékhez eltároljuk, hogy az mely tartományban érvényes – oszlopalapú RDBMS-ekben a tartományokat a sor azonosítókkal jelölhetjük ki. Például a személy reláció "nem" oszlopa a következőképpen tömöríthető futamhossz kódolással:

Nem
(Férfi, 1,4)
(Nő, 5, 5)

A módszer annál hatékonyabb, mennél kisebb egy attribútum kardinalitása – ennek megfelelően az ID, keresztnév, életkor és lakhely oszlopok futamhossz kódolása nem járna előnyökkel a személy reláción.

4.2 Szótáralapú kódolás

A szótáralapú kódolás során egy oszlop értékeit egy ún. szótárba (*dictionary*) szervezzük, ahol egy szótár bejegyzésnek az oszlop egy értékét feleltetjük meg. A módszer

hatékonysága elsősorban abban rejlik, hogy a változó hosszúságú értékek hosszát korlátok közé tudjuk szorítani. Példaként lássuk a személy reláció foglalkozás oszlopának szótár alapú tömörítését!

Szótár:

Kód	Leírás
1	informatikus
2	labdarúgó
3	kocsmáros

Tömörített oszlop:

Foglalkozás
1
1
2
3
1

4.3 Delta tömörítés

Az eddig ismertetett tömörítési módszerek a tömörített oszlopok viszonylag alacsony kardinalitása mellett voltak jól alkalmazhatók. A delta tömörítés ezzel szemben kiválóan alkalmazható nagy kardinalitású – elsősorban numerikus – oszlopok esetén.

A delta tömörítés lényege, hogy csak néhány értéket tárolunk el az oszlopban, majd az oszlop többi eleme az előttük lévő értékekhez viszonyított (jellemzően pozitív) eltéréseket mutatják meg.

A delta tömörítés jól alkalmazható egy időbélyegeket vagy egyedi azonosítókat tartalmazó oszlopban. Az alábbiakban a személy reláció életkor oszlopának delta tömörítését láthatjuk.

Életkor
42
29
1
21
46

4.4 Lempel-Ziv-Welch (LZW) tömörítés

A Lempel-Ziv-Welch tömörítés stringek tömörítésére alkalmazható. Az algoritmus elsőként felépít egy szótárat (ld. 4.2) minden lehetséges bemeneti karakterből (praktikusan betűből), majd elkezdi beolvasni a kódolandó sztringet.

Az algoritmus működése a következő:

INPUT: S sztring, s rész-sztringje S-nek

1. Szótár inicializálása a lehetséges karakterekkel

2. Leghosszabb, a szótárban megtalálható s megkeresése S-ben
3. s-nek megfelelő kód kiolvasása a szótárból, kiadása a kimenetre
4. s és az azt követő karakter bejegyzése a szótárba
5. GOTO 2.

Leállási feltétel: nincs több karakter S-ben

A keresési mintát addig bővíti kódolandó sztring soronkövetkező karakterével, amíg olyan sztring részletet (*substring*) nem talál, amely még nincs a szótárban. Ekkor a megtalált sztring részlet utolsó karaktere nélküli sztring részletet (amely a szótárban megtalálható leghosszabb string részlet) kódját a kimenetre adja, egyúttal a megtalált sztring részletet bejegyzi a szótárba. A keresést a bejegyzett string részlet utolsó karakterével folytatja.

Az LZW tömörítésre példaként tekintsük a „TRALALA” szó kódolását! A szótárban az angol ABC betűi vannak incializálva 1-26-ig terjedő kódokkal, az ABC-ben elfoglalt pozíció szerint (pl. B kódja 2). Első lépésben vesszük a T értéket, mely megtalálható a szótárban, a TR érték azonban már nem, ezért a TR-t felvesszük a szótárban, a T-nek megfelelő kódértéket pedig a kimenetre írjuk. Mivel a T már kikerült a kimenetre, ezért R-rel folytatjuk: RA még nincs a szótárban, ezért felvesszük bele, és R kódját pedig a kimenetre írjuk. A következő érdekes lépés az ötödik, ahol már egy két karakterből álló rész-sztringet találunk a szótárban, ezért ezt (AL) a kimenetre írjuk és egy három karakteres rész-sztringet (ALA) veszünk fel a szótárba. Az algoritmus a 6. lépés után áll le, amikor az utolsó karaktert (A) lekódoltuk.

