Balogh Bernadett - YTJ7JD
Cseh Tamás - H70JTT
Kresz Marcell - HSU5NP

Automatikus fizikai tervezési javaslatok XML adatbázisokhoz

1. Rövid összefoglalás

Az XML alapú adatbázisrendszerek egyre nagyobb teret hódítanak az adatok tárolásánál, ezt mutatja az is, hogy az XML formátum támogatása napjainkra megjelent az ismertebb adatbáziskezelő rendszereknél.

Az adatbázisok tervezése során nem elhanyagolható annak vizsgálata, hogy milyen lépéseket tehetünk annak érdekében, hogy a jövőbeni lekérdezéseket minél optimálisabban, gyorsabban tudjuk végrehajtani. Az optimalizáció fontos kellékei többek között az indexek és a nézetek.

A cikk témája az XML adatbázisok fizikai felépítésére vonatkozó automatikus javaslatok készítése, azaz a legmegfelelőbb teljesítményjavító struktúrák kiválasztása egy megadott adatbázishoz és a hozzá tartozó lekérdezésekhez.

Két fizikai struktúra használatát vizsgáljuk a lekérdezések gyorsítására, az XML indexekét és az XML adatokat tároló relációs nézetekét. Ezen struktúrák ajánlására készítünk több Tancsádaó megoldást is.

Három tervezési tanácsadót (design advisor) mutatunk be:

- XML indexekhez,
- materializált nézetekhez,
- az előző kettőt egybe integrálót.

Ezek a tanácsadók szorosan összekapcsolódnak az adatbáziskezelő lekérdezés optimalizálójával és támaszkodnak annak működésére.

2. Bevezetés

2.1. XML adatok tárolása

Napjainkban egyre nagyobb mértékű az adatok XML formátumban történő továbbítása, tárolása. Ezáltal az XML adattárolás egyre hangsúlyosabb szerepet kap. Három elterjedt tárolási mód létezik, ezek:

- Natív XML adatbázisok
- Az XML dokumentum relációs adatbázisban való tárolása

- XML oszlop típus

Ebben a cikkben az adatokat XML típusú oszlopokban tároló, relációs tárolási móddal foglalkozunk.

2.2. Lekérdezések végrehajtásának optimalizálása

XML adatbázisok esetén az alábbi módszerekkel tudjuk a lekérdezések végrehajtását gyorsítani:

- közvetlen elérés biztosítása az XML dokumentum bizonyos adataihoz anélkül, hogy az egész dokumentumot végig kellene olvasni (pl. indexek)
- az adatok bizonyos részének egy logikai egységbe való csoportosítása, amelyben az eredeti dokumentumtól függetlenül kereshetünk (nézetek, partícionálás)
- a lekérdezések átírása kisebb adathalmazra

Ezek használata növelheti az adatbázis teljesítményét, azonban ehhez a készítőnek el kell tudnia dönteni, hogy egy adott adatbázishoz és lekérdezésekhez mely módszerek működnek megfelelően. Ebben nyújt segítséget a Tervezési Tanácsadó (Design Advisor), amely automatikusan ajánlja a megfelelő fizikai tervet.

2.3. Fizikai tervezés általánosan

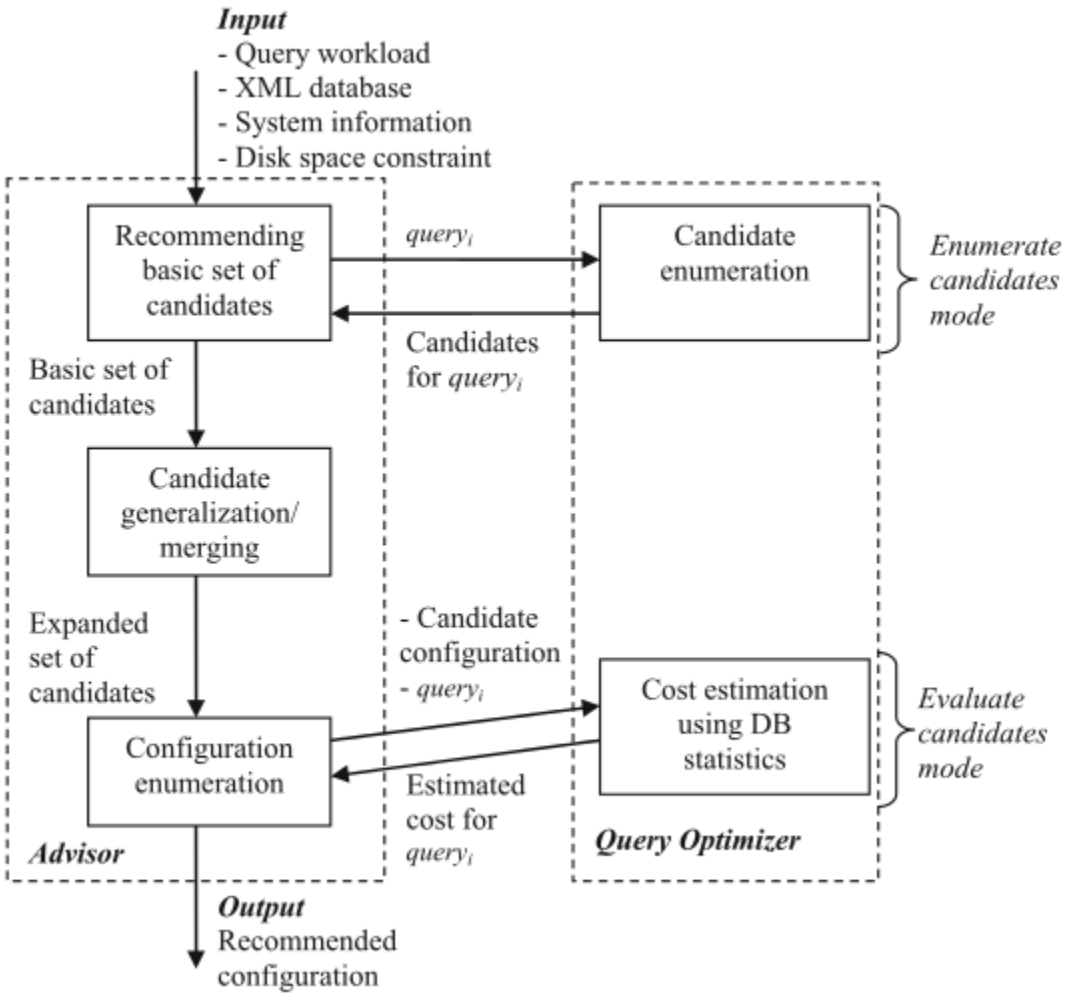
Egy programnak, mely javaslatokat ad a fizikai adattárolás struktúrájára alapvetően négy kérdést kell megválaszolnia, melyek a következők:

