Adatmodellezés Alzabóval

Reuven ebben a hónapban egy kis kitérőt tesz, hogy megmutassa, miképpen hidalhatjuk át az objektumrelációs adatbázis-szakadékot.

Az utóbbi néhány hónapban több szemszögből is megnéztük a kiszolgálóoldali Java-programozás művészetét. A servletektől kezdve a JSP-ken keresztül az Enhydra alkalmazáskiszolgálóig számos módszert ismertünk meg a nyílt forrású Javát használó dinamikus és adatbázis-vezérelt honlapok készítése terén.

Eredetileg e hónapban is ezt az irányvonalat szerettem volna követni, belepillantva az Enhydra érdekes, DODS objektum-relációs modellezőprogramjába. A DODS magas szintű Java-elvonatkoztatási (abstraction) réteget nyújt, amelyen keresztül elérhetők a relációs adatbázis kezelőtáblái.

A DODS-eljárások önműködően a megfelelő SQL-utasítássá alakítódnak, majd az adatbázis-kezelőhöz kerülnek. Az eredmény: Java-objektumokat és -eljárásokat látunk, az adatbázis(kezelő) pedig táblákat és SQL-lekérdezéseket, így mindenki boldog lehet.

A Lutris- (az Enhydra anyacége) dolgozók jó szándéka és segítőkészsége sajnos nem igazán talált partnerre az izraeli kiszállítócégekben és helyi FedEx-társaságban. A CD és a könyv – amely a DODS-ról szóló további leírásokat tartalmazza – e sorok írása közben is valamelyik raktárban pihen, ezért kénytelen vagyok eredeti tervemtől eltérni.

Az e havi cikk elkészítéséhez a DODS körül vizsgálódtam, ugyanis az érdeklődésemet felkeltette az objektumvonatkozású leképezés témaköre. Az egyik legérdekesebb és legkönnyebben használható eszköz, amit e témakörben láttam, az Alzabo, amely tulajdonképpen egy Perl-modulgyűjtemény. Ez teszi lehetővé a kiszolgálóoldali Perl-programozóknak, hogy relációs adatbázisaikat objektumok formájában képezzék le (a projekt a Gene Wolfe című sci-fi egyik teremtménye után kapta a nevét). Teljesen lenyűgözött, amit láttam, és biztos vagyok benne, hogy számos Perl-programozó ugyanilyen boldog, ha ilyen hatékony eszközre bukkan.

A nehézség

A programozók az objektumok használatával számos előnyre tehetnek szert: az újrafelhasználhatóságtól kezdve az öröklésen át az egymásba ágyazhatóságig. Csakhogy míg a programozók tömegesen állnak át objektumközpontú programozásra, az objektumalapú adatbázisok különféle okokból kifolyólag sokkal kevésbé népszerűek. Ehelyett az utóbbi pár évben a relációs adatbázisok váltak egyre népszerűbbé, hiszen időközben hihetetlen mennyiségű adatot halmoztak fel bennük. A feladat ezután természetesen az, miképpen modellezzük táblában tárolt adatainkat objektumként. Az egyik lehetőség, hogy minden táblát külön osztályként modellezünk, minden táblaoszlopot egy-egy példányváltozónak tekintünk, és minden sort az osztály példányaiként fogunk fel. Csakhogy bárki, aki már próbálta, tudhatja, hogy mennyivel könnyebb ezt mondani, mint megvalósítani, különösképpen webalkalmazások készítése során, például hogyan tudunk egyesíteni (`join`) két táblát? Mi történik, ha két program ugyanazt a sort fogja módosítani a memóriában, és csak később írják vissza (`commit`) ezeket a változásokat az adatbázisba? Hogyan tudjuk biztosítani, hogy az adatbázis tükrözze az osztálymeghatározások változtatásait is, és fordítva?

Másik lehetőségként az egész táblát egyetlen objektumpéldányként olvassuk be, módosítjuk az objektumot, majd visszaírjuk, amikor valamilyen eljárást meghívunk. Ez kis táblák esetében nagyon jól működik, de mi történik, ha a tábla néhány megabájt (vagy gigabájt, esetleg terabájt) méretű? Főnökünk esetleg vesz még egy kis memóriát a webkiszolgálóba, de nem azért, hogy teljes táblák memóriába olvasására pocsékoljuk! Emellett ha a táblákat az objektumokban modellezzük, megfelelő zárolási mechanizmust is be kell építenünk a `commit` és `rollback` szolgáltatásokkal, és ennek végrehajtásához a felkészült programozó még kevés.

