NEUMANN JÁNOS
INFORMATIKAI KAR

SZAKDOLGOZAT

BMF-NIK
2007

KOMÁROMI
ZOLTÁN
NIK-O-EI-02-024

Budapesti Műszaki Főiskola
Neumann János Informatikai Kar
Szoftvertechnológia Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: **Komáromi Zoltán**
Törzskönyvi száma: NIK-O-NI-02-33

A dolgozat címe:

Adattárházak tervezése és implementációja Oracle OLAP DML nyelv segítségével

Főiskolai konzulens: **Nagy István**
Külső konzulens: Y

Beadási határidő: 2007. május 18.

A záróvizsga tárgyai: Architektúrák
Adatbázisok

A feladat:

A dolgozatnak tartalmaznia kell:

Ph.

……
intézetigazgató

A szakdolgozat elévülésének határideje: **2008. május 15.**
(BMF TVSz 26.§ 9. pont szerint)

A dolgozatot beadásra alkalmasnak tartom:

……..
külső konzulens

……
főiskolai konzulens

HALLGATÓI NYILATKOZAT

Alulírott szigorló hallgató kijelentem, hogy a szakdolgozat saját munkám eredménye. A felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Egyéb jelentősebb segítséget nem vettem igénybe.
Az elkészült szakdolgozatban talált eredményeket a főiskola, a feladatot kiíró intézmény saját céljaira térítés nélkül felhasználhatja.

Budapest, 2007. május 15.

.....
Hallgató aláírása

Tartalomjegyzék

1. Bevezetés

1.1 Célok

Az információs társadalomban az adat központi szerepet tölt be. A felhasználók óriási energiát fordítanak az adatok beszerzésébe, előállításába, majd az így gyűjtött robbanásszerűen növekvő mennyiségű adattömeg archiválására és tárolására. Emiatt már igen korán határozott és sürgető igény jelentkezett olyan technológiákra, melyek az értékes adattömegből hasznosítható információkat nyernek ki. Az adatok mennyiségének növekedésével párhuzamosan egyre jobb hatásfokú és szélesebb körű információkinyerési rendszerek alakultak ki, melyekkel nem csupán a múltbeli információk rendszerezése és lekérdezése biztosított, hanem bevonhatók a magasabb szintű stratégiai, döntéshozatali folyamatokban.

Mára az ügyviteli, a termelési, és a vállalkozások működése során keletkező egyéb adatokat tranzakciós adatbázisok tárolják, azonban ezek hatékonysága nem teszi lehetővé az információk eredményes elemzését, így nem alkalmasak a hosszabb távú tervezésre. Mivel OLTP nem támogatja kellőképp a stratégiai tervezést, megjelentek a kifejezettem e célra kialakított OLAP rendszerek. Szakdolgozatom célja, hogy egy mintapéldán keresztül bemutassam egy adattárház tervezését és felépítését Oracle OLAP DML nyelv segítségével.

1.2 Adattárházak kialakulásához vezető okok

Mivel az elektronikus információk beszerzése és gyűjtése az elektronikus dokumentumkezelés hatására vállalaton belül nem jár számottevő többletköltséggel, a tárolt adattömeg sokszorosára növekedett az évek folyamán. A felhalmozódott adatok kora akár 10-15 év is lehet, melyeket történeti adatoknak neveznek, s segítségükkel számos, a jövőre vonatkozó trend jelezhető előre. Az előrejelzés biztonsága annál nagyobb, minél nagyobb a múltbeli adattömeg, amelynek tárolása nem közvetlenül az operatív rendszerek mögötti adatbázisokban történik, hanem biztonsági és működési okokból általában archiválni szokták őket.

Az adattárházak ötlete nem mai eredetű, egészen a 80-as évek végéig nyúlik vissza, amikor Bill Inmon nevéhez kapcsolódóan elindult az adattárházak kialakulása, melynek fő okaként az OLAP rendszerek igényeihez igazított adatbázis kezelő rendszer hiánya tekinthető. Ugyanis az OLAP rendszereket a hagyományos, OLTP igényekhez szabott adatbázis kezelő rendszerekre alapozva is meg lehet valósítani, azonban az így létrejövő rendszerek hatékonysága, rugalmassága közel sem optimális, mivel a hagyományos adatbázis kezelők belső motorja az OLTP igényekhez optimalizált.

Hatékony OLAP rendszerekhez egy megfelelő, az OLAP igényekhez igazított adatbázis kezelő rendszerre volt szükség. Az igény már viszonylag régen jelentkezett, majd amikor a technikai feltételek is biztosítottak voltak, minden adott volt ahhoz, hogy ténylegesen is létrejöjjenek az OLAP igényekhez igazított adatbázis kezelő rendszerek, melyeknek a hagyományos adatbázis kezelőktől való megkülönböztetésként *adattárházaknak* (*DW*, data warehouse) nevezték el.

1.3 Adattárházak fogalma

Definíció

Az adattárház Bill Inmon által megfogalmazott definíciója a következő: "*A data warehouse is a subject oriented, integrated, nonvolatile, and time variant collection of data in support of management's decisions.*" - Az adattárház egy témaorientált, integrált, tartós és különböző időpontokra vonatkozó adatok gyűjteménye, mely támogatja a vállalati menedzsmentet a stratégiai döntéshozatalban.

OLTP – OLAP összehasonlítás

Az OLTP (On-Line Transaction Processing) rendszerek vállalatok üzletmenetének, üzleti tevékenységének támogatására jöttek létre azáltal, hogy homogén adatbázisokon, egy adatforrásra támaszkodva lehetővé teszik az adatok manipulációját, módosítását, frissítését. Bár a rendszerek a kevesebb adaton történő keresést támogatják, képesek az üzletmenet nyomon követésére és összesítések készítésére a napi eseményekről.

Azonban időközben az igények egyre jobban nőttek az OLTP rendszerek által nem biztosított elemző folyamatokra, ezért a probléma megoldására kidolgozták az OLAP rendszereket, melyek középpontjában az adatok analizálása és jelentések készítése áll. Az OLAP rendszerekkel az adatokat egy homogén egységként kezelve lehetővé vált a vállalati információk olyan szinten történő kezelése, amely segíti a döntéshozatalt.

Míg az OLTP rendszerek hagyományos adatbázis struktúrára támaszkodva korlátokat szabott a lekérdezések gyorsíthatóságának, ezáltal az alkalmazások teljesítményének, addig az OLAP rendszerek lehetővé tették a számos szempont szerinti csoportosítást, aggregációt. Ezek meglehetősen hosszú futási időt is eredményeztek, ezért az ilyen lekérdezéseket a szakemberek „drága” lekérdezéseknek is nevezik.

Az OLTP rendszerek jelenleg is igen elterjedtek, mivel az alkalmazások döntő többsége ilyen jellegű feladatokat lát el, s számukra a következő előnyöket biztosítják:

- Hatékony adatkezelés az OLTP rendszerek igen gyors belső végrehajtó motorjának köszönhetően, mely lehetővé teszi a nagyobb adatmennyiségekben való művelet végrehajtást elfogadható válaszidőn belül.
- Rugalmas tervezési és kezelési felület, mely megkönnyíti a változások nyomon követését és realizálását.
- A tranzakció kezeléstől megkövetelt integritási elvek betartatása a rendszerben biztosított.

A felsorolt előnyök mellett azonban megállapítható, hogy a felhasználóbarát kezelőfelülettel rendelkező információs rendszer megvalósításához a meglévő OLTP lehetőségek nem kínálják a leghatékonyabb megoldást. Az alkalmazási terület igényei, elvárásai alapján kifejlesztették az OLAP rendszereket, melyek az adatokat dimenziók mentén (idő, értékesítési csatorna, földrajzi elhelyezkedés, beszállítói csoportok, alkalmazott technológia stb.) több irányból teszik hozzáférhetővé, elemezhetővé és megjeleníthetővé.

Az OLAP rendszerek legfontosabb tulajdonságai a következők:

- Az adatkezelés adatlekérdezés orientált, az alkalmazások főként az adatértékek

lekérdezésének szándékával kapcsolódnak az adatbázishoz. A végrehajtandó lekérdezések egyik fő jellemzője azok elemzési jellege, ezáltal sokkal nagyobb adatmennyiséget is érintenek, mint az OLTP rendszerekben szokásos lekérdezési műveletek. A nagyobb adatmennyiséget megmozgató, komplexebb számítások miatt az utasítások időszükséglete magasabb (akár több órás is lehet), mint az OLTP rendszert esetén.

- Az adatbázis a kurrens mellett a múltbeli adatokat is tartalmazza.
- Kicsi az egyes alkalmazások közötti konkurencia, ugyanis egyidejűleg csak egy vagy néhány alkalmazás kapcsolódik az adatbázishoz. Ennek oka, hogy a menedzsment részére készült OLAP rendszereknél szűk a rendszerhez hozzáférési jogokkal rendelkező felhasználók köre, továbbá a lekérdezések nem a napi munka irányítására vonatkoznak.
- Az adatrendszer elsődleges lekérdezési funkciót szolgál ki, s ritka a módosítási művelet, ezért itt a legfontosabb feladatot nem az adatrendszer konzisztenciájának megőrzése jelenti, hanem a lekérdezési műveletek - a lehetőségekhez mérten - minél rövidebb idő alatti végrehajtása. Ehhez az OLTP rendszertől eltérő optimalizálási modulok, módszerek beépítésére van szükség az OLAP rendszert megvalósító rendszerekben.
- Több különböző inhomogén adatforrással dolgoznak, tehát a műveletbe bevont adatok több különböző adatbázis kezelőből kerülnek beolvasásra, így az egyes adatelemek különböző adatmodell szerint tárolva és különböző csomóponton foglalhatnak helyet.
- Az adatok belső tárolási formátuma még jobban rejtve marad a felhasználók előtt, így egy hatékony, rugalmas, felhasználói szemlélethez közel álló lekérdezői felülettel rendelkeznek.
- Az OLAP alkalmazások az adataikat más rendszerektől veszik át, így az OLAP rendszer nem maga az adatforrás, inkább egy adatintegráló modul.
- Nagyobb adatrendszerek ellenére is hatékony lekérdezések a hatékony válasz generálási algoritmusok eredményeképp.
- Egyszerűbb lekérdezések helyett összetett elemzési funkciókat megvalósító lekérdezéseket tartalmaznak, melyek a nagy adathalmazból összesítő, aggregált, szabályok és trendek feltárására alkalmas adatokat állítanak elő.
- Osztott adatforrással dolgozik, így egyidejűleg többen használhatják (azonban nem szabad figyelmen kívül hagyni, hogy hatékonysági megfontolásokból kiindulva egy OLAP adatforráshoz a hasonló jellegű alkalmazásokat célszerű hozzákapcsolni).
- Multidimenzionális adatmodellre épül.

Tulajdonságok	OLTP	OLAP
Orientáció	tranzakciók	adatanalízis
Felhasználó	vállalat adminisztrációt végző alkalmazottai	döntéshozók és őket információval támogató alkalmazottak
Feladat	napi folyamatok követése	döntéstámogatás, hosszútávú információgyűjtés és szolgáltatás
Adatbázis tervezése	Egyed-Kapcsolat modell, alkalmazás orientált	tárgy-orientált, csillagséma
Adatok	aktuális, up-to-date	történeti adatok, időben archiválva
Aggregált adatok	nem jellemző; részletes felbontás	felösszegzett, egyesített adatok
Adatok nézete	részletezett, relációs	felösszegzett, multidimenzionális
Felhasználók hozzáférése	olvasás/írás	legtöbbször olvasás, adattárház adatait nem módosítják
Hangsúly	adatbevitelen	információkinyerésen
Feldolgozandó rekordszám	tízes nagyságrendű rekord alkalmanként	akár milliós rekordszám
Felhasználók száma	viszonylag sok	kevés, közép és felsővezetők ált.
Prioritás	állandó rendelkezésre állás és megbízhatóság	rugalmasság, felhasználói önállóság

1. táblázat: OLTP és OLAP rendszerek összehasonlítása
(forrás: Sidló Csaba: Összefoglaló az adattárházak témaköréről)

Codd 12 kritériuma

Az OLAP rendszerek legfontosabb jellemzőinek összefoglalására az OLAP technológia atyja Dr. E. F. Codd, 1993-ban definiált egy 12 pontból álló feltételrendszert, melynek célja annak meghatározása, hogy mikor tekinthető egy rendszer OLAP rendszernek. A kritériumok a következők:

1. **Többdimenziós nézet:** az adatok modellje többdimenziós és több változós így lehetővé válik a felhasználók számára a fent felsorolt tevékenységek végrehajtása.
1. **Transzparens támogatás:** a felhasználónak ne kelljen ismernie, hogy az adatok milyen módon kerülnek tárolásra
2. **Elérhetőség:** az OLAP eszközt, mint közvetítő motort határozta meg a tároló egység és a frontend eszköz között.
3. **Állandó teljesítmény:** a dimenziók száma és az adatok mennyisége ne befolyásolja a felhasználó által érezhető teljesítményt.
4. **Kliens-Szerver** architektúra
5. **Általános dimenzió fogalom**: a dimenzióknak struktúrájában azonosnak kell lennie.
6. **Dinamikus ritkamátrix-kezelés**: biztosítani kell a ritka mátrixok optimális feldolgozási teljesítményét
7. **Több konkurens felhasználó támogatása**: biztosítani kell egyszerre egy idő-

ben több felhasználó hozzáférését ugyanazon modellhez és az ezzel kapcsolatos biztonsági kérdéseket.

8. **Korlátozás nélküli dimenzió műveletek**: a dimenziók közötti műveletek összes formáját meg lehessen valósítani.
9. **Intuitív adatkezelés**: az adatok manipulálása a felhasználók számára közvetlenül a megjelenítési felületen megtörténhet
10. **Rugalmas jelentések**: adat megjelenítés különböző módjait támogassa.
11. **Korlátlan dimenziók**: Codd szerint a rendszernek képesnek kell lennie kezelni akár 15-20 dimenziót is.

Codd kritériumai iránymutatóként alkalmazhatók az OLAP rendszerek azonosításában, azonban a felsorolt kritériumok jelentőségét csökkenti, hogy igen sok közöttük a termék specifikus tulajdonság, mely a gyakorlatban nem vált kizárólagossá.

Adattárház – adatpiac

A centralizáltság foka alapján megkülönböztethetők adattárházak és adatpiacok (Data Mart-ok). A centralizált, amely alatt az egységes adattárházakat, azaz egy adott cég minden igényét kielégítő megoldásokat értjük; és a decentralizált, amely a különböző specifikus területekkel foglalkozó adatpiacokat jelenti. Az adattárházak megfelelően strukturált, ill. elemzéshez előkészített vállalati adatbázisok fejlett, feltöltést, karbantartást és lekérdezést végző eszközökkel kiegészítve, amelyek a vállalat teljes egészét átölelik. Az adatpiacok ezzel szemben egy konkrét téma köré épülnek, bizonyos funkcionális vállalati területekre koncentráló információgyűjtemények. Az adatpiacok összevonásával kialakítható az egységes vállalati adattárház.

A centralizált adattárház és az adatpiac közti legfőbb különbség nem a méretükből, hanem a hatáskörükből, a megcélzott témakör nagyságából, összetettségéből adódik. Ez a hatásköri különbség eltérő felépítettséget is feltételez. Mindazonáltal az adatpiacnak nagyfokú skálázhatósággal kell bírnia, mivel mérete igen nagyra növekedhet.

Az adattárházak felhasználói lehetnek a vállalat bármely alkalmazottai, közép- és felsővezetői és mindenki számára hasznos adatokat tartalmazz. Az adatpiacokat viszont egy szervezeti egység, osztály, részleg részére hozzák létre, gyakran csak egy-két témaköré csoportosítva. Túlnyomó többségében ezek a részlegek alkalmazottai és vezetői lesznek a végfelhasználói is.

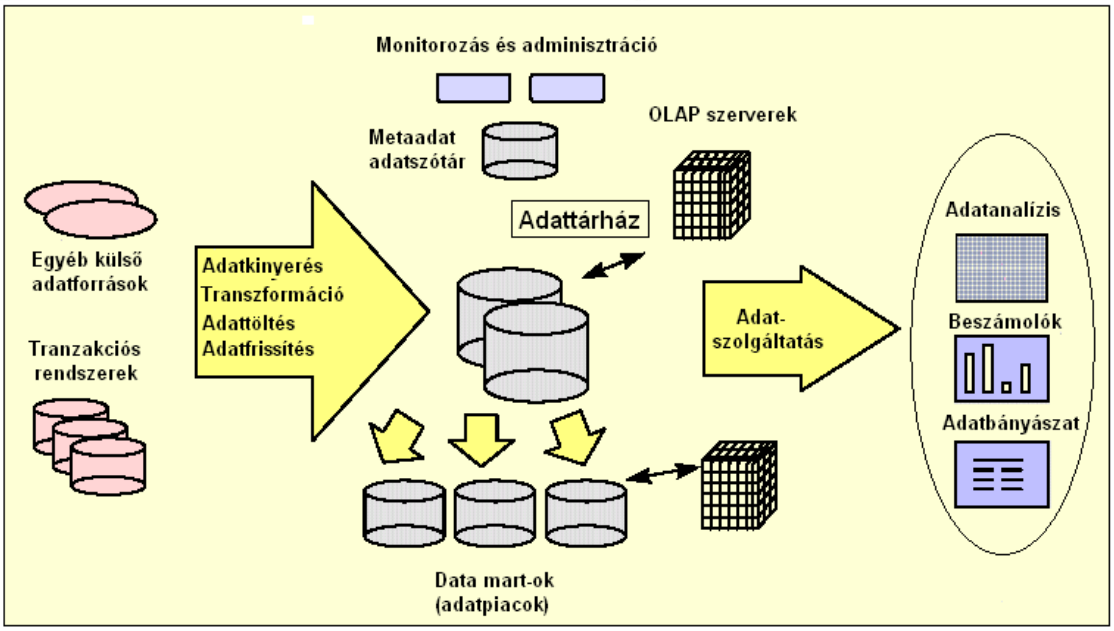
Adattárházak jellemzői:

- alapvető célja a tárolt információk elemzése,
- adatait a tranzakciós „forrás” rendszerektől elkülönítve tárolja,
- adatbázis struktúrája lekérdezések szempontjából optimalizált,
- vállalati szintű átfogó, integrált adatokat tartalmaz,
- tartalma nem változik, csak bővül (nem állapotot, hanem történéseket regisztrál)…
- megvásárlásukkor üresek, a vállalat tölti fel üzleti célok alapján megfelelő adattisztítás után

Az adattárházak fizikai felépítésében három fő szereplőt célszerű megkülönböztetni:

- OLTP adatforrások
- Maga a DW rendszer
- Felhasználói, kliens alkalmazások, segédeszközök

mely komponensek fizikai topológiája alapján különböző csoportok különíthetők el.



1. ábra: Szokványos adattárház komponensek
(forrás: Sidló Csaba: Összefoglaló az adattárházak témaköréről)

Az adatpiacok és adattárházak kiépítése nagyon hasonló feladat, a köztük levő különbség a következőkben fogható meg:

Tulajdonságok	**Data Warehouse**	**Data Mart**
Hatáskör	Vállalat	Részleg, osztály
Téma	Több témát felölel	Egy témára koncentrál
Adatforrás	Sok	Néhány
Tipikus méret	100 GB-tól akár 1 TB fölé	100GB alatti
Implementációs idő	Hónapok, akár évek	hónapok

2. táblázat: Adattárházak és adatpiacok összehasonlítása
(forrás: Adattárház készítés alapjai I.)

Adatpiacok fajtái:

Az adatpiacok két csoportra oszthatók aszerint, hogy mely adathalmazból táplálkozik az adatpiac. Így megkülönböztetünk:

- Függő adatpiacot (Dependent data mart)
- Független adatpiacot (Independent data mart)

Függő adatpiac: az adattárházból származtatjuk a funkcionális adatpiacokat.

Független adatpiac: a független adatpiac, egy elkülönült önálló rendszer, amelyet közvetlenül az operatív adatokból és az esetleges külső forrásokból építenek fel. Majd igény szerint ezekből alakítják ki az adattárházat.

A két csoport összehasonlítását az alábbi táblázat tartalmazza:

Függő adatpiacok	Tulajdonságok	Független adatpiacok
adattárház	Forrás	operatív rendszerek és külső források
egyszerű	ETL folyamat	bonyolult
a teljesítmény növelése, rendelkezésre állás javítása	Telepítés célja	elemzési igények kielégítésére
konzisztens	Adatforrás	tisztítatlan vagy inkonzisztens

3. táblázat: Adatpiacok összehasonlítása

Az adattárházak, és adatpiacok között szoros kapcsolat van, ugyanis az adattárház egy másik megközelítésben a következőképpen is megfogalmazható: „Az adattárház az adatpiacok összességeként kialakuló rendszer”, vagyis ebből a megfogalmazásból kiderül, hogy az adatpiac egy vállalaton belül, ha van adattárház, akkor annak része. Ez a megfogalmazás továbbá előre vetíti azt is, hogy az adattárházak egy vállalaton belül kiépíthetők a már meglévő adatpiacokból is, továbbá arra is utal, hogy az adatpiacok a már meglévő vállalati adattárházból is kialakíthatók.

2. Adattárházak tervezése

2.1 Általános tervezési szempontok

Az adattárházak, mint az az előző fejezetből is kiderült, alapvetően lekérdezés és elemzés orientáltak, ennek megfelelően tervesési szempontjaik jelentősen eltérnek a tranzakciós rendszerekétől. Míg azoknál a cél egy lehető legkevésbé redundáns struktúra kialakítása, addig az analitikus rendszereknél ez kifejezetten hátrányos, mivel ilyen esetekben a lekérdezések egyszerűsítése a cél. Például, ha egy dátum alapján szeretnénk egy lekérdezést készíteni, és azt összesíteni hónap, negyedév és év alapján, akkor a csoportosításnál három számításra lesz szükségünk, míg ha már ezeket az értékeket is tároljuk, akkor csak három mezőhivatkozásra. Ez azonban amiatt, hogy az adattárházba csak ritkán kerülnek adatok, és azok is előre megtervezett, tesztelt gépi úton, a tranzakciós rendszereknél, ahol az adatbevitel, és módosítás a fő feladat, ilyen esetekben megjelenő anomáliák fellépésére nem kell reálisan számítani.

Ennek megfelelően míg az OLTP rendszerek tervezésekor az ősmodellből kiindulva a normalizálás viszonylag szabványos lépéseit követve eljutunk egy harmadik, sőt esetenként Boyce-Codd normálformában lévő adatbázisig, addig az OLAP rendszer tervezésekor a denormalizálás során visszalépünk a második, sőt időnként az első normálformás állapothoz. Az a tervezési folyamat, mely során az adattárház létrejön, erősen függ mind a kiindulási rendszer struktúrájától és adattartalmától, mind pedig azoktól a kérdésektől amelyekre az elkészült OLAP rendszerből szeretnénk választ kapni. Fokozott problémákat jelent, hogy az adattárház általában heterogén rendszerekből nyeri az adatait, melyek között van egy vagy több OLTP rendszer, de lehetnek szövegfájlok és speciális adatforrások is, pl. internet. Ennek megfelelően a tervezési folyamat adattárházak esetében nem általánosítható, csak a tervezés egyes szakaszaira, és az egyes szakaszokban elvégzendő részfeladatok célkitűzéseire van általánosan elfogadott leírás.

Adattárházak esetében strukturális döntésekre is szükségünk van a tervezés elején. A döntés lényege, hogy a létrejövő OLAP rendszer relációs alapú (ROLAP), multidimenzionális (MOLAP), vagy a kettő kombinációja ként hibrid (HOLAP) lesz-e. Hibrid rendszerre természetesen akkor van szükségünk, ha mind a relációs, mind a multidimenzionális rendszer előnyeit szeretnénk kihasználni. A ROLAP rendszerekből könnyen készíthetők jelentések, de például adatbányászat, előrejelzés készítésére a MOLAP rendszerek alkalmasabbak. A MOLAP rendszereknek viszont pazarlóak, hiszen azoknak a celláknak is foglalnak helyet, amelyek valójában adatot nem tartalmaznak. Emiatt és azért, mert a számításokat ezeken az adatokon is elvégzi kevesebb adattal képes dolgozni.

	ROLAP	MOLAP
Fő kimenet	Jelentés	Analízis
Adatok	Közepes részletezettség	Összesített
Előny	Sok adat kezelésére képes Az eredeti relációs adatbázis funkcióit használhatja	Nagy teljesítmény Komplex számítások (pl. előrejelzések) végzésére is lehetőséget biztosít Nem korlátozza az SQL technológia
Hátrány	Kisebb a teljesítménye Az SQL lehetőségei limitálják az elérhető funkcionalitást	Kevesebb adattal dolgozhat Extra befektetést igényel, egyrészt a kocka technológia, másrészt az egyedi nyelv miatt

4. táblázat: ROLAP és MOLAP összehasonlítása

2.2 ETL folyamat

Az ETL a fent említett egységes tervezési „leírás”, amely összefoglalja egyrészt a tervezés szakaszait és azokat az elveket, melyeket a tervezés során célszerű követni. Másrészt a korábbi tervezések és megvalósítások során fellépő általános problémák leírását is tartalmazza. Neve egy betűszó, mely az egyes szakaszok nevének kezdőbetűiből áll össze:

- **E**xtract (adatkiválasztás)
- **T**ransforming (transzformáció)
- **L**oading (adatbetöltés)

Az ETL folyamatban nincs tényleges adatmozgatás, az első két lépésben (Extract, Transforming) csak a metaadatok meghatározása történik, míg a harmadik lépésben határozzuk meg azt, hogy mely forrásadat milyen módon és honnan kerüljön be az adattárházba.

A tervezés minősége alapvetően meghatározza a létrejövő adattárház minőségét is egyben. Amennyiben itt hibát követünk el, és az csak az elemzések során derül ki, akkor rossz esetben a teljes addigi munkát elölről kell kezdeni. Ezért a tervezési szakasz a teljes adattárház létrehozási folyamat legidőigényesebb része, és indokolja, hogy a „Big bang” módszer helyett az inkrementális fejlesztést válasszuk.

2.2.1 Extract (adatkiválasztás)

A kiválasztás szakaszában kell dönteni arról, hogy a rendelkezésre álló adatok közül melyek azok, amelyeket az adattárházban használni szeretnénk, és az esetlegesen hiányzó adatokat be tudjuk-e szerezni, és ha igen mely forrásból tesszük ezt meg, valamint az adatok tisztításáról is. Ez a lépés a kiindulási adatok és adatszerkezetek valamint az elérendő célok mély ismeretét igényli, s igen kis részben automatizálható.

2.2.1.1 A tranzakciós adatbázisokban fellépő problémák

Kitöltetlen és szemetet tartalmazó mezők

Az adatok rögzítésekor következik be, általában felhasználói tevékenység eredménye. Például szöveges mezők esetén a rögzítéskor a tényleges adat bevitele helyett csak tartalom nélküli szöveget visznek be. A kitöltetlenség viszont tervezési hiba eredménye, például hiányzó NOT NULL megszorítás.

El kell dönteni, hogy ezeket az adatokat miként kezeljük. Megpróbálhatjuk „kitalálni", hogy milyen adatnak kellene ott szerepelnie. Ilyenkor viszont ennek a módszernek algoritmizálhatónak kell lennie. A másik lehetőségünk, hogy ezeket az adatokat kihagyjuk az elemzésből. Esetenként dönthetünk ezen adatok változatlanul hagyásáról, esetleg külön kategóriába való sorolásáról is.

Szöveges mezőkben a szabványok hiánya

Szöveges adatok, például címek esetében általában még egy részlegen belül sincsenek szabványok, vagy ha léteznek is, betartatásuk igencsak nehézkes.

Ki kell dolgozni egy módszert, mellyel ezekből az adatokból kinyerhetők az információk. Valamint mindent meg kell tenni, hogy a jövőben keletkező adatok már szabványosak legyenek.

Adatbeviteli hibák

Ide értjük mindazokat a beviteli hibákat, melyek nem szándékosak, de „adatszemetet" hoznak létre. Például az elütések tartozhatnak ide.

Ezeket igen nehéz kivédeni, viszont szerencsére hasonlósági függvények segítségével viszonylag jól kezelhetőek.

Többszörös adatok

Előfordulhat, hogy bizonyos adatok, bármilyen gondos is volt a tervezés, többször is szerepelnek a forrásadatok között. Például egy ügyfelet véletlenül többször is felvisznek, illetve az adatbeviteli hibák javításakor is kiderülhet, hogy a javítás többszörös adathoz vezet.

Ezeket az adatokat ilyenkor egyesíteni kell. És vagy már az adatforrásokban is gondoskodni kell az egyesítésről, vagy pedig az egyesítést algoritmizálni kell.

Tervezési hibák

Egy részük a már korábban említett hibákat okozza, más részük viszont nem javítható adathiányokhoz vezet. Szerencsére az adattárházak tervezésekor már igen ritkán fordulnak elő, mert a tranzakciós adatbázisokban is anomáliákat okoznak és javításra kerülnek.

Szélsőséges, torzító, zajos adatok

A forrásadatokban lehetnek olyanok, amelyek nem illenek a nagy átlagba. Ezek nem hibásak, csak esetleg nem tekinthetőek a mindennapi működés eredményének. Például ha valamely ügyfélnek egy vagy több termékből hirtelen nagy mennyiségre van szüksége, és kereskedőnél lévő teljes készletet felvásárolja. Ha ez egyedi eset, akkor ez az eset hibás stratégiai döntésekhez vezethet.

Zajos adatok esetében a simítási technikák lehetnek a segítségünkre. Ilyen esetben végezhetünk lokális, csak a zajos adatelemekre és azok közvetlen környezetére vonatkozó, és globális, minden adatelemre vonatkozó simítást is.

2.2.1.2 Adattisztítás feladatai

Az adattisztítás egy része a fenti problémák kezelését jelenti, hogy azok nem kívánt hatásai az adattárházban már ne jelenjenek meg. Ez a lépés nem a tényleges adatok módosítását jelenti, hanem csak a kezelésük módját határozzuk meg, és szükség esetén a tisztítási metodikát dolgozzuk ki.

Az adattisztítás másik része a forrásadatok az adattárház szempontjából releváns és irreleváns részének meghatározása.

2.2.2 Transforming (adattranszformáció)

Az adattranszformáció során azt kell meghatároznunk, hogy a releváns forrásadatok mely adattárházi struktúrákba kerülnek. Ahhoz, hogy ezeket megtehessük, elengedhetetlen az adatkiválasztás elvégzése, az adattranszformáció lépései nem végezhetőek el az előző szakaszban meghozott döntések nélkül.

Dimenziók meghatározása

Meg kell határoznunk, hogy mely adatokból lesznek dimenziók, magyarázó adatok. A dimenziók száma, bár nem korlátozott, de multidimenzionális struktúra esetében a szükséges tárhelyet jelentősen befolyásolja. Egy új dimenzió az adatkocka méretét a dimenzió elemszám-szorosára növeli. Ezért van az, hogy a MOLAP kockák nem szoktak 6-7 dimenziónál többet tartalmazni. Például van egy kockánk, amely három dimenziós (dimenziónként 10, 12 és 17 elemmel, így 2040 cellával), és hozzá veszünk egy új dimenziót 15 elemmel, akkor a cellák száma 15-szörösére (30600) növekszik. Sajnos a dimenziók gyakran inkább 100-as nagyságrendbe tartoznak.

Hierarchiák kialakítása

Egyes dimenziók esetében különféle csoportosításokat, hierarchiákat szeretnénk. Jellemző példa sz idő dimenzió hierarchiája, mely minden adattárházban megtalálható. Más dimenziók esetében is szükség lehet a kategóriákba sorolásra.