1. Hogyan állapítsuk meg a lehetséges struktúrákat, melyek hasznosak a lekérdezés halmazon?
2. Hogyan terjesszük ki a jelölteket általánosabb javaslatokra?
3. Hogyan számítsuk ki a becsült költségét a fizikai tervnek?
4. Hogyan válasszuk ki a lehetséges konfigurációk közül a legjobbat?

A lekérdezés optimalizáló két móddal egészül ki. Az első olyan fizikai struktúrákat ajánl, amely egy adott lekérdezéshez ajánl megoldásokat, míg a második megbecsüli a lekérdezés költségét.

Az XML és relációs adatbázisok fizikai struktúrái különböznek, amely olyan problémákat vet fel, mint például az indexek mintáinak felismerése egy XML adatra, ezeknek általános

formára hozatala, valamint nagy mennyiségű lehetséges minták keresése. A felsoroltak jelentősen megnehezítik az fizikai struktúra automatizált tervezését a relációs adatbázisokhoz képest.



Kép 1. Tervezési Tanácsadó általános szerkezete

3. Kapcsolódó munkák

Két korábbi munka jelent meg az XML adatbázisok automatikus index ajánlásával kapcsolatban [5, 6]. Mindkettőben kezdetleges technikát alkalmaztak a javasolt indexek előállítására, a költség becslésre és a javasolt indexeket tartalmazó konfigurációk felsorolására. Ezekben a munkákban nem támaszkodtak az adatbázis lekérdezés optimalizálójára, azaz nem biztosított, hogy az ajánlott indexeket az optimalizáló használni is fogja, illetve hogy a költségek az adott optimalizáló futása során azonosak lesznek a becslésekkel.

A [6] tanulmányban egy olyan eszközt készítettek, mely egy XML adatbázishoz indexeket javasolt. A hangsúly a jó költség modell megtaláláson volt, melyeket a strukturális információk és adatbázis statisztikák alapján készítettek el. Az általunk készített tanulmány ezen a ponton gyökeresen eltér az imént említettől, ugyanis mi a lekérdezés optimalizáló költségmodelljét használjuk ugyanerre a célra.

Egy másik index ajánlót mutattak be a [4, 5] munkákban. Nagy különbség az általunk használt megoldáshoz képest az, hogy ez a tanácsadó periodikusan elemezte a bejövő lekérdezéseket és az XML indexeket működés közben cserélte és módosította. A költségmodell ebben a megoldásban is független volt a lekérdezés optimalizálótól.

Ezekben a munkákban a költségszámításoknál nem vették figyelembe a frissítő, törlő és beszúró műveletek utáni index karbantartási költségeket, így a kapott eredmény nem tükrözi az indexek használatával járó költségkülönbségeket.

4. Alapfogalmak

4.1. XPath

Olyan lekérdező nyelv, amely XML dokumentum csomópontjainak kiválasztását teszi lehetővé.

4.2. XQuery

XML dokumentumok hatékony lekérdezését szolgálja. Be- és kimeneti adata is XML formátumú, nem úgy, mint a parancsok. Az egyes csomópontok kiválasztását XPath kifejezésekkel írhatjuk le.

4.3. SQL/XML

Az SQL nyelv egy olyan kiegészítése, amely XML dokumentumokra is kiterjeszti a lekérdezés lehetőségét. Ezzel megjelent az XML típus, kapcsolódó eljárások és funkciók, valamint az XML és SQL közötti megfeleltetések, támogatva az XML adat tárolását az adatbázisban.

4.4. XML index

Az XML lekérdező nyelvek XPath útvonal kifejezéseket használnak a dokumentum elemeinek hivatkozására. Az XML dokumentum elemeinek lekérdezését segíthetik az XML indexek. Ezeket kétféleképpen csoportosíthatjuk: strukturált és érték index (structural, value index). Előbbi felgyorsítja a navigációt a hierarchikusan felépített XML adatban, míg utóbbi segít az

XML elemek kinyerésében valamilyen feltétel alapján, az elem értékétől függően. Az indexek tárolási mérete akkorára nőhet, mint maga az eredeti adatbázis, ezért nem biztos, hogy gyorsítják a lekérdezést, valamint ezek karbantartása is költséges lehet. Parciális indexek használatával ez elkerülhető. Ezek csak azokat az XML elemeket tartalmazzák, amelyek elérhetők valamilyen index minta alapján. A minták általában lineáris XPath kifejezések.