Amíg kis alkalmazásokkal dolgozunk, könnyedén megbirkózhatunk az ilyen nehézségekkel, de amint az alkalmazás és az adatbázis növekszik, meg szeretnénk bizonyosodni róla, hogy az események a várakozásoknak megfelelően zajlanak. Ez különösképpen igaz egy olyan objektumrelációs leképezőrendszer esetében, mint az Alzabo. Tavaly az egyik alkalmazottammal egy egyszerű objektumrelációs leképező középréteget (middleware layer) készítettünk, és nagyon elégedettek voltunk azzal, amit csináltunk – egészen addig, amíg rá nem jöttünk, hogy közel sem minden lehetőséget vettünk számításba, így végül kivételek és alapértékek tömegével lettünk gazdagabbak.

Minden Perl-programozó szerencséjére, egyikünk, ***Dave Rolsky*** vette a fáradságot, leült és az összes ilyen gondot megoldotta, illetve számos egyéb feladattal is megbirkózott. Az Alzabo objektumközpontú középréteget képez, amely feleslegessé teszi, hogy közvetlenül az adatbázissal kelljen bajlódnunk. Az Alzabo azonban többet nyújt pusztán magas szintű adatbázis-kezelő felületnél. Lehetőséget ad az adatbázisséma-meghatározások programozott megváltoztatására, ideértve a böngészőalapú táblakészítést, valamint az olyan karbantartó eszközöket, amelyek az SQL-parancsokat önműködően állítják elő a számunkra. Ezenkívül az Alzabo képes egy már létező adatbázis visszafejtésére is, lehetővé téve, hogy ezeket az adatbázisainkat is ugyanúgy kezelhessük Alzabóval, mint az újonnan készítetteket.

Az Alzabo telepítése

Mint a legtöbb Perl-modul, az Alzabo is letölthető a CPAN-ról. A telepítése azonban valamivel bonyolultabb feladat, mint más moduloké, mert az Alzabó számos egyéb modultól függ. Nemcsak a DBI (az adatbázis-eléréshez), a ***DBD::mysql*** vagy a ***DBD::Pg*** (PostgreSQL-eléréshez) használatát követeli meg, hanem a böngészőalapú sémakészítéshez a ***HTML::Mason***-t is használja, ennek pedig szüksége van a mod_perl-re. Ha a

rendszer mindezeket már tartalmazza, az Alzabo telepítése sem lesz túl bonyolult.
Minden nehézség nélkül sikerült telepítenem az Alzabót azáltal, hogy a megfelelő CPAN-modulokat letöltöttem és önműködően telepítettem, majd magával az Alzabóval is így jártam el.
A program beállítása során csaknem minden esetben az alapértelmezett értéket fogadtam el, kivéve a ***.mhtml*** utótag-beállítást, amelyet az Alzabo alapértelmezés szerint a Mason-elemekhez feltételez. A Mason-elemeknek általában az egyszerű ***.html*** utótagot adom, ugyanis Apache rendszerem nem tud mit kezdeni a ***.mhtml*** kiterjesztéssel és ***Content-type text/plain*** tartalomtípus szerint küldi el, ezért a böngésző ablakában a Mason-alkalmazás forráskódja jelenik meg. Rendszeremen a telepített Mason-alkalmazások utótagjának ***.html***-re történő módosítása megoldotta ezt a gondot, de az is igaz, hogy ugyanilyen könnyűszerrel változtathattam volna meg a Mason- vagy az Apache beállításokat is.
Az Alzabo minden sémát a saját könyvtárában követ nyomon. Alapesetben ez a könyvtár a ***/usr/local/alzabo***. Ebben a könyvtárban található a sémakönyvtár – minden egyes Alzabo által modellezett adatbázissémához egy-egy alkönyvtárral. Például az ***appointments*** sémák a ***/usr/local/alzabo/schemas/appointments*** könyvtárba kerülnének.
Mindössze két buktatóval találkoztam az Alzabo telepítése során, amelyeket meg kellett szüntetnem. Elsőként meg kellett változtatnom a ***/usr/local/alzabo*** könyvtár jogosultságát, hogy a webfelhasználóm képes legyen olvasni és írni. Másodikként pedig a PostgreSQL-indítófájlt kellett módosítanom, hogy a `-i` kapcsolót tartalmazza, és az ügyfél TCP/IP-hálózaton keresztül is csatlakozhasson. Alapértelmezés szerint a legtöbb PostgreSQL-telepítés (ideértve az RPM-változatokat is) nem állítja be a `-i` kapcsolót, ami azt jelenti, hogy még a legegyszerűbb beállítások sem fognak működni a ***pg_hba.conf***-ban (a PostgreSQL-elérést szabályozó fájlja). Bár egyébként Unix-socketek segítségével hálózat nélkül is hozzá tudunk kapcsolódni a PostgreSQL-hez, az Alzabo mindig megadja a gazdagép nevét, ami azt eredményezi, hogy még a helyi hálózaton elhelyezkedő gép használata esetén is hálózati kapcsolat épül fel.
A webalapú sémakészítő telepítéséhez Apache-kiszolgálónk legalább egy könyvtárát a ***HTML::Mason***-nak kell irányítania. Az Alzabo telepítőfájl egy új ***new/alzabo*** alkönyvtárat fog készíteni azokkal a Mason-elemekkel együtt, amelyek az általunk készítendő sémameghatározásokat fogják felépíteni és módosítani. Munkaállomásomon például az összes ilyen Mason-elem a ***/usr/local/apache/mason*** könyvtárban található, amely a ***/mason*** kezdetű URL-ekhez van rendelve. Alzabo-telepítésem tehát a ***/usr/local/apache/mason/alzabo*** könyvtárban található, és a ***/mason/alzabo*** URL-en keresztül érhető el. Ha eddig még nem tettük volna meg, az Apache-nak a `DirectoryIndex` segítségével megmondhatjuk, hogy az ***index.mhtml*** elfogadható a könyvtárhoz rendelt kezdőlapként.

