

# ***CSeRe-BeRe Börze***

Rendszerdokumentáció

*Készítette:* Lückl Roland  
*Szak:* Mérnök Informatikus (BSc)  
*Neptun:* DBRZR5

## Bevezető

Ebben a dokumentumban le fogom írni (nagyvonalakban) hogy hogyan is működik a „CSeRe-BeRe Börze” (a továbbiakban: CSBB). Elsősorban az kerül majd megvitatásra, hogy mi is történik, ha valaki megnyom egy gombot a börze kezelőfelületén, vagy kiválaszt egy menüpontot, stb. A kód (túl) részletes megvitatásába nem megyek bele, ha valaki nem ért egy parancsot vagy hogy mit hogyan használtam, az keressen fel személyesen vagy MSN-en (elérhetőség ld. a dokumentáció végén). Elsőként az adatbázis felépítését fogom részletezni, majd jön maga a börze és annak a kódja.

## 1. Az adatbázis felépítése

Az adatbázist MySQL-ben hoztam létre. Azért erre a rendszerre esett a választásom, mert már korábbról ismertem (*Projekt Labor*) és mert szerintem végtelen egyszerű kezelni. Az adatbázis 4 táblából áll, melyek a következők:

- *cards*: ez tárolja az egyes kártyákat
- *numbers*: ez a tábla tárolja hogy melyik felhasználónak hány darab kártyája van az egyes színekből
- *objects*: itt vannak eltárolva az adatbázisba felvett tárgyak
- *users*: ebben pedig a felhasználók adatai vannak tárolva

Az egyes táblázatok belső felépítése pedig az alábbiak szerint látható:

### a) cards

- *c\_cardid* - **INT(5)**: a kártya azonosítója, az adatbázis generálja, minden kártyának egyedi azonosítója van, a többi táblában ez alapján lehet keresni. 5 bájttal hosszúságú integer típus.
- *c\_color* - **VARCHAR(255)**: a kártya színe, hogy a felhasználók könnyebben tudják azonosítani a kártyáikat. Max. 255 karakter hosszúságú sztringet lehet megadni.
- *c\_minvalue* - **INT(11)**: a kártya minimum értéke, azaz alsó értékhatára. Egy 11 bájttal hosszú integer, azaz nagyon nagy értékű kártyákat is létre lehet hozni ☺.
- *c\_maxvalue* - **INT(11)**: a kártya maximum értéke, szintén 11 bájttal egész.

### b) numbers

- *n\_userid* - **INT(5)**: a felhasználó egyedi azonosítója, kié a kártya (= *u\_userid*). Maximum 5 bájttal hosszúságú egész, ugyanúgy ahogy az összes többi azonosító.
- *n\_cardid* - **INT(5)**: a kártya egyedi azonosítója, azaz hogy melyik kártyáról van szó.
- *n\_howmany* - **INT(11)**: egy 11 bájttal nagyságú integer, mely megmondja, hogy az *n\_userid* (= *u\_userid*) azonosítójú felhasználónak az *n\_cardid* (= *c\_cardid*) azonosítójú kártyából hány darabja van.

