BUDAPESTI MŰSZAKI ÉS GAZDASÁGTUDOMÁNYI EGYETEM
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Integrált Információs Rendszerek
Adatbázisok

Kovács Ferenc

2004

Tartalom

1 Bevezetés ... 4
1.1 Adatbázis definíció ... 4
1.2 Adatbázis-kezelő rendszerek alkalmazásának előnyei és hátrányai ... 5
2 Relációs adatmodell ... 8
2.1 A relációs adatstruktúra ... 8
2.2 Integritási kritériumok ... 8
2.2.1 Alapfogalmak ... 8
2.2.2 Integritási kritériumok ... 9
2.3 Adatmanipulációs műveletek ... 9
2.3.1 Kiválasztás ... 9
2.3.2 Vetítés ... 10
2.3.3 Descartes szorzat ... 10
2.3.4 Táblák összekapcsolása, join műveletek ... 10
2.3.5 Halmazműveletek ... 10
2.4 A relációs adatmodell kapcsolata a relációs adatbázis-kezelő rendszerekkel ... 11
2.5 Relációs adatmodell használata ... 11
2.5.1 Anomáliák ... 11
2.5.2 Normalizáció ... 12
2.5.3 Denormalizáció ... 12
3 Adatbázisok szerver oldali programozása ... 14
3.1 Szerver-oldali programozás előnyei ... 14
3.2 Szerver-oldali programozás hátrányai ... 14
3.3 Szerver-oldali programozás eszközei ... 15
3.3.1 Tárolt eljárások ... 15
3.3.2 Tárolt függvények ... 16
3.3.3 Triggerek ... 16
4 Adatbázisok elérése ... 19
4.1 Kapcsolódás az adatbázisszerverhez ... 19
4.2 Szabványos osztálykönyvtárak használata ... 19
4.2.1 Adatbázis meghajtó (driver) használata ... 20
4.2.2 Adatbáziskliens szerepe ... 20
4.2.3 Osztálykönyvtárak felépítése ... 20
4.3 JDBC felépítése ... 21
4.3.1 JDBC driver ... 21
4.3.2 Tranzakció kezelés ... 21
4.3.3 SQL utasítások végrehajtása ... 21
4.3.4 ResultSet használata ... 22
4.4 ODBC használata ... 22
4.5 OLE DB / ADO használata ... 23
5 Adattárházak ... 25
5.1 Tranzakciós és adattárház rendszerek összehasonlítása ... 25
5.2 Adattárház architektúrák ... 26
5.3 Adattárház építésének lépései ... 26
5.3.1 Adatok kinyerése ... 27
5.3.2 Adattisztítás ... 27
5.3.3 Adat transzformáció ... 28
5.3.4 Adatok betöltése és indexelése ... 28
5.4 Multidimenzionális adatmodell ... 29
5.4.1 Csillag séma ... 30
5.4.2 Hópehely séma ... 30
6 OLAP rendszerek ... 31
6.1 OLAP rendszerek definíciója ... 31
6.1.1 Codd teszt ... 31

6.1.2 FASMI teszt 31
6.2 OLAP kocka 32
6.2.1 OLAP kocka felépítése 32
6.2.2 OLAP kocka műveletek 33
6.2.3 OLAP kocka tárolása 34
6.2.4 Előkalkuláció mértéke 34
7 Adatbányászat 36
7.1 Gépi tanulás 36
7.2 Adatbányászati alapfeladatok 37
7.3 Asszociációs szabályok keresése 38
7.3.1 Termékhalmazok és bevásárlói kosarak 38
7.3.2 Asszociációs szabály 38
7.3.3 Érvényes asszociációs szabályok meghatározása 38
7.4 Osztályozás 39
7.4.1 Döntési fa 40
7.4.2 Neurális hálók 40
7.4.3 K legközelebbi szomszéd elve 41
7.5 Klaszterezés 41
7.5.1 Elvárások a klaszterező algoritmusoktól 41
7.5.2 Elemek távolsága 41
7.5.3 K közép algoritmus 42
8 Irodalomjegyzék 43

1 Bevezetés

Napjainkban szinte mindennap kapcsolatba kerülünk valamilyen adatbázis-kezelő rendszerrel. A bankban pénzt utalunk egyik számláról a másikra, az Interneten keresztül rendelünk az Amazon.com-ról, számítógépen jelentkezünk kedvenc tárgyunkra (ha működik a rendszer), és így tovább. Napjainkban az üzleti szereplők egyik legnagyobb értéke az adat. Ezeket az adatokat biztonságban kell tudni, viszont a megfelelő embereknek hatékony és gyors hozzáférést kell biztosítanunk. A nyers adatok nagyon sokféleképpen felhasználhatók. Lekérhetjük egy állandó vásárlónk adatát, és automatikusan számlát generálhatunk. A vásárlási szokásokat térképezhetjük fel adatbányászati eszközökkel, és kideríthetjük, hogy aki csipszet vásárol az általában üdítőt is vásárol; ezek után az üdítős pultot a csipszes pult mellet helyezzük el. Adattárházunk segítségével kideríthetjük, hogy az ország mely területein csökkentek az eladások. Ahhoz, hogy az adatbázisban lévő adat információvá váljon, megfelelően kell adatainkat kezelni.

1.1 Adatbázis definíció

Ahhoz, hogy adatbázisokról lehessen beszélni először is tisztázni kell magának az adatbázis fogalomnak a jelentését. Adatbázist a következőképpen lehet definiálni: logikailag összefüggő adatok rendezett gyűjteménye. Az adat régebbi definíciója: ismert tények, amelyek számítógépes adattárolón rögzíthetők. Például egy vásárló neve címe, adószáma, ÁFA- kategóriája, stb. Ezek az adatok például megtalálhatók egy vállalat területi képviselőjének a laptopján. Napjainkban azonban a számítástechnika fejlődése lehetővé tette, hogy nemcsak ezen hagyományos adatokat tárolhatjuk számítógépünkön, hanem képet, hangot, videót is. Ezek után úgy módosíthatjuk az adat definíciónkat, hogy olyan tény, szöveg, grafika, kép, hang, videó, amely valamilyen jelentést hordoz a felhasználó számára. Összefüggő adatok alatt olyan adatokat értünk, mely lefedi egy felhasználó csoport érdeklődési területét. A rendezett gyűjtemény azt jelenti, hogy úgy rendezzük az adatainkat, hogy azt könnyű legyen tárolni, módosítani, illetve könnyen nyerjünk információt ezekből az anyagokból.

Egy adatbázis adatai önmagukban nem elégségesek, szükség van leíró adatokra is, az úgynevezett meta adatokra, melyek az egyes adatelemek értelmezését adják.

Kovács Béla	12345678
Nagy Aladár	24567990
Kiss István	56833217

ábra

Önmagában, csak az adatokból kiindulva elég nehéz kitalálni, hogy az 1.1 ábrán látható táblázatban milyen adatok szerepelnek. A nevet valószínűleg mindenki kitalálja, de a számcsoportnak számtalan jelentése lehet. Ebben az esetben egy könyvtári kölcsönzési rendszer adatait látjuk, és a számcsoport az adott tagnál lévő könyv egyedi nemzetközi azonosítóját (ISBN) tartalmazza. Ha ezt nem írjuk le valahol, akkor különböző felhasználók különböző jelentést tulajdonítanak ugyanannak az adatnak. A meta adat az adat leírója. Jelentést párosít az adathoz, meghatározza az adat struktúráját, valamint azt, hogy milyen szabályoknak illetve

korlátozásoknak kell megfelelnie az adatoknak. Az adatok leírói, a meta adatok az adatraktárban (repository) találhatók. Fontos, hogy az adatbázis mellett fenntartsunk egy ilyen raktárt.

Adatelem Neve	Típus	Méret	Leírás
Név	Szöveg	30 karakter	Név(Vezetéknév Keresztnév)
Könyv	Szám	9 jegyű szám	Könyv ISBN száma

1.2 ábra

Az 1.2 és ábrán egy egyszerűsített meta adat leírás látható. A név konkrétan meghatározza, hogy a vezeték- és keresztnév egyaránt ebben a mezőben tárolandó. Maximum 30 karaktert tudunk itt tárolni. A könyv adat egy 9-jegyű szám, ami a kölcsönzött könyv ISBN számát tartalmazza. Ezen kívül még olyan korlátozásokat definiálhatunk, hogy az ISBN csak olyan könyvre utalhat, amiből a könyvtárban van legalább egy példány, és más nem kölcsönözte ki ugyanabban az időszakban. A különböző adatbázis-kezelők különböző eszközök segítségével biztosítják az adatok integritását, amelyek ismertetése túlmutat a tárgy keretein.

1.2 Adatbázis-kezelő rendszerek alkalmazásának előnyei és hátrányai

Adatbázis-kezelő rendszerek alkalmazása számos előnnyel jár, melyek megmutatkoznak mind a rendszerfejlesztési és integrációs feladatoknál, mind a rendszer üzemletetés során:

- *Információ nyerhető az adatokról:* Az adatbázis-kezelők megkönnyítik, hogy az adatokból információt nyerjünk. Mivel az adatok egy helyen, a központi adatbázisban, vannak tárolva, és ezért komplex lekérdezések is végrehajthatók. A kliens-eszközök által nyújtott lehetőségek segítségével akár komplex analitikus elemzéseket is végezhetünk. A szabványosított lekérdező nyelv (SQL) segítségével könnyen juthatunk új információhoz, ami külön programozást igényelne egy fájlokon alapuló rendszerben.
- *Program adat függetlenség:* Az adatbázis-kezelők egyik legnagyobb előnye, hogy az adat definíciója az adathoz közel, vagyis az adatbázisban, illetve az adatraktárban (repository) található, ezért független az őt használó alkalmazástól. Például egy alfanumerikus adat egy C programban karakter-tömbbé, míg egy Visual Basic programban egy sztring típusú adattá válhat. Mindkét program ugyanazt az adatot éri el. Ha az adat szerkezetén változtatni kell, akkor azt elég adatbázis-szinten elvégezni, és a kliens program ettől függetlenül normálisan működik.
- *Minimális redundancia:* A redundáns adatok sok problémát okozhatnak. Egyrészt az értékes tárolási kapacitásunkat pazaroljuk el azzal, hogy ugyanazon adatot több helyen is tárolunk. Másrészt, ha az adatot módosítani kell, akkor ezt a módosítást több helyen kell elvégezni, és mivel több program is dolgozhat ugyanazon az adaton, erre többszörösen kell figyelni. Az adatbázis-kezelők fő célja, hogy az adatfájlokat egyetlen logikus struktúrába integrálja. Néha az adatbázis-kezelőkben is tárolunk redundánsan adatokat, aminek jobb elérhetőség, illetve gyorsabb lekérdezési sebesség a célja. Ez az

egyik alapja az elosztott adatbázis-kezelő rendszereknek. Nagyon fontos, hogy mielőtt bármiféle redundanciát építünk az adatbázis-kezelőnkbe, azelőtt készítsünk egy normalizált redundancia-mentes rendszert, és aztán ezt a rendszert denormalizáljuk, mert így minden egyes redundanciára fény derül.

- *Adatkonzisztencia:* Az adat-konzisztencia egyrészt a redundancia-csökkentés eredménye, másrészt annak tulajdonítható, hogy adatbázis-szinten írhatjuk le az adataink közötti kapcsolatokat. Ezek lehetnek egyszerű kapcsolatok, de akár összetett ún. üzleti szabályok is. Fontos itt is az adatraktár szerepe, ahol az adatok, és a közöttük lévő kapcsolatok vannak tárolva.
- *Adatok megosztott elérése:* Ez talán az egyik legnagyobb előnye az adatbázis-kezelőknek. Különösebb programozás nélkül biztosított az adtok egyidejű elérése több felhasználó számára. Az adatbázis-kezelő elvégzi helyettünk a különböző zárolási feladatokat, amelyek akkor lépnek fel, ha ugyanahhoz az adathoz többen kívánnak egyszerre hozzáférni, akár olvasás, akár írás szándékával. Egy adott felhasználó számára biztosítható egy saját nézet, ami testre-szabott, és független a többi felhasználó nézetétől. Mivel az adatok egy helyen vannak tárolva, ezért az adatok biztonsága is könnyebben biztosítható.
- *Gyorsabb alkalmazásfejlesztés:* Egy adatbáziskezelő-rendszer segítségével gyorsabban fejleszthetünk alkalmazást, mert az adatbázis-kezelőben rengeteg olyan feladat már meg van oldva, ami egyébként külön programozási feladat lenne egy fájl-kezelésen alapuló rendszerben. Ugyanakkor kliens-oldalon is rengeteg olyan fejlesztő-eszköz létezik, amely szabványos módon tud hozzáférni adatbázis-kezelőkhöz.
- *Szabványok:* Adatbázis-kezelőkre több szabvány is létezik, amelynek segítségével a rendszerünk sokkal áttekinthetőbb, és ebből következően könnyebben fenntartható lesz. Egyrészt vannak standard adatbázis-tervezési eljárások, másrészt a szabványos adatbázis-elérés alkalmazásával biztosíthatjuk a rendszerünk könnyebb szállíthatóságát egyik platformról a másikra. Az adatbázis-adminisztrátor feladat, hogy ellenőrizze, hogy ezeket a szabványokat betartják-e.
- *Biztonság:* Az adatbázis-kezelők alkalmazásával az adatok biztonságát is könnyen megoldhatjuk. Az adatbázisokban sokszor többféle adat található meg többféle felhasználó számára. Vannak olyan adatok, amelyeket csak bizonyos felhasználók láthatnak, és vannak teljesen publikus adatok is. Az adatok biztonságos hozzáférését az adatbázis-kezelők biztosítják. Egyrészt minden felhasználónak azonosítania kell magát, mielőtt beléphet az adatbázisban, és ezek után az azonosított felhasználó csak bizonyos nézetét látja az adatbázisnak. Az adminisztrátor feladata a megfelelő felhasználók létrehozása, és hozzáférési jogának menedzselése.
- *Egyszerűbb karbantartás:* Egyszerűbb program-karbantartás annyit jelent, hogy a program-adat szétválasztás miatt az adat módosítása nem befolyásolja a program működését

Természetesen, mint minden eszköznek, nemcsak előnyei vannak az adatbázis-kezelő rendszerek használatának, hanem néhány hátrányt is jelenthet az alkalmazásuk:

- *Új, speciális munkaerő:* Egy adatbázis-kezelő rendszer bevezetése azt is magával hordozza, hogy olyan új munkaerőre van szükség, amely hatékonyan ki tudja használni ezen rendszerek előnyeit. Mindamellett egy adatbázis

adminisztrátorra is szükség van, aki a napi teendőket ellátja. Természetesen az adminisztrátornak ismernie kell az adott rendszer sajátosságait. Mivel egy adatbázis-kezelő alkalmazása új fejlesztési módszereket igényel, ezért szükség van olyan szakemberre, aki ezeket az új módszereket ismeri, és képes menedzselni egy ilyen fejlesztő csapatot.

