# autoSQL és autoXML: a Genome Projekt kódelőállítói

Ezek az eszközök ismétlődő kódsorok tízezreinek megírásától mentettek meg minket.

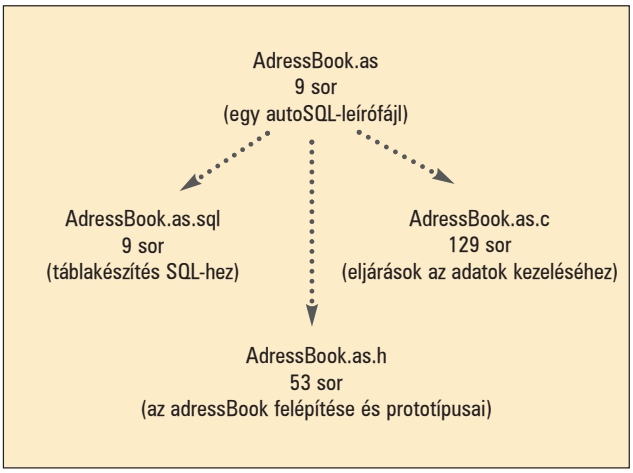
A datok mozgatása egyik forrásból a másikba általános esetben nem olyan bonyolult. Ha például van egy TABULATOR-ral elválasztott adatokat tartalmazó állományunk, amit SQL relációs adatbázisba szeretnénk helyezni, egyszerűen írunk egy rövid SQL-parancsot, majd létrehozunk például egy rövid C-programot, amely a forrásfájlból beolvassa az adatot és kiírja az adatbázisba. Csakhogy amikor nagy adatbázisokkal dolgozunk, vagy mint a mi esetünkben, rettentő nagyokkal, hirtelen azon kapjuk magunkat, hogy tucatnyi forrásból származó gigabájtos nagyságrendű adattal van dolgunk. Az efféle kódok írása többé már nem jelent kiutat.
E feladat megoldásához mutatunk be most két eszközt, amelyek megszüntetik a gondokat. Együtt adatbázis-meghatározásokat képesek készíteni SQL-hez, létrehozni C-fejléc-fájlokat, -függvényeket és -prototípusokat adatmeghatározásainkhoz, megírják helyettünk a C-szerkezetekbe író és az onnan olvasó C-kódot, illetve a XML-értelmezőhöz tartozó kódot is elkészítik.

## A Human Genome böngésző

Az emberi genom (human genome) olyan utasítás-kézikönyv, amely a DNS-ünkben kódolódik. Körülbelül hárombillió kémiai betűpárból áll, amelyeket általában kezdőbetűikkel jelölünk: G, C, A és T. A genomadat e betűk 24 hosszú szálából áll, ami nem olvasható igazán könnyen. A Human Genome Browser a Santa Cruz-i székhelyű Californiai Egyetem honlapja, amely szerte a világon számos tudósnak teszi elérhetővé ennek a hegynyi adatnak a képi megjelenítését. A böngésző az adatsort egy magasabb szintű jelölésformával ábrázolja – a genom egyes részeihez tartozó feladatoknak megfelelően. A felhasználók megkereshetik az érdekesnek talált géneket, belenagyíthatnak, csatolást hozhatnak létre a genom adott részéhez, ami az idetartozó kutatások lapjára mutat, illetve a genom adatait összehasonlíthatják más fajok adataival. A böngésző bizonyos fajta jelöléseket a genom koordinátahelyek alatt sávként jelenít meg.
A Genome Browser HTML/CGI-felülettel bír, amely lehetővé teszi, hogy a felhasználó megtekintse, illetve (a dinamikusan készített képtérkép (image map) segítségével) rákattinthasson a genom sávjaira. A sávok nagyítási beállításait, illetve sűrűségét a felhasználók űrlapmezőkön keresztül állíthatják be.
A CGI forráskódja C-ben készült, a genomadat pedig SQL-adatbázisban tárolódik.
Itt igen sok adatról van szó. A böngésző forráskódjának kezelnie kell a gén-előrejelzések (genom prediction) és az emberi, illetve más fajok genomja közti hasonlóság formátumait.
A dolgokat tovább bonyolítja, hogy legalább egy tucat különféle külső forrásból dolgozunk, amelyek mind a saját külön forrásukat használják. Még ha nem is akarjuk ezeket az adatformátumokat belsőleg használni, akkor is meg kell írnunk a hozzájuk tartozó értelmezőket, hogy beolvashassuk, és a saját formátumunkba menthessük őket. A kódelőállítókat körülbelül felerészben más csoportok állományainak beolvasását megkönnyítő feladatra használtuk.

