Gyakorlat – reflexió – innováció

Nevelési-oktatási programok részvételi alapú fejlesztése

VIII. MELLÉKLET

VIII. Melléklet

AZ ADATBÁZISRÓL

RDBMS
A relációsadatbázis-kezelő rendszer (angol rövidítéséből: RDBMS) egy olyan adatbázis-kezelő rendszer, amelynek logikai adatbázisának szoftverkomponensei a relációs adatmodellek elvén épülnek fel, illetve kérdezhetőek le. A relációs adatmodell műveletei között pontosan öt alapműveletet és számos származtatott műveletet definiálhatunk. Az alapműveletek rendre a következők:

- Descartes-szorzat
- Halmazelméleti unió
- Halmazelméleti különbség
- Kiválasztás (szelekció)
- Vetítés (projekció)

Az adatok tárolására és feldolgozására alkalmasnak tűnt egy RDBMS (Relational Database Management System) rendszer használata. Számos ilyen rendszer áll rendelkezésre, azonban a feladatra az ingyenesen elérhető rendszerek közül a PostgreSQL volt a legalkalmasabb. Fizetős alternatívaként az Oracle adatbázis-kezelő rendszer használatát javasolnánk. Ezek a rendszerek számos más szoftver megoldással együtt igen egyszerűvé teszik a jelentések írását, az adattárolást, az adatok lekérdezését és megjelenítését, elősegítik az adatgyűjtés egyszerűsítését, illetve az adatrögzítési hibák kiszűrését.

Az RDBMS rendszerek adattárolási modellje a relációs adatmodell, amely azt jelenti, hogy alapvetően az entitásokat halmazokban tárolja, amelyek úgy képzelhetőek el, mintha táblázatok lennének, egy sor egy entitás attribútumait tárolja. A sorok és oszlopok metszéspontjában lévő elemek a mezők. Két tábla (entitáshalmaz) között létrehozható egy reláció egy kapcsolóoszlopon keresztül. Egy entitás akkor van egy kapcsolóoszlopon keresztül összekapcsolva egy másik táblában lévő entitással, hogyha a kapcsolóoszlop értéke és a hivatkozott érték megegyezik. A relációs adatmodell legalapvetőbb művelete az összekapcsolás, amikor különböző adatok között - a valóságban is létező - kapcsolatot építünk fel az adatmodell szintjén.

Adatmodellezés szempontjából úgy célszerű eljárni, hogy az entitástípusok a való világban létező objektumtípusokat képezzék le, míg a relációk a valós objektumok közötti relációkat reprezentálják. Az eddigiek alapján világosan látszik, hogy a relációs adatmodell egy pontcímkézett gráfosztályt reprezentál. A pontok az entitások, a címkék az attribútumaik. Azok a speciális attribútumok, amelyek hivatkozások, képzelhetők a gráf éleinek. Általános gráfél úgy képezhető, hogy egy új halmazt hozunk létre, amelyben két hivatkozás is van, melyek az él kezdő, illetve végpontját reprezentálják. Mivel ennek is lehetnek egyéb attribútumai, ezért az élcímkézett gráfok is reprezentálhatók így. Mivel minden gráf reprezentálható így, ezért ez a modellezési megközelítés meglehetősen általános.

POSTGRESQL-BEN HASZNÁLHATÓ KOMPLEX ADATTÍPUSOK

Széles a használható adattípusok köre, és ezeken felül saját típust is készíthetünk
CREATE TYPE <típus neve> (típus1, típus2, ...);
segítségével. Az itt használható alaptípusok:

- karakteres alaptípusok fix hosszúságra illetve 1GB határig a hosszúság megkötése nélkül használható text
- numerikus alaptípusok 8-64 bitszélességre fixpontos és lebegőpontos ábrázolással illetve tetszőleges adatszélességre numeric(a,b) típuson keresztül (esetünkben „egész" adattípust használtunk)
- idővel kapcsolatos alaptípusok: dátum; időzóna beállítással és enélkül; időintervallum
- bináris alaptípus (ún. boolean) 3 állapottal: igaz, hamis és üres/nem definiált
- halmaztípus (ún. enum)
- speciális adattípusok: MAC-címek, UUID, IPv4 és IPv6 címek, CIDR címek, bitekre bontott típus (ún. bit string), deviza
- XML adattípus
- geometriai típusok
- mindezen típusok tömb (array) megvalósításban, ahol az elemszámot nem szükséges előre megkötni, hanem szükség szerint növekszik.[1]

A NULL egy speciális érték, melyet bármely adattípus felvehet (kivéve a not null megkötéssel definiált adatok), jelentése: nincsen ott adat. A tanulók nemét meghatározó táblázatban például a NULL érték azt jelenti, hogy nincs információnk az adott kódszámú tanuló neméről. UNIQUE megkötésnél az adott oszlopban csak különböző értékek szerepelhetnek. A konkrét adatmodell a jelen munka során a következő alakot öltötte:

GYEREKEK
- kódszámuk(id) – egész, referenced by valaszok.gyerek, unique, not null, primary key
- intézmény(intezmeny) – egész, foreign key references intezmenyek.kodszam, not null
- évfolyam(class) – egész , not null
- kísérleti(kiserleti) – bool , not null

INTÉZMÉNYEK
- kódszáma(id), egész, referenced by gyerekek.intezmeny, unique, not null, primary key
- név(name) – szöveg, not null
- település(city) – szöveg , not null

KÉRDÉSCSOPORTOK
- kódszám(id), egész, referenced by kerdesek.kerdescsoport, unique, not null, primary key
- szöveg(text), szöveg,not null
- válaszlehetőségek száma(max), egesz,not null
- egyenesskala(skala) – bool, not null