Sémák szerkesztése

Az Alzabo telepítése után a böngészőalapú tervezőeszközzel készítsünk egy egyszerű adatbázissémát. Kétségtelenül nem olyan tiptop, mint az üzleti vagy az ügyféloldali eszközök, viszont jól megfelel a célnak.
Kezdjük egy új séma készítésével (PostgreSQL vagy MySQL értelemben adatbázissal), amelynek természetesen nevet kell adnunk. A sémának a PostgreSQL és az MySQL szerint is érvényes adatbázisnévnek kell lennie. A PostgreSQL-t választottam a beépített hivatkozásépség (referential integrity), az idegen kulcsok (foreign keys), a nézetek (views) és a kioldók (triggers), illetve az általánosabb SQL-nyelvjárás azon képessége végett, mert így számos nyelven nyílik lehetőségünk belső eljárások írására.
Készítsünk egy egyszerű telefonkönyvet és találkozónaptárat az Alzabo segítségével! Nyomon követhetjük az általunk ismert embereket, a címüket és telefonszámukat, illetve a velük megbeszélt találkozók időpontjait. Ezen adatbázis segítségével megtudhatjuk, milyen emberekkel kell találkoznunk egy adott napon, illetve lekérdezhetjük egy adott emberrel megbeszélt összes találkozásunkat.

➲ sourcefog.net/projects/alzabo

A séma elkészítéséhez a böngészőt az ***alzabo/schema*** URL-re kell irányítanunk az imént említett Mason-könyvtár alatt (tehát a gépemen a böngészőt a ➲ http://localhost/mason/alzabo/schema címre kellett irányítani). Ez felhozza a sémakészítő és szerkesztő lapot, amely lehetővé teszi, hogy egy már létező sémát átszerkesszünk, újat készítsünk vagy a már létezőt visszafejtsük. Az utolsó lehetőség a legérdekesebb, hiszen megengedi, hogy az Alzabo segítségével az öröklött adatbázishoz is hozzányúljunk. Mi most új sémát fogunk készíteni. Be kell gépelünk a nevet (jobb ötlet híján az addressbook – címjegyzék – nevet választottam), illetve jelölnünk kell, hogy a PostgreSQL-t fogjuk háttéradatbázisként használni.
A ***Submit*** gombra kattintás után több lehetőség közül választhatunk: hozzáadhatunk egy új táblát a sémához, törölhetjük az egész sémát vagy megnézhetjük az SQL-sort, amelyet az Alzabo önműködően hozott létre. Egyelőre természetesen még semmilyen megjeleníthető SQL-kód nincsen, idővel azonban láthatjuk majd, ahogy az SQL-kód egyre növekszik.
Az Alzabo ugyan semmilyen SQL-sort nem készített még, de ez nem azt jelenti, hogy a háttérben nem is végzett semmilyen előkészítő munkát. Valójában az Alzabo a ***/usr/local/alzabo/schemas***-ban önműködően elkészítette a címtárkönyvtárat, amely három fájlt tartalmaz: az ***addressbook.create.alz***-t, az ***addressbook.runtime.alz***-t (mindkettőt futtatható formátumban) és az ***addressbook.rdbms***-t, amely egyetlen „PostgreSQL" szót tartalmaz. Ezzel a módszerrel az Alzabo nyomon követheti az adatbázis-kiszolgálót, ahol a séma tárolódik.
A címjegyzéksémába a belépést követően az ***Add a table*** szövegmezőbe az „Emberek" szót begépelve, majd a ***Submit***