Új értéktartományok meghatározása, normalizálás

A forrásadatok attribútumainak egy része széles értéktartományt foglal el, az adattárházban viszont már a konkrét értékekre nincs szükségünk. Ilyenkor az értéktartomány csökkentésével, és átskálázással az adott értékek egymáshoz közelebb kerülnek amellett, hogy az információ nem vész el.

Adatok általánosítása

Általánosítás során a kezdeti primitív, „nyers” adatokat fogalmi hierarchiák alapján magas szintű fogalmakra cseréljük. Ilyen általánosítást szokás végezni például a hely és életkor esetében, ahol a kezdeti adatokat (pontos cím, életkor-év) olyan fogalmakra cseréljük, mint ország-kontinens, öreg-fiatal.

Adatok konstrukciója

Azt a metódust értjük adatok konstrukcióján, amikor már meglévő attribútumokból újakat hozunk létre. Jellemző művelete nominális adatok esetében a szorzás, míg bináris adatok esetén a logikai és művelet. Például az eladott mennyiség és az egységár konstrukciójával létrehozhatjuk az érték attribútumot.

Simítás

Simítás során az adatokból eltávolítjuk a zajt, ehhez a kosarazási és klaszterezési technikákat használhatjuk. Kosarazáskor az zajos adatokat kosarakba osztjuk, és az adott kosárban lévő adatokat statisztikai módszerekkel a kosár alapján számított értékekkel helyettesítjük. A klaszterezés lényege, hogy a hasonló, egymáshoz közeli adatokat klaszterekbe gyűjtjük, és az egyetlen klaszterbe sem tartozó adatokat tekintjük szélsőséges értéknek.

Összevonás

Összevonás alatt az adatok aggregációját értük. Ez a művelet egyrészt az adattárház feltöltése során jelentkezik, hiszen az egyes dimenziók felbontása[1] gyakran nem egyezik meg a forrásadatok felbontásával. Másrészt összevonást kell végezni az esetlegesen definiált hierarchia magasabb szintjein is, ezt már általában az adattárházon belül kell elvégezni.

2.2.3 Loading (adatbetöltés)

Az adatbetöltés a forrásadatok adattárházba mozgatásának tervezését jelenti. Miután megterveztük az adatkiválasztást és -transzformációt, még meg kell határoznunk, hogy az adatok milyen úton kerülnek az adattárházba. A tervezés során nem csak a kezdeti adatfeltöltésre kell gondolnunk, hanem a későbbi frissítésekre is. Itt célszerű azt is meghatározni, hogy a frissítések inkrementálisak lesznek-e, vagy teljes újratöltést végzünk-e ilyenkor. Az első lépés több kezdeti erőforrást igényel, hiszen meg kell terveznünk azt is, miként döntjük el egy forrásadatról, hogy már bekerült-e az adattárházba, vagy pedig még nem, míg teljes újratöltés esetén erre nincs szükség.

A betöltést bonyolítja, hogy a forrásadatok nem egy, homogén rendszerből származnak, hanem több különbözőből. Ezek lehetnek egy- vagy többféle implementációt használó relációs adatbázisok, szövegfájlok, objektum orientált rendszerek, vagy például egy internetes forrás.

Az egyszerűsítés kedvéért tekintsünk egy olyan helyzetet, amikor a tisztított forrásadatokat egy Oracle adatbázisban szeretnénk tárolni, mielőtt azokat áttöltenénk közvetlenül az adattárházba.

Ebben az esetben a használni kívánt adatforrások függvényében dönthetünk a betöltési technikákról. Általánosságban elmondható, hogy importálási feladatokra mindig írhatunk speciális alkalmazásokat, melyek az igényeinknek megfelelő betöltést elvégzik. Egy-egy ilyen alkalmazás kifejlesztése viszont rendkívül sok erőforrást igényelhet.

Az Oracle, s más gyártók is, ha az ő megoldásukat választjuk, biztosít számunkra eszközöket, melyek az ilyen feladatok megoldásában segítségünkre lehetnek. A következők-

1 A dimenzió felbontásán értjük azt a legkisebb adategységet, melyet a dimenzió tartalmazhat. Amennyiben meghatároztunk hierarchiát, akkor a hierarchia legalsó szintjén helyezkednek el.

ben áttekintjük a rendelkezésre álló Oracle megoldásokat.

DB-Link

Akkor lehet hasznunkra, ha a forrásadat más szervereken lévő Oracle adatbázisokban van, valamint a két szerver között létrehozható TCP/IP kapcsolat. Ebben az esetben, ha definiáljuk az adatbázis kapcsolatot, akkor hivatkozhatunk a távoli adatbázisban lévő táblákra. Ehhez nem kell mást tennünk, csak a tábla nevét megfelelően minősíteni:

```
<schema>.<table or view>@<db-link>
```

Az adatbázisszerver ilyenkor igyekszik optimalizálni az átvitelt a két szerver között, de sokat javíthatunk a teljesítményen, ha követünk néhány szabályt. Egy lekérdezés önállóan tudjon futni egy adatbázisban, tehát ne kérdezzen le adatokat két különböző adatbázisból is egyszerre. Ilyenkor ugyanis nem optimalizálható a lekérdezés és túlzott adatforgalom generálódhat. Ha a lekérdezésnek mégis több adatbázist kell érintenie, akkor célszerű lehetőségekhez képest homogén allekérdezéseket használni, és azokat a DRIVING_SITE parancsmegjegyzés segítségével egy adott adatbázisban való végrehajtásra utasítani.

Oracle SQL Developer Migration Workbench

Microsoft SQL Server adatbázisból, Microsoft Accessből és MySQL adattáblákból teszi lehetővé az adatimportot. Java JDBC kapcsolaton keresztül szólítja meg a forrásadatokat. Ezért elvben lehetőség van más adatforrások megszólítására is.

Oracle Migration Workbench

Sybase, Indormix és DB2 adatbázisokból biztosít adatimportot. Teljes adatbázis sémákat importál, beleértve a triggereket és tárolt eljárásokat is.

External table

SQL DDL utasításon keresztül van lehetőségünk szövegfájlok használatára Oracle táblaként. Ha a szövegfájlt külső táblaként csatoltuk, akkor lehetőségünk van a tartalmának módosítására is.

SQLLoader

Szöveges adatok importját teszi lehetővé, hasonlóan a külső adattáblákhoz, azonban az adatimport célobjektuma egy reguláris tábla, de előnye azzal szemben, hogy azonos formátumú, frissebb adatokat tartalmazó szövegfájlok betöltése teljesen automatizálható. Így például e-mail útján érkező adatfrissítések importja válik rendkívül kényelmessé. A paraméterekben megadhatóak SQL jellegű szűrési feltételek, melyek az adatok kiválasztását, előzetes tisztítását is lehetővé teszik. Mindemellett a szövegfájlok betöltését igen hatékonyan végzi el.

3. Oracle OLAP DML

A MOLAP struktúrát implementáló adattárházak, mint azt már korábban említettem, jelentős erőforrás befektetést igényelnek. Létre kell hozni a többdimenziós adatbázis objektumait, és ezeket módosítani, lekérdezni is tudni kell.

3.1 A Oracle OLAP DML jellemzői

Az Oracle OLAP DML az a nyelv, mely lehetővé teszi az objektumok létrehozását és manipulálását az analitikus munkaterületen - beleértve magát a munkaterületet is -, továbbá tartalmaz beépített utasításokat, függvényeket, valamint a programíráshoz szükséges nyelvi elemeket is.

Az analitikus munkaterületek az adatbázison belül, speciálisan definiált BLOB mezőkben tárolódnak. Ennek megfelelően az Oracle OLAP használata igényeli egy relációs Oracle adatbázis meglétét. Ezt az adatbázist viszont, a megfelelő teljesítmény érdekében, teljesen másként kell konfigurálni, mint ha OLTP adatbázist hoznánk létre. Például a munkaterületeket tartalmazó táblák szegmenseit megfelelő méretűnek kell definiálni.

Objektum	Cél
Dimension	Dimenziók
Variable	Tényadatok
Relation	Adatkapcsolatok (hierarchia, csoportosítás)
Surrogate	Dimenzióhoz kapcsolt kiegészítő információ
Composite	Dimenzió-érték kombinációk listája
Valueset	Dimenzió értékek halmaza (aldimenzió)
Program	OLAP DML utasítások sorozata
Aggmap	Összegzési szabályok
Formula	Tárolt kalkulációk, kifejezések
Model	Adatkalkuláció halmaz

5. táblázat: Oracle OLAP DML alapobjektumok

Az objektumok neveinél be kell tartanunk a más programnyelvekben is meglévő névadási szabályokat. A névnek karakterrel vagy ponttal kell kezdődnie, nem tartalmazhat vezérlő karaktert, szóközt valamint műveleti jelet. Az ékezetes karakterek használatával is óvatosan kell bánni, mivel a programok azokat nem mindig kezelik helyesen. A nevek case insensitivek, tehát a nagy- és kisbetűk különbségére érzéketlenek, továbbá nem lehet két azonos nevű objektum, függetlenül a típusuktól.

3.2 Adattípusok

A multidimenzinális objektumok esetén, hasonlóan a relációs táblák attribútumaihoz, különböző típusokat használhatunk.

3.2.1 Numerikus adatok

Egész típus

Létezik belőle SHORTINTEGER (16 bit), INTEGER (16 bit) és LONGINTEGER (32 bit) változat.

Lebegőpontos típus

Három változata létezik, attól függően, hogy hány értékes jegyet szeretnénk tárolni. Ennek megfelelően van 7 (SHORTDECIMAL), 15 (DECIMAL) és 38 (NUMBER) értékes jegyet tartalmazó típus is. Ezek közül a NUMBER az alapértelmezett. Ha egy változónak nem adunk meg típust, akkor automatikusan ilyen típusú lesz.

3.2.2 Szöveges típusok

TEXT

Szöveges adattípus, mely maximum 4000 bájt UTF-8 kódolású szöveget tartalmazhat.

NTEXT

Szintén 4000 bájtnyi szöveget tartalmazhat, de az adatbázis karakterkészletében kódolva.

ID

Nyolc karakter hosszú azonosító szöveg tárolására alkalmas.

3.2.3 Egyéb típusok

DATE

Dátum típus, mely isz. 1000 január 01. és isz 9999. december 31. közötti dátumot tartalmazhat, idő nélkül

DATETIME

Dátum típus, mely ie. 4712. január 01. és isz. 9999. december 31. közötti időpontokat tárolhat.

BOOLEAN

Logikai érték tárolására. Lehetséges értékei: Yes – No; True – False; On - Off

3.3 Az Oracle OLAP alap utasításai

Az OLAP DML utasítások egy része a különféle objektumműveletekre szolgál. Segítségükkel új objektumokat hozhatunk létre, illetve már létezőket módosíthatunk, esetleg szükség esetén törölhetjük azokat.

3.3.1 Munkaterület manipuláló utasítások

AW CREATE

Ezzel az utasítással hozhatunk létre új munkaterületet. Paraméterei között megadhatjuk a létrehozandó partíciók számát, mely az adatbiztonságot hivatott növelni, hiszen egy esetleges sérülés esetén csak az adatok egy része lesz érintett. Megadhatjuk továbbá a szegmentumok méretét is, ami a lefoglalt terület növekedésének mértékét befolyásolja.

Példák:

```
AW CREATE demo_aw
AW CREATE test_aw SEGMENTSIZE 200M
```

AW ATTACH|DETACH

Munkaterületek fel- és lecsatlakozása érhető el segítségükkel. Lecsatlakoztatás esetén figyelni kell arra, hogy minden olyan módosítás, mely legutolsó UPDATE és COMMIT óta történt, törlésre kerül, ilyenkor ugyanis végrehajtódik egy adatbázisszintű rollback. Felcsatlakoztatás esetén több mód közül választhatunk:

- **RO** : Csak olvasható mód. A változások nem menthetők el. Ez az alapértelmezett.
- **RW** : A munkaterület módosításai elmenthetők
- **RWX** : Az RW módnak megfelelő, de kizárólagos írási jogot biztosít.
- **MULTI** : Az RW módnak egy kifejezetten multisessionos módosítást biztosító módja.

Csatlakozhatunk egyszerre több munkaterülethez is. Ilyenkor megadható, hogy az éppen felcsatlakoztatni kíván munkaterület mely munkaterület elé vagy mögé kerüljön, esetleg a legelső, vagy legutolsó pozícióba.

Példák:

```
AW ATTACH demo_aw RO
AW ATTACH test_aw RWX FIRST
AW DETACH demo_aw
```

UPDATE

A munkaterületek működésének jellemzője, hogy mindig egy átmeneti tárterületet használnak az utasítások kiadásakor, az objektumok, és az adatok ezen a tárterületen kerülnek elmentésre. Akkor, amikor kiadjuk az UPDATE parancsot, akkor íródnak be a munkaterületnek megfelelő BLOB mezőkbe. Mivel az adattárházban a fő művelet a lekérdezés, ez a metódus biztosítja, hogy az egyes elemzésekhez használt ideiglenes objektumok ne okozzanak ütközést.

COMMIT

A relációs adatbázisok COMMIT utasításának felel meg. Véglegesíti a megelőző COMMIT óta kiadott UPDATE utasítások által beírt adatokat. Fontos, hogy csak azokat, amelyeket már egy UPDATE utasítás beírt, tehát ha például a legutolsó UPDATE óta létrejött egy új objektum, az a COMMIT parancs hatására nem mentődik el.

3.3.2 Objektumokat kezelő utasítások

DEFINE

Ezzel a paranccsal definiálhatóak a multidiemziós adatbázis objektumai. Általános alakja:

```
DEFINE {objektumnév} {objektumtípus} {karakterisztika}
```

Az objektumtípus határozza meg azt, hogy milyen objektumot fogunk létrehozni. A karakterisztika pedig az objektum tulajdonságait szabja meg. Ide értjük az objektum adattípusát, mely lehet alaptípus, vagy annak szűkített változata is, és az esetleges dimenzióit is. A TEMP paraméter megadásával olyan objektumot hozhatunk létre, mely csak a jelenlegi kapcsolatunk alatt él, és az UPDATE parancs hatására sem mentődik el. Paraméterként adhatjuk meg azt is, hogy amennyiben több munkaterületet is felcsatlakoztattunk, akkor melyiken jöjjön létre az új objektum.

Példák:

```
DEFINE time DIMENSION ID
DEFINE adatok VARIABLE NUMBER(15,2) <time geo>
```

Parancs	Hatás
LD	Leírást adhatunk az objektumhoz
VNF	Valódi idő dimenziók esetén a formátumot adhatjuk meg vele
EQ	Képletek esetében úgy számítási szabályt adhatunk meg
PROGRAM	Eljárások kódja adható meg a segítségével
MODEL	Modell számítási szabályai adhatók meg
PERMIT	Az adott objektum módosítását tilthatjuk le, illetve engedélyezhetjük
PROPERTY	Extra tulajdonságokat adhatunk az objektumnak

6. táblázat: Objektummódosító utasítások

CONSIDER

A CONSIDER utasítás lehetővé teszi, hogy egy már definiált objektum tulajdonságait módosítsuk. DEFINE, RENAME és COPYDFN utasítások implicit tartalmaznak egy CONSIDER utasítást is. Hatása addig tart, míg egy nem tulajdonságmódosító parancsot nem adunk ki.

MAINTAIN

A MAINTAIN parancs a dimenziók, kompozitok és partíciós minták adatkészletének módosítását teszi lehetővé. Segítségével hozzáadhatunk érékeket a dimenzióhoz, törölhetünk, átnevezhetünk értékeket, módosíthatjuk az értékek sorrendjét, sőt egy másik dimenzióból is átvehetünk értékeket. Valódi időmenzió esetén lehetőség van n érték hozzáadására a dimenzió elejéhez illetve végéhez, valamint amikor új értéket adunk hozzá, a létrejövő két szélső érték között minden, a dimenzióhoz tartozó periódusérték létrejön.

Utasítás	Eredmény
ADD	Az argumentumként megadott értékeket hozzáadja a célobjektumhoz.
DELETE	Az argumentumként megadott értékeket törli a célobjektumból.
MERGE	Kompozitok esetén a két alapdimenzió értékeit rendelhetjük össze.
MOVE	Értékek listán belüli helyét változtathatjuk meg
RENAME	Értékek átnevezésére ad lehetőséget.

7. táblázat: MAINTAIN parancs műveletei

Példák:

```
MAINTAIN time ADD 'Q03_2005'
MAINTAIN prods RENAME 'Alma' 'Zöldalma' -
                      'Körte' 'Vilmoskörte'
MAINTAIN geo_prod MMERGE <geo prod>
```

LIMIT

Az egyes dimenziók és értékhalmazok értékkészletének beállítására szolgáló utasítás, tehát azok státuszát állítja be. Természetesen nem töröl adatokat a dimenzióból, csak az éppen használható értékek körét befolyásolja. Speciális programként rendelkezésünkre áll az ALLSTAT parancs, mely minden dimenzió limitjét ALL értékre állítja. Ez a parancs fontos szerepet játszik a relációk létrehozásában, az adatok aggregálásakor, és a jelentések készítése esetén is. Az utasítás általános alakja a következő:

```
LIMIT [dimenzió|surrogate|valueset] [limit típus] -
      [limit záradék] [IFNONE címke]
```

A limit záradék alapvetően két csoportba tartozó lehet, az egyikben konkrét értékeket, elempozíciókat adunk meg, a másik esetén pedig egy reláció alapján állítjuk be a státuszt. Az IFNONE kiegészítés eljárások esetén hasznos, mivel automatikus elágazást tesz lehetővé abban az esetben, ha a státusz NULL értékre változna, azaz a dimenziónak egyetlen eleme sem felem meg az új limit kritériumoknak.

Limit záradék	Példa
Értéklista	LIMIT time TO 'London' 'Paris' TO 'Berlin'
VALUESET	LIMT geo KEEP continents
ALL	LIMIT prod TO ALL
Logikai kifejezés	LIMIT prods TO ANY(sales GT 1000 prods)
BOTTOM|TOP	LIMIT district TO TOP 5 BASEDON costs
NTH {n|n TO n}	LIMIT geo TO NTH 1 TO 10

8. táblázat: Értékkészleten alapuló limit záradékok

Limit típus	Művelet
TO	A jelenlegi státuszt helyettesíti a megadott értékkészlettel
ADD	A megadott új értékekkel bővíti a jelenlegi státuszt, természetesen csak azokkal, melyek még nem részei
INSERT	Szintén hozzáadja az új értékeket, de az általunk megadott pozícióba szúrja be.
KEEP	A jelenlegi értékek közül csak azokat tartja meg, melyek a megadottak között is szerepelnek. A REORDER módosítóval, az új értékek sorrendjében újra is rendezi őket
REMOVE	A megadott értékek közül azokat, melyek szerepelnek a jelenlegi státuszban eltávolítja onnan
COMPLEMENT	Azok az értékek lesznek az új státusz elemei, melyek nem szereplenek a megadottak között
SORT	A megadott értékek sorrendjének megfelelően újrarendezi a státusz elemeit.

9. táblázat: Limit típusok

Limit záradék	Hatás
LEVELREL	A dimenzió azonos szintjén lévő elemek kerülnek egymás mellé
PARENTS, CHILDRENS	A jelenlegi státusz az elemeinek közvetlen szülőit illetve gyermekeit választja ki
ANCESTORS, DESCENDANTS	Az előzőhöz hasonló, de minden szintű őst illetve leszármazottat kiválaszt (szülő nagyszülő, stb.)
SIBLINGS	A jelenlegi elemek testvérelemei, azaz akiknek azonos a szülőjük
TOPANCESTORS, BOTTOMDESCENDANTS	Csak a legfelső szintű szülők, illetve legalsó leszármazottak
HIERARCHY	Hierarchikus elrendezés, megadható, hogy hány szintű legyen, mennyi szint maradjon ki, illetve hogy a szülők előre vagy hátra kerüljenek.

10. táblázat: Relación alapuló limit záradékok

STATUS

A parancs egy dimenzió aktuálisan használható értékkészletét írja ki, tehát praktikusan azt a LIMIT argumentumot, amellyel e jelenleg elérhető értékek köre elérhető. Argumentumként dimenziók felsorolását kaphatja, de meghívható argumentum nélkül is, ilyenkor minden dimenzió limitjét kilistázza.

Példák:

```
STATUS time
STATUS geo prods
STATUS
```

REPORT

Objektumok értékeinek kilistázására szolgál, legyenek azok dimenziók, értékhalmazok, változók, vagy bármely más értékekkel rendelkező objektum. Mivel ez a parancs használható adatkockák megjelenítésére is, így paraméterei között megadhatók formázási utasítások, az oszlopokban és sorokban szereplő adatok, valamint a jelentésre vonatkozó összegzési információk is, emiatt a későbbiekben az elemzési lehetőségek kapcsán még visszatérek rá.

Példák:

```
REPORT time
REPORT DOWN geo prods ACROSS time : sales
```

3.4 Dimenziók

A dimenziók, mint arra a multidimenzionális jelző is utal, alapvető elemei a MOLAP struktúrájú adattárházaknak. Ezek nélkül csak egy értékkel rendelkező változókat hozhatnánk létre, amelyek bár hasznosak, elemzési szempontból mégis értéktelenek. A dimenziók tartalmazzák a többdimenziós objektumok magyarázó adatait, segítségükkel azonosítható az n-dimenziós kocka egy vagy több (n-1)-dimenziós lapja, azaz szelete, mely szelet természetesen maga is kocka. Ennek megfelelően az Oracle OLAP DML több különféle dimenziótípus definiálására is lehetőséget biztosít.

3.4.1 Egyszerű dimenziók

A dimenziók alaptípusát képviselik, ennek megfelelően az általuk használható adattípusok köre is korlátozott. Csak szöveges, mely hosszát természetesen korlátozhatjuk, 32 bites egész (INTEGER) vagy rögzített pontosságú lebegőpontos szám lehet az adattípusa. A legáltalánosabban használt adattípus az ID és a TEXT. Amennyiben egy dimenzió szöveges típusú, és a méreténél hosszabb szöveget szeretnénk elhelyezni benne, akkor az csonkolódni fog, ez nem várt ütközéseket okozhat.

Példák:

```
DEFINE time DIMENSION ID
DEFINE prods DIMENSION TEXT WIDTH 20
DEFINE order_num DIMENSION NUMBER(10)
```

3.4.2 Idő dimenziók

Az adattárház definíciójából is következik, hogy az idő dimenzió a multidimenzionális adatbázisok alapvető eleme, hiszen történeti adatokat szeretnénk elemezni, és azokból a jövőre vonatkozó döntéseinket támogatni. Ennek megfelelően létezik egy dimenziótípus, mely az időszakok kezelését támogatja. Lehetőségünk van napi, heti, havi, negyedéves és éves felbontást választani, valamint a megjelenítés módját is megadhatjuk, például a hónap megjelenítése teljes névvel. Ezen felül, az OLAP DML ha több ilyen dimenziót is definiálunk, például egyet havi és egyet éves felbontással, akkor azonnal létrejön köztük egy implicit reláció, melyet természetesen lehetőségünk van ezt felülbírálni, ehhez nem kell mást tennünk, minthogy a két dimenzió között definiálni egy új kapcsolatot.

Ennek a dimenziótípusnak van egy lényeges korlátozása: nem hozhatunk létre rajta hie-

rarchiát, így az ilyen dimenzió felett nem végezhető tárolt aggregáció. Amennyiben így szeretnénk összesíteni, akkor az idő dimenziót szöveges típussal kell létrehoznunk, és a felett kell az aggregálást elvégezni. Természetesen ez nem jelenti azt, hogy nem lehetne idő dimenziók alapján összesíteni, csak a technikát kell megváltoztatnunk. A TCONVERT függvény segítségével kérhetünk az idő dimenziókon összesítéseket, ezek azonban a lekérdezés pillanatában értékelődnek ki.

Az idő dimenzió legfontosabb szerepe a speciális időszakok definiálásának lehetősége, például a pénzügyi év általában nem egyezik meg a naptári évvel, mondjuk márciustól februárig tart, vagy igény lehet kéthetes, féléves időszakokra is. Ehhez az idő dimenzió maximális támogatást nyújt azzal, hogy - természetesen csak dimenzió szinten - megadható, egy adott hónap, negyedév vagy év felbontású dimenzió kezdete és vége, illetve az, hogy a hét vagy hónap dimenzió egy értéke hány hetet, hónapot foglal el. A kiírási formátumot a VNF paranccsal adhatjuk meg, ha ezt nem tesszük meg akkor az alapértelmezett formátumban kerül kijelzésre.

Példák:

```
DEFINE honap DIMENSION MONTH
VNF '<yyyy> <mtextl0>
DEFINE ev DIMENSION YEAR
DEFINE kethet DIMENSION 2 WEEK
DEFINE penzugyi_ev DIMENSION YEAR ENDING jun
```

3.4.3 Összetett dimenziók

Lehetőségünk van több már létező dimenzióból összekapcsolással egy újat létrehozni, melyhez kétféle összekapcsolási mód áll rendelkezésünkre. A CONCAT függvény segítségével definiált összetett dimenziók esetében az értékek egymás után másolódnak, és kapnak az új dimenzióban egy kiegészítést, mely az eredeti dimenzióra utal. A másik lehetőségünk a dimenziók tényleges összekapcsolása, mely az alapdimenziókból való csoportképzés útján valósul meg. Fontos tudnunk azonban, hogy az összekapcsolt dimenziók, ellentétben az összemásoltakkal nem frissítődnek automatikusan, ha az alapdimenziókban változás áll be, azok feltöltéséről, nekünk kell gondoskodnunk.

Példák:

```
DEFINE cityandstate DIMENSION <city state>
MAINTAIN cityandstate ADD < 'London' 'England' >
DEFINE ido DIMENSION CONCAT (honap ev)
```

3.4.4 Dimenzió alias

Az OLAP DML eszközt ad arra is, ha több dimenziót szeretnénk egy azon értékkészlettel használni. Erre jellemző példa, amikor az alkalmazottakra vonatkozó adatokat szeretnénk vizsgálni, és szükségünk van a főnökök alapján való bontásra is, a főnökök pedig maguk is alkalmazottak. Ilyen esetekben lehetőségünk van alias definiálására a dimenzió felett. Innentől kezdve a két vagy több dimenzió ugyanazt az értékkészletet fogja használni, de minden más szempontból különböző dimenziónak fognak minősülni.

Példa:

```
DEFINE sell_time DIMENSION ALIASOF time
```

3.5 Értékhalmazok, relációk, csoportosítások

3.5.1 Valueset

Gyakran lehet szükségünk arra, hogy egy dimenzión belül bizonyos értékhalmazokat határozzunk meg, például azért, mert gyakran szeretnénk az éppen elérhető elemek körét egy ilyen halmaz tagjaira korlátozni. Ebben van segítségünkre a VALUESET objektum, mely a dimenzió elemeinek egy meghatározott halmazát reprezentálja, és könnyen korlátozható vele az elérhető dimenzióértékek köre. Egy adott dimenzió felett korlátlan számú értékhalmaz definiálható.

Példa:

```
DEFINE lineset VALUESET line
LD 'Adatsorok halmaza'
LIMIT lineset TO line
```

3.5.2 Relation

A dimenzióértékek között nem csak összetett dimenziók útján hozhatunk létre kapcsolatot, hanem definiálhatunk relációkat is. Ezek a relációk képezik az alapját az aggregálási szabályoknak. Két csoportba oszthatjuk, hierarchiák, ahol a reláció egy azon dimenzión belül van definiálva és csoportosítások, ahol különböző dimenziók között definiálunk kapcsolatot. Az ilyen relációknak két helyen van jelentős szerepük, az egyik a már korábban is említett aggregációk esetében, hiszen azokat kifejezetten hierarchiák mentén tudjuk kiértékeltetni. A másik fontos szerep, mint láttuk a LIMIT parancs kapcsán jelentkezik, itt ugyanis használhatunk relációkat a dimenzió elemeinek szűrésére, valamint a dimenzió elemeinek sorrendjét is meghatározhatjuk segítségével. Például kialakíthatunk hierarchikus elrendezést egy riport kapcsán, egy a dimenzión definiált reláció segítségével, esetleg megadva a maximális mélységet is.

Példák:

```
DEFINE time.time RELATION time<time>
DEFINE pros.categ RELATION categ<prod>
```

3.5.3 Surrogate

A SURROGATE objektumok segítségével további szűrési információkat adhatunk a dimenzióhoz. Például ha egy bizonyos dimenziónak sorszámozást is szeretnénk adni, akkor azt egy ilyen helyettesítő objektummal tehetjük meg, de megadhatunk kiegészítő szöveges információkat is. Az alapdimenzió és a SURROGATE objektum értékei között kölcsönösen egyértelmű megfeleltetés jön létre, ennek megfelelően teljes mértékben helyettesítheti az alapdimenziót. Adattípusait tekintve ugyanaz a korlátozás érvényes, ami az egyszerű dimenziók esetében ismertetésre került.

Példák:

```
DEFINE country.code SURROGATE country ID
LIMIT country TO 'Hungary'
country.code = 'hu'
```

3.5.4 Composite

Ha két vagy több dimenziót gyakran együtt szeretnénk használni, de a két dimenziót nem szeretnénk ténylegesen összekapcsolni, akkor hozhatunk létre kompozit objektumot. Az ilyen objektum automatikusan követi az alapdimenziók változását, legyen az értékkészlet, vagy limit változás.

Példa:

```
DEFINE market.product COMPOSITE <market product>
```

3.6 Változók

A dimenziók mellett a munkaterület másik legfontosabb objektumai a változók. Ezek több funkciót is betöltenek. Az egyik, hogy segédadatokat, számítási eredményeket tárolhatunk bennük, a másik, és messzemenően fontosabb, hogy a tényadatokat is ezekben tároljuk, tehát a „kockák" is változók. Ennek megfelelően lehetnek dimenzionáltak és dimenzió nélküliek is. Adattípusaikat tekintve értelemszerűen nincs korlátozás, tehát bármely adattípust használhatjuk. A definiálás során lehetőségünk van a SPARSE parancs segítségével névtelen kompozit dimenziókat megadni.

A változóknak értéket az '=', értékadó utasítással tudunk adni. Ekkor amennyiben dimenzionált változóról van szó, annak minden cellája, mely a dimenziókon beállított aktuális limiteknek megfelelően elérhető felveszi a megadott értéket. Ha a megadott érték és a változó adattípusa eltér, akkor automatikus konverzió történik, már amennyiben erre lehetőség van. Dimenzionált változók esetében lehetőség van egyes cellák elérésre, azáltal, hogy minősített indexelést adunk meg, ezt a változó neve után zárójelbe tett dimenzionév-dimenzióérték párokkal tehetjük meg.

Példák:

```
DEFINE x VARIABLE number(12,2)
x = 8
DEFINE sales VARIABLE number <time COMPOSITE<prod geo>>
cost(time 'M01_2000' geo 'Boston') = 0
```

3.7 Számítást segítő objektumok

3.7.1 Aggregáció

Az AGMAP objektum segítségével definiálhatunk a már említett aggregálásokhoz szabályokat. E szabályok jellemzője, hogy hierarchiákon, dimenzión belüli relációkon alapulnak. Megadható az, hogy az AGGREGATE parancs kiadásakor mely relációk mentén milyen feltételekkel és milyen függvény szerint kell az aggregációnak kiértékelődnie.

Példa:

```
DEFINE time.agg AGGMAP
AGGMAP
  RELATION time.rel
END
```

3.7.2 Függvények

Amellett, hogy az OLAP DML beépített függvények armadáját bocsátja rendelkezésünkre, lehetőséget biztosít saját függvények megadására is. Ennek egyszerűbb változata a FORMULA objektum, mely segítségével egy változóból kiindulva számított értékeket hozhatunk létre. Ilyenkor a függvény ugyanúgy lesz használható mint az eredeti változó, beleértve azt is, hogy a dimenzionálásuk is teljes mértékben meg fog egyezni. Lehetőség van ezen kívül visszatérési értékkel rendelkező programok írására is, mely eszközzel bonyolult számítások is elvégezhetőek.