KÉRDÉSEK
- id – egész, referenced by valaszok.kerdes, unique, not null, primary key
- szövege – szövege, not null
- kérdéscsoport – egész, foreing key references kerdescsoportok.kodszam, not null
- alsós – bool, not null

[1] www.wikipedia.org

LÁNYOKFIÚK

- kód(kod), egész,not null, primary key
- lany boolean

VÁLASZOK

- id – egész, unique, not null, primary key
- gyerek(et) – egész,foreign key references gyerekek.kodszam, not null
- kerdes(t) – egész,foreign key references kerdesek.id, not null
- valasz – egész, not null
- bemeneti – bool, not null

Az adatbázis szerkezetének kialakításánál figyelembe vettük a korábban lefektetett adatmodellezési alapelveket.

> **VIEW (NÉZET)**
> Egy olyan tábla, melynek eredményei egy lekérdezés alapján képződnek. A lekérdezés létező táblák vagy más viewk adatai alapján rendszerezetten, szűrten, kapcsoltan jeleníti meg az adatokat. Amennyiben a lekérdezésben érintett adatok az eredeti táblákban megváltoznak, a view eredménye is automatikusan módosul.

Az egyik legfontosabb view a következő volt:

```
 CREATE VIEW osszesito AS
  SELECT gyerekek.id,
     gyerekek.intezmeny,
     gyerekek.class,
     gyerekek.kiserleti,
     bemeneti.kerdes,
     kerdesek.kerdescsoport,
     bemeneti.valasz AS bevalasz,
     kimeneti.valasz AS kivalasz,
     lanyokfiuk.lany
    FROM ((((gyerekek
      JOIN ( SELECT valaszok.id,
             valaszok.gyerek,
             valaszok.kerdes,
             valaszok.valasz,
             valaszok.bemeneti
            FROM valaszok
           WHERE valaszok.bemeneti) bemeneti ON ((bemeneti.gyerek = gyerekek.id)))
      JOIN ( SELECT valaszok.id,
             valaszok.gyerek,
             valaszok.kerdes,
             valaszok.valasz,
             valaszok.bemeneti
            FROM valaszok
           WHERE (NOT valaszok.bemeneti)) kimeneti ON (((kimeneti.gyerek = gyerekek.id)
AND (kimeneti.kerdes = bemeneti.kerdes))))
       JOIN kerdesek ON ((kerdesek.id = bemeneti.kerdes)))
       LEFT JOIN lanyokfiuk ON ((gyerekek.id = lanyokfiuk.kod)))
    WHERE ((bemeneti.valasz IS NOT NULL) AND (kimeneti.valasz IS NOT NULL));
```

Érdekes még a definíció végén látható LEFT JOIN utasítás, amely ahhoz szükséges, hogy megmaradjanak azok az adatok, amelyeknél nem áll rendelkezésre a tanuló neméről információ

A csvlinearize program forráskódja

```
     csvlinearize
         Copyright (C) 2014 Boskovits Gábor

     This program is free software: you can redistribute it and/or modify
     it under the terms of the GNU General Public License as published by
     the Free Software Foundation, either version 3 of the License, or
         (at your option) any later version.

     This program is distributed in the hope that it will be useful,
     but WITHOUT ANY WARRANTY; without even the implied warranty of
         MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  Seethe
         GNU General Public License for more details.

     You should have received a copy of the GNU General Public License
     along with this program.  If not, see<http://www.gnu.org/licenses/>.

#! /bin/bash

infile=$1
outfile=$2

cat$infile | tr-d '\015'| sed 's/$/;;/' >in.csv
./a.out
rmin.csv
mvout.csv$outfile

exit 0

#include<iostream>
#include<fstream>
#include<string>
#include<vector>
usingnamespacestd;

int main(int argc, char* argv[])
{
constchar* infile_name="in.csv";
constchar* outfile_name="out.csv";
constchardelimiter=';';
unsigned int num_of_columns;
unsigned int num_of_rows;
stringtemp;
vector<string>column_header;
vector<string>row_header;
```

```
 ifstreaminfile(infile_name);
 ofstreamoutfile(outfile_name);
 getline(infile,temp,delimiter);
 num_of_columns=stoi(temp);
 cout<<endl;
 cout<<"Number of columns is: "<<num_of_columns<<endl;
 getline(infile,temp,delimiter);
 num_of_rows=stoi(temp);
 cout<<"Number of rows is: "<<num_of_rows<<endl<<endl;
 getline(infile,temp);
 for(int i=0;i<num_of_columns;i++)
  {
 getline(infile,temp,delimiter);
 column_header.push_back(temp);
  }
 getline(infile,temp);
 for(int i=0;i<num_of_columns;i++)
  {
 cout<<"Number "<<i<<" columnheader is "<<column_header[i]<<endl;
  }
 cout<<endl;
 for(int i=0;i<num_of_rows;i++)
  {
 getline(infile,temp,delimiter);
 row_header.push_back(temp);
  }
 getline(infile,temp);
 for(int i=0;i<num_of_rows;i++)
  {
 cout<<"Number "<<i<<" rowheader is "<<row_header[i]<<endl;
  }
 cout<<endl;
 for(int i=0;i<num_of_rows;i++)
  {
 for(int j=0;j<num_of_columns;j++)
   {
 getline(infile,temp,delimiter);
 outfile<<column_header[j]<<delimiter<<row_header[i]<<delimiter<<temp<<endl;
   }
 getline(infile,temp);
  }
 cout<<endl<<endl;
 return 0;
}
```