gombra kattintva az „Emberek” táblát hozzáadtam az adatbázishoz. (A PostgreSQL ugyan figyelmen kívül hagyja a kis- és nagybetű különbségeit a tábla- és mezőnevekben, de én a Táblanevek kezdőbetűit illetőleg ***Joe Celko*** szabályait szeretem alkalmazni, illetve a mezőneveket csupa kisbetűvel írom, és csupa nagybetűvel az SQL FENNTARTOTT SZAVAIT.)
Az Emberek táblában létrehoztam néhány különböző típusú mezőt. Az Alzabo menüben ajánlja fel a leggyakrabban használt adattípusokat, de akár saját készítésűt is begépelhetünk, ha kívánjuk; ez különösképpen PostgreSQL alatt lehet hasznos, mivel lehetővé teszi a számunkra, hogy saját adattípusokat hozzunk létre.
Az ilyen táblák esetében általában mesterséges elsődleges kulcsokkal szeretek dolgozni, minden egyes sorhoz egyedi értéket rendelve. PostgreSQL alatt ezt a `SERIAL` adattípussal adhatjuk meg. Rögtön észrevehetjük, hogy az Alzabo választéklistájában nincs ilyen típusú mező. Esetleg kísértést érezhetünk, hogy a mezőt `INTEGER` típusúnak jelöljük meg, és bekapcsoljuk hozzá a „sequenced” jelölőnégyzetet a mezőszerkesztő alsó részén. Ilyenkor ugyanis egy `INTEGER`-típusú oszlop, és egy ettől teljesen független PostgreSQL-szekvencia jön létre. Ezért a mesterséges elsődleges kulcshoz a `<select>` mezőtípuslista alatti szövegmezőbe a `SERIAL` kulcsszót inkább kézzel gépeljük be.
Egy további jelölőnégyzet segítségével az oszlopot elsődleges kulcsként adhatjuk meg, amely ezt követően `pk` jelöléssel szerepel a mezőlistában. Végül a harmadik jelölőnégyzet mutatja meg, hogy a mező tartalmazhat-e `NULL` értéket. Figyelmeztetek minden kezdő adatbázis-tervezőt, hogy a `NULL`-ok, azaz az üres mezők túlbonyolítják az életünket, és amikor csak lehetséges, célszerű elkerülnünk a használatukat.
Az idegen kulcsok (`REFERENCES` vagy `CHECK` záradék) megadásához a megfelelő sort a HTML-űrlap alsó részén adjuk az ***attributes*** szövegmezőhöz. Ne feledjük, a Perlben ezen a ponton még csak modellezzük a sémát, ami azt jelenti, hogy bármilyen záradékot később is hozzáadhatunk vagy eltávolíthatunk, anélkül, hogy az adatbázisnak `ALTER TABLE` lekérdezéseket kellene küldenünk. Az Alzabo szerkesztőjével egy vagy több oszlopra indexeket is készíthetünk.
Az Alzabo tábla- és oszlopszerkesztőjével több táblát is gyárthatunk, amelyek között a többszintű menü- és listarendszer segítségével mozoghatunk. Az Alzabo minden egyes oszlop mellett megjelenít egy „<” és „>” jelet, amelyekkel egymáshoz képest az adott meghatározásnak megfelelően mozdíthatjuk el őket.
Ha böngészőalapú sémaszerkesztővel dolgozunk, javaslom, vessünk néha egy pillantást az Alzabo által készített SQL-kódra is. Nemcsak azért, hogy meggyőződjünk róla, hogy az Alzabo helyesen építi-e fel (emlékezzünk a `SERIAL` oszlopra), hanem mert így jobb képet kapunk az éppen készülő séma alacsony szintű szerkezetéről.
Ha elkészültünk a sémával, használjuk az ***execute SQL*** gombot az ***SQL preview*** űrlapon, és az SQL-lekérdezés(eke)t küldjük el az adatbázis-kiszolgálónak. Ha az adatbázis-kiszolgáló hibaüzenetet küld, az Alzabo hosszú és részletes hibaüzenetet fog visszaadni, amely leírja, mi is történt tulajdonképpen.
Néhány esetben a tábla- vagy mezőmeghatározásokat kell módosítanunk, ám az is előfordulhat, hogy a kiszolgáló megfelelő jogosultságbeállításaival akadt gond. Arról is győződjünk meg, hogy létrehoztunk-e olyan PostgreSQL-felhasználót (a parancssoros `createuser` programmal), akinek a neve megegyezik azzal a névvel, ami alatt az Apache fut – hacsak nem akarunk a HTML-űrlapon közvetlen módon más felhasználót megadni.