- *Telepítési, menedzselési költségek:* Újabb költség maga, az adatbázis-kezelő rendszer beszerzése. Mivel ezek a programcsomagok nagyon sokrétűek, és rengeteg olyan beépített lehetőséggel rendelkeznek, amelyek esetleg az adott pillanatban nem szükségesek, de később szükség lehet rájuk, ezért ezek a rendszerek drágák. Azonban az is igaz, hogy ezt az árat csak egyszer kell megfizetnünk, és a rendszer akár több alkalmazást is ki tud szolgálni. Az adatbázis-kezelőkre is igaz, ami a számítógép hardverre, mégpedig az, hogy nagyon gyorsan elavulttá válik, ezért újabb költség a rendszer állandó frissítése, valamint ezzel párhuzamosan a munkaerő továbbképzése. Az újabb verziók természetesen egyre erősebb hardvert igényelnek, tehát ezt is állandóan frissíteni kell.
- *Konverziós költségek:* Újabb költség magának, az eddig működő programnak az átírása, átvitele az új rendszerre. A régi rendszer úgy, ahogy működik, ezért ez az átmenet mindig kockázatot hordoz magában, de megfelelő tervezéssel a kockázatok csökkenthetők. A konverzió maga időbe, pénzbe kerül, és a cég részéről (leginkább a menedzsment részéről) szükség van arra, hogy elkötelezzék magukat a változtatás mellett.
- *Mentés és visszaállítás:* Az adatbázis-kezelők általában egy központi helyen tárolják az adatokat, amelyek kritikusak a vállalat működése számára. Ezek az adatok nagyon érzékenyek, ezért fontos egy jól megtervezett mentési-visszaállítási stratégia alkalmazása. Szerencsére erre minden adatbázis-kezelőben nagyon jó segédeszközök találhatók. Talán ez az adatbázis adminisztrátor legfontosabb feladata.

2 Relációs adatmodell

A relációs adatmodellt E. F. Codd alkotta meg 1970-ben az IBM-nél. Ekkor két prototípus készült, az egyik a az IBM San Jose kutatási központjában, a másik pedig a Californiai Egyetemen. Kereskedelmi relációs adatbázis-kezelő rendszerek 1980-tól kezdtek megjelenni a piacon. Mielőtt megkezdenénk a relációs adatmodell tárgyalását, fontos leszögezni, hogy a relációs adatmodell nem azonos az entitás-kapcsolat diagramokkal. Az entitás-kapcsolat diagramok leképezhetők a relációs adatmodellre, de más adatmodellekre is, bár ez a legelterjedtebb leképezés. Az E-R diagramokat a koncepciós tervezés szintjén használjuk, míg a relációs adatmodell a logikai tervezés része. A relációs adatmodell matematikailag megalapozott modell, melynek építőkövei a következők:

- *Reláció.* Más néven tábla. Az adatok tárolására szolgáló struktúra
- *Integritási kritériumok.* Az adatmodell tartalmaz eszközöket az adat konzisztencia biztosítására.
- *Adatmanipulációs műveletek.* Az adatokkal végezhető műveletek is részét képezik a matematikai modellnek.

2.1 A relációs adatstruktúra

A relációs adatstruktúra egy kétdimenziós táblázatot jelent, mely tetszőleges számú, megnevezett oszlopot tartalmaz, és tetszőleges számú sorból állhat. A táblázat egy sorát rekordnak nevezzük, egy nevezett oszlopát attribútumnak. Egy relációban az attribútumok számát a reláció fokának a rekordok számát a reláció kardinalitásának nevezzük. Egy reláció definíciójának nem része a benne található adatok megadása, azaz a reláció csak az adattárolási struktúrát definiálja. A relációk alaptulajdonságai:

- minden relációnak egyedi neve van
- egy sor és egy oszlop kereszteződésében egy érték szerepel
- oszlopok nevei egyediek egy relación belül
- oszlopok sorrendje lényegtelen egy relación belül
- egy reláció tetszőleges számú rekordot tartalmazhat
- a rekordok sorrendje lényegtelen egy relación belül
- egy relációban nem lehet két azonos sor

2.2 Integritási kritériumok

Az integritási kritériumok az adatbázisban található adatok konzisztenciájának modell szintű leírására szolgálnak.

2.2.1 Alapfogalmak

Az integritási kritériumok definiálásához szükséges b evezetni néhány alapfogalmat:

- Funkcionális függőség: Egy R reláció A attribútum halmaza funkcionálisan függ B attribútum halmaztól, ha B értéke egyértelműen meghatározza A értékét. Jelölése: B → A. B attribútum halmazt a függőség determinánsának nevezzük.
- Kulcs(jelölt): Egy R reláció kulcsának nevezünk egy A attribútum halmazt, ha:
    - A reláció összes nem kulcs attribútuma funkcionálisan függ a kulcstól. Azaz a kulcs egyértelmű.

    - Amennyiben elhagyunk egy attribútumot a kulcsból akkor az előző állítás már nem igaz. Azaz a kulcs minimális.
- Elsődleges kulcs: Egy reláció kulcsai közül szabadon kiválasztva megkülönböztetünk egy kulcsot, melyet a reláció elsődleges kulcsának nevezünk.
- Idegen (külső) kulcs: Egy reláció azon attribútumait, melyek kulcsai egy másik relációnak idegen kulcsoknak nevezzük.

2.2.2 Integritási kritériumok

A relációs adatmodell három integritási kritériumot tartalmaz:

- Tartományi integritás: megkötés az egyes attribútumok értékkészletére. Ilyen megkötésre lehet példa, hogy egy útlevél számot tartalmazó attribútum csak olyan értékeket vehet fel, ami első két karakterén betűt tartalmaz és utána következő hat helyen számjegyet.
- Entitás integritás: minden relációnak van elsődleges kulcsa. Az elsődleges kulcs egyetlen része sem vehet fel NULL (azaz meg nem határozott) értéket.
- Referenciális integritás: A definiált külső kulcsok mentén megköveteli, a hivatkozott rekord létezését, amennyiben az idegen kulcs nem NULL. Valamint egy rekord addig nem törölhető az adatbázisból, amíg van rá mutató érvényes idegen kulcs. Éppen ezért egy rekord törlése során definiálható, hogy mi történjen a rá mutató hivatkozásokkal:
    - Törlés visszautasítása, amennyiben van hivatkozás. Általában ez az alapértelmezett működés.
    - Cascade delete: az összes hivatkozott rekord törlése a szülő rekorddal együtt.
    - Cascade update: az összes hivatkozott rekordban az idegen kulcs értékének NULL-ra állítása. A gyakorlatban ezt a megoldást kevés adatbázis-kezelő támogatja.

A fenti három integritási kritériumon kívül még beszélhetünk egy negyedikről is, amit működési korlátozásoknak neveznek. Ez alatt azon integritási kritériumokat értjük, melyek az üzleti logikából következnek, de túlmutatnak a relációs adatmodellen. Ilyen megkötések például azok, melyek egy relációban található sorok számára adnak megkötéseket.

2.3 Adatmanipulációs műveletek

A relációs adatmodell része az adatmanipuláció is, mely műveleteket definiál relációkon és eredményül származtatott relációkat szolgáltatnak. Jelen jegyzettben a számos adatmanipulációs műveletek definiálására alkalmas leírás közül a legegyszerűbbet a relációs algebrai műveleteket ismertetjük. Mivel a relációs algebrában alkalmazott gondolkodásmód áll legközelebb a gyakorlati életben használt SQL nyelvhez.

2.3.1 Kiválasztás

A kiválasztás operátor sorokat választ ki az adott táblából. Jele a σ, és meg kell adnunk azt a feltételt, aminek teljesülnie kell ahhoz, hogy az adott sor szerepeljen az eredményhalmazban.

2.3.2 Vetítés

A vetítés operátor bizonyos oszlopokat (attribútumokat) választ ki a táblából, de minden sor benne van az eredményhalmazban. Jele a π, és az oszlopok listáját kell megadnunk. Viszont vetítés során az eredmény reláció kevesebb rekordot is tartalmazhat, mint a kiindulási reláció, mivel egy reláció nem tartalmazhat két azonos sort, és vetítés során veszíthetünk olyan attribútumokat, melyek megkülönböztetnek sorokat egymástól.

2.3.3 Descartes szorzat

A Descartes szorzat minden sort minden sorral összepárosít a hivatkozott táblákból.

2.3.4 Táblák összekapcsolása, join műveletek

Táblák összekapcsolását legegyszerűbben a Descartes szorzatból lehet leszármaztatni. Az összekapcsolás során egyéb szűrő feltételeket lehet definiálni, azaz egy leválogatást végzünk a relációk Descartes szorzatán. A join műveletek jele: $\bowtie$, valamint egy feltétel lista, mely az összekapcsolási feltételeket tartalmazza. Tipikus join műveletek:

- Theta join: Tetszőleges logikai műveletekkel leírt összekapcsolási feltételek
- Equi join: Attribútumok egyezőségén alapuló összekapcsolás. Az eredmény reláció ugyan annyi attribútumot tartalmaz, mint a kiinduló relációk összesen.
- Natural join: Tulajdonképpen ugyan az, mint az equi join, de minden attribútumot csak egyszer tartalmaz. Azaz az egyenlő attribútumok közül csak az egyik szerepel az eredmény relációban.
- Outer join: A reláció olyan módon történő összekapcsolása, hogy azokat a rekordokat is tartalmazza az eredmény reláció, melyhez a natural join nem talált párt. Éppen ezért beszélhetünk left és right outer joinról attól függően, hogy a join művelet bal vagy jobb oldalán álló relációnak szeretnénk megkapni minden sorát függetlenül attól, hogy a másik relációban talált e párt a natural join műveletek. A párokkal nem rendelkező rekordokban a másik relációból származó attribútumok NULL értéket vesznek fel. Az outer join jelölésére a következő jeleket használatosak: $⟕$, $⟖$ attól függően, hogy bal vagy jobb oldali outer joinról beszélünk.

2.3.5 Halmazműveletek

Az unió, metszet, és a különbség a halmazelméletben megszokott jelentéssel bír. Feltételük, hogy unió-kompatibilis relációkról legyen szó. Unió-kompatibilis azt jelenti, hogy ugyanannyi attribútumuk van, és a megfelelő attribútumoknak ugyanaz az értelmezési tartományuk. Az unió operátor minden sort mindekét táblából belefoglalja az eredményhalmazba, de az azonos sorok csak egyszer szerepelnek. A metszetben azok a sorok szerepelnek, amelyek mindkét táblában megtalálhatók. A különbség azon sorok összessége, amelyek csak az első táblában szerepelnek a másodikban nem.

2.4 A relációs adatmodell kapcsolata a relációs adatbázis-kezelő rendszerekkel

A relációs adatmodell egy matematikai modell, melynek egy-egy implementációját adják a relációs adatbázis-kezelő rendszerek. Ez azt jelenti, hogy a matematikai modell tulajdonképpen egy követelményrendszert (elvárást) jelent az implementációval szembe. Azaz elvárjuk, hogy az adatok fizikai tárolásának (adat fájlok) ismerete nélkül az egész adattárolási rendszert magasabb absztrakciós szinten lehessen leírni. A relációs adatbázis-kezelők biztosítják a relációk definiálásának lehetőségét (tábla létrehozása) és az adatainkhoz a definiált logikai sémán keresztül tudunk hozzáférni. Teljesen hasonlóan közelíthető meg az integritási kritériumok használata is, azaz definíciós szinten jelennek meg. A betartandó integritási kritériumokat csak definiálni kell, a kritériumok betartatása az adatbázis-kezelő szerver feladata. Habár a működési korlátozások kimutatnak a relációs adatmodellből, a kereskedelmi forgalomban lévő relációs adatbázis-kezelő rendszerek nyújtanak támogatást implementálásukhoz (lásd szerver oldali programozás). Az adatmanipulációs műveletek SQL utasításokon keresztül érhetők el, az SQL utasítások által megfogalmazható adatmanipulációs műveletek nagymértékben hasonlítanak a bemutatott relációs algebrai műveletekhez. Fontos megjegyezni, hogy az SQL utasítások is egy definíciós környezetet jelentenek, ahol a kívánt eredmény halmazt csak definiáljuk, viszont magának az eredmény halmaz eleminek a meghatározása már az adatbázis-kezelő szerver feladata. Ugyanazt az eredményhalmazt több különböző SQL utasítással is elő lehet állítani, elméletileg a megoldások egymással ekvivalense, a különbözőképpen megfogalmazott SQL utasításokhoz általában más végrehajtási terv és válaszidő tartozhat. Éppen ezért a gyakorlati élet szempontjából fontos, hogy az SQL lekérdezéseinket miképp fogalmazzuk meg.

