Debreceni Egyetem
Informatika Kar

Az SQL és implementációi

avagy

adatbázis-függetlenség a gyakorlatban

Témavezető:
Dr. Juhász István
egyetemi adjunktus

Készítette:
Nagy László
programtervező matematikus

Debrecen
2008

Tartalomjegyzék

1. fejezet

Bevezetés

Ahogyan az információs társadalom nem képzelhető el adattárolás nélkül, úgy az adattárolás sem lehet meg az adatbázisok nélkül. Napjaink adatbázis-kezelő rendszerei az informatika rohamos fejlődése ellenére évtizedek óta a Codd által megálmodott relációs modellen alapulnak és a modell kezelésére kifejlesztett szabványos SQL nyelvet használják. A különböző technológiák akkor maradnak életképesek, ha sikerül szabványosítani őket. Egy technológiának útban a szabványosítás felé sok, különböző megvalósításokkal rendelkező versenytárssal kell megküzdenie, miközben igazán kiforrottá és érdemessé válik a szabványosításra. Ez természetesen nem meglepő, hiszen minden pontosan a piac törvényeinek megfelelően történik. Az azonban meglepő lehet, hogy egy szabványosított technológiának miért lehetnek különböző, egymással nem teljesen kompatibilis implementációi. A gépiparban például nehezen képzelhető el, hogy az egyik cég által gyártott 8-as csavarhoz nem passzol az anya csak azért, mert azt egy másik cég más menetemelkedéssel gyártotta le, hiszen a szabvány belépő szintje ezt az előírást nem tartalmazza. A csavargyártást az SQL utasítással összehasonlítani nyílván nem szerencsés, de azt kijelenthetjük, hogy a szabványoknak való maradéktalan megfelelés csak előnyünkre válhat.

Dolgozatomban be szeretném mutatni, hogy az SQL szabvány esetén milyen jellegű implementációs különbségekkel kell számolni. Abban az esetben, ha minden implementáció mereven ragaszkodna a szabványban foglaltakhoz, nyilván beszélhetnénk adatbázis független adatbázis kezelésről. Mivel ez az eset nem áll fent, célszerűnek tűnik feltenni a kérdést, hogy az implementációs különbségek ellenére megvalósítható-e. A nyelvi függetlenséget biztosító megoldások keresése alatt nem feltétlenül azt értem, hogy rendelkezésre állnak-e a komplett SQL utasításkészlet fordítását egyik nyelvjárásról a másikra végző szoftver komponensek. Már

az is megfelelő lehet, ha egy-egy alkalmazási területen megoldást nyújtanának az ilyen jellegű problémáinkra.

Azonban az adatbázis függetlenséget nem csak nyelvi aspektusból kell megvizsgálni. A gyakorlatban még az első SQL utasítás kiadása előtt fel fog merülni az a kérdés, hogy milyen módon hozzuk létre az alkalmazásunk és az adatbázis-kezelő rendszer közötti kapcsolatot. Az adatbázis-kezelő rendszerek túlnyomó többsége ad valamilyen programozói interfészt – általában valamilyen statikusan vagy dinamikusan linkelhető programozói könyvtár formájában –, de ezeket csak abban az esetben használhatjuk, ha nem célunk az adatbázis-kezelő rendszertől való függetlenség. A dolgozatom szempontjából természetesen az, ezért más megoldások után kell nézni. A független adatbázis elérés iránti igény nem új keletű, több megvalósítás is ismert. Ezek többsége szabványosnak vagy kvázi szabványosnak mondható valamely platform vagy programozási nyelv tekintetében. A dolgozatban ezen megvalósítások közül is megvizsgálok néhányat.

2. fejezet

Az adatbáziskezelés története

Az adatbázis kezelő alkalmazások iránti igény viszonylag hamar felmerült az informatika történetében. Az első adatbázis kezelő alkalmazások esetén még igazából csak adattároló rendszerekről beszélhetünk, hiszen minden alkalmazásnak saját maga kellett megoldania az adatok tárolásának problémáját. Ugyan a különböző állományszervezési megoldások között lehettek hasonlóságok, azonban az adatok leképezése teljes mértékben alkalmazásfüggő volt. Az egyik alkalmazás által készített adatállományok egy másik alkalmazás számára ismeretlenek voltak, ezt úgy is fogalmazhatnám, hogy az alkalmazások az adatkezelés szempontjából teljesen inkompatibilisek voltak. Valójában az adattárolás beli eltéréseken túl több, talán sokkal komolyabb probléma is felmerült. Egyik probléma, hogy az új alkalmazások készítőinek az alapoktól kellett a fejlesztéseket készíteniük, tehát a helyett, hogy az üzleti folyamatok leképezésével foglalkoztak volna az adattárolást kellett először megvalósítaniuk minden egyes alkalommal. Abban az esetben ha később, az üzleti modelljük megváltozott, akkor az nem csak az alkalmazást érintette, hanem az adatkezelő alrendszerüket is. Ez a másik irányból nézve is problémát okozhatott. Ha egy alkalmazást felkészítettek más alkalmazások adatállományainak kezelésére, akkor az adatállományok szerkezeti változásait csak az alkalmazás módosításával lehetett kezelni. Újabb problémaként merült fel a jogosultságkezelés, melyre az állománykezelő rendszerek ugyan adtak valamiféle választ, de ezt semmiképpen nem lehetett kielégítőnek nevezni. Nem is beszélve az egyszerre több felhasználót kiszolgáló rendszerekről, ahol a konkurens hozzáférést az állománykezelő rendszer nem biztosította. A problémák megoldására a CODASYL(Conference on Data System Languages)-on belül létrejött DBTG(Database Task Group) megfogalmazta a modern adatbázis kezelőkkel szemben támasztott követelményeit. Ezek a következők voltak:

- összetett logikai adatszerkezetek kezelése
- irányított redundancia
- konkurens hozzáférés biztosítása
- többféle elérési mód egy időben
- programozási nyelvek támogatása
- emberi logika támogatása (imperatív helyett deklaratív)
- adatmodell szemléletet valósítson meg(a jogosultság kezelés szempontjából is fontos)
- legyen visszaállítható
- adat és program függetlenség logikai és fizikai szinten
  - Logikai szint: az adatok szerkezetének megváltozása a programszerkezettől független legyen
  - Fizikai szint: egy adott program tudja feldolgozni az adathalmazt annak fizikai helyétől függetlenül

Ezeknek az elveknek a megszületésétől és az első implementációk megjelenésétől beszélhetünk adatbázis-kezelő rendszerekről. Mivel az egyes adatbázis-kezelő rendszerek mindíg valamely adatmodellel vannak szoros kapcsolatban, ezért a következőkben röviden áttekintem azok kapcsolatát.

2.1. A hierarchikus modellen alapuló adatbázis kezelők

A hierarchikus volt a legelső és egyben a leginkább korlátozott modell, melyre az adatbázis-kezelőkben megjelent. Például az IBM IMS adatbázis-kezelő rendszere alkalmazta ezt a modellt. A neve is utal rá, hogy az adatok hierarchikusan voltak elrendezve. Ezt egy fa szerkezettel tehetjük szemléletessé. Az adatbázis több egymástól független fából állhatott. A fa csomópontjaiban és leveleiben helyezkednek el az adatok. A közöttük lévő kapcsolat a szülő-gyermek kapcsolatnak felelt meg. Segítségével közvetlenül csak az 1:N típusú kapcsolatok képezhetők le(Az 1:N kapcsolat azt jelenti, hogy az adatszerkezet egyik típusú adata a hierarchiában alatta elhelyezkedő egy vagy több más adattal áll kapcsolatban). A hierarchikus modell természetéből adódóan explicit módon nem ábrázolható benne M:N típusú kapcsol, de ezt az előfordulások ismétlésével feloldhatták. Emellett további hátránya volt, hogy az adatok elérése csak egyféle sorrendben volt lehetséges, a tárolt hierarchiának megfelelő sorrendben. A hierarchikus adatmodell alkalmazására a legkézenfekvőbb példa a családfa, vagy a főnök-beosztott viszony.

2.2. A hálós modellen alapuló adatbázis kezelők

A CODASYL által létrehozott DBTG 1971-ben nyilvánosságra hoz egy, akkor még forradalmi adatbázis elvet (a hálós modell filozófiáját), amelyben összetettebb adatszerkezeteket is definiálni lehet.

A hálós adatmodell esetén az egyes azonos vagy különböző összetételű adategységek (rekordok) között a kapcsolat egy gráffal volt leírható. A gráf csomópontok és ezeket összekötő élek rendszere, melyben tetszőleges két csomópont között akkor van adatkapcsolat, ha őket él köti össze egymással. Egy csomópontból tetszőleges számú él indulhat ki, de egy él csak két csomópontot köthet össze. Azaz minden adategység tetszőleges más adategységekkel lehetett kapcsolatban. Ebben a modellben már az M:N típusú adatkapcsolatok is leírhatók voltak az 1:N típusúak mellett. A hierarchikus – és a hálós modell – esetén az adatbázisba fixen beépített kapcsolatok következtében csak a tárolt kapcsolatok segítségével bejárható adat-visszakereséseket lehetett hatékonyan megoldani. Alkalmazására példa az iskolai tanár-diák viszony. Minden diákot több tanár tanít, és minden tanár több diákot tanít.

Ilyen szemléletű adatbázis-kezelő rendszerek voltak az ADABAS, DBMS(DEC), IDMS(IBM), és a Honywell IDS-2 rendszere. A DBTG a COBOL nyelv gazdanyelvként való használatát javasolta. A COBOLT hamar felváltotta a PL/1, majd az interaktív feldolgozási igény térnyerésével – a kötegelt adatfeldolgozással szemben –, az ezt kihasználó felhasználói felületek is megjelentek.

2.3. A relációs modellen alapuló adatbázis kezelők

1969-ben Edgar F. Codd, az IBM munkatársaként kidolgozta a mai napig népszerű logikai modelljét, de csak 1970-ben publikálta. Téved azonban az, aki azt hiszi, hogy a relációs adatbázis sikertörténetnek indult. Magával az adatbázis logikai szerkezetével semmi baj nem volt, kísértetiesen hasonlít egy közönséges táblázatkezeléshez, csakhogy a lekérdezéssel gondok adódtak. A lekérdezés nyelve ugyanis halmazelméleti és logikai ismeretek igényelt – ami még a számítógép-programozásnál is mostohább feltételnek tűnt a felhasználók szemében. Maga az IBM sem fűzött hozzá komolyabb reményeket és csak hat év alatt hozták létre az első modern adatbázis-kezelőt, a System-R (ma DB/2) kódnevű szoftvert.