Példák:

```
DEFINE tax FORMULA sales * 0.2
DEFINE tax.p PROGRAM NUMBER(12,2)
PROGRAM
  ARGUMENT val NUMBER(12, 2)
  RETURN val * 0.2
END
```

3.7.3 Modell

Az analitikus munkaterületen belül a MODEL objektum összekapcsolt számítások halmazaként van értelmezve, melynek során új értékek hozhatók létre változók, illetve dimenzió értékek alapján. Ha a modell dimenzió értékeket d meg, vagy dimenzió értékekre hivatkozik, akkor dimenzió alapú modellről beszélünk. Ilyen esetben szükségünk van egy változóra, mely egyrészt a kiindulási adatokat szolgáltatja, másrészt az eredmények tárolására is szolgál. A modellek egymásba ágyazhatóak, így bonyolultabb modellek építhetőek fel. A modellek gyakorlatilag programokká fordulnak le, viszont áttekinthetőbb kódot eredményeznek. Azoknál a dimenzióknál, melyek a számításokban részt vesznek, az értékeknél is be kell tartanunk az objektumokra vonatkozó névadási szabályokat.

Példa:

```
DEFINE actual.model MODEL
MODEL
  DIMENSION line
  profit = income - cost
  tax = profit * 0.2
END
```

4. Oracle OLAP DML eljárások

4.1 Az OLAP eljárások szerepe

Az adattárházak feltöltése és elemzése során gyakran találkozunk olyan feladatokkal, melyek megoldása manuálisan nehézkes, sok hibalehetőséget tartalmaz, viszont jelentős mértékben algoritmizálható. Ilyen esetek előfordulhatnak az adatbázis objektumainak definiálása, adatainak feltöltése során, de akkor is, amikor jelentéseket, elemzéseket szeretnénk készíteni. Például a hierarchiák feltöltése, és az adatok közötti kapcsolatok beállítása tipikusan ilyen.

A visszatérési értékkel rendelkező programok ezen kívül bonyolult számításokat végző függvények létrehozására is lehetőséget biztosítanak.

A programok egy speciális csoportját alkotják a triggerek, melyek olyan eljárások, melyek a munkaterület fel- illetve lecsatlakozásakor, objektumok módosításakor automatikusan lefutnak, és lehetőséget biztosítanak ellenőrzések végzésére. Ha egy trigger, mely egy logikai értékkel visszatérő program hamis értékkel tér vissza, akkor az adott feladat meghiúsul, és hibaüzenetet kapunk.

Működés szempontjából alapvetően két csoportját különböztethetjük meg a programoknak, az egyik az úgynevezett OLAP DML programok, melyek tisztán OLAP DML utasításokat tartalmaznak. A másik csoport az SQL alapú programok, melyek kapcsolatot teremtenek a relációs és a többdimenziós adatbázis között.

A munkaterületekhez való csatlakozáskor automatikusan felcsatolódó SYS.EXPRESS workspace rengeteg előre megírt programot, függvényt bocsát rendelkezésünkre. Ilyen például az AWDESCRIBE program, mely az munkaterület részletes leírását adja meg.

Eljárások definiálása egy PROGRAM objektum definiálásával történik. Ha megadtuk a végrehajtandó utasítás szekvenciát, akkor a COMPILE paranccsal fordíthatjuk le az eljárást, és a CALL paranccsal hívhatjuk meg. A munkaterületek működése miatt, nevezetesen, hogy átmeneti területen dolgoznak, a programok tesztelése előtt célszerű egy UPDATE és egy COMMIT utasítást kiadni, mert egy esetleges programhiba a korábbi munkánk elvesztését jelentheti.

4.2 Alap nyelvi elemek

Az OLAP DML nyelv is biztosítja a programíráshoz szükséges nyelvi elemeket. Használhatunk változókat elágazásokat, ciklusokat és vezérlésátadást. Egy programon belül más eljárások mint szubrutinok meghívhatóak, a rendszer megengedi a rekurzív eljáráshívást is, tehát a program saját magát is meghívhatja.

A programokon belül minden a munkaterületen elérhető parancs, függvény, eljárás használható. Ennek megfelelően új objektumokat is létrehozhatunk az eljáráson belül, és ezek az objektumok az eljárás végeztével is megmaradnak, és a továbbiakban is elérhetőek, sőt ha nem TEMP paraméterrel lettek definiálva, akkor el is menthetőek.

Az eljáráson belül belső blokkokat is megadhatunk a DO-DOEND utasításpár segítségével. A programértelmező működése miatt fontos, hogy a blokkindító és blokkzáró utasításokat külön sorba írjuk, mert csak így tudja a fordító összeegyeztetni őket.

4.2.1 Változók

Az eljáráson belül definiálhatunk lokális változókat, melyek azonban nem azonosak a munkaterületen definiált változókkal, bár használatuk azokéhoz hasonló. Definiálásuk során nem a DEFINE parancsot használjuk, hanem a VARIABLE utasítást, melyet a változó neve és a típusa követ. Itt is minden adattípust használhatunk. Az itt definiált lokális változók élettartama megegyezik az eljárás futási idejével, láthatósági köre pedig az eljárás blokkja.

Az eljárás látja ezen kívül a munkaterületi változókat is, sőt mivel az új objektumok definiálása is megengedett, új változókat is létrehozhat. Mivel ezek a változók a munkaterületen helyezkednek el reguláris objektumként, ezért sem láthatóságuk, sem pedig élettartamuk nem korlátozódik az eljárás blokkjára, továbbá a rajtuk végzett változtatások is megmaradnak.

A változók harmadik csoportját alkotják a program által argumentumként kapott értékek. Ezek a paraméterek lehetnek explicitek, tehát az eljárás meghívásakor kötelezően megadandóak, illetve implicit argumentumok, melyek megadása nem kötelező, legalábbis a program hívásának szintaktikai ellenőrzésekor hiányuk nem vált ki hibát. Az explicit argumentumok definiálása az ARGUMENT parancs használható, melynek szintaktikája megegyezik a VARIABLE utasításéval, és ezek a paraméterek a továbbiakban mint lokális változók használhatóak. Az implicit argumentumokat ellenben nem kell definiálnunk, viszont használatuk is kicsit nehézkesebb. Ezen paraméterek elérése az ARG függvény segítségével történik, mely függvény egyetlen argumentuma a paraméter sorszáma. Az ARG függvény visszatérési értéke TEXT típusú, és automatikus konverzióra kerül, ha szükséges. Az ARGCOUNT függvény adja vissza az átadott paraméterek számát. Az eljárások általában explicit argumentumátadást használnak, hiszen a várt paraméterek száma és típusa ismert, ha azonban változó számú vagy típusú argumentumra van szükség, akkor célszerű az implicit paraméterátadás alkalmazása. Az argumentumok láthatósága és élettartama természetesen a blokkra korlátozott.

Időnként szükséges lehet, hogy változókban, különösen paraméterekben tárolt nevek alapján érjünk el munkaterületi objektumokat, például általános, adott dimenzionálással rendelkező „kockák” adataiból készített jelentések esetén. A lokális változókat - s mivel az argumentumok is annak tekinthetőek - is használhatjuk hivatkozási változóként. Ehhez nem kell mást tennünk, mint a változó neve elé tenni egy '&' karaktert, ezáltal a program futásakor a változó aktuális értéke behelyettesítődik, mint objektumnév, vagy akár mint teljes .utasítás.

4.2.2 Elágazások

Két típusú elágazást különböztetünk meg, az egyik az egyszerű elágazás, melyet az IF-THEN-ELSE szerkezettel érhetünk el. Ez egy általunk megadott feltétel alapján választja az egyik, vagy a másik blokkot. Ha az adott végrehajtási ág csak egyetlen utasítást tartalmaz, akkor nem kell köré blokkot definiálnunk, de ilyenkor a blokkot meghatározó elemmel egy sorba kell írni, míg ha több utasítást is végre kell hajtani akkor a blokk megadása kötelező. A THEN és az ELSE elem is külön sorba írandó, blokkindító és blokkzáró elemekhez hasonló okokból. Feltételként alkalmazhatunk logikai értékkel visszatérő függvényeket, eljárásokat, logikai változókat és összehasonlításokat is. Az OLAP DML programokban nem a már jól ismert relációs jeleket használhatjuk összeha-

sonlításra, hanem speciális rövidítéseket vezettek be e célra.

Operátor	Jelentés	Példa	Prioritás
NOT	Logikai negáció	NOT YES = NO	1
EQ	Egyenlő	7 EQ 8 = NO	2
NE	Nem egyenlő	7 NE 8 = YES	3
GT	Nagyobb, mint	9 GT 8 = YES	4
LT	Kisebb, mint	7 LT 9 = YES	5
GE	Nagyobb vagy egyenlő	8 GE 8 = YES	6
LE	Kisebb vagy egyenlő	7 LE 9 = YES	7
IN	A dátum az intervallumban van -e	'01jan02' IN v2.02 = NO	8
LIKE	Szöveg mintaillesztése	'Veszprém' LIKE '%prém%' = YES	9
AND	Logikai és kapcsolat	8 GE 8 AND 6 LT 5 = NO	10
OR	Logikai vagy kapcsolat	8 GE 8 OR 6 LT 5 = YES	11

11. táblázat: OLAP DML logikai operátorok

A másik elágazástípus a többválasztásos elágazás, mely a SWITCH szerkezettel került megvalósításra. Változóként bármilyen típusú változó használható, nem csak felsorolási típusok. Minden esetet CASE taggal kell kezdeni, és BREAK utasítással kell befejezni, BREAK hiányában a C típusú nyelvekhez hasonlóan a végrehajtás a következő CASE szakaszon folytatódik. DEFAULT taggal adható meg alapértelmezett ág.

4.2.3 Ciklusok

Ciklusok tekintetében szintén két típus áll rendelkezésünkre, az egyik a már jól ismert léptető ciklus, mely itt is a FOR utasítással van megvalósítva. Azonban ennek a ciklusnak fontos korlátozása, hogy csak dimenziók felett van értelmezve, tehát egy dimenzió aktuálisan látható értékein lépkedhetünk vele végig.

A másik ciklus, ami rendelkezésünkre áll, az előltesztelő ciklus a WHILE paranccsal megvalósítva. A ciklusok megvalósításának ez a korlátozása nem jelent hátrányt, hiszen elöltesztelő ciklussal minden ciklust igénylő algoritmus megvalósítható.

Minden cikluson belül lehetőségünk van a ciklus végére ugrani a CONTINUE parancscsal, ilyenkor a ciklus a következő iterációra lép, feltéve, hogy a WHILE utasításban szereplő feltétel igaz, illetve hogy a FOR ciklusban szereplő dimenzió még rendelkezik nem iterált értékkel. A BREAK utasítás hatására viszont azonnal kilép a ciklusból, és a végrehajtás a ciklust követő első utasításon folytatódik. A ciklusból a vezérlésátadó utasítások használatával is kiléphetünk.

4.2.4 Vezérlésátadás

A korábban ismertetett alprogram híváson kívül vannak más módok is amelyekkel a vezérlést egy másik programrésznek adjuk át, mégpedig a vezérlésátadó utasítások segítségével. Az egyik vezérlésátadó utasítás a RETURN parancs, mely azonnal kilép az eljárásból. Segítségével adhatunk vissza értékeket, a paramétereként megadott érték

lesz az eljárás visszatérési értéke.

A másik vezérlésátadó utasítás a GOTO, mely egy az eljárásban a LABEL elem segítségével definiált címkénél folytatja a végrehajtást.

4.2.5 Hibakezelés

A programok futása során óhatatlanul előfordulhatnak hibák, ezek alatt nem a programozási hibákat kell most értenünk, hanem azokat a hibákat, melyek azért következnek be, mert az adattárház adataiban, beállításaiban vannak olyanok, melyekkel előre nem tudtunk számolni. Ilyenkor nem feltétlenül szeretnénk, hogy az eljárás végrehajtása megszakadjon, például egy több jelentést is létrehozó program esetében elegendő lenne, ha az érintett riport helyén jelenne meg hibaüzenet, a többi pedig hibátlanul megjelenne.

A TRAP parancs segítségével állíthatjuk a hibakezelés paramétereit. Az ON argumentum után adható meg az a címke, amelyre hiba esetén ugrani kell, egyszerre egy ilyen címke adható meg, a később kiadott TRAP ON parancs felülírja a korábbit. Ez egy kicsit nehézkessé teszi a hibakezelő blokkok egymásba ágyazását, de nem lehetetlen, bár egymásba ágyazott eljárások esetén más komoly tervezést igényel. A hibakezelés működése az egyetlen olyan dolog, mely a GOTO utasítás létét indokolttá teszi. A NOPRINT paraméterrel megakadályozhatjuk, hogy a hibaüzenet automatikusan kiíródjon a kimenetre, míg a PRINT paraméterrel visszaállítható az alapértelmezett működés.

Hiba esetén a hiba típusát az ERRORNAME, a leírását pedig az ERRORTEXT opciók tartalmazzák. Az ERRORNAME és a SWITCH szerkezet segítségével igen jó hibakezelés érhető el.

Ha arra van szükségünk, hogy valamely feltételek esetén egy hibát váltsunk ki, akkor arra a SIGNAL utasítás segítségével nyílik lehetőségünk. Az utasítás első paramétere a hiba neve, amely lehet szabvány név és saját megnevezés is, a második pedig a hibaüzenet.

4.2.6 Külső kapcsolatok

A munkaterületet tartalmazó relációs adatbázis és a többdimenziós adatbázis között SQL utasítások segítségével hozhatunk létre kapcsolatot, ezt a e fejezet SQL alapú programokról szóló részében térek ki.

A másik lehetőség a fájlok elérése, melyek szerepel kimenetként és bemenetként egyaránt. Ahhoz, hogy fájlokat használhassunk az adattárházban rendelkeznünk kell legalább egy DIRECTORY objektummal a relációs adatbázisban, csak az ilyen módon csatolt könyvtárakban lévő fájlok érhetők el, és csak ilyenekben hozhatunk létre újakat. A legegyszerűbb fájlhasználat a kimenet fájlba irányítása, melyet az OUTFILE utasítással érhetünk el. A másik lehetőség a fájlok parancsok általi kezelése, ehhez rendelkezésünkre áll a megnyitás (FIELOPEN), melynél az is megadható, hogy milyen célra szeretnénk megnyitni a fájlt, és a karakterkódolást is megadhatjuk. A parancs visszaad egy INTEGER típusú azonosítót, mellyel kérőbb hivatkozhatunk a megnyitott állományra. Természetesen a fájlt be is zárhatjuk (FILECLOSE), olvashatunk belőle (FILEREAD, FILEGET) vagy akár írhatunk is bele (FILEPUT). A FILEQUERY utasítással a fájl tulajdonságait, míg a CDA paranccsal az aktuális könyvtárat kérdezhetjük le. Speciális kimeneti fájlként a DBGOUTFILE utasítással megadható egy fájl, melybe a hibakeresési információk kerülenk.

4.3 Programtípusok

4.3.1 OLAP DML Programok

A tisztán OLAP DML utasításokat tartalmazó eljárások négy fő területen kapnak szerepet, az egyik a dimenziók, hierarchiák és más objektumok karbantartása, a másik a bonyolult függvények megvalósítása, a harmadik a triggerek megvalósítása, a negyedik pedig a jelentések készítése. Mind a négy esetben szükségünk van arra, hogy egyes dimenziók limitjeit igényeink szerint módosítsuk, ez azonban nem lehet hatással az eljárás hívásakor a munkaterületen beállított limitekre, és opciókra. Tehát a dimenziók limitjei és a beállítások ugyanabban az állapotban kell hogy legyenek az eljárásból való visszatérés után, mint a hívása előtt voltak. E cél elérésére a környezet objektum és a verem áll rendelkezésünkre.

A verem, mint az köztudott egy LIFO típusú tároló, melyben dimenziókat és skalár változókat helyezhetünk el. A struktúrának megfelelően a berakás (PUSH) és a kivétel (POP) művelet került megvalósításra, viszont van néhány speciális tulajdonsága. Az egyik, hogy valójában nem is egy klasszikus verem, hanem inkább mint veremhalmaz képzelhető el, ennek megfelelően a POP parancs nem a legutolsóként behelyezett objektumot adja vissza, hanem név szerint keresve, a legutolsó adott névvel rendelkezőt. A másik speciális tulajdonság, vagyis inkább funkció, hogy a PUSHLEVEL paranccsal definiálhatunk visszaállítási pontokat, melyeknek nevet kell adnunk. A visszaállítási pontok a teljes veremstruktúrára vonatkoznak. A POPLEVEL utasítás hatására a paraméterben megadott nevű visszaállítási pontig minden veremelemre meghívódik a POP utasítás, természetesen figyelembe véve a behelyezésük sorrendjét.

A környezet objektumot egy CONTEXT utasítás útján érhetjük el, melynek első paramétere a környezet neve, második pedig az egy környezeti parancs, amit az esetlegesen szükséges egyéb paraméterek követnek. Ennek megfelelően egy adott munkaterületen, illetve eljárásban több különböző környezetünk is lehet, és azok objektumait közvetlenül nem csak elmenthetőek és visszaállíthatóak, de akár módosíthatóak is. Szokás, és célszerű a munkaterülethez való csatlakozáskor egy saját környezetet létrehozni, és az aktuális állapotokat elmenteni, hogy szükség esetén azok visszaállíthatóak legyenek, ennek legegyszerűbb megoldása egy fel- és egy lecsatlakozási trigger létrehozása, vagy az AW ATTACH esetén az ONATTACH, paraméterben egy program megadása. Ez utóbbi viszont, mivel nem automatikus, hibalehetőségeket tartalmaz.

Sem a környezet, sem a verem nem munkaterületi objektum, így a munkaterület elmentésekor nem mentődik el, élettartama a kapcsolatéval egyezik meg.

4.3.2 SQL alapú programok

Az SQL alapú programok kapcsolatot teremtenek az adattárházat tartalmazó relációs adatbázissal. Ennél fogva fő szerepük az adatok importja, legyen szó akár dimenzió vagy surrogate értékekről, akár tényadatokról. Természetesen a kapcsolat nem egyirányú, így lehetőségünk van a relációs táblák adatainak módosítására is, valamint tárolt eljárásokat is meghívhatunk, sőt amennyiben igény mutatkozik rá, akár új táblákat, nézeteket, eljárásokat is létrehozhatunk.

A relációs adatbázissal kapcsolatot teremtő utasítások közös jellemzője, összetett parancsok, azaz az SQL utasítás paramétereként lehet megadni a tényleges műveletvégző uta-

sítást.

Relációs táblákban lévő adatok elérése

Lekérdezések kezelésére a PL/SQL tárolt eljárásokhoz hasonlóan itt is rendelkezésünkre áll az egyszerű SELECT INTO szerkezet, mely azonban csak egy sort adhat vissza. A másik lehetőségünk az eredményhalmazok használata, melyeknél lényeges különbség a tárolt eljárásokkal szemben, hogy itt nincs implicit kurzor, tehát minden használni kívánt eredményhalmazt előre deklarálnunk kell a DECLARE CURSOR paranccsal. Az eredményhalmazok megnyitása az OPEN, bezárása padig a CLOSE paranccsal lehetséges. Adatok beolvasása egyrészt az IMPORT INOT, másrészt a FETCH INTO szerkezet segítségével lehetséges, a két utasítás között az alapvető különbség, hogy a FETCH nagyobb funkcionalitást nyújt, az IMPORT viszont sokkal hatékonyabb akkor, ha sok adatot kell egyszerre beolvasni. A FETCH parancs a LOOP paraméter segítségével biztosít lehetőséget a kurzor teljes további tartalmának beolvasása. Mindkét utasításnál a *target* szakaszban adhatóak meg azok a dimenziók, és mezők, melyekbe a beolvasás megtörténik. Ciklusos beolvasás esetén a célobjektumok lehetnek dimenziók, surrogate objektumok és „kocka” változók. Beolvasáskor az eredményhalmaz dimenzió mezői által meghatározott surrogate elemekbe, „kockacellákba” kerülnek a tényadatok. A dimenziók esetén választhatunk, hogy abban az esetben, ha a még nem szerepel az aktuális érték az objektumban, akkor hozzáadjuk (APPEND), hibát váltunk ki (MATCH) vagy kihagyjuk az aktuális rekordot (MATCHSKIPPERR). A THEN paraméter után megadhatunk olyan parancsokat, melyek minden olyan esetben lefutnak, amikor egy rekord sikeresen bekerül az adattárházba. Ezzel a betöltés hatékonysága növelhető, hiszen bizonyos esetekben hatékonyabban elvégezhetők az utólagos beállítások, mint a teljes adatimport befejezése után. Ha eredményhalmazt használunk, akkor a CLEANUP paranccsal tudjuk az általa használt memóriaterületet felszabadítani, s ez erősen ajánlott is egyben, hiszen az adatok mennyisége igen jelentős.

SQL utasítások, tárolt eljárások

Mint arról korábban szó volt, nem csak lekérdezések futtatására van lehetőségünk, hanem gyakorlatilag bármilyen RDBMS utasítást kiadhatunk. Ehhez nem kell mást tennünk, mint a relációs adatbázisban használt utasítást az SQL parancs után írni, és az automatikusan ott fog végrehajtódni. Ezekben az utasításokban használhatunk paramétereket, melyeket ':'-tal vezetünk be, azonosan mint az RDBMS-ben. A paraméterek itt változók, és dimenziók lehetnek. Változók esetében csak a dimenzió nélküliek használhatóak, dimenziók esetében pedig vagy egyetlen értékre kell limitálnunk vagy pedig, ami ezzel egyenértékű egy FOR ciklus belsejében kell a műveletet végrehajtani. Ciklusban végrehajtott azonos utasítások esetén jól használható a PREPARE és EXECUTE parancs, melyek segítségével az utasítást még a cikluson kívül értelmeztetjük és lefordíttatjuk az RDBMS motorral, majd a cikluson belül már ezt a lefordított utasítást hajtatjuk végre a megfelelő paramétereket behelyettesítve. Ez lényegesen hatékonyabb végrehajtást tesz lehetővé, hiszen az elemzést, szintaktikai ellenőrzést, optimalizálást és fordítást csak egyszer kell elvégezni. A PL/SQL programok meghívását a PROGRAM parancs teszi lehetővé, természetesen itt is használhatunk változókat és dimenziókat paraméterként.

5. Mintapélda

Mintapéldaként egy olyan adattárház építésének lépéseit kívánom bemutatni, mely a Rendezvényjegy Kft. részére készül. A Kft. részben különböző cégek rendezvényeire történő jegyértékesítést támogatja azok jegypénztári és saját jegyirodai értékesítő rendszerét biztosítva, részben pedig webáruházat üzemeltet, hogy ezen partnerek és más cégek rendezvényeinek jegyeit értékesítse.

A jegyértékesítési rendszerrel támogatott partnerek adatbázisai egységes felépítésűek, továbbá mindegyik saját adatbázissal rendelkezik, az egyéb rendezvények jegyei pedig jegykontingens formájában áll rendelkezésre, melyet saját erre a célra fenntartott adatbázisban tárolnak. Ezeket közösen a továbbiakban lokális adatbázisoknak nevezzük.

Az internetes értékesítést egy külön adatbázis támogatja, mely DB-Linkeken keresztül kapcsolódik a többi adatbázishoz, ezt a továbbiakban központi adatbázisnak nevezzük ez az adatbázis tartalmazza minden rendezvények alapadatait, melyek replikáció útján kerülnek át a lokális adatbázisokból, és az internetes vásárlások alapadatait, melyek viszont az érintett lokális adatbázisba kerülnek át replikációval. Az, hogy az adott vásárlás során mely jegyeket vették meg, csak a lokális adatbázisokban tárolódik, ily módon egy csillag struktúrájú adatbázisháló alakult ki.

2. ábra: A központi és a lokális adatbázisok

A példa érthetőségét elősegítendő a következő egyszerűsítéseket vezetjük be:

1. A lokális adatbázisokat az internetes adatbázisban lévő sémákkal reprezentáljuk, de továbbra is, mint lokális adatbázisokra hivatkozunk rájuk.
2. Csak egy jegyértékesítési rendszerrel támogatott partnert veszünk figyelembe, és a saját egyéb rendezvényeket tartalmazó adatbázisunkat, és azt is csak két rendezvényszervezővel.
3. Egy teremnek csak egy nézőtéri elrendezése lehet.
4. Egy rendezvényre csak maximum egy bérlet lehet érvényes.
5. Egy bérlet csak egy terem előadásaira érvényes.
6. Egy vásárlásban csak egy partner jegyei vehetnek részt, de azok több rendezvényére is szólhatnak.
7. A webáruházban csak bankkártyás fizetésmód megengedett.
8. Figyelmen kívül hagyjuk mindazon táblákat, melyek nem közvetlenül kapcsolódnak az értékesítéshez.

9. Az érintett táblák struktúráját is a maximális mértékben redukáljuk.
10. A rendszer többnyelvű, de egyszerűsítésként csak a magyar nyelvet vesszük figyelembe.

5.1 Specifikáció

Az adattárház célja a Webáruházzal kapcsolatos stratégiai döntések meghozatalának támogatása. Ennek megfelelően a jegypénztári és jegyirodai eladások tételesen nem érdekesek, csak összesítésként, havi bontásban az internetes értékesítés arányának megállapításához használjuk.

A következő kérdésekre szeretnénk választ kapni:

1. Hogyan változott a webáruház forgalma az idők folyamán?
2. Hogyan változott az internetes és a jegyirodai értékesítés aránya?
3. Hány nappal az előadás előtt vásárolnak a vevők jegyet?
4. Milyen az internetes vásárlók összetétele lakóhely szerint?
5. Hogyan változik a szállítási módok aránya?
6. Milyen ársávokban lévő jegyek illetve bérletek fogynak a legjobban?
7. Hogyan oszlik meg az eladott jegyek száma és a bevétel műfajok szerint?
8. Milyen a sikeres és sikertelen vásárlási kísérletek aránya, és hibacsoportonkénti megoszlása?
9. Melyik volt a legjobb tíz nap illetve öt hónap a jegyértékesítés szempontjából, a bevétel és az eladott jegyek száma, valamint jegy és bérlet szerint külön-külön?

5.2 Nyilvántartott adatok:

Az itt megadott adatok csak ősmodellnek tekinthetőek, nem célom a tényleges struktúra ismertetése, az az ábráról leolvasható. Természetesen minden táblának létezik elsődleges kulcsa ezek felsorolása szintén nem célom.

5.2.1 Internetes eladások

Minden internetes vásárlásról rögzítjük a következő adatokat:

- A vevő IP címe
- A vevő azonosítója
- A fizetendő összeg
- A szállítás módja, és annak költsége
- A vásárlás kezdete és vége
- A vásárlási kísérlet sikeressége, illetve sikertelensége, és sikertelenség esetén annak oka
- A lokális adatbázis kódja
- A kiválasztott jegyek száma (nem azonos a megvásárolt jegyek számával)

5.2.2 Eladások

Minden eladásról, tehát nem csak az internetesről tárolódnak ezek az adatok, helyük a lokális adatbázisokban van.

- A vásárlás ideje
- A vevő azonosítója
- A megvásárolt jegyek, és azok eladási ára
- A fizetés adatai (fizetésmód, fizetendő, kapott és visszajáró összeg)
- Az eladó azonosítója
- Kért-e számlát
- Internetes vagy jegyirodai eladás-e
- A szolgáltatási díj neve és összege (itt tárolódik a szállítási díj)

5.2.3 Vevő adatai

Minden vevőről akár internetes, akár nem tároljuk a következő adatokat. Ezek mind a lokális, mind a központi adatbázisban jelen vannak.

- Neve
- Címei (szállítási cím, lakcím)
- Telefonszám
- E-mail cím

5.2.4 Rendezvény adatai

A rendezvény alapadatai, ezek is mind a központi, mind a lokális adatbázisban tárolódnak.

- Megnevezése
- Időpontja
- Tulajdonosa
- Műfaj
- Értékesítési szint
- Státusz (még nem értékesíthető, értékesíthető, értékesítés befejezve)
- Terme
- A rá érvényes bérlet

5.2.5 Bérlet adatai

A partner cégek egy része bérleteket is árul.

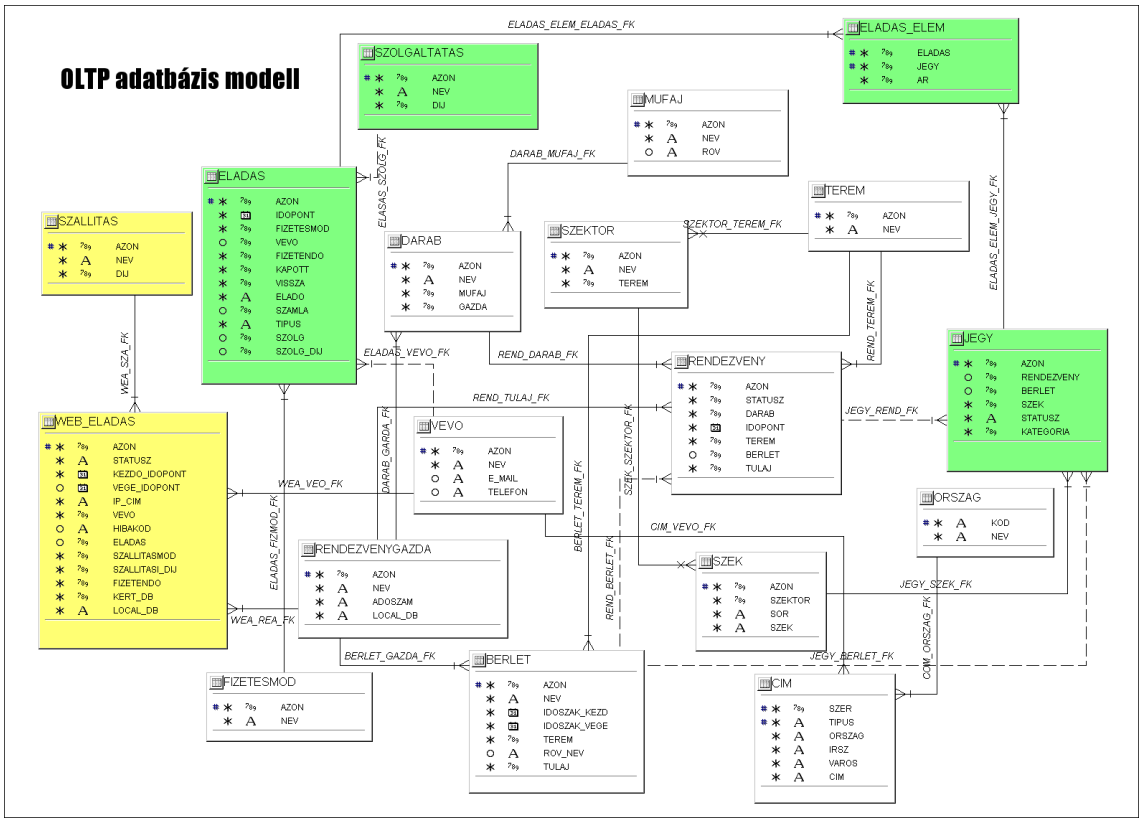
- Bérlet neve
- Időszak kezdete és vége

5.2.6 Jegyek adatai

Az egyes eladható jegyek adatai, csak a lokális adatbázisban vannak jelen.

- A jegy széke, ahová szól
- A rendezvény illetve bérlet, amelyre szól
- A jegy státusza
- A rendezvényszervező által meghatározott kategória, mely a jegy „jóságát”, és árát határozza meg.

