JEGYZŐKÖNYV

Modern Adatbázis Rendszerek
Féléves feladat

Áruház-raktár hálózat
Blog oldal

Készítette: Stremler László
Neptunkód: AQYO8L

Dátum: 2025.04.14.

MISKOLC, 2025

Tartalomjegyzék

MISKOLC, 2025

Bevezetés

A feladat egy áruház-beszállító hálózat struktúráját mutatja be. A feladat célja egy komplexebb felépítésű XML struktúra bemutatása. Többfajta kapcsolat jelenik meg a struktúrában: 1:1, 1:több, több:több kapcsolat. A féléves feladat egy mindennapi életünket meghatározó rendszer, az áruházak és raktáraik, beszállítóik (egyszerűsített) kapcsolatát hivatott bemutatni. Leegyszerűsített formában kerülnek prezentálásra benne a termékek és azok jellemzőik, illetve a beszállítók és típusaik.
A felépítés szöveges leírása:
Az áruházban található termékek utánpótlásai az áruház raktárában kapnak helyet, ahová a beszállító a saját raktárából szállítja a termékeket. A raktárakban lévő termékek különböző tulajdonságokkal vannak ellátva (termék azonosító, név, darabszám, kategória). Amint egy termék bekerül az áruház raktárába, bővül pár tulajdonsággal (áruház azonosító, leírás, ár). Megjelenítésre kerül az áruház-beszállító kapcsolat is, ahol egy kapcsolathoz tartozó változóban tárolva van az adott áruház heti átlagos termékberendelésének száma. A struktúra részletesebb bemutatása érdekében készült egy ER modell is, amely tartalmazza az egyedeket, illetve az egyedek közötti kapcsolatokat.

Az ER-modell összesen 5 egyedet tartalmaz, melyek a következők:

- **Áruház,**
- **Beszállító,**
- **Beszállító raktár,**
- **Áruház raktár tartalom,**
- **Akciós termékek**

Először is az **Áruház** egyedet szeretném bemutatni. Ez az egyed tárolja az áruházak legfőbb tulajdonságait (*név, cím, áruház azonosító*). A **cím** egy összetett tulajdonságként jelenik meg, amely *az irányitószám, település, utca, házszám* elemekből épül fel. Ez az egyed az összeköttetés a beszállító és az áruházi raktár között. Elsődleges kulcs az AruhazID, amely minden áruház esetében egyedi és ez alapján lehet beazonosítani a raktárban, hogy az adott termék melyik áruház polcain található meg, illetve hogy az adott áruház mely beszállítóktól rendeli az áruit.

A következő fontos egyed a **Beszállító**. Ez az egyed tárolja a beszállító cégek adatait (*azonosító, név, termék kategória, átlagos kiszállítási idő*). A kategória egy előre meghatározott értéket felvevő egyed (*élelmiszer, üdítő, autó alkatrész, stb.*) Itt van meghatározva hogy mely beszállítók mely áruházaknak szállítanak. Mivel több áruház több beszállítótól is rendelhet árut és több beszállító is beszállíthat ugyanolyan kategóriájú árut

az áruházba, ezért a beszállító és az áruház **között N:M (több:több) kapcsolat** van. A kapcsolatot pedig az **Áruház-beszállító kapcsolat** jellemzi, amely összeköti az Áruház és Beszállító egyedeket, valamint meghatározza az *átlagos rendelt árumennyiséget*.

Mivel a beszállító saját árukészlet nélkül nem ér sokat, ezért egy **beszállító raktár termék** egyedet is létrehoztam. Ebben a példában egy beszállító (mivel csak egy kategóriájú terméket szállít) egy raktárral rendelkezik és egy raktárhoz csak egy beszállító tartozik, ezért ez a kettő között 1:1 kapcsolat található. Egy beszállító raktárban lévő termék tulajdonságai a következők**: termék azonosító, név, darabszám, kategória**.