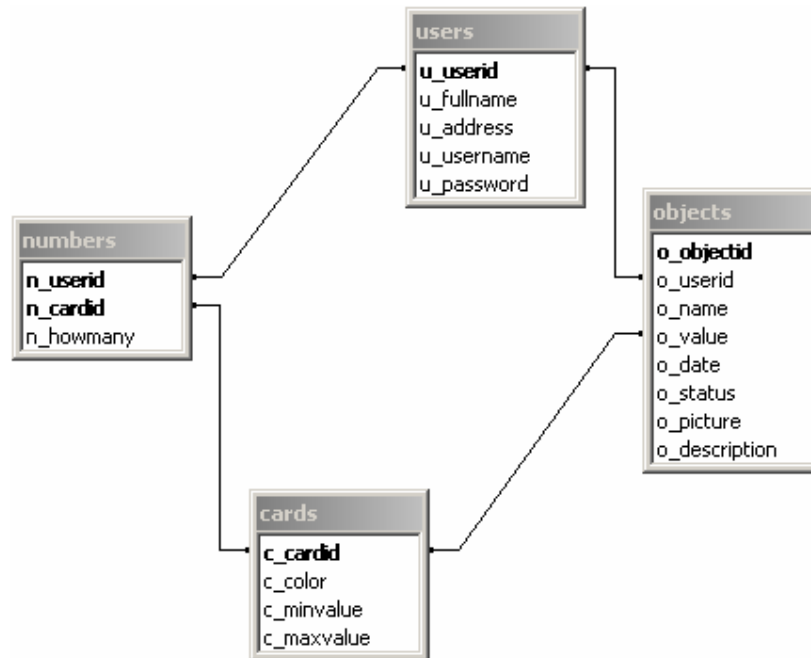
### c) objects

- *o\_objectid* - **INT(5)**: a tárgyak egyedi azonosítója, 5 bájt méretű. Az adatbázis automatikusan generálja, amikor egy tárgy fel lesz véve. Később ez alapján lehet azonosítani a tárgyakat.
- *o\_userid* - **INT(5)**: annak a felhasználónak az egyedi azonosítója, aki a tárgyat felvette a rendszerbe. Vagy ha a tárgy már el van adva, akkor annak a felhasználónak az azonosítója, aki a tárgyat megvette.
- *o\_name* - **VARCHAR(255)**: a tárgy megnevezése, maximum 255 karakter hosszan.
- *o\_value* - **VARCHAR(255)**: ez a mező két funkciót tölt be egyszerre: amíg a tárgy „*todo*”, azaz függő státuszban van, addig a felhasználók által „*Szerintem ennyit ér*” mező tartalma van benne, miután az admin megbecsülte a tárgyat annak a kártyának az egyedi azonosítója szerepel itt, amelyik asztalra került.
- *o\_date* - **DATETIME**: a tárgy adatbázisba való felvételének a dátuma másodperc pontossággal. A PHP *date()* függvénye generálja automatikusan, amikor a felhasználó rányom a „*Hozzáad*” gombra.
- *o\_status* - **VARCHAR(255)**: a tárgy állapota. 3 értéket vehet fel, melyek a következők:
  - todo: a felvételtől a becslésig tartó állapot. Az ilyen állapotú tárgyakat csak a becsüs láthatja, a „*Függőben lévő tárgyak becslése*” panelen.
  - ok: a már megbecsült, de még nem eladott tárgyak, tehát azok, amelyek az egyes asztalokon vannak. Ezeket minden felhasználó láthatja a megfelelő asztal kiválasztásával vagy a keresés eredményeként.
  - sold: a már eladott tárgyak lesznek így jelölve. Csak az a felhasználó láthat ilyen jelöléssel ellátott tárgyat, aki a kérdéses tárgyat saját maga vette meg.
- *o\_picture* - **LONGBLOB**: ha a tárgyhoz lett kép feltöltve, akkor az ebben a mezőben lesz eltárolva mint bináris adat. Maximum 2MB méretű fájlokat lehet feltölteni.
- *o\_description* - **TEXT**: a tárgy leírása, ha felvételkor lett megadva. TEXT típusú, tehát elvileg bármilyen hosszúságú lehet.

### d) users

- *u\_userid* - **INT(5)**: a felhasználó egyedi azonosítója, a MySQL generálja automatikusan minden felhasználónak regisztrációkor.
- *u\_fullname* - **VARCHAR(255)**: a felhasználó teljes neve, maximum 255 karakter.
- *u\_address* - **VARCHAR(255)**: a felhasználó lakcíme, szintén max. 255 karakter.
- *u\_username* - **VARCHAR(20)**: a felhasználó loginneve, amit a regisztrációkor választott. UNIQUE kulcsú, tehát minden felhasználónévnek egyedinek kell lennie. Maximum 20 karakter lehet.
- *u\_password* - **VARCHAR(20)**: a felhasználó regisztrációkor választott jelszava, MD5-ös kódolással titkosítva. Szintén maximum 20 karakter lehet.