A rendszer működéséhez nyilvánvalóan ennél sokkal több adatra van szükség, az adattárház szempontjából viszont ezek irrelevánsak, és csak feleslegesen bonyolítanák a példát.



3. ábra: Relációs adatbázis séma

A 2. ábrán látható az egyszerűsített relációs adatbázis sémája.[2] Mint látható a kezdeti egyszerűsítések sem vezettek túlegyszerűsített sémához.

2 A zöld színű táblák a lokális, a sárgák a központi adatbázisban, a fehérek pedig mindkét típusban releváns tartalommal bírnak. A3-as méretben megtekinthető a mellékletben.

5.3 Tervezés

5.3.1 Általános tervezés

Első lépésként a dimenziók és a tényadatok meghatározását kell elvégeznünk.

A specifikáció során megadott kérdések alapján látható, hogy tényadatnak az eladott jegyek darabszáma és a keletkező bevétel tekintendő, kivétel a 8. kérdés, mely a tranzakciók számára vonatkozik. A dimenziók meghatározásnál célszerű a meghatározott kérdéseket sorra venni, és azok alapján egyenként meghatározni a szükséges adatokat.

1. kérdés: Idő és tulajdonos dimenzió létrehozása szükséges, hiszen nyílván partnerenként is szeretnénk ezt látni.
2. kérdés: Figyelembe véve azt a döntést, hogy a jegyirodai eladásokat csak összegszerűen, kizárólag idő és tulajdonos szerint dimenzionálva importáljuk az adattárházba, új dimenzió nem lenne szükséges. A teljesítmény figyelembevételével viszont célszerűbb úgy dönteni, hogy létrehozunk még egy dimenziót, mely két értékkel rendelkezik (jegyirodai és internetes), és ezzel, valamint egy idő és a tulajdonos dimenzió segítségével külön adatkockát, illetve modellt hozunk létre ehhez az elemzéshez.
3. kérdés: Szükség van egy olyan dimenzióra, mely a vásárlás és a rendezvény időpontja között eltelt időt tartalmazza.
4. kérdés: A lakhely dimenzió létrehozását teszi szükségessé.
5. kérdés: A szállítási mód dimenziót indokolja.
6. kérdés: E kérdés alapján két dimenzió kell, az egyik az árkategória, a másik pedig azt jelzi, hogy jegy vagy bérlet vásárlásról van-e szó.
7. kérdés: E kérdés miatt egy műfaj dimenzió is nélkülözhetetlen.
8. kérdés: Egy idő és egy hibaok dimenzió kell az elemzéshez, itt is a 2. kérdésnél leírtak szerint célszerű eljárni.
9. kérdés: A szükséges adatok már rendelkezésre állnak.

Az 2. kérdésen összesítésen alapuló kimutatások elkészítéséhez egy modell létrehozása célszerű, hiszen a két alapadatból (jegyirodai és internetes értékesítés) már minden további mutató kiszámítható. Így a fent meghatározott három dimenzió megmarad, csak az értékesítési módot tartalmazó dimenziót kell bővíteni mindazon mutatókkal, melyekre szükségünk van. Az idő dimenzió kapcsán, mivel az adatok jelentős része amúgy is számított, dönthetünk egy valódi idő dimenzió létrehozása mellett is. Ez utóbbit indokolja az a tény is, hogy a rendezvényszervezők működése szempontjából nem a naptári év a mérvadó, hanem az évad, mely általában szeptembertől júniusig-júliusig tart, az értékesítés pedig gyakran augusztusban kezdődik. Ennek megfelelően a bevételi kimutatások esetében célszerű egy évad bontást is végezni, ami csak valódi idő dimenzió felett oldható meg egyszerűen, hiszen normál dimenzió felett nem definiálható kétféle hierarchia. Itt az idő hierarchia felbontása döntés alapján havi, mivel a cégvezetés úgy gondolja, hogy ez az a legkisebb időegység, amely összehasonlításra alkalmas adatokat szolgáltat. Ezen kívül negyedéves, éves és évad szintű összesítésekre lesz szükség.

A 8. kérdés esetén az adatokat szintén célszerű egy külön adatkockába helyezni. hiszen

mindösszesen egy idő és egy hibaok dimenzióra van szükségünk. A tényadat ebben az esetben a tranzakciók száma. Az arányt külön függvény segítségével határozhatjuk meg. A cégvezetés itt is a havi felbontást tartja informatívnak, és csak esetenként szeretne negyedéves és éves összesítést. Ennek megfelelően az idő dimenziót, valódi dátum alakú dimenziónak választjuk.

A többi kérdés esetén, mivel azok jellemzően kisebb időszakok összehasonlításán, illetve teljes összesítésre vonatkoznak, célszerűbb az idő dimenzión egy hagyományos hierarchiát létrehozni, és az összesítéseket aggregáció útján elvégezni. Ennél az időmenziónál a legkisebb értékelhető időtartomány a nap, melyet harmadhónapok, hónapok, negyedévek és évek szerint kell összesíteni. Az előre vásárlás esetén csak az érdekes, hogy hány héttel korábban történt a vásárlás. Lakhely esetén a vezetőség kétféle csoportosítást jelölt meg, egyrészt szüksége van a hazai és külföldi vevők megoszlására, másrészt a külföldi vevőket országonként, ha hazaiakat pedig budapesti és vidéki kategóriák szerint szeretné megtekinteni. Az idő dimenzió kivételével mindnél van egy olyan hierarchia szint, mely minden elemet magába foglal. Ezt mindig „Mind” elemmel reprezentáljuk.

A fenti adatok alapján két nyolc-, egy háromdimenziós és egy kétdimenziós adatkockánk fog létrejönni. A nyolc dimenzió első pillantásra soknak tűnhet, hiszen a tervezési alapoknál azt állapítottuk meg, hogy 5-6 dimenziónál több túlzottan megnöveli a méretet. Azonban az egyik dimenziónk csak két elemű (jegy vagy bérlet), és van két kategóriákon alapuló is, melyeknek szintén nem lesz nagyszámú eleme, így az adatkockák mérete sem lesz túlzottan nagy.

A létrejövő adatkockák, az azonos dimenzióval rendelkezőket együtt kezelve:

I. Internetes eladások

a. *Tényadat* : Interneten eladott jegyek összértéke, illetve darabszáma

b. *Dimenziók*:

1. Idő, napi felbontás, harmadhavi, havi, negyedéves és éves hierarchia
2. Tulajdonos, minden partner összesítéssel
3. Előrevásárlás, hetes bontással
4. Hely dimenzió, hazai és külföldi kategóriák, a hazaiak Budapest és vidék a külföldiek ország szerint szétválogatva, hierarchikus szerkezetben
5. Szállítási mód, minden módot tartalmazó szinttel
6. Jegy vagy bérlet, és természetesen mindkettő
7. Az árkategória, melynél szintén van mindet tartalmazó szint
8. Műfaj, természetesen az összeset tartalmazó hierarchiával

II. **Forgalmi összesítések** (modell):

a. *Tényadat*: Eladott jegyek összára, azaz a bevétel

b. *Dimenziók*:

1. Idő, havi felbontással, mely összesíthető negyedévenként, évenként és augusztustól júliusig tartó évadonként.

2. Rendezvényszervezők
3. Forgalmi adatok dimenzió, a következő elemekkel:
    1. Jegyirodai értékesítés
    2. Internetes értékesítés
    3. Összes értékesítés
    4. Internetes értékesítés aránya az összes értékesítéshez viszonyítva
    5. Jutalék, mely az internetes értékesítés 10%-a
    6. Nettó árrés, mely a jutalék ÁFA-val csökkentett értéke (ÁFA-t az egyszerűsítés miatt tekintsük végig 20%-nak)

III. Sikeres és sikertelen tranzakciók

a. *Tényadatok*: a tranzakciók száma

b. *Dimenziók*:

1. Idő, mely havi felbontású és negyedéves és éves összesítésre is lehetőséget biztosít. E célra teljes mértékben használható a forgalmi összesítéseknél használt dimenzió.
2. Hibakategóriák, a hibakódokat csoportokba soroljuk.

5.3.2 Adatkiválasztás

Miután a célok alapján meghatároztuk az adattárház elméleti struktúráját, el kell döntenünk, hogy a rendelkezésre álló adatokból hogyan állíthatjuk elő a szükséges tartalmat. Ehhez ismét célszerű típusonként külön tárgyalni adatkockákat, a dimenziók esetében egyenként dönteni az adat forrásáról, és az esetleges problémák kezeléséről.

Internetes eladások

Ezt az adatkockát tekintve triviális a WEB_ELADAS relációs tábla, mint központi tábla használata a forrásadatok tekintetében. Ehhez kapcsoljuk mindazon táblákat, melyekből forrásadatra van szükségünk.

Tényadatok

Az összeg, és a vásárolt jegyek száma az ELADAS_ELEM táblából kerül kiszámításra. Az összegben nem vesszük figyelembe a szállítási díjat, csak a jegyek árát.

Idő dimenzió

A WEB_EALDAS tábla két időadatot is tartalmaz, egyrészt a vásárlás kezdetének, másrészt a végének idejét. A működést ismerve az a döntés született, hogy ezek közül a végidőpontot tekintjük a vásárlás idejének, mégpedig azért, mert a jegyek véglegesen vevőhöz rendelése ekkor történik meg. Ez a mező nem tartalmaz anomáliákat, mert a kitöltése a tranzakció lezárásakor automatikusan megtörténik, és nyitott tranzakció, tehát amely 0-s státuszban van, nem maradhat.

Tulajdonos dimenzió

Elemeiben a partnerek azonosítóit tartalmazza, és surrogate objektumban kötjük hozzá a

neveket, ezek az adatok szintén nem tartalmaznak hibákat, hiszen központi adatbevitel útján jönnek létre.

Előrevásárlás dimenzió

Bérlet esetén megállapodás szerint az időszak kezdetét tekintjük kiindulási időpontnak, rendezvény esetén pedig természetesen a kezdési időpontot. Ezen értékek meghatározásához egyrészt szükséges a vásárlás ideje, másrészt az alapidőpont, mely a RENDERVENY illetve a BERLET táblákban található. Szerencsére itt sem kell hibákkal számolnunk.

Hely dimenzió

A vevők címét tekintjük releváns adatnak, mégpedig ha több címe is létezik, akkor a hivatalos címet (H típus). Miután a cím kitöltése internetes vásárlásnál kötelező, de mivel az in office rendszerekkel támogatott partnerek esetén, a közös vevő adatbázis miatt elképzelhető, hogy a cím később törlésre kerül. Ezért, ha nincs cím akkor megpróbálunk az e-mail címből kiindulni, ha az e-mail cím ország alapú domainhez tartozik, akkor azt az országot tekintjük lakhelynek, ha viszont nem, akkor egyszerűen magyar budapesti lakosként kerül be a statisztikába. Ez némileg torzíthatja az eredményt, de figyelembe véve, hogy csak jegyirodában történhetett meg a törlés, a hatás elhanyagolható.[3]

Szállítási mód dimenzió

A dimenzióba a SZALLITAS tábla azonosítója kerül, a nevet pedig itt is külön surrogate objektumba helyezzük el. Amely szállításmódhoz nem tartozik tényadat, azt kihagyjuk.

Jegy vagy bérlet dimenzió

Két konstans érték, a „Jegy” és a „Bérlet”, melyeket a kezdőbetűvel jelölünk, és surrogate objektumba helyezzük el a mengevezést.

Árkategória dimenzió

Az árkategóriák esetében két választás áll előttünk, az egyik szerint a tényleges eladási ár szerint soroljuk kategóriákba a jegyeket, ekkor azonban szembekerülünk azzal a problémával, hogy egy bérlet és egy rendezvény jegyárai nehezen összevethetőek, akár nagyságrendi különbség is lehet közöttük, másrészt a rendezvények árai is nagy szórást mutathatnak, hiszen egy rendszeres színházi előadás lényegesen olcsóbb lehet, mint például egy népszerű zenekar koncertje. A rendezvényszervezők viszont eleve kategóriákba sorolják a jegyeket, elhelyezkedésük, és az esetlegesen hozzájuk kapcsolódó extra szolgáltatások alapján. Ez a besorolás sajnos nem egységes, van partner, aki 12 és van aki 6 kategóriába sorolja a jegyeit, ráadásul ezen kategóriák száma akár rendezvényről rendezvényre is változhat. A cégvezetéssel való konzultációt követően a rendezvényszervezők adott rendezvényre érvényes kategóriabeosztása végül három kategóriába kerül besorolásra. A dimenzióba a kategória sorszáma kerül, és egy surrogate objektum segítségével kötjük hozzá a kategória megnevezését.

3 Használhatnánk az IP-címet, mint iránymutató adatot a hely megállapítására, ez viszont jelentős erőforrás befektetést igényelne, és messze meghaladná a jelen szakdolgozat kereteit.

Műfaj dimenzió

A dimenzió alapja a MUFAJ tábla, a korábbiakhoz hasonlóan itt is az azonosító kerül a dimenzióba, a név pedig egy magyarázó objektumba. Csak azok kerülnek a dimenzióba, melyekhez tényadat is kapcsolható.

Forgalmi adatok

A központi tábla jelen esetben az ELADAS tábla, melynél külön figyelmet kell fordítani arra, hogy ezek a táblák a lokális adatbázisokban bírnak adattartalommal, tehát nincs igazi központi tábla, csak egy táblahalmaz.

Tényadatok

Az összeg az ELADAS_ELEM táblából kerül kiszámításra. Az összegben nem vesszük figyelembe a szállítási díjat, csak a jegyek árát. A rendezvényszervezők bizonyos esetekben vissza is vesznek jegyeket, ami az internetes értékesítés során nem fordulhat elő. Ezeket a visszavételeket beleszámítjuk az összesített forgalomba, hiszen a visszavett jegyeket általában újra eladják, így a kihagyásuk jelentős torzítást okozhatna.

Idő dimenzió

Alapját az eladás ideje (ELADAS.IDOPONT) képezi, valódi hónap alapú dimenzió. Az OLTP adatbázis ezen mezője nem tartalmaz hibákat, lévén a rekord létrejöttekor automatikusan töltődik ki, és NOT NULL megszorítás van rajta.

Rendezvényszervező

Teljes mértékben azonos az Internetes vásárlások adatkockánál leírt tulajdonos dimenzióval.

Forgalmi adatok dimenzió

Konstans elemeket tartalmaz, mégpedig az általános tervezési részben leírtakat. Ezek közül a Jegyirodai és az Internetes értékesítés elemekhez tartozó lapokat töltjük fel tényadatokkal, a többi pedig modell alapú számítás eredményeként keletkezik. A dimenzióban célszerű rövidítéseket alkalmazni, és külön magyarázni azokat.

Sikeres és sikertelen tranzakciók

Ehhez az adatkockához minden adatot a WEB_ELADAS táblából vehetünk, hiszen mind a vásárlás sikerességére, mind pedig az esetleges hibára vonatkozó adatokat tartalmazza.

Tényadatok

A tranzakciók száma, tehát gyakorlatilag a WEB_ELADAS rekordok száma

Idő dimenzió

Teljes mértékben azonos a Forgalmi adatok kocka idő dimenziójával

Hibakódok

A hibakódokat öt kategóriába soroljuk, úgymint:

- Sikeres tranzakciók

- Vevőnek felróható hibák, például a kártyaadatok hibás megadása
- Banki rendszer hibái, alapvetően a rendszer- és kommunikációs hibák
- Kártya problémák, például fedezethiány vagy internetes vásárlásra nem alkalmas kártya használata.
- Egyéb hibák, például a vásárló visszautasította a tranzakciót a kártyaadatok megadásánál, illetve ide kerülnek az időkeret lejárásakor automatikusan lezárult vásárlások is.

A dimenzióban csak csoportkódokat tárolunk, és egy surrogate objektummal magyarázzuk.

5.3.3 Transzformáció

A transzformációk kapcsán ismét kövessük az adatkiválasztásnál meghatározott módot, azzal a különbséggel, hogy csak azokkal az adatokkal foglalkozunk, például a műfaj esetében az adatok transzformáció nélkül kerülhetnek át a relációs adatbázisból az adattárházba.

Internetes eladások

Tényadatok

A tényadatokat napi szinten összesítjük.

Idő dimenzió

Miután a dimenzió alapegysége a nap, a nyolc karakter éppen elegendő, a dátumok kódolására a következő szabványokat vezetjük be:

- Nap : [év][hónap][nap] formátumban, a nap és hónap két, míg az év négy karakteren tárolva.
- Harmadhónap: [év][hónap]H[harmad sorszáma] formátum szerint, az évet négy, a hónapot két, a harmad sorszámát pedig egy karakterrel megjelenítve
- Hónap : [év]_[hónap] formátum szerint, a hónapot két, az évet pedig négy karakterrel megjelenítve
- Negyedév : Q[negyedév]_[év] minta alapján, ahol a negyedév egy az év pedig négy karakteren jelenik meg
- Év : négy karakteres formában

A fenti sablonok alapján a hierarchia létrehozása és a jelentések értelmezése is könnyebb.

Hely dimenzió

Ha van releváns cím rekord, akkor az ország meghatározása egyértelmű, ha nincs, akkor az adatkiválasztásnál leírt metódust követjük. Amennyiben az ország Magyarország, akkor ha a város tartalmazza a Budapest szót, akkor fővárosinak tekintjük, egyébként vidékinek. Minden nem magyarországi vásárló a külföld főkategóriába kerül.

Árkategória

A partnerek által megadott kategóriákat egyszerű felosztás segítségével osztjuk be az általunk tervezett három árkategóriába. Megtartjuk a kategóriáknak azt a jellemzőjét, hogy a kisebb szám drágább jegyet, és ezzel jobb helyet jelent.

Forgalmi adatok

Tényadatok

A forgalmi adatokat havi szinten összesítjük.

Idő dimenzió

Az eladási időpontot hónappá konvertáljuk, és így kerül be az értékek közé.

Forgalmi adatok dimenzió

A „J" típusú eladások adatai a Jegyirodai, míg az „I" típusúak az Internetes vásárlásokat tartalmazó elem lapjára kerül.

Sikeres és sikertelen vásárlások

Hibakódok dimenzió

A hibakódokat külön kódtábla segítségével soroljuk kategóriákba.

5.3.4 Betöltés

Mivel az adatok alapvetően nem egy adatbázisból származnak, illetve az egyszerűsítéseket tekintve nem egy sémából, célszerű a betöltendő adatokat egy ideiglenes, ROLAP jellegű struktúrába betölteni, majd onnan áttölteni a MOLAP adattárházba.

Az áttöltés során a dimenziókat. és az esetlegesen hozzájuk tartozó magyarázó adatokat és hierarchiákat visszük át elsőként, és a tényadatokat csak ezután, MATCH kapcsolóval importáljuk.

A hibakódok csoportba sorolását tartalmazó kódtáblát szövegfájlból importáljuk az átmeneti ROLAP struktúrába.

A frissítésekre inkrementális modellt alkalmazunk, mely adott időszakokat importál. A frissítésekre jellemzően évente kétszer kerül sor, egyszer január első felében, amikor az évadkezdés (augusztus 1) és december 31 közötti, és egyszer az évad lezárását követően, augusztus elején, amikor január 1 és az évadzárás (július 31) közötti adatokat töltjük fel. A frissítés a következő metodika szerint történik: egy adatbázis job-ként futó tárolt eljárás adott időben meghatározza a frissítendő időszakot, és azt beírja egy journal jellegű táblába új státusszal, majd áttölti az új adatokat a ROLAP struktúrába, és az időszak rekord státuszát áttölthetőre állítja. Egy másik automatikusan futó folyamat a ROLAP struktúrából átviszi az adatokat a MOLAP adattárházba és a journal tábla megfelelő rekordjának státuszát áttöltöttre állítja.[4]

4 E frissítési folyamat implementálása meghaladja a jelen szakdolgozat kereteit, így arra nem tudok kitérni.

5.4 MOLAP definíció

Első lépésként egy munkaterületet kell létrehoznunk, célszerű ehhez egy külön felhasználót és táblaterületet definiálnunk. Ezután csatlakozzunk az Analytic Workspace Manager segítségével az adatbázishoz, és nyissuk meg az Oracle OLAP Worksheet programot. Ezután már a következő paranccsal létrehozhatjuk a munkaterületet:

```
aw create ATH segmentsize 20M
```

Ha a megfelelő táblaterületet adtuk meg a felhasználó alapértelmezett táblaterének, akkor azt már nem kell itt megadnunk. Ezzel létrejött a munkaterület, és a felcsatolása is megtörtént, mégpedig írható módban. A biztonság kedvéért célszerű a munkaterületet lecsatolni, és exclusive módban újra csatlakozni:

```
aw detach ATH;
aw attach ATH rwx
```

5.4.1 Dimenzióobjektumok létrehozása

A surrogate objektumok neveinél a következő szabályt követjük: az objektum neve két részből áll, ponttal elválasztva, az első rész a kapcsolt dimenzió neve, míg a második a jellemző neve, például tulaj.nev .

Internetes eladások

Minden csak ezeknél az adatkockáknál használatos dimenziót „i_” karaktersorozattal kezdünk.

Idő

A kódolási szabványainkat úgy alakítottuk ki, hogy nyolc karakter elegendő legyen bármely dátum tárolására. A karbantartás szempontjából jobb ha nem egyetlen dimenziót hozunk létre, hanem külön egyet a napoknak, egyet a hónapoknak, egyet a negyedéveknek, és egyet az éveknek, majd ezekből összefűzéssel hozzuk létre a tényleges idő dimenziót, majd ezen definiáljuk a hierarchiát.

```
define i_nap dimension id;
ld 'Nap dimanzió';
define i_harmadhonap dimension id;
ld 'Harmadhónapokat tartalmazó dimenzió';
define i_honap dimension id;
ld 'Hónap dimenzió';
define i_negyedev dimension id;
ld 'Negyedév dimenzió';
define i_ev dimension id;
ld 'Év dimenzió';
define i_ido dimension concat -
      (i_nap i_harmadhonap i_honap i_negyedev i_ev);
ld 'Idő dimenzió';
define i_ido.rel relation i_ido<i_ido>;
ld 'Idő hierarchiát definiáló reláció'
```

Tulajdonos

A dimenzió típusa ID, melyben az azonosítót tároljuk, kapcsolódik hozzá egy 300 byte

hosszúságú teljes és egy 50 byte-os rövid név. Mivel több adatkockában, modellben is szerepet játszik, „c_” karakterekkel kezdjük a nevét.

```
define c_tulaj dimension id;
ld 'A rendezvénygazda azonosítóját tartalmazó dimenzió';
define c_tulaj.nev surrogate c_tulaj text w 300;
ld 'A rendezvénygazda neve';
define c_tulaj.rov_nev surrogate c_tulaj text w 50;
ld 'A rendezvénygazda rövid neve';
define c_tulaj.rel relation c_tulaj <c_tulaj>;
ld 'Rendezvénygazdáktól független jelentések készítéséhez'
```

Előrevásárlás

A dimenzió típusa legyen itt is ID, tartalma pedig egész szám, s kapcsolódjon hozzá egy rövid, 50 byte hosszúságú név.

```
define i_elotte dimension id;
ld 'A vásárlás a rendezvény előtt mennyivel történt hétben';
define i_elotte.nev surrogate i_elotte text w 50;
ld ' A vásárlás a rendezvény előtt mennyivel történt, név';
define i_elotte.rel relation i_elotte <i_elotte>;
ld 'Előrevásárlás összesítésére használható'
```

Hely

Az egyes hierarchia szinteknek megfelelő elemeket három különböző dimenzióban tároljuk.

```
define i_orszag dimension text w 100;
ld 'Ország dimenzió';
define i_videk_fovaros dimension text w 100;
ld 'Vidék – Főváros felosztást tartalmazó dimenzió';
define i_hazai_kulfoldi dimension text w 100;
ld 'Hazai és külföldi felosztást tartalmazó dimenzió';
define i_orszag.kod surrogate i_orszag id;
ld 'Ország kódja';
define i_videk_fovaros.kod surrogate i_videk_fovaros id;
ld 'Vidék és főváros kódja';
define i_hazai_kulfoldi.kod surrogate i_hazai_kulfoldi id;
ld 'Hazai és külföldi kódja';
define i_hely dimension concat -
     (i_orszag, i_videk_fovaros, i_hazai_kulfoldi);
ld 'Hely dimenzió';
define i_hely.rel relation i_hely <i_hely>;
ld 'Hely hierarchiát leíró reláció'
```

Szállítási mód

```
define i_szall dimension id;
ld 'Szállításmód dimenzió';
define i_szall.nev surrogate i_szall text w 100;
ld 'Szállításmód leírása';
define i_szall.rel relation i_szall <i_szall>;
ld 'Szállítási módok összesítéséhez használható hierarchia'
```

Jegy és bérlet

```
define i_jegy_berlet dimension id;
ld 'Jegy és bérlet kódját tartalmazó dimenzió';
define i_jegy_berlet.nev surrogate i_jegy_berlet text w 100;
ld 'Jegy és bérlet megnevezése';
define i_jegy_berlet.rel relation -
     i_jegy_berlet <i_jegy_berlet>;
ld 'Jegy és bérlet összesítést lehetővé tevő reláció'
```

Árkategória

```
define i_arkat dimension integer;
ld 'Árkategória azonosítóját tartalmazó dimenzió';
define i_arkat.nev surrogate i_arkat text w 50;
ld 'Árkategória megnevezése';
define i_arkat.rel relation i_arkat <i_arkat>;
ld 'Árkategória összesítést biztosító reláció'
```

Műfaj

```
define i_mufaj dimension integer;
ld 'Műfaj azonosítóját tartalmazó dimenzió';
define i_mufaj.nev surrogate i_mufaj text w 100;
ld 'Műfaj megnevezése';
define i_mufaj.rel relation i_mufaj <i_mufaj>;
ld 'Műfaj összesítést lehetővé tevő hierarchia';
```

Forgalmi adatok

Minden csak erre az adatkockára vonatkozó objektum neve „f_” karaktersorozattal kezdődik. A rendezvényszervezőket tartalmazó dimenzió már létezik (c_tulaj), így azt már nem kell létrehoznunk.

Idő dimenzió

Valódi idő dimenzió, hónap felbontással, negyedéves éves és évad szerinti összegzést lehetővé téve. Mivel ezeket a dimenziókat több adatkockában, modellbem is használni fogjuk a nevük „c_” karakterekkel kezdődik

```
define c_honap dimension month;
vnf '<yyyy>. <mtextl>';
ld 'Hónap dimenzió';
define c_negyedev dimension quarter;
vnf '<ffff> <p>. negyedév';
ld 'Negyedév dimenzió';
define c_ev dimension year;
vnf '<yyyy>';
ld 'Év dimenzió';
define c_evad dimension year beginning 'augusztus 01';
vnf '<yyyy> évad';
ld 'Évad dimenzió'
```

Mivel valódi idő dimenziókról van szó, az egyes szintek között a reláció automatikusan létrejön, de hierarchia nem definiálható felettük.

Forgalmi adatok

```
define f_adat dimension id;
ld 'Forgalmi adatsorok kódja';
define f_adat.nev surrogate f_adat text w 100;
ld 'Forgalmi adatsorok leírása'
```

Sikeres és sikertelen tranzakciók

Az idő dimenziót már a Forgalmi adatok kapcsán létrehoztuk, az itt szükséges hibakód dimenzió nevét pedig „t_” karakterekkel kezdjük.

Hibakódok

A könnyebb elemzés miatt létrehozunk egy hierarchiát is.

```
define t_hibacsoport dimension id;
ld 'Hibacsoportok dimenzió';
define t_hibacsoport.nev surrogate t_hibacsoport text w 50;
ld 'Hibacsoportok neve';
define t_sikeres_sikertelen dimension id;
ld 'Sikeres és sikertelen csoportok';
define t_sikeres_sikertelen.nev surrogate -
     t_sikeres_sikertelen text w 50;
ld 'Sikeres és sikertelen megnevezés';
define t_eredmeny dimension concat -
     ( t_hibacsoport t_sikeres_sikertelen );
ld 'Tranzakció eredménye dimenzió';
define t_eredmeny.rel relation t_eredmeny <t_eredmeny>;
ld 'Tranzakció eredménye hierarchiát leíró reláció'
```

5.4.2 Adatkockák és modellek létrehozása

Internetes vásárlások

Összeg

```
define i_osszeg variable number(12,2) -
     < i_ido c_tulaj i_elotte i_hely i_szall -
     i_jegy_berlet i_arkat i_mufaj>;
ld 'Internetes vásárlások összege'
```

Eladott jegyek

```
define i_jegyszam variable integer -
     < i_ido c_tulaj i_elotte i_hely i_szall -
     i_jegy_berlet i_arkat i_mufaj>;
ld 'Internetes vásárlások során eladott jegyek száma'
```

Forgalmi adatok

Ez egy modell, így szükséges az f_adat dimenzió feltöltése is.

```
define f_adatok variable number(12,2) -
     < c_honap c_tulaj f_adat>;
```

```
ld 'Forgalmi adatokat tartalmazó adatkocka';
maintain f_adat add 'J' 'I' 'Osszes' 'Iarany' 'Jut' 'Netto';
f_adat.nev(f_adat 'J') = 'Jegyiroda';
f_adat.nev(f_adat 'I') = 'Internet';
f_adat.nev(f_adat 'Osszes') = 'Összesen';
f_adat.nev(f_adat 'Iarany') = 'Internet aránya';
f_adat.nev(f_adat 'Jut') = 'Jutalék';
f_adat.nev(f_adat 'Netto') =  'Nettó árrés';
define f_adatok.model model;
ld 'Forgalmi adatok számítását leíró modell';
```

Ezután meg kell nyitnunk a modellt szerkesztésre:

```
edit model f_adatok.model
```

A megjelenő szerkesztő ablakba kell beírnunk a következő sorokat:

```
dimension f_adat
Osszes = J + I
Iarany = I / (if NAFill(Osszes,0) eq 0 then 1 else Osszes)
Jut = I * .1
Netto = Jut / 1.2
```

Mentsük el a kódot, majd fordítsuk le a modellt, és alkalmazzuk is az adatkockára:

```
compile f_adatok.model;
f_adatok.model f_adatok
```

Sikeres és sikertelen tranzakciók

```
define t_adatok variable integer < c_honap t_eredmeny>;
ld 'Tranzakciók sikerességének adatai'
```

5.4.3 Aggregációs szabályok létrehozása

Aggregációs szabályokra mindhárom adatkocka esetén szükségünk van. A forgalmi adatokat és a sikeres és sikertelen tranzakciókat tartalmazó kockák esetén viszont az idő dimenzió felett, lévén valódi dátum alapú, nem kell, és nem is lehet aggregációt végezni.