A reláció nem más, mint egy táblázat, a táblázat soraiban tárolt adatokkal együtt. A relációs

adatbázis pedig relációk és csak relációk összessége. A relációk oszlopaiban azonos mennyiségre vonatkozó adatok jelennek meg. Az oszlopok névvel rendelkeznek, melyeknek a reláción belül egyedieknek kell lenniük, de más relációk tartalmazhatnak azonos nevű oszlopokat. A reláció soraiban tároljuk a logikailag összetartozó adatokat. A reláció sorainak sorrendje közömbös, de nem tartalmazhat két azonos adatokkal kitöltött sort. Egy sor és oszlop metszésében található táblázat elemet mezőnek nevezzük, a mezők tartalmazzák az adatokat. A mezők oszloponként különböző típusúak (numerikus, szöveges stb.) lehetnek. A reláció helyett sokszor a tábla vagy táblázat, a sor helyett a rekord, az oszlop helyett pedig az attribútum elnevezés is használatos. A relációktól általában megköveteljük, hogy ne tartalmazzanak más adatokból levezethető vagy kiszámítható információkat. A relációs modell előnyei a következők:

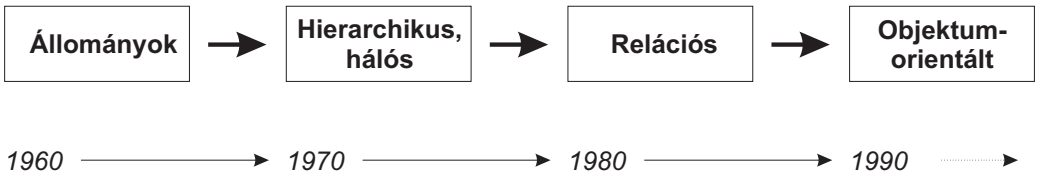
- a relációs adatszerkezet egyszerűen értelmezhető a felhasználók és az alkalmazás készítői számára is, így ez lehet közöttük a kommunikáció eszköze
- a logikai adatmodell relációi egy relációs adatbázis-kezelő rendszerbe módosítások nélkül átvihető
- a relációs modellben az adatbázis tervezés a normál formák bevezetésével egzakt módon elvégezhető

2.4. A relációs modellen túl

Az adatbázis-kezelő rendszerek fejlődése nem állt meg, még akkor sem, ha a mai napig legtöbbjük a relációs modellen alapul. Már 1986-ban megjelentek az első objektumorientált elveket is valló objektum-relációs adatbázis-kezelő rendszerek, majd később az objektumorientáltak is. Az ezekre való igény folyamatosan nő, köszönhetően az objektumorientált technológiák és nyelvek népszerűségének. A ma már oly népszerű XML dokumentumok tárolására is inkább az objektumorientált irányba elmozduló adatbázis-kezelő rendszerek az alkalmasabbak.

2.5. Az adatbázis-kezelők fejlődése

Az egyes adatbázis-kezelő rendszerek az adatkezelés különböző aspektusaira koncentráltak. Míg a hálós és hierarchikus modellen alapulók például elsősorban a helyes adatszerkezetek



2.1. ábra. Az adatkezelés generációi

kialakítására, a valós kapcsolatok fizikai leképezésére koncentráltak, addig a relációs adatbázis-kezelők már nagy hangsúlyt fektettek az adatok hatékony elérésére. Az objektumorientáltaknál pedig az osztályok, objektumok leképezésének és viselkedésének is fontos szerepe van.

Modell	Adatbázis-kezelő rendszer
hierarchikus	DBOMP, IMS, BTRIEVE, DL/I
hálós	IDS,IDMS, DMS, DBMS
relációs	System R, SQL/DS, DB2, dBase IV., IMS/DC, DATAFLEX,RBASE, K-Man, INGRES, Borland PARADOX, ADABAS, SYBASE, FoxPro, Informix, Access
objektumrelációs	Postgres, Iris, Montage,Starburst, UniSQL, Persistence, Algres, Xidak, Orion, Ingres, Sybase, ORACLE
objektumorientált	C++: ONTOS, ObjectStore, VERSANT, POET SmallTalk: GemStone, OAL CommonLisp: ORION, ITASCA

Néhány adatbázis-kezelő rendszer és az általuk támogatott adatmodellek kapcsolata látható az előző táblázatban. Természetesen az egyes adatbázis-kezelő rendszerek az idők folyamán fejlődtek, ezért van az, hogy például az ORACLE hasonló nevű adatbázis-kezelő rendszerét a relációsként ismerhettük meg, de ma már a bevezetett objektum típusoknak köszönhetően inkább objektum-relációsnak nevezhető.

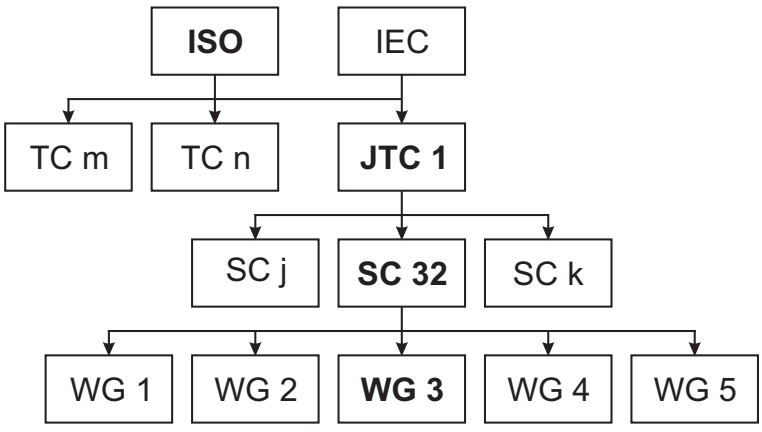
3. fejezet

A szabványos SQL

Napjaink kedvelt adatbázis-kezelő rendszerei a relációs modellen alapulnak. Ezen adatbázisok kezelésére kifejlesztett nyelvek közül kétségkívül az ANSI és az ISO által szabványosított SQL az egyeduralkodó. Ebben a fejezetben röviden áttekintjük a szabványosító szervezeteket, a szabvány életciklusát, majd magának az SQL szabványnak a fejlődését. Az SQL szabvány ismertetésére természetesen nem vállalkozhatom, hiszen a maga néhány ezer oldalas terjedelmével jó néhány dolgozat alapjául is szolgálhatna.

3.1. Adatbázis szabványosító szervezetek

3.1.1. Az ISO és az IEC



3.1. ábra. Az ISO szervezet

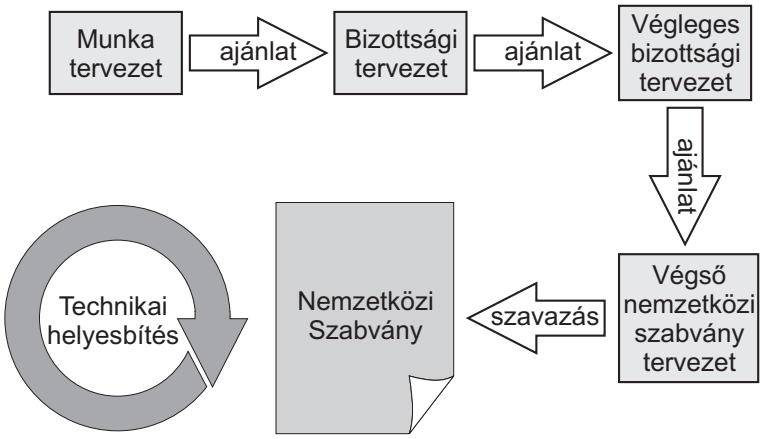
Az 1947 február 23-án alakult ISO (International Organization for Standardization)

nemzetközi szövetsége több mint 100 nemzetnek. Az 1906-ban alapított IEC(International Elektrotechnika Comission) csatlakozott az ISO-hoz és ma is az elektronika területe tartozik hozzá. Tagszervezeteinek száma megközelíti a 100-at, melyeknek kb. 70%-a kormányzati intézmény. Már több mint 10000 szabvány kiadása fűződik a nevükhöz. Műszaki tevékenységét a műszaki bizottságok (TC) végzik. A munkák előkészítése albizottságokban(SC) és munkacsoportokban(WG) folyik. Amint a 3.1. ábrán látható az ISO és az IEC a közös JTC 1 bizottságon keresztül kapcsolódik egymáshoz.

3.1.2. ANSI

Az ANSI-t 1918. október 19-én alapította az Amerikai Műszaki Szabványok Tanácsa (American Engineering Standards Committee), és 1928-ban szervezték át amerikai szabványok társaságára (American Standards Association, ASA). 1966-ban a szervezet új neve United States of American Standards Institute lett, majd 1969-ben kapta a manapság is használatos nevét. Az ANSI (American National Standards Institute) egy nonprofit magánvállalkozás, melynek célja ipari szabványok kidolgozása az Amerikai Egyesült Államok számára. Az ANSI tagja a Nemzetközi Szabványügyi Szervezetnek (ISO) és a Nemzetközi Elektrotechnikai Tanácsnak (IEC)

3.2. Egy szabvány életciklusa



3.2. ábra. Az ISO szabványok életciklusa

Minden szabványosítási folyamatra jellemző, hogy rendelkezik egy életgörbével. Egy szab-

vány általában több év alatt születik meg. Bizonyos esetekben egy már meglévő technológia válik olyan sikeressé, hogy a gyártók vagy alkalmazók kívánják azt szabványosítani. Más esetekben egy még elméleti megoldás (vagy esetleg igény) kerül szabványosításra. Egy szabvány élettartama alatt több alkalommal kiegészíthetik, bővíthetik, de eljön az az idő, amikor már gátolja az újabb technológiák elterjedését. Ekkor történik meg, hogy egy újabb szabvány leváltja.

3.3. Az SQL szabvány fejlődése

SQL 86

- A kiindulási alap az IBM SQL implementációja volt, melynek a szabvány csak egy részhalmaza lett.
- A közös jellemzők és az ortogonalitás hiányát kritizálták
- Három lehetőséget definiáltak a DML utasítások feldolgozására
  - közvetlen feldolgozás
  - modul nyelv
  - beágyazott SQL
- Cobol, Fortran, Pascal és PL/1 binding

SQL 89

- az SQL/86 szuperhalmaza
- a C és az ADA nyelvek hozzá lettek adva a meglévő nyelvi kötésekhez
- a DDL megjelenése séma definíciós nyelvként (`CREATE TABLE`, `CREATE VIEW`, `CREATE PRIVILEGES`)
- Integritási megszorítások megjelenése (`DEFAULT`, `UNIQUE`, `NOT NULL`, `WITH CHECK OPTION`, `PRIMARY KEY`, `CHECK` megszorítások, hivatkozási integritás)

SQL 92

- az SQL/89 szuperhalmaza(néhány módosítással, amit egy mellékletben dokumentáltak)
- a mérete jelentősen megnőtt (120-ról 579 oldalra)
  - nagyobb ortogonalitás

  - adattípus kiterjesztések (varchar, bit, character sets, date, time, interval)
  - többszörös join
  - származtatott táblák a from záradékban
  - átmeneti táblák
  - dinamikus sql
  - kurzorok
- sok – de nem minden – jellemző létező termékben már jelen van
- a szabványt három részre osztják fel
  - belépő szint: majdnem az, mint az SQL/89 az integritási megszorításokkal
  - közép
  - teljes

SQL 99

- a fejlesztés még az SQL/92 kiadása előtt megkezdődött
- az ANSI és az ISO is szabványosította
- az SQL/92 szuperhalmaza (felfele teljesen kompatibilis)
- a mérete jelentősen megnőtt (120-ról 579 oldalra)
  - objektum-relációs kiterjesztés
    - felhasználó által definiált adattípusok
    - referencia típus
    - kollekció típusok (pl.: tömb)
    - nagy méretű objektumok (`LOB`) támogatása
    - tábla hierarchia
  - triggerek
  - tárolt és felhasználó által definiált funkciók
  - OLAP kiterjesztés (`CUBE` és `ROLLUP`)
  - procedurális SQL szerkezet
  - mentési pontok
  - kifejezések az `ORDER BY` záradékban
  - `UPDATE` összekapcsolt táblákra

SQL 2003

- Az SQL 99-hez képest kiegészítésre kerül egy új résszel: SQL/XML
- Az SQL 99 ki lett egészítve a következőkkel:
  - Előredefiniált adattípusok
  - Tipus konstruktorok
  - Skalár és tábla kifejezések
  - Predikátumok
- Új adattípusok
  - `BIGINT` (`BIGINT` >= `INTEGER` >= `SMALLINT`)
  - `MULTISET`
- Meglévő adattípusok törlése
  - `BIT`
  - `BIT VARYING`
- Új séma objektumok
  - Szekvencia generátor
- Új DML utasítások
  - `MERGE` utasítás
  - `SET COLLATION` / `SET NO COLLATION` utasítások
  - Többszörös utasítás hozzárendelés
- Beágyazott `SAVEPOINT`

4. fejezet

SQL implementációk

Ebben a fejezetben öt népszerű adatbázis-kezelő rendszert SQL nyelvjárásainak az összehasonlítását és a szabványtól való eltérését tekintem át. Ez az összehasonlítás már csak a szabvány mérete miatt sem lehet teljes körű. Az összehasonlításban elsősorban azokra az érdekes eltérésekre térek ki, melyek vagy a mindennapi gyakorlatban előfordulnak, vagy amelyekben éppen a nyelvfüggetlenség megvalósíthatóságának kisebb-nagyobb akadályát látom.