Az alábbi képen látható az adatbázis logikai modellje:



## 2. A rendszer felépítése

Mielőtt elkezdem a rendszer részletes elemzését elsőként szeretnék egy pár dolgot megjegyezni. Ezek általában igazak a teljes rendszerre és sokszor előfordulnak, ezért nem fogom őket minden egyes alkalommal újra leírni. Tehát a következőkről van szó:

- a felhasználók a **\$user** változón keresztül lesznek azonosítva, ami bejelentkezéskor lesz beállítva a felhasználó **u\_userid** azonosítójára. Ez a változó minden **form**-ban rejtett attribútumként utazik. Sajnos ez nem egy bombabiztos módszer, de ezt tudtam a legegyszerűbben megvalósítani. Ha valaki a **user.php**-t anélkül próbálná megnyitni hogy nem adja át a „User ID”-t a rendszer nem engedi használni az egyes funkciókat. Ugyanez igaz az **admin.php**-re is.
- a form-okban a „User ID”-n kívül még egy adat „utazik” rejtetten minden egyes alkalommal, méghozzá a **\$doWhat**. Ez a változó jelöli, hogy épp melyik panel volt nyitva és azért kell, hogy a **form** elküldése után is nyitva maradjon az a panel, és persze a hozzá tartozó PHP kódrészlet lefusson.
- az előzőhöz kapcsolódik a **\$doIt** változó is, ami szintén rejtettként utazik általában. Ez a változó azért kellett, mert a fájl megnyitásakor automatikusan lefut minden rész ami a fájlban van. És pl. ha megnyitjuk az „Új tárgy hozzáadása” panelt, akkor a hozzáadás résznek mindaddig nem kell lefutnia, amíg a felhasználó ténylegesen meg nem nyomta a „Hozzáad” gombot. Ez igaz az összes többi panelre is.
- a rendszerben minden paraméter **POST** formában lesz átadva, azaz nem látszik a böngésző címsávjában, hogy épp mit küld. Erre részben azért volt szükség, mert a képmegjelenítés csak így működik, részben pedig azért, mert a felhasználók számára felesleges információkat nem kell megjeleníteni.

A következőkben a rendszerhez tartozó fájlokat külön pontokba szedve ismertetni fogom.

#### a) connect.php

Ez a fájl, mint a neve is mutatja, csatlakozik. Méghozzá a szerverhez való csatlakozást és az adatbázis kiválasztását végzi el. Más egyéb funkciók nincsenek benne. Ez a fájl az összes többi fájl legelején be van „include”-olva így később nem kell foglalkozni az adatbázishoz való csatlakozással.

#### b) index.php

A *CSBB* nyitólapja, nem tartalmaz mást, mint a bejelentkezéshez szükséges mezőket, úgymint felhasználónév és jelszó. Ez egy **form**-ban foglal helyet, majd ha a felhasználó megnyomta a „Bejelentkezés” gombot továbbirányítja a **login.php**-re és a megfelelő paramétereket átadva az elvégezheti az ellenőrzést, stb...

Egy másik **form**-ban, a bejelentkező doboz alatt található a regisztrációhoz vezető gomb, mely azért kellett hogy **form**-al legyen megvalósítva, hogy a **register.php**-ben a mezőket alapértelmezésben kinullázva találjuk és a *PHP* ne jelezzen hibát nem inicializált változók miatt. Később rájöttem, hogy ezt is meg lehetett volna csinálni egy **\$doIt** rejtett paraméter segítségével, de már nem javítottam át, így változatosabb lett a megvalósított kód. A többi fájlban már csak a **\$doIt**-es megoldást használtam.

#### c) register.php

Ez a fájl lényegében egyetlen **form**-ból áll, melyben a regisztrációhoz szükséges adatokat lehet megadni. Majd a „**Regisztrálok**” gomb segítségével egy gyors hibaellenőrzésen átfuttatva (mindenhol minimum egy karakternek meg kell lennie adva, a maximum értékeket pedig az adatbázis leírásából lehet kiolvasni (ld. 1. pont) és a két megadott jelszónak egyeznie kell) a *PHP* elküldi az adatbázisnak egy **INSERT** segítségével az új felhasználó adatait.