2.5 Relációs adatmodell használata

A relációs adatmodell használata során az adatbázis logikai struktúrájának a kialakítása a feladat, azaz a relációs séma megalkotása. A relációs séma összeállítása során figyelembe kell venni, hogy a kialakítandó sémában a redundás adattárolásból adódóan számos anomália léphet fel, melyek kezelésére külön figyelmet kell fordítani.

2.5.1 Anomáliák

Az anomáliák ismertetése során vegyük a következő relációt használjuk példának: BetegVizsgálat(OrvosAzonosító, OrvosNév, BetegAzonosító, BetegNév, Időpont, Rendelő). Egy relációs adatstruktúrában a következő anomáliák léphetnek fel:

- Beszúrási anomália: Relációs adatstruktúrában olyan ellentmondás található, mely megnehezíti új rekordok felvitelét, mégpedig oly módon, hogy az új rekord rögzítésének pillanatában még nem áll rendelkezésre minden szükséges információ. A bemutatott példában egy orvost addig nem tudunk nyilvántartásba venni, amíg nem jelentkezik hozzá beteg.

- Törlési anomália: Egy rekord törlésével olyan információk is elvesznek az adatbázisból, melyet meg szerettünk volna tartani. A példa relációban, ha egy orvos utolsó betege is lemondja a vizsgálatot, akkor elveszítjük azt az információt is, hogy az orvos nálunk dolgozik.
- Módosítási anomália: Egy adott információt módosítás során több rekordban is meg kell változtatni, máskülönben nem lesz konzisztens az adatbázis. Például ha egy orvos nevet változtat (p.l.: házasságkötés következtében), akkor az összes érintett rekordban át kell írni a nevét.

2.5.2 Normalizáció

A fellépő anomáliák eltüntetése végett a relációs adatmodell normalizációja szükséges. A normalizációs algoritmus mechanikusan végrehajtható és eredményül a kiválasztott normál formájú relációs séma adódik. A relációs sémát több különböző normál formájú alakban lehet megfogalmazni, a gyakorlati életben az alábbi formák a legelterjedtebbek.

- 1NF (első normál forma): Egy sor és oszlop kereszteződésében pontosan egy érték szerepel, azaz relációs adatmodellel dolgozunk.
- 2NF (második normál forma): Egy reláció 1NF-ban van, és nem tartalmaz részleges kulcsfüggést. Azaz egyetlen nem kulcs attribútum sem függ kulcs egy valódi részhalmazától.
- 3NF (harmadik normál forma): Egy reláció 2NF-ban van és nincs benne tranzitív függőség. Azaz egyetlen attribútum sem függ tranzitíven a kulcstól.
- BCNF (Boyce-Codd normál forma): A reláció 1NF-ban van és a funkcionális függőségek determinánsai kulcsjelöltek.

Minden relációnak van veszteségmentes függőségőrző 3NF felbontása. Minden relációnak van veszteségmentes BCNF felbontása (mely nem feltétlenül függőségőrző). Ha egy reláció BCNF alakban van akkor a reláció 3NF-ban is van. Ha egy reláció 3NF-ban van és csak egyszerű kulcsok vannak vagy összetett kulcsok közt nincs attribútum átlapolódás akkor a reláció BCNF formában is van.

A normalizációs algoritmus az 1NF formából indul ki és a felírt funkcionális függőségek alapján, ellenőrzi a normál forma definíciókat. Azon függőségek mentén, melyek sértik az adott normál forma definícióját új relációt definiál, valamint az eredeti relációból törli a függőség jobb oldalán szereplő attribútumokat. Azaz az egyes relációkat több kisebb relációra darabolja fel a függőségek mentén.

2.5.3 Denormalizáció

A relációs séma normalizálása során az adatmodellben található anomáliák csökkentése a cél, ez viszont azzal jár, hogy a relációk száma csökken és az egyes lekérdezésekre adott válaszok táblák összekapcsolásával lehet meghatározni. A táblák összekapcsolása időigényes művelet éppen ezért előfordulhat, hogy nagyobb adatmennyiségek és összetett lekérdezések esetén a kívánt válaszidőket nem tudja az adatbázis-kezelő szerver biztosítani. Ezekben az esetekben a normalizált adatmodellt szándékosan megváltoztatva denormalizált adatmodellt hozunk létre. Az alkalmazott denormalizációs technológiák:

- Felfelé történő denormalizáció: Ebben az esetben a gyerekrekordokból számítható vagy származtatható információ megjelenik a szülő rekordban. P.l: Egy számla fejléc tartalmazza a számla végösszegét is.
- Lefelé történő denormalizáció: A szülő rekordból származó egyes attribútumok megjelennek a gyerek rekordokban. P.l: Egy számla tétel rekordban megjelennek a vevő adatok is.
- Oldalra történő denormalizáció: Egy rekordon belüli adatokból is meghatározható érték külön letárolásra kerül a rekordban. Ez a módszert általában akkor használatos ha keresési kritériumok közt szerepel egy összetett kifejezés, melyre indexet szeretnénk készíteni a gyorsabb keresés miatt.

A bemutatott denormalizációs lehetőségekkel teljesítménynövekedést lehet elérni, viszont könnyen inkonzisztens adatbázishoz juthatunk, ezért ezen mezők karbantartásáról külön gondoskodni kell.

3 Adatbázisok szerver oldali programozása

Többrétegű vállalati információs rendszer tervezése során fontos feladat elkülöníteni az egyes komponensek által megvalósítandó feladatokat. A tervezés során szükségesek azon alapismertetek, melyek segítenek eldönteni, hogy az egyes eszközökkel milyen bonyolultságú probléma oldható meg. A mai adatbázis-kezelő rendszerek általában biztosítanak eszközt arra, hogy az adatbázis oldal komplex szolgáltatásokat tudjon nyújtani.

3.1 Szerver-oldali programozás előnyei

Szerver oldali programmodulok használata számos előnnyel jár, éppen ezért került egyre inkább előtérbe a használatuk:

- Teljesítménynövekedés: Kevesebb információt kell átvinni a hálózaton. Összetett programlogika valósítható meg szerver oldalon, nem kell feleslegesen adatot mozgatni a kiszolgáló és a felhasználó között.
- Lefordított és optimalizált lekérdezéseket tárol az adatbázis-kezelő szerver. A szerver memóriájába betöltött tárolt eljárásokon több folyamat is osztozhat.
- Biztonság: Az egyes felhasználóknak nem kell egyenként jogosultságot adni az egyes táblákra, hanem csak tárolt eljárások futtatására elég megadni a jogosultságot. Ebben az esetben nem tudják megkerülni a tárolt eljárásokban leprogramozott ellenőrzéseket.
- Termelékenység: Az egyes kliensek osztoznak a szerver oldalon lehelyezett programmodulokon, azaz a közös funkciókat csak egyszer kell megvalósítani, melyeket aztán közösen tudnak használni.
- A programlogika változása esetén a változtatást csak egyszer kell megvalósítani, ráadásul az összes kliens számára módosul a logika.
- Adatintegritás: Az adatintegritási szabályok közül a működési korlátozásokban megfogalmazott kritériumok túlmutatnak a relációs adatmodellen. Ezek betartására és betartatására csak programlogikával van lehetőség. Ezt a programlogikát célszerű minél közelebb vinni az adatbázishoz a konzisztencia könnyebb fenntartása miatt.

3.2 Szerver-oldali programozás hátrányai

A felsorolt előnyök mellett néhány hátrányát is fel lehet sorolni a szerver-oldali programozásnak:

- Adatbázis függés: Miden adatbázis kezelőnek saját programozási nyelve van. Ezáltal az elkészült rendszer szorosan integrálódik a választott szerverhez, így nehezebb többplatformos környezetet kialakítani: minden platformra el kell készíteni a program modulokat, a párhuzamos tesztelésükről, integrációjukról és karbantartásukról is gondoskodni kell.
- Adatbázis-kezelő szerver terhelése: Az egyes felhasználók a szerver erőforrásait használva futtatják a programmodulokat, ezáltal megnőhet az adatbázisszerver terheltsége.

3.3 Szerver-oldali programozás eszközei

Számos közös vonást mutatnak az egyes adatbázis-kezelő rendszerek szerver oldali programmoduljai. Közösen megtalálható programmodulok jellemzően a következők:

- Tárolt eljárás
- Tárolt függvény
- Trigger

A szerver oldali programmoduloknak másik közös tulajdonsága, hogy nem tisztán SQL nyelvi elemeket tartalmazhatnak, hanem számos elem megjelenik a procedurális nyelvekből:

- Változók
- Elágazások
- Ciklusok
- Kivételek

SQL parancsok közül általában csak a DML nyelvi elemek, valamint a tranzakció vezérlő utasítások érhetők el.

3.3.1 Tárolt eljárások

Tárolt eljárás tulajdonképpen az adatbázis-kezelő szerveren futó nevesített utasítás sorozat. Jellemzően összetett műveletek megvalósítására szolgálnak. Működésüket tekintve két típust különböztetünk meg:

- Belső tárolt eljárások
- Külső tárolt eljárások

3.3.1.1 Belső tárolt eljárások

A belső tárolt eljárások olyan tárolt eljárások, melyet az adatbázis-kezelő szerver programozási nyelvén definiálunk. Az egész tárolt eljárás futtatását a szerver végzi. Ebből adódóan adatbázis-kezelésen túlmutató feladatok megvalósítására nem alkalmasak. A belső tárolt eljárásokat az adatbázis-kezelő szerver, jellemzően előfordított, és előfeldolgozott állapotban tárolja. Futtatás során ezen előfeldolgozott kódot értelmezve hajtja végre a programmodult.

3.3.1.2 Külső tárolt eljárások

A külső tárolt eljárások készítése tetszőleges programozási nyelven történhet, az adatbázissal való kapcsolattartáshoz az egyes szerverek speciális programozási felületet biztosítanak. Mivel tetszőleges programozási nyelven készülhetnek ezért olyan feladatokat is meg lehet velük oldani, amely túlmutatat az adatbázis-kezelő rendszeren. Például egy könyvtári rendszerben a tagokat elektronikus levélben értesítjük, ha megérkezett az általuk előjegyzett könyv. Ebben az esetben az email küldés túlmutatat az adatbázis-kezelők programozási lehetőségein.

Mivel a külső tárolt eljárások kódját nem ismeri és nem is tudják a futását felügyelni az adatbázis-kezelő szerverek ezért ezek futtatása kritikus lehet a szerver számára. Egy esetleges súlyos hiba esetén az adatbázis-kezelő rendszer összeomlását is okozhatják, ezért ezek futtatása minidig elkülönítve történik. Azaz a külső tárolt eljárásoknak egy külön processzt indít az adatbázis-kezelő szerver, ami azt is jelenti, hogy a külső tárolt eljárások külön címtartományban futnak, ezáltal az adatbázis-kezelő rendszer védve lesz az esetleges programhibáktól.

3.3.2 Tárolt függvények

A tárolt függvények nagymértékben hasonlítanak a tárolt eljárásokhoz, az egyetlen különbség az, hogy van visszatérési értékük. Azaz alapvetően értékkiszámításra használják a tárolt függvényeket. Jellemzően minden olyan kifejezésben szerepelhet függvényhívás, ahol a visszatérési értékének megfelelő típusú kifejezés elfogadható, kivéve az értékadó utasítás baloldala. A tárolt függvények használatával kapcsolatban fontos tisztázni azok hatását egy-egy utasítás végrehajtására pl.:

```
select *
from emp
where GetYear(birthdate)=1980
```

A fenti lekérdezésben a GetYear egy általunk készített függvény, amely egy dátumból kinyeri az évszámot. A probléma a fenti lekérdezéssel az, hogy a where feltételt a lekérdezés optimalizáló csak úgy tudja kiértékelni, hogy az emp tábla összes sorára lefuttatja a függvényt. Ebből adódóan a lekérdezés végrehajtása egy összetettebb esetben, illetve nagy rekordszám esetén nagyon lassúvá válik.

3.3.3 Triggerek

3.3.3.1 Triggerek használata

A triggerek tulajdonképpen eseménykezelő tárolt eljárások. Azaz az adatbázis-kezelő rendszeren belül fellépő különböző eseményekhez tudunk eseménykezelő tárolt eljárást rendelni. Ezek az eseménykezelők automatikusan lefutnak, a felhasználóknak rendszerint nincs lehetőségük (jogosultságuk) ezen eseménykezelők lefutását megakadályozni. Triggerek tipikus alkalmazási területei:

- Származtatott értékek kiszámítása, karbantartása
- Érvénytelen tranzakciók kiszűrése
- Bonyolult biztonsági ellenőrzés
- Referenciális integritás szerverek között
- Működési korlátozások biztosítása
- Naplózás
- Auditálás
- Replikáció
- Statisztika-gyűjtés

3.3.3.2 Trigger Események

Triggerek a következő eseményekhez kacsolódhatnak:

- DML utasítások
    - Update: Egy táblán update utasítást hajtottak végre.
    - Delete: Egy táblából töröltek rekordokat.
    - Insert: Egy táblába új rekordokat vittek fel.
    - Select: Egy táblából sorokat választottak ki. Csak kevés adatbázis-kezelő támogatja, általában nem lehet Select utasításhoz triggert kapcsolni.
    - Instead of: Nézetek használata esetén, akkor hívódnak meg, ha egy nézeten adatmódosító utasítást hajtanak végre.

- DDL utasítások
    - Create: Felhasználói objektumot hoztak létre.
    - Alter: Felhasználói objektumot módosítottak.
    - Drop: Felhasználói objektumot töröltek.
- Rendszeresemények:
    - Logon/Logoff: Felhasználó belép a szerverre, felhasználó kilép a szerverről.
    - Restart: Az adatbázis-kezelő szerver újraindult.
    - Server error: A szerveren fellépő különböző hibákhoz (hibakódonként általában) lehet triggert kapcsolni.