Sémahasználat programból

Miután az SQL-t futtattuk a sémaszerkesztőben, két lehetőségünk kínálkozik az adatok elérésére. Természetesen elérhetjük őket közvetlenül a DBI-n keresztül (vagy hasonló felületen más nyelvek esetében), ha létrehozzuk és végrehajtjuk az SQL-lekérdezéseket.
Például a címjegyzéksémát a következő táblának megfelelően építettem fel:

```
CREATE TABLE Emberek (
    szemely_id    SERIAL    NOT NULL,
    first_name     TEXT      NOT NULL,
    last_name    TEXT      NOT NULL,
    birthday          DATE      NOT NULL,
    PRIMARY KEY(person_id)
);
```

Tegyük még érdekesebbé, ezért gépeljünk pár adatot is a táblába:

```
INSERT INTO Emberek (first_name, last_name,
                     ➥birthday)
VALUES (·Reuven·, ·Lerner·, ·1970-Jul-14·);

INSERT INTO People (first_name, last_name,
                    ➥birthday)
VALUES (·Atara Margalit·, ·Lerner-Friedman·,
       ➥·2000-Dec-16·);
```

Az *1. lista* egy egyszerű Perl-programot tartalmaz, amely DBI-t használ a nevek (és születésnapok) lekéréséhez, tehát azon emberek neveinek (és születésnapjainak) a lekérésére a címtárból, akik megegyeznek a parancssorba gépelt SQL-mintával. (Az SQL-minták sokkal egyszerűbbek, mint a Unix szabványos kifejezései és mindössze két karaktert használnak: a % minden nulla vagy hosszabb karaktersorozattal, a _ pedig pontosan egyetlen karakterrel egyezik.)
A felhasználó által beírt bemenetet beolvassuk a parancssorból, elé és utána % jeleket helyezünk, így a karaktersorozat találatot ér el, függetlenül attól, hogy hol található meg a `first_name` (keresztnév) vagy a `last_name` (vezetéknév) oszlopban.
Ezután csatlakozunk az adatbázishoz, bekapcsoljuk az `AutoCommit` szolgáltatást (miként azt a DBI-leírás ismerteti), és felélesztjük a `RaiseError` és `PrintError` hibaelemző eszközöket.
Végül a `$sql` változóban előállítjuk az SQL-lekérdezést, nem feledve, hogy a közvetlen változóbehelyettesítés helyett inkább helyettesítő karaktert (?) használunk . Ez nemcsak annak a veszélyét csökkenti, hogy valaki belerondít az SQL-ünkbe, de néhány adatbáziskezelő-vezérlő a következő lekérdezésben ki is használja a helyettesítő karaktereket, ez pedig jelentős sebességnövekedést eredményezhet.

Ugyanez Alzabóval

Most írjuk újra a fenti programot a közvetlen DBI-használat helyett az Alzabo segítségével. Nem fogunk közvetlenül SQL-t írni, és az adatbázishoz sem csatlakozunk. Ehelyett új sémaobjektumot készítünk, elnevezve az Alzabo interaktív eszközével készített sémát. Ez az objektum számos eljárással rendelkezik, a segítségükkel csomó olyan feladatot is elvégezhetünk, amelyekhez egyébként a DBI-t kellene használnunk.
A két változat között nem sok különbséget találunk – miként

1. lista retrieve-today-birthday.pl, amely DBI segítségével keresi vissza a címjegyzéktáblából azoknak az embereknek a neveit, akiknek ma van a születésnapjuk

```
#!/usr/bin/perl

use warnings;     # Perl 5.6.x version of
                  # "-w" option
use strict;
use DBI;

# beÆll tunk nØhÆny alapvÆltoz t
my $dbname = ·addressbook·;
my $username = ·reuven·;
my $password = ··;

# milyen nØvre keres nk?
my $look_for_name = $ARGV[0];
die "Nem  rtÆl be keresendı nevet!"
    unless $look_for_name;

# hozzÆkapcsol dunk az adatbÆzishoz
my $dbh = DBI->connect
               ("dbi:Pg:dbname=addressbook",
                $username, $password,
                {RaiseError => 1,
                PrintError => 1,
                AutoCommit => 1});

# Ellenırizz k, hogy sikeresen csatlakoztunk-e
# az adatbÆzishoz
die "No connection to the database:
    ➥·$DBI::errstr· "
    unless $dbh;

# %-jel ragasztÆsa el lre Øs hÆtra az SQL
# regexp elıÆll tÆsÆhoz
$look_for_name = "%" . $look_for_name . "%";

# SQL-lekØrdezØs kØsz tØse
my $sql = "SELECT first_name, last_name,
          ➥birthday ";
$sql .= "  FROM Emberek ";
$sql .= "  WHERE (first_name LIKE ?)
           ➥OR (last_name LIKE ?) ";

# felkØsz lØs a lekØrdezØshez
my $sth = $dbh->prepare($sql);

# lekØrdezØs vØgrehajtÆsa
$sth->execute($look_for_name, $look_for_name);

# vØgiglØpked nk az eredmØnysorokon
while (my $row_ref = $sth->fetchrow_arrayref)
{
    my ($first_name, $last_name, $birthday)
      ➥= @{$row_ref};
    print "$first_name $last_name
           ➥(birthday: $birthday)\n";
}

# Ellenırizz k, hogyan j ttek vissza a sorok
if ($sth->rows == 0)
{
    print "Nem volt talÆlat a c mtÆrban.\n";
}

# vØgezt nk az SQL-rØsszel
$sth->finish;

# lezÆrjuk a kapcsolatot Øs az adatbÆzist
$dbh->disconnect;
```