Az összehasonlításban szereplő adatbázis-kezelő rendszerek

Az összehasonlításban szereplő adatbázis-kezelő rendszerek mindeggyike ingyenes. Vagy nyílt forrású, vagy valamely kereskedelmi termék ingyenes, és korlátozott képességű verziója. Természetesen a korlátozás nem az SQL nyelv megvalósításban jelentkezik, ezért az itt leírtak a kereskedelmi verziókra is vonatkoznak.

Adatbázis-kezelő	Verzió
PostgreSQL	PostgreSQL 8.2.0
DB2	DB2 Express-C v 9.1
MSSQL	MS SQL Server
MySQL	MySQL 5.0.18
Oracle	Oracle 10g Express Edition

Módosítható nézetek

Az SQL lehetőséget biztosít arra, hogy egy vagy több úgynevezett fizikai táblából logikai táblát, nézettáblát hozzunk létre. A nézettábla önmagában nem tárol adatokat, csak egy `SELECT`

utasítást. Gyakorlati jelentősége abban van, hogy segítségével a gyakran használt lekérdezések eltárolhatók, illetve az összetett lekérdezések egyszerűbben kezelhetők. Előnyeként említhetjük meg azt is, hogy segítségével szabályozható az adatokhoz való hozzáférés, hiszen a nézetben csak a benne hivatkozott oszlopok (és sorok) láthatók. `SELECT` utasításban ugyan úgy használható, mint egy hagyományos adattábla. Az `INSERT` és `UPDATE` használata már nem ilyen egyszerű. A szabvány szerint a nézetek frissíthetők.

SQL:2003 Bonyolult szabályokat határoz meg, hogy egy nézet mikor frissíthető

SQL:92 Korlátozóbb volt, a frissített nézetek csak egy alaptáblából származhattak

PostgreSQL	vannak nézetek, de nem felel meg a szabványnak, mert nem engedi a nézet frissítését
DB2	az SQL:92-nek megfelel
MSSQL	az SQL:92-nek megfelel
MySQL	az SQL:92-nek megfelel
Oracle	az SQL:92-nek megfelel

Tábla összekapcsolás

Az adattáblák közötti kapcsolat kialakítása jellemzően a kulcsok és külső kulcsok segítségével történik. Az SQL szabvány több összekapcsolás típust is leír. A gyakorlatban ezek közül a belső és a külső összekapcsolásokra van igazából szükség, a természetes összekapcsolás és a kereszt-összekapcsolás használata ritkának mondható.

```
SELECT tábla1.oszlop, tábla2.oszlop FROM tábla1
[NATURAL JOIN tábla2] |[JOIN tábla2 USING (oszlopnév)] |
[JOIN tábla2 ON (tábla1.oszlopnév = tábla2.oszlopnév)]|
[LEFT|RIGHT|FULL OUTER JOIN tábla2
ON (tábla1.oszlopnév = tábla2.oszlopnév)]|
[CROSS JOIN tábla2];
```

A vizsgált adatbázis-kezelő rendszerek támogatják a belső összekapcsolást (`INNER JOIN`), de a többi összekapcsolás típus megvalósításában jelentős eltérések vannak.

Join típusa	PostgreSQL	DB2	MSSQL	MySQL	Oracle
NATURAL JOIN	✓	–	–	✓	✓
USING záradék	✓	–	–	✓	✓
LEFT, RIGHT, FULL JOIN	✓	✓	✓	✓	✓
CROSS JOIN	✓	–	✓	✓	✓

Egyes adatbázis-kezelő rendszerek – mint például az Oracle – lehetőséget biztosít a külső tábla összekapcsolások speciális szintaxissal történő jelölésére. Ezeknek a használata igen elterjedt, de használatuk – amennyiben az adatbázis függetlenség szempont – nem javasolt.

NULL értékek rendezése

A szabvány szerint a relációk rendezetlenek, mindazonáltal lehetőséget biztosít, hogy a lekérdezések eredményhalmaza rendezett legyen.

```
SELECT ... FROM ... WHERE ...
ORDER BY oszlopnév1,oszlopnév2,...
```

A szabvány nem rendelkezik arról, hogy a `NULL` értéket hogyan kell rendelkezni a nem `NULL` értékekkel szemben.

PostgreSQL	a `NULL` értéket minden más értéktől nagyobbnak tekinti
DB2	a `NULL` értéket minden más értéktől nagyobbnak tekinti
MSSQL	a `NULL` értéket minden más értéktől kisebbnek tekinti
MySQL	a `NULL` értéket minden más értéktől kisebbnek tekinti van egy nem dokumentált tulajdonsága: az oszlopnév elé mínusz jelet téve a rendezési sorrendet ellenkezőjére változtatja (használata nem javasolt!!!)
Oracle	a `NULL` értéket alapértelmezettként minden más értéktől nagyobbnak tekinti, de a `NULLS FIRST` és `NULLS LAST` használatával befolyásolható a `NULL` értékek helyzete

Egyszerre több sort beszúró INSERT utasítás

Nagy mennyiségű adatbeszúráskor jó szolgálatot tehet az adatbázisba egyszerre több rekordot beszúró `INSERT` utasítás. Az SQL szabvány opcionális lehetőségként említi a használatát.

```
INSERT INTO táblanév VALUES(értéka1,értéka2),
(értékb1,értékb2), (értékc1,értékc2)
```

Ez egyenértékű az

```
INSERT INTO táblaneve VALUES(értéka1,értéka2)
INSERT INTO táblaneve VALUES(értékb1,értékb2)
INSERT INTO táblaneve VALUES(értékc1,értékc2)
```

utasítás sorozattal.

PostgreSQL	támogatja
DB2	támogatja
MSSQL	nem támogatja
MySQL	támogatja
Oracle	nem támogatja

A BOOLEAN adattípus

A `BOOLEAN` adattípus a szabvány szerint szabadon választható, ami egy kissé talán meglepő egy alapvető adattípusnál. A szabvány szerint a `BOOLEAN` értéke a következők közül az egyik lehet:

- `TRUE`
- `FALSE`
- `UNKNOWN` vagy `NULL`

Az adatbázis-kezelő rendszer értelmezheti a `NULL` értéket `UNKNOWN`-ként.

PostgreSQL	támogatja a szabványt, a `NULL`-t elfogadja `BOOLEAN` típusúként, az `UNKNOWN`-t nem
DB2	nem támogatja a `BOOLEAN` típust, a `CHAR(1)` mezőt ajánlja logikai érték tárolásra (0,1 és `NULL`)
MSSQL	nem támogatja a `BOOLEAN` típust. Lehetséges alternatíva a `BIT` típus (0,1 vagy `NULL`)
MySQL	a `BOOLEAN` típusa igazából csak a `TINYINT(1)`
Oracle	nem támogatja a `BOOLEAN` típust, a `NUMBER(1)`-et javasolja helyette

A CHAR adattípus

A szabvány két adattípust határoz meg a karakteres adatok tárolására. A változó hosszúságúak számára a `VARCHAR`, a fix hosszúságúakhoz pedig a `CHAR` típust. A `CHAR(n)` n karakter befogadására képes mezőt definiál. Abban az esetben, ha a tárolni kívánt karaktersorozat hossza kisebb, mint `n`, akkor jobbról szóköz karakterekkel tölti ki. Ha a beszúrni kívánt karaktersorozat hossza nagyobb mint `n`, akkor hibával tér vissza kivéve, ha a túlnyúló karakterek mindeggyike szóköz karakter. A szabvány azt mondja, hogy típuskényszerítés és más karakteres értékekkel való összehasonlításkor szóközökkel kell kitölteni, ha szükséges.

PostgreSQL	alapvetően a szabványt követi, de néhány függvény használatakor csonkít (pl.: `CHARACTER_LENGTH`)
DB2	követi a szabványt
MSSQL	alapvetően a szabványt követi, de néhány függvény használatakor csonkít (pl.: `LEN`)
MySQL	a szabványtól eltérő módon működik, a mezőnél hosszabb sztring beszúrásakor csonkít és nem tér vissza hibával
Oracle	követi a szabványt apró eltéréssel (nem távolítja el a felesleges szóközöket a sztring hosszának megállapításakor)

Az időbélyeg

Ennek a típusnak az a feladata, hogy tárolja az évet, hónapot, napot, órát, percet, másodpercet(tört értékként). Létezik egy kiterjesztett időbélyeg típus, mely az időzónát is tárolja
Példa az időbélyegre:

```
TIMESTAMP: '2008-01-01 23:33:45'
TIMESTAMP: '2008-01-01 23:33:45.5'
```

Példa az időzónával kiegészített időbélyegre:

```
TIMESTAMP WITH TIME ZONE: '2008-01-01 23:33:45+02:00'
TIMESTAMP WITH TIME ZONE: '2008-01-01 23:33:45.5+02:00'
```

PostgreSQL	követi a szabványt egy kivétellel: a `'2008-01-01 23:33:45+02:00'`-t `TIMESTAMP WITHOUT TIME ZONE` típusként értelmezi
DB2	rendelkezik `TIMESTAMP` típussal, de nincs időzónával kiterjesztett változata
MSSQL	rendelkezett egy látszólagos `TIMESTAMP` típussal, de már nem használatos. Ami legközelebb áll a szabvány `TIMESTAMP` típushoz, az a `DATETIME`
MySQL	a `TIMESTAMP` típusa meglehetősen különbözik a szabványostól. Rendelkezik egy `DATETIME` típussal is, de az nem képes a másodperc tört részét tárolni
Oracle	követi a szabványt, rendelkezik `TIMESTAMP` és `TIMESTAMP WITH TIME ZONE` típussal is

Minden vizsgált adatbázis-kezelő képes felismerni a hibás dátumokat. Nem úgy, mint a MySQL. A többi adatbázis kezelőtől eltérően a `TIMESTAMP: '2003-02-29 23:33:45'` érték rögzítésekor nem ad kivételt, hanem a `'0000-00-00 00:00:00'` időbélyeget tárolja mindenféle figyelmeztetés nélkül.

A CHARACTER_LENGTH függvény

Karakteres típusok esetén sok esetben szükség lehet a karaktersorozat hosszára. Erre a szabvány a `CHARCTER_LENGTH(argumentum)` függvényt definiálja. A szabvány opcionálisan tartalmazza a `CHARCTER_LENGTH(argumentum USING szöveg-egység)` formát is. A sztring egység a következők valamelyike lehet:

- `UTF8`
- `UTF16`
- `UTF32`

Visszatérési értéke `NUMERIC` típusú illetve `NULL`, ha az argumentuma `NULL`. Az argumentum típusa `CHAR` vagy `VARCHAR` lehet. A `CHAR_LENGTH` ennek a függvénynek a szinonimája.