## Bevezetés az autoSQL-be

Az autoSQL SQL- és C-kódot készít az adatok adatbázisból való kiolvasásához, illetve az oda való betöltéshez. Az autoSQL használatával nem kell a fárasztó adatmeghatározásokat megírnunk, a beolvasó és mentő kódot is beleértve. Például a böngésző közelítőleg harminc nyilvános és harminc kísérleti sávot




Egy autoSQL-leírófájl feldolgozása

tartalmaz. Minden egyes sáv a relációs adatbázis egy-egy táblájához van rendelve. Minden egyes modul, amely egy sávtáblát a memóriába tölt, autoSQL-lel készült.

## XML-kapcsolat

A projekt későbbi részeiben elkezdtünk XML-t használni, hogy a japán kutatócsoporttal együtt tudjunk működni. Az XML a DAS-rendszeren keresztül (a Distributed Annotation System a genomadatok internetes átadásának formátuma) egyúttal a más oldalakkal való adatcserére is alkalmas.
Az autoXML az XML DTD fájl alapján elkészíti az XML-értelmező számára a C-kódot. Mivel az XML-adatcsere még a SQL-adatcserénél is feldolgozásigényesebb, az autoXML igen hasznosnak bizonyult.

## Értékük a kódolás szempontjából

Az autoSQL és az autoXML együtt felbecsülhetetlen értékű időmegtakarító alkalmazásnak bizonyult. Az autoSQL a böngészőprojekt létfontosságú igáslovává vált. A maga 1200 sorával a böngészőprogram felét készítette el: kódsorok tízezreit.
Bár nem használunk annyi XML-kódot, mint SQL-t, mégis még az autoXML-lel is kezdünk beragadni. Egyetlen adatbeolvasó projektben, amely a japán Riken egérgénbejegyzéseket tartalmazza, az autoXML körülbelül 1500 sornyi kódot készített (saját maga csak 1200 soros).
Az autoSQL és autoXML futtatható állományait a ➲ http://www.soe.ucsc.edu/~kent/exe címen érhetjük el. A for-

ráskód, a Linux végrehajtható állományok, és az ebben a cikkben szereplő példák a 38. CD Magazin/autoSQL könyvtárában találhatók meg.

## autoSQL-áttekintés

Az autoSQL olyan program, amely egy adott objektummeghatározás alapján a megfelelő SQL táblalétrehozó kódot és az adatbeolvasáshoz, illetve mentéshez szükséges C-kódot önműködően képes létrehozni (a folyamat lépéseit az *ábrán* találhatjuk).
A meghatározó nyelv kicsit ugyan cikornyás, a legtöbb feladat esetében azonban bizonyítottan igen hatékony. Az autoSQL-t eredetileg relációs adatbázisokhoz fejlesztettük ki; később kiderült, hogy olyan kódot készít, amelyet más egyértelműen határozott formátumokból is be tud tölteni, feltéve, hogy szöveges állományról van szó.

## Egy egyszerű példa

Képzeljünk el egy egyszerű címtárat, ahol csak a nevet, az utcát, az irányítószámot és az országot tároljuk. Az autoSQL-meghatározás a következőképpen nézne ki:

```
table addressBook
"A simple address book"
  (
  string name;
    "Name - first, last, both, we don't care"
  lstring address;  "Street address"
  string city;  "City"
  uint zipCode; "A zip code
   ➥is always positive, so can be unsigned"
  char[2] state;
    "Just store the abbreviation for the state"
   )
```

Ha egy kicsit úgy tűnik, mintha a nyelv a C-szerkezetek és az SQL-táblameghatározások hibridje lenne, az azért van, mert amikor Jim elkészítette az autoSQL nyelvet, C-ről éppen SQL-re váltott.
Ha a címtárpéldát átfuttatjuk az autoSQL-en, a program elkészíti az SQL-táblameghatározásokat:

```
  #A simple address book
  CREATE TABLE addressBook (
  name varchar(255) not null,    # Name -
       ➥first, last, both, we don't care
  address longblob not null,     # Street
       ➥address
  city varchar(255) not null,    # City
  zipCode int unsigned not null, # A zip code
       ➥is always positive, so can be unsigned
  state char(2) not null,        # Just store
       ➥the abbreviation for the state Indices
  PRIMARY KEY(name)
   );
```

illetve a C-szerkezetmeghatározásokat:

```
struct addressBook
/* A simple address book */
    {
    struct addressBook *next;
      /* Next in singly linked list. */
    char *name;
      /* Name - first, last, both, we don·t
      ➥*care */
    char *address;     /* Street address */
    char *city;        /* City */
    unsigned zipCode; /* A zip code is always
      ➥*positive, so can be unsigned */
    char state[3];     /* Just store
      ➥*the abbreviation for the state */
    };
```