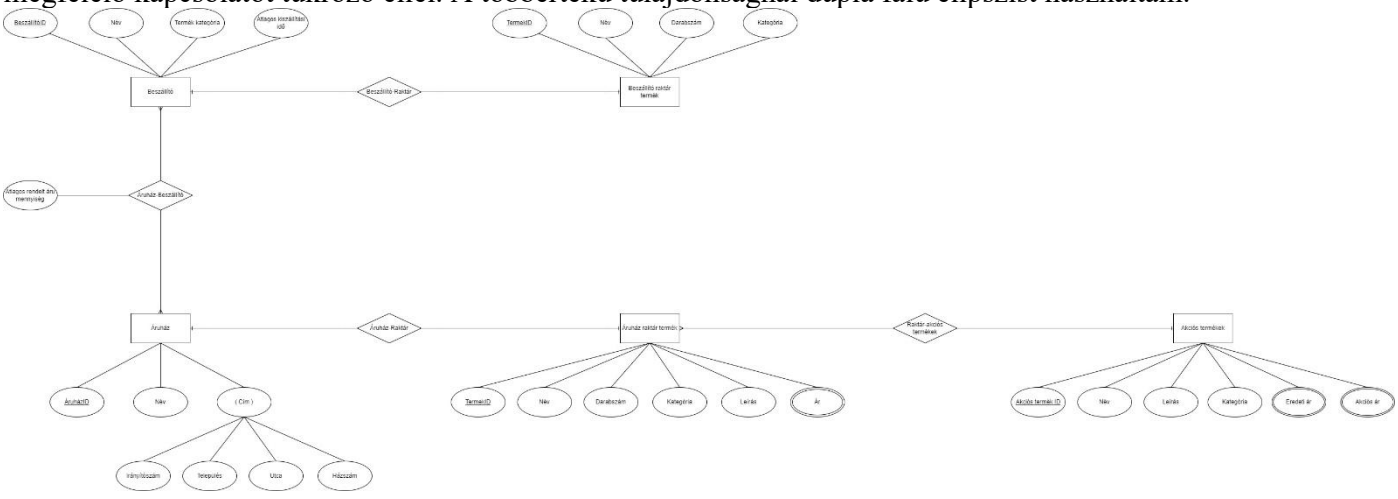
Az áruháznak is rendelkeznie kell saját raktárral, ahonnan feltölti az áruit és ahol tárolja a beérkezett árukat, így egy áruház raktár egyed is megtalálható a modellemben. Hasonlóképp, mint a beszállítónál, itt is egy áruház egy raktárral rendelkezik és egy raktár csak egy áruházat "szolgál ki", ezért a két egyed között 1:1 kapcsolat van. Tulajdonságok tekintetében magában foglalja a beszállító raktár tulajdonságait, illetve kibővítésre került **a kategória, leírás és ár** tulajdonságokkal. *Az ár tulajdonság egy több értékű tulajdonság, ezáltal megadható hazai és külföldi valutában is az adott termék ára.*

Végül pedig létrehoztam egy akciós termékek egyedet is, mivel az áruházak csak úgy tudnak gördülékenyen működni, ha néha engednek az árból. Az akciós termékek egyed közvetlen kapcsolatban áll az áruház raktárával, ezáltal könnyedén nyomonkövethető mely termékek akciósak. A két egyed között 1:N (egy:több) kapcsolat található, mivel egy raktárbeli elemnek csak egy akciós variánsa lehet, viszont egy akciós termék több áruház raktárában is megjelenhet. Egy akciós termék a következő tulajdonságokkal van ellátva: **akciós termék azonosító, név, leírás, kategória, eredeti ár, akciós ár**. Ahogy az áruház raktárban, itt is az árra vonatkozó tulajdonságok többértékűek, ezáltal megadhatóak hazai és külföldi valutában is.