azt a 2. *listában* láthatjuk –, amíg hozzá nem kapcsolódunk az adatforráshoz. A DBI programváltozatban az adatforráshoz a `DBI->connect` paranccsal kapcsolódtunk; az Alzabóban ezzel szemben a sémához, amely viszont feltételezhetően az adatbázishoz kapcsolódik, és hozzárendeljük a `$schema` objektumhoz.
A `$schema` használatával az egyik táblánkhoz tartozó tábla-objektumot szerezhetjük meg:

```
my $emberek = $schema->table("Emberek");
```

Most, hogy az Emberek táblához tartozó objektumunk már létezik, könnyűszerrel hozzáférhetünk a tábla kiválasztott soraihoz. A sorok lekérésére a legegyszerűbb mód a `rows_where` eljárás használata, ezáltal egyetlen `Alzabo::Runtime::RowCursor:` típusú objektumot kapunk vissza.

```
my $row_cursor = $people->rows_where
   ➥(where => [[$people->column(·first_name·),
   ➥·LIKE·, $look_for_name], ·or·,
   ➥[$people->column(·last_name·), ·LIKE·,
   ➥$look_for_name]]);
```

Az Alzabo `WHERE`-szerkezetei általában három elemet tartalmaznak: az oszlopobjektumot, az összehasonlító műveleti jelet és az értéket, vagy egy másik oszlopobjektumot. Végigvizsgálhatjuk például a keresztnévoszlopot, hogy akad-e valahol ***Ferenc*** értékű mező:

```
where => [$table->column(·first_name·),
         ➥·=·, ·Ferenc·]
```

A 2. listában ezt egy kicsit tovább bonyolítottuk: a két vektorhivatkozást az `OR` logikai műveletjellel kapcsoltuk össze:

```
where => [[$people->column(·first_name·),
         ➥·LIKE·, $look_for_name], ·or·,
         ➥[$people->column(·last_name·),
         ➥·LIKE·, $look_for_name]]
```

Az Alzabo elég okos ahhoz, hogy felismerje: `WHERE` meghatározása első és harmadik eleme vektorhivatkozás, így a fenti kódot a megfelelő SQL-utasítássá formálja.
Ha hozzáfértünk a `RowCursor` objektumhoz, a `next_row` eljárással játszi módon lépdelhetünk végig az egyes sorokon:

2. lista retrieve-birthday-alzabo.pl, az 1. lista programjának Alzabo-változata

```
#!/usr/bin/perl

use warnings;
use strict;
use Alzabo::Runtime::Schema;

# beÆll tunk nØhÆny alapvÆltoz t
my $schema_name = ·addressbook·;
my $username = ·reuven·;
my $password = ··;

# milyen nØvre keres nk?
my $look_for_name = $ARGV[0];
die "Nem  rtÆl be keresendı nevet!"
    unless $look_for_name;

# %-jel ragasztÆsa el lre Øs hÆtra az SQL
# regexp elıÆll tÆsÆhoz
$look_for_name = "%" . $look_for_name . "%";

# sØma bet ltØse a lemezrıl
my $schema =
    ➥Alzabo::Runtime::Schema->load_from_file
    ➥( name => $schema_name );

$schema->set_user( $username );
$schema->set_password( $password );
$schema->connect;

# megszerezz k a tÆblaobjektumot, amin
# dolgozni szeretnØnk
my $people = $schema->table("Emberek");

# az  sszes, a lekØrdezØs nkkel egyezı sor
# begyßjtØse
my $row_cursor = $emberek->rows_where
    (where => [[$emberek->column(·first_name·),
                ➥·LIKE·, $look_for_name],
                ➥·or·,
               ➥[$emberek->column(·last_name·),
                  ➥·LIKE·,
                  ➥$look_for_name]]);

my $rows_returned = 0;

# vØgiglØpked nk a sorokon a cursor seg tsØgØvel
while (my $row = $row_cursor->next_row)
{
    my $first_name = $row->select(·first_name·);
    my $last_name = $row->select(·last_name·);
    my $birthday = $row->select(·birthday·);

    print "$first_name $last_name
           ➥(birthday: $birthday)\n";

    $rows_returned++;
}

# jelezz k, ha volt hiba
if ($rows_returned == 0)
{
     print "Nem volt talÆlat a c mtÆrban.\n";
}
```

```
while (my $row = $row_cursor->next_row)
{
   my $first_name = $row->select(·first_name·);
   my $last_name = $row->select(·last_name·);
   my $birthday = $row->select(·birthday·);

   print "$first_name $last_name
         ➥(birthday: $birthday)\n";

   $rows_returned++;
}
```

Gyorstárak és kivételek

Ha az Alzabo mindössze csak néhány SQL-készítő eljárást nyújtana, nem volna valami hatékony eszköz. Az Alzabo azonban a csomag részeként lehetőséget ad a gyorstárazásra és a kivételkezelésre is, ami megkönnyíti az adatbázisokkal való munkát.