A C-ben az SQL-adatbázis egyetlen sorát általában egy időben érjük el. A sort karaktersorozatok tömbjeként kapjuk meg.
A C programtól függ, hogy ezeket az ASCII számmegfelelőket alakítja-e át bináris számokká. Ez nem túl nehéz munka, viszont miután begépeltünk húsz-harminc sort, amelyek mind nagyjából a következőképpen néznek ki:

```
point->x = atoi(row[1]);
point->y = atoi(row[2]);*
```

az autoSQL által készített két eljárást hamar megtanuljuk értékelni. Lásd még:

```
void addressBookStaticLoad(char **row,
      ➥struct addressBook *ret);
/* Load a row from addressBook table
➥into ret. */
/* The contents of ret will be replaced
➥at the */
/* next call to this function. */

struct addressBook
➥*addressBookLoad(char **row);
/* Load a addressBook from row
➥*fetched with */
/* select * from addressBook from
➥*database.*/
/* Dispose of this with addressBookFree(). */
```

Az első eljárást általában akkor használjuk, amikor egyszerre csak egyetlen elemet szeretnénk feldolgozni. Mivel dinamikusan nem foglal memóriát, igen gyors. A második eljárás ellenben az adatszerkezetet dinamikusan foglalt memóriába menti. Mivel a C-szerkezet minden esetben tartalmaz egy next (következő) mezőt, ezt az eljárást címbejegyzésliták kialakításához is könnyen felhasználhatjuk.
Az egyetlen gond a dinamikus memóriahasználattal az, hogy nem szabad megfeledkezni a felszabadításáról. Bár az autoSQL nem tud helyettünk emlékezni, képes olyan eljárásokat készíteni, amelyek egy dinamikusan foglalt adatszerkezetet vagy ezek listáját szabadítják fel. A következő két eljárás pontosan ezt teszi:

```
void addressBookFree(struct addressBook **pEl);
/* Free a single dynamically allocated
 * addressBook such as created with
 * addressBookLoad(). */

void addressBookFreeList
     ➥(struct addressBook **pList);
/* Free a list of dynamically
 * allocated addressBook·s */
```

*2. lista* Az autoSQL-írásmód

```
declarationList:
    declaration
        declarationList declaration

declaration:
    declareType declareName comment '(' fieldList
')'

declareType:
    'simple'
    'object'
    'table'

declareName:
    name

comment:
    quotedString

fieldList:
    field
    fieldList field

field:
    fieldType fieldName ';' comment
    fieldType '[' fieldSize ']'  name ';'
comment

fieldName:
    name

fieldType:
    'int'
    'uint'
    'short'
    'ushort'
    'byte'
    'ubyte'
    'float'
    'char'
    'string'
    'lstring'
    declareType declareName

fieldSize:
    number
    fieldName
```

Megjegyzések:

- name: betűk és számok sorozata, amely betűvel kezdődik
- number: számjegyek sorozata
- quotedString: bármilyen szöveg dupla idézőjelek közé zárva

Igazán kényelmes dolog, hogy kódolás nélkül tudunk adatszerkezetekbe adatbázismezőt beolvasni, de néha szerkezeteket is ki kell tudnunk írni. Az autoSQL azt feltételezi, hogy az adatszerekezeteket TABULATOR-ral vagy vesszővel határolt formátumban mentjük. Olyan eljárást készít, amely bármelyik feladatot el tudja végezni:

```
void addressBookOutput(struct addressBook *el,
    ➥FILE *f, char sep, char lastSep);
/* Print out addressBook. Separate fields
➥*with sep. Follow last field with lastSep. */
```

illetve makrókat, amelyek a vesszős és tabulátoros változatokhoz kényelmes elérést adnak:

```
#define addressBookTabOut(el,f)
 ➥addressBookOutput(el,f,'\t','\\n');
/* Print out addressBook as a line in
➥*a tab-separated file. */

#define addressBookCommaOut(el,f)
 ➥addressBookOutput(el,f,',',',');
/* Print out addressBook as a comma
➥*separated list including final comma. */
```

Az autoSQL egy olyan eljárást is készít, amely vesszővel határolt listákat olvas. Mivel ezt az eljárást valószínűleg soha nem fogjuk közvetlenül meghívni, a mezők kicsit összetettebbek, mint azok az egyszerű karaktersorozatok és számok, amelyeket az adatbázisba mentünk. Ez az eljárás lehetővé teszi, hogy az autoSQL olyan objektumokat kezeljen, amelyek további objektumokat tartalmazhatnak.