1.feladat:

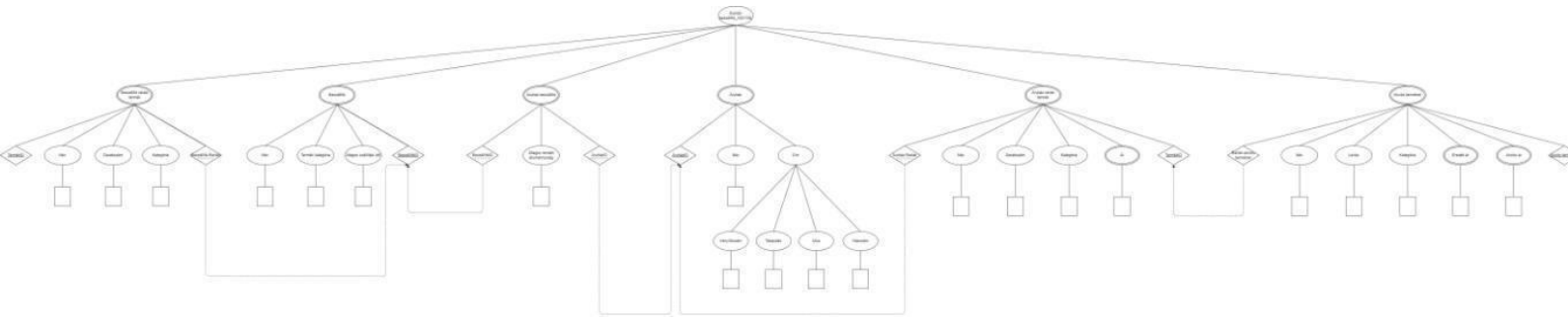
1.a feladat: Az adatbázis ER modell tervezése

Az ER-modellt a Draw.io nevezetű programban készítettem el. Először az egyedeket hoztam létre, amiket téglalappal jelöltem, majd azok tulajdonságait, amit elipszis alakzattal reprezentálok. Ezt követte az egyedek és tulajdonságaik összeköttetése, végül az egyedek összekötése a megfelelő kapcsolatot tükröző éllel. A többértékű tulajdonságnál dupla falú elipszist használtam.



1.b feladat: Az adatbázis konvertálása XDM modellre

Az ER modell elkészítése után létrehoztam az XDM modellt, amely reprezentálja az egyedek közötti kapcsolatokat egy mélyebb szinten. Itt már megjelennek az elsődleges, valamint idegenkulcsok. Az idegenkulcsok rámutatnak egy másik egyed elsődleges kulcsára, ezt nevezzük hivatkozásnak.



1.c feladat: XML dokumentum készítése XDM modell alapján

Az XDM modell elkészítése után elkezdtem felépíteni az XML dokumetumot, amely reprezentálja a modellt. Az XML dokumentumban megjelenik struktúrált formában a fentebb említett cím tulajdonság, amely gyerekelemek segítségével írja le egy áruház pontos címét. Érdemes továbbá megemlíteni, hogy az akciós termékek elemben az árat tükröző Eredet_ar és Akcios_ar elemek mindig párban jelennek meg, tehát egy terméknek több pénznemben is megjelenik mind az akciós, mind az eredeti ára, ezzel szerettem volna tükrözni az elem többértékűségét. A pénznemet attribútumként lehet megadni az egyes ár elemekben.

1.d feladat: XMLSchema készítése XML dokumentum alapján

Az XML dokumentumban szükség van megkötésekre, hogy létrejöjjön a helyes struktúra, ezért elkészítettem hozzá az XMLSchema-t, ami leírja az XML dokumentum felépítéséhez szükséges elemeket, valamint az elemek típusait. Továbbá lekorlátozza bizonyos elemek számosságát, ami szükséges az adott struktúrában. Definiálásra kerülnek benne az elsődleges, illetve idegenkulcsok is, amik összeköttetést biztosítanak az elemek között, ezáltal egyszerűbb lekérdezést tesz lehetővé.