Természetesen nem mindegyik esemény kezelését támogatják az egyes rendszerek, nagymértékben gyártófüggő. Egy adott eseményhez általában több trigger is kapcsolható, viszont a triggerek lefutási sorrendje általában nem befolyásolható.

3.3.3.3 Triggerek végrehajtása

A triggerekhez kapcsolódóan az is fontos kérdés, hogy milyen részletességgel lehet megkapni az egyes eseményeket. Ez főleg DML triggerek esetén lehet hangsúlyos, mivel egy utasítás több rekordot is érinthet, ráadásul meg lehet különböztetni azt is, hogy az adatmódosítás előtt vagy után fut le egy trigger. Ebből kifolyólag a lehetséges típusok:

- Utasítás előtti: olyan trigger, mely a teljes SQL utasításra vonatkozik és az utasítás végrehajtása előtt hajtódik végre.
- Utasítás utáni: olyan trigger, mely a teljes SQL utasításra vonatkozik és az utasítás teljes végrehajtása után hajtódik végre
- Rekordmódosítás előtti: minden módosított rekordra végrehajtódik, a rekord adatainak módosítása előtt.
- Rekordmódosítás utáni: minden módosított rekordra végrehajtódik, a rekord adatainak módosítása után.

A DDL és rendszereseményekhez kötődő triggerek esete sokkal egyszerűbb, mivel csak esemény bekövetkezte után lefutó triggert lehet definiálni.

Természetesen az is fontos, hogy az egy eseményhez definiált triggerek miképp hajtódnak végre, valamint az adatok megváltoztatása és a konzisztencia ellenőrzés mikor történik. Az egyes triggerek lefutása az alábbi sorrendben hajtódik végre:

1. Az összes utasítás előtti trigger végrehajtódik
2. Az összes sor módosítás előtti trigger
3. A rekord módosítása, konzisztencia ellenőrzés, zárolás
4. Az összes sor módosítás utáni trigger
5. Az összes utasítás utáni trigger végrehajtódik

A 2,3,4-es lépés ciklikusan ismétlődik minden egyes rekordra, melyet érint az adatmódosító utasítás. Fontos megjegyezni, hogy az adatmódosítást megelőző triggerek esetében az adatbázis-kezelő szerver még nem végzi el a konzisztencia ellenőrzést és a zárolási feladatokat.

3.3.3.4 Triggerek és tranzakciók

Fontos kérdés még az is, hogy az egyes triggerekben kiadott adatmanipulációs utasítások miképp viszonyulnak a felhasználók tranzakcióihoz.

- DML triggerek esetében, részét képezik annak a tranzakciónak, melyben a DML utasítás fut. Így az ott kiadott adatmanipulációs utasítások sorsa (commit/rollback) összekapcsolódik a triggert kiváltó utasítás sorsával.
- DDL utasítások esete különbözik a DML utasításoktól, mivel a DDL utasítások tranzakciótól függetlenül hajtódnak végre. Ezért inkonzisztenciához vezetne, ha a felhasználó futó tranzakciójához kötődnének, így a DDL triggereknek az adatbzázis-kezelő szerver nyit egy külön tranzakciót a DDL utasítást kiadó felhasználó nevében, és automatikusan commit utasítással le is zárja azt.
- Rendszeresemények hez kötődő triggerek esetében még sok esetben felhasználói tranzakcióról sem lehet beszélni, így ebben az esetben az adatbázis-kezelő szerver egy külön tranzakciót nyit a rendszerfelhasználó (adatbázis-szerver tulajdonos, rendszergazda) nevében és automatikusan commit utasítással zárja le.

4 Adatbázisok elérése

Adatbázis-szerverekkel kapcsolatban fontos kérdés, hogy miképpen kapcsolódnak az egyes kliensalkalmazások az adatforráshoz. Az egyes szoftverfejlesztési környezetek számos hasonlóságot mutatnak a tekintetben, hogy miképp történik a kapcsolódás az adatbázis szerverhez.

4.1 Kapcsolódás az adatbázisszerverhez

A kapcsolat felépítésének egyik legkézenfekvőbb módja, hogy az adatbázist egy hálózati kiszolgálónak tekintjük, és a teljes adatkommunikációs protokollt implementáljuk.

Ebben az esetben a teljes hálózati kommunikációs protokollt implementálni kell, ezen megoldás számos hátránnyal jár:

- A kommunikációs protokollt az adatbázis-kezelő szerver gyártója definiálja, protokoll gyártó függő, éppen ezért minden szervertípushoz saját protokoll implementációt kell készíteni.
- Amennyiben a gyártó módosítja a kommunikációs protokollt (p.l.: verzióváltáskor), akkor a változásokat követni kell a kommunikációs protokoll kliens oldali implementációjában is.
- Egy adatbázis szerver esetén is több különböző implementációra lehet szükség, mivel elképzelhető, hogy a szerverhez való kapcsolódás több különböző hálózati vivőprotokollon keresztül is elképzelhető.

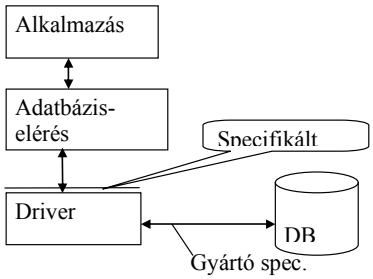
Éppen azért, hogy a protokollt ne kelljen minden kliensalkalmazásban implementálni, az adatbázis-kezelő szerverekhez általában készít a gyártó egy protokoll implementációt, így a gyártó által szállított library segítségével lehet kapcsolódni az adatforráshoz. Ezen megoldás előnye, hogy a protokoll implementációt a gyártó készíti el, és biztosítja a verziókövetést és karbantartást. Viszont továbbra is hátrány, hogy az egyes gyártók által készített osztálykönyvtárak teljesen különbözőek lehetnek egymástól.

4.2 Szabványos osztálykönyvtárak használata

A szabványos osztálykönyvtárak pont arra a problémára adnak megoldást, hogy a fejlesztőknek csak adatbázis elérési API-t kelljen megtanulniuk, mely független az adatbázis-kezelő szervertől.

4.2.1 Adatbázis meghajtó (driver) használata

Az alábbi ábra az adatbázis meghajtóval kiegészített szabványos osztálykönyvtár felépítését mutatja.



Természetesen az egyes osztálykönyvtáraknak is valamilyen módon kapcsolódniuk kell az adatbázisszerverhez. A kapcsolat az úgynevezett adatbázis meghajtón keresztül történik. A meghajtó hívási interfészét az osztálykönyvtár gyártója definiálja, azaz rögzíti, hogy milyen szolgáltatást vár el az adatbázistól, valamint azt is rögzíti, hogy ezeket a szolgáltatásokat miképp tudja elérni. Azaz az osztálykönyvtáron belül az adatbázis meghajtó tulajdonképpen egy interfész specifikációt jelent. Ezt a specifikált interfészt kell implementálnia az egyes adatbázis meghajtóknak, magukat a meghajtókat az adatbázis-kezelő szerver gyártója implementálja általában.

4.2.2 Adatbáziskliens szerepe

Az osztálykönyvtáraktól és meghajtó programoktól függetlenül szükséges általában az adatbázis kliens is az adatbázis-kezelő szerver eléréséhez. Tulajdonképpen az adatbázis kliens tartalmazza a tényleges protokoll implementációt, valamint a kliens tartalmazza a protokollokhoz kapcsolódó szükséges egyéb szolgáltatásokat (p.l.: névfeloldás, hálózati vivő protokoll beállításai,…). Az egyes osztálykönyvtárakhoz tartozó meghajtók az adatbázisklienssel veszik fel a kapcsolatot, így a gyártóknak is csak egyszer implementálják a hálózati kommunikációs protokollt, az összes adatbázis driver (általában) ugyan azt a kommunikációs protokoll implementációt használja.

4.2.3 Osztálykönyvtárak felépítése

A fejlesztési környezettől függetlenül gyakorlatilag az összes osztálykönyvtár felépítése teljese hasonló:

- Connection objektum: az adatbázis kapcsolatért felelős. Az egyes kérések ezen a kapcsolaton keresztül jutnak el a szerverre
- Command objektum(ok): az adatbázis-kezelő szerver felé továbbítandó SQL parancs(ok) összeállításáért felelős.
- Recordset objektum: egy lista jellegű objektum, mely egy lekérdezés eredményeképp kapott rekordokon történő navigálásért felelős.
- Exception objektum(ok): az esetlegesen fellépő hibák, kivételek kezelésére (leírására) szolgáló objektum csoport.

4.3 JDBC felépítése

Java fejlesztői környezetben az adatbázis kapcsolatok leírásáért felelős objektumokat és interfészeket a java.sql csomag tartalmazza. A JDBC a Java DataBase Connectivity rövidítése.

4.3.1 JDBC driver

A JDBC csomaghoz kapcsolódóan több JDBC meghajtó program típust lehet megkülönböztetni:

- JDBC — saját protokoll meghajtó: Ebben az esetben a teljes protokoll implementáció Java-ban készült, és közvetlenül kapcsolódik az adatbázis szerverhez. A megoldás előnye, hogy teljesen Java-ban készült, nem szükséges egyéb szoftver a kapcsolat felépítéséhez.
- JDBC — adatbáziskliens átjáró: Ez a meghajtó program az adatbázis klienssel veszi fel a kapcsolatot, azaz a JDBC hívásokat alakítja át közvetlen kliens hívásokká. Ebben az esetben szükséges kliens oldalon az adatbáziskliens telepítésére.
- JDBC — ODBC átjáró: ODBC meghajtó programot használja fel adatbázisok elérésére.

A driver manager objektum felelős az egyes meghajtó programok nyilván tartásáért és betöltéséért, valamint a kapcsolatok felépítéséért (a meghajtókkal együtt).

4.3.2 Tranzakció kezelés

A tranzakció kezeléshez kapcsolódó metódusok a connection interfészen jelennek meg, mivel a tranzakció magához az adatbázis kapcsolathoz kacsolódik. Egy új kapcsolat automatikus nyugtázással jön létre alapesetben, azaz minden egyes kiadott utasítás commit-tal zárul amennyiben sikeresen lefutott, és rollback-et hajt végre ha hiba lép fel. Tranzakció kezeléshez kapcsolódik, hogy ezen automatikus üzemmód kikapcsolható, ebben az esetben a commit és rollback utasítások manuálisan kiadhatók. Tranzakció kezeléshez szorosan kapcsolódik az izolációs szint megválasztása is (habár ehhez szerver oldali támogatás is szükséges).

4.3.3 SQL utasítások végrehajtása

A JDBC API biztosítja, hogy tetszőleges SQL utasítást (szöveges parancsot) eljuttassunk a szerverre. Az egyes utasításokat az alábbi interfészek segítségével lehet végrehajtani:

- Statement interfész: Egyszerű SQL utasítások végrehajtására szolgál.
- PreparedStatement interfész: Bemenő paraméterekkel is rendelkező SQL utasítások végrehajtására használható.
- CallableStatement: Tárolt eljárások végrehajtására használható.

Az egyes interfészeket implementáló objektumokat a connection objektumon keresztül lehet létrehozni, a createStatement, prepareStatement, prepareCall metódusok segítségével. A fenti interfészek alapmetódusai a következők:

- executeQuery: olyan SQL utasítás végrehajtása mely több rekordot is visszaadhat (Select utasítás), a választ egy ResultSet interfészt implementáló objektumon keresztül lehet elérni.
- executeUpdate: Olyan SQL utasítás mely nem ResultSet jellegű eredményt szolgáltat (Insert, Update, Delete)

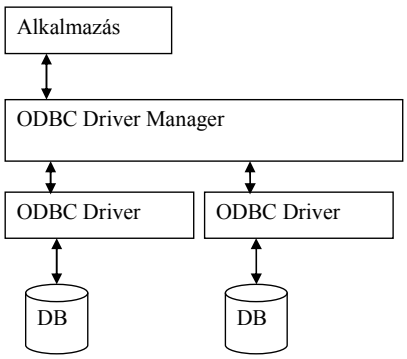
A kódbiztonsághoz kacsolódó kérdéskör az is, hogy a paramétereket tartalmazó SQL utasítások esetén mindenképpen ellenőrzni kell az input paramétereket, SQL injection támadás elkerülése miatt. Ez az ellenőrzés fejlesztői környezettől és platformtól függetlenül szükséges. A PreparedStatement interfész esetén az ellenőrzés automatikusan megtörténik, így elkerülhetők az ilyen jellegű támadások. A preparedstatement objektumok másik nagy előnye, hogy az objektum létrehozásakor már a lekérdezést átküldi a szerverre és megkezdődhet a lekérdezés előzetes feldolgozása (pl.: végrehajtási terv készítése).

4.3.4 ResultSet használata

A ResultSet interfész segítségével lehetséges egy lekérdezés eredményhalmazán végignavigálni. Tulajdonképpen az eredményt adó rekordok egy verem jellegű objektumba kerülnek, melynek egy elemét tudja egyszerre mutatni. Alapesetben csak egy irányban lehet mozogni rajta és az adatai nem módosíthatók. De megfelelő driver támogatás esetén lehetséges kétirányban navigálható ResultSet is készíthető, valamint lehetséges módosítható ResultSet készítése, azaz ha felülírjuk az egyes attribútumait egy resultSet-nek akkor azt vissza lehet írni az adatbázisba. Azt, hogy milyen típusú ResultSet-et hozunk létre, azt a Statement objektum létrehozásakor kell megadni. Módosítható ResultSet esetén az is definiálható, hogy a módosított rekordot ellenőrizze, hogy azt esetleg más felhasználó még nem módosította.