## Az objektumok típusai

Az autoSQL három objektumtípust ismer:

- ***Simple***: olyan objektum, amely nem tartalmaz változó méretű tömböt.
- ***Object***: olyan objektum, amely tartalmazhat változó méretű tömböket. A következő mutató önműködően beszúródik az objektumnak megfelelő C-szerkezet első mezőjébe.
- ***Table***: hasonló az objektumhoz, azonban a program C-meghatározásokon kívül SQL-meghatározásokat is készít.

A ***Simple*** objektum a program tömbmegadás-kezelésében különbözik a többitől. A következő mezőmegadás esetében

```
simple point[3] triangle; "A three sided figure"
```

a három pont a memóriában C tömbként tárolódik. Ellenben ha a szerkezetet a

```
object point[3] triangle; "A three sided figure"
```

módon adjuk meg, a három pont a memóriában egyenként csatolt listaként jelenik meg.

### A mezők típusai

A következő alap mezőtípusokat támogatja:

- `int`: 32-bites előjeles egész;
- `uint`: 32-bites nem negatív egész;
- `short`: 16-bites előjeles egész;
- `ushort`: 16-bites nem negatív egész;
- `byte`: 8-bites nem negatív egész;
- `ubyte`: 8-bites nem negatív egész;
- `float`: egyszeres pontosságú IEEE lebegőpontos szám;
- `char`: 8-bites karakter (csak tömbben használható);
- `string`: változó hosszúságú karaktersorozat, legnagyobb hossza 255 bájt;

*3. lista* autoXML-kód előállítása

```
A k vetkezı tartalmœ polygon.dtdx fÆjl
<!ELEMENT POLYGON (DESCRIPTION? POINT+)>
<!ATTLIST POLYGON id CDATA #REQUIRED>
<!ELEMENT DESCRIPTION (#PCDATA)>
<!ELEMENT POINT>
<!ATTLIST POINT x INT #REQUIRED>
<!ATTLIST POINT y INT #REQUIRED>
<!ATTLIST POINT z INT "0">


Øs az alÆbbi parancssor

**autoXml polygon.dtdx poly*

a k vetkezı poly.h fÆjlt eredmØnyezi:

/* poly.h autoXml generated file */
#ifndef POLY_H
#define POLY_H

struct polyPolygon
    {
    struct polyPolygon *next;
    char *id;    /* Required */
    struct polyDescription *polyDescription;
        /** Optional (may be NULL). **/
    struct polyPoint *polyPoint;
        /** Non-empty list required. **/
    };

void polyPolygonSave(struct polyPolygon *obj,
    ➥int indent, FILE *f);
/* Save polyPolygon to file. */

struct polyPolygon *polyPolygonLoad(char
    ➥*fileName);
/* Load polyPolygon from file. */

struct polyDescription
    {
    struct polyDescription *next;
    char *text;
    };

void polyDescriptionSave(struct
    ➥polyDescription *obj, int indent, FILE *f);
/* Save polyDescription to file. */

struct polyDescription
     *polyDescriptionLoad(char *fileName);
/* Load polyDescription from file. */

struct polyPoint
    {
    struct polyPoint *next;
    double x;       /* Required */
    double y;       /* Required */
    double z;       /* Defaults to 0 */
    };

void polyPointSave(struct polyPoint *obj,
    ➥int indent, FILE *f);
/* Save polyPoint to file. */

struct polyPoint *polyPointLoad(char *fileName);
/* Load polyPoint from file. */

#endif /* POLY_H */
```

- `lstring`: változó hosszúságú karaktersorozat, legnagyobb hossza kétbillió bájt.

A ***Simple***, ***Object*** és ***Table*** típusok is használhatók mezőként.

## Állandó hosszúságú és változó tömbmegadások

Egy tömböt állandó vagy változó hosszúságúnak lehet megadni. A változó hosszúságú tömböket úgy adjuk meg, hogy tömb megadáskor a mező nevét zárójelek közé tesszük. Ennek a mezőnek a tömb előtt kell meghatározva lennie.

