JEGYZŐKÖNYV

Webes adatkezelő környezetek

Féléves feladat

Rbeer

Készítette: Váradi Bence Lénárd
Neptunkód: **FJYXPC**
Dátum: **2025. december**

Miskolc, 2025

Tartalomjegyzék

Bevezetés

A jegyzőkönyv célja egy sörértékelő alkalmazás XML-alapú adatstruktúrájának megtervezése. Ennek részeként elkészülnek az adatmodellek – először az ER, majd az XDM –, valamint bemutatásra kerül a modell alapján létrehozott XML dokumentum és az azt leíró XML séma.

A második rész a dokumentum feldolgozására koncentrál a DOM szabvány segítségével: kitér az adatok beolvasására, lekérdezésére és módosítására Java nyelven.

Az ER modell a fogalmi szintű tervezést támogatja: definiálja az egyedeket, azok kapcsolatát és a hozzájuk tartozó attribútumokat. A koncepcionális modellből kiindulva az XDM a hierarchikus, fastruktúrájú ábrázolást adja meg: kijelöli a gyökérelemet, az elemek közötti hivatkozási viszonyokat, valamint az ismétlődő elemeket.

Az XML dokumentum a modellt konkrét adatokkal szemlélteti, míg az XSD séma formálisan rögzíti a szerkezetet, típusokat és az érvényességi feltételeket.

A DOM feldolgozás célja, hogy a FJYXPC_XML.xml tartalmát programból is kezeljük:

- *olvasás*: a teljes dokumentum blokkos megjelenítése,
- *lekérdezés*: legalább négy hasznos információ kigyűjtése
- *módosítás*: legalább négy, adatintegritást megőrző változtatás, a műveletek és az érintett egyedek blokkos kiírásával.

1. Az ELSŐ feladat címe

Fogalmi és logikai tervezés, majd XML és XSD létrehozása.

1.1 Az adatbázis ER modell tervezése megvalósítása

A rendszer adatainak logikai felépítését az elkészült **ER (Entity-Relationship) modell** ábrázolja, amely a főbb entitásokat és azok kapcsolatait mutatja be. Az ER modellben azonosíthatók a kulcsfontosságú entitások, mint például a **felhasználók, sörök, főzdék, értékelések, vásárlások, forgalmazók és üzletek**, valamint a köztük fennálló kapcsolatok. Például a modell jelzi, hogy mely felhasználók értékelnek vagy vásárolnak bizonyos söröket, illetve mely söröket mely főzdék készítik.

Az entitások közötti kapcsolatok, az **egy-a-többhöz** és **több-a-többhöz** típusú relációk, valamint a kulcsattribútumok lehetővé teszik az adatok logikai szerkezetének áttekinthetőségét és a redundanciák minimalizálását. Az ER modell így alapot biztosít az adatbázis hatékony tervezéséhez, valamint az adatok későbbi kezeléséhez és lekérdezéséhez.

Egyedek és tulajdonságok:

1.1.1. Felhasználó (felhasznalo)

Tulajdonságok:

- feId – egyedi azonosító (primary key)
- nev – a felhasználó neve
- elerhetoseg:
    - email – e-mail cím
    - telefonszam – telefonszám
- jelszo – felhasználói jelszó
- profilkep – profilkép fájl neve

Magyarázat: ez az entitás reprezentálja a rendszer felhasználóit, akik vásárolhatnak és értékelhetnek söröket. Az elérhetőségi adatok egy al-elemen belül vannak összefoglalva.

1.1.2. Sör (sor)

Tulajdonságok:

- sId – sör azonosító (primary key)
- foId – a főzde azonosítója, amely készítette (foreign key)
- nev – a sör neve
- mufaj – sör típusa (pl. Ale, Lager)
- alkohol – alkoholtartalom (%)

Magyarázat: ez az entitás a különböző sörfajtákat tartalmazza. A foId segítségével kapcsolódik a főzdékhez.

1.1.3. Főzde (fozde)

Tulajdonságok:

- fozdeId – egyedi azonosító (primary key)
- nev – főzde neve
- szlogen – marketing szlogen
- alapitva – alapítás éve
- cim:
    - varos – város
    - utca – utca
    - hazszam – házszám

Magyarázat: a főzdék entitása tartalmazza a sörgyártó vállalkozás adatait és címét. Egy főzde több sört is készíthet.