Lépés	Leghosszabb s a szótárban	Következő karakter	Output	Új szótárbejegyzés (kód – érték)	Szótár tartalma
1	T	R	20	100 – TR	1-26 (A-Z)
2	R	A	18	101 – RA	1-26, 100
3	A	L	1	102 – AL	1-26, 100-101
4	L	A	12	103 – LA	1-26, 100-102
5	AL	A	102	104 – ALA	1-26, 100-103
6	A	–	1	-	1-26, 100-104

Táblázat 2 LZW tömörítés

5 Kiegészítő technikák

Az alábbiakban néhány olyan technikáról esik szó, melyek nem feltétlenül egyedi sajátosságai az oszlopalapú adatbázis-kezelőknek, azonban lényegesek a téma tárgyalásakor.

5.1 Elosztott tárolás [1]

Napjaink adatbázis-kezelő rendszereivel szemben számos esetben merül fel az igény az adatok elosztott tárolásának igénye, például geográfiai távolság, párhuzamos feldolgozhatóság vagy az adatok biztonsága miatt. Az elosztott tárolásnak két típusa van:

- replikáció (*replication*)
- sharding.

A replikáció során ugyanazt az adategységet (többszörözve) tárolják több szerveren. Az ún. multi-master replikáció esetén bármelyik szerver kiszolgálhat kéréseket, legyen szó akár írásról, akár olvasásól – utóbbi esetben erről értesíti a többi szervert is.

Sharding ugyanannak az adategységnek több, diszjunkt szeletét (horizontális fragmentáció) tárolják több szerveren. Így például a 2.2 alfejezetben ismertett személy relációt földrészenként eltérő szervereken sharding alkalmazásával tároljuk, akkor az 1000, 1002 és 1003 értékű ID-val rendelkező személyek adatai az európai, az 1001-es ID-val ellátott személyé pedig az ázsiai szerveren lesznek találhatók.

A sharding és a replikáció használható külön-külön és együtt is. Az előbbi példában említett shardingolt adatbázisunkat replikációval egészítjük ki, akkor az európai és ázsiai szervereink mellé még üzembe helyezünk egyet-egyet, melyekre lemásoljuk az eredeti szerveren található adatbázist.

5.2 MPP [12]

Az analitikus rendszerekben található adatok nagyságrendrendekkel való megnövekedése, illetve az azok gyorsabb feldolgozására vonatkozó igény új, az adatbázis-kezelőket futtató szervertechnológiák fejlesztését tette indokolttá.

A hagyományos archikektúrákat (szerver + diszk alrendszer) előbb az egyes komponensek teljesítményének növelésével gyorsították, azonban az I/O ezt egyre kevésbé tudta kiszolgálni. Ezt követően újabb szervereket kapcsoltak a megosztott diszk alrendszerekhez, azonban így a diszkek kihasználtsága szuboptimális lett, a lehetséges

szerverkapcsolatok pedig hamar beteltek. Erre válaszul jelentek meg az ún. MPP (*Massively Parallel Processing,* erőteljesen párhuzamos feldolgozás) rendszerek. Az MPP-k olyan rendszerek, melyek jellemzően a *shared nothing* architektúrájúak, azaz a komponensek között semmilyen erőforrás (CPU, memória, diszk) nincs megosztva. Az MPP rendszerek a hagyományos architekúrákhoz képest több, kompakt komponensből állnak, melyek között az adatátvitel nagy sávszélességű. Az MPP rendszerek skálázhatósága, hibatűrése és diszk kihasználása a többi architektúrához viszonyítva sokkal jobb. Az MPP adatbázis-kezelőket elsősorban ún. *appliance*-ekként, azaz olyan komplett rendszerekként értékesítik, melyekben adatbázis-kezelésre optimalizált hardver és szoftver (operációs rendszer, adatbázis szerver) elemek egyaránt megtalálhatók.