## Egy összetettebb példa

Képzeljük el, hogy éppen most készültünk el egy csodálatos 3D-s modellező programmal. Az egyetlen gond, hogy az adatokat most adatbázisba kellene menteni. Az *1. lista* mutatja meg, hogyan építhetjük fel ezt az adatbázist autoSQL-lel. A meghatározást *threeD.as*-ként mentve és futtatva az

```
autoSql threeD.as threeD
```

utasítást máris megkaptuk: 393 sornyi – remélem – hibamentes C-kódot és 14 sor SQL-t cserébe a mindössze 33 soros meghatározásért (a teljes autoSQL nyelv megismeréséhez nézzük át a *2. listát*!).

## autoXML-ismertető

Az autoXML XML DTD fájl alapján hoz létre C-kódot az XML-értelmezőhöz. A DTD minden egyes mezőjéhez egy-egy C-szerkezetet állít elő, és az eredeti szerkezet minden egyes tulajdonságához egy-egy mezőt készít. Alapértelmezés szerint olyan értelmezőt állít elő, amely figyelmen kívül hagyja az olyan elemeket, amelyek nincsenek megadva a DTD-ben, de egyébként hitelesítő értelmező is. Ha a `-picky` kapcsolót használjuk, teljes mértékben hitelesítővé (validating) válik.
Az autoXML-értelmező a teljes fájlt a memóriába tölti. Ha ez nehézséget okoz, az alacsonyabb szintű xap értelmezőt kell igénybe vennünk, amely nagyon hasonlít a gyakran használt expat értelmezőhöz, de egy kicsit gyorsabb.

## Gyors XML- és DTD-ismertető

Ha az eddig olvasott rövidítések valakit zavarba ejtettek, akkor valószínűleg nemigen hallott még az XML-ről (eXtensible Markup Language). E nyelvnek tagalapú formátuma van.

Egyszerű példa egy ilyen XML-dokumentumra:

```
<POLYGON id="square">
   <DESCRIPTION>
      This is soooo square man
   </DESCRIPTION>
   <POINT x="0" y="0" />
   <POINT x="0" y="1" />
   <POINT x="1" y="1" />
   <POINT x="1" y="0" />
</POLYGON>
```

XML-ben minden <TAG></TAG> párok közt van. A tagokhoz szöveg, tulajdonság és altag lehet hozzárendelve. A fenti példában a POLYGON két altagja a DESCRIPTION és a POINT , tulajdonsága az id, szövege pedig nincs. A DESCRIPTION tagnak van szövege („This is soooo square man”), viszont nincs altagja és tulajdonsága. A POINT az x és y tulajdonságokkal bír. A POINT egyben egy XML-rövidítést is bemutat: a kizárólag tulajdonságokat tartalmazó tagokat <TAG att="valami" /> formában is lehet írni, a <TAG att="valami"></TAG> rövidítéseként.
Az XML sokban hasonlít a HTML-re, de jelentős különbségek is akadnak. XML alatt minden egyes tulajdonságnak idézőjelek közt kell szerepelnie, míg HTML-ben ezek elhagyhatók. A tagoknak XML alatt szigorúan keretet kell alkotniuk, míg a HTML megengedi, hogy a tagokat megnyissuk és ne zárjuk le. HTML-ben a tagok előre megadottak, XML alatt a meghatározásaik tőlünk függenek.
XML alatt két módon adhatunk meg tagokat: DTD fájllal avagy XML-sémák alapján. Mindkét módszer mellett és ellen is szólnak érvek. A DTD fájlok viszonylag egyszerűek és igen sok értelmező és XML-böngésző felismeri őket. Másrészről, a DTD fájlok nem tudják leírni, hogy egy bizonyos tulajdonság számformátumú-e. Az XML-sémák összetettebbek: maguk is XML formátumban íródtak, ami bizonyos szempontból nem rossz. Egyelőre azonban nem olyan széles körben támogatottak. Jelenleg az autoXML is csak DTD fájlokkal működik, néhány egyszerű kiegészítéssel.
Íme, a fenti POLYGON formátumot leíró DTD:

```
<!ELEMENT POLYGON (DESCRIPTION? POINT+)>
<!ATTLIST POLYGON id CDATA #REQUIRED>

<!ELEMENT DESCRIPTION (#PCDATA)>

<!ELEMENT POINT>
<!ATTLIST POINT x CDATA #REQUIRED>
<!ATTLIST POINT y CDATA #REQUIRED>
<!ATTLIST POINT z CDATA "0">
```