Innentől kezdve él a regisztráció és be lehet lépni. A sikeres regisztrációt egy kis üzenettel jelzi a rendszer, majd 5 másodperc elteltével automatikusan átirányítja a nyitólapra.

A *MySQL*-től visszajövő adatokon csak egyetlen hibaellenőrzés lesz végrehajtva (**if ... mysql\_erro() ... =1062**) mely azt nézi meg, hogy létezik-e már olyan loginnevű felhasználó a rendszerben. Ha igen, akkor ezt egy hibaüzenettel jelzi a felhasználó felé. Ezt a tényt a *MySQL* az **1062**-es hibakóddal jelzi (*DUPLICATE UNIQUE KEY*), ezért lesz az vizsgálva, hogy **1062**-es hibát kaptunk-e.

#### d) login.php

Ez a fájl nézi, meg ha valaki be szeretne jelentkezni a *CSBB*-be, hogy létezik-e az adott felhasználónév a rendszerben és stimmel-e a hozzá megadott jelszó. Ezt az ellenőrzést végzi el a fájl elején a lekérdezés, mely a szükséges adatokat **POST** paraméterként kapja az **index.php**-től. Ha az adatbázisból üres eredmény jött vissza, azaz nem létezik olyan felhasználó, akkor az első feltétel nem teljesül és hibaüzenet jelenik meg.

Ha kaptunk vissza „használható” adatot, akkor a feltétel „else” ága fog lefutni, melyben egy újabb feltétel található. Ez azt figyel, hogy ki szeretne belépni a rendszerbe. Ha az 1. számú felhasználó – aki az **admin**, **u\_serid = 1** – akkor teljesül a feltétel és egy kicsit más üdvözlő-üzenet fogadja, mint sima felhasználó esetén, melyet a feltétel nem teljesülése esetén jelenít meg a rendszer.

## e) admin.php

Ez a fájl tartalmazza a teljes „*Admin Kezelőfelületet*”. Itt végezheti el a rendszergazda a „rendszergazdai” funkciókat, mint pl. a függőben lévő tárgyak becslése vagy új kártya létrehozása. A fájl felépítése viszonylag egyszerű: minden megjelenítendő szöveg, táblázat, stb. egyetlen nagy feltételben foglal helyet. Ez a feltétel vizsgálja, hogy be van-e jelentkezve valaki. Ha igen, akkor láthatjuk a felső menüt és az abból megnyitott paneleket. Ha nincsen bejelentkezve senki, de valaki mégis el akarja érni az **admin.php**-t (közvetlen hivatkozik rá), akkor nem teljesül ez a szinte egész fájlt magába foglaló feltétel és egy hibaüzenet kerül csak megjelenítésre (a fájl legvégén található az erre szolgáló táblázat).

A „nagy” külső feltételen belül a fájl további két részre bontható: a felső menüt megjelenítő **form**-ra és minden egyébre. A minden egyé alatt egy többes feltételt kell érteni, melynek minden egyes ága egy-egy panel megjelenítésével és az ahhoz tartozó funkciók megvalósításával foglalkozik. A felső menüt egy **form**-ban található **select**-ben lévő **option**-ök jelentik. Minden ilyen opció egy funkciót jelképez. Ha valaki kiválaszt egyet, akkor a **\$doWhat** változó mondja meg, hogy melyik funkcióról van szó és az alatta lévő nagy feltételben melyik ág fusson a továbbiakban. Pl. ha meg szeretnénk tekinteni a jelenlegi kártyákat a rendszerben, a **\$doWhat** értéke 2-re állítódik és innentől kezdve csak az „**else if (\$doWhat==2) { ... }**” részben levő sorok futnak le. Természetesen mindaddig, amíg valaki valami mást nem választ a menüben és a „*Meher*” gombbal másra nem állítja a **\$doWhat** értékét.