4.5. Materializált nézet (materialized view)

Olyan adatbázis objektum, amely egy lekérdezés eredményét tárolja. Ezzel a gyakran használt, költséges lekérdezéseket nem kell újra és újra kiszámolni. Fizikai megközelítésben a materializált nézetek olyanok akár a táblák, ugyanis saját adatszegmenssel rendelkeznek, azonban mutatnak hasonlóságot a nézetekkel is, hiszen valamilyen előre megadott lekérdezés eredményét tárolja, és mutatja a felhasználó felé.

4.6. XMLTable nézetek

Fizikai szinten az XML adatok materializált nézetei az alábbi alakokban fordulhatnak elő:

- a nézet és a lekérdezés nyelve is XQuery,
- a nézet nyelve XPath és a lekérdezés nyelve XQuery,
- a nézet nyelve az XPath és SQL kombinációja, a lekérdezés nyelve XQuery.

A relációs materializált nézetek használata egyszerű és hatékony módja, hogy javítsunk az XML adatokon történő lekérdezések gyorsaságán. Ilyen nézeteket az XMLTable függvénnyel készíthetünk, amelynek szintaxisa az alábbi:

- sorgenerátor (row generator), amely XQuery kifejezés,
- oszlop navigátor (column navigator), amely XPath kifejezés.

Az XMLTable függvény egy relációs táblával tér vissza, abból készítünk nézetet.

4.7. Parciális index

Az adathalmaz egy részén értelmezett index, amely kisebb tárolási mérethez vezet. A karbantartása ezzel hatékonyabb és jelentősen felgyorsítja a keresést azokhoz az indexekhez képest, amelyek a teljes adathalmazon értelmezettek.

4.8. Virtuális index

A lekérdezések optimalizálása során gyakran előfordul, hogy különböző indexeket kell kipróbálni a végrehajtási tervekhez. Indexek hozzáadásával csökkenne a sebesség az index létrehozása miatt, valamint nőne a tárolási méret. Ezzel szemben a virtuális indexekenek nincs külön szegmense, ezért sem a létrehozás ideje, sem a tárolási mérete nem befolyásolja a becslést.

4.9. 0/1 hátizsák probléma

A hátizsák probléma röviden a következőképpen fogalmazható meg: adott egy hátizsák, melynek ismerjük a tárolókapacitását, illetve vannak elemek, amikhez méretet és értéket rendelünk. Feladatunk hogy megadjuk az egyes elemekhez, hogy mennyit tudunk belőlük a hátizsákba tenni úgy, hogy a hátizsákban lévő elemek értékének összege maximális legyen és a bent lévő tárgyak méretösszege pedig a hátizsák mérete alatt maradjon (azaz beférjenek a zsákba).

A probléma egy specializációja a 0/1 hátizsák probléma, ekkor minden elem maximum egyszer kerülhet be a zsákba.

4.10. Index illesztés

A lekérdezés optimalizáló egy funkciója, melynek segítségével eldönti egy lekérdezéshez, hogy mely indexek használhatóak fel a lekérdezés gyorsabbá tételéhez.

4.11. Tervezési Tanácsadó (Design Advisor)

Feladata hogy egy adott adatbázishoz és a hozzá tartozó lekérdezéskhez automatikusan fizikai struktúrákat ajánljon, melyek segítségével a lekérdezés kiértékelő működését gyorsíthatjuk.

5.Eredmények

A cikkben három Tervezési Tanácsadót mutatunk be, melyek feladatai: XML indexek ajánlása, XML materializált nézetek ajánlása és ezen kettő együttes ajánlása.

5.1. XML indexek ajánlása

Célünk egy Tervezési Tanácsadó létrehozása, amely automatikusan a legjobb rész-index minták ajánlását végzi el a megadott adatbázishoz és lekérdezéshalmazhoz, figyelembe véve a létrehozott indexek frissítésével és az adatok változásával járó többletköltségeket.

Az adatbáziskezelő lekérdezés optimalizáló funkcióit két további móddal egészítjük ki:

1. XML index felsorolási mód,
2. XML index kiértékelése mód (itt az optimalizáló szimulálja az index konfigurációt és kiértékeli a lekérdezések költségét e konfiguráció mellett)

5.1.1. Index jelöltek felsorolása

Annak eldöntése, hogy mely indexek vehetnek részt egy lekérdezésben, a lekérdezés optimalizáló implementációjától függ. Emiatt a lehetséges indexek felsorolását az optimalizáló beépített, index illesztő funkciójával szoros összhangban használjuk.

5.1.2. Jelöltek általánosítása

Az optimalizáló által kapott indexek felsorolását követően lekérdezés specifikus indexeket kapunk. A hatékonyabb működés érdekében ezeket az indexeket általánosítjuk, így több lekérdezés kiszolgálására lesznek alkalmasak.

A jelölt általánosító algoritmus feladata, hogy általánosabb index mintákat találjon azáltal, hogy különböző általánosítási szabályokat használ fel. A folyamat addig tart, amíg nem lehet további indexeket generálni a már meglévőkből.

Egyszerre két index mintát (XPath kifejezést) hasonlít össze és megpróbálja az szűkebbet kiterjeszteni a bővebbre, feltéve, hogy van közös részük. Ha az új útvonal nem a gyökér (“/”) kifejezés, akkor felveszi az általánosított index minták közé.