Internetes vásárlások

Itt az összes dimenzió felett kell aggregációt végezni.

```
define i_adatok.agg aggmap;
ld 'Internetes eladások adatainak összesítési szabálya'
```

Ezután nyissuk meg szerkesztésre.

```
edit aggmap i_adatok.agg
```

A megnyíló szerkesztőablakba írjuk be a következő sorokat.

```
relation i_ido.rel
relation c_tulaj.rel
relation i_elotte.rel
relation i_hely.rel
relation i_szall.rel
```

```
relation i_jegy_berlet.rel
relation i_arkat.rel
relation i_mufaj.rel
```

Mentsük el a kódot, zárjuk be a szerkesztőt és fordítsuk le a szabályt.

```
compile i_adatok.agg
```

5.4.4 Forgalmi adatok

Itt egyedül a rendezvényszervezők dimenzió felett kell aggregálni, az idő felett nem is lehet, a forgalmi adatok felett pedig értelmetlen, hiszen számított sorokat tartalmaz.

```
define f_adatok.agg aggmap;
ld 'Forgalmi adatok aggregációs szabálya'
```

Ezt is nyissuk meg az EDIT paranccsal, ér írjuk be a következő sort:

```
relation c_tulaj.rel
```

Mentés és a szerkesztőablak bezárása után ezt is le kell fordítani.

```
compile f_adatok.agg
```

5.4.5 Sikeres és sikertelen tranzakciók

Itt a hibakódokat tartalmazó dimenzió felett kell aggregációt végezni.

```
define t_adatok.agg aggmap;
ld 'Sikeres és sikertelen tranzakciók összesítéséhez'
```

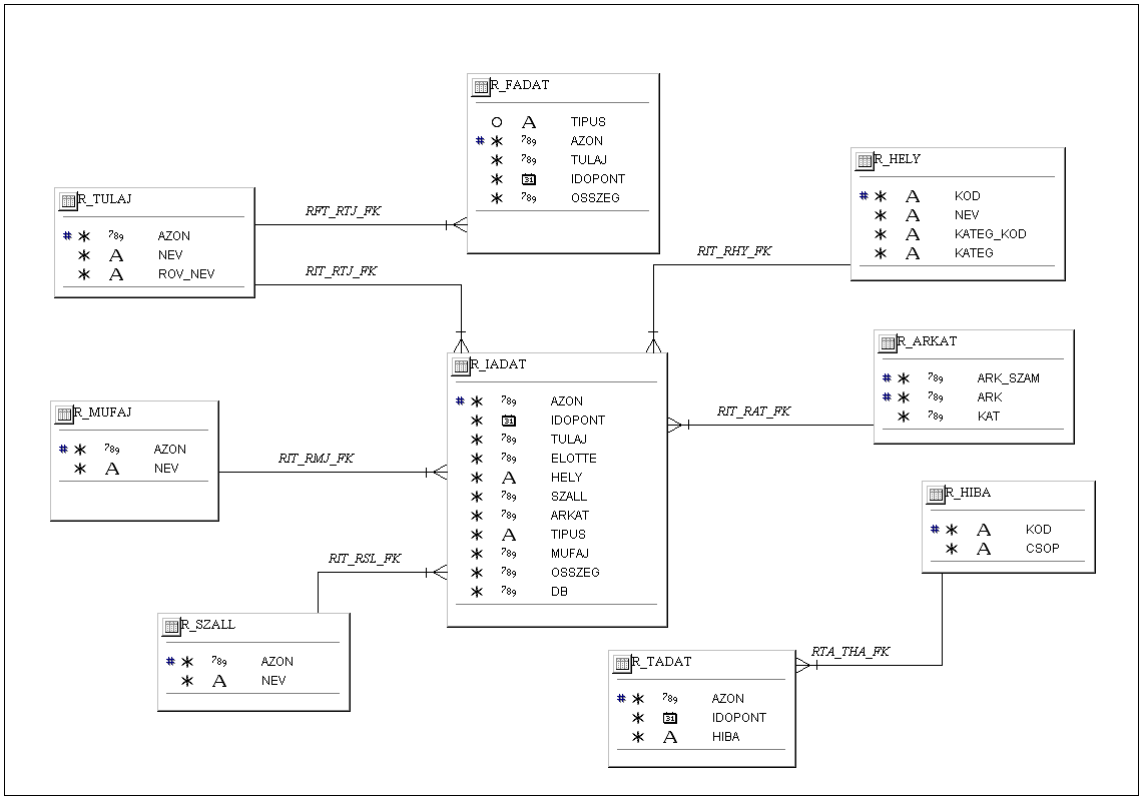
A szerkesztőablakba a következő sort kell beírni:

```
relation t_eredmeny.rel
```

Mentés és bezárás után fordítsuk le. Ezzel minden szükséges objektumot definiáltunk.

5.5 MOLAP adatfeltöltés

Az adatfeltöltéshez először az átmeneti ROLAP struktúrát kell létrehoznunk, ez elengedhetetlen az összesítések elvégzéséhez, s ha ez megvan, még mindig átmeneti táblákat kell használnunk, hogy az egyes lekérdezések, melyek az OLTP rendszerből kinyerik az adattárház számára szükséges adatokat, vége időn belül lefussanak. Például az internetes eladások adatait kigyűjtő lekérdezés az összesítések, és kategóriákba soro-



4. ábra: Az átmeneti tárolóként használt ROLAP séma modellje

lás miatt annyira bonyolult, hogy a legjobb optimalizáció mellett sem fut le 10 percen belül, egy félévnyi adatmennyiséget figyelembe véve. Ha viszont ideiglenes táblákat használunk, akkor ez az idő kevesebb, mint negyedére csökkenthető.

Miután a ROLAP struktúra tábláit feltöltöttük adatokkal, megkezdhetjük a MOLAP objektumok feltöltését. Erre a célra legegyszerűbb, ha kis SQL alapú programokat írunk, és azokat egy wrapper (burkoló) programból hívjuk meg. Minden ilyen kis programot az eltölteni kívánt alapobjektum nevéhez fűzött „_l_prog” karaktersorozattal nevezünk el. Ismét fel szeretném hívni a figyelmet arra, hogy egy program, különösen egy frissen megírt eljárás futtatása előtt menteni kell a korábbi munkákat, egy hibás program megállítása ugyanis csak kilépéssel oldható meg, ami a nem mentett adatok végleges elvesztését jelenti.

Dimenziók feltöltése

Minden dimenziókat feltöltő program alkalmas mind a kezdeti adatok, mind pedig a későbbi frissítések elvégzésére. Ez ilyen eljárások esetén fontos szempont kell, hogy legyen.

I_IDO

Az i_ido dimenziót a részdimenziók feltöltésével kezeljük. Az adatok a R_IADAT tábla időpont oszlopából származnak. Első lépésként definiáljuk a programot

```
define i_ido.l_prog program;
ld 'Internetes eladások idő dimenzióját tölti fel'
```

Meg kell nyitni a programot szerkesztésre.

```
edit i_ido.l_prog
```

A megjelenő szerkesztőablakba kell a következő kódot beírni.

```
"Deklaráljuk a szükséges változókat
variable d date
variable nap id
variable hh id
variable h id
variable q id
variable yr id
variable temp shortint
variable s text

"Létrehozzuk az adatfeltöltéshez szükséges kurzort
"Azt a dátumot, amelynél régebbi adatok nem szükségesek
"a d változóban várja
SQL DECLARE cr_ido CURSOR FOR -
     select distinct idopont -
          from r_iadat where idopont > :d

"Elmentjük a jelenlegi állapotot
pushlevel 'Start'
push i_ido

"Vegyük a legutolsó napot
limit i_nap to last 1

"Ha nincs, akkor egy alap dátumot veszünk, aminél biztos
"nincs korábbi adat.
if statlen(i_nap) lt 1
  then d = to_date('19000101', 'yyyymmdd')
  else d = to_date(i_nap, 'yyyymmdd')

"Megnyitjuk a kurzort
SQL OPEN cr_ido

"Lekérjük az első rekordot, ha van
SQL FETCH cr_ido INTO :d

"Addig folytatjuk ciklusosan, amíg van adat
while SQLCODE eq 0
do

  "Meghatározzuk a nap dimenzió értékét
  nap = to_char(d, 'yyyymmdd')

  "Meghatározzuk a harmadhónap dimenzió értékét
  temp = to_char(d, 'dd')
  temp = intpart(temp / 10)
```

```
    if temp ne 3
      then temp = temp + 1
    hh = joinchars( to_char(d, 'yyyymm') 'S' temp)

    "Meghatározzuk a hónap dimenzió értékét
    h = to_char(d, 'yyyy_mm')

    "Meghatározzuk az év dimenzió értékét
    yr = to_char(d, 'yyyy')

    "Meghatározzuk a negyedév dimenzió értékét
    temp = to_char(d, 'mm')
    temp = intpart( (temp -1) / 3) + 1
    q = joinchars( 'Q' temp '_' yr )

    "Az idő dimenziót alapstátuszba állítjuk
    limit i_ido to all

    "Hozzáadjuk a nap dimenzióhoz az új értéket
    maintain i_nap add nap

    "A többi dimenzióhoz csak akkor adunk értéket,
    "ha szükséges.
    if not isvalue(i_harmadhonap, hh)
      then maintain i_harmadhonap add hh

    if not isvalue(i_honap, h)
      then maintain i_honap add h

    if not isvalue(i_negyedev q)
      then maintain i_negyedev add q

    if not isvalue(i_ev yr)
      then maintain i_ev add yr

    "Beállítjuk a hierarchiákat
    limit i_ido to <i_nap: nap>
    i_ido.rel = i_ido(i_harmadhonap hh)

    limit i_ido to <i_harmadhonap: hh>
    i_ido.rel = i_ido(i_honap h)

    limit i_ido to <i_honap: h>
    i_ido.rel = i_ido(i_negyedev q)

    limit i_ido to <i_negyedev: q>
    i_ido.rel = i_ido(i_ev yr)

    "Lekérjük a következő rekordot
    SQL FETCH cr_ido INTO :d

  doend

  "Felszabadítjuk a lefoglalt kurzorterületet
  SQL CLEANUP;

  "Visszaállítjuk a ketdeti állapotot
  poplevel 'Start'
```

Ezután le kell fordítanunk a programot, és akár meg is hívhatnánk.

```
compile i_ido.l_prog
```

C_TULAJ

Adattartalma az R_TULAJ táblából származik, s elég ritkán kerül bele új adat. Definiáljuk a C_TULAJ.L_PROG programot, majd nyissuk meg szerkesztésre. A szerkesztőablakba a következő kódot írjuk:

```
"Elmentjük a jelenlegi állapotot
pushlevel 'Start'
push c_tulaj

"Deklaráljuk a forrás kurzort
SQL DECLARE cr_tulaj CURSOR FOR -
  select azon, nev, rov_nev from r_tulaj order by azon

"A minden rendezvényszervező elemet'0'-es értékkel fogjuk
"jelölni
"Ha még nem létezik, akkor hozzáadjuk
limit c_tulaj to all
if not isvalue(c_tulaj '0')
  then do
    show 'nincs 0'
    maintain c_tulaj add '0'
    limit c_tulaj to '0'
    c_tulaj.nev = 'Minden rendezvénygazda'
    c_tulaj.rov_nev = 'Mind'
    limit c_tulaj to all
  doend

"Importáljuk az adatokat
SQL OPEN cr_tulaj
SQL FETCH cr_tulaj LOOP INTO :APPEND c_tulaj -
                             :ASSIGN c_tulaj.nev -
                             :ASSIGN c_tulaj.rov_nev

limit c_tulaj remove '0'
c_tulaj.rel = '0'

poplevel 'Start'
```

Fordítás után már használható is. A továbbiakban csak azokkal a dimenziókkal foglalkozunk, melyek kezelése eltér a C_TULAJ kezelésétől.

I_HELY

A hely dimenzió kezelése azért különleges, mert a kategória szerint más-más dimenzióba kerül az érték. Az I_HELY.L_PROG forrása a következő, kommenteket csak az egyedi részekhez tettem:

```
variable o text
variable ok id
variable k text
variable kk id

pushlevel 'I_HELY start'
push i_orszag
push i_videk_fovaros
push i_hazai_kulfoldi
push i_hely
```

```
SQL DECLARE cr_hely CURSOR FOR -
     select kod, nev, kateg_kod, kateg from r_hely -
          where kod in (select hely from r_iadat)

SQL OPEN cr_hely
SQL FETCH cr_hely INTO :ok :o :kk :k

while SQLCODE eq 0
do
  limit i_orszag to all
  limit i_videk_fovaros to all
  limit i_hazai_kulfoldi to all
  limit i_hely to all
  if not isvalue( i_hazai_kulfoldi k)
    then do
      maintain i_hazai_kulfoldi add k
      limit i_hazai_kulfoldi to k
      i_hazai_kulfoldi.kod = kk
    doend

  "Ha külföldi a kategória, akkor az I_ORSZAG dimenzióba
  "kerül
  if kk eq 'kulf'
    then do
      if not isvalue(i_orszag o)
        then do
          maintain i_orszag add o
          limit i_orszag to o
          i_orszag.kod = ok
          i_hely.rel(i_orszag o) = -
                i_hely(i_hazai_kulfoldi k)
        doend
     doend
     "Egyébként az I_VIDEK_FOVAROS dimenzióba
     else do
       if not isvalue(i_videk_fovaros o)
         then do
           maintain i_videk_fovaros add o
           limit i_videk_fovaros to o
           i_videk_fovaros.kod = ok
           i_hely.rel(i_videk_fovaros o) = -
                 i_hely(i_hazai_kulfoldi k)
         doend
     doend
doend

SQL CLEANUP

poplevel 'I_HELY start'
```

I_JEGY_BERLET

E dimenzió feltöltése rendhagyó, mivel nem érdemes SQL alapú programot írni.

```
pushlevel 'JEGY_BERLET start'
push i_jegy_berlet
```

```
limit i_szallmod to all

"Minden típus
if not isvalue(i_jegy_berlet 'A')
  then do
    maintain i_jegy_berlet add 'A'
    limit i_jegy_berlet to 'A'
    i_jegy_berlet.nev = 'Minden típus'
    limit i_jegy_berlet to all
  doend

"Jegy
if not isvalue(i_jegy_berlet 'J')
  then do
    maintain i_jegy_berlet add 'J'
    limit i_jegy_berlet to 'J'
    i_jegy_berlet.nev = 'Jegy'
    i_jegy_berlet.rel = 'A'
    limit i_jegy_berlet to all
  doend

"Bérlet
if not isvalue(i_jegy_berlet 'B')
  then do
    maintain i_jegy_berlet add 'B'
    limit i_jegy_berlet to 'B'
    i_jegy_berlet.nev = 'Bérlet'
    i_jegy_berlet.rel = 'A'
  doend

poplevel 'JEGY_BERLET start'
```

I_ARKAT

Itt ismét az a különlegesség, hogy nem kell SQL utasítás, ráadásul a kialakult kategóriák miatt egy ciklussal is megoldható a feltöltés:

```
variable i integer
variable s id
pushlevel 'ARKAT start'
push i_arkat

limit i_arkat to all

if not isvalue(i_arkat 'A')
  then do
    maintain i_arkat add 'A'
    limit i_arkat to 'A'
    i_arkat.nev = 'Minden kategória'
    limit i_arkat to all
  doend

i = 1

"Három kategóriára van szükségünk
while i le 3
do
  s = to_char(i)
```

```
  if not isvalue(i_arkat s)
    then do
      maintain i_arkat add s
      limit i_arkat to s
      i_arkat.nev = joinchars( s '. kategória')
      i_arkat.rel = 'A'
      limit i_arkat to all
    doend
  i = i + 1
doend

poplevel 'ARKAT start'
```

C_HONAP

A C_HOANP.L_PROG valójában nem csak hónapokat, hanem a valódi idő dimenzió minden elemét feltölti.

```
variable maxd date
variable mind date

SQL DECLARE cr_ido CURSOR FOR -
     select min(idopont) k, max(idopont) v from r_fadat

pushlevel 'HONAP start'
push c_honap
push c_negyedev
push c_ev
push c_evad

SQL OPEN cr_ido
SQL FETCH cr_ido INTO :mind :maxd

limit c_honap to last 1
if statlen( c_honap ) lt 1
  then do
   "Ha még nincs elem a dimenziuóban,
   "akkor a tejes intervallumot hozzáadjuk
    maintain c_honap add mind maxd
    maintain c_negyedev add mind maxd
    maintain c_ev add mind maxd
    maintain c_evad add mind maxd
  doend
  else do
    "Ha van elem, és az utolsó kisebb, mint a max
    "dátum, akkor csak a legnagyobbat kell hozzáadni
    if maxd gt c_honap
      then do
        maintain c_honap add maxd
        maintain c_negyedev add maxd
        maintain c_ev add maxd
        maintain c_evad add maxd
      doend
  doend

poplevel 'HONAP start'
```

A T_EREDMENY dimenzió feltöltéséhez ismét a már ismertetett SQL ciklusos megoldást választjuk. Már csak az a programot kell megírnunk, mely mindezeket meghívja, ezt az eljárást nevezzük LOAD.DIM-nek

```
show 'I_IDO dimenzió'
I_IDO.L_PROG

show 'C_TULAJ dimenzió'
C_TULAJ.L_PROG

show 'I_ELOTTE dimenzió'
I_ELOTTE.L_PROG

show 'I_HELY dimenzió'
I_HELY.L_PROG

show 'I_SZALL dimenzió'
I_SZALL.L_PROG

show 'I_JEGY_BERLET dimenzió'
I_JEGY_BERLET.L_PROG

show 'I_ARKAT dimenzió'
I_ARKAT.L_PROG

show 'I_MUFAJ dimenzió'
I_MUFAJ.L_PROG

show 'C_HONAP dimenzió'
C_HONAP.L_PROG

show 'T_EREDMENY dimenzió'
T_EREDMENY.L_PROG

SQL CLEANUP

show 'Dimenziók betöltése kész'
```

Adatok betöltése

Ezeket a programokat egyszerűen úgy nevezzük el, hogy a kocka neve után írjuk a „.LOAD” karaktersorozatot. Fontos, hogy a betöltés után aggregálni is kell. Minden feltöltési eljárás a kezdeti adatfeltöltést fogja megvalósítani, viszont kialakításuk olyan, hogy minimális átalakítás után alkalmasak legyenek frissítésre is, és csak a tényadatok feltöltését végzik, mivel a dimenziókat már korábban feltöltöttük.

Internetes eladások

Mindkét, az internetes eladásokhoz kapcsolódó adattáblát egyazon eljárással töltjük fel, emiatt a programnak az I_ADAT.LOAD nevet adjuk. Az adatok a R_IADAT táblából kerülnek betöltésre, mely már eleve aggregált adatokat tartalmaz.

```
vrb ido id
vrb tul id
vrb el  id
vrb he  text
vrb sz  id
vrb a   id
vrb t   id
```

```
vrb m   id
vrb am  number
vrb db  integer
vrb kk  id
vrb c   integer
SQL DECLARE cr_iadat CURSOR FOR -
      select to_char(idopont, 'yyyymmdd'), tulaj, elotte, -
             rh.nev, szall, arkat, tipus, mufaj, osszeg, -
             db, kateg_kod -
      from r_iadat ri, r_hely rh -
      where ri.hely = rh.kod

"Mentsünk el mindent
pushlevel 'I_ADATOK'
push C_EV C_EVAD C_HONAP C_NEGYEDEV C_TULAJ -
     F_ADAT I_ARKAT I_ELOTTE I_EV I_HARMADHONAP -
     I_HAZAI_KULFOLDI I_HELY I_HONAP I_IDO -
     I_JEGY_BERLET I_MUFAJ I_NAP I_NEGYEDEV -
     I_ORSZAG I_SZALL I_SZALLMOD I_VIDEK_FOVAROS -
     T_EREDMENY T_HIBACSOPORT T_SIKERES_SIKERTELEN

ALLSTAT

SQL OPEN cr_iadat
c = 0

SQL FETCH cr_iadat -
      INTO :ido :tul :el :he :sz :a :t :m :am :db :kk

while SQLCODE eq 0
do
  limit i_ido to i_ido(i_nap ido)
  if statlen( i_ido ) ne 1
    then signal 'Nap hiba'
  limit c_tulaj to tul
  if statlen( c_tulaj ) ne 1
    then signal 'Tulaj hiba'
  limit i_elotte to el
  if statlen( i_elotte ) ne 1
    then signal 'Elotte hiba'
  if kk eq 'kulf'
    then limit i_hely to i_hely(i_orszag he)
    else limit i_hely to i_hely(i_videk_fovaros he)
  if statlen( i_hely ) ne 1
    then signal 'Hely hiba'
  limit i_szall to sz
  if statlen( i_szall ) ne 1
    then signal 'Száll hiba'
  limit i_arkat to a
  if statlen( i_arkat ) ne 1
    then signal 'Arkat hiba'
  limit i_jegy_berlet to t
  if statlen( i_jegy_berlet ) ne 1
    then signal 'Tipus hiba'
  limit i_mufaj to m
  if statlen( i_mufaj ) ne 1
```

```
      then signal 'Mufaj hiba'

    i_osszeg = am
    i_jegyszam = db

    c = c +1
    SQL FETCH cr_iadat -
          INTO :ido :tul :el :he :sz :a :t :m :am :db :kk

doend

SQL CLENUP

update; commit

allstat
aggregate i_osszeg i_jegyszam using i_adatok.agg

updte; commit

poplevel 'I_ADATOK'
```

Az aggregate parancs segítségével egyszerre több adatkockát is összesíthetünk egyazon szabály alapján. Az aggregálás kapcsán viszont hosszú, akár több órás futási időkre is fel kell készülnünk, ezért célszerű minden befejezett részfolyamat előtt menteni az adattárház tartalmát.

Forgalmi adatok

Az adatokat a R_FADAT relációs táblából importáljuk, mivel ez egy modell, így a feltöltésnél viszonylag egyszerű dolgunk van, de nem szabad megfeledkezni a modell futtatásáról. A F_ADATOK.LOAD eljárás forrása így a következőképpen alakul:

```
SQL DECLARE cr_fadat CURSOR FOR -
      select idopont, tulaj, tipus, osszeg -
      from r_fadat -
      order by idopont, tulaj, tipus

pushlevel 'F_ADATOK start'
push C_EV C_EVAD C_HONAP C_NEGYEDEV C_TULAJ -
     F_ADAT I_ARKAT I_ELOTTE I_EV I_HARMADHONAP -
     I_HAZAI_KULFOLDI I_HELY I_HONAP I_IDO -
     I_JEGY_BERLET I_MUFAJ I_NAP I_NEGYEDEV -
     I_ORSZAG I_SZALL I_SZALLMOD I_VIDEK_FOVAROS -
     T_EREDMENY T_HIBACSOPORT T_SIKERES_SIKERTELEN

ALLSTAT

SQL OPEN cr_fadat

SQL FETCH cr_fadat LOOP INTO :MATCH c_honap -
                             :MATCH c_tulaj -
                             :MATCH f_adat  -
                             :F_ADATOK

SQL CLEANUP

aggregate F_ADATOK using F_ADATOK.agg
F_ADATOK.MODEL F_ADATOK

poplevel 'F_ADATOK start'
```

Sikeres és sikertelen tranzakciók

Ennél az adatkockánál az egyetlen problémát a T_EREDMENY összetett reláció jelenti. Ennek ellenére a T_ADATOK.LOAD program forrása már nem tartalmaz újdonságokat:

```
vrb d date
vrb h integer
vrb c integer;

SQL DECLARE cr_tadat CURSOR FOR -
              select rt.idopont, rh.csop, sum(db) -
              from r_tadat rt, r_hiba rh -
              where rt.hiba = rh.kod -
              group by rt.idopont, rh.csop -
              order by rt.idopont, rh.csop

pushlevel 'T_ADATOK start'
push C_EV C_EVAD C_HONAP C_NEGYEDEV C_TULAJ F_ADAT -
     I_ARKAT I_ELOTTE I_EV I_HARMADHONAP -
     I_HAZAI_KULFOLDI I_HELY I_HONAP I_IDO -
     I_JEGY_BERLET I_MUFAJ I_NAP I_NEGYEDEV -
     I_ORSZAG I_SZALL I_SZALLMOD I_VIDEK_FOVAROS -
     T_EREDMENY T_HIBACSOPORT T_SIKERES_SIKERTELEN

ALLSTAT

SQL OPEN cr_tadat

SQL FETCH cr_tadat INTO :d :h :c
while SQLCODE eq 0
do
  limit c_honap to to_char(d, 'yyyy mon dd')
  if statlen( c_honap) ne 1
    then show 'Hiba hónap'
  limit t_eredmeny to t_eredmeny(t_hibacsoport h)
  t_adatok = c
  SQL FETCH cr_tadat INTO :d :h :c
doend

SQL CLEANUP

ALLSTAT

aggregate T_ADATOK using T_ADATOK.agg

poplevel 'T_ADATOK start'
```

A programok a futtatásával az adatfeltöltés elkészült, s ezzel az adattárház már használatra kész.

6. Elemzési lehetőségek

6.1 Általános elemzési lehetőségek

Mivel az adattárházak célja nem az adatok tárolása, hanem azok elérhetővé tétele, és az hogy segítségükkel új információhoz juthassunk. A legegyszerűbb elemzési lehetőség a jelentések készítése, amikor az adatokat táblázatos formában jelenítjük meg. Az OLAP DML szöveges jelentések készítését támogatja. A OLAP DML által támogatott elemzési lehetőség a MOLAP adatbázis adatainak megjelenítése relációs adatbázisban. Jelen fejezet ezzel a két lehetőséggel foglalkozik, viszont az egyéb elemzési lehetőségekről is kell néhány szót ejtenünk.

E két lehetőség azonban a tényleges igényeket általában nem képes kielégíteni, hiszen a vezetők mindig más szempontok alapján szeretnék az adatokat megtekinteni, és egyáltalán nem vonzó számukra az a lehetőség, hogy parancssori utasításokkal végezzenek elemzéseket. A felhasználók számára sokkal könnyebben használható, minél jobban testreszabható felület szükséges. Erre az több gyártó, így az Oracle is nyújt eszközt (Oracle Discoverer), ezek azonban speciálisan felépített adattárházakon képesek csak működni. Az Oracle igényli, hogy az adattárház úgynevezett standard formában legyen, ami azt jelenti, hogy tartalmaznia kell a saját metaadatait leíró objektumokat (szerencsére utólag is transzformálhatjuk az elkészült struktúrát ilyen szabványos formába). Ehhez az Analytic Workspace Manager „Object view” üzemmódjában a munkaterület nevén jobb gombbal klikkelve, a felugró menüből válasszuk a „Transform Analytic Workspace to Standard Form” menüpontot.

Az elemzések egy sokkal mélyebb szintjét jelenti az adatbányászat, amikor rejtett, egyszerű elemzésekkel nem feltárható információkat hozunk felszínre az adattárház adatainak elemzésével. Ehhez szintén számos eszköz áll rendelkezésünkre, mint például az Oracle Data Miner, viszont ezek is csak standard form adattárházak felett képesek működni. Ezekkel az eszközökkel olyan információk nyerhetők ki, mint például a vásárlási szokások, vagy egyes hirdetési formák hatékonysága.

Az Oracle ezen kívül gondolt azokra az adattárház építőkre, használókra is, akik nem szeretnének „dobozos” termékeket használni, vagy csak a már meglévő intra- vagy internetes portáljukon is szeretnék az adattárházból kinyerhető információkat megjeleníteni, ezért a BI keretrendszeréhez kapcsolódóan elérhetővé tett egy Java API implementációt is, mely segítségével az adattárház elemei, metaadat és adat szinten egyaránt a elérhetőek, sőt adatbányászatra is lehetőség nyílik. Ezzel tehát hogy hivatalos és támogatott kapcsolatot biztosít a multidimenzionális adattárház és egy robusztus objektumorientált programozási nyelv között, nagy szabadságot adott a felhasználóknak abban, hogy az általa biztosított OLAP technológiában rejlő lehetőségeket milyen módon és mértékben használják ki. Az Oracle saját eszközeinek egy része is ezen a technológián alapul, többek között az OLAP Worksheet is.

6.2 Jelentések készítése

6.2.1 OLAP DML lehetőségek

Az elemzések egyik lehetséges formája a jelentések készítése, ehhez az OLAP DML

rendelkezésünkre bocsájtja a REPORT parancsot. Segítségével lehetővé válik a szöveges, táblázatos megjelenítés. Igen sokrétűen paraméterezhető utasítás, paraméterei között megadhatjuk hogy melyik dimenzió szereplejen a sorokban, melyik az oszlopokban, valamint a dimenziók limitjeit is beállíthatjuk, a LIMIT parancsnál megismert limit-záradékok segítségével, de csak a TO limit-típus használható. Gyakorlatilag mindennek adható fejléc, és formátum. Természetesen a dimenziók helyett használhatunk hozzájuk kötött magyarázó adatokat is, de csak a sorok megnevezéséhez. Lehetőség van nem csak egy, hanem több változó megjelenítésére is egymás mellett, valamint az oszlopdimenziók is egymásba ágyazhatóak.

Ezeknek a jelentéseknek jelentős hátránya, hogy több mint két dimenzió esetén már nagy odafigyelést igényel a dimenziók limitjeinek beállítása, ha ugyanis valamely nem az oszlopokban és nem a sorokban szereplő dimenzió egynél több elérhető értéket tartalmaz, akkor a REPORT parancs annyi lapot fog képezni a lapoktól, ahányféleképpen ezeknek a dimenzióknak az elérhető értékei permutálhatóak. Például egy, a mintapéldánkhoz hasonló nyolcdimenziós kocka esetében, ha minden dimenzió csak két értéket tartalmaz, akkor a lapok száma 2^6, azaz 64, ez pedig nem szolgálja az áttekinthetőséget. Ezt csökkenthetjük az oszlopdimenziók egymásba ágyazásával, ekkor viszont figyelnünk kell arra, hogy egy sor nem tartalmazhat 4000 byte-nál hosszabb szöveget.

Parancs	Eredmény
BLANK *n*	n darab üres sort szúr be, az alapértelmezett: 1
HEADING oszlopleírás(ok)	Teljes és oszlopszintű feliratokat hoz létre, az itt megadott azámok nem számítanak bele az összegzésbe.
PAGE	Ha a PAHING beállítás értéke igaz, akkor oldaltörést szúr be.
ZEROTOTAL	Mind a 32 összegzési szintet nullázza minden oszlopra
ZEROTOTAL ALL *oszlop(ok)*	Mind a 32 összegzési szintet nullázza a megadott oszlopokra
ZEROTOTAL szint *oszlop(ok)*	A megadott szint összegzéseit nullázza a megadott oszlopokra, illetve ha nincs oszlop megadva, akkor mindre.

12. táblázat: Jelentés kinézetét befolyásoló parancsok

A másik lehetőség, hogy a jelentést mi magunk állítjuk elő programból, a program írása a REPORT parancs esetén is ajánlott. A mintapélda kapcsán is látni fogjuk a REPORT parancs segítségével előállított jelentés néha hibás adatokat tartalmaz, ennek esetünkben az az oka, hogy a TCONVERT függvénynek, mely valódi idő alapú dimenziók felett képes összesítést végezni csak egy összesítő függvény adható meg. Tehát ha az egyik dimenzió valamely eleme szerint más aggregációs függvényre van szükségünk, akkor azt az elemet ki kell hagynunk, vagy megtarthatjuk, de az általa indexelt adat hibás lesz. Eljárásokkal készített jelentések esetén viszont a ROW parancs segítségével, mely a REPORT parancs alapját is képezi, lehetőségünk van soronként felépíteni a jelentést, melyből az éppen aktuális sorra kell megadnunk, hogy mit szeretnénk megjeleníteni. Lehetőségünk van maximum 32 szintű oszlop összegzésre is, mely teljes és részleges összegzést is jelenthet. valamint hozzáférhetünk bármely oszlop értékéhez is, ami számítások végzésére ad lehetőséget. Az eljárásokon belüli jelentés készítéshez néhány extra parancs is, melyek a teljes riport kinézetére vannak hatással (12. táblázat).

6.2.2 A mintapélda jelentései:

A mintapéldában az első nyolc kérdés megválaszolása lehetséges jelentések segítségével. Ennek megfelelően célszerű sorra venni, s megválaszolni a kérdéseket. Mivel minden kérdés megválaszolásához limiteket is állítani kell, a jelentéseket eljárásokkal fogjuk létrehozni, melyeknek KERDES.<sorszám>.REP típusú nevet adunk.