1.1.4. Értékelés (ertekeles)

Tulajdonságok:

- feId – a felhasználó azonosítója (foreign key)
- sId – a sör azonosítója (foreign key)
- pontszam – 1–5 pont közötti értékelés
- ertekeles – szöveges értékelés

Magyarázat: ez az entitás jelzi, hogy mely felhasználó milyen pontszámot és véleményt adott egy sörhöz. Több értékelés is tartozhat ugyanahhoz a sörhöz.

1.1.5. Címke (cimke)

Tulajdonságok:

- cId – egyedi azonosító (primary key)
- kaloria – kalóriatartalom
- osszetevok – hozzávalók listája

- nettoTerfogat – térfogat literben
- ibu – keserűség (International Bitterness Unit)

Magyarázat: a címke entitás a sör részletes jellemzőit tartalmazza, például összetevők és kalóriatartalom alapján.

1.1.6. Vásárlás (vasarlas)

Tulajdonságok:

- uId – a felhasználó azonosítója (foreign key)
- sId – a sör azonosítója (foreign key)
- ar – vásárlás ára
- datum – vásárlás dátuma

Magyarázat: Ez az entitás rögzíti, hogy mely felhasználó melyik sört vásárolta meg, mennyiért és mikor. Kapcsolódik a felhasználókhoz és a sörökhöz.

1.1.7. Forgalmazó (forgalmazo)

Tulajdonságok:

- foId – egyedi azonosító (primary key)
- nev – cég neve
- webhely – weboldal URL
- szekhely – telephely városa
- vevoszolgalat – ügyfélszolgálati e-mail

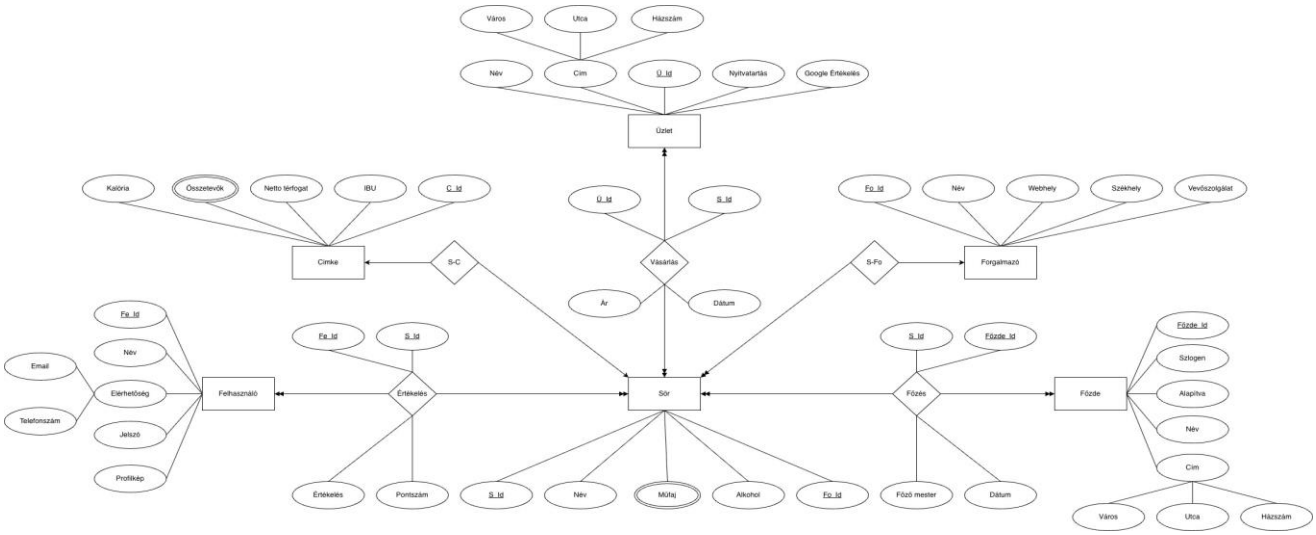
Magyarázat: a forgalmazók azok a cégek, amelyek a söröket terjesztik. Egy forgalmazó több sörhöz is kapcsolódhat.