Az útvonal kifejezés mintákat láncolt listaként ábrázoljuk, ahol minden csomópont egy útvonal lépés. Az általánosításhoz a listák elejétől kezdve egyszerre járjuk be az összehasonlítandó listákat és minden lépésben összehasonlítjuk a két listát, ha megegyeznek az adott csomóponton akkor azt bevesszük a generált mintába. Ha különböző csomópontokat találunk, akkor a generált csomóponthoz hozzávesszük a * lépést és nem keresünk tovább.

5.1.3. A költségek becslése

Ahhoz, hogy megtaláljuk a legjobb index konfigurációt, az Index Tanácsadónak meg kell tudnia állapítani az egyes konfigurációkhoz tartozó költségeket.

Itt használjuk az optimalizálóhoz korábban hozzáadott új módot, mely az XML indexek költségének kiértékelését végzi. Ez a mód képes a virtuális indexek alapján kiszámolni a lekérdezések költségét, mindezt az adatbázis statisztikák alapján előállított költség modell segítségével. Az alábbi költségeket számoljuk:

- A konfiguráció használatával járó költség-előnyök számítása (ez röviden a lekérdezés költségének csökkenése az indexek használata által)
- A konfiguráció használata miatt szükséges többlet költségek (UPDATE, DELETE, INSERT utasítások használatával járó karbantartási költségek) számítása

5.1.4. Az optimális konfiguráció megkeresése

A konfigurációk közül azt tekintjük optimálisnak, mely a legnagyobb költség-előnnyel jár úgy, hogy közben a felhasználó által megadott méret korlátot a legjobban kihasználja.

A keresés a 0/1 hátizsák problémával ekvivalens, amely egy NP teljes probléma. Minden index-jelöltnek kiszámoljuk a költségét (amely a mérete a lemezen) és a hozzá tartozó hasznot. A hátizsák probléma legegyszerűbb megoldása a mohó keresés (greedy search). Ez önmagában számunkra nem elég, mert nem veszi figyelembe a kiválasztott indexek közötti interakciót, ezért további heurisztikát építünk a keresésbe. Így elérhetjük azt, hogy annyi indexet használunk magas haszonnal amennyit csak tudunk úgy, hogy közben mindegyiket ténylegesen felhasználjuk az optimalizáció során (azaz nincsenek egymást fedő indexek, amelyeknek használata kizárja egymását).

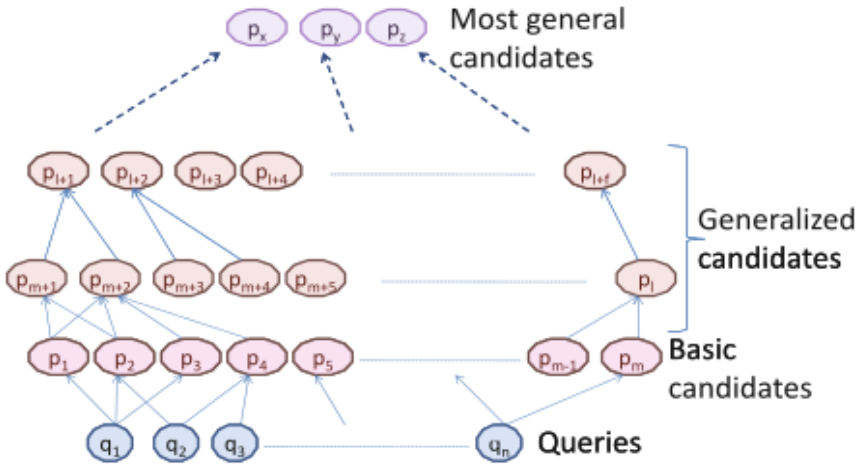
Először becsüljük az egyes indexek méretét illetve a hasznot amit jelentenek. Ezt követően a jelölteket a haszon/méret arányuk alapján rendezzük, csökkenő sorrendbe. Ha az adott index megfelel a heurisztikával járó szabályoknak, akkor bekerül a választott konfigurációba. A heurisztika azt vizsgálja, hogy egy index által és a már választott konfigurációban lévő indexek által lefedett lekérdezések között van-e átfedés. Ha van, és az adott index nagyobb lefedettséggel bír mint a konfigurációban szereplők, akkor ha rendelkezésre áll elég hely, akkor az adott indexre cseréljük a konfigurációt, ha nincs átfedés az indexek között és az új index bővíti a lefedettséget, akkor hozzávesszük a konfigurációhoz.

Algorithm 1 heuristicSearch(*candidates*, *diskSize*)

```
1: sort candidates according to their
   benefit(cand)/cand.size ratio
2: recommended ← ∅, recommended.size ← 0,
   recommended.coverage ← ∅
3: while recommended.size < diskSize do
4:    bestCand ← pick the next best cand in candidates
5:    if recommended.coverage ∩ bestCand.coverage = ϕ
      then
6:       addCandIfSpaceAvail(bestCand,recommended)
7:    else if recommended.coverage ≤ best.coverage then
8:       replaceCandIfSpaceAvail
         (bestCand,recommended,recommended)
9:    else
10:      overlapConfig ←
         overlapCoverage(bestCand, recommended)
11:      replaceCandIfSpaceAvail
         (bestCand,overlapConfig,recommended)
12:   end if
13: end while
14: return recommended
```

Kép 2. Mohó keresés heurisztikával

A mohó keresés eredményeként előáll a legjobb haszonnal járó konfiguráció, ez azonban nem elég általános ahhoz, hogy más lekérdezésekhez is használhassuk. Szükségünk van tehát egy heurisztikában felülről lefelé haladó keresésre, ezzel biztosítva hogy annyi általános indexet választunk ki, amennyire csak lehetőségünk van a hely korláttal. Így általánosabb indexek kerülnek a konfigurációba, amiket a jövőbeni, még nem ismert lekérdezések is tudnak hasznosítani. A keresést egy irányított, aciklikus gráfon hajtjuk végre, ahol a levelek szintjén a lekérdezések szerepelnek, a gyökér felé haladva pedig az egyre általánosabb indexek.