5.3 Bitmap indexek [10]

Az ún. *bitmap* (bittérkép) indexek szinte minden analitikus célra használt adatbázis-kezelőben megtalálhatók, ugyanis nagy mértékben meggyorsítják az alcsony kardinalitású attribútumokon végzett szelekciók kiértékelését.

A bitmap indexek olyan bitvektorok, melyek megmutatják, hogy egy adott értéket a tábla mely sorain vesz fel (az érték felvételének helyén 1-et, különben 0-t vesz fel).

Példaként tekintsünk egy, a személy(név, életkor, nem, foglalkozás) sémára illeszkedő relációt, melynek nem és foglalkozás oszlopaira bitmap indexet építünk!

Keresztnév	Életkor	Nem		Foglalkozás		
		Férfi	Nő	informatikus	labdarúgó	kocsmáros
Klemens	42	1	0	1	0	0
Rajesh	29	1	0	1	0	0
Francesco	30	1	0	0	1	0
Colin	51	1	0	0	0	1
Mária	46	0	1	1	0	0

A bitmap indexelt oszlopokon a szelekciók kiértékelése során az egyes szelekciós feltételek közti logikai műveleteket (pl. és, vagy) végezzük el.

Például, ha a fenti relációból szeretnénk megkapni a női informatikusokat, akkor az alábbiak szerint tudjuk alkalmazni a bitmap indexeket, hogy megtudjuk, Mária az egyetlen női informatikus a relációban.

Projekció/ Szelekció	**Klemens**	**Rajesh**	**Francesco**	**Colin**	**Mária**
Nő	0	0	0	0	1
Informatikus	1	1	0	0	1
ÉS	**0**	**0**	**0**	**0**	**1**

5.4 Sűrű csomagolás

Az oszlopalapú adatbázis-kezelőknél természetesnek vettük [1], hogy a rekordok sosem lépik át a fizikai blokkok határát, azaz egy rekord minden attribútumának értéke egy blokkon belül található. Mivel az oszlopalapú adatbázis-kezelők sajátságos fizikai szervezése ezt nem teszi lehetővé, ezért számos oszlopalapú rendszerben ezt az elvet figyelmen kívül hagyják és a blokkokat az utolsó bitig kihasználva tárolják az adatokat – ez a módszer az ún. sűrű csomagolás (*dense packing*) [2][10].

5.5 Párhuzamos töltés és lekérdezés [13]

A 3. fejezetben már láthattuk, hogy az oszlopalapú RDBMS-ek az olvasás-intenzív műveleteket jellemzően sokkal gyorsabban végzik el, mint soralapú társaik.

A írás-intenzív műveletek esetén azonban komoly hátrányban vannak az oszlopalapú adatbázis-kezelők. Mennél több oszlopból áll egy tábla, annál több blokkműveletet kell végeznie az oszlopalapú rendszernek. Így könnyen elképzelhető, hogy amíg egy közel száz oszlopból álló dimenzió tábla egy rekordját egy soralapú adatbázis-kezelő egy blokkművelettel beírja, addig az oszlopalapúnak erre legalább annyi blokkműveletre van szüksége, ahány oszlopa van a táblának.