A DTD két f meghatározástípussal rendelkezik: az ELEMENT és ATTLIST (vagy attributes) kulcsszóval. Az element meghatározás tartalmazza az elem nevét és az alelemek zárójeles listáját, ez utóbbi elhagyható. Az alelemeknek valahol meg kell lenniük adva a DTD-ben, kivételt képez a #PCDATA alelem, amely azt jelzi, hogy az elem a tagjai között szöveget tartalmazhat. Minden alelem megadását az alábbi karakterek egyike követheti:

- ?: az alelem elhagyható.
- +: az alelem legalább egyszer előfordul.
- *: az alelem 0 vagy több esetben ismétlődik.

Ha az alelemet semmilyen karakter nem követi, akkor az alelem pontosan egyszer szerepel. Az ATTLIST tulajdonságokat ad meg és hozzárendeli őket valamelyik elemhez. Jó szokás, ha az ATTLIST -eket a megfelelő ELEMENT mellett tartjuk. Az ATTLIST a következő mezőket tartalmazhatja:

- elem: a vonatkozó elem neve
- név: a tulajdonság neve
- típus: általában CDATA . Lehet reference (hivatkozás) vagy date (dátum), viszont az autoXML nem kezeli őket.
- alapértelmezés: itt találjuk a felhasználandó alapértelmezett értéket arra az esetre nézvést, ha a tulajdonság nem lenne jelen. A #REQUIRED kulcsszó ebben a mezőben azt jelenti, hogy a tulajdonság megadása kötelező. A #IMPLIED kulcsszó jelentése: az adott tulajdonság hiányozhat (ebben az esetben – miután az autoXML beolvasta – az értéke NULL vagy zérus).

## Az autoXML kiegészítései és korlátai

Az autoXML az ATTLIST kulcsszót az INT vagy FLOAT típusokkal bővíti ki, hogy karakteres érték helyett számtípus is megadható legyen. Ennek megfelelően a #PCDATA helyén a #INT vagy a #FLOAT jelöléseket is használhatjuk, ha a szövegmezőbe számszerű adatot szeretnénk helyezni. Ha ezeket a kiterjesztéseket használjuk, DTD fájlunkon inkább a ***.dtdx*** kiterjesztést alkalmazzuk az egyszerű ***.dtd*** utótag helyett.
Jelenleg az autoXML csak akkor kezeli a DTD-megjegyzéseket, ha egymagukban állnak az adott sorban. Az autoXML elvárja, hogy minden ELEMENTS és ATTLIST meghatározás egyetlen sorban férjen el. Továbbá nem kezeli a hivatkozó adattípust azon túl, hogy a hivatkozási azonosítót szövegként menti.
A *3. listában* egy teljes, autoXML által készített forráskódpéldát figyelhetünk meg. A 3. listában látható ***.h*** fájlon kívül az autoXML a megfelelő ***.c*** fájlt is elkészíti. Minden XML-fájlnak egy fő objektummal kell bírnia. Jelen esetben ez a fő objektum a POLYGON (a DTD jelen formájában nem teszi lehetővé, hogy egy fájlban több mint egy sokszögünk legyen). A fenti DTD-t figyelembe vevő XML-fájlokat a polyPolygonLoad() függvénnyel tölthetjük be, majd a polyPolygonSave függvénnyel menthetjük vissza.
Mint láthattuk, az autoSQL és autoXML jól működik az adatok széles skáláján, legyen az címtár vagy akár génkövetés. Reméljük, ezeket az eszközöket a saját projektjeidben is hasznosnak fogod találni.

***A kapcsolódó címek, valamint az 1. lista megtalálható a 38. CD Magazin/autoSQL könyvtárában.***

*Linux Journal 2002. július, 99. szám*

*Jim Kent*
PhD-fokozattal bíró kutató, a Human Genome Projectben végzett munkájáról a New York Times, a San Francisco Chronicle és más médiumok közöltek cikkeket. Jelenleg a fajok közti genom-összehasonlításon dolgozik.

*Heidi Brumbaugh*
a késő ’80-as évek óta végez írói tevékenységet a számítógépes kiadás iparágában.
A projektjeihez vezető hivatkozások, illetve írásai a ➲ http://www.heidi.to címen érhetők el.