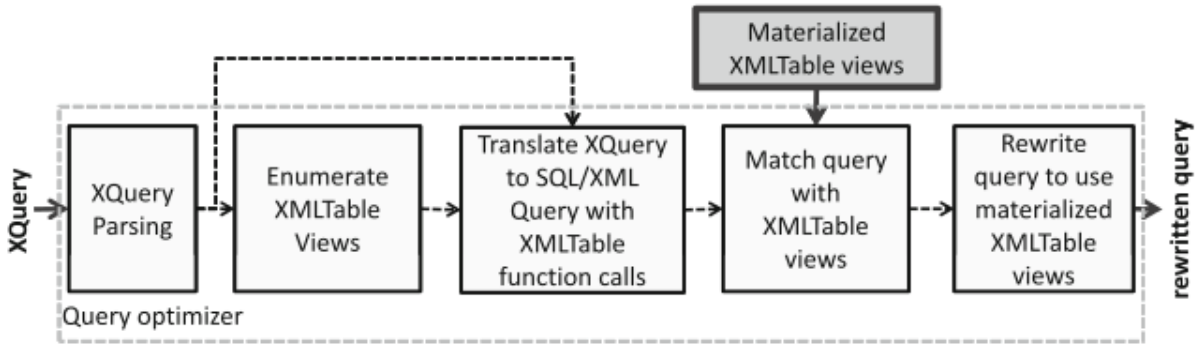
Kép 3. Az index jelöltek és a lekérdezések kapcsolata

A keresés előtt végrehajtunk egy nulladik lépést, ebben eltávolítjuk a gráfból azokat a csúcsokat, amelyek 0 vagy negatív haszonnal járnak. Ezt követően az általános indexeket addig cseréljük le a specifikus gyerek indexeire míg olyan konfigurációt nem kapunk, amely már megfelel a megadott helyfoglalási igényeknek. Azt, hogy melyik általános indexet cseréljük, a használatával járó haszon és hely hányados alapján választjuk ki, mindig a legkisebb ilyen hányadossal rendelkezőt.

5.2. XMLTable nézetek ajánlása

Az XMLTable nézetek használatának előnye, hogy gyorsíthatják a lekérdezéseket az adatbázison. Ahhoz, hogy az adatbázis lekérdezésoptimalizálója használni tudja e nézeteket, szükséges az XQuery lekérdezések átalakítása SQL/XML lekérdezésekké, melyek az XMLTable funkcióit használják. A lekérdezések átírásának lépései a következők:

- Az XQuery részekre bontása
- XMLTable nézetek felsorolása: felsoroljuk az ajánlott XMLTable nézeteket az egyes részekben szereplő XPath kifejezések alapján.
- SQL/XML lekérdezés generálása, mely a felsorolt nézeteket használja
- A lekérdezésre legjobban illeszkedő nézetek kiválasztása
- Az eredeti lekérdezés átírása, hogy a javasolt nézeteket használja



Kép 4. Az XQuery lekérdezések átírása XML materializált nézeteket használó lekérdezésekké

A fő cél, hogy azonosítsuk a közös elérési út mintákat a megadott XQuery halmazon és XMLTable nézetekbe csoportosítsuk a használt XML adatokat. A nézetek bonyolultabb fizikai felépítése miatt itt nem alkalmazhatjuk az indexeknél használt “//*” megoldást. Így az elkészített megoldási folyamat nem támaszkodik a lekérdezés optimalizálóra. A javaslatok felsorolásához az XQuery lekérdezéseket felbontjuk FOR, LET, WHERE és RETURN részekre, melyeket

később komponensekre bontunk tovább. A FOR és a LET utasításoknak külön XMLTable nézeket hozunk létre.

A FOR és LET utasítások esetén a viselkedés közel azonos, felbontjuk az utasítást a változó nevére, a hozzárendelt kifejezésre és ehhez tartozó állításokra. A FOR utasítás esetében, a kifejezést hozzáadjuk mint sorgenerátort a nézethez, a LET utasításnál pedig az iterált viselkedés hiánya miatt hozzáadunk a nézethez egy "." kifejezést mint oszlop navigátor, majd a keletkezett rekordokat csoportosítjuk egy GROUP BY kifejezéssel. Végül az állításokat pedig mint oszlop navigátor hozzáadjuk a nézethez.

Ha a LET utasításban olyan kifejezés áll, mely másik változóra hivatkozik, akkor ennek értékét kifejtve adjuk meg a generátor kifejezést és hozzáadunk egy további navigátort, mely a kifejtett változó szerinti szülőre hivatkozik.

A WHERE utasítás esetében megkeressük a megadott XPath kifejezéshez tartozó nézetet és hozzáadunk a hivatkozott adattag szerint egy oszlop navigátort.

A RETURN utasításnál szintén a hivatkozott adattagok alapján megkeressük a nézetet és hozzáadunk a nézethez egy oszlopot mely az adattagokat tartalmazza.