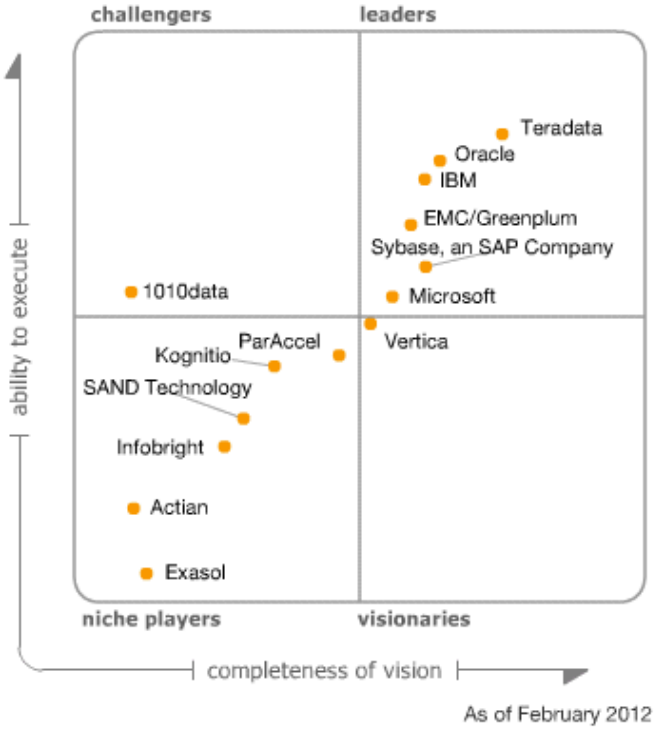
A fent vázolt hátrány kiküszöbölésére alkotta meg a Vertica azt a folyamatos töltés és lekérdezés (*continous load and query*) módszerét, mely arra alapul, hogy az írási és olvasási műveleteket két külön tárolón hajtják végre (megj.: a módszert később több gyártó, pl. az ExaSol is alkalmazta). A betöltések a WOS-ra (*write optimized storage*) történnek, a lekérdezések kiszolgálása pedig a ROS-on (*read optimized storage*) megy végbe. A két tároló között az ún. *tuple mover* (rekord mozgató) végzi az áttöltéseket aszinkron módon. A WOS terület memóriaalapú, nem rendezett, nem tömörített módon tartalmazza az adatokat, ezzel is csökkentve a késleltetést. A ROS ezzel szemben a már bemutatott módszerekkel (pl. tömörítés) támogatva, oszlopok szerint szervezve tárolja az

adatokat. A WOS méreténél nagyobb adategységeket természetesen lehetőség van közvetlenül a ROS-ba tölteni.

6 Piaci kitekintés

Az alábbiakban egy rövid kitekintést teszünk az oszlopalapú adatbázis-kezelők piacára, melyen között az elsősorban egyetemi kutatási témaként funkcionáló rendszertől a több évtizedes múltra visszatekintő, piacvezetőig számos szereplő megtalálható.

A világ vezető technológiai elemző cége, a Gartner rendszeresen kiadja a különböző technológiai területek *magic quadrant*-ját (mágikus négyszög), melyben az adott piacon versenyző vállalatok találhatók meg négy csoportba (kispályások, látnokok, kihívók és vezetők) osztva piaci helyezetük alapján [11]. Az adattárházak (az oszlopalapú RDBMS-ekre nem készül önálló magic quadrant, az ilyen rendszereket elsősorban adattárházak megvalósítására használják) 2012-es magic quadrantja az Ábra 5. ábrán látható.



Ábra 5 Adattárházak magic quadrantja [16]

A tisztán oszlopalapú adatbázis-kezelők rendszereket készítő vállalatok között elsősorban a Stonebreaker fennhatósága alatt megalkotott Vertica-t, illetve az első professzionális felhasználásra szánt oszlopalapú RDBMS-t, az IQ-t 1994-ben kiadó, az SAP által

tulajdonolt Sybase-t kell megemlíteni. A Hewlett-Packard által 2011-ben felvásárolt Vertica több mint 500 ügyfele a hibrid tárolás (oszlopszinten megadható, hogy sor- vagy oszlopalapú szervezéssel szeretnénk tárolni) lehetőségeit és a Vertica appliance jó tulajdonságait emelte ki előnyökként. A Sybase-nek a kb. 10 évvel nagyobb tapasztalata (is) eredményezi, hogy több ezer ügyféllel rendelkeznek világszerte, mellyel a legelterjedtebb tisztán oszloporientált rendszernek számít az IQ. A Sybase jövője azonban közel sem biztos, mivel az SAP a saját feljesztésű Business Warehouse (BW) és HANA rendszereit részesítheti előnyben.

További oszlopalapú rendszereket fejlesztő vállalatok a mindössze negyven ügyféllel, de számos nagy nevű partnerrel (pl. Amazon) rendelkező ParaAccel, illetve a TPC benchmarkokon a rangsorok elején szereplő VectorWise-t készítő Actian és ExaSol. Az ExaSol adatbázis-kezelője egy oszlop- és memóriaalapúm, elosztottan működő rendszer, melyhez önálló operációs rendszert is kínál a vállalat.