4.4 ODBC használata

ODBC (Open DataBase Connectivity) MS platformon az egyik legrégebbi szabványos library, mely adatbázis kapcsolatok kezelésével foglalkozik. ODBC API nem objektum orientált, hanem C jellegű interfészleírást tartalmaz. Alapvetően csak DML utasítások végrehajtását támogatja, ma már önmagában általában nem használatos, tulajdonképpen csak a széled meghajtó program ellátottság miatt maradt fent.



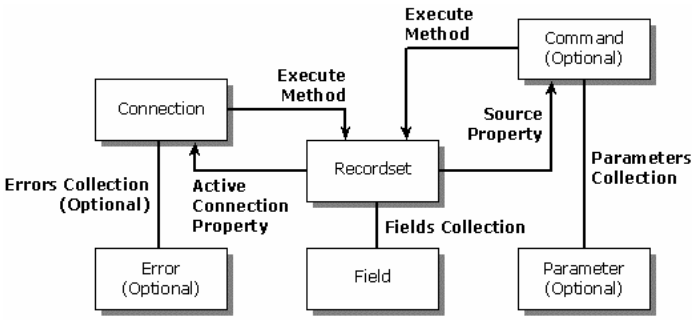
Az alkalmazások közvetlenül a driver manager-rel veszik fel a kapcsolatot, mely betölti a megfelelő meghajtót. Maga a dirver manager az operációs rendszer része, külön konfigurálható, hogy az egyes adatbázis hivatkozásokhoz milyen meghajtók töltődjenek be, valamint a meghajtó paramétereket is külön lehet beállítani.

4.5 OLE DB / ADO használata

Az OLEDB adatelérés alapvetően COM objektumokra épül. Az OLEDB komponenst definiál:

- OLE DB Data Provider: Adatszolgáltatásért felelős COM interfész. Az adatbázis szerver gyártójának feladata egy COM objektum készítése, mely implementálja ezt az interfészt. Igazából ez a komponens tölti be az adatbázis meghajtó szerepét.
- OLE DB Data Consumer: COM interfész, melyet az adatbázishoz kapcsolódó alkalmazásnak kell implementálja.
- OLE DB Service Component: Olyan funkcionalitásokat lehet implementálni, melyeket esetleg néhány egyszerűbb OLE DB Driver nem támogat (pl: rendezés, szűrés, keresés). Ezeket külső service komponensként rendelkezésre állnak, így egységes felületet nyújtanak felfelé az OLE DB Provider-en keresztül az egyes adatbázisszerverek.

Mivel COM komponensek és interfészek készítése nem mindig egyszerű feladat ezért az OLE DB fölé (más adatforrásra is épülhet) definiálták az ADO (Active Data Object) objektum könyvtárat, mely COM komponensekből épül fel. Az alábbi ábra az ADO objektumok egymáshoz való viszonyát mutatja.



Az ADO objektummodell részei:

- Conncection: Adatbázis kapcsolatért felelős. Betölti a szükséges adatbázis meghajtót (általában OLE DB Provider-t). Ezen keresztül hajtódnak végre az egyes utasítások, valamint a tranzakció kezelő utasítások kiadását is biztosítja.
- Command: Tetszőleges SQL utasítást reprezentálhat, beleértve tárolt eljárást is
- Paramter: Amennyiben paramétereket is tartalmazó SQL utasítást készítünk, akkor az egyes paraméterek leírását tartalmazó objektum. Paraméteren keresztül lehet megadni a tárolt eljárások ki és bemenő paramétereit
- Recordset: Egy SQL lekérdezés eredmény rekordjain navigáló objektum.
- Field: Az aktuális rekord egy attribútuma.
- Error: Egy hiba leírását tartalmazó objektum.

- Errors collection: Hibák gyűjteménye, melyeket a provider jelzett az utolsó utasítás óta.
- Fields Collection: A recordsethez kapcsolódó mezők gyűjteménye.
- Parameters Collection: A command objektumhoz tartozó paraméterek gyűjteménye.

Fentiek az alapobjektumai az ADO objektummodellnek, viszont a modell tartalmaz támogatást nem rekord alapú adatforrások kezeléséhez is.

5 Adattárházak

A vállalatok a napi ügymenet meggyorsítása és adminisztrálása érdekében számos szigetszerű nyilvántartó alkalmazást fejlesztettek ki. Ezek az alkalmazások általában kielégítők abban a tekintetben, hogy egy részprobléma (részfeladat) adminisztrálását elvégzik, viszont átfogó képet nem képesek szolgáltatni a döntéshozók felé a vállalat aktuális helyzetéről. Másrészről a döntéshozók elvárják (elvárhatják), hogy ha az elemi üzleti folyamatok rögzítésre kerültek, akkor ezekből az alapadatokból kiindulva számos kimutatást, statisztikát lehet készíteni, melyek segítségül szolgálhatnak stratégiai döntések meghozatalában. Éppen ezért azt is lehet mondani, hogy a vállalatok fulladoznak az adatokban, de éheznek az információért. → Szükséges egy integrált, az egész vállalatra kiterjedő jó minőségű információs bázisra, ezt nevezzük adattárháznak (Data Warehouse). Az adattárházakat teljesítmény és működésbeli különbségből adódó okok miatt el kell különíteni a napi üzletmenetet biztosító nyilvántartó rendszerektől. Adattárházak alaptulajdonságai:

- Időfüggő: az adattárházban tárolt adatok időpontokhoz kötöttek, megállapítható, hogy mely időpontra vonatkozik, mikor kerültek be az adatbázisba.
- Változatlan: adattárházakban lévő adatok nem változnak, csak lekérdezést hajtanak végre rajtuk, legfeljebb új időponthoz tartozó rekorddal bővül az adatbázis adatbetöltés folyamán.
- Tematikus: Nem tartalmazzák az operatív adatbázisok teljes tartalmát, csak azokat az adatokat, melyek szükségesek az elemzések elvégzéséhez.
- Integrált: Az egyes elszigetelt rendszerekből származó adatokat egy egységes nyilvántartórendszerbe gyűjti össze.

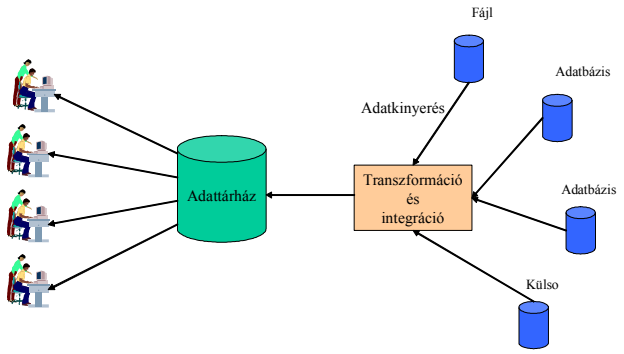
5.1 Tranzakciós és adattárház rendszerek összehasonlítása

Tranzakciós vagy OLTP (On-Line Transaction Processing) rendszereknek nevezzük azokat a nyilvántartó rendszereket, melyek feladat a napi üzleti működés biztosítása. Ezen rendszereknek a feladata az egyes üzleti tranzakciók rögzítése és követése. A döntéstámogató vagy más néven DSS (Decision Support System) rendszerek arra szolgálnak, hogy egy vállalat stratégiai üzleti döntéseinek előkészítését támogassák, valamint segítséget nyújtsanak a vállalat aktuális állapotának az elemzéséhez. Mivel a megcélzott feladatok teljesen különböznek ebből adódóan ezek a rendszerek nagymértékben különböznek egymástól. Az alábbi táblázat néhány főbb szempont szerint összehasonlítja egymással az OLTP és adattárház rendszereket.

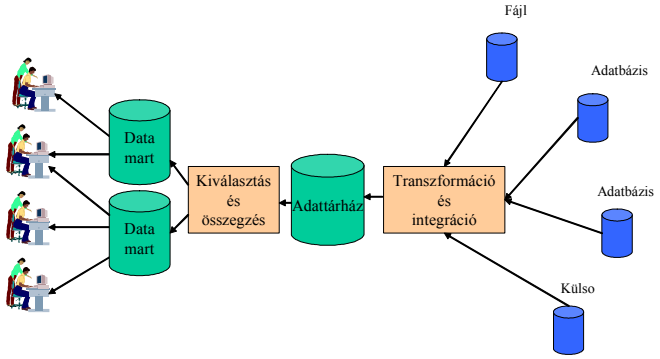
Szempont	**OLTP rendszerek**	**Adattárház**
Elsődleges cél	Aktuális adatok feldolgozása	Döntéstámogatás
Felhasználók	Ügyintézők	Menedzsment
Terhelés jellege	Egyszerű lekérdezések, adatmódosítások	Összetett lekérdezések
Jellemző mérőszám	Feldolgozott tranzakciók száma	Lekérdezésre adott válasz megtalálásának ideje

5.2 Adattárház architektúrák

Az adattárház építése során két architektúrát szoktak megkülönböztetni, a két és háromrétegű adattárházakat. A kétrétegű adattárház építése során az egyes felhasználók közvetlenül az adattárházhoz kapcsolódnak. Az adattárház az OLTP rendszerekből nyeri az elemi adatait, az egyes OLTP rendszerekben megtalálható üzleti tranzakciók kerülnek be az adattárházba. Az egyes felhasználók közvetlenül az adattárházhoz kapcsolódnak.



A háromrétegű adattárház architektúra továbbfejlesztett változata a kétrétegű modellnek. Ebben az esetben az egyes felhasználók nem közvetlenül az adattárházhoz kapcsolódnak, hanem egy ráépülő adatréteghez az úgynevezett adat piacokhoz (data mart) kapcsolódnak. Az adatpiacok egyrészt egy tematikusan kiválasztott részét tartalmazzák az adattárháznak, másrészt nemcsak alapadatokat, hanem származtatott, előkalkulált adatokat is.



5.3 Adattárház építésének lépései

Az adattárház építése során négy alapfeladatot kell megoldani:

- Adatok kinyerése az OLTP rendszerekből.
- Adatok tisztítása.
- Adatok transzformálása.
- Adatok betöltése és indexelése.

5.3.1 Adatok kinyerése

Adatok kinyerése során heterogén adatforráshoz kell kapcsolódni, ezt általában már a korábban megismert adatbázis meghajtók biztosítják. Viszont számos rendszer van, mely nem relációs adatmodellre épül, ezekben az esetekben általában egyedi áttöltésre.fejlesztésekkel lehet megoldani az adatok kinyerését. Adatok kinyerése során meg kell határozni, hogy a tranzakciós rendszerekben található adatok közül mely rekordokra van szükség. Az adattárház építése során az OLTP rendszerekből az összes szükséges adat áttöltésre kerül, későbbiek során csak a változtatások kerülnek

5.3.2 Adattisztítás

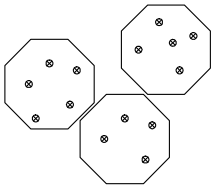
Adattisztítás során az OLTP rendszerekben meglévő adatok minőségének javítása a cél, mivel az üzleti döntések előkészítéséhez jó minőségű, nem ellentmondó adatok szükségesek. Adattisztítás szükséges, mert az OLTP rendszerek általában nem mindig tartatnak be minden adatkonzisztenciát biztosító követelményt, ráadásul az egyes OLTP rendszerekben található adatok ellentmondhatnak is egymásnak. Adattisztítás folyamán két alapfeladat fogalmazható meg: hiányzó adatok kezelése, zajszűrés.

Hiányzó adatok kezelése során a következő megoldások alkalmazhatók általában:

- A teljes rekord elvetése.
- A hiányzó adat bekérése.
- A hiányzó adat helyettesítése
    - Konstans értékkel
    - Átlagértékkel
    - Csoportra jellemző átlagértékkel
    - Legvalószínűbb értékkel

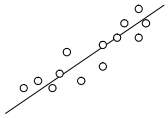
Zajszűrés során használt algoritmusok abból indulnak ki, hogy az OLTP rendszerekből érkező adatok önmagukban nem fogadhatók el, zajjal terheltek. Alapvetően más algoritmusokat használnak szöveges és szám alapú attribútumok zajszűrésére. Numerikus attribútumok szűrése:

- Binning: Az attribútum értékeit egy számsorozatnak tekintik, a sorozat egymás után következő elemeit k elemű csoportokra osztják. A k elemű csoport minden egyes tagját helyettesítik egy a csoport elemeiből kiszámított értékkel. Ez a kiszámított érték lehet az átlagérték, értékben a középső … Így a kiugró értékek eltüntethetők az adatbázisból.
- Csoportképzés (clustering): Általában több attribútum együttes előfordulását vizsgálják, majd az egymás közelében elhelyezkedő adatokat a csoportot jellemző adatokkal helyettesítjük (csoport súlypontjával)



- Regresszió: Ebben az esetben is több attribútum együttes előfordulását vizsgáljuk, az egyes attribútumok közt valamilyen függvénykapcsolatot

keresünk (általában lineárisat → lineáris regresszió), majd az egyes értékeket a regressziós függvény értékeivel helyettesítjük.



Szöveges információk zajszűrése során az egyes szövegekben előforduló elgépelések felderítése a cél. Elgépelések felderítésére alkalmazott technológiák:

- Szótár alapú javítás. Ritkán alkalmazott módszer, előnye, hogy az egyes szövegek a helyesen leírt szóval helyettesítődnek. Általában rögzített, szűk szótárból érkező attribútumok javítására használják
- Klaszterező algoritmusra épülő: Az egyes szövegeket egy hash függvény segítségével számokká lépezzük le. Ezeken a numerikus attribútumokon klaszterező algoritmusok segítségével csoportokat képeznek, majd a csoport minden egyes tagját a leggyakrabban előforduló értékhez tartozó szöveggel helyettesítik. Klaszterező algoritmusok használhatók numerikus adatok szűrésére is.