A nézetek általánosítása ebben az esetben is segíthet, hogy egy nézet több lekérdezéshez is használható legyen, illetve előre nem látott lekérdezések megválaszolásában is hasznosíthatóak legyenek.

Az egyik mód lehet, hogy az oszlop navigátorok által tárolt értékek helyett részfákat tárolunk XML oszlop típussal, illetve összevonhatunk nézeteket, melyek azonos sorgenerátorral rendelkeznek és az újonnan készített nézet tartalmazza az oszlopnavigátorait az összevont nézeteknek.

Továbbá, hogy még hasznosabbá tegyük a nézeteket alkalmazhatunk rajtunk relációs indexeket. A cikkben heurisztikus megközelítéssel használnak egy indexet, mely tartalmazza az összes olyan oszlopot, mely adattagja megjelenik állításként a lekérdezésben.

A legoptimálisabb nézet konfiguráció meghatározásához meg kell keresnünk azt a nézet-jelölt halmazt, mely a leghasznosabb és a megadott lemezterület korláton elfér. Ehhez a már bemutatott heurisztikus moho keresést alkalmazzuk, melyet egy picit módosítani szükséges, a nézetek egymás vonatkoztatott teljesítménybeli kihatása miatt.

translateXQuery(*xquery*)	
lineNo	Added code
L0a L0b L0c	*selectElementsList* ← ϕ *fromViewsList* ← ϕ *wherePredicatesList* ← ϕ
L4a	add *view* to *fromViewsList*
L8a	add *view* to *fromViewsList*
L14a	add *comparisonExpr* to *wherePredicatesList*
L20a L20b	construct return value *returnVal* add *returnVal* to *selectElementsList*
L22a	generateQuery(*selectElementsList*, *fromViewsList*, *wherePredicatesList*)

translateXQueryAndCreateViewFromExpr(*varName*, *expr*)	
lineNo	Added code
L6a L6b	construct predicate *joinPred* to join columns "." in *refView* and "*refCol*" in *view* add predicate *joinPred* to *wherePredicatesList*
L13a	add predicate *p* to *wherePredicatesList*

Kép 5. A korábban megadott algoritmusban szükséges változtatások

5.3 További ajánlások

Az XMLTable funkcióval létrehozott materializált nézetek ugyan növelhetik a lekérdezések végrehajtási idejét, ugyanakkor előfordulhat hogy olyan nagy méretűvé válnak, hogy már nem előnyös a használatuk. Emellett a parciális indexek kisebb méretűek és drasztikusan csökkenthetik a végrehajtási időt, azonban vannak olyan esetek, ahol az indexek használata nem előnyös. Ezért ajánlunk három további módot, hogy az indexeket és a materializált nézeteket együtt tudjuk használni.

5.3.1. Előre navigálás

Az XMLTable funkció lehetőséget ad arra, hogy eltároljuk az előre kiszámolt értékeket egy relációs táblában. Az XML dokumentum lekérdezésben használt részeit tárolhatjuk így el relációs nézetekben, így a bonyolult XQuery lekérdezéseket egyszerű, ezeket a nézeteket használó select utasításokra írhatjuk át.

5.3.2. Táblák összekapcsolása

A join két oldalán álló inputok elemszáma nem csökken az összekapcsolás alkalmazásával, viszont a végrehajtási költségen spórolhatunk: az optimalizálónak lehetősége van a legalkalmasabb összekapcsolási módot választani, másrészt a materializált nézet olvasása jóval olcsóbb, mint az XML dokumentumot tartalmazó táblában való keresés.

5.3.3. Aggregáció

A csoportosító és aggregáló műveleteket tartalmazó lekérdezések optimalizálhatóak az XMLTable nézetekkel, melyek előre tartalmazhatják a csoportosított adatokat. Ez a lehetőség csak a nézetek használatánál áll elő.

5.4 Közös indexeket és nézeteket ajánló Tervezési Tanácsadó

Az XML indexek és az XMLTable nézetek különböző lekérdezések gyorsításában hatékonyak. Érdemes készíteni egy olyan Tervezési Tanácsadót, mely olyan legjobb konfigurációt ajánlj, melynek létrehozásakor figyelembe veszi mind az indexek mind a nézetek nyújtotta optimalizációt.

A Tanácsadó elkészítésénél a lehetséges jelöltek felsorolását és általánosítását különválasztjuk az indexekre és a nézetekre, mivel a két típushoz tartozó jelöltek előállítása különböző. A művelet eredményeképp három típusú jelöltek állnak elő:

- XML indexek
- XMLTable materializált nézetek
- XMLTable materializált nézetek relációs indexekkel

Ezeket a jelölteket egy közös gyűjteménybe vesszük, és közöttük keressük a legjobb konfigurációt. A kereső algoritmus eltér a korábban ismertetettektől, mivel itt különböző típusú jelölteink vannak.

Az algoritmus működésének első lépése a jelöltek haszon/méret arány szerinti rendezése. Ezután végigiterálunk a jelölteken, minden iterációban ha a jelölt új lefedést ad az ajánlottakhoz és van számára elég hely, akkor bevesszük az ajánlott struktúrák közé. Egyébként ha átfedés van az ajánlott konfiguráció és az aktuális jelölt között, akkor heurisztikát alkalmazunk annak eldöntésére hogy hozzáadjuk-e a jelöltet az ajánlottakhoz.

5.5. Kísérletek

Az elkészült Tervezési Tanácsadókat külön-külön teszteltük.