PostgreSQL	követi a szabványt, elérhető a `CHARCTER_LENGTH` és a `CHAR_LENGTH` függvény is
DB2	van `CHARCTER_LENGTH` függvénye, de csak a szövegegységgel kiegészített változata. Nagyobb probléma, hogy a szövegegység különbözik a szabványban foglalttól (`CODEUNITS16`, `CODEUNITS32`, `OCTETS`) Rendelkezik viszont egy `LENGTH` függvénnyel, ha nem kívánunk szövegegységet megadni.
MSSQL	nem rendelkezik `CHARCTER_LENGTH` függvénnyel. Helyette a `LEN` és a `DATALENGTH` függvényt használható.
MySQL	rendelkezik `CHARCTER_LENGTH` függvénnyel, mely `CHAR_LENGTH` és `LENGTH` néven is elérhető.
Oracle	nem rendelkezik `CHARCTER_LENGTH` függvénnyel, helyette a `LENGTH` függvény használható

A SUBSTRING függvény

A szabvány a rész-sztring képzésre két megoldást definiál

- A szokásos `SUBSTRING` függvény, mely karaktereket nyer ki egy sztringből:
  `SUBSTRING(szöveg FROM kezdő-pozíció [FOR hossz])`

A kezdő pozíció az 1, ha a hossz nincs megadva, akkor végtelennek tekinti. Az eredmény `NULL`, ha bármelyik argumentuma `NULL`. Ha a rész-sztring kívül esik a szövegen, akkor üres sztringgel tér vissza.

- Opcionálisan ajánl egy reguláris kifejezéses változatot:
  `SUBSTRING(szöveg SIMILAR minta ESCAPE elválasztó karakter)`
  A minta kötelezően három részből kell, hogy álljon:
  - a kívánt rész-sztring előtti karaktersorozat
  - a kívánt rész-sztring
  - a kívánt rész-sztring utáni karaktersorozat

  Ezeket a részeket az elválasztó karakter és az idézőjel kell, hogy elválassza
  pl.: a `SUBSTRING('fgh' SIMILAR 'f@"g@"h' ESCAPE '@')` eredménye `'g'`. A minta leírására szolgáló szabályok nem teljesen egyeznek meg a POSIX reguláris kifejezésekkel

PostgreSQL	Három függvényt is ad a rész-sztring képzéshez: - a szabvány `SUBSTRING` egy változatát - POSIX reguláris kifejezést használó `SUBSTRING` - a reguláris kifejezést használó szabványos `SUBSTRING`
DB2	A megvalósított `SUBSTRING` függvényének – a `CHARCTER_LENGTH` függvényhez hasonlóan – csak szövegegységgel kiegészített változata létezik. A 9-es verzió előtt használható volt a nem szabványos `SUBSTR` függvény is. Reguláris kifejezést használó változattal nem rendelkezik
MSSQL	Van `SUBSTRING` függvénye, de szintakszisa különbözik a szabványostól: `SUBSTRING(szöveg,kezdő-pozíció,hossz)` Reguláris kifejezést használó változattal nem rendelkezik
MySQL	Rendelkezik a hagyományos `SUBSTRING` függvénnyel. Reguláris kifejezést használó változattal nem rendelkezik
Oracle	Nem rendelkezik szabványos `SUBSTRING` függvénnyel. Helyette a `SUBSTR` használható: `SUBSTR(szöveg,kezdő-pozíció,[hossz])` Reguláris kifejezést használó változata a `REGEXP_SUBSTR`.

A LOCALTIMESTAMP függvény

Gyakran szükség lehet az aktuális dátum és idő ismeretére, például akkor, amikor naplózást kell megvalósítani. Az aktuális időbélyeg (időzóna nélkül) kérdezhető le a `LOCALTIMESTAMP` függvénnyel.

PostgreSQL	Követi a szabványt.
DB2	Nincs `LOCALTIMESTAMP` függvénye. Helyette a `CURRENT_TIMESTAMP` függvény használható.
MSSQL	Nincs `LOCALTIMESTAMP` függvénye. Helyette a `CURRENT_TIMESTAMP` függvény használható.
MySQL	Követi a szabványt.
Oracle	Követi a szabványt.

Sztringek összefűzése

A szabvány két sztring összefűzésére a `||` operátort definiálja:

```
szöveg1 || szöveg2
```

A szabvány szerint, ha valamelyik operandusa `NULL`, akkor az eredmény is `NULL`.

PostgreSQL	Követi a szabványt. Automatikusan konvertálja az operandusokat, ha szükséges.
DB2	Részben követi a szabványt, de nem képes automatikus konverzióra.
MSSQL	Nem követi a szabványt, mert a `||` helyett a +-t használja. Nem végez automatikus konverziót.
MySQL	Nem követi a szabványt, mert az összefűzésre az `OR` operátort használja. Helyette használható a `CONCAT(sztring1,sztring2)` függvény.
Oracle	Részben követi a szabványt. Az automatikus konverziót is ismeri. Ellenben a `NULL` értéket üres sztringként értelmezi, tehát ha valamelyik operandus `NULL`, az eredmény nem lesz `NULL` érték.

5. fejezet

Implementáció-független adatbáziskezelés

5.1. Az implementáció-függetlenség, mint igény

Az adatbázis-független alkalmazásokra való igény jogosan merülhet fel az informatika bármely területén. Gondoljunk csak arra, hogy mekkora plusz – és felesleges – kiadást jelenthet egy vállalkozás számára, ha jól működő információs rendszereit csak azért kell lecserélni, mert az adatbázis-kezelő rendszer, melyhez kifejlesztették már nem képes kiszolgálni a megnövekedett igényeket. Az ehhez hasonló problémák legegyszerűbb megoldása nyilvánvalóan az, hogy egyszerűen lecseréljük az adatbázis-kezelő rendszert. Sajnos az SQL nyelv implementációi közötti különbségek miatt ez önmagában nem elegendő. Az érintett alkalmazások minden SQL utasítását át kell alakítani a megfelelő nyelvjárásúra. További probléma lehet, ha az egyes adatbázis-kezelő rendszerek eléréséhez más és más módszereket kell használni, hogy az adatelérést biztosító komponenseket is le kell cserélni. Abban az esetben, ha a használt adatbázis-kezelő rendszer eléréséhez speciális függvénykönyvtárakat – esetleg más függvényeket – kell használni ez a probléma már igen komoly fejtörésre adhat okot.

A fent említett problémák áthidalására az alkalmazásokban én a következő két szinten való függetlenség megvalósítását tartom szükségesnek:

- egységes nyelv, melyet minden adatbázis-kezelő egyformán megért
- egységes interfész a különböző adatbázisok eléréshez

A függetlenség alatt az alkalmazástól és az adatbázis-kezelő rendszertől való függetlenséget értem. Ugyanakkor a függetlenség jelentheti a szabványtól való bizonyos szintű elszakadást, vagy éppen a szabványtól független nyelvi megvalósítást is.

5.2. Egységesített adatbázis elérés

A gyakorlatban természetesen mások is megfogalmazták az implementáció-függetlenség iránti igényüket. Ezen igények kielégítésére számos megvalósítás született. A következő megoldások elsősorban az egységes adatbázis interfész megvalósításokkal jelentkeztek, de vannak közöttük olyan megoldások is, melyek minimális nyelvi szintű konverziós lehetőséget is biztosítanak. Minden megvalósítás az alkalmazás és az adatbázis szerver között egy olyan réteget hoz létre, mely az alkalmazások számára egységes elérést biztosít és elrejti előlük a különböző adatbázis-rendszerekhez való hozzáférés sajátosságait.

5.2.1. PEAR::MDB2

A PEAR egy bővítmény- és alkalmazás gyűjtemény, mely teljes egészében php nyelven íródott. Jelenleg 232 csomagot tartalmaz mintegy 32 kategóriában. Itt az MDB2 csomagját fogjuk áttekinteni.

- objektumorientált API
- adat absztrakció és konverzió
- egységes hibakódok
- bufferelt és nem bufferelt lekérdezések
- Szekvencia/autoincrement emuláció
- `REPLACE` emuláció
- korlátozott alkérdés emuláció
- tranzakció és mentési pont támogatás
- `LOB` támogatás
- `INDEX`, `UNIQUE KEY`, `PRIMARY KEY` támogatás
- olvassa az `INFORMATION_SCHEMA`-t
- `CREATE`, `DROP`, `ALTER` támogatás
- teljes PEAR integráció

Az MDB2 jelenleg a következő adatbázisszervereket támogatja:

- FrontBase
- Firebird/InterBase
- Microsoft SQL Server

- MySQL
- Oracle 7/8/9/10
- PostgreSQL
- QuerySim
- SQLite 2

Kapcsolódás. Az adatbázishoz való csatlakozáskor a kapcsolat sztringet a következő formában kell megadni: `meghajtó://felhasználó:jelszó@hoszt/adatbázis`.

```
<?php
    require_once("MDB2.php")
    $con = MDB2::factory("pgsql://felhasznalo:" .
    "jelszo@localhost/Rabbit");
?>
```

Hibakezelés. A PEAR:MDB2 egységes megoldást biztosít a hibák felismerésére. Annak megállapítása, hogy hiba történt egyszerűen a `PEAR::isError(visszatérési_érték)` visszatérési értékének vizsgálatával dönthető el.

```
<?php
    require_once("MDB2.php");
    $url = "pgsql://felhasznalo:jelszo@localhost/Rabbit"
    $con = MDB2::factory($url);

    if(PEAR::isError($con)) {
        die("Hiba a csatlakozáskor: " . $con->getMessage());
    }
?>
```

Lekérdezés eredményhalmazzal. Az eredményhalmazt visszaadó lekérdezések készítésére a kapcsolatobjektumra meghívott `query` metódus alkalmas.

```
<?php
```

```
    require_once("MDB2.php");
    $url = "pgsql://felhasznalo:jelszo@localhost/Rabbit"
    $con = MDB2::factory($url);

    $sql = "SELECT * FROM users";
    $resultset = $con->query($sql);
    if(PEAR::isError($resultset)) {
        die('A lekérdezés nem végrehajtható: ' .
        $resultset->getMessage());
    }
    while($row = $resultset->fetchRow(MDB2_FETCHMODE_ASSOC))
    foreach($row as $field => $value) {
        echo "$field / $value \n";
    }
?>
```

Adatmódosító utasítások. A PEAR:MDB2 az `INSERT`, `UPDATE` és a `DELETE` utasítások végrehajtását is az előző pontban leírt `query` metódus hívásával teszti lehetővé.

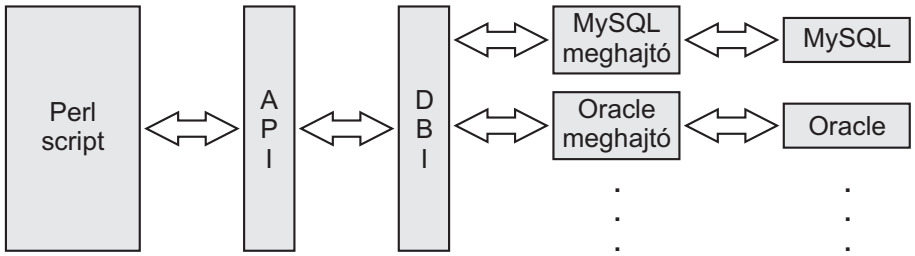
```
<?php
    require_once("MDB2.php");
    $url = "pgsql://felhasznalo:jelszo@localhost/Rabbit"
    $con = MDB2::factory($url);

    $sql = "UPDATE users SET irszam=4283 WHERE id = 1";
    $result = $con->query($sql);
    if(PEAR::isError($result)) {
        die('Az UPDATE nem végrehajtható: ' .
        $resultset->getMessage());
    }
?>
```

5.2.2. Perl:DBI

A Perl-t eredetileg az adminisztrációs feladatainak megkönnyítésére írta Larry Wall, mert nem volt kedve a meglévő eszközök korlátaival foglalkozni. Stílusában és funkcionalitásában sokat merít a C nyelvből, valamint a sed, awk programokból. A nyelv egyik legerősebb része a reguláris kifejezések. Egyik legelterjettebb alkalmazása a CGI szkriptek készítése.