2.feladat (Java CRUD):

2.a feladat: Adatolvasás és másolása (Create & Read)

A dokumentum olvasó Java program először beolvassa a megadott fájlt, ezután különböző függvényekkel kiírom struktúrált formában a konzolra. A struktúráltságért egyedi formázó függvényeket hoztam létre, amelyek bemenetként megkapják a beolvasott dokumentumot, valamint a kimeneti fájlt. Minden elemhez létrehoztam egy read függvényt, amely az adott elem összes elemét lekéri, megfelelő nevű változókba, majd ebből felépíti a struktúrát. A struktúra felépítésért a *printToFileAndConsole* függvény felel, amely bemenetként megkapja az elemet String-ként, valamint megkapja a konzol elérési útját (System.out) és a fájlt, amibe írni szeretnénk. Kimeneti fájl neve: *XMLAQYO8L_Copy.xml*
Ez a művelet a DomCopyAQYO8L.java fájlban lett megvalósítva.

2.b feladat: Adatmódosítás (Update)

Az adatmódosító függvényben először beolvasásra kerül az XML dokumentuma, majd annak elemei kerülnek módosításra. Módosításra kerülnek az áruházak és beszállítók nevei, valamint az áruház termékek árainak pénzneme. Ez összesen 9 db módosítás. A módosított dokumentumot végül kiírja a konzolra.
Ez a művelet a DomModifyAQYO8L.java fájlban került megvalósításra.

2.c feladat: Adatlekérdezés (Read)

Ebben a feladatban próbáltam egyszerű és összetett lekérdezéseket is készíteni. Az első lekérdezés lekéri az összes áruházat.

A második lekérdezésben lekérdezésre kerülnek az áruház-beszállító kapcsolatok annyi csavarral, hogy nem az ID-kat íratom ki, hanem az áruház és a beszállító nevét, ezáltal több elemcsoporton átívelő lekérdezés jön létre.

A harmadik lekérdezésben implementáltam egy maximum keresést az átlagos rendelt árumennyiség tekintetében. Végigiterál az összes áruház-beszállító kapcsolaton és megkeresi a legtöbb árut rendelt áruházat, majd ennek a nevét írja ki konzolra. (Tehát itt is több elemcsoporton átívelő lekérdezés történik).

A negyedik lekérdezésben az összes Miskolcon található áruházat kérdezem le. Ezt úgy valósítottam meg, hogy az összes áruházon végigiterálok és lekérem a címeiket. Ahol a település gyerekelem megegyezik Miskolccal, azt az áruházat kiíratom a konzolra.

MISKOLC, 2025

Az utolsó lekérdezésben pedig azokat a beszállítókat kérdezem le, akik maximum 2 óra alatt képesek teljesíteni a rendelést. Végigiterálok az összes beszállítón és ahol a kiszállítási idő 2 óránál vagy 48 percnél nem nagyobb, azt kiíratom a konzolra.
Ez a művelet a DomQueryAQYO8L_szoveges.java fájlban lett megvalósítva.

2.d feladat: Adatírás (Create)

Ebben a feladatban létrehozok áruházakat, beszállítókat, áruház-beszállító kapcsolatokat, beszállító raktárakat, áruház raktárakat, valamint akciós termékeket és ezeket struktúráltan iratom ki a konzolra, majd ugyanígy fájlba. A működési elve hasonló az adatolvasás feladatéhoz, mivel egy dokumentumot kapnak a kiíró függvények és azt dolgozzák fel. Azonban itt a dokumentumot nem beolvasom, hanem felépítem saját függvények segítségével. Az *add* kezdetű függvények egészítik ki a dokumentumot a megfelelő elemekkel, teljesen tetszőleges névvel, illetve további tulajdonságokkal lehet ellátni egy-egy elemet (nyílván a struktúra határain belül). A *read* kezdetű függvények pedig paraméterként megkapják a kitöltött dokumentumot és kiírják a konzolra, valamint fájlba. A fájl amibe kiír a következő: *XMLAQYO8L_New.xml*. Fontos még megjegyezni, hogy az akciós termékek árai HashMap-ekben vannak tárolva, ezáltal egyszerű kivenni belőlük a pénznemet és az adott valutában az értékét.
Ez a művelet a DomWriteAQYO8L.java fájlban lett megvalósítva.