Mindkét megoldás esetén fontos, hogy a szöveges attribútumokat miképpen képezik le numerikus értékekké, azaz az egymásra hasonlító, egymáshoz közel álló szövegekhez, egymáshoz közel eső numerikus értékek rendelődjenek. Ezen a területen még ma is aktív kutatások folynak.

5.3.3 Adat transzformáció

Adat transzformáció során az adatokat már önmagában elfogadottnak tekintjük, viszont az adattárházban megtalálható egységes sémára transzformáljuk. Transzformáció során az alábbi műveletcsoportokat különböztetjük meg:

- Rekord szintű függvények:
  - Kiválasztás. A rekordok egy részhalmazának a kiválasztása, általában az adatkinyerés része.
  - Összekapcsolás. Több rekord összekapcsolása (join) előfordulhat, hogy több különböző OLTP rendszerben megtalálható rekordokat kell összekapcsolni
- Mező szintű függvények, egy bizonyos adatreprezentációból egy másikra transzformáljuk az egyes attribútumokat (pl: string kódolási-szabvány váltás, Fahrenheit fok, Celsius fok transzformáció):
  - Algoritmikus. A transzformációt algoritmussal lehet megadni.
  - Lekérdezés/ Tábla alapú: A transzformációt az adatbázisban megtalálható adatok alapján lekérdezés alapján lehet elvégezni (pl: Ország kód → Ország)

5.3.4 Adatok betöltése és indexelése

Adatok betöltése során megkülönböztetik pillanatkép és historikus adatokat. Pillanatkép adatok esetén felülírják az adattárházban meglévő adatokat az aktuális adatokkal, historikus adatok egy időbélyeggel ellátva kerülnek be az adattárházba, így követhető az egyes rekordok értékeinek változásai. Az adatbetöltés során az

adattárházban megtalálható indexeket is újra kell építeni. Az adattárházakban gyakoriak a bitmap indexek.

A bitmap indexek olyan attribútumok indexelése alkalmasak, melyek értékkészlete szűk, ritkán változik, intervallum jellegű lekérdezések meggyorsítására alkalmas. Minden rekordhoz járulékos attribútumoknak lehet elképzelni a bitmap indexet, egy attribútum összes értékének megfelelő számú oszlop jelenik meg, és az az oszlop vesz fel logikai 1-et, amelyik a konkrét értéknek megfelel. Az alábbi ábra egy példát mutat a bitmap index működésére.

CUSTOMER #	MARITAL_ STATUS	REGION	GENDER	INCOME_ LEVEL
101	single	east	male	bracket_1
102	married	central	female	bracket_4
103	married	west	female	bracket_2
104	divorced	west	male	bracket_4
105	single	central	female	bracket_2
106	married	central	female	bracket_3

REGION='east'	REGION='central'	REGION='west'
1	0	0
0	1	0
0	0	1
0	0	1
0	1	0
0	1	0

```
SELECT COUNT(*)
FROM CUSTOMER
WHERE MARITAL_STATUS = 'married' AND REGION
  IN ('central','west');
```

status = 'married'		region = 'central'		region = 'west'						
0		0		0		0		0		0
1		1		0		1		1		1
1	AND	0	OR	1	=	1	AND	1	=	1
0		0		1		0		1		0
0		1		0		0		1		0
1		1		0		1		1		1

5.4 Multidimenzionális adatmodell

A származtatott adatrétegben fontos a ráépülő döntéstámogató rendszerek támogatása. A lekérdezések gyors végrehajtása a cél, ezért más szemléletet igényel. A multidimenzionális adatmodell alapelemei:

- Dimenziós adatok: Azon adatok melyek mentén történik az elemzések végrehajtása. A dimenziós adatokat a leíró adatoknak is lehet tekinteni (pl: hely, idő, termék).
- Hierarchia szintek: A dimenziós adatok kvantált és hierarchiába szervezhető adatok (pl: idő dimenzió mentén: nap, hét, hónap, negyedév, év). A hierarchia szintek definiálják az elemzések felbontását (granualitását).

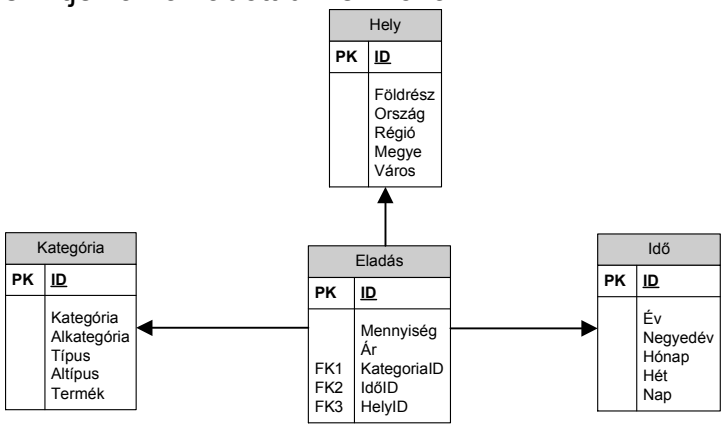
- Tény adatok: Tény adatok rögzítik az egyes üzleti tranzakciókat, a tény adatok definiálják, hogy mit elemzünk.

Multidimenzionális adatmodell relációs adatbázisban történő tárolására kétféle leképzés történhet:

- Csillag sémába
- Hópehely sémába

5.4.1 Csillag séma

A csillag sémában a tény adatokat tartalmazó ténytáblához, idegen kulcsokon keresztül csillagszerűen kapcsolódnak a dimenziós táblák. Minden egyes dimenziónak egy-egy tábla felel meg, a dimenziós táblának annyi attribútuma van ahány hierarchia szintje van az adott dimenziónak.



5.4.2 Hópehely séma

A hópehely séma a csillag séma módosítása, oly módon, hogy dimenziós adatok normalizálása során csökkenthető az adattárház mérete. A normalizáció során a hierarchia szintek mentén feldaraboljuk az egyes dimenziós táblákat és a legalacsonyabb hierarchia szintet képviselő tábla kapcsolódik közvetlenül a tény táblához, a többi szintet külső kulcsok mentén történő join műveletekkel lehet elérni. A fenti példából kiindulva például az idő dimenziót tovább normalizálva oly módon, hogy az év, negyedév, hónap, hét, nap attribútumok külön táblába kerülnek, és idegen kulcsokkal kapcsolódnak össze (és a napot tartalmazó tábla kapcsolódik közvetlenül a tény táblához). A hópehely séma hátránya, hogy megnöveli a végrehajtandó join műveletek számát, ezáltal lassul az egyes lekérdezések végrehajtása.

6 OLAP rendszerek

OLAP (On-Line Analytical Processing) rendszerek alapvetően a multidimenzionális adatmodellre épülő döntéstámogató rendszerek. A cél az információ gyors megtalálása és szemléletes megjelenítése.

6.1 OLAP rendszerek definíciója

Az irodalomban legelfogadottabb definíciója, független a rendszer implementációjától, hanem az elvárt viselkedésből indul ki. A legelfogadottabb definíciók az ún. FASMI teszt, ill. Codd szabályok.

6.1.1 Codd teszt

Az első OLAP definíciós Codd fogalmazta meg, mely a következő 12 szabályt tartalmazza:

- Multidimenzionális koncepciós séma. Adatok elemzése, megjelenítése multidimenzionális nézeteken és műveleteken keresztül történik.
- Átlátszóság, a felhasználó számára átlátszó (lényegtelen), az OLAP rendszer implementációja.
- Elérhetőség. OLAP rendszerek függetlenek a kapcsolódó adatforrástól. OLAP eszközök változatos adatforráshoz kapcsolódva képesek a logikai sémájuknak előállítására.
- Konzisztens válaszidő. Az elemzés komplexitásától független a riportok előállítási ideje.
- Kliens szerver architektúrára épül.
- Egyes dimenziók egymással ekvivalensek adatszerkezeti és elemzési képességek tekintetében. Azaz nincsenek kitüntetett Dimenziók.
- Dinamikus memóriakezelés. Az alkalmazott tárolási struktúra képes kezeli ritka és sűrű adatok kezelését, elfogadható memóriahasználat mellett. Azaz ritka adatmátrixok esetében nem foglal feleslegesen tárhelyet.
- Több felhasználós környezet támogatása.
- Dimenziók közötti elemzések nem korlátozottak, tetszőleges számú dimenzió mentén lehet elemzést végezni.
- Intuitív adatelemzés. A felhasználói felületnek biztosítania kell, hogy az egyes elemzések egyszerűen (cella műveletekkel) könnyen elkészíthetők legyenek.
- Flexibilis riport készítés. A nyomtatandó riportok összeállítása során tetszőleges dimenziók, dimenzió számára vonatkozó korlátozás nélkül felhasználhatók.
- Dimenziók száma és aggregációk száma nem korlátozott.

6.1.2 FASMI teszt

FASMI a Fast Analysis of Shared Multidimensional Information kifejezésből származó rövidítés. Egy rendszert, akkor tekintünk OLAP rendszernek, a FASMI teszt szerint, ha megfelel az alábbi követelményeknek:

- Fast: A rendszer a legtöbb esetben 5 másodpercen belül választ szolgáltat, az egyszerűbb esetekben a válaszidő legfeljebb 1 másodperc, az összetett bonyolult elemzések közül is csak néhány tart 20 másodpercnél tovább.

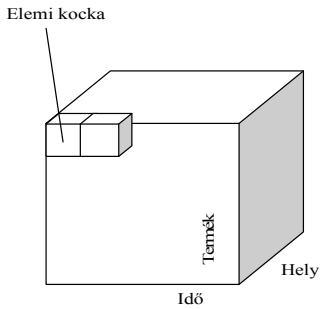
- Analysis: A felhasználó számára szükséges üzleti és statisztikai elemzések elkészítésére alkalmas. Néhány előre elkészített elemzést tartalmazhat, de ad hoc jelleggel is lehessen lekérdezéseket, elemzéseket megfogalmazni. Gyakorlatilag tetszőleges összefüggést meg lehet mutatni, az elemző kívánsága (fantáziája) szerint.
- Shared: Adatok elosztott hozzáférését teszi lehetővé, több felhasználó kapcsolódhat az adatokhoz. Elosztott hozzáféréshez a biztonsági rendszer is hozzátartozik, azaz korlátozható az is, hogy az egyes felhasználók, milyen adatelemekhez férhetnek hozzá.
- Multidimensional: Kulcskövetelménye az OLAP rendszereknek a multidimenzionális adattárolás. A definiált dimenziószám nem fontos, a lényeg az adatmodell és szemlélet.
- Information: Összes adat és származtatott adat elérhető, amelyre szükség lehet. Az OLAP adatbázisnak annyi adatot kell tartalmaznia, mely lehetővé teszi komplex elemzések végrehajtását.

Manapság már a FASMI tesztet használják inkább OLAP rendszerek definiálására.

6.2 OLAP kocka

Az OLAP rendszerek kulcseleme az OLAP kocka, mely az adatelemzési egységet definiálja. Sok esetben az OLAP kocka adattárolási egységet is definiál, de igazából, mint logikai sémát kell elképzelni és használni. A felhasználók számára az elvégezhető elemzéseket definiálja.

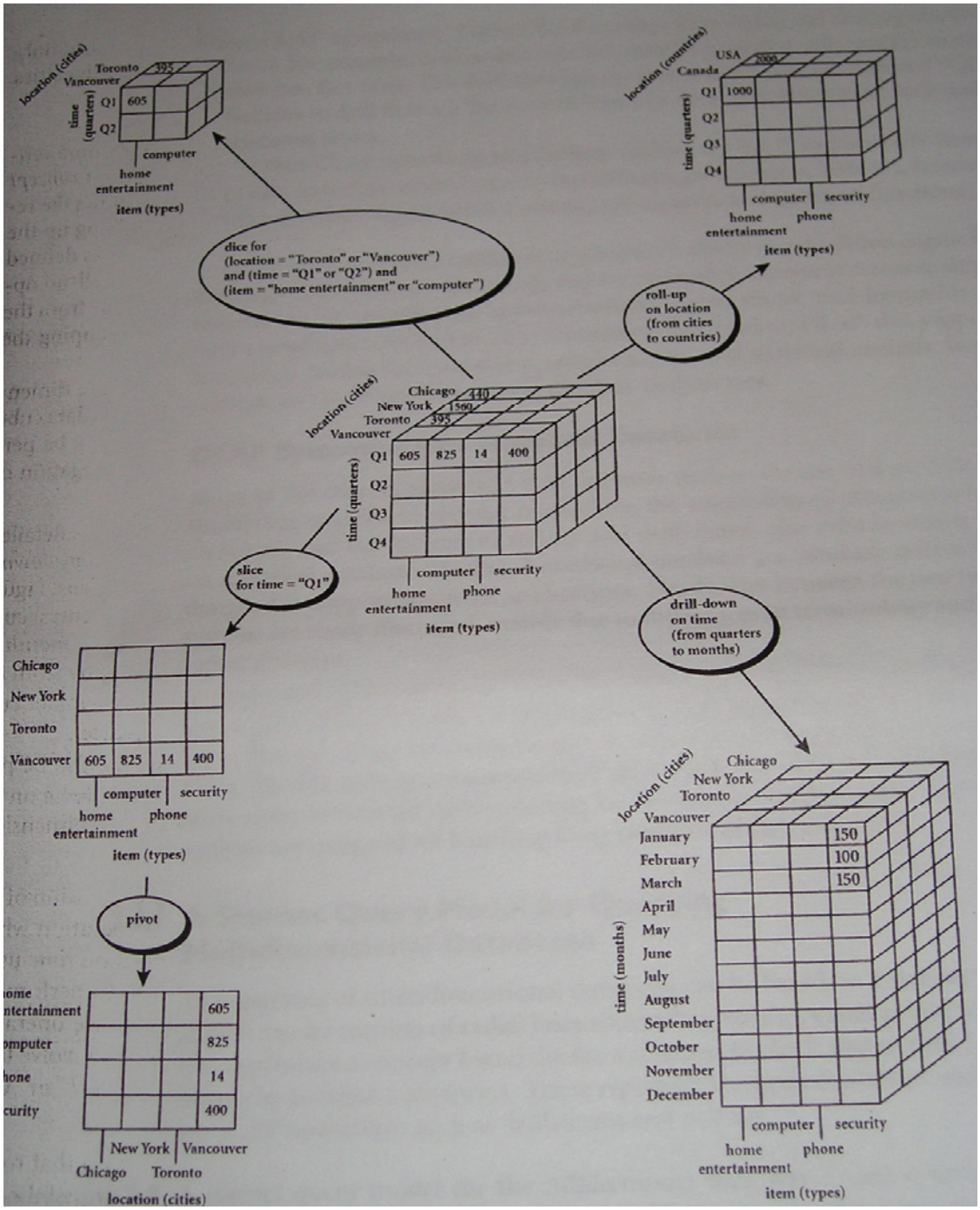
6.2.1 OLAP kocka felépítése



Az OLAP kocka multidimenzionális szemléletre épít, mely az egyes dimenziókból felépülő hiperkockán alapszik. A kocka definiálása során az alábbi elemek használatosak:

- Dimenzió: A hipekocka éleit definiálják. Az egyes elemzések során a definiált dimenziókat lehet felhasználni.
- Hierarchia: Dimenziók mentén értelmezett hierarchia szintek az elvégezhető elemzések mélységét definiálják.
- Aggregátum: A tény adatokból származtatható mennyiségek (pl.: összegzés, átlag …). Az elemzések során az aggregátumok alakulását lehet követni a különböző dimenziók és hierarchiaszintek függvényében. Azaz a dimenziók által kifeszített hiperkocka elemeit képezik az aggregátumok. Maguk az aggregátumok kiszámításának „granulaitása” a definiált hierarchia szintektől függ.