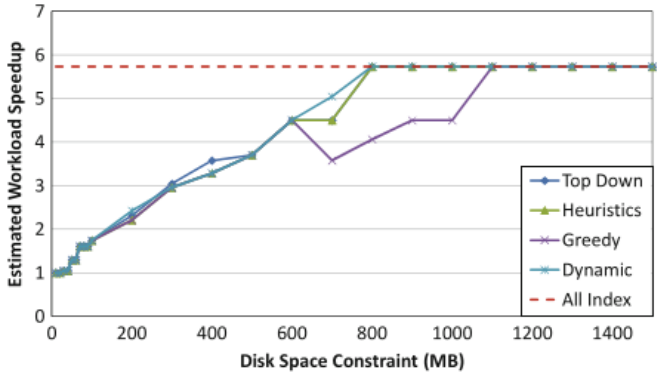
5.5.1. Index Tanácsadó

A következő részben kísérleti eredményekkel támasztjuk alá, hogy az általunk ajánlott Index Tanácsadó jó index ajánlásokat tesz és hatásosan használja ki a rendelkezésre álló lemezterületet.

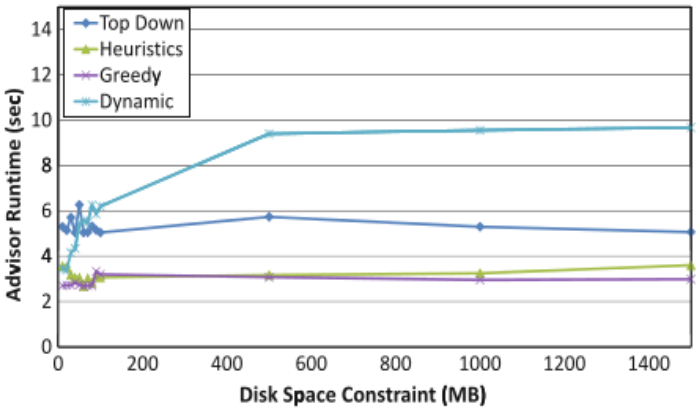
- Indexek ajánlása lépés

Az általunk adott megoldásban az indexek közötti keresési módot variáljuk, ezzel akarjuk belátni, hogy a mohó keresés a heurisztikával hatásosan működik. Az alábbi keresési módokat próbáljuk ki:

  - mohó keresés
  - mohó keresés heurisztikával (ezt építettük be az Index Tanácsadóba)
  - felülről lefelé haladó keresés
  - dinamikus programozás



Kép 6. A lekérdezés kiértékelésének sebességnövekedése az index ajánlásokkal

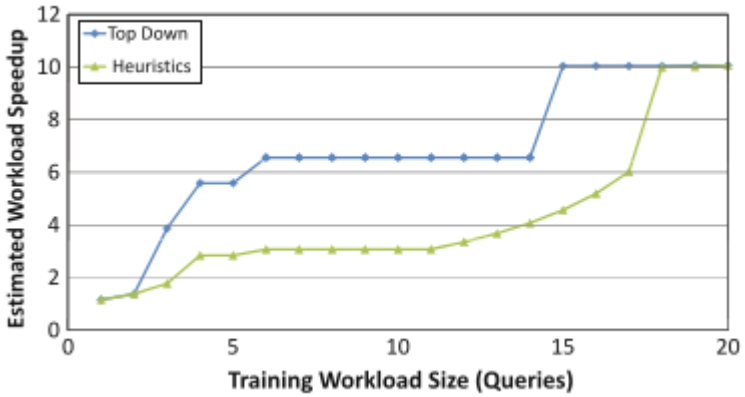


Kép 7. A Tanácsadó futási ideje a különböző keresési tehcnikákkal.

A lekérdezés kiértékelésének gyorsasága és a Tanácsadó futási ideje alapján is a mohó keresés heurisztikával produkálta a legjobb eredményeket.

- Indexek általánosítása lépés

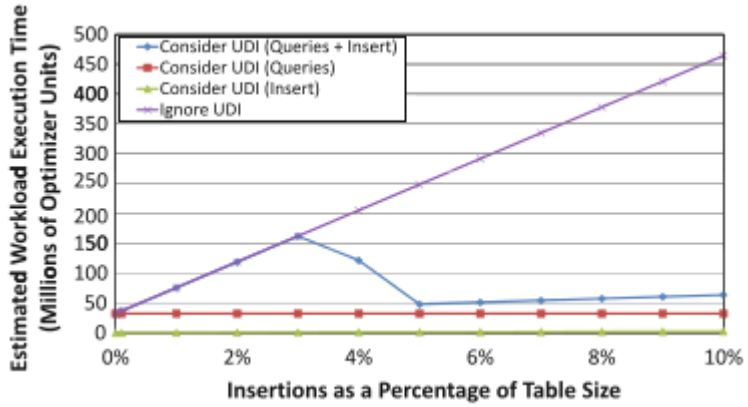
Itt azt demonstráljuk, hogy a Tanácsadónk az általánosítás lépésben az alap indexeknél sokkal általánosabb indexeket ajánl, olyanokat, amiket később fel tudunk használni oylan lekérdezéseknél is, amelyekre nem volt még példa.



Kép 8. Index Tanácsaó általánosításának következménye a még nem ismert lekérdezéskhez

- A jelölt konfigurációk költségének számítása

Ezen a ponton azt szükséges vizsgálnunk, hogy mennyire pontosak az index költség-előnyre és az index fentartásához szükséges költségekre vonatkozó számítások.