A DBI egy perl modul formájában megjelenő programozói felület, amely mögött számos elterjedt adatbázis-kezelő rendszerhez találunk meghajtó programot. Minden támogatott rendszert egy-egy DBD modul képvisel, ezek jelentik a DBI mögötti meghajtókat. A rövidítések a logikai kapcsolatrendszerre utalnak:

- DBD - DataBase Dependent (adatbázis függő)
- DBI - DataBase Independent (adatbázis független)



5.1. ábra. A Perl DBI architektúra

Jelenleg a következő adatbázisrendszerekhez van megbízható támogatás:

- Oracle
- Informix
- mSQL
- MySQL
- Ingres
- Sybase
- DB2
- Empress
- SearchServer
- PostgreSQL
- XBase

Kapcsolódás. Az adatbázishoz való csatlakozáskor a meghajtó megadása a következő formában történik: `dbi:meghajtó:adatbázis`.

```
#!/usr/bin/perl

use DBI;
my $dbh = DBI->connect( "dbi:Oracle:Rabbit",
"felhasznalonev", "jelszo");

$dbh->disconnect;
```

Hibakezelés. Hibakezelésnél a Perl programokban hagyományosnak számító vagy(OR) rövidzár operátor használható.

```
#!/usr/bin/perl

use DBI;
my $dbh = DBI->connect( "dbi:Oracle:Rabbit",
"felhasznalonev", "jelszo" )
or die "Nem lehet csatlakozni: $DBI::errstr\n";

$dbh->disconnect or warn "$DBI::errstr\n";
```

Lekérdezés eredményhalmazzal.

```
#!/usr/bin/perl

use DBI;
my $dbh = DBI->connect( "dbi:Oracle:Rabbit",
"felhasznalonev", "jelszo" )
or die "Nem lehet csatlakozni: $DBI::errstr\n";

my $sth = $dbh->prepare( "SELECT * FROM users" )
    or die "$DBI::errstr\n";
```

```
$sth->execute
    or die "$DBI::errstr\n";

my @row;
while ( @row = $sth->fetchrow_array() ) {
      print "Row: @row\n";
}

$dbh->disconnect;
```

Adatmódosító utasítások.

```
#!/usr/bin/perl

use DBI;
my $dbh = DBI->connect( "dbi:Oracle:Rabbit",
"felhasznalonev", "jelszo" )
or die "Nem lehet csatlakozni: $DBI::errstr\n";

$dbh->do( "UPDATE users SET irszam=4283 WHERE id=1" )

$dbh->disconnect;
```

5.2.3. ODBC

Az előző megoldások mindegyike valamely programozási nyelvhez(Php, Perl) volt köthető. Az ODBC programozási nyelvtől független megoldás. Az ODBC (Open DataBase Connectivity) a Microsoft korai megoldása az adatbázis független adatelérésnek. Mivel a Microsoft felismerte, hogy a különböző adatbázis-kezelők eltérő módon programozhatóak illetve az SQL nyelvet eltérően valósítják meg – és ez jelentős terhet ró a fejlesztőkre – ezért a következő elvárásokat fogalmazta meg :

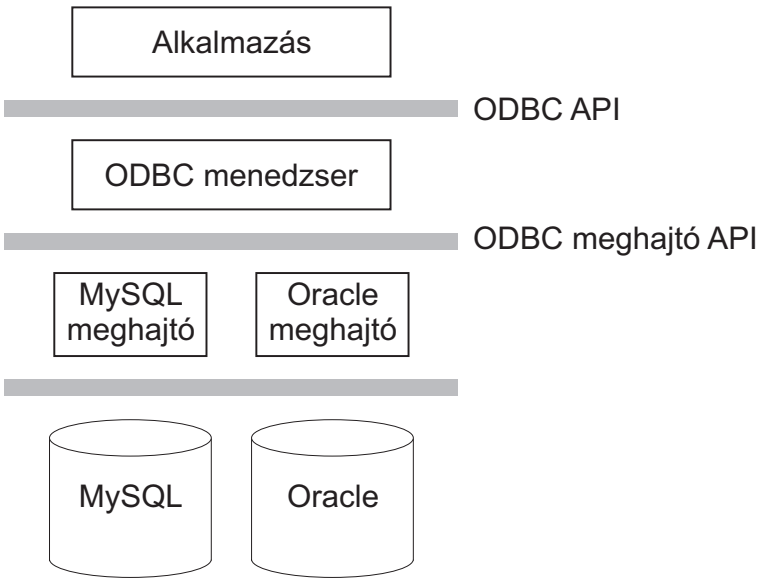
- hálózat függetlenség

- konverzió
- több forrás párhuzamos használata
- egységesítés
- egyszerű kezelés

Az ODBC egy C nyelven alapuló interfészt ad, mely egységes felületet biztosít az adatbázisokhoz a következő területeken:

- hibakódok
- csatlakozás és bejelentkezés
- adattípus és adatábrázolás

Az ODBC felépítését tekinthetjük át az 5.2. ábrán.



5.2. ábra. Az ODBC architektúrája

ODBC menedzser. Fogadja és értelmezi az alkalmazástól érkező ODBC-CLI hívásokat, szintaktikai ellenőrzéseket hajt végre. Szükség esetén betölti a megfelelő meghajtót a memóriába, nyilvántartja a kapcsolatokat, valamint továbbítja a megkapott parancsot a megfelelő meghajtónak. Az ODBC menedzser biztosítja, hogy a felhasználónak ne kelljen ismernie az egyes meghajtók betöltésének feladatát, egyszerűsíti az egyes meghajtók elérését és konverziós lehetőséget biztosít.

ODBC meghajtó. Az egységes, szabvány CLI hívásokat konvertálja adatforrás specifikus utasításokká, felveszi a kapcsolatot a megadott adatforrással. Feladata az adatforrással való kapcsolattartás, a hibakezelés. A számunkra különösen érdekes feladata az SQL konverzió és információ lekérdezés. A konverzió alatt a következőkhöz hasonló átalakításokat kell érteni:

- operátor csere (pl.: <> helyett !=)
- explicit nem jelölt mező egyes adatbázis kezelőknél NOT NULL, másoknál NULL

Mivel az SQL szabvány nem rendelkezett az adatbázis információk lekérdezésről, ezért egységesíteni kellett az annak formátumát. Az ODBC függvényeket definiál a metaadatok lekérdezésére és lehetőséget biztosít az egyedi funkcionális lehetőségek lekérdezésére is. Az ODBC az adatok elérését un. kurzorok segítségével biztosítja. Három féle kurzortípust támogat:

- Statikus kurzor: A lekérdezés eredményeként visszaadott értékek a kurzor megnyitásakor meghatározódnak.
- Kulcs-vezérelt: A kurzor megnyitásakor meghatározódik, hogy mely rekordok és milyen sorrendben kerülnek be az eredménybe.
- Dinamikus: Mindig az adatbázis aktuális állapotát tükrözi, tehát a rekordok darabszáma, értékei megváltozhatnak a kurzor kezelése közben

Ahhoz, hogy alkalmazásaink az adatbázis elérését ODBC-n keresztül végezhessék el szükség van arra, hogy az adatforrás regisztrálva legyen a Windows ODBC adatforrás felügyelőben. Itt kell kiválasztani a meghajtó típusát is, mint ahogy az az 5.3. ábrán látható. Meghajtótól függően más és más felület az, ahol az adatforrás adatai beállíthatóak. Természetesen a kötelező paramétereken kívül sok egyéb, az adott adatbázis-kezelő rendszerre jellemző beállítás is elvégezhető. Ilyen lehet például a típuskonverziós szabályok megadása. A MySQL meghajtó kötelező paramétereinek megadására láthatunk egy példát az 5.4. ábrán.

Az alkalmazásokban az ODBC-n keresztüli adatbázis elérésre láthatunk egy kódrészletet a következő példában. Az adatbázis adatainak eléréséhez dinamikus kurzort használ, amint az a kapcsolatot megnyitó `OpenConnection` metódus második paraméterében látható. A példába ugyan nem került bele, de a kurzort az `rs.Next` metódus segítségével lehet léptetni és az `rs.EOF` igaz értéke jelzi, ha nincs több betölthető rekord.

```
# Adatbázis kapcsolat ODBC használatával
```

```
# Visual Basic 6 példakód

Dim connString As String
Dim rs as RecordSet
Dim sqlText as String
Dim ws as Workspace
dim conn as Connection

connString = "ODBC;DSN=Rabbit;DATABASE=Rabbit;SERVER=localhost;
UID=felhasznalo; PWD=jelszo"

Set ws = DBEngine.Workspaces(0)
Set conn = ws.OpenConnection("Rabbit", dbDriverNoPrompt, ,
connString)

sqlText = "SELECT fullname FROM users WHERE id=1"
Set rs = conn.OpenRecordset(sqlText, dbOpenDynamic)

msgbox "A felhasználó neve: " & rs.Fields(0)

conn.Close
ws.Close
```

Adattípus konverzió. Az ODBC egyik nagy előnye, hogy egységes adatkonverziót ír elő, mely bizonyos szempontból adatbázis függetlenné teszi az alkalmazásainkat. Definiálja az úgynevezett Szimbolikus SQL adattípusokat és az ezeket reprezentáló C nyelvi típusokat. Az ODBC meghajtó írójának a feladata, hogy a saját adattípusai és a szimbolikus SQL adattípusok közötti összerendelést elvégezze. A következő táblázat bemutatja, hogy mely szimbolikus SQL típus hogyan kerül leképezésre.

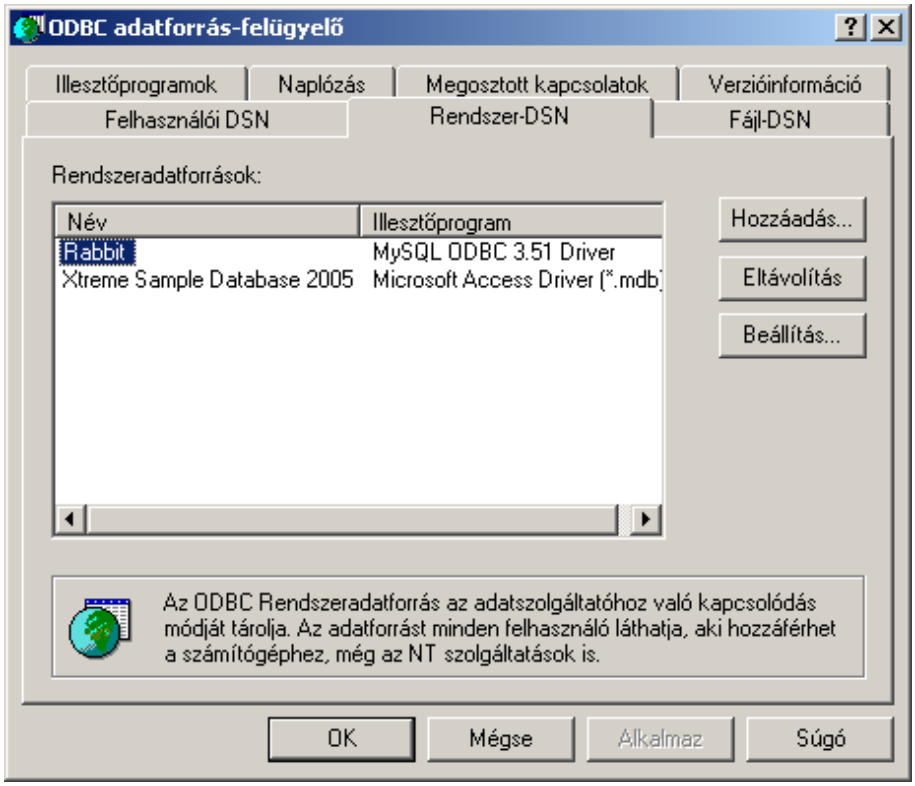
Szimbolikus SQL típus	C típus
SQL_C_BIGINT	long int
SQL_C_CHAR	unigned char
SQL_C_BIT	unsigned char vagy char
SQL_C_TINYINT	signed char
SQL_C_SHORT	short int
SQL_C_LONG	long int
SQL_C_DOUBLE	double
SQL_C_FLOAT	float
SQL_C_CLOB_LOCATOR	long int
SQL_C_BINARY	unsigned char
SQL_C_BLOB_LOCATOR	long int
SQL_C_DBCHAR	unsigned short int
SQL_C_DBCLOB_LOCATOR	long int
SQL_C_WCHAR	wchar_t