Az Alzabo gyorstárazó eszköze a memóriában tartja a táblát ahelyett, hogy minden egyes lekérést rögtön az adatbázis-kiszolgálóhoz továbbítana. Nyilvánvaló, hogy a gyorstár használata az olyan táblák esetében nem megfelelő, amelyek rendszeresen megváltoznak, viszont ritkán változó táblák esetén nyugodtan beindíthatjuk, és élvezhetjük a kellemes sebességnövekedést.

A gyorstárazást az `Alzabo::ObjectCache` modul programba töltésével indíthatjuk be. A `RowCursor` objektum, amelyet a *2. listában* a sorok lekérésére használtunk, a `next_row` eljárás minden egyes újabb ismétlésénél `Row` objektumokat ad vissza. A különféle elérhető gyorstártípusokról és a velük kapcsolatos tudnivalókról az ***Alzabo::Runtime::Row*** és ***Alzabo::ObjectCache*** leírásában további adatok találhatók.

Az Alzabo a Perl beépített kivételkezelő rendszerét is használja, azaz ha valami nem megfelelően működik, meghívja a `die` parancsot. Így Alzabo- programjainkat (vagy akár a bennük található egyedi hívásokat) `eval`-blokkokba csomagolhatjuk be:

```
# pr bÆljuk meg futtatni ezt a k dot
eval {
    my $row_cursor = $people->rows_where(
    ➥where => [[$people->column(·first_name·),
    ➥·LIKE·, $look_for_name], ·or·,
    ➥[$people->column(·last_name·), ·LIKE·,
    ➥$look_for_name]]);
};
```

A `$@` Perl-változó vizsgálatával megtudhatjuk, ha valami nem megfelelően működött, mivel ez a változó akkor kerül beállításra, ha az előző `eval` során valamilyen hiba lépett fel. Az Alzabo használja az `Exception::Class` objektumot is

(amely a CPAN-on érhető el), így még kifinomultabb Perl-kivételkezelést képes nyújtani. A `$@` változóba nem a hibát leíró karaktersorozat kerül, hanem a megfelelő kivételosztály példánya. Így aztán a `$@`-t kipróbálhatjuk az `UNIVERSAL::isa` feltétellel, hogy megállapítsuk, miféle objektum is ez, illetve milyen hiba történt a kódunkban. Az Mason által irányított Apache tartalomkönyvtárban található ***Alzabo*** könyvtárba telepített ***common/exception*** Mason-elemek részletesen bemutatják, hogyan is kell ezt megtenni.

Tudnivalók

Nyilvánvalóan hátránya is akad az Alzabo használatának, akárcsak bármely olyan programnak, amely az objektumrelációs szakadékot próbálja meg áthidalni. Először is az SQL meglehetősen jó általános módszer a relációs adatbázisokkal végzett munkához.

Az Alzabo használata azt is jelenti, hogy a szabványtól egy másik megoldás felé távolodunk el, amely semmi mással nem egyeztethető össze. Természetesen nem vagyok ellensége az ilyen megoldásoknak, és számos jelentős előnye létezik az Alzabo használatának, azonban óvatos vagyok, amikor egy régi, bevált megoldás új, szokatlan módszerrel történő helyettesítéséről esik szó.

Bár a tábláimat egyébként kézzel összerakott SQL-utasításokkal szoktam megalkotni, ez a módszer nem alkalmazható tíz vagy húsz tábla felett anélkül, hogy az Emacs átmeneti tárban vad görgetésre ne késztetne. Az Alzabo webalapú sémaszerkesztője megkönnyíti a nagy számú táblák nyomon követését, a köztük lévő kapcsolatok készítését, illetve módosításukat. Előfordult már, hogy félórát töltöttem azzal, hogy megpróbáltam visszaemlékezni, miben különböznek az Oracle formai követelményei a PostgreSQL-éitől ilyenkor nagyon örültem volna egy Alzabo-szerű eszköznek.

Amint korábban láttuk, az Alzabóval könnyen előállíthatók egyenlőségen alapuló összetett lekérdezések, még akkor is, ha `OR` és `AND` műveleteket is tartalmaznak.