Kép 9. Az UPDATE műveletek mellékhatásai

5.5.2. XMLTable nézet Tanácsadó

- Összevont XMLTable nézetek ajánlása
  - Alap konfiguráció: minden nézetet tartalmaz amelyet a lekérdezésekhez felsoroltunk

- Összevont: általánosított nézeteket tartalmaz, melyek az alapnézetek összevonásával keletkeztek

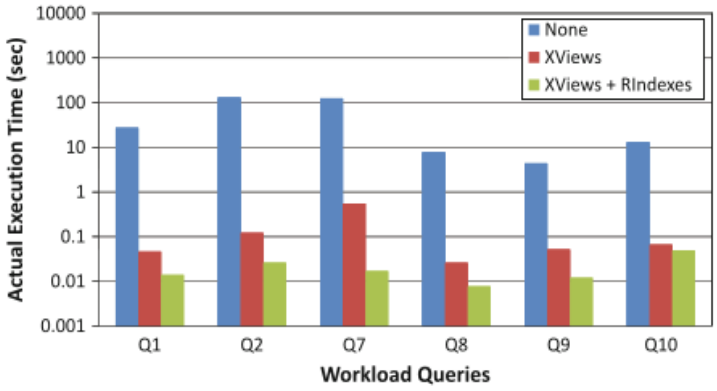
Table 3 Effect of merging views on performance

Configuration	Size (MB)	Speedup	Benefit/size ratio
Basic	58.2	354.6	5.3
Generalized	48.8	198.0	6.3

A nézetek összevonásával járó hatékonyság növekedés

A táblázat alapján látható, hogy a nézetek összevonásával a konfiguráció összmérete csökke. A gyorsasági növekedés csökken az összevonásokkal, azonban összesítve az eredményeket a haszon/méret arány az összevont nézetek esetén jobb. Tehát indokolt a nézetek általánosítása.

- XMLTable nézetek relációs indexekkel
  Megvizsgáljuk az előnyeit annak, ha nem csak XML nézeteket, de ezek alapján készült indexeket is használunk.

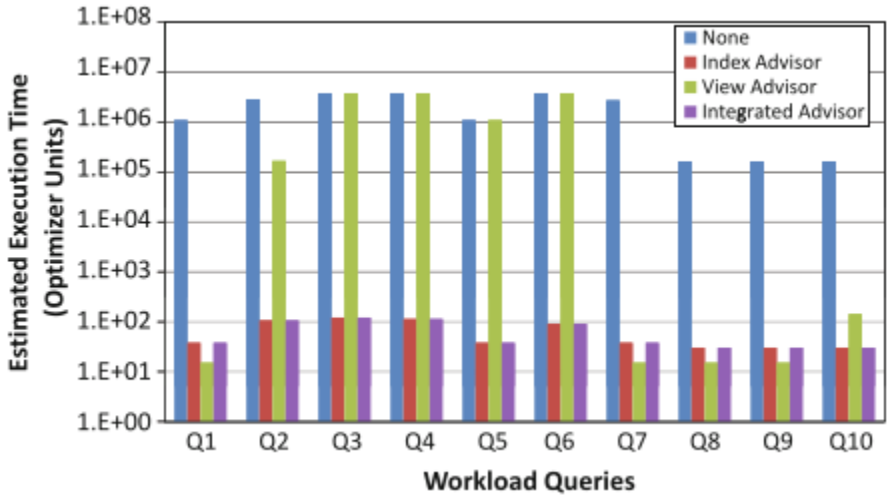


Kép 10. A lekérdezések végrehajtási ideje nézetek nélkül, nézetek használatával illetve nézetek és a hozzájuk készült indexek használatával

Az ábra jól mutatja, hogy a nézetekhez készült indexek használatáéval további javulást érhetünk el a végrehajtáshoz szükséges időt nézve.

5.5.3. Integrált index és nézet Tanácsadó

Itt az általunk létrehozott tanácsadók által ajánlott struktúrák használatával előforduló különbségeket vizsgáljuk.



Kép 11. A különböző tervezési tanácsadók által adott ajánlások használatával elérhető végrehajtási idő csökkenés

Az ábrán jól látható, hogy bizonyos esetekben az Index Tanácsadó által adott javaslatok, máskor pedig a Nézet Tanácsadó ajánlásai az optimálisabbak. Jól látszik az is, hogy míg ezek teljesítmény javítása ingadozó, addig az Integrált Tanácsadó kiegyensúlyozott teljesítményt nyújt. Ez igazolja létjogosultságát.

6. További kutatás terv

Bemutattunk egy a fizikai tervezést segítő eszközt, amely automatikusan ajánl XML indexeket és materializált nézeteket XML adatbázisokhoz. Mivel az XML adatok adatbázisban való tárolása még nem kiforrott, ezért mindig van további vizsgálandó a témában.

7. Irodalomjegyzék

1. Iman Elghandour, Ashraf Aboulnaga, Daniel C. Zilio, Calisto Zuzurte: Recommending XML Physical Designs for XML Databases (2013)

2. Thurzó Máté István: Adattárházak és üzleti intelligencia (2008)

3. Dr. Kovács László: Az XQuery szabvány elemei rendszer

4. Hammerschmidt, B.C., Kempa, M., Linnemann, V.: A selective key-oriented XML index for the index selection problem in XDBMS. In: DEXA (2004)

5. Hammerschmidt, B.C., Kempa, M., Linnemann, V.: Autonomous index optimization in XML databases. In: SMDB (2005)

6. Runapongsa, K., Patel, J.M., Bordawekar, R., Padmanabhan, S.: XIST: An XML index selection tool. In: XSym (2004)