6.2.2 OLAP kocka műveletek



Az elemzések során a felhasználók a definiált OLAP kockán végezhetnek műveleteket, ezen műveletek során nézhetik meg a definiált aggregátumok értékeit. A felhasználók által használható műveletek:

- Szeletelés (slice): Egy dimenzió mentén egy érték rögzítése az elemzés során→ a megjelenítendő eredmény kocka dimenziószáma eggyel csökken.
- Részkocka kiválasztása (dice): Egy vagy több dimenzió mentén értékek megkötése, így az eredeti kocka egy része marad meg.
- Lefúrás (drill down): Egy dimenzió mentén a hierarchiaszint csökkentése (felbontás növelése), azaz a kocka kinyitása, az adatok részletezése.

- Felfelé görgetés (roll up): A lefúrás művelet ellentéte, egy dimenzió mentén a hierarchiaszint növelése (felbontás csökkentése), azaz a kocka összecsukása egy dimenzió mentén.
- Forgatás (pivot): Igazából csak megjelenítést szolgáló művelet, mely a megjelenítendő eredményt elforgatja.

Nem minden OLAP motor által támogatott műveletek:

- Drill across: Olyan művelet, elemzés, mely több ténytáblát érint, OLAP kockák közötti elemzés.
- Drill through: Olyan elemzés, mely a hierarchia legalsó szintje alá menve a tényadatokat is meg tudja mutatni.

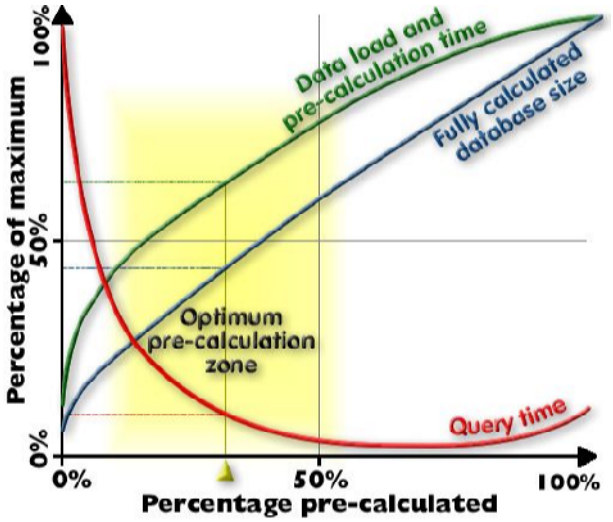
6.2.3 OLAP kocka tárolása

A megfelelő válaszidők biztosítása érdekében, az OLAP rendszerek a kockában definiált aggregátumokat kiszámítják és letárolják. Az OLAP kocka tárolására az alábbi technológiák használatosak:

- ROLAP (Relational OLAP) rendszerek: Relációs adatbázisban tárolják az adataikat.
- MOLAP (Multidiemnsional OLAP) rendszerek: Saját multidimenzionális szemléletet követő tárolási struktúra. A kocka egyes elemei általában közvetlen indexeléssel elérhetők. Általában külön tárolási struktúrában tárolják a ritka és sűrű adatokat helytakarékossági okok miatt.
- HOLAP (Hybrid OLAP): A fenti két rendszer kombinációját használják, így kombinálják a fenti két megoldás előnyeit. Általában a részletezett adatokat ROLAP szerverként a magasabb hierarchiaszinthez tartozó adatokat MOLAP módon tárolják.

6.2.4 Előkalkuláció mértéke

Az OLAP rendszerek a gyors válaszidőt az aggregátumok adatbetöltéskor végzendő előállításával tudják biztosítani. Felmerül a kérdés, hogy a minden hierarchia szinten és dimenzió mentén, teljesen kitöltött kocka előállítása szükséges-e? Magának a teljes kockának az előállítása adatbázis robbanáshoz vezet, mivel a hierarchia szintek és dimenziók számának növelése során a letárolandó adatok mennyisége robbanásszerűen növekszik, gyakorlatban éppen ezért nem végeznek teljes kocka előkalkulációt. Előkalkuláció során másik nagy probléma a ritkán kitöltött dimenziók kezelése, mivel ebben az esetben az előkalkulált kocka nagyon sok olyan részkockát tartalmaz, ahol az aggregátumok nem vesznek fel értéket. Éppen ezért teljes előkalkulációt végezni nem szoktak akkor, ha van kb 5 ritkán kitöltött dimenzió. A következő ábra az OLAP kocka építését, adatkocka méretét és a lekérdezésekre adott válaszidőt mutatja. A sárga terület mutatja az optimumot. Érdekes, hogy a válasz idő nőhet az előkalkuláció függvényében, ennek magyarázata a kocka méretének növekedése, és a kockán belüli navigálásra fordított idő megnövelheti a válaszidőt.



Előkalkuláció során a következő aggregátumokat célszerű előállítani:

- Sok cellát érintő aggregátumok, melyek előállítása időigényes
- Bonyolult kifejezést tartalmazó aggregátumok, melyek számítása időigényes
- Gyakran használt aggregátumok és nézetek
- Alapjai ráépülő kalkulációknak

Teljes előkalkulációt célszerű használni:

- Alacsony dimenziószám esetén (kb 5).
- Kis méretű kockák esetén, ahol a diszk méret nem okozhat gondot.
- Bonyolult kifejezéseket tartalmazó aggregátumok esetén.
- Válaszidő kritikus rendszerek esetén.
- Sok felhasználó esetén (több száz).

7 Adatbányászat

Adatbányászat feladata, hogy az adattárházakban megtalálható nagy mennyiségű adatokon automatikus tudásfeltárást hajtson végre. Szemben az OLAP rendszerekkel, melyek csak az információ megtalálásában segítenek, az adatbányászati algoritmusok feladata, hogy aktívan segítsék az elemzők munkáját, megmutassák az érdekes és fontos információkat. Az adatbányászat legelfogadottabb definíciója az, hogy adatbázisokban tárolt adatokon végzett olyan tudásfeltárás, mely korábban nem ismert, potenciálisan hasznosnak vélt, nem triviálisan előállítható információk megtalálására irányul. Számos alkalmazási területe van, nemcsak marketing stratégiák tervezésénél:

- Kereskedelem: vásárlási szokások követése, célzott ajánlatok készítése
- Pénzügy: Hitelbírálat, bankkártya csalások felderítése
- Orvostudomány: Diagnosztika, tünetek időbeli megjelenésének elemzése (→rákkutatás). DNS láncok elemzése, örökletes betegségek felkutatása, valamint az azokért felelős gének azonosítása (→örökletes cukorbetegség).
- Távközlés, telekommunikáció: Forgalom elemzés segítségével hacker és vírus támadások felderítése.

7.1 Gépi tanulás

Általánosságban is elmondható (nemcsak a gépi tanulásra), hogy egy a valóság egy megfigyelt részéből indulunk ki, a megfigyelt részhalmazból következtetéseket vonunk le, szabályokat fogalmazunk meg, melyek segítségével a valóságban megfigyelhető jelenségek bekövetkeztének az eredményét jósoljuk meg. Az ilyen formában történő tanulásnak a problémája, hogy aszimmetrikus: a megfogalmazott hipotézisek nem igazolhatók 100%-ig, viszont egy ellenpéldával megdönthetők a hipotézisek (pl.: Minden hattyú fehér, egy ellenpélda elég). → A megfogalmazott hipotézisek csak statisztikailag igazolhatók. A tanuló algoritmusokkal kapcsolatos elvárások és alapfogalmak:

- Osztályozás pontossága: Mennyire pontos az algoritmus, az esetek hány százalékában ad helyes választ és hány százalékában hoz téves döntést.
- Statisztikai szignifikancia: A tanuló programot akkor fogadunk el használhatónak, ha az általa adott válasz jobb (nagyobb statisztikai valószínűséggel ad helyes választ), mint amit véletlen találgatással el lehet érni. Például ha állatok osztályozása a cél, öt csoportot különböztetünk meg, a nem kenguru szabályt 80%-os valószínűséggel akkor is eltalálhatjuk, ha semmilyen tanulást nem végzünk.
- Keresési tér bonyolultsága: A keresési tér (lehetséges hipotézisek száma), lehet egészen egyszerű, de végtelen számú hipotézist tartalmazhat. Sajnos a gépi tanuló algoritmusok nem tudnak mit kezdeni a bonyolult keresési terekkel, ezért egyszerűsítéseket kell végezni, a hipotézisekhez mérőszámokat rendelünk, és csak a megfelelően nagy mérőszámok közelében végzünk részletesebb keresést. A keresés gyorsítása érdekében különböző heurisztikákat használnak, melyek azok a keresési irányok mely mentén nem várható eredmény. A keresési térben alkalmazott heurisztikák általában csak akkor működnek, ha az egyes hipotézisekhez tarozó mérőszámok jól megkülönböztetik a hipotéziseket, és nincsenek nagy

változások, a probléma hasonlít a gradiens módszer nehézségeire, mivel a keresés során gradiens módszerre épülő szélsőérték keresést szoktak használni. Sajnos a módszerek csak a lokális szélsőértékeket találják meg → több pontból kell indítani a keresést.

- Felügyelt és nem felügyelt tanulás: A felügyelt tanulás során emberi beavatkozás segítségével történik a tanulási folyamat ellenőrzése és vezérlése. Nem felügyelt tanulás esetén a tanuló algoritmus a megadott minták alapján automatikusan tanul.

7.2 Adatbányászati alapfeladatok

Adatbányászati projektek megvalósítása során fontos tisztázni, hogy mi a megoldandó feladat és a megfelelő adatbányászati módszert és eszközrendszert kell hozzátársítani. Az egyes lehetséges adatbányászati alapfeladatok az alábbiak lehetnek:

- Asszociációs szabálykeresés: Ha akkor jellegű szabályok megtalálására szolgál, melyek egy időpontban történő eseményekhez kapcsolódónak. Tipikus felhasználási területe a bevásárlói kosár elemzése (aki sört vesz az chipset is vásárol).
- Sorrendi elemzés: Egymás után bekövetkező események elemzése, fontos tényező az idő dimenzió. Sorrendi elemzés során fontos tényező az időablak mérete, azaz milyen időintervallumon belül előforduló események egymásutánisága érdekes számunkra. Tipikus alkalmazási terülte az orvosi diagnosztika (tünetek egymás után történő megjelenése) , hálózati és egyéb riasztások kezelése (egy atomerőműben egy meghibásodás során az első percekben több tízezer hibajelzés és érzékelő adat futhat be), de kereskedelemben is használatos aki egy adott terméket vásárol az rövid időn belül egy másik terméket is vásárol (→ pontgyűjtő kártyaakciók).
- Eltérés elemzés: Sok esetben épp az átlagostól, szokásostól való eltérő viselkedés megtalálása a feladat. Tipikus alkalmazása a rosszindulatú viselkedési formák felderítése (vírustámadás, behatolási kísérlet, DOS támadás).
- Osztályozás: Osztályozás során az egyes elemek tulajdonságaik alapján történő csoportosítás a feladat. A kiindulás során a tanuló mintát manuálisan csoportokba sorolják, valamint minden egyes mintaelemhez megadják a leíró tulajdonságaik mely értékeket vesznek fel. A tanulási folyamatot követően egyes új elemeket kell a begyűjtött tulajdonságaik alapján kell csoportokba rendezni. Tipikus alkalmazási területek: hitelbírálat, biztosítási kockázatelemzés.
- Klaszterezés: Klaszterező algoritmusok hasonló feladatot látnak el mint az osztályozó algoritmusok, viszont nem felügyelt tanulási algoritmusok. A cél a csoportok meghatározása, megtalálása, mely mintaelemek tartoznak egy csoportba. Általában mérési eredmények elemzésére, zajszűrésre használják.