3. feladat:

Bevezetés

Az utolsó feladatnak egy egyszerű blog oldalt készítettem, ahol lehet regisztrálni, bejelentkezni, posztot feltölteni és posztot törölni. A bejelentkezést követően a felhasználó sütiben kap egy tokent, amely egy órán keresztül érvényes. Ezt követően a újbóli bejelentkezés szükséges. Poszt feltöltéshez egy címsorra és tartalomra van szükség. A felhasználó az oldalon megtekintheti a saját posztjai és törölheti vagy szerkesztheti azokat.

3.a feladat: MongoDB adatbázis

Adattároláshoz egy nem relációs adatbázist, a MongoDB-t választottam. Az adatbázis egyedisége, hogy dokumentumokként menti el az objektumot. Egy-egy ilyen dokumentum JSON formátumot vesz fel, ezáltal egyszerű kiolvashatóságot és bővíthetőséget tesz lehetővé. A dokumentumokat kollekciókban tárolja, amik lényegében egy keretbe foglalják a dokumentumokat. Egy-egy kollekcióra megkötéseket lehet létrehozni, amiket be lehet állítani hogy cask az új dokumentumok esetében nézze, vagy futtassa le a meglévő dokumentumokra is.

A feladathoz két kollekció készült: users és posts.
A users kollekció tárolja a felhasználókat. Egy user dokumentum két mezővel rendelkezik: username és password. Mindkét mező megadása kötelező, a validator üres dokumentumot nem enged beszúrni.
A felhasználó validátora:

```
{
  $jsonSchema: {
    bsonType: 'object',
'username'
                ,
],'passwor
             d'
properties:
                {
  username:
                 {
       bsonType: 'string',
       description: 'A felhasználónévnek szövegesnek kell lennie és kötelező
     megadni!' },
     password: {
       bsonType: 'string',} } description: 'A jelszó megadása kötelező.'
} }
```

A postos kollekcióban kerülnek tárolásra a felhasználók által létrehozott blog bejegyzések. Két adattagal rendelkezik egy-egy dokumentum: cím és tartalom. A kollekció validátora úgy van beállítva, hogy mindkettő megadása kötelező.
A posztok validaátora:

```
{
  $jsonSchema: {
    bsonType: 'object',
      'title',
   ],'content'
    properties: {
      title: {
        bsonType: 'string',
        description: 'Cim megadása kötelező!'
      },
      content: {
        bsonType: 'string',
    } } description: 'Tartalom megadása kötelező!'
} }
```

Az adatábázis neve: sl_moderndb_systems

3.b feladat: Python Flask backend

A webalkalmazás backendjét a Flask keretrendszerben valósítottam meg, ami Python nyelven íródott. A végpontok a következőek:

- bejelentkező végpont
- regisztrációs végpont
- poszt felvitel végpont
- poszt törlés végpont
- posztok lekérdezése végpont
- más ember posztjainak lekérdezése végpont
- poszt módositás végpont
- token ellenőrző végpont

Bejelentkezéskor a backend vissza ad egy JWT tokent, ami egy óra időtartamig él, ezt követően inaktívvá válik és újat kell kérni (újra be kell bejelntkezni).

A backend a bejelentkező és regisztrációs végpontokon kívül minden HTTP kérésnél elvárja a token meglétét és validságát.
A token ellenőrző végponton kívül az összes végpont végez adatbázis műveletet valamilyen formában.
A backend 2 fájlból áll: config.py és app.py. A config fájlban találhatóak a csatlakozáshoz szükséges adatok, míg az app.py tartalmazza a végpontokat és az alkalmazás logikát.

3.c feladat: HTML-JS-CSS frontend

A webalkalmazás frontendje sima HTML-JS-CSS-ben lett megvalósítva, kihasználva a JavaScript aszinkron előnyeit. Minden HTTP kérés async kérésként fut le, tehát bevárásra kerül a kérés eredménye.

A stack választásának legfőbb indoka az egyszerűség volt, nem gondoltam szükségesnek egy keretrendszer bevonását.
A frontend forrásfájlok a frontend mappában találhatóak. 3 darab fájlból áll a frontend: index.html, ez az oldal elrendezését tartalmazza, styles.css, itt találhatóak a stílusra vonatkozó paraméterek, és app.js, ahol az alkalmazás logika található.