Mivel napjainkra (2012) már a piacvezető soralapú RDBMS-ekben – így például az Oracle 11gR2-ben és a Microsoft SQL Server 2012-ben is is megjelent az oszlopalapú szervezés mint opció, ezért ezeket a gyártókat is lényeges megemlíteni, különösen, hogy ezen a területen az Oracle rendelkezik a legtöbb, több mint 300000 ügyféllel. A Gartner magic quadrant legjobb pozíciójában található Teradata nem véletlenül érdemelte ki ezt a címet: több mint 30 éves tapasztalatuk van döntéstámogatási felhasználású adatbázis-kezelők fejlesztésében, illetve elsőként tették lehetővé rendszerükben egy táblán belül egyszerre az oszlop- és soralapú tárolást [14].

Egyes vállalatok, mint a SAND Technologies és a 1010 Data a fellendülőben lévő felhőszolgáltatásokban (SaaS, Software as a Service) érdekeltek a saját analitikus adatbázis-kezelő rendszereikkel.

Az oszlopalapú adatbázisok elméletének és fejlődésének szempontjából lényeges megemlíteni a MonetDB, illetve az ugyancsak Stonebreaker-illetőségű C-Store rendszereket [2][10].

7 Források

[1] Gajdos Sándor – Adatbázisok, 2012.

[2] Abadi, Boncz. Harizopoulos – Column-oriented database systems URL: http://www.vldb.org/pvldb/2/vldb09-tutorial6.pdf és http://cs-www.cs.yale.edu/homes/dna/talks/Column_Store_Tutorial_VLDB09.pdf hozzáférés ideje: 2012. 10. 05.

[3] Abadi, Myers, DeWitt, Madden – Materialization strategies in a column-oriented DBMS URL: http://db.lcs.mit.edu/projects/cstore/abadiicde2007.pdf hozzáférés ideje: 2012.10.03.

[4] Abadi, Madden, Hachem: Column-stores vs. row-stores: How different are they really? URL: http://db.csail.mit.edu/projects/cstore/abadi-sigmod08.pdf hozzáférés ideje: 2012.10.03.

[5] TPC Benchmark H URL: http://www.tpc.org/tpch/spec/tpch2.14.4.pdf , hozzáférés ideje: 2012. 10. 05.

[6] Stonebreaker et al. – One size fits all – Part 2. Benchmarking results URL: http://www.cs.brown.edu/~ugur/osfa.pdf hozzáférés ideje: 2012. 10. 05.

[7] TPC Top Ten Performance Results URL: http://www.tpc.org/tpch/results/tpch_perf_results.asp hozzáférés ideje: 2012. 10. 05.

[8] Goldstein et al. – Compressing relations and indexes URL: http://www.cs.brown.edu/courses/cs227/archives/2008/Papers/Compression/Ramakrishnan.pdf, hozzáférés ideje: 2012. 10. 05.

[9] Paraccel - A technical overview URL: http://www.paraccel.com/resources/Whitepapers/ParAccel-Technical-Overview-White-Paper.pdf

[10] Stonebreaker et. al. - C-Store: A column-oriented DBMS URL:http://people.csail.mit.edu/tdanford/6830papers/stonebraker-cstore.pdf

[11] Gartner: Magic Quadrant for Data Warehouse Database Management Systems URL: http://www.gartner.com/technology/reprints.do?id=1-196T8S5&ct=120207&st=sb hozzáférés ideje: 2012. 10. 07.

[12] Dési Balázs – MPP adattárházak, URL: https://www.db.bme.hu/sites/default/files/mpp.adattarhazak_desi.pdf, hozzáférve: 2010. április

[13] Vertica Architecture Whitepaper, URL: http://184.106.12.19/wp-content/uploads/2011/01/VerticaArchitectureWhitePaper.pdf, hozzáférés ideje: 2012.10.07.

[14] McKnight - Introducing Teradata Columnar URL: http://tinyurl.com/HDB-TERA

[15] Greenplum – Polymorphic data storage URL: http://www.greenplum.com/technology/polymorphic-data-storage , hozzáférés ideje: 2012. 10. 05.

[16] Magic Quadrant for data warehouses URL: http://imagesrv.gartner.com/reprints/219200/219281/219281_1.png