Ez a táblázat nem tartalmazza az olyan szimbolikus SQL típusokat, melyek csak összetett adattípusokra képezhetőek le. Például a `SQL_C_TYPE_DATE` típus megvalósítása a következő:

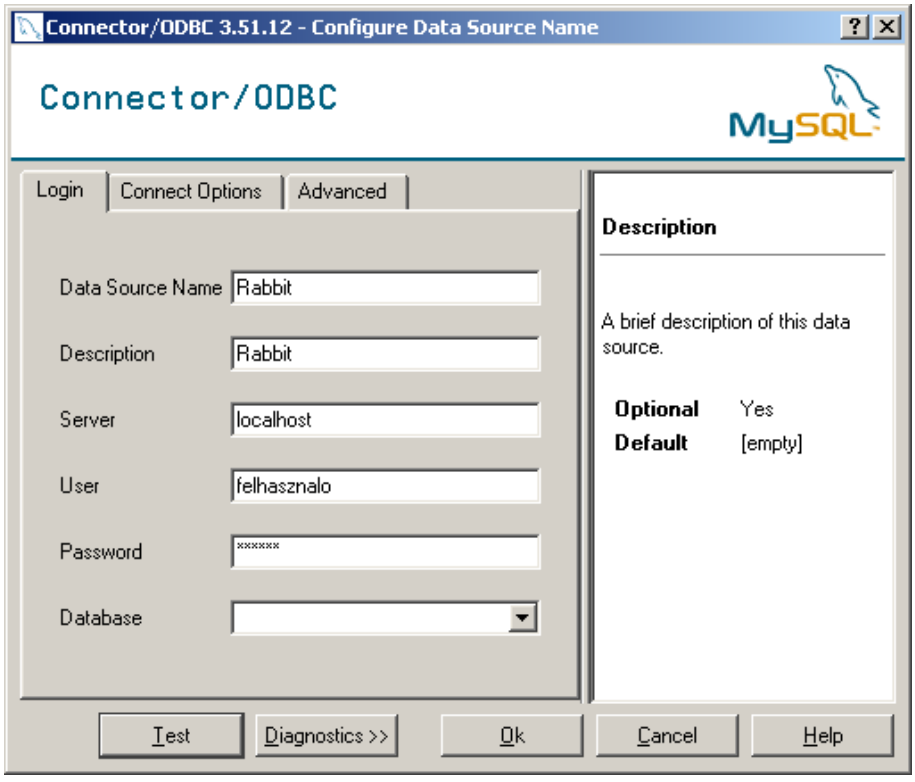
```
typedef struct DATE_STRUCT{
    short int    year;
    short int    month;
    short int    day;
} DATE_STRUCT;
```

Az, hogy a C típusok nem C nyelven írt alkalmazásokban hogyan jelennek meg, az az alkalmazott nyelv sajátosságaitól függ.

Az ODBC ugyan a Microsoft terméke, ez mégsem jelenti azt, hogy kizárólag Windows platformon lenne elérhető. ODBC-vel a Linux/Unix rendszerekben is találkozhatunk. Ugyanakkor az ODBC-nél ma már vannak fejlettebb adatelérési módszerek is, de mégis van jelentősége, hiszen gyakorlatilag az összes adatbázis-kezelő rendszerhez van meghajtója, amit az újabb megoldások nem mindig mondhatnak el magukról.



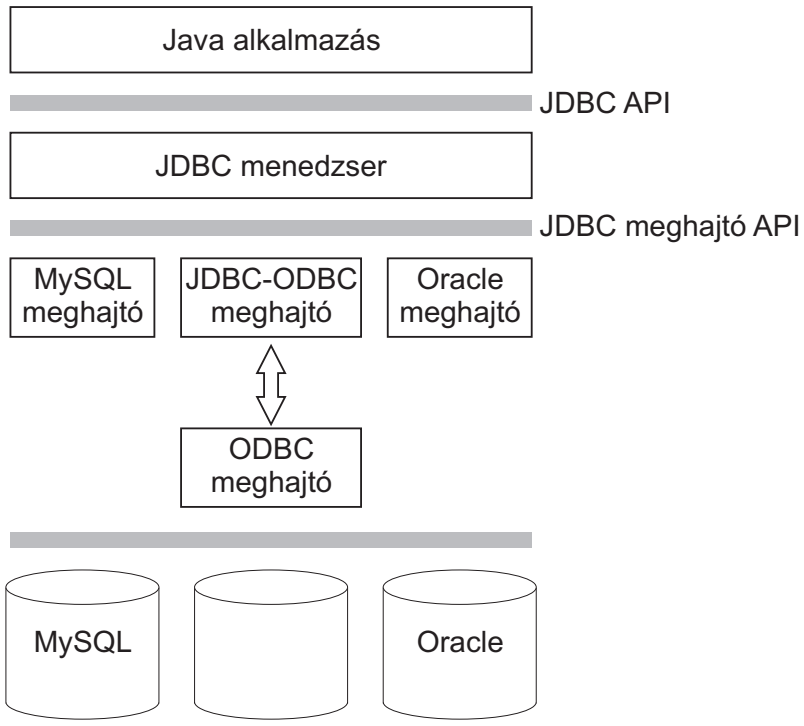
5.3. ábra. ODBC adatforrás-felügyelő



5.4. ábra. MySQL ODBC meghajtó beállítása

5.2.4. JDBC

A JDBC(Java DataBase Connectivity)-t az adatbázisok elérésére szánt alacsonyszintű könyvtárnak szánták. A könyvtár tervezésénél nem kellett teljesen az alapoktól építkezniük, mert az X/OPEN kunzorcium SQL CLI(Call Level Interface) specifikációjából indultak ki, amely nagyon hasonlít az ipari szabványnak tekintett Microsoft féle ODBC specifikációhoz. Ettől igazából csak azért kellett eltérniük, mert a Java nyelv a C-vel szemben nem tartalmaz mutatókat.



5.5. ábra. A JDBC architektúrája

JDBC menedzser. A JDBC menedzser két felületet definiál. A felső felület egy olyan programozói interfész, melynek segítségével a programozó elérheti az adatbázis szolgáltatásait. Az alsó felület pedig a különböző adatbázis-kezelő rendszerek elérését lehetővé tevő meghajtóprogramok elérésére szolgál.

JDBC meghajtó.

- JDBC-ODBC bridge + ODBC driver: JDBC - ODBC konverziót hajt végre, használata akkor indokolt, ha nem áll rendelkezésre megfelelő JDBC meghajtó

- Nativ-API partly Java driver: a meghajtó csak részben íródott Java nyelven, ezért platform specifikus
- JDBC-Net pure Java driver: egyszerű Java kliens könyvtár, mely adatbázis független hálózati protokolnon keresztül kommunikál egy szerver komponenssel, mely továbbítja azt az adatbázisnak
- Native-protocol pure Java driver: egyszerű Java könyvtár, mely közvetlenül az adatbázissal kommunikál

Egy Java alkalmazás JDBC-n keresztüli adatbázis elérésére láthatunk egy kódrészletet a következő példában. Ebben a példában már a kurzor használata is látható. Ismerve azt, hogy a JDBC megvalósítása mennyire hasonlít az ODBC-hez nem meglepő – a Visual Basic és a Java nyelv sajátosságaitól eltekintve – a szembetűnő hasonlóság.

Példa.

```
# Adatbázis kapcsolat JDBC használatával

import java.sql.*;

class JDBCTest
{
    public static void main(String args[])
    {
        try
        {
            Class.forName("com.mysql.jdbc.Driver");
            Connection con = DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/Rabbit",
            "felhasznalonev","jelszo");
            Statement st = con.createStatement();
            ResultSet rs = st.executeQuery(
            "SELECT fullname FROM users");
            while(rs.next())
            {
```

```
                System.out.println(rs.getString(0));
            }
            rs.close;
            st.close;
            con.close;
        }catch(Exception e){}
    }
}
```

A JDBC hasonlóan az ODBC-hez adattípus konverziót is végrehajt. A JDBC által definiált konverziók láthatók a következő táblázatban:

SQL típus	Java típus
CHAR	String
VARCHAR	String
LONGVARCHAR	java.io.InputStream
NUMERIC	java.sql.Numeric
DECIMAL	java.sql.Numeric
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT	float
DOUBLE	double
BINARY	byte[]
VARBINARY	byte[]
LONGVARBINARY	java.io.InputStream
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp

5.3. Egységes lekérdezőnyelv

A következő megoldások a nyelvi szintű egységesítés jegyében született. Azonban alapvető különbség van közöttük annak tekintetében, hogy míg a Hibernate nyílt forrású project keretében született és bárki számára elérhető, addig az SAP OpenSQL nyelve kifejezetten a méltán nagy nevű vállalatirányítási rendszerének sajátja és más rendszerek számára nem hozzáférhető.

5.3.1. Hibernate

A Hibernate egy nagy teljesítményű objektumrelációs perzisztencia és adatbázislekérdező (query) szolgáltatás, mely Java és .Net alatt érhető el jelenleg. A Hibernate lehetővé teszi az objektumorientált elvet követő perzisztens osztályok létrehozását beleértve az asszociációt, öröklődést, polimorfizmust, kompozíciót, kollekciót. A Hibernate segítségével saját, hordozható SQL kiterjesztésében (HQL) is történhet a lekérdezés. Lehetőséget biztosít natív SQL utasítások kiadására is, valamint a Criteria rendszerének használatával a lekérdezések objektumorientált módon is megfogalmazhatóak.

A következő részben röviden áttekintem a Hibernate HQL nyelve által nyújtott lehetőségeket. Ahhoz, hogy a Hibernate-t használni tudjuk természetesen nem elegendő a HQL ismerete, szükség van még az osztályokat leíró mapping állományokra, valamint a beállításokat tartalmazó konfigurációs állományra. A konfigurációs állományban adható meg – többek között – , hogy milyen JDBC meghajtót használjon az adatbázissal való kapcsolattartáshoz. Ebből azt hiszem látható, hogy a Hibernate használatával (és az általa használt JDBC kapcsolattal) az adatbázis függetlenség megvalósításához szükséges mindkét követelmény teljesül.

A Hibernate az adatbázissal természetesen SQL utasítások segítségével tartja a kapcsolatot. A HQL utasításokat tehát le kell fordítania az éppen használt adatbázis-kezelő SQL dialektusára. Jelenleg a következő dialektusokat ismeri:

- DB2
- PostgreSQL
- MySQL
- Oracle
- Sybase
- Microsoft SQL Server

- SAP DB
- Informix
- HypersonicSQL
- Ingres
- Progress
- Mckoi SQL
- Interbase
- Pointbase
- Frontbase
- Firebird

A HQL használatának előnyei.