7.3 Asszociációs szabályok keresése

Asszociációs szabályok keresése az egyik alapfeladata az adatbányászati algoritmusoknak, számos más feladat (osztályozás, sorozatelemzés) vezethető vissza rá.

7.3.1 Termékhalmazok és bevásárlói kosarak

Az asszociációs szabály keresés modellezése és leírása bevásárlói kosarakon keresztül történik. Az egy időben bekövetkezhető lehetséges eseményeket termékekkel modellezzük, az egyszerre bekövetkező események kerülnek egy bevásárlói kosárba (→ az egyes minták egy kosárnak felelnek meg). Egy termékhalmaz támogatottságán (support), a K kosárhalmazban mérhető relatív gyakoriságát értjük, ami közelítőleg a termékhalmaz az előfordulási valószínűségét méri. Egy termékhalmazt gyakorinak nevezünk, ha támogatottsága nagyobb, mint egy megadott minimális támogatottsági küszöb.

7.3.2 Asszociációs szabály

Egy K kosárhalmazba, T termékhalmazon $X \xrightarrow{c,\,s} Y - t$

c bizinyosságú és s támogatottságú asszociációs szabálynak nevezzük,

ha X, Y $\subseteq$ T és $X \cap Y = \emptyset$, ahol

$$C = \frac{\mathrm{supp}_K(X \cup Y)}{\mathrm{supp}_K(X)} \quad (\approx P(Y|X))$$

$$S = \mathrm{supp}_K(X \cup Y)$$

Egy asszociációs szabályt akkor tekintünk érvényesnek, ha támogatottsága nagyobb, mint egy magadott minimális támogatottsági küszöb, és bizonyossága nagyobb, mint egy megadott minimális bizonyossági küszöb. Adatbányászat során a feladat az érvényes asszociációs szabályok előállítása.

7.3.3 Érvényes asszociációs szabályok meghatározása

Érvényes asszociációs szabályok előállítása két lépésre bontható:

1. Gyakori termékhalmazok előállítása.
2. Érvényes asszociációs szabályok meghatározása a gyakori termékhalmazokból.

Mivel az első lépés sokkal nehezebb ezért az egyes asszociációs szabálykeresési algoritmusok főleg az első feladatra koncentrálnak. A gyakori elemhalmazok megtalálásának bonyolultságát az alábbi példa szemlélteti:

- Legyen a termékek halmaza mindössze 60 elemű
- Az egyes lehetséges termékhalmazokhoz rendeljünk egy 4 byte-os számlálót
- Olvassuk be a bevásárlói kosarakat és a kosarakban előforduló összes termékhalmaz kombinációt állítsuk elő.

- Mire végigolvastuk a bevásárlói kosarakat elő tudjuk állítani a gyakori termékhalmazokat.

A fenti egyszerű algoritmusnak a hátránya, hogy 2^{64} byte memóriát igényel. Igazából, ahogy a fenti példa is szemléltette, a gyakori termékhalmazok megtalálása NP teljes feladat.

7.3.3.1 Apriori feltétel, és algoritmus

Az asszociációs szabályok keresése során a nagyon gyakran használják a következő tételt (apriori feltétel): ha egy termékhalmaz gyakori, akkor annak összes részhalmaza is gyakori. A tétel következménye, egy gyakori termékhalmaznak nem lehet nem gyakori részhalmaza. Így a gyakori termékhalmazokat szintenként haladva fel lehet térképezni, mivel az apriori feltétel alapján erősen lehet vágni a keresési teret. A legelső, ilyen feltétel alapján dolgozó algoritmust apriori algoritmusnak nevezik, menete a következő:

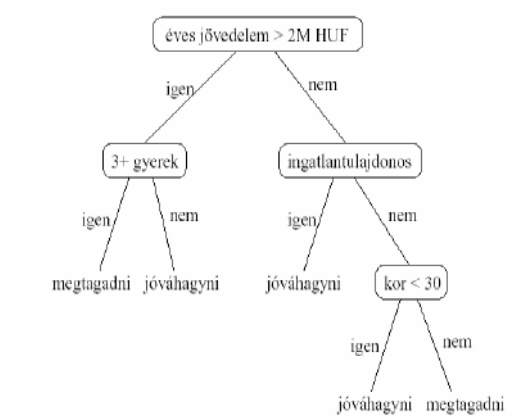
- Meghatározzák az i elemű gyakori termékhalmazokat
- Az i elemű gyakori termékhalmazokból i+1 elemű jelölt halmazokat generálnak. Egy halmaz akkor lesz jelölt, ha az összes részhalmaza gyakori.
- Az i+1-edik adatbázis olvasás során leszámolják az i+1 elemű jelölthalmazok előfordulását.
- A fenti lépéseket addig ismétlik, amíg találnak gyakori termékhalmazt.

Az apriori algoritmus előnye, hogy a keresési teret, és ezáltal a memória igényt, jól kézben tartja, viszont az adatbázist többször is végig kell olvasni, ami tekintettel a diszkműveletek sebességére, igen költséges. Az adatbázist többször végigolvasó, apriori algoritmusra épülő megoldásokat szintenként haladó algoritmusoknak nevezik. Az I/O költségek csökkentésére számos algoritmus született, viszont ezek memóriaigénye lényegesen nagyobb (és nem annyira kézben tartható), mint az apriori algoritmusé.

7.4 Osztályozás

Az osztályozás során az alapfeladat, hogy az egyes beérkező minta elemeket előre definiált osztályok valamelyikéhez rendeljék. Az egyes osztályok meghatározása mintaelemek (tanító halmaz) alapján történi. Minden egyes minta számos leíró tulajdonsággal rendelkezik, ezen leíró tulajdonságok alapján jellemezhetők az egyes osztályok. A tanítási fázist követően a beérkező új elem elhelyezése is az elemről összegyűjtött leíró tulajdonságok alapján történik. Számos, mesterséges intelligencia kutatások során kifejlesztett algoritmust, használnak osztályozásra, jelen jegyzetben, csak néhány gyakrabban használt módszer kerül ismertetésre.

7.4.1 Döntési fa

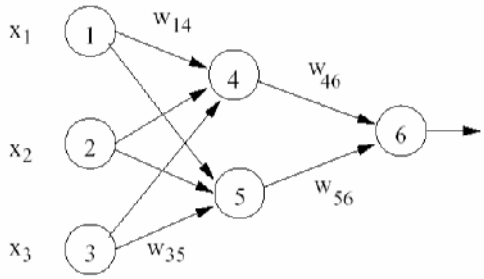


A döntési fa alapötlete, hogy összetett döntési folyamatot is egyszerű, egymás után meghozható döntések sorozatára bont fel. Egy ismeretlen minta osztályozásakor a fa gyökeréből kiindulva, minden egyes csomópontot a döntésnek megfelelő irányba haladva, egy levélcsomóponthoz jutunk, mely a döntésünket reprezentálja. Egy egyszerű hitelbírálatot mutat be a fenti döntési fa (a példa csak szemléltetés, semmi köze nincs a valósághoz).

A döntési fa előnye, hogy jól szétválasztja a lényeges és kevésbé fontos tulajdonságokat. Valamint az egyes döntéshez vezető utat is megmutatja, azaz ember számára is érthető, hogy egy adott döntés miért született. A döntési fa előállítása (tanítása) iteratív úton történik. Minden egyes szinten olyan kérdést teszünk fel, mely mentén az adott szinten megtalálható tanulóhalmaz elemek jól szétválaszthatók, azaz a keletkező új halmazokban az adatok homogenitása növekszik. A felosztás leáll ha:

- Az adott halmazba, minden mintaelem ugyan azon osztályhoz tartozik.
- Nincs több attribútum, mely alapján meg lehet különböztetni az elemeket. → Többségi szavazás határozza meg a döntést (azaz az adott szinten legtöbb mintaelemet tartalmazó osztály lesz a döntés).
- Döntési fa mélysége elér egy előre megadott korlátot, döntés többségi szavazás alapján születik.
- Nem tartozik a keletkezett csomóponthoz mintaelem.

7.4.2 Neurális hálók



A fenti példa egy előre csatolt neurális hálózatot mutat. A legegyszerűbb esetben az egyes neuronok az input adatuk súlyozott összegét állítják elő. A tanítási folyamat során az éleken elhelyezkedő súlyokat állítják be úgy, hogy a kimenet hibája a legkisebb legyen a tanuló adatbázison. A neurális hálók hátránya, hogy az általuk

hozott döntéseket nehéz megmagyarázni, mivel az egyes súlyoknak nincs önálló jelentésük.

7.4.3 K legközelebbi szomszéd elve

Ez az osztályozási technika nem épít előre modellt, hanem a mintaelemek távolságán (lásd klaszterezés) alapszik. Meghatározzák az új mintaelem távolságát a tanító mintaelemektől, majd veszik ezek közül a k legközelebbi szomszédot. A besorolásról a k legközelebbi szomszéd dönt többségi szavazással.

7.5 Klaszterezés

A klaszterezés során az alapfeladat maguknak az osztályoknak a megtalálása nagyméretű adathalmazban. Az irodalomban számos klaszterező algoritmus található, melynek oka, hogy jó klaszterező algoritmust nem lehet készíteni, csak részfeladatokra optimalizált megoldások vannak.

7.5.1 Elvárások a klaszterező algoritmusoktól

Egy jó klaszterező algoritmustól elvárhatók az alábbi alaptulajdonságok:

- Skála invariancia: Ha minden elempár távolsága helyett, annak α szorosát vesszük, ahol α>0, akkor a csoportosítás ne változzon
- Gazdagság (richness): Tetszőlegesen előre megadott csoportokhoz lehessen megadni úgy távolságokat, hogy a klaszterező eljárás az adott módon csoportosítson
- Konzisztencia: Ha egy csoportosítás után, a távolságokat úgy változtatjuk, hogy az egy csoportba tartozó elemek közt csökkentjük a távolságot, míg a különböző csoportba tartozók közt növeljük, akkor a csoportosítás eredménye ne változzon.

Sajnos nem lehet, olyan klaszterező eljárást készíteni, mely kielégíti a fenti három kritériumot, olyan viszont készíthető, mely tetszőleges két kritériumot kielégít.

7.5.2 Elemek távolsága

A klaszterezés során fontos, az elemek távolságának a definiálása, a távolság definiálása igazából feladat függő, viszont van néhány tulajdonság amit ki kell elégítenie a definíciónak. Igazából nem is az elemek távolságát szokták értelmezni, hanem az elemek eltérését, ami rokon fogalma a távolságnak:

- d(x,x)=0, egy elem önmagától való eltérése nulla
- d(x,y)=d(y,x), az eltérés szimmetrikus
- d(y,z)<=d(x,y)+d(x,z), azaz igaz rá a háromszög egyenlőtlenség.

Felmerül a kérdés,hogy jó e mindig a távolság alapú szemlélet, természetesen nem minden esetben vezet jó eredményre, erre jó példa az alábbi pontsor:

…….

…….

Szinte mindenki az egymás mellett elhelyezkedő pontokat tekintené egy klaszternek, viszont a pontok egymástól való távolságának minimalizálása során valamilyen

függőleges szeparáció adódna. Éppen ezért folynak kutatások más mérték alapján történő klaszterezésre (például sűrűség alapú eljárások).

7.5.3 K közép algoritmus

Az egyik legrégebben használt és legelterjedtebb algoritmus. Ez az algoritmus csak vektortereken értelmezhető, mivel szükséges az egyes klaszterekben a középpont definíciója. Az algoritmus négyzetes hibát próbál minimalizálni, azaz az egyes klasztereken belüli eltérések négyzetes hibáinak az összege minimális legyen. A hibát a klaszter középpontjához viszonyítja. A klaszterezés során előre definiálni kell a kialakítandó klaszterek számát (k). Az algoritmus menete a következő: A mintákból véletlenszerűen kiválaszt k pontot, ezek lesznek a kialakítandó klaszterek középpontjai. Az egyes mintaelemeket elhelyezi úgy, az egyes klaszterekben, hogy a négyzetes hiba minimális legyen. Ezt követően, a kialakult klaszternek meghatározzuk a tényleges középpontját, majd ezekhez az új referencia pontokkal végezzük el a klaszterezést. Az iterációt addig ismételjük amíg, történik mintaelem áthelyezés az egyes klaszterek között. Az algoritmus hátránya, hogy könnyen lokális minimumban ragadhat meg és nem találja meg a globális minimumát a négyzetes hibának. Ezért szokták többször egymásután lefuttatni különböző kiindulópontokból, így közelebb lehet jutni a globális minimumhoz. Az algoritmus másik hátránya, hogy csak eliptikus klasztereket tud generálni, mivel középponttal és sugárral dolgozik. A fenti hátrányok ellenére igen elterjedt vektorterek klaszterezésére, mivel egyszerű gyors és jó eredményeket szolgáltat jól elkülönülő klaszterek esetében.

8 Irodalomjegyzék

1. Fred R. McFadden, Jeffrey A. Hoffer and Mary B. Prescott: Modern Database Management
2. Jeffrey D. Ullman Jennifer Widom: Adatbázisrendszerek, alapvetés
3. Hector Garcia-Molina Jeffrey D. Ullman Jennifer Widom: Adatbázisrendszerek megvalósítása
4. Nyékyné G. Judit(szerk): Java 2 útikalauz programozóknak
5. Davis Sceppa: Programming ADO
6. Joyce Chen and Richard Patterson: Microsoft ADO and SQL Server Developer's Guide (MSDN Library)
7. ODBC Architecture (MSDN Library)
8. Rob Caron and Paul Larsen: Data Interoperability Using Enterprise-Wide Data Sources (MSDN Library)
9. Pieter Adriaans, Dolf Zantinge: Adatbányászat
10. Bodon Ferenc: Adatbányászati tanulmány (www.cs.bme.hu/~bodon)
11. Oracle Server 8 books online