1. kérdés: *Hogyan változott a webáruház forgalma az idők folyamán?*

Egy paraméterezhető lekérdetést jelentést készítünk, ahol a paraméter annak az idő dimenziónak a neve, mely az összesítési szintet jelzi. A forrás a következőképpen alakul:

```
argument dim text
argument htext text

pushlevel 'KERDES1 start'

push C_TULAJ I_ARKAT I_ELOTTE I_EV I_HARMADHONAP -
     I_HAZAI_KULFOLDI I_HELY I_HONAP I_IDO -
     I_JEGY_BERLET I_MUFAJ I_NAP I_NEGYEDEV -
     I_ORSZAG I_SZALL I_SZALLMOD I_VIDEK_FOVAROS

limit C_TULAJ to all
limit I_ELOTTE to all
limit I_HELY to all
limit I_SZALL to all
limit I_JEGY_BERLET to all
limit I_ARKAT to all
limit I_MUFAJ to all

limit C_TULAJ to hierarchy inverted
limit I_ELOTTE to ancestors
limit I_HELY to topancestors
limit I_SZALL to ancestors
limit I_JEGY_BERLET to ancestors
limit I_ARKAT to ancestors
limit I_MUFAJ to ancestors


report heading 'Tulajdonos' -
       w 22 truncate down c_tulaj.nev -
       heading htext across  &dim : -
       heading 'Bevételek évenként' -
       <w 12 decimal 0 heading 'Bevétel' i_osszeg -
        w 7 decimal 0 Heading 'DB' i_jegyszam>

poplevel 'KERDES1 start'
```

Ezt az 'i_ev' és 'Év' paraméterekkel meghívva a következő eredményt kapjuk:

	Bevételek					
	Év					
	2004		2005		2006	
Tulajdonos	Bevétel	DB	Bevétel	DB	Bevétel	DB
Terem Színház	1 108 200	465	4 294 700	1 716	4 477 300	1 664
Kisszínház	3 272 400	1 610	14 944 600	6 620	11 809 800	4 843
Park Színház	7 037 700	4 085	23 830 300	12 385	22 420 900	10 470
Minden rendezvénygazda	11 418 300	6 160	43 069 600	20 721	38 708 000	16 977

2. kérdés: *Hogyan változott az internetes és a jegyirodai értékesítés aránya?*

Erre a kérdésre a F_ADATOK kocka alapján tudunk választ adni. Mivel itt jelentkezik a korábban említett összesítési probléma, a jelentést soronként kell felépítenünk. A KERDES.2.REP kódja a következő:

```
argument time_dim id
variable cf text
variable coeff integer

pushlevel 'KERDES2'
push c_tulaj
push c_honap
push f_adat

limit c_tulaj to all
limit c_honap to all
limit f_adat to all
limit c_tulaj to hierarchy inverted

for c_tulaj
do
  blank
  coeff = 20 + statlen(&time_dim) * 21
  ROW w coeff under '=' c_tulaj.nev

  ROW under '-' w 20 skip under '-' -
      across &time_dim : w 20 center  c_ev

  for f_adat
  do
    if f_adat eq 'Iarany'
      then do
        cf = 'average'
        coeff = 100
      doend
      else do
        cf = 'sum'
        coeff = 1
      doend

    ROW w 20 f_adat.nev rowtotals -
        across &time_dim: w 20 decimal 2 -
        tConvert( (f_adatok * coeff) &time_dim &cf)

  doend
doend

poplevel 'KERDES2'
```

A 'c_év'[5] paraméterrel meghívva kapjuk az eredményt, melynek az utolsó szakasza a következő:

5 Azért használom minden esetben az év szintű összesítést, mert a jelentések így is elégé szélesek, a következő szint, mely negyedéveket jelent értelemszerűen kb. négyszeres szélességet jelentene.

```
Minden rendezvénygazda
================================================================================
                          2004               2005               2006
-------------------- ------------------ ------------------ ------------------
Jegyiroda             339 382 764,00     820 731 386,00     445 744 774,00
Internet               11 418 300,00      43 045 600,00      38 703 800,00
Összesen              350 801 064,00     863 776 986,00     484 448 574,00
Internet aránya                 3,00               6,17               9,43
Jutalék                 1 141 830,00       4 304 560,00       3 870 380,00
Nettó árrés               951 525,00       3 587 133,32       3 225 316,66
```

3. kérdés: *Hány nappal az előadás előtt vásárolnak a vevők jegyet?*

Itt is tekintsük külön a jegyeket és bérleteket, s az időszak legyen paraméter. A jelentés soraiban legyenek a típusok és a előrevásárlás értékei, mégpedig úgy, hogy a típus csak az első sorban szerepel, és csak azok a sorok jelenjenek meg, melyekben legalább annyi jegyet vagy bérletet vettek, amennyit paraméterként megadtunk, ez lesz a harmadik paraméter. Ennek megfelelően a KERDES.3.REP a követezőképpen alakul:

```
argument dim text
argument htext text
argument minval integer
vrb f boolean
vrb s integer
vrb fc text

pushlevel 'KERDES1 start'
push C_TULAJ I_ARKAT I_ELOTTE I_EV I_HARMADHONAP -
     I_HAZAI_KULFOLDI I_HELY I_HONAP I_IDO -
     I_JEGY_BERLET I_MUFAJ I_NAP I_NEGYEDEV -
     I_ORSZAG I_SZALL I_SZALLMOD I_VIDEK_FOVAROS

limit I_IDO to all
limit C_TULAJ to all
limit I_ELOTTE to all
limit I_HELY to all
limit I_SZALL to all
limit I_JEGY_BERLET to all
limit I_ARKAT to all
limit I_MUFAJ to all
limit C_TULAJ to topancestors
limit I_ELOTTE to bottomdescendants
limit I_HELY to topancestors
limit I_SZALL to ancestors
limit I_JEGY_BERLET to bottomdescendants
limit I_ARKAT to ancestors
limit I_MUFAJ to ancestors
limit &dim to all

„Meghatározzuk a sorhosszt, a fejléchez
s = 26 + statlen(&dim) * 11;
row under '=' w s center 'Előrevásárlások'
blank
row w 26 skip under '-' w s-26 center htext
row under '-' w 10 center 'Típus' -
    under '-' w 15 center 'Előrevásárlás' -
    across &dim : under '-' center &dim

for I_JEGY_BERLET
do
```

```
  f = true
  for i_elotte
  do
    s = 0
    for &dim
    do
     „Számításokhoz a for ciluson belül pontosan
     „dimenzionálni kell a kockát
      s = s + nafill(i_jegyszam(&dim &dim -
                     i_elotte i_elotte -
                     i_jegy_berlet i_jegy_berlet), 0)
    doend
    if s ge nafill(minval 0)
    then do
      if f
        then fc = 'w 10 i_jegy_berlet.nev '
        else fc = 'w 10 skip'

      row &fc w 15 right i_elotte.nev across &dim : -
          w 10 decimal 0 nafill(i_jegyszam 0)

      f = false
    doend
  doend
  blank
doend

poplevel 'KERDES1 start'
```

A **KERDES.3.REP 'i_ev' 'ÉV' 50** parancs kiadása után a következő eredményt kapjuk (részlet):

```
                        Előrevásárlások
=============================================================


                                        Év
                          ---------------------------------
  Típus     Előrevásárlás     2004       2005       2006
---------- --------------- ---------- ---------- ----------
Jegy              0 héttel        676      2 471      2 318
                  1 héttel        605      2 320      2 181
                  2 héttel        641      2 164      1 709
                  3 héttel        857      3 007      2 049
                  4 héttel      1 008      3 446      2 641
                  5 héttel      1 014      2 795      2 170
                  6 héttel        666      2 343      1 784
                    . . .         . . .    . . .      . . .
```

4. kérdés: *Milyen az internetes vásárlók összetétele lakóhely szerint?*

A KERDES.4.ALL eljárás esetén csak a REPORT parancsot és az eredményt emelem ki a következőkben[6], az eddigiek alapján az eljárás már könnyen megírható. Az egyetlen extra, hogy a lakhely dimenziót is paraméterként kérjük be.

```
report heading hely_htext w 20 down &hely_dim
       heading ido_htext across &ido_dim :
       heading 'Bevételek és eladott jegyek lakhely - sze-
rinti megoszlása'  -
       <heading 'Bevétel' w 12 decimal 0 i_osszeg -
        heading 'Jegy db' w 7 decimal 0 i_jegyszam>
```

6 Természetesen a mellékelt lemezen lévő adattárházban minden program forrása megtalálható.

```
                    ---Bevételek és eladott jegyek lakhely szerinti megoszlása----
                    -------------------------------Év-------------------------------
                    --------2004-------- --------2005-------- --------2006--------
Lakhely               Bevétel    Jegy db   Bevétel    Jegy db   Bevétel    Jegy db
------------------- ------------ ------- ------------ ------- ------------ -------
Magyar                11 172 000   6 005   42 190 300  20 296   37 905 200  16 628
Külföldi                 246 300     155      879 300     425      802 800     349
Minden lakhely        11 418 300   6 160   43 069 600  20 721   38 708 000  16 977
```

5. kérdés: *Hogyan változik a szállítási módok aránya?*

Az előzőekből már létrehozható a megfelelő eljárás, mely évek szerinti összesítés kérésekor a következő eredményt adja:

```
                    -----------------Bevételek és eladott jegyek -----------------
                    -------------szállitási módok szerinti megoszlása-------------
                    -------------------------------Év-------------------------------
                    --------2004-------- --------2005-------- --------2006--------
Szállításmód          Bevétel    Jegy db   Bevétel    Jegy db   Bevétel    Jegy db
------------------- ------------ ------- ------------ ------- ------------ -------
Pénztári jegyátvétel  10 737 900   5 813   41 000 100  19 807   36 853 100  16 327
Postai kézbesítés        680 400     347    2 069 500     914    1 582 500     599
GLS csomagküldo szol           0       0            0       0      272 400      51
```

6. kérdés: *Milyen ársávokban lévő jegyek illetve bérletek fogynak a legjobban?*

A választ tartalmazó jelentés létrehozásához a KERDES.3.REP eljárást tekintsük alapnak, és annak módosításával már előállítható a szükséges program.

```
              Árkategóriák szerinti megoszlás
============================================================

                                    Év
                          ---------------------------------
  Típus      Áakategória       2004       2005       2006
---------- ---------------- ---------- ---------- ----------
Jegy          1. kategória      4 784     15 416     12 845
              2. kategória      1 376      4 883      3 574
              3. kategória          0          0          2

Bérlet        1. kategória          0        407        543
              2. kategória          0         15         13
              3. kategória          0          0          0
```

7. kérdés: *Hogyan oszlik meg az eladott jegyek száma és a bevétel műfajok szerint?*

Ez szintén egy viszonylag egyszerű, REPORT alapú eljárással előállítható, és a következő eredményt adja:

```
                    ---Bevételek és eladott jegyek műfajok szerinti megoszlása----
                    -------------------------------Év-------------------------------
                    --------2004-------- --------2005-------- --------2006--------
Műfaj                 Bevétel    Jegy db   Bevétel    Jegy db   Bevétel    Jegy db
------------------- ------------ ------- ------------ ------- ------------ -------
Bérlet                        0       0    2 384 600     422    3 872 300     556
Előadóest                 3 000       3        5 800       5            0       0
Musical               2 019 500   1 127    5 048 700   2 729    4 869 600   2 562
Rendezvény                    0       0       84 000      49            0       0
Színmű                2 922 400   1 577   13 661 600   6 716   12 632 300   5 834
Táncjáték               706 300     246    2 803 000     971    1 532 300     569
Tragédia                 74 000      37      116 400      60       32 400      18
Vígjáték              3 841 200   2 116   12 908 700   6 729   11 613 200   5 709
Koncert                 200 100     173      592 300     398            0       0
Zenés darab             368 800     186      897 000     446      476 200     205
Dráma                   886 300     474    2 844 800   1 331    2 336 500     973
Mesejáték               108 000      90      252 000     210       97 200      81
Gálakoncert                   0       0       22 800      10            0       0
Családi Musical           142 900      50            0       0            0       0
Komédia                   145 800      81    1 447 900     645    1 246 000     470
```

8. kérdés: *Milyen a vásárlási kísérletek hibacsoportonkénti megoszlása?*

Ezt a kérdést a T_ADATOK tábla alapján készített jelentéssel tudjuk megválaszolni, és a 2. valamint a 4. kérdés megválaszolásához használt program alapján könnyen megvalósítható, eredménye pedig a következő.

```
                    -Vásárlások hibák szerinti megoszlása-
                    ------------------Év------------------
                    ----2004---- ----2005---- ----2006----
Sikeresség          Vásárlásszám Vásárlásszám Vásárlásszám
------------------- ------------ ------------ ------------
Sikeres                    2 244        7 724        6 658
Sikertelen                 1 764        4 889        3 034
```

6.3 OLAP_TABLE

6.3.1 Az OLAP_TABLE lehetőségei

Az OLAP_TABLE függvény célja, hogy az analitikus munkaterületet lekérdezhetővé váljon SQL utasításokkal, tehát a multidimenzionális és a relációs adatbázis közötti kapcsolatot biztosítja, hasonlóan az SQL alapú OLAP DML eljárásokhoz, csak a relációs adatbázis oldaláról. A függvény meghívásával egy relációs nézetet kapunk eredményül. Definiálhatunk SQL objektum típusokat, melyek leírják egy OLAP_TABLE által visszaadott sor felépítését, majd ennek segítségével e tábla típust, de ennek megadása nem kötelező, dinamikusan is felépülhet. Általában nézettáblát szokás definiálni egy kialakított OLAP_TABLE elérésére, ezáltal a rá épülő lekérdezések sokkal áttekinthetőbbek lesznek, és nem is kell bonyolult parancsokat megjegyezni. A ROLAP-ként való lekérdezés előtt általában szükség van néhány új objektum létrehozására a MOLAP adattárházban, például hierarchiák szintjetinek leírására.

Az OLAP_TABLE függvény általános alakja:

```
OLAP_TABLE (
  aw_attach      IN VARCHAR2,
  table_name     IN VARCHAR2,
  olap_command   IN VARCHAR2,
  limit_map      IN VARCHAR2
);
```

Ahol az egyes paraméterek a következőket jelentik:

aw_attach

Definiálja a munkaterülethez való csatlakozást. A DURATION rész határozza meg a kapcsolódás idejét. A QUERY csak e lekérdezés idejére, míg a SESSION a teljes munkamenetre kapcsolódik. Többszöri lekérdezés esetén a SESSION javasolt. Szintaxis:

```
aw_nev DURATION QUERY|SESSION
```

table_name

Ha korábban létrehoztunk az eredményt leíró képes tábla típust, akkor annak nevét itt adhatjuk meg, dinamikusan létrejövő tábla esetén hagyjuk üresen.

olap_command

Itt adhatunk meg olyan parancsot, vagy parancsokat, melyeket közvetlenül a lekérdezés előtt szeretnénk futtatni, például itt hívhatunk meg előkészítő programokat.

limit_map

Ebben a paraméterben írhatjuk le azt a szabályrendszert, amely szerint a visszaadott tábla létrejön Alapvetően a visszaadott dimenziók (DIMENSION) és azok hierarchiáinak (HIERARCHY), valamint a tényadatok (MEASURE) megadásából áll, de ezeken belül sok beállítás lehetséges.

Az OLAP_TABLE függvényt tartalmazó SELECT utasításban megadható egy MODEL záradék is, mely segítségével azt adatbázismotor jelentő optimalizációt tud végezni.

6.3.2 OLAP_TABLE a mintapéldában

Mintapéldánkban az utolsó kérdés megválaszolására célszerű ROLAP lekérdezést készíteni.

> ***9. kérdés:*** *Melyik volt a legjobb tíz nap illetve öt hónap a jegyértékesítés szempontjából, a bevétel és az eladott jegyek száma, valamint jegy és bérlet szerint külön-külön?*

A válaszhoz két nézettáblát fogunk létrehozni, az egyiket a napi, a másikat a havi adatok lekérdezéséhez, mindkettő struktúrája azonos, és leírásához objektumtáblát hozunk létre. Elsőként definiáljuk az objektum és a tábla típust:

```
create or replace type KERDES9_SOR as object (
      DATUM date,
      OSSZEG number(12,2),
      JEGYSZAM number(10)
);

create or replace type KERDES9_TABLA as
      table of KERDES9_SOR;
```

Ezután a segítségével létrehozzuk a nézeteket, az egyik a napi adatokat tartalmazza, és a V_NAPI_VASARLASOK nevet kapta, míg a másik a havi adatokat tartalmazza és a V_HAVI_VASARLASOK névvel érhető el. Mindkét esetben csak az I_IDO és az I_JEGY_BERLET dimenziókra szerinti felbontásra van szükségünk, így limitálást kell végeznünk, melyet a legegyszerűbb, ha egy előre definiált eljárásban végzünk el. Ez az eljárás a PRE.KERDES9.LIM nevet kapta, és a forrása a következő kódot tartalmazza:

```
allstat
limit C_TULAJ to ancestors
limit I_ELOTTE to ancestors
limit I_HELY to topancestors
limit I_SZALL to ancestors
limit I_JEGY_BERLET to bottomdescendants
limit I_ARKAT to ancestors
limit I_MUFAJ to ancestors
```

Ezt az eljárást hívjuk meg az OLAP_TABLE függvény „olap_command” paraméterében. Akkor definiálhatjuk is a két nézettáblát.

```
create or replace view V_NAPI_VASARLASOK as
select to_date(regexp_substr(DATUM, '[[:digit:]]{8}'),
                'yyyymmdd') as DATUM,
       TIPUS as TIPUS,
       OSSZEG as BEVETEL,
       JEGYSZAM as JEGYSZAM
from table(olap_table('ATH duration query',
            'KERDES9_TABLA',
            'call PRE.KERDES9.LIM',
            'dimension DATUM as varchar2(100) ' ||
            '  from I_IDO ' ||
            'dimension TIPUS as char(1) ' ||
            '  from I_JEGY_BERLET ' ||
            'measure OSSZEG as number(12,2) ' ||
```

```
            '  from I_OSSZEG ' ||
            'measure JEGYSZAM as number(10,0) ' ||
            '  from I_JEGYSZAM'))
where DATUM like '<I_NAP%';

create or replace view V_HAVI_VASARLASOK as
select to_date(regexp_substr(DATUM,
         '[[:digit:]]{4}_[[:digit:]]{2}'), 'yyyy_mm')
         as DATUM,
       TIPUS as TIPUS,
       OSSZEG as BEVETEL,
       JEGYSZAM as JEGYSZAM
from table(olap_table('ATH duration query',
            'KERDES9_TABLA',
            'call PRE.KERDES9.LIM',
            'dimension DATUM from I_IDO ' ||
            'dimension TIPUS from I_JEGY_BERLET ' ||
            'measure OSSZEG from I_OSSZEG ' ||
            'measure JEGYSZAM from I_JEGYSZAM'))
where DATUM like '<I_HONAP%';
```

Ezzel rendelkezésünkre áll a két szükséges nézettábla, ezekből már egyszerű SELECT utasításokkal lekérdezhetjük a választ a kérdésünkre. Elsőnek a top 10 napi bevételt listázzuk ki:

```
set pages 50
set feedback off
set linesize 60
ttitle 'Legjobb 10 nap a jegyeladásokból származó - bevétel
alapján'
column nap heading 'Nap' format A20
column bevetel heading 'Bevétel' format '999,999,999'

select to_char(datum, 'yyyy. month dd.') as nap,
       bevetel
  from (select *
          from v_napi_vasarlasok
          where bevetel is not null
                and tipus = 'J' order by bevetel desc)
  where rownum < 11;
```

Ennek eredménye:

```
Szo Máj 12                                              lap    1
A legjobb 10 nap a jegyeladásokból származó bevétel alapján

Nap                      Bevétel
-------------------- ------------
2006. február    06.    1,130,900
2005. november   07.    1,115,800
2006. április    03.    1,004,700
2006. március    06.      996,300
2006. március    02.      856,900
2006. március    01.      770,000
2006. április    04.      710,800
2005. november   02.      689,500
2006. február    01.      669,900
2006. március    07.      656,200
```

Ezután e legjobb 5 hónap következik szintén a bevétel alapján:

```
set pages 50
```

```
set feedback off
set linesize 60
ttitle 'Legjobb 10 nap a jegyeladásokból származó - bevétel
alapján'
column nap heading 'Nap' format A20
column bevetel heading 'Bevétel' format '999,999,999'

select to_char(datum, 'yyyy. month dd.') as nap,
       bevetel
  from (select *
          from v_napi_vasarlasok
          where bevetel is not null
                and tipus = 'B' order by bevetel desc)
  where rownum < 11;
```

Ennek eredménye:

```
V.  Máj 13                                              lap    1
A legjobb 5 hónap a bérleteladásokból származó  bevétel alapján

Nap                    Bevétel
-------------------- ------------
2006. május           1,971,700
2006. április           996,200
2005. augusztus         593,700
2006. június            527,400
2005. szeptember        498,000
```

Hasonló lekérdezések alapján az eladott jegyek száma szerint a következő eredményeket kapjuk:

```
V. Máj 13                                       lap    1
A legjobb 5 hónap az eladott bérletek száma alapján

Nap                  Eladott bérlet
-------------------- --------------
2006. május                     276
2006. április                   157
2005. augusztus                  97
2005. szeptember                 83
2005. június                     80

Szo Máj 12                                       lap    1
A legjobb 10 nap az eladott jegyek száma alapján

Nap                  Eladott jegy
-------------------- ------------
2006. február    06.          519
2005. november   07.          488
2006. április    03.          454
2006. március    06.          451
2006. március    02.          349
2006. április    04.          340
2006. március    01.          309
2006. március    07.          303
2006. február    01.          299
2006. január     04.          298
```

Ezzel az utolsó kérdésre is megadtuk a választ.

7. Zárszó

Szakdolgozatomban egy mintapéldán keresztül nyújtottam bepillantást az adattárház tervezés és az Oracle OLAP DML nyelv lehetőségeibe, természetesen ez utóbbival kapcsolatban nem törekedhettem mindenre kiterjedő ismertetésre, az utasítások szintaxisának leírása egy százötven oldalas dokumentum. Ez az a nyelv, mely minden Oracle OLAP eszköz műveleti bázisát képezi. Sajnos a szakdolgozatban nem térhettem ki mindenre, a későbbiekben érdemes vizsgálni az Oracle által nyújtott nyelvi kiterjesztéseket, például a DBMS_AW_XML-t, mely segítségével egy XML dokumentumban definiált logikai séma alapján hozható létre egy adatbázis. Említettem a frissítési rendszert, de a részleteibe terjedelmi okokból nem térhettem ki, ugyanúgy, mint az objektumorientált programozási nyelvekkel való kapcsolatra sem. Ezek a kapcsolatok azonban az OLAP DML alapjainak, valamint az adott nyelv jellegzetességeinek ismeretével már megismerhetőek. A JAVA nyelvű implementációról az Oracle részletes dokumentációt is készített (OLAP_APP_DEV_GUIDE_10.2.pdf), mely a honlapjukról szabadon letölthető a szükséges csomagokkal együtt.

Az adatbányászat eszközeivel, melyeket szintén csak említeni tudtam olyan bonyolult kérdésekre is választ kaphatunk, hogy milyen vásárlási szokások érvényesülnek a vásárlások során, például a visszatérő vevők mindig adott rendezvényszervezőnek az eseményire vesznek-e jegyet, vagy több különbözőre is, esetleg azonos típusú eseményeket látogatnak. A cég ezen kívül küld ki elektronikus hírleveleket is, illetve már promóciók is léteznek. Ezek hatékonyságának vizsgálata is igen hasznos a stratégiai tervezésben. Ahhoz azonban, hogy adatbányász eszközökkel tárjuk fel ezeket az információkat az adattárházunkból meg kell ismernünk ezen eszközök elvárásait azzal szemben.

A szakdolgozat készítésének kutatási szakaszában rengeteg kérdés vetődött fel, azonban ezek leginkább az OLAP DML mint adattárház definíciós nyelv szükségessége és használhatósága köré csoportosultak. A téma fokozatos körvonalazódása, majd a kidolgozás és a mintapélda elkészítésének időszaka ugyan a kérdéseket tisztázta, de további, az előzőnél konkrétebb és egyben nagyobb mennyiségű, megválaszolásra váró kérdést vetett fel, melyek válaszai újabb kutatásokat igényelnek, különösen az objektumorientált nyelvekből való elérés területén.

8. Táblázat és ábrajegyzék

Táblázatjegyzék

Ábrajegyzék

9. Irodalomjegyzék

Nyomtatott források

1. Kende Mária, Kotsis Domokos, Nagy István: Adatbázis-kezelés az Oracle rendszerben
   [Panem Könyvkiadó, 2002]
2. Jeffrey D. Ullman Jennifer Widom: Adatbázisrendszerek, alapvetés [Panem Könyvkiadó, 1998]
3. Jiawei Han, Micheline Kamber: Adatbányászat (Koncepciók és technikák) [Panem Könyvkiadó, 2004]
4. Learn Oracle From Oracle: Using OLAP DML [Oracle University, 2005]
5. Learn Oracle From Oracle: Program with OLAP DML [Oracle University, 2003]

Elektronikus források

1. http://www.oracle.com
   1. http://otn.oracle.com
   2. OLAP_DML_10.2.pdf
   3. OLAP_Ref_10.2.pdf
2. http://users.iit.uni-miskolc.hu/~kovacs/db2/ora10.html
3. http://www2.iit.uni-miskolc.hu/~kovacs/hirdetes/olapdoc.pdf

10. Mellékletek

Összefoglalás

Az adattárház technológia csupán mintegy másfél évtizede létező tudományág, ennek ellenére e relatíve rövid időszak alatt tetemes mennyiségű szakirodalom gyűlt össze a témában. A szakirodalom azonban a tudományág egyes területeit nem fedi le arányosan, míg az adatbányászat témaköre alaposan feltárt, az adattárak tervezésének és építésének témakörében az irodalmi források száma minimális. Ezért szakdolgozatom témaválasztásánál célul tűztem ki a meglevő szakirodalom összegyűjtését és rendszerezését, továbbá - a gyakorlati tapasztalatok dokumentálásával – kiegészítését az adatpiacok és adattárházak tervezési technikáinak témakörében.

Az irodalmi forráskutatás alapján a szakdolgozat első fejezeteiben történeti áttekintést adok a tudományág kialakulásának főbb lépéseiről és okairól, majd az általános tervezési elvekkel és a széles körben elfogadott ETL tervezési technikával foglalkozom, részletesen tárgyalva a kiindulási adatokban fellépő anomáliákat és azok kezelését.

Ezt követően az adattárház építésének gyakorlati megvalósítására térek rá, melyhez az Oracle által rendelkezésre bocsátott eszközöket használtam, így a kiindulási pont egy Oracle relációs adatbázis volt, az adattárház megvalósítása pedig az Oracle OLAP DML nyelv segítségével történt. Ezért részletesen bemutatom az Oracle OLAP DML nyelvet, mely az Oracle multidimenzionális adatbázisok létrehozására, karbantartására és lekérdezésére szolgáló legalacsonyabb szintű eszköz. Az Oracle minden magasabb szintű eszköze alapvetően erre a nyelvre épül, legyen az létrehozó és feltöltő (Oracle Warehouse Builder), vagy adatbányászati (Oracle Data Miner) célú alkalmazás.

Az adattárház létrehozása során az Analytic Workspace Manager és az OLAP Worksheet programokat használtam, mely utóbbi az OLAP DML utasítások parancssorból való kiadására, valamint SQL parancsok kiadására is alkalmas.

Az eszközök részletes ismertetése során kitérek az alaputasításoktól az objektumokon keresztül a programokig minden fontosabb elemre, átfogó képet adva az Oracle adattárházak építéséhez biztosított lehetőségeiről.

Az eszköz-háttér ismertetését követően részletesen ismertetem egy adattárház létrehozásának folyamatát egy jegyértékesítési rendszer felépítésének példáján. E mintapéldán keresztül bemutatom a korábbi fejezetekben ismertetett lépések gyakorlati megvalósítását, a specifikálástól a MOLAP adattárház tényleges létrehozásáig, és elemzési lehetőségekig. A konkrét eseten nem csupán az építés folyamatát, hanem a közben fellépő problémákat, s azokra adott megoldási javaslataimat kívánom bemutatni, hogy ezáltal a téma iránt érdeklődőknek egyfajta gyakorlati segítséget nyújthassak.

Summary in English

The data warehousing is a relatively new technology, its born is dated to 1992, when Bill Inmon published the the following definition:

> *"A data warehouse is a subject oriented, integrated, nonvolatile, and time variant collection of data in support of management's decisions."*

Since this time numerous books an articles was published about Data Mining, but almost nothing in subject of Data Warehouses' planning and building. The purpose of my choice for the theme my case study was, to collect and organize literature references and supplement them with my practical experiences in Warehouse-planning. Accordingly my thesis is running on techniques of plan and build Data Marts and Warehouses.

I was endeavoring to process the references and give an historical overview from the development of this discipline and the general planning principles. Than I explain the widely used ETL technology, the anomalies of the raw data and how to avoid them.

In the next three chapters I explained a practical implementation for which I used tools and applications produced by Oracle, so initially I built up an relational data base, than a data warehouse using OLAP DML. Therefore I show in detail this language, which is an low-level tool, and allows to create, maintenance and query Oracle multidimensional databases.

During the implementation I used two tools of Oracle, Analytic Workspace Manager and OLAP Worksheet, which are specially designed for executing OLAP DML and SQL commands.

In showing the language I expatiated on fundamental commands, objects and programming choices, to provide an overall picture of Oracle OLAP DML's abilities in building data-warehouses.

After exposition of OLAP DML I presented the utilization of theory in practice, when I built the Data Warehouse of Rendezvenyjegy Ltd., witch is engaged in the retail of tickets for various performances. In 5th chapter I demonstrated all steps of design and implementation of the data-warehouse. These steps were: specification, when I determined the problems to be solved, the design by ETL method, when I separated the relevant data from irrelevant, handled the anomalies and panned the way of data loading. After this, I defined the MOLAP structure, and filled with data from original relational database, and finally I answered the questions, what were defined during specification.

I hope that my thesis shows not only a special data warehouse implementation, but suggests some general problem solving theory, which are useful when somebody designs own Data Mart or Warehouse.