- a Hibernate lekérdező nyelve osztályokat és tulajdonságokat használ táblák és oszlopok helyett
- a lekérdezések eredménye objektumok formájában jelenik meg az alkalmazásokban, melyeknek használata egyszerűbb és nyilvánvalóbb egy objektumorientált nyelvben
- támogatja a polimorfikus lekérdezéseket, vagyis a lekérdezésekben megadott osztályok és az azt kiterjesztő osztályok minden példányát visszaadja
- könnyű megtanulni és implementálni az alkalmazásokban
- támogatja a következő SQL jellemzőket
  - tábla összekapcsolás(`INNER` / `OUTER` / `FULL JOIN`, Descartes szorzat)
  - aggregátum függvények(`MAX`, `AVG`)
  - csoport képzés(`GROUP BY`)
  - rendezés(`ORDER BY`)
  - alkérdések
- a HQL-ben írt lekérdezések adatbázis függetlenek

HQL. Egy HQL utasítás a következő elemekből állhat:

- záradékok
  - from
  - select

  - where
  - order by
  - group by
- aggregátum függvények
  - avg
  - sum
  - min
  - max
  - count
- alkérdések

HQL alkalmazási példa. A következő példában egy adatbázis USERS táblájának lekérdezését fogom bemutatni. Ehhez először létre kell hozni a User osztályt a szükséges metódusokkal.

```
package pelda;
pubic class User{
    private String fullname;
    private long id;

    public String getFullName(){return fullname;}
    public String setFullName(String _fullname)
    {fullname=_fullname;}
    public String getId(){return id;}
    public String setId(long _id){id=_id;}
}
```

Az osztályhoz tartozó map állomány pedig a következő:

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
```

```
<hibernate-mapping>
  <class name="pelda.User" table="USERS">
      <id name="id" type="long" column="ID" >
          <generator class="assigned"/>
      </id>
      <property name="FullName">
          <column name="FULLNAME" />
      </property>
  </class>
</hibernate-mapping>}
```

Egy User példány felvétele(leképezése) az adatbázisba:

```
package pelda;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernatePelda{

public static void main(String[] args) {
    Session session = null;
    try{
        SessionFactory sessionFactory = new Configuration().
        configure().buildSessionFactory();
        session =sessionFactory.openSession();
        User user = new User();
        user.setId(1);
        user.setFullName("Nagy László");
        session.save(user);
    }catch(Exception e){
```

```
      System.out.println(e.getMessage());
    }finally{
      session.flush();
      session.close();
    }
}
}
```

A tárolt User példány lekérdezése HQL segítségével:

```
package pelda;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HQLSelect{

public static void main(String[] args) {
Session session = null;
try{
    SessionFactory sessionFactory = new Configuration().
    configure().buildSessionFactory();
    session =sessionFactory.openSession();

    String sql = "select fullname from User as user" +
                 " where user.id<10";
    Query query = session.createQuery(sql);
    for(Iterator it=query.iterate();it.hasNext();)
    {
        Object[] row = (Object[]) it.next();
        System.out.println("Név: " + row[0]);
    }
```

```
    }catch(Exception e){
      System.out.println(e.getMessage());
    }finally{
      session.close();
    }
}
}
```

5.3.2. Open SQL

Az Open SQL az SAP ABAP programozási nyelvének egyik fő jellegzetessége. Segítségével a felhasználó elérheti az alkalmazások mögötti adatbázis szintet. Az Open SQL a szabványos SQL nyelv egy független változata és valójában annak utasításai közül csak az adatmanipuláló utasítások elérését teszi lehetővé. Ilyenek a `SELECT`, `INSERT`, `UPDATE` és a `DELETE`. Igazából nincs is szüksége olyan SQL utasításokra, mint például a `COMMIT` vagy a `ROLLBACK`. Ugyanis az SAP tranzakció nem feltétlenül felel meg az adatbázis tranzakció fogalmának. Valójában az SAP tranzakció az üzleti logikával, az üzleti folyamatokkal van összefüggésben. Az Open SQL utasítások végrehajtáskor természetesen az éppen használt adatbázis-kezelő rendszer nyelvjárására kerül lefordításra.

Az Open SQL utasításai hasonlóan jelennek meg az ABAP programokban, mint más programnyelveknél a beágyazott SQL utasítások, de itt a nyelv része és nem előfordító dolgozza fel. Egy egyszerű ABAP program, mely az Open SQL használatával kérdezi le a SPFLI tábla sorainak a számát, a következő:

```
REPORT YNL_DIP1.
DATA ROWS TYPE P.
SELECT COUNT(*) FROM SPFLI INTO ROWS.
WRITE: / 'Az SPFLI tábla ', ROWS , ' sort tartalmaz'.
```

Természetesen az Open SQL ettől sokkal többre képes, `SELECT` utasítása például a következő lehetőségeket biztosítja:

- tábla összekapcsolás `INNER JOIN` és `LEFT OUTER JOIN` segítségével

- alkérdések
- `WHERE` záradék
- `GROUP BY` záradék
- `MIN`, `MAX`, `AVG`, `SUM`, `COUNT` aggregátum függvények
- `HAVING` záradék
- Eredményhalmazok egyesítése a `UNION` segítségével
- eredményhalmaz rendezése az `ORDER BY` záradék segítségével

Az `INSERT`, `UPDATE` és a `DELETE` utasításainak szintaxisa megfelel a szabványos SQL utasításának:

```
INSERT INTO táblanév (mező1,...) VALUES(érték1,...)
UPDATE táblanév SET mező1=érték1,... WHERE ...
DELETE FROM táblanév WHERE ...
```

Természetesen az ABAP rendelkezik a `SELECT` kurzoros megoldásával is, de talán érdekesebb megoldás – a program áttekinthetőségét javító – `SELECT LOOP` ciklusa:

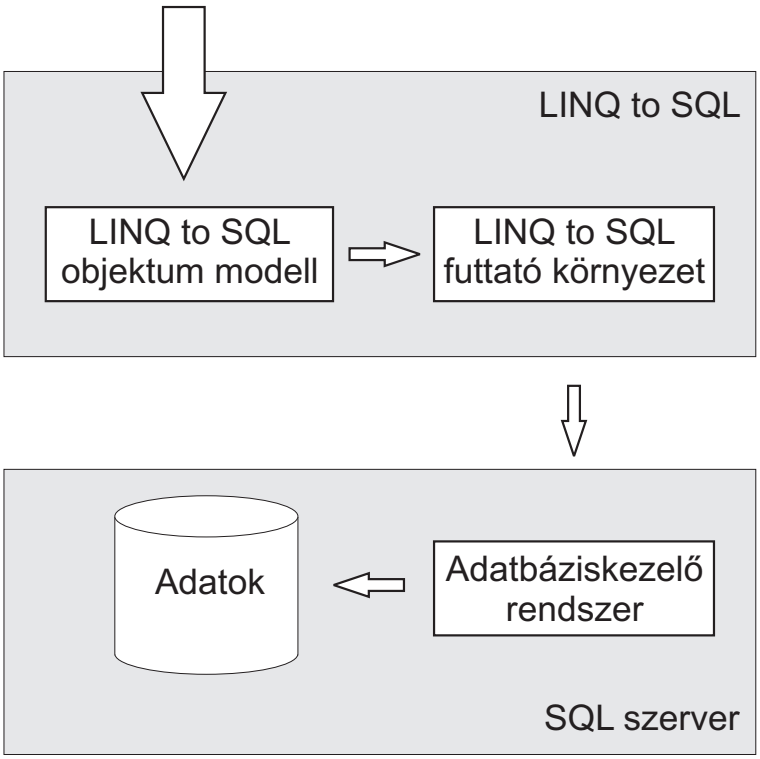
```
SELECT CITYFROM CITYTO FROM SPFLI INTO (CFORM,CTO).
    WRITE: \ CFROM, ' -> ',CTO.
ENDSELECT.
```

5.3.3. LINQ to SQL

A Microsoft által készített .NET keretrendszer egy gyors alkalmazásfejlesztést, platformfüggetlenséget és hálózati átlátszóságot támogató szoftverfejlesztői platform. A .NET a Microsoft stratégiai kezdeményezése a kiszolgáló-oldali és asztali fejlesztésekhez a következő évtizedre. Számunkra a .Net adatbázis kezelés támogatása az, ami érdekes. A .Net-nek része az ADO.Net, mely az ActiveX Data Objects(ADO) továbbfejleszett verziója, mely ODBC-n és OLEDB-n keresztül biztosítja az adatbázis kiszolgálók elérését. Annak oka, hogy a .Net-et a nyelvfüggetlen megvalósítások között említem meg az az, hogy a korábbi verzióihoz képest a 3.5-ös verziója olyan új, az adatbázis függetlenség szempontjából érdekes eszközöket bocsátott a programozók rendelkezésére, amely mellett nem mehetek el szó nélkül.

A LINQ(Language Integrated Query) keretrendszer több részre bomlik attól függően, hogy milyen objektumokkal dolgozik. A LINQ to XML az XML-állományok, míg a LINQ

to SQL a relációs adatbázisok lekérdezésére ad hatékony nyelvi eszközöket. A LINQ to SQL – korábbi nevén DLINQ – feladat, hogy a relációs és objektumorientált világot közel hozzák egymáshoz. Egészen annyira, hogy a programok írása közben a programozónak nem kell kilépnie a megszokott környezetből és az adatbázis lekérdezéseket a használt programozási nyelven tudja megfogalmazni. Ezt a LINQ to SQL objektum modeljét felhasználva tehetjük meg. A LINQ to SQL és az adatbázis-kezelő rendszer kapcsolatát láthatjuk az 5.6. ábrán.



5.6. ábra. A LINQ to SQL szerepe

A LINQ to SQL használatának nagy előnye, hogy a korábbi .NET-es adatelérési modell általános objektumai(DataTable, DataSet, DataRow) helyett erősen típusos objektumokat használ, melyeknek közvetlen jelentése van az alkalmazás üzleti logikájában. Ilyen objektum lehet például egy számlázó rendszerben a SZÁMLA vagy a VEVŐ, mely a SZÁMLA és VEVŐ tábla leképezése azzal a fontos többlettel, hogy a kapcsolatokat is visszakapjuk a kulcs - idegen kulcs kapcsolatok ismerete nélkül. Jelenleg a C# 3.0 és a Visual Basic 9.0 alatt érhetőek el a LINQ to SQL szolgáltatásai.

A következő példa bemutatja a LINQ to SQL használatát. Az első osztály a VEVO adatbázis tábla úgynevezett entitás osztálya.

vevo.cs