Most vegyük egyesével a menüpontokat, azaz a többes feltétel egyes részeit:

*\$doWhat = 1*

Ez a rész foglalkozik a függőben lévő tárgyak megjelenítéséről, és ha az **admin** megbecsülte őket, akkor a megfelelő változtatások elvégzéséről az adatbázisban. Az elején kapásból egy bonyolult kódrészlet következik az „**if (\$doWhat==1) { ... }**”-ben. Ezt a részt később tárgyaljuk, most ugorjuk át.

Elsőként egy lekérdezés megkeresi az adatbázisban az összes olyan tárgyat, amely állapota „*todo*”, azaz függőben van. Majd ezeket a tárgyakat egy táblázatba rendezi a „**while (\$row ...) { ... }**” cikluson belül. Itt talán csak annyit érdemes megemlíteni, hogy a táblázat generálása közben az egyes *checkbox*-ok, legördülő menük (ahol a színt lehet kiválasztani) értékei és a tárgyak azonosítói egy-egy tömbbe kerülnek. Így később könnyebb lesz velük dolgozni. Ez a táblázat egy **form**-ban foglal helyet, melynek a végét a ciklus utáni „**echo**” parancs „*zárja le*”.

Ha valaki megnyomja a „*Függőben lévő tárgyak becslése*” gombot az „**if (\$doWhat==1)**” feltétel elején lévő bonyolult rész fut le, amelyet az előbb átugrottunk. Ez a rész elsőként leellenőrzi, hogy le kell-e futnia („**if (\$doIt==1)**”), ha igen, akkor további ellenőrzések következnek egymásba ágyazva. Az első feltétel megnézi, hogy a tárgyak azonosítóit tartalmazó tömb („**if (isset(\$\_POST['objectid']) && is\_array(\$\_POST['objectid']))**”) létezik-e. Ha igen, akkor szétbontja kulcs-érték párokra egy „**foreach**” segítségével, majd egy újabb „**foreach**” ugyanígy szétbontja a kártya-azonosítókat. Ha az előző kettő kulcsa megegyezik (tehát be volt jelölve a tárgy és ugyanabban a sorban van legördülő menü is), akkor következik a felhasználó-azonosítók szétbontása. Ezután egy újabb feltétel leellenőrzi, hogy a felhasználó-azonosító kulcsa és a *checkbox* kulcsa megegyezik-e.

Ezzel a 3 egymásba ágyazott **if**-el és 3 **foreach**-el lényegében összerendeztük a bejelölt tárgyakat a hozzájuk kiválasztott színnel és a felhasználóval aki a tárgyat felvette. Most

következik az értékadás. A **\$setValue** beállítja a tárgy értékét a legördülő menüből kiválasztott színre, a **\$setStatus** pedig függőről készre állítja a tárgyat. Ezzel a tárgy kész is.

Ekkor még vissza van a felhasználónak a megfelelő számú kártyák jóváírása. Itt elsőként egy feltétel megnézi, hogy van-e már az adott színű kártyából a felhasználónak. Ha van, akkor egyszerűen hozzáad egyet, de ha még nincsen akkor egy új sort szűr be „**numbers**” táblába. Ha mindhárom lekérdezés sikeresen lefutott, akkor ezt egy üzenet nyugtázza („**if (\$resValue && \$resStatus && \$resCards)**”). Ezzel a tárgyak becslése részen végig is értünk.

*\$doWhat = 2*

A 2. rész foglalkozik az új kártya felviteléről a rendszerbe. Egy **form** gondoskodik arról, hogy a megfelelő adatokat be lehessen vinni a rendszerbe. Ha megnyomtuk az „*Új kártya felvétele*” gombot, a **\$addCardInput** értéke 1 lesz és a **form** után következő rész lefuthat. Ez megpróbálja a kapott adatokat berakni az adatbázisba egy **INSERT** segítségével.