Analitikus munkaterület

Ez a melléklet az adattárház szöveges leírását tartalmazza, melyet az AWDESCRIBE programmal állítottam elő.

```
                    ATH Analitikus munkaterület-lista
                    =================================

Utolsó frissített elem:  13Máj07     Idő: 20:51:31

Nyomtatás dátuma:    13Máj07     Idő: 20:59:23

ATH a következőket tartalmazza:

   24 DIMENSIONS
    4 VARIABLES
   24 PROGRAMS
    9 RELATIONS
    1 MODEL
    3 AGGMAPS
   13 SURROGATES

A riport két részből áll:

     - Objektumlistából: az analitikus munkaterület-objektumok betűrendes felsorolásából,
     mely a következő oldalon kezdődik.

     - Objektumleírásokból: az analitikus munkaterület-objektumok részletes leírásaiból,
       melyek objektumtípus szerint és név alapján betűrend szerint rendezve jelennek meg.

NAME                 TYPE       DESCRIPTION
____                 ____       ___________

C_EV                 DIMENSION  'Év dimenzió'
C_EVAD               DIMENSION  'Évad dimenzió'
C_HONAP              DIMENSION  'Hónap dimenzió'
C_HONAP.L_PROG       PROGRAM
C_NEGYEDEV           DIMENSION  'Negyedév dimenzió'
C_TULAJ              DIMENSION  'A rendezvénygazda azonosítóját tartalmazó dimenzió'
C_TULAJ.L_PROG       PROGRAM    'Rendezvénygazdák dimenzót tölti fel'
C_TULAJ.NEV          SURROGATE  'A rendezvénygazda neve'
C_TULAJ.REL          RELATION   'Rendezvénygazdáktól független jelentések készítéséhez'
C_TULAJ.ROV_NEV      SURROGATE  'A rendezvénygazda rövid neve'
F_ADAT               DIMENSION  'Forgalmi adatsorok kódja'
F_ADAT.NEV           SURROGATE  'Forgalmi adatsorok leírása'
F_ADATOK             VARIABLE   'Forgalmi adatokat tartalmazó adatkocka'
F_ADATOK.AGG         AGGMAP     'Forgalmi adatok aggregációs szabálya'
F_ADATOK.LOAD        PROGRAM
F_ADATOK.MODEL       MODEL      'Forgalmi adatok számítását leíró modell'
I_ADATOK.AGG         AGGMAP     'Internetes eladások adatainak összesítési szabálya'
I_ADATOK.LOAD        PROGRAM
I_ADATOK.REP         PROGRAM
I_ARKAT              DIMENSION  'Árkategória azonosítóját tartalmazó dimenzió'
I_ARKAT.L_PROG       PROGRAM
I_ARKAT.NEV          SURROGATE  'Árkategória megnevezése'
I_ARKAT.REL          RELATION   'Árkategória összesítést biztosító reláció'
I_ELOTTE             DIMENSION  'A vásárlás a rendezvény előtt mennyivel történt hétben'
I_ELOTTE.L_PROG      PROGRAM    'Az előrevásárlás dimenziót tölti fel'
I_ELOTTE.NEV         SURROGATE  'A vásárlás a rendezvény előtt mennyivel történt, név'
I_ELOTTE.REL         RELATION   'Előrevásárlás összesítésére használható'
I_EV                 DIMENSION  'Év dimenzió'
I_HARMADHONAP        DIMENSION  'Harmadhónapokat tartalmazó dimenzió'
I_HAZAI_KULFOLDI     DIMENSION  'Hazai és külföldi felosztást tartalmazó dimenzió'
I_HAZAI_KULFOLDI.KOD SURROGATE  'Hazai és külföldi kódja'
I_HELY               DIMENSION  'Hely dimenzió'
I_HELY.L_PROG        PROGRAM    'Hely dimenziót tölti fel'
I_HELY.REL           RELATION   'Hely hierarchiát leíró reláció'
I_HONAP              DIMENSION  'Hónap dimenzió'
I_IDO                DIMENSION  'Idő dimenzió'
I_IDO.L_PROG         PROGRAM    'Internetes eladások idő dimenzióját tölti fel'
I_IDO.REL            RELATION   'Idő hierarchiát definiáló reláció'
I_JEGY_BERLET        DIMENSION  'Jegy és bérlet kódját tartalmazó dimenzió'
I_JEGY_BERLET.L_PROG PROGRAM
I_JEGY_BERLET.NEV    SURROGATE  'Jegy és bérlet megnevezése'
I_JEGY_BERLET.REL    RELATION   'Jegy és bérlet összesítést lehetővé tevő reláció'
I_JEGYSZAM           VARIABLE   'Interneten vásárolt jegyek száma'

NAME                 TYPE       DESCRIPTION
____                 ____       ___________

I_MUFAJ              DIMENSION  'Műfaj azonosítóját tartalmazó dimenzió'
I_MUFAJ.L_PROG       PROGRAM
I_MUFAJ.NEV          SURROGATE  'Műfaj megnevezése'
I_MUFAJ.REL          RELATION   'Műfaj összesítést lehetővé tevő hierarchia'
I_NAP                DIMENSION  'Nap dimanzió'
I_NEGYEDEV           DIMENSION  'Negyedév dimenzió'
I_ORSZAG             DIMENSION  'Ország dimenzió'
I_ORSZAG.KOD         SURROGATE  'Ország kódja'
I_OSSZEG             VARIABLE   Internetes vásárlások összege
I_SZALL              DIMENSION  'Szállításmód dimenzió'
I_SZALL.L_PROG       PROGRAM
I_SZALL.NEV          SURROGATE  'Szállításmód leírása'
```

```
I_SZALL.REL          RELATION   'Szállítási módok összesítéséhez használható hierarchia'
I_VIDEK_FOVAROS      DIMENSION  'Vidék - Főváros felosztást tartalmazó dimenzió'
I_VIDEK_FOVAROS.KOD  SURROGATE  'Vidék és főváros kódja'
KERDES.1.REP         PROGRAM
KERDES.2.REP         PROGRAM
KERDES.3.REP         PROGRAM
KERDES.4.REP         PROGRAM
KERDES.5.REP         PROGRAM
KERDES.6.REP         PROGRAM
KERDES.7.REP         PROGRAM
KERDES.8.REP         PROGRAM
LOAD.DIM             PROGRAM
PRE.KERDES9.LIM      PROGRAM
T_ADATOK             VARIABLE   'Tranzakciók sikerességének adatai'
T_ADATOK.AGG         AGGMAP     'Sikeres és sikertelen tranzakciók összesítéséhez'
T_ADATOK.LOAD        PROGRAM
T_EREDMENY           DIMENSION  'Tranzakció eredménye dimenzió'
T_EREDMENY.L_PROG    PROGRAM
T_EREDMENY.REL       RELATION   'Tranzakció eredménye hierarchiát leíró reláció'
T_HIBACSOPORT        DIMENSION  'Hibacsoportok dimenzió'
T_HIBACSOPORT.NEV    SURROGATE  'Hibacsoportok neve'
T_SIKERES_SIKERTELEN DIMENSION  'Sikeres és sikertelen csoportok'
T_SIKERES_SIKERTELEN SURROGATE  'Sikeres és sikertelen megnevezés'
.NEV

DEFINE C_EV DIMENSION YEAR
LD 'Év dimenzió'
VNF '<yyyy>'

     Hivatkozva a következők által:
         C_HONAP.L_PROG   I_ADATOK.LOAD    F_ADATOK.LOAD    T_ADATOK.LOAD
         KERDES.2.REP     KERDES.8.REP

DEFINE C_EVAD DIMENSION YEAR ENDING JÚLIUS
LD 'Évad dimenzió'
VNF '<yyyy> évad'

     Hivatkozva a következők által:
         C_HONAP.L_PROG   I_ADATOK.LOAD    F_ADATOK.LOAD    T_ADATOK.LOAD
         KERDES.8.REP

DEFINE C_HONAP DIMENSION MONTH
LD 'Hónap dimenzió'
VNF '<yyyy>. <mtextl>'

     Hivatkozva a következők által:
         C_HONAP.L_PROG   I_ADATOK.LOAD    F_ADATOK.LOAD    T_ADATOK.LOAD
         KERDES.2.REP     KERDES.8.REP

DEFINE C_NEGYEDEV DIMENSION QUARTER
LD 'Negyedév dimenzió'
VNF '<ffff> <p>. negyedév'

     Hivatkozva a következők által:
         C_HONAP.L_PROG   I_ADATOK.LOAD    F_ADATOK.LOAD    T_ADATOK.LOAD
         KERDES.8.REP

DEFINE C_TULAJ DIMENSION ID
LD 'A rendezvénygazda azonosítóját tartalmazó dimenzió'

     Hivatkozva a következők által:
         C_TULAJ.L_PROG   I_ADATOK.LOAD    F_ADATOK.LOAD    T_ADATOK.LOAD
         I_ADATOK.REP     PRE.KERDES9.LIM  KERDES.1.REP     KERDES.3.REP
         KERDES.4.REP     KERDES.5.REP     KERDES.6.REP     KERDES.2.REP
         KERDES.7.REP

DEFINE F_ADAT DIMENSION ID
LD 'Forgalmi adatsorok kódja'

     Hivatkozva a következők által:
         F_ADATOK.MODEL   I_ADATOK.LOAD    F_ADATOK.LOAD    T_ADATOK.LOAD
         KERDES.2.REP

DEFINE I_ARKAT DIMENSION ID
LD 'Árkategória azonosítóját tartalmazó dimenzió'

     Hivatkozva a következők által:
         I_ARKAT.L_PROG   I_ADATOK.LOAD    F_ADATOK.LOAD    T_ADATOK.LOAD
         PRE.KERDES9.LIM  KERDES.1.REP     KERDES.3.REP     KERDES.4.REP

         KERDES.5.REP     KERDES.6.REP     KERDES.7.REP

DEFINE I_ELOTTE DIMENSION ID
LD 'A vásárlás a rendezvény előtt mennyivel történt hétben'

     Hivatkozva a következők által:
         I_ELOTTE.L_PROG  I_ADATOK.LOAD    F_ADATOK.LOAD    T_ADATOK.LOAD
         PRE.KERDES9.LIM  KERDES.1.REP     KERDES.3.REP     KERDES.4.REP
         KERDES.5.REP     KERDES.6.REP     KERDES.7.REP

DEFINE I_EV DIMENSION ID
LD 'Év dimenzió'

     Hivatkozva a következők által:
         I_IDO.L_PROG     I_ADATOK.LOAD    F_ADATOK.LOAD    T_ADATOK.LOAD
         KERDES.1.REP     KERDES.3.REP     KERDES.4.REP     KERDES.5.REP
         KERDES.6.REP     KERDES.7.REP
```

```
DEFINE I_HARMADHONAP DIMENSION ID
LD 'Harmadhónapokat tartalmazó dimenzió'

     Hivatkozva a következők által:
         I_IDO.L_PROG     I_ADATOK.LOAD    F_ADATOK.LOAD    T_ADATOK.LOAD
         KERDES.1.REP     KERDES.3.REP     KERDES.4.REP     KERDES.5.REP
         KERDES.6.REP     KERDES.7.REP

DEFINE I_HAZAI_KULFOLDI DIMENSION TEXT WIDTH 100
LD 'Hazai és külföldi felosztást tartalmazó dimenzió'

     Hivatkozva a következők által:
         I_HELY.L_PROG    I_ADATOK.LOAD    F_ADATOK.LOAD    T_ADATOK.LOAD
         KERDES.1.REP     KERDES.3.REP     KERDES.4.REP     KERDES.5.REP
         KERDES.6.REP     KERDES.7.REP

DEFINE I_HELY DIMENSION CONCAT (I_ORSZAG I_VIDEK_FOVAROS I_HAZAI_KULFOLDI)
LD 'Hely dimenzió'

     Hivatkozva a következők által:
         I_HELY.L_PROG    I_ADATOK.LOAD    F_ADATOK.LOAD    T_ADATOK.LOAD
         PRE.KERDES9.LIM  KERDES.1.REP     KERDES.3.REP     KERDES.4.REP
         KERDES.5.REP     KERDES.6.REP     KERDES.7.REP

DEFINE I_HONAP DIMENSION ID
LD 'Hónap dimenzió'

     Hivatkozva a következők által:
         I_IDO.L_PROG     I_ADATOK.LOAD    F_ADATOK.LOAD    T_ADATOK.LOAD
         KERDES.1.REP     KERDES.3.REP     KERDES.4.REP     KERDES.5.REP
         KERDES.6.REP     KERDES.7.REP

DEFINE I_IDO DIMENSION CONCAT (I_NAP I_HARMADHONAP I_HONAP I_NEGYEDEV I_EV)
LD 'Idő dimenzió'

     Hivatkozva a következők által:
         I_IDO.L_PROG     I_ADATOK.LOAD    F_ADATOK.LOAD    T_ADATOK.LOAD

         KERDES.1.REP     KERDES.3.REP     KERDES.4.REP     KERDES.5.REP
         KERDES.6.REP     KERDES.7.REP

DEFINE I_JEGY_BERLET DIMENSION ID
LD 'Jegy és bérlet kódját tartalmazó dimenzió'

     Hivatkozva a következők által:
         I_JEGY_BERLET.L_ I_ADATOK.LOAD    F_ADATOK.LOAD    T_ADATOK.LOAD
         PROG
         PRE.KERDES9.LIM  KERDES.1.REP     KERDES.3.REP     KERDES.4.REP
         KERDES.5.REP     KERDES.6.REP     KERDES.7.REP

DEFINE I_MUFAJ DIMENSION ID
LD 'Műfaj azonosítóját tartalmazó dimenzió'

     Hivatkozva a következők által:
         I_MUFAJ.L_PROG   I_ADATOK.LOAD    F_ADATOK.LOAD    T_ADATOK.LOAD
         PRE.KERDES9.LIM  KERDES.1.REP     KERDES.3.REP     KERDES.4.REP
         KERDES.5.REP     KERDES.6.REP     KERDES.7.REP

DEFINE I_NAP DIMENSION ID
LD 'Nap dimanzió'

     Hivatkozva a következők által:
         I_IDO.L_PROG     I_ADATOK.LOAD    F_ADATOK.LOAD    T_ADATOK.LOAD
         KERDES.1.REP     KERDES.3.REP     KERDES.4.REP     KERDES.5.REP
         KERDES.6.REP     KERDES.7.REP

DEFINE I_NEGYEDEV DIMENSION ID
LD 'Negyedév dimenzió'

     Hivatkozva a következők által:
         I_IDO.L_PROG     I_ADATOK.LOAD    F_ADATOK.LOAD    T_ADATOK.LOAD
         KERDES.1.REP     KERDES.3.REP     KERDES.4.REP     KERDES.5.REP
         KERDES.6.REP     KERDES.7.REP

DEFINE I_ORSZAG DIMENSION TEXT WIDTH 100
LD 'Ország dimenzió'

     Hivatkozva a következők által:
         I_HELY.L_PROG    I_ADATOK.LOAD    F_ADATOK.LOAD    T_ADATOK.LOAD
         KERDES.1.REP     KERDES.3.REP     KERDES.4.REP     KERDES.5.REP
         KERDES.6.REP     KERDES.7.REP

DEFINE I_SZALL DIMENSION ID
LD 'Szállításmód dimenzió'

     Hivatkozva a következők által:
         I_SZALL.L_PROG   I_ADATOK.LOAD    F_ADATOK.LOAD    T_ADATOK.LOAD
         PRE.KERDES9.LIM  KERDES.1.REP     KERDES.3.REP     KERDES.4.REP
         KERDES.5.REP     KERDES.6.REP     KERDES.7.REP

DEFINE I_VIDEK_FOVAROS DIMENSION TEXT WIDTH 100
LD 'Vidék - Főváros felosztást tartalmazó dimenzió'

     Hivatkozva a következők által:
         I_HELY.L_PROG    I_ADATOK.LOAD    F_ADATOK.LOAD    T_ADATOK.LOAD
         KERDES.1.REP     KERDES.3.REP     KERDES.4.REP     KERDES.5.REP
         KERDES.6.REP     KERDES.7.REP
```

```
DEFINE T_EREDMENY DIMENSION CONCAT (T_HIBACSOPORT T_SIKERES_SIKERTELEN)
LD 'Tranzakció eredménye dimenzió'

     Hivatkozva a következők által:
         T_EREDMENY.L_PRO I_ADATOK.LOAD    F_ADATOK.LOAD    T_ADATOK.LOAD
         G
         KERDES.8.REP

DEFINE T_HIBACSOPORT DIMENSION ID
LD 'Hibacsoportok dimenzió'

     Hivatkozva a következők által:
         T_EREDMENY.L_PRO I_ADATOK.LOAD    F_ADATOK.LOAD    T_ADATOK.LOAD
         G
         KERDES.8.REP

DEFINE T_SIKERES_SIKERTELEN DIMENSION ID
LD 'Sikeres és sikertelen csoportok'

     Hivatkozva a következők által:
         T_EREDMENY.L_PRO I_ADATOK.LOAD    F_ADATOK.LOAD    T_ADATOK.LOAD
         G
         KERDES.8.REP

DEFINE F_ADATOK VARIABLE NUMBER (12,2) <C_HONAP C_TULAJ F_ADAT>
LD 'Forgalmi adatokat tartalmazó adatkocka'

     Hivatkozva a következők által:
         F_ADATOK.LOAD    KERDES.2.REP

DEFINE I_JEGYSZAM VARIABLE INTEGER <I_IDO C_TULAJ I_ELOTTE I_HELY I_SZALL I_JEGY_BERLET -
        I_ARKAT I_MUFAJ>
LD Interneten vásárolt jegyek száma

     Hivatkozva a következők által:
         I_ADATOK.LOAD    KERDES.1.REP     KERDES.3.REP     KERDES.4.REP
         KERDES.5.REP     KERDES.6.REP     KERDES.7.REP

DEFINE I_OSSZEG VARIABLE NUMBER (12,2) <I_IDO C_TULAJ I_ELOTTE I_HELY I_SZALL I_JEGY_BERLET -
       I_ARKAT I_MUFAJ>
LD Internetes vásárlások összege

     Hivatkozva a következők által:
         I_ADATOK.LOAD    I_ADATOK.REP     KERDES.1.REP     KERDES.4.REP
         KERDES.5.REP     KERDES.7.REP

DEFINE T_ADATOK VARIABLE INTEGER <C_HONAP T_EREDMENY>
LD 'Tranzakciók sikerességének adatai'

     Hivatkozva a következők által:
         T_ADATOK.LOAD    KERDES.8.REP

DEFINE C_TULAJ.REL RELATION C_TULAJ <C_TULAJ>
LD 'Rendezvénygazdáktól független jelentések készítéséhez'

     Hivatkozva a következők által:
         I_ADATOK.AGG     F_ADATOK.AGG     C_TULAJ.L_PROG

DEFINE I_ARKAT.REL RELATION I_ARKAT <I_ARKAT>
LD 'Árkategória összesítést biztosító reláció'

     Hivatkozva a következők által:
         I_ADATOK.AGG     I_ARKAT.L_PROG

DEFINE I_ELOTTE.REL RELATION I_ELOTTE <I_ELOTTE>
LD 'Előrevásárlás összesítésére használható'

     Hivatkozva a következők által:
         I_ADATOK.AGG     I_ELOTTE.L_PROG

DEFINE I_HELY.REL RELATION I_HELY <I_HELY>
LD 'Hely hierarchiát leíró reláció'

     Hivatkozva a következők által:
         I_ADATOK.AGG     I_HELY.L_PROG

DEFINE I_IDO.REL RELATION I_IDO <I_IDO>
LD 'Idő hierarchiát definiáló reláció'

     Hivatkozva a következők által:
         I_ADATOK.AGG     I_IDO.L_PROG

DEFINE I_JEGY_BERLET.REL RELATION I_JEGY_BERLET <I_JEGY_BERLET>
LD 'Jegy és bérlet összesítést lehetővé tevő reláció'

     Hivatkozva a következők által:
         I_ADATOK.AGG     I_JEGY_BERLET.L_
                          PROG

DEFINE I_MUFAJ.REL RELATION I_MUFAJ <I_MUFAJ>
LD 'Műfaj összesítést lehetővé tevő hierarchia'

     Hivatkozva a következők által:
         I_ADATOK.AGG     I_MUFAJ.L_PROG

DEFINE I_SZALL.REL RELATION I_SZALL <I_SZALL>
LD 'Szállítási módok összesítéséhez használható hierarchia'
```

```
     Hivatkozva a következők által:
         I_ADATOK.AGG     I_SZALL.L_PROG

DEFINE T_EREDMENY.REL RELATION T_EREDMENY <T_EREDMENY>
LD 'Tranzakció eredménye hierarchiát leíró reláció'

     Hivatkozva a következők által:

         T_ADATOK.AGG     T_EREDMENY.L_PRO
                          G

DEFINE F_ADATOK.AGG AGGMAP
LD 'Forgalmi adatok aggregációs szabálya'
AGGMAP
relation c_tulaj.rel
END

     Hivatkozva a következők által:
         F_ADATOK.LOAD

DEFINE I_ADATOK.AGG AGGMAP
LD 'Internetes eladások adatainak összesítési szabálya'
AGGMAP
relation i_ido.rel
relation c_tulaj.rel
relation i_elotte.rel
relation i_hely.rel
relation i_szall.rel
relation i_jegy_berlet.rel
relation i_arkat.rel
relation i_mufaj.rel
END

     Hivatkozva a következők által:
         NONE

DEFINE T_ADATOK.AGG AGGMAP
LD 'Sikeres és sikertelen tranzakciók összesítéséhez'
AGGMAP
relation t_eredmeny.rel
END

     Hivatkozva a következők által:
         T_ADATOK.LOAD

DEFINE C_TULAJ.NEV SURROGATE C_TULAJ TEXT WIDTH 300
LD 'A rendezvénygazda neve'

     Hivatkozva a következők által:
         C_TULAJ.L_PROG   I_ADATOK.REP     KERDES.1.REP     KERDES.2.REP

DEFINE C_TULAJ.ROV_NEV SURROGATE C_TULAJ TEXT WIDTH 50
LD 'A rendezvénygazda rövid neve'

     Hivatkozva a következők által:
         C_TULAJ.L_PROG

DEFINE F_ADAT.NEV SURROGATE F_ADAT TEXT WIDTH 100
LD 'Forgalmi adatsorok leírása'

     Hivatkozva a következők által:
         KERDES.2.REP

DEFINE I_ARKAT.NEV SURROGATE I_ARKAT TEXT WIDTH 50
LD 'Árkategória megnevezése'

     Hivatkozva a következők által:
         I_ARKAT.L_PROG   I_ADATOK.REP     KERDES.6.REP

DEFINE I_ELOTTE.NEV SURROGATE I_ELOTTE TEXT WIDTH 50
LD ' A vásárlás a rendezvény előtt mennyivel történt, név'

     Hivatkozva a következők által:
         I_ELOTTE.L_PROG  KERDES.3.REP     KERDES.7.REP

DEFINE I_HAZAI_KULFOLDI.KOD SURROGATE I_HAZAI_KULFOLDI ID
LD 'Hazai és külföldi kódja'

     Hivatkozva a következők által:
         I_HELY.L_PROG

DEFINE I_JEGY_BERLET.NEV SURROGATE I_JEGY_BERLET TEXT WIDTH 100
LD 'Jegy és bérlet megnevezése'

     Hivatkozva a következők által:
         I_JEGY_BERLET.L_
         PROG

DEFINE I_MUFAJ.NEV SURROGATE I_MUFAJ TEXT WIDTH 100
LD 'Műfaj megnevezése'

     Hivatkozva a következők által:
         I_MUFAJ.L_PROG   KERDES.7.REP

DEFINE I_ORSZAG.KOD SURROGATE I_ORSZAG ID
LD 'Ország kódja'

     Hivatkozva a következők által:
```

```
        I_HELY.L_PROG

DEFINE I_SZALL.NEV SURROGATE I_SZALL TEXT WIDTH 100
LD 'Szállításmód leírása'

     Hivatkozva a következők által:
        I_SZALL.L_PROG   KERDES.5.REP

DEFINE I_VIDEK_FOVAROS.KOD SURROGATE I_VIDEK_FOVAROS ID
LD 'Vidék és főváros kódja'

     Hivatkozva a következők által:
        I_HELY.L_PROG

DEFINE T_HIBACSOPORT.NEV SURROGATE T_HIBACSOPORT TEXT WIDTH 50
LD 'Hibacsoportok neve'

     Hivatkozva a következők által:
        T_EREDMENY.L_PRO
        G

DEFINE T_SIKERES_SIKERTELEN.NEV SURROGATE T_SIKERES_SIKERTELEN TEXT WIDTH 50
LD 'Sikeres és sikertelen megnevezés'

     Hivatkozva a következők által:
        T_EREDMENY.L_PRO
        G

DEFINE C_HONAP.L_PROG PROGRAM
PROGRAM
variable maxd date
variable mind date
variable ks text
variable vs text
SQL DECLARE cr_ido CURSOR FOR select min(idopont) k, max(idopont) v from r_fadat

pushlevel 'HONAP start'
push c_honap
push c_negyedev
push c_ev
push c_evad

SQL OPEN cr_ido
SQL FETCH cr_ido INTO :mind :maxd
ks = to_char(mind, 'yyyy mon dd')
vs = to_char(maxd, 'yyyy mon dd')
show joinchars( ks ' - ' vs)
limit c_honap to last 1
if statlen( c_honap ) lt 1
  then do
   "Ha még nincs elem a dimenziuóban,
   "akkor a tejes intervallumot hozzáadjuk
    maintain c_honap add ks vs
    maintain c_negyedev add ks vs
    maintain c_ev add ks vs
    maintain c_evad add ks vs
  doend
  else do
    "Ha van elem, és az utolsó kisebb, mint a max
    "dátum, akkor csak a legnagyobbat kell hozzáadni
    if maxd gt c_honap
      then do
        maintain c_honap add vs
        maintain c_negyedev add vs
        maintain c_ev add vs
        maintain c_evad add vs
      doend
  doend

poplevel 'HONAP start'
END

     Hivatkozások:
        C_EV            C_EVAD          C_HONAP         C_NEGYEDEV

     Hivatkozva a következők által:
        LOAD.DIM

DEFINE C_TULAJ.L_PROG PROGRAM
LD 'Rendezvénygazdák dimenzót tölti fel'
PROGRAM
"Elmentjük a jelenlegi állapotot
pushlevel 'Start'
push c_tulaj

"Deklaráljuk a forrás kurzort
SQL DECLARE cr_tulaj CURSOR FOR -
  select azon, nev, rov_nev from r_tulaj order by azon

"A minden dimenziót '0'-es értékkel fogjuk jelölni
"Ha még nem létezik, akkor hozzáadjuk
limit c_tulaj to all
if not isvalue(c_tulaj '0')
  then do
    show 'nincs 0'
    maintain c_tulaj add '0'
    limit c_tulaj to '0'
    c_tulaj.nev = 'Minden rendezvénygazda'
```

```
    c_tulaj.rov_nev = 'Mind'
    limit c_tulaj to all
  doend

SQL OPEN cr_tulaj
SQL FETCH cr_tulaj LOOP INTO :APPEND c_tulaj -
                        :ASSIGN c_tulaj.nev -
                        :ASSIGN c_tulaj.rov_nev

limit c_tulaj remove '0'
c_tulaj.rel = '0'

poplevel 'Start'
END
```

Hivatkozások:

C_TULAJ	C_TULAJ.NEV	C_TULAJ.REL	C_TULAJ.ROV_NEV

Hivatkozva a következők által:

LOAD.DIM

```
DEFINE F_ADATOK.LOAD PROGRAM
PROGRAM
SQL DECLARE cr_fadat CURSOR FOR -
      select idopont, tulaj, tipus, osszeg -
      from r_fadat -
      order by idopont, tulaj, tipus

pushlevel 'F_ADATOK start'
push C_EV C_EVAD C_HONAP C_NEGYEDEV C_TULAJ F_ADAT I_ARKAT I_ELOTTE I_EV I_HARMADHONAP -
     I_HAZAI_KULFOLDI I_HELY I_HONAP I_IDO I_JEGY_BERLET I_MUFAJ I_NAP I_NEGYEDEV -
     I_ORSZAG I_SZALL I_VIDEK_FOVAROS T_EREDMENY T_HIBACSOPORT T_SIKERES_SIKERTELEN

ALLSTAT

SQL OPEN cr_fadat

SQL FETCH cr_fadat LOOP INTO :MATCH c_honap -
                             :MATCH c_tulaj -
                             :MATCH f_adat  -
                             :F_ADATOK

SQL CLEANUP

aggregate F_ADATOK using F_ADATOK.agg
F_ADATOK.MODEL F_ADATOK

poplevel 'F_ADATOK start'
END
```

Hivatkozások:

C_EV	C_EVAD	C_HONAP	C_NEGYEDEV
C_TULAJ	F_ADAT	F_ADATOK	F_ADATOK.AGG
F_ADATOK.MODEL	I_ARKAT	I_ELOTTE	I_EV
I_HARMADHONAP	I_HAZAI_KULFOLDI	I_HELY	I_HONAP
I_IDO	I_JEGY_BERLET	I_MUFAJ	I_NAP
I_NEGYEDEV	I_ORSZAG	I_SZALL	I_VIDEK_FOVAROS
T_EREDMENY	T_HIBACSOPORT	T_SIKERES_SIKERTELEN	

Hivatkozva a következők által:

NONE

```
DEFINE I_ADATOK.LOAD PROGRAM
PROGRAM
vrb ido id
vrb tul id
vrb el  id
vrb he   text
vrb sz  id
vrb a   id
vrb t   id
vrb m   id
vrb am  number
vrb db  integer
vrb kk  id
vrb c   integer
SQL DECLARE cr_iadat CURSOR FOR -
      select to_char(idopont, 'yyyymmdd'), tulaj, elotte, -
             rh.nev, szall, arkat, tipus, mufaj, osszeg, db, -
             kateg_kod -
      from r_iadat ri, r_hely rh -
      where ri.hely = rh.kod
"Mentsünk el mindent
pushlevel 'I_ADATOK'
push C_EV C_EVAD C_HONAP C_NEGYEDEV C_TULAJ F_ADAT I_ARKAT I_ELOTTE I_EV I_HARMADHONAP -
     I_HAZAI_KULFOLDI I_HELY I_HONAP I_IDO I_JEGY_BERLET I_MUFAJ I_NAP I_NEGYEDEV -
     I_ORSZAG I_SZALL I_VIDEK_FOVAROS T_EREDMENY T_HIBACSOPORT T_SIKERES_SIKERTELEN

ALLSTAT

SQL OPEN cr_iadat
c = 0

SQL FETCH cr_iadat INTO :ido :tul :el :he :sz :a :t :m :am :db :kk
while SQLCODE eq 0
do
  show c
```

```
  limit i_ido to i_ido(i_nap ido)
  if statlen( i_ido ) ne 1
    then signal 'Nap hiba'
  limit c_tulaj to tul
  if statlen( c_tulaj ) ne 1
    then signal 'Tulaj hiba'
  limit i_elotte to el
  if statlen( i_elotte ) ne 1
    then signal 'Elotte hiba'
  if kk eq 'kulf'
    then limit i_hely to i_hely(i_orszag he)
    else limit i_hely to i_hely(i_videk_fovaros he)
  if statlen( i_hely ) ne 1
    then signal 'Hely hiba'
  limit i_szall to sz
  if statlen( i_szall ) ne 1
    then signal 'Száll hiba'
  limit i_arkat to a
  if statlen( i_arkat ) ne 1

    then signal 'Arkat hiba'
  limit i_jegy_berlet to t
  if statlen( i_jegy_berlet ) ne 1
    then signal 'Tipus hiba'
  limit i_mufaj to m
  if statlen( i_mufaj ) ne 1
    then signal 'Mufaj hiba'

  i_osszeg = am
  i_jegyszam = db

  c = c +1
  SQL FETCH cr_iadat INTO :ido :tul :el :he :sz :a :t :m :am :db :kk
doend

SQL CLENUP

"allstat
"aggregate i_osszeg using i_adatok.agg
"aggregate i_jegyszam using i_adatok.agg

poplevel 'I_ADATOK'
END
```

Hivatkozások:

C_EV	C_EVAD	C_HONAP	C_NEGYEDEV
C_TULAJ	F_ADAT	I_ARKAT	I_ELOTTE
I_EV	I_HARMADHONAP	I_HAZAI_KULFOLDI	I_HELY
I_HONAP	I_IDO	I_JEGY_BERLET	I_JEGYSZAM
I_MUFAJ	I_NAP	I_NEGYEDEV	I_ORSZAG
I_OSSZEG	I_SZALL	I_VIDEK_FOVAROS	T_EREDMENY
T_HIBACSOPORT	T_SIKERES_SIKERTELEN		

Hivatkozva a következők által:

NONE

```
DEFINE I_ADATOK.REP PROGRAM
PROGRAM
argument time_dim id
variable cf text
variable coeff integer
row skip across &time_dim : across i_arkat.nev : &time_dim
row skip across &time_dim : across i_arkat.nev : i_arkat.nev
for c_tulaj
do
  ROW w 20 c_tulaj.nev across &time_dim : across i_arkat.nev : w 20 decimal 2 i_osszeg
doend
END
```

Hivatkozások:

C_TULAJ	C_TULAJ.NEV	I_ARKAT.NEV	I_OSSZEG

Hivatkozva a következők által:

NONE

```
DEFINE I_ARKAT.L_PROG PROGRAM
PROGRAM
variable i integer
variable s id
pushlevel 'ARKAT start'
push i_arkat

limit i_arkat to all

if not isvalue(i_arkat 'A')
  then do
    maintain i_arkat add 'A'
    limit i_arkat to 'A'
    i_arkat.nev = 'Minden kategória'
    limit i_arkat to all
  doend

i = 1

while i le 3
do
  s = to_char(i, '9')
```

```
  if not isvalue(i_arkat s)
    then do
      maintain i_arkat add s
      limit i_arkat to s
      i_arkat.nev = joinchars( s '. kategória')
      i_arkat.rel = 'A'
      limit i_arkat to all
    doend
  i = i + 1
doend

poplevel 'ARKAT start'
END

     Hivatkozások:
         I_ARKAT          I_ARKAT.NEV      I_ARKAT.REL

     Hivatkozva a következők által:
         LOAD.DIM

DEFINE I_ELOTTE.L_PROG PROGRAM
LD Az előrevásárlás dimenziót tölti fel
PROGRAM
pushlevel 'Start'
push i_elotte

SQL DECLARE cr_elotte CURSOR FOR select distinct x.elotte, x.elotte || ' héttel' from r_iadat x order by 1

limit i_elotte to all
if not isvalue(i_elotte 'a')
  then do
    maintain i_elotte add 'a'
    limit i_elotte to 'a'
    i_elotte.nev = 'Minden időpont'
    limit i_elotte to all
  doend

SQL OPEN cr_elotte
SQL FETCH cr_elotte LOOP INTO :APPEND i_elotte -
                              :ASSIGN i_elotte.nev

limit i_elotte remove 'a'
i_elotte.rel = 'a'

SQL CLEANUP

poplevel 'Start'
END

     Hivatkozások:
         I_ELOTTE         I_ELOTTE.NEV     I_ELOTTE.REL

     Hivatkozva a következők által:
         LOAD.DIM

DEFINE I_HELY.L_PROG PROGRAM
LD Hegy dimenziót tölti fel
PROGRAM
variable o text
variable ok id
variable k text
variable kk id

pushlevel 'I_HELY start'
push i_orszag
push i_videk_fovaros
push i_hazai_kulfoldi
push i_hely

SQL DECLARE cr_hely CURSOR FOR select kod, nev, kateg_kod, kateg from r_hely where kod in (select hely from r_iadat)


SQL OPEN cr_hely
SQL FETCH cr_hely INTO :ok :o :kk :k
while SQLCODE eq 0
do
  limit i_orszag to all
  limit i_videk_fovaros to all
  limit i_hazai_kulfoldi to all
  limit i_hely to all
  if not isvalue( i_hazai_kulfoldi k)
    then do
      maintain i_hazai_kulfoldi add k
      limit i_hazai_kulfoldi to k
      i_hazai_kulfoldi.kod = kk
    doend


  if kk eq 'kulf'
    then do
      if not isvalue(i_orszag o)
        then do
          maintain i_orszag add o
          limit i_orszag to o
          i_orszag.kod = ok
          i_hely.rel(i_orszag o) = i_hely(i_hazai_kulfoldi k)
        doend
     doend
```

```
     else do
       if not isvalue(i_videk_fovaros o)
         then do
           maintain i_videk_fovaros add o
           limit i_videk_fovaros to o
           i_videk_fovaros.kod = ok
           i_hely.rel(i_videk_fovaros o) = i_hely(i_hazai_kulfoldi k)
         doend
     doend
  SQL FETCH cr_hely INTO :ok :o :kk :k
doend

poplevel 'I_HELY start'
END
```

```
     Hivatkozások:
         I_HAZAI_KULFOLDI I_HAZAI_KULFOLDI I_HELY           I_HELY.REL
                          .KOD
         I_ORSZAG         I_ORSZAG.KOD     I_VIDEK_FOVAROS  I_VIDEK_FOVAROS.
                                                            KOD

     Hivatkozva a következők által:
         LOAD.DIM
```

```
DEFINE I_IDO.L_PROG PROGRAM
LD 'Internetes eladások idő dimenzióját tölti fel'
PROGRAM
"Deklaráljuk a szükséges változókat
variable d date
variable nap id
variable hh id
variable h id
variable q id
variable yr id
variable temp shortint

variable s text

"Létrehozzuk az adatfeltöltéshez szükséges kurzort
"Azt a dátumot, amelynél régebbi adatok nem szükségesek a d változóban várja
SQL DECLARE cr_ido CURSOR FOR select distinct idopont from r_iadat where idopont > :d

"Elmentjük a jelenlegi állapotot
pushlevel 'Start'
push i_ido

"Vegyük a legutolsó napot
limit i_nap to last 1
"Ha nincs, akkor egy alap dátumot veszünk, aminél biztos nincs korábbi adat.
if statlen(i_nap) lt 1
  then d = to_date('19000101', 'yyyymmdd')
  else d = to_date(i_nap, 'yyyymmdd')

"Megnyitjuk a kurzort
SQL OPEN cr_ido

"Lekérjük az első rekordot, ha van
SQL FETCH cr_ido INTO :d
"Addig folytatjuk ciklusosan, amíg van adat
while SQLCODE eq 0
do
  "Meghatározzuk a nap dimenzió értékét
  nap = to_char(d, 'yyyymmdd')

  "Meghatározzuk a harmadhónap dimenzió értékét
  temp = to_char(d, 'dd')
  temp = intpart(temp / 10)
  if temp ne 3
    then temp = temp + 1
  hh = joinchars( to_char(d, 'yyyymm') 'S' temp)

  "Meghatározzuk a hónap dimenzió értékét
  h = to_char(d, 'yyyy_mm')

  "Meghatározzuk az év dimenzió értékét
  yr = to_char(d, 'yyyy')

  "Meghatározzuk a negyedév dimenzió értékét

  temp = to_char(d, 'mm')
  temp = intpart( (temp -1) / 3) + 1
  q = joinchars( 'Q' temp '_' yr )

  "Az idő dimenziót alapstátuszba állítjuk
  limit i_ido to all

  "Hozzáadjuk a nap dimenzióhoz az új értéket
  maintain i_nap add nap

  "A többi dimenzióhoz csak akkor adunk értéket, ha szükséges.
  if not isvalue(i_harmadhonap, hh)
    then maintain i_harmadhonap add hh
  if not isvalue(i_honap, h)
    then maintain i_honap add h
  if not isvalue(i_negyedev q)
    then maintain i_negyedev add q
  if not isvalue(i_ev yr)
    then maintain i_ev add yr
```

```
  "Beállítjuk a hierarchiákat
  limit i_ido to <i_nap: nap>
  i_ido.rel = i_ido(i_harmadhonap hh)
  limit i_ido to <i_harmadhonap: hh>
  i_ido.rel = i_ido(i_honap h)
  limit i_ido to <i_honap: h>
  i_ido.rel = i_ido(i_negyedev q)
  limit i_ido to <i_negyedev: q>
  i_ido.rel = i_ido(i_ev yr)

  "Lekérjük a következő rekordot
  SQL FETCH cr_ido INTO :d
doend

"Felszabadítjuk a lefoglalt kurzorterületet
SQL CLEANUP


"Visszaállítjuk a ketdeti állapotot
poplevel 'Start'

END
```

```
     Hivatkozások:
         I_EV             I_HARMADHONAP    I_HONAP          I_IDO
         I_IDO.REL        I_NAP            I_NEGYEDEV

     Hivatkozva a következők által:
         LOAD.DIM
```

```
DEFINE I_JEGY_BERLET.L_PROG PROGRAM
PROGRAM
pushlevel 'JEGY_BERLET start'
push i_jegy_berlet

limit i_szallmod to all

"Minden típus
if not isvalue(i_jegy_berlet 'A')
  then do
    maintain i_jegy_berlet add 'A'
    limit i_jegy_berlet to 'A'
    i_jegy_berlet.nev = 'Minden típus'
    limit i_jegy_berlet to all
  doend

"Jegy
if not isvalue(i_jegy_berlet 'J')
  then do
    maintain i_jegy_berlet add 'J'
    limit i_jegy_berlet to 'J'
    i_jegy_berlet.nev = 'Jegy'
    i_jegy_berlet.rel = 'A'
    limit i_jegy_berlet to all
  doend

"Bérlet
if not isvalue(i_jegy_berlet 'B')
  then do
    maintain i_jegy_berlet add 'B'
    limit i_jegy_berlet to 'B'
    i_jegy_berlet.nev = 'Bérlet'
    i_jegy_berlet.rel = 'A'
  doend

poplevel 'JEGY_BERLET start'
END
```

```
     Hivatkozások:
         I_JEGY_BERLET    I_JEGY_BERLET.NE I_JEGY_BERLET.RE
                          V                L

     Hivatkozva a következők által:
         LOAD.DIM
```

```
DEFINE I_MUFAJ.L_PROG PROGRAM
PROGRAM
pushlevel 'MUFAJ start'
push i_mufaj

SQL DECLARE cr_mufaj CURSOR FOR select azon, nev, 'A' from r_mufaj where azon in (select mufaj from r_iadat) order by
azon

limit i_mufaj to all
if not isvalue(i_mufaj 'A')
  then do
    maintain i_mufaj add 'A'
    limit i_mufaj to 'A'
    i_mufaj.nev = 'Minden műfaj'
    limit i_mufaj to all
  doend

SQL OPEN cr_mufaj
SQL FETCH cr_mufaj LOOP INTO :APPEND i_mufaj -
                             :ASSIGN i_mufaj.nev -
                             :ASSIGN i_mufaj.rel
```

```
SQL CLEANUP
poplevel 'MUFAJ start'

END

     Hivatkozások:
         I_MUFAJ          I_MUFAJ.NEV      I_MUFAJ.REL

     Hivatkozva a következők által:
         LOAD.DIM

DEFINE I_SZALL.L_PROG PROGRAM
PROGRAM
pushlevel 'SZALLMOD start'
push i_szall

SQL DECLARE cr_szall CURSOR FOR select azon, nev from r_szall where azon in (select szall from r_iadat)

limit i_szall to all

if not isvalue(i_szall 'a')
  then do
    maintain i_szall add 'a'
    limit i_szall to 'a'
    i_szall.nev = 'Minden szállításmód'
    limit i_szall to all
  doend

SQL OPEN cr_szall

SQL FETCH cr_szall LOOP INTO :APPEND i_szall -
                             :ASSIGN i_szall.nev

limit i_szall remove 'a'
i_szall.rel = 'a'

SQL CLEANUP

poplevel 'SZALLMOD start'
END

     Hivatkozások:
         I_SZALL          I_SZALL.NEV      I_SZALL.REL

     Hivatkozva a következők által:
         LOAD.DIM

DEFINE KERDES.1.REP PROGRAM
PROGRAM
argument dim text
argument htext text
pushlevel 'KERDES1 start'
push C_TULAJ I_ARKAT I_ELOTTE I_EV I_HARMADHONAP -
     I_HAZAI_KULFOLDI I_HELY I_HONAP I_IDO -
     I_JEGY_BERLET I_MUFAJ I_NAP I_NEGYEDEV -
     I_ORSZAG I_SZALL I_VIDEK_FOVAROS

limit C_TULAJ to all
limit I_ELOTTE to all
limit I_HELY to all
limit I_SZALL to all
limit I_JEGY_BERLET to all
limit I_ARKAT to all
limit I_MUFAJ to all
limit C_TULAJ to hierarchy inverted
limit I_ELOTTE to ancestors
limit I_HELY to topancestors
limit I_SZALL to ancestors
limit I_JEGY_BERLET to ancestors
limit I_ARKAT to ancestors
limit I_MUFAJ to ancestors

report heading 'Tulajdonos' w 22 truncate down c_tulaj.nev -
       heading htext across  &dim : -
       heading 'Bevételek évenként' -
       <w 12 decimal 0 heading 'Bevétel' i_osszeg -
        w 7 decimal 0 Heading 'DB' i_jegyszam>

poplevel 'KERDES1 start'
END

     Hivatkozások:
         C_TULAJ          C_TULAJ.NEV      I_ARKAT          I_ELOTTE
         I_EV             I_HARMADHONAP    I_HAZAI_KULFOLDI I_HELY
         I_HONAP          I_IDO            I_JEGY_BERLET    I_JEGYSZAM
         I_MUFAJ          I_NAP            I_NEGYEDEV       I_ORSZAG
         I_OSSZEG         I_SZALL          I_VIDEK_FOVAROS

     Hivatkozva a következők által:
         NONE

DEFINE KERDES.2.REP PROGRAM
PROGRAM
argument time_dim id
variable cf text
variable coeff integer
pushlevel 'KERDES2'
push c_tulaj
```

```
push c_honap
push f_adat

limit c_tulaj to all
limit c_honap to all
limit f_adat to all
limit c_tulaj to hierarchy inverted

for c_tulaj
do
  blank
  coeff = 20 + statlen(&time_dim) * 21
  ROW w coeff under '=' c_tulaj.nev

  ROW under '-' w 20 skip under '-' across &time_dim : w 20 center  c_ev
  for f_adat
  do
    if f_adat eq 'Iarany'
      then do
        cf = 'average'
        coeff = 100
      doend
      else do
        cf = 'sum'
        coeff = 1
      doend
    ROW w 20 f_adat.nev rowtotals across &time_dim: w 20 decimal 2 tConvert( (f_adatok * coeff) &time_dim &cf)
  doend
doend

poplevel 'KERDES2'
END

     Hivatkozások:
         C_EV             C_HONAP          C_TULAJ          C_TULAJ.NEV
         F_ADAT           F_ADAT.NEV       F_ADATOK

     Hivatkozva a következők által:
         NONE

DEFINE KERDES.3.REP PROGRAM
PROGRAM
argument dim text
argument htext text
argument minval integer
vrb f boolean
vrb s integer
vrb fc text

pushlevel 'KERDES3'
push C_TULAJ I_ARKAT I_ELOTTE I_EV I_HARMADHONAP -
     I_HAZAI_KULFOLDI I_HELY I_HONAP I_IDO -
     I_JEGY_BERLET I_MUFAJ I_NAP I_NEGYEDEV -
     I_ORSZAG I_SZALL I_VIDEK_FOVAROS

limit I_IDO to all
limit C_TULAJ to all
limit I_ELOTTE to all
limit I_HELY to all
limit I_SZALL to all
limit I_JEGY_BERLET to all
limit I_ARKAT to all
limit I_MUFAJ to all
limit C_TULAJ to topancestors
limit I_ELOTTE to bottomdescendants
limit I_HELY to topancestors
limit I_SZALL to ancestors
limit I_JEGY_BERLET to bottomdescendants
limit I_ARKAT to ancestors
limit I_MUFAJ to ancestors
limit &dim to all

s = 26 + statlen(&dim) * 11


row under '=' w s center 'Előrevásárlások'
blank
row w 26 skip under '-' w s-26 center htext
row under '-' w 10 center 'Típus' under '-' w 15 center 'Előrevásárlás' across &dim : under '-' center &dim

for I_JEGY_BERLET
do
  f = true
  for i_elotte
  do
    s = 0
    for &dim
    do
      s = s + nafill(i_jegyszam(&dim &dim i_elotte i_elotte i_jegy_berlet i_jegy_berlet), 0)
    doend
    if s ge nafill(minval 0)
    then do
      if f
        then fc = 'w 10 i_jegy_berlet.nev '

        else fc = 'w 10 skip'
      row &fc w 15 right i_elotte.nev across &dim :  w 10 decimal 0 nafill(i_jegyszam 0)
      f = false
```

```
    doend

  doend
  blank
doend

poplevel 'KERDES3'
END

     Hivatkozások:
         C_TULAJ          I_ARKAT          I_ELOTTE         I_ELOTTE.NEV
         I_EV             I_HARMADHONAP    I_HAZAI_KULFOLDI I_HELY
         I_HONAP          I_IDO            I_JEGY_BERLET    I_JEGYSZAM
         I_MUFAJ          I_NAP            I_NEGYEDEV       I_ORSZAG
         I_SZALL          I_VIDEK_FOVAROS

     Hivatkozva a következők által:
         NONE

DEFINE KERDES.4.REP PROGRAM
PROGRAM
argument ido_dim text
argument ido_htext text
argument hely_dim text
argument hely_htext text
vrb f boolean
vrb s integer
vrb fc text

pushlevel 'KERDES4'
push C_TULAJ I_ARKAT I_ELOTTE I_EV I_HARMADHONAP -
     I_HAZAI_KULFOLDI I_HELY I_HONAP I_IDO -
     I_JEGY_BERLET I_MUFAJ I_NAP I_NEGYEDEV -
     I_ORSZAG I_SZALL I_VIDEK_FOVAROS

limit I_IDO to all
limit C_TULAJ to all
limit I_ELOTTE to all
limit I_HELY to all
limit I_SZALL to all
limit I_JEGY_BERLET to all
limit I_ARKAT to all
limit I_MUFAJ to all
limit C_TULAJ to topancestors
limit I_ELOTTE to topancestors
limit I_SZALL to ancestors
limit I_JEGY_BERLET to topancestors
limit I_ARKAT to ancestors
limit I_MUFAJ to ancestors
limit &ido_dim to all
limit &hely_dim to all

report heading hely_htext w 20 down &hely_dim -
       heading ido_htext across &ido_dim : -
       heading 'Bevételek és eladott jegyek lakhely szerinti megoszlása' -
       <heading 'Bevétel' w 12 decimal 0 i_osszeg heading 'Jegy db' w 7 decimal 0 i_jegyszam>

poplevel 'KERDES4'
END

     Hivatkozások:
         C_TULAJ          I_ARKAT          I_ELOTTE         I_EV
         I_HARMADHONAP    I_HAZAI_KULFOLDI I_HELY           I_HONAP
         I_IDO            I_JEGY_BERLET    I_JEGYSZAM       I_MUFAJ
         I_NAP            I_NEGYEDEV       I_ORSZAG         I_OSSZEG
         I_SZALL          I_VIDEK_FOVAROS

     Hivatkozva a következők által:
         NONE

DEFINE KERDES.5.REP PROGRAM
PROGRAM
argument ido_dim text
argument ido_htext text
vrb f boolean
vrb s integer
vrb fc text

pushlevel 'KERDES5'
push C_TULAJ I_ARKAT I_ELOTTE I_EV I_HARMADHONAP -
     I_HAZAI_KULFOLDI I_HELY I_HONAP I_IDO -
     I_JEGY_BERLET I_MUFAJ I_NAP I_NEGYEDEV -
     I_ORSZAG I_SZALL I_VIDEK_FOVAROS

limit I_IDO to all
limit C_TULAJ to all
limit I_ELOTTE to all
limit I_HELY to all
limit I_SZALL to all
limit I_JEGY_BERLET to all
limit I_ARKAT to all
limit I_MUFAJ to all
limit C_TULAJ to topancestors
limit I_HELY to topancestors
limit I_ELOTTE to topancestors
limit I_SZALL to bottomdescendants
limit I_JEGY_BERLET to topancestors
limit I_ARKAT to ancestors
```

```
limit I_MUFAJ to ancestors
limit &ido_dim to all

report heading 'Szállításmód' w 20 truncate down i_szall.nev -
       heading ido_htext across &ido_dim : -
       heading 'Bevételek és eladott jegyek -
szállitási módok szerinti megoszlása' -
       <heading 'Bevétel' w 12 decimal 0 nafill(i_osszeg 0) -
        heading 'Jegy db' w 7 decimal 0 nafill(i_jegyszam 0)>
poplevel 'KERDES5'
END

     Hivatkozások:
         C_TULAJ          I_ARKAT          I_ELOTTE         I_EV
         I_HARMADHONAP    I_HAZAI_KULFOLDI I_HELY           I_HONAP
         I_IDO            I_JEGY_BERLET    I_JEGYSZAM       I_MUFAJ
         I_NAP            I_NEGYEDEV       I_ORSZAG         I_OSSZEG
         I_SZALL          I_SZALL.NEV      I_VIDEK_FOVAROS

     Hivatkozva a következők által:
         NONE

DEFINE KERDES.6.REP PROGRAM
PROGRAM
argument dim text
argument htext text
argument minval integer
vrb f boolean
vrb s integer
vrb fc text

pushlevel 'KERDES6'
push C_TULAJ I_ARKAT I_ELOTTE I_EV I_HARMADHONAP -
     I_HAZAI_KULFOLDI I_HELY I_HONAP I_IDO -
     I_JEGY_BERLET I_MUFAJ I_NAP I_NEGYEDEV -
     I_ORSZAG I_SZALL I_VIDEK_FOVAROS

limit I_IDO to all
limit C_TULAJ to all
limit I_ELOTTE to all
limit I_HELY to all
limit I_SZALL to all
limit I_JEGY_BERLET to all
limit I_ARKAT to all
limit I_MUFAJ to all
limit C_TULAJ to topancestors
limit I_ELOTTE to topancestors
limit I_HELY to topancestors
limit I_SZALL to ancestors
limit I_JEGY_BERLET to bottomdescendants
limit I_ARKAT to bottomdescendants
limit I_MUFAJ to ancestors
limit &dim to all

s = 26 + statlen(&dim) * 11


row under '=' w s center 'Árkategóriák szerinti megoszlás'
blank
row w 26 skip under '-' w s-26 center htext
row under '-' w 10 center 'Típus' under '-' w 15 center 'Áakategória' across &dim : under '-' center &dim

for I_JEGY_BERLET
do
  f = true
  for i_arkat
  do
    if f
      then fc = 'w 10 i_jegy_berlet.nev '
      else fc = 'w 10 skip'
    row &fc w 15 right i_arkat.nev across &dim :  w 10 decimal 0 nafill(i_jegyszam 0)
    f = false
  doend
  blank
doend

poplevel 'KERDES6'
END

     Hivatkozások:
         C_TULAJ          I_ARKAT          I_ARKAT.NEV      I_ELOTTE
         I_EV             I_HARMADHONAP    I_HAZAI_KULFOLDI I_HELY
         I_HONAP          I_IDO            I_JEGY_BERLET    I_JEGYSZAM
         I_MUFAJ          I_NAP            I_NEGYEDEV       I_ORSZAG
         I_SZALL          I_VIDEK_FOVAROS

     Hivatkozva a következők által:
         NONE

DEFINE KERDES.7.REP PROGRAM
PROGRAM
argument ido_dim text
argument ido_htext text
vrb f boolean
vrb s integer
vrb fc text

pushlevel 'KERDES7'
```

```
push C_TULAJ I_ARKAT I_ELOTTE I_EV I_HARMADHONAP -
     I_HAZAI_KULFOLDI I_HELY I_HONAP I_IDO -
     I_JEGY_BERLET I_MUFAJ I_NAP I_NEGYEDEV -
     I_ORSZAG I_SZALL I_VIDEK_FOVAROS

limit I_IDO to all
limit C_TULAJ to all
limit I_ELOTTE to all
limit I_HELY to all
limit I_SZALL to all
limit I_JEGY_BERLET to all
limit I_ARKAT to all
limit I_MUFAJ to all
limit C_TULAJ to topancestors
limit I_HELY to topancestors
limit I_ELOTTE to topancestors
limit I_SZALL to topancestors
limit I_JEGY_BERLET to topancestors
limit I_ARKAT to ancestors
limit I_MUFAJ to bottomdescendants
limit &ido_dim to all

report heading 'Műfaj' w 20 truncate down i_mufaj.nev heading 'Előre' w 10 truncate i_elotte.nev -
       heading ido_htext across &ido_dim : -
       heading 'Bevételek és eladott jegyek műfajok szerinti megoszlása' -
       <heading 'Bevétel' w 12 decimal 0 nafill(i_osszeg 0) -
        heading 'Jegy db' w 7 decimal 0 nafill(i_jegyszam 0)>
poplevel 'KERDES7'
END

     Hivatkozások:
         C_TULAJ          I_ARKAT          I_ELOTTE         I_ELOTTE.NEV
         I_EV             I_HARMADHONAP    I_HAZAI_KULFOLDI I_HELY
         I_HONAP          I_IDO            I_JEGY_BERLET    I_JEGYSZAM
         I_MUFAJ          I_MUFAJ.NEV      I_NAP            I_NEGYEDEV
         I_ORSZAG         I_OSSZEG         I_SZALL          I_VIDEK_FOVAROS

     Hivatkozva a következők által:
         NONE

DEFINE KERDES.8.REP PROGRAM
PROGRAM
argument ido_dim text
argument ido_htext text
argument hiba_dim text
argument hiba_htext text
vrb f boolean
vrb s integer
vrb fc text

pushlevel 'KERDES8'
push C_HONAP C_NEGYEDEV C_EV C_EVAD T_HIBACSOPORT T_SIKERES_SIKERTELEN T_EREDMENY

limit C_HONAP to all
limit C_NEGYEDEV to all
limit C_EV to all
limit C_EVAD to all
limit T_HIBACSOPORT to all
limit T_SIKERES_SIKERTELEN to all

report heading hiba_htext w 20 down &hiba_dim -
       heading ido_htext across &ido_dim : -
       heading 'Vásárlások hibák szerinti megoszlása' -
       <heading 'Vásárlásszám' w 12 decimal 0 tconvert(t_adatok &ido_dim SUM) >

poplevel 'KERDES8'
END

     Hivatkozások:
         C_EV             C_EVAD           C_HONAP          C_NEGYEDEV
         T_ADATOK         T_EREDMENY       T_HIBACSOPORT    T_SIKERES_SIKERT
                                                            ELEN

     Hivatkozva a következők által:
         NONE

DEFINE LOAD.DIM PROGRAM
PROGRAM
show 'I_IDO dimenzió'
I_IDO.L_PROG

show 'C_TULAJ dimenzió'
C_TULAJ.L_PROG

show 'I_ELOTTE dimenzió'
I_ELOTTE.L_PROG

show 'I_HELY dimenzió'
I_HELY.L_PROG

show 'I_SZALL dimenzió'
I_SZALL.L_PROG

show 'I_JEGY_BERLET dimenzió'
I_JEGY_BERLET.L_PROG

show 'I_ARKAT dimenzió'
I_ARKAT.L_PROG
```

```
show 'I_MUFAJ dimenzió'
I_MUFAJ.L_PROG

show 'C_HONAP dimenzió'
C_HONAP.L_PROG

show 'T_EREDMENY dimenzió'
T_EREDMENY.L_PROG

SQL CLEANUP

show 'Dimenziók betöltése kész'
END

     Hivatkozások:
         C_HONAP.L_PROG   C_TULAJ.L_PROG   I_ARKAT.L_PROG   I_ELOTTE.L_PROG
         I_HELY.L_PROG    I_IDO.L_PROG     I_JEGY_BERLET.L_ I_MUFAJ.L_PROG
                                           PROG
         I_SZALL.L_PROG   T_EREDMENY.L_PRO
                          G

     Hivatkozva a következők által:
         NONE

DEFINE PRE.KERDES9.LIM PROGRAM
PROGRAM
allstat
limit C_TULAJ to ancestors
limit I_ELOTTE to ancestors
limit I_HELY to topancestors
limit I_SZALL to ancestors
limit I_JEGY_BERLET to bottomdescendants
limit I_ARKAT to ancestors
limit I_MUFAJ to ancestors
END

     Hivatkozások:
         C_TULAJ          I_ARKAT          I_ELOTTE         I_HELY
         I_JEGY_BERLET    I_MUFAJ          I_SZALL

     Hivatkozva a következők által:
         NONE

DEFINE T_ADATOK.LOAD PROGRAM
PROGRAM
vrb d date
vrb h integer
vrb c integer


SQL DECLARE cr_tadat CURSOR FOR -
              select rt.idopont, rh.csop, sum(db) -
              from r_tadat rt, r_hiba rh -
              where rt.hiba = rh.kod -
              group by rt.idopont, rh.csop -
              order by rt.idopont, rh.csop

pushlevel 'T_ADATOK start'
push C_EV C_EVAD C_HONAP C_NEGYEDEV C_TULAJ F_ADAT I_ARKAT I_ELOTTE I_EV I_HARMADHONAP -
     I_HAZAI_KULFOLDI I_HELY I_HONAP I_IDO I_JEGY_BERLET I_MUFAJ I_NAP I_NEGYEDEV -
     I_ORSZAG I_SZALL I_VIDEK_FOVAROS T_EREDMENY T_HIBACSOPORT T_SIKERES_SIKERTELEN

ALLSTAT

SQL OPEN cr_tadat

SQL FETCH cr_tadat INTO :d :h :c
while SQLCODE eq 0
do
  show joinchars(d ' ' h ' ' c)
  limit c_honap to to_char(d, 'yyyy mon dd')
  if statlen( c_honap) ne 1
    then show 'Hiba hónap'
  limit t_eredmeny to t_eredmeny(t_hibacsoport h)
  t_adatok = c
  SQL FETCH cr_tadat INTO :d :h :c
doend

SQL CLEANUP

ALLSTAT

aggregate T_ADATOK using T_ADATOK.agg

poplevel 'T_ADATOK start'
END

     Hivatkozások:
         C_EV             C_EVAD           C_HONAP          C_NEGYEDEV
         C_TULAJ          F_ADAT           I_ARKAT          I_ELOTTE
         I_EV             I_HARMADHONAP    I_HAZAI_KULFOLDI I_HELY
         I_HONAP          I_IDO            I_JEGY_BERLET    I_MUFAJ
         I_NAP            I_NEGYEDEV       I_ORSZAG         I_SZALL
         I_VIDEK_FOVAROS  T_ADATOK         T_ADATOK.AGG     T_EREDMENY
         T_HIBACSOPORT    T_SIKERES_SIKERT
                          ELEN
```

```
     Hivatkozva a következők által:
         NONE

DEFINE T_EREDMENY.L_PROG PROGRAM
PROGRAM
variable g id
variable gn text
variable x id
SQL DECLARE cr_ered CURSOR FOR -
   select distinct csop, csop_nev -
   from r_hiba where kod in (select hiba from r_tadat) order by csop

pushlevel 'EREDMENY start'

limit t_hibacsoport to all
limit t_sikeres_sikertelen to all
limit t_eredmeny to all

if not isvalue( t_sikeres_sikertelen 'ok')
  then do
    maintain t_sikeres_sikertelen add 'ok'
    limit t_sikeres_sikertelen to 'ok'
    t_sikeres_sikertelen.nev = 'Sikeres'
    limit t_sikeres_sikertelen to all
  doend

if not isvalue( t_sikeres_sikertelen 'fail')
  then do
    maintain t_sikeres_sikertelen add 'fail'
    limit t_sikeres_sikertelen to 'fail'
    t_sikeres_sikertelen.nev = 'Sikertelen'
    limit t_sikeres_sikertelen to all
  doend

SQL OPEN cr_ered
SQL FETCH cr_ered INTO :g :gn
while SQLCODE eq 0
do
if not isvalue( t_hibacsoport g)
  then do
    maintain t_hibacsoport add g
    limit t_hibacsoport to g
    t_hibacsoport.nev = gn
    if g eq '1'
      then x = 'ok'
      else x = 'fail'
    t_eredmeny.rel(t_hibacsoport g) = t_eredmeny( t_sikeres_sikertelen x)
    limit t_hibacsoport to all
  doend
  SQL FETCH cr_ered INTO :g :gn
doend

SQL CLEANUP

poplevel 'EREDMENY start'
END

     Hivatkozások:

         T_EREDMENY       T_EREDMENY.REL   T_HIBACSOPORT    T_HIBACSOPORT.NE
                                                            V
         T_SIKERES_SIKERT T_SIKERES_SIKERT
         ELEN             ELEN.NEV

     Hivatkozva a következők által:
         LOAD.DIM

DEFINE F_ADATOK.MODEL MODEL
LD 'Forgalmi adatok számítását leíró modell'
MODEL
dimension f_adat
Osszes = J + I
Iarany = I / (if NAFill(Osszes,0) eq 0 then 1 else Osszes)
Jut = I * .1
Netto = Jut / 1.2
END

     Hivatkozások:
         F_ADAT

     Hivatkozva a következők által:
         F_ADATOK.LOAD
```