```
using System.Data.Linq;
using System.Linq;

namespace pelda
{
    [Table(Name= "Vevo")]
    class Vevo
    {
        [Column(Id=true)]
        public int VevoAzon;
        [Column]
        public int VevoNev;
    }
}
```

A lekérdezést végző alkalmazás pedig a következő.

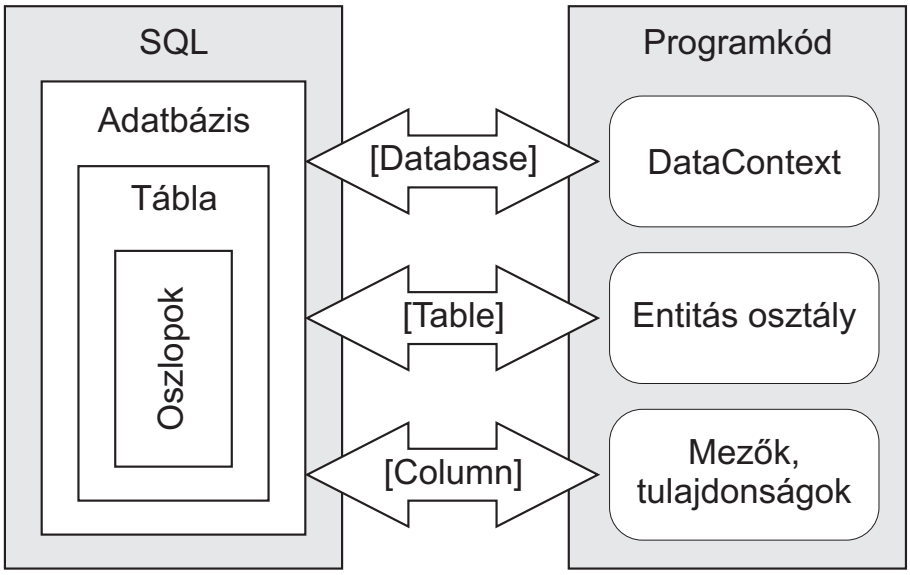
main.cs

```
using System.Data.Linq;
using System.Linq;

namespace pelda
{
    public static void main(String[] args)
    {
        String connStr = @"Data Source=.\SQLExpress;
                           Initial Catalog=Szamla";
        DataContext dc = new DataContext(connStr);

        var q = from e in dc.GetTable<Vevo>();
```

```
                where e.VevoAzon < 10
                select e;

        foreach(Vevo e in q)
        {
            console.WriteLine(e.VevoNev);
        }
    }
}
```

Amint a példa programban látható az objektumokra való leképezésnek feltétele, hogy entitás osztályokat hozzunk létre. A sok táblát használó alkalmazások íróinak az összes tábla leképezése nyílván nagy feladat. Ez a tervezők is felismerték ezért eszközöket adtak a Visual Studio alkalmazásfejlesztő környezethez, melyek automatikusan végzik a leképezést.

Az adatbázis és a nyelv objektumai közötti megfeleltetést szemlélteti az 5.7. ábra.



5.7. ábra. A LINQ to SQL és az objektumok

A LINQ to SQL természetesen a bemutatott példától sokkal többre is képes. Sajnos nincs lehetőségem a szolgáltatásainak részletes bemutatására, de annak érzékeltetésére, hogy milyen lehetőségek rejlenek benne álljon itt egy felsorolás:

- késleltetett vagy azonnali végrehajtás

- ad hoc kapcsolatok
- öröklés
- módosítás és változáskezelés
- konkurencia-kezelés
- tranzakció kezelés
- öröklés

Bár a LINQ to SQL-ben nagy lehetőségek vannak, azt a készítői sem titkolják, hogy nem csodafegyver, vagyis nem oldható meg benne minden, az adatbázis kezelés területén felmerült probléma. Az alkalmazások egy jó részének azonban elegendő az a szolgáltatás, amit nyújtani tud.

6. fejezet

Összefoglalás

Az egyes adatbázis-kezelő rendszerek SQL megvalósításainak vizsgálatával csak a felszínt kapargattuk meg azzal, hogy néhány érdekesebb jellemzőt megvizsgáltunk, mégis néhány érdekes megállapítást tehetünk. Számomra elsősorban a PostgeSQL és az Oracle szabványokhoz való viszonya volt meglepő tudva azt, hogy a PostgreSQL-t igen gyakran az Oracle-lel hasonlítják össze. Úgy tűnik a PostgreSQL jobban igyekszik a szabványoknak megfelelni. Ha egy kicsit gúnyosan akarnék erről nyilatkozni, akkor azt mondanám – ismerve az Oracle piacvezető szerepét –, hogy amíg a PostgreSQL alkalmazkodik a szabványokhoz, az Oracle írja azokat. Másik meglepetés, hogy a MySQL 5-ös verziói a 4-es sorozathoz képest mennyit implementáltak az SQL eszközkészletéből. Aminek – valljuk be – éppen itt volt már az ideje. Nem lepett meg viszont a Microsoft SQL Server implementációja, mely a vizsgált jellemzők tekintetében talán a legnagyobb inkompatibilitást mutatta fel.

Azt a kérdést, hogy miért nem törekednek a teljeskörű kompatibilitásra, nem tudom egyértelműen megválaszolni. Bizonyára a korábbi verziókkal való kompatibilitás az egyik ok. A másik talán az, hogy sok esetben akkor történik meg a szabványosítás, amikor már sok rendszer a saját megoldását implementálta és ettől bizonyára nem könnyű megválni. A harmadik ok – amit igazából félve említek meg tekintettel arra, hogy erős kritikája a piacvezető cégeknek –, hogy egy kereskedelmi adatbázis-kezelő rendszert készítő cégnek nem érdeke, hogy egy elkészült alkalmazás más adatbázis-kezelő rendszerekkel is együttműködjön. Talán ezt támasztja alá az a tény is, hogy sorra jelentek meg az ugyan korlátozott tudású, de ingyenes verzióikkal abban a piaci szegmensben (kis és közép vállalkozások számára készített információs rendszerek) ahol elsősorban az ingyenes rendszerek használata volt elterjedt és talán indokolt is.

Mindezektől függetlenül arra a kérdésre, hogy megvalósítható-e adatbázis független adatbázis-kezelés apró fenntartásokkal igennel felelhetünk. Mivel az adatbázis függetlenséget két szinten definiáltam, ezért a megvalósíthatóságot is külön-külön vizsgálom meg.

Az adatbázisok egységes elérését biztosító interfészek remek megoldásnak tűnnek, amennyiben minden olyan adatbázis-kezelő rendszerhez elérhető meghajtó, melyek felmerülnek célrendszerként. Amennyiben valamilyen speciális vagy nem túl elterjedt adatbázis-kezelő rendszert kívánunk használni, akkor saját megoldás után kell nézni. Ilyen megoldás lehet saját adatbázis-kezelő könyvtár létrehozása, mely plugin rendszerű. Vagyis a használni kívánt adatbázis kezelőnek megfelelő könyvtár – mely természetesen az adott adatbázis-kezelő rendszer eléréséhez mellékelt speciális függvénykönyvtárakat használja fel – használatával történik az adatbázis elérése. Megfontolásra javaslom ezt a megoldást akkor is, ha a szóbajöhető adatbázis-kezelő rendszerek mindegyikéhez van elérhető meghajtó. Az alkalmazás – az alatta elhelyezett újabb réteg segítségével – magától az egységes elérést biztosító adatbázis interfésztől is függetleníthető, ami egyrészt támogatja az adatbázis-kezelő rendszertől való nyelvi függetlenség megvalósítását, másrészt segítheti az alkalmazás más platformra való átültetését is.

A nyelvi függetlenség megvalósítására három megoldást láthattunk. Az egyik megoldás az objektumok perzisztens tárolását megvalósító Hibernate HQL nyelve. Ez bizonyos alkalmazás típusok esetén tökéletes megoldás lehet, de nem biztos, hogy például egy erős riport-támogatású információs rendszerben ez megfelelő lenne. A LINQ to SQL is hasonló elveket vall, de a másik irányból közelíti meg a problémát. Elsősorban nem az objektumok relációs adatbázisban történő perzisztens tárolására született, hanem az adatbázis adatok objektumokra való, az üzleti logikát szem előtt tartó leképezés a célja. A harmadik megoldás, mely a nyelvi függetlenség szempontjából talán a legközelebb áll az elképzeléseimhez az SAP megoldása. Azonban az ilyen megoldások saját megvalósítása nem kevés befektetést igényel, melynek megtérülése csak igen nagy rendszerek esetén valószínű. Valójában meglehetősen szűk azoknak az alkalmazásoknak a köre, melyeknél elkerülhetetlen az SAP Open SQL-jéhez hasonló megoldás. Ettől függetlenül jó szolgálatot tehetne egy nyílt forrású (vagy akár kereskedelmi) SQL nyelvi fordító megvalósítás, de az SQL implementációk vizsgálata után sajnos arra a megállapításra jutunk, hogy az egyes implementációk között nem csak nyelvi, pontosabban szintaktikai eltérések vannak. A szemantikai eltérések azok, melyek megnehezítik az ilyen jellegű általános megoldások elkészítését. Másik nagy probléma, hogy maguknak az SQL utasításoknak a vizsgálatával nem tudhatjuk meg egy-egy mező típusát, mely például a konkatenáció operátora esetén feltétlenül

szükséges azoknál az implementációknál, melyek nem rendelkeznek automatikus konverzióval. Ezek az akadályok legyőzhetők, de annak aki a nyelvi fordító megvalósítását célul tűzi ki mindenképpen komoly befektetéssel kell számolnia. Ha már a befektetés szóba került felmerül annak a kérdése, hogy megtérülhet-e egy ilyen beruházás. Abban az esetben, ha valaki a vállalatirányítási rendszerek piacára akar betörni, vagy kereskedelmi szoftver-komponensként kívánja értékesíteni, akkor nyilvánvalóan igen. Minden más esetben nemmel kell válaszolnunk. Talán az még elképzelhető, hogy egy nyílt forrású projekt kereteiben megjelenhet, hiszen az ilyen típusú fejlesztések egyre gyakoribbak. Ameddig ilyen nem áll rendelkezésre az egységes elérésnél említett saját adatbázis réteg megvalósítása alkalmazható. Ekkor a nyelvi konverzió igazából megvalósítható célfüggvények segítségével, melyek paramétereinek felhasználásával az adatbázis-kezelő rendszer által használt nyelvjárásnak megfelelően állítható össze az SQL utasítás.

Ábrák jegyzéke

Irodalomjegyzék

[1] Kende - Kotsis - Nagy: *Adatbázis-kezelés az ORACLE rendszerben*
Panem, Budapest, 2002

[2] Jeffrey D. Ullman - Jennifer Widom: *Adatbázisrendszerek*
Panem, Budapest, 1998

[3] Raffai Mária: *Információs rendszerek tervezése - Fizikai szint*
Panem, Budapest, 1998

[4] Nelson M. Mattos: *SQL99,SQL/MM, and SQLJ: An Overview of the SQL Standards*
IBM

[5] Jim Melton: *SQL:1999 - A tutorial*
Oracle

[6] Krishna Kulkarni: *Overview of SQL:2003*
Silicon Valley Laboratory, IBM Corporation

[7] Troels Arvin: *Comparison of different SQL implementations*

[8] ANSI/ISO/IEC IS: *Database Language SQL - Part 1: SQL/Framework*

[9] ANSI/ISO/IEC IS: *Database Language SQL - Part 2: SQL/Foundation*

[10] SAP AG: *Introduction to the ABAP Workbench - BC400 tanfolyami jegyzet*

[11] Christian Bauer, Gavin King: *Java Persistence with Hibernate*
Manning Publications Co., New York, 2007

[12] Balássy Gyögy: *LINQ to SQL, avagy vége a DAL-nak*, Visual Studio „Orcas” Konferencia

[13] PostgreSQL felhasználói dokumentáció
http://www.postgresql.org/docs/8.2/static/index.html

[14] IBM DB2 felhasználói dokumentáció
http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp

[15] Microsoft SQL Server felhasználói dokumentáció
http://msdn.microsoft.com/en-us/library/ms189826.aspx

[16] MySQL felhasználói dokumentáció
http://dev.mysql.com/doc/refman/5.0/en/

[17] Oracle felhasználói dokumentáció
http://download.oracle.com/docs/cd/B19306_01/server.102/b14200/toc.htm

[18] Perl DBI felhasználói dokumentáció
http://search.cpan.org/ timb/DBI/DBI.pm

[19] Pear MDB2 felhasználói dokumentáció
http://pear.php.net/package/MDB2