Ha hibát kapunk vissza, megnézi, hogy az **1062**-es hibaszámmal van-e dolgunk. Ha igen, akkor ez azt jelenti, hogy egy **UNIQUE** kulcsú mezőben 2 azonos bejegyzés akart létrejönni, aminek nem szabad megtörténnie. Magyarra fordítva ez annyit jelent, hogy olyan kártyát akartunk létrehozni, amelynek a színe megegyezik egy már létező kártyával. Ekkor kapunk egy hibaiüzenetet, hogy ilyen színű kártya már létezik. Ha sikeres volt a felvétel, akkor szintén egy üzenet nyugtázza.

*\$doWhat = 3*

A 3. rész valójában a második a kezelőfelületen, de mivel később lett megvalósítva ezért lett harmadik. Nem sok mindent csinál, csak annyit, hogy megjeleníti a rendszerbe felvitt kártyák összes tulajdonságát. Ehhez egy **SELECT**, majd a kapott adatokat megjelenítő **while** ciklus bőven elég.

*\$doWhat = 4*

Ez a rész foglalkozik a statisztikák megjelenítésével. Lényegében csak pár egymás után rakott lekérdezés és azok eredményeit egy táblázatba beíró ciklusokról van szó. A lekérdezések sorban: az összes kártya szám, a kártyák színekre bontva, az összes felvett tárgy száma, a felvett tárgyak asztalonkénti bontása, a felvett tárgyak állapotonkénti bontása.

*\$doWhat = ''*

Ha a **\$doWhat** értéke nincs beállítva (azaz = 0, amit a fájl legelején automatikusan állítunk), akkor a nagy elágazásunk utolsó „**else**” része fut le, mely csak egy rövidke üzenetet ír ki a felhasználónak, hogy válasszon valamit a felső menüből.

És végül de nem utolsó sorban bezárásra kerül a „külső” feltételünk is, mely azt ellenőrizte, hogy be vagyunk-e jelentkezve. Ha nem, akkor itt a fájl végén található rövid kis részlet fut le, mely egy hibaiüzenetet ír csak ki.

## f) user.php

A **user.php** mint a neve is mondja a felhasználókkal foglalkozik. Pontosabban a teljes felhasználói kezelőfelület ebben az egy fájlban foglal helyet. A fájl felépítése kísértetiesen hasonlít az **admin.php**-hoz, ezért az azonos részeket nem is írom le, csak a **\$doWhat** által jelzett szakaszokat. Az azon kívüli részek szinte szó szerint ugyanazok.

*\$doWhat = 1*

E rész segítségével lehet új tárgyakat felvenni a rendszerbe. Elsőként egy **form** kerül megjelenítésre, majd ha abban valaki megnyomja a „*Hozzáad*” gombot a **\$doIt** 1-re állítódik és a **PHP**-s rész is lefuthat, mely hozzáadja az adatbázishoz a **form**-ba beírt adatokat. Ha sikeres volt az **INSERT** akkor egy üzenettel nyugtázza. Egyetlen feltétel-vizsgálat van csak, méghozzá az, hogy legyen a felvett tárgynak neve („**if (strlen(\$objName)==0)**”).

*\$doWhat = 2*

Ez a rész foglalkozik az egyes asztalok megjelenítésével és azzal, hogy ha valaki egy tárgy előtt bekapcsolja a *checkbox*-ot és rányom a „*Megveszem*” gombra, akkor az adott tárgy lekerüljön az asztalról és a felhasználó megvásárolt tárgyai közé kerüljön. És természetesen, hogy a megfelelő számú jegy levonásra kerüljön

Elsőként pár változó be lesz állítva, majd ezután következik egy többszörösen egymásba ágyazott **if-foreach** rész, mely hasonlóan az **admin.php**-ben látott megoldáson alapul. Annyi különbséggel, hogy itt nem adunk jegyeket, hanem veszünk el. Természetesen jegyet csak akkor tudunk elvenni egy felhasználótól, ha van neki, ezt vizsgálja az „**if (\$userCards['n\_howmany']>0)**”. Ha a felhasználónak van megfelelő színű jegye, akkor az 1. **UPDATE**-el elveszünk tőle egy darabot, a 2. **UPDATE**-el pedig átállítjuk a tárgy állapotát „*sold*”-ra (tehát eladva).