Az `Alzabo::Runtime::Table` objektum egy eljárást tartalmaz, amely tetszőleges SQL-függvény futtatására szolgál, viszont az Alzabo `WHERE` részének elkészítését meglehetősen nehéznek, néha egyenesen lehetetlennek találtam. Ennek segítségével többszörös függvényhívásokon alapuló SQL-lekérdezést lehetett volna összeállítani. Bevallom, eléggé újonc vagyok még az Alzabo világában, és mindössze egy vagy két órát próbálkoztam, de ha egy lekérdezést húsz másodpercig tart SQL-ben megírni, akkor Alzabo alatt sem szabadna sokkal tovább készülnie.

Az egyik legnehezebb dolog az objektumok relációs adatbázisának leképezésekor a `join`-ok kezelése. A `join`-ok igen fontosak a táblák esetében, de jelentésük sokkal kevésbé nyilvánvaló, amikor objektumokkal dolgozunk. Az Alzabo rendelkezik ugyan néhány beépített `join`-támogatással, de ezt a részt meglehetősen újként és kísérleti jellegűként jelölték meg.

Végül itt van még a sebesség kérdése, amely minden középréteg esetében felmerül. Az 1. és 2. lista közötti különbség erősen érzékelhető volt, amikor parancssorból hajtottam végre őket, ami jórészt annak köszönhető, hogy az Alzabo igen nagy számú Perl-osztályt olvas be. A ***mod_perl*** környezetben (ahol az Alzabót tervezték), a sebességkülönbség már jóval kisebb, mivel a legtöbb idő a modulok lemezről való betöltésével telik el. Mivel a ***mod_perl*** futtatás előtt csak egyszer fordítja le a programokat, a sebességkülönbség az Alzabo és a nyers DBI-hívások között valószínűleg már nem olyan nagy.

Összegzés

Az Alzabo viszonylag egyszerű módszert nyújt a relációs adatbázistáblák objektumokba burkolására. Számos jó hírt közölhetek ennek kapcsán: az adatmodellező eszköz meglehetősen kifinomult, ügyes képességekkel rendelkezik, az eljárások eléggé értelmesek, a leírás vaskos és jól megfogalmazott. Ahogy a Nyílt Forráskód Közössége régen hangoztatja: ugyanannak a feladatnak a megoldására egy régi, jól bevált, hadviselt és nyílt eszköz majdnem mindig jobb, mint egy új üzleti csomag bevezetése.

Az relációs adatbázistáblák objektumokba csomagolása azonban veszélyekkel és hibákkal terhes, ami alól az Alzabo sem kivétel: a `join`-ok kezelése még mindig hagy némi kívánnivalót maga után, és az sem látható tisztán, hogy néhány lekérdezést miképpen lehet összeállítani. Ez nem az Alzabo hibája: kikerülhetetlen kérdés, ha két olyan módszerrel dolgozunk, amely a világot teljesen eltérő módon látja.

Teljesen nyilvánvaló, hogy az Alzabót a jövőben valamilyen kiszolgálóoldali munkámban fel fogom használni, különösen ha kifinomult gyorstárazásra és kivételkezelésre lesz szükségem, amit egyébként nekem kellene megírnom.

A következő hónapban visszatérünk eredeti, kiszolgálóoldali Java-körutunkhoz, és az Enhydra DODS csomagját az Alzabóval és rokonaival hasonlítjuk össze.

Reuven M. Lerner (reuven@lerner.co.il) kisebb webes és internetes módszerekkel foglalkozó tanácsadó cég tulajdonosa és vezetője. A cikk megjelenésének időpontjában valószínűleg már végleg elkészült Core Perl című könyvével, melyet idén jelentet meg a Prentice-Hall. Az ATF honlapon érhető el (➲ http://www.lerner.co.il/atf/).

További érdekességek

Az Alzabo-modulok elérhetők a CPAN-on keresztül („Alzabo" néven) vagy a projekt honlapján, a SourceForge-on (➲ http://sourceforge.net/projects/alzabo). Ugyanezen a honlapon bőséges leírás is elérhető modulonkénti bontásban, hagyományos perldoc formátumban. Majdnem minden Alzabóval kapcsolatos fejlesztés *Dave Rolsky* nevéhez fűződik, aki igen tevékenyen vesz részt a *HTML::Mason* és a *mod_perl* világában, akárcsak ebben a projektben is.

Az Alzabo nem az első és nem is az egyetlen Perl-rendszer, amely táblákat objektumoknak feletet meg. Egy másik megoldás a Tangram, amit JLL készített és a ➲ http://www.tangram-persistence.org címen érhető el. A Tangram néhány ügyes lehetőséget tartalmaz, az adatbázis-eléréshez pedig kizárólag DBI-t használ, így a Tangram bármely DBD-hez használható, nem csak azokhoz, amelyekhez egyedi alkalmazásvezérővel bír, mint az Alzabo esetében. A Tangram ugyanakkor nem tartalmaz olyan adatbázis-visszafejtő képességet, mint az Alzabo, amelynek segítségével korábbról örökölt adatbázisokkal is dolgozni tudunk.