A fenti rész csak akkor fut le ha a **\$doIt** 1-re van állítva, de valójában most következik csak a megjelenítés: elsőként végzünk egy lekérdezést, hogy a felhasználónak melyik színű kártyából hány darab van és ezt egy táblázatban megjelenítjük. Majd egy újabb táblázat kerül kirajzolásra, melyben a felhasználó ki tudja választani, hogy melyik asztal tartalmát szeretné látni. Ha a „*Megnézem*” gombra kattintottunk egy újabb **\$doIt** állítódik 1-re és megkezdődhet az asztalok tartalmának megjelenítéséért felelős **PHP** kód futása. A **\$selectColor**-al kiválasztjuk az összes olyan színű tárgyat, amelyik asztalt a felhasználó kiválasztotta. Majd egy **while** ciklussal kiírjuk az asztal tartalmát egy táblázatba.

A táblázat végé található „*A kijelöltek megvétele*” gomb, mellyel átugrunk a **\$doWhat = 2** rész elejére és elvégezzük a „*vásárlást*” a fent leírtak szerint.

*\$doWhat = 3*

A keresés annyiban tér el az asztal-kiválasztós módszertől, hogy a szín kiválasztása helyett a felhasználó megadhat egy keresési sztringet és azt, hogy a tárgyak neveiben vagy leírásában történjen a keresés. Mivel a kód 95%-a ugyanaz mint a **\$doWhat = 2** esetén ezért nem részletezném különösebben. A lényegi változás csak 1 **MySQL** lekérdezésben történt (amelyikben megkeressük a kérdéses tárgyakat). Minden egyéb hasonlóan működik a korábban leírtakkal.

*\$doWhat = 4*

E részben nézhetjük meg, hogy milyen jegyeink vannak. A kód felépítése nagyon egyszerű: elsőként egy **SELECT**-el kiválasztjuk a felhasználóhoz tartozó kártyákat, majd egy **while** cikluson belül szépen táblázatba rendezve kiírjuk az eredményt.

*\$doWhat = 5*

Ebben az ágban az általunk megvásárolt tárgyak listáját találjuk. Szintén nagyon egyszerű a felépítése: egy *MySQL* lekérdezés megkeresi az összes olyan tárgyat, aminek az állapota „eladva” (**o\_status = 'todo'**) és amelynek a tulajdonosa az éppen bejelentkezett felhasználó (**o\_userid = '\$user'**). Ha nem kaptunk egyetlen rekordot sem, akkor az illető még nem vásárolt semmit, ezt egy üzenet jelzi. Ha kaptunk adatot az adatbázistól, akkor egy szokásos **while** ciklussal táblázatba rendezzük a tárgyakat és kész.

Ezzel be is fejeztük a **user.php** tárgyalását. Végül, de nem utolsó sorban pedig következzen a fájl, mely a képek megjelenítéséért felelős.

#### g) getimage.php

Ez a fájl foglalkozik az adatbázisban tárolt képek megjelenítésével. 8 lényegi sorból áll, melyből az első négy kiválasztja az adatbázisból a paraméterként kapott azonosítójú tárgy képét. Majd a következő 3 sorban a generált fájl fejléce lesz beállítva a *JPG* képek megjelenítésének megfelelően. Végül egy **echo**-val a kép bináris tartalma bele lesz „töltve” a fájlba, létrehozva ezzel a megjelenített képet.

### 3. Végszó

Ezzel végig is értünk volna az egyes fájlok és ezzel együtt a CSBB tárgyalásán. A forráskódok az alábbi címen lesznek elérhetőek, így mindenki megtekintheti, hogy miről is volt szó az elmúlt pár oldalon:

<http://619media.uw.hu/abkra>

Aki nem értett volna valamit, annak itt van az MSN címem, ahogy a dokumentum elején ígértem. Bárkinek szívesen segítek:

[drunken\\_m@freemail.hu](mailto:drunken_m@freemail.hu)