1.1.8. Üzlet (uzlet)

Tulajdonságok:

- uId – egyedi azonosító (primary key)
- nev – üzlet neve
- nyitvatartas – nyitvatartási idő
- googleErtekeles – értékelés átlaga
- cim:
    - varos
    - utca
    - hazszam

Magyarázat: az üzletek az értékesítés helyszíneit jelentik. Kapcsolódhatnak vásárlásokhoz és forgalmazókhoz.

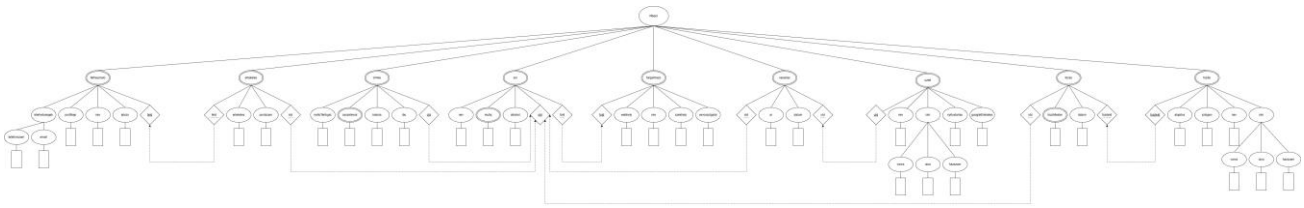


1. Ábra ER modell

1.2 Az adatbázis konvertálása XDM modellre

Az XDM modellben háromféle jelölést használok: **ellipszis, rombusz és téglalap**. Az **ellipszis** az elemeket jelöli, ide tartoznak minden egyed és tulajdonság (kivéve a kulcsok). A **rombusz** az attribútumokat ábrázolja, amelyek a kulcs tulajdonságokból származnak. A **téglalap** a szöveget jelöli, amely az XML dokumentumban fog megjelenni.

Azok az elemek, amelyek többször is előfordulhatnak. Az idegen kulcsok és a kulcsok közötti kapcsolatot **szaggatott vonalas nyíllal** ábrázolom. A modellben a vonalak soha nem keresztezik egymást, így biztosítva a tiszta és áttekinthető ábrázolást.



2. ábra XDM modell

1.3 Az XDM modell alapján XML dokumentum készítése

Az XML dokumentum felépítését az **XDM modell** alapján kezdtem, a gyökérelem létrehozásával, amely a teljes struktúra kiindulópontját jelenti. Ezt követően a főbb gyermekelemeket részletesen definiáltam, és mindegyikből **kettő-kettő példányt** készítettem, hogy a modell különböző előfordulásait is szemléltetni tudjam. Az **azonosítók és hivatkozások** kezelésére az attribútumokat alkalmaztam, elsődleges kulcsokként (primary key) és idegen kulcsokként (foreign key), biztosítva ezzel az adatok egyértelmű kapcsolódását és az integritás megőrzését.

Minden szülőelemhez felvettem a hozzá tartozó gyermekelemeket, ügyelve arra, hogy a **többértékű tulajdonságok** legalább két előfordulást tartalmazzanak, így modellezve a valós adatszerkezetet. Az **összetett tulajdonságokat**, amelyek további részleteket tartalmaznak, al-elemekre bontottam, hogy a struktúra áttekinthető, logikus és könnyen kezelhető legyen. Az így kialakított dokumentum tükrözi az ER modellből és az XDM jelölésekből levezetett hierarchikus kapcsolatokat, és lehetővé teszi az adatok egyértelmű, jól strukturált tárolását, amely alkalmas további feldolgozásra, például lekérdezésekre vagy exportálásra.

XML-részlet:

```
<!-- Felhasználó entitás -->
<felhasznalo feId="1">
   <nev>Kiss Adam</nev> <!-- Felhasználó neve -->
   <elerhetoseg>
      <email>adam.kiss@example.com</email> <!-- Felhasználó e-mail címe -->
      <telefonszam>06201234567</telefonszam> <!-- Felhasználó telefonszáma -->
   </elerhetoseg>
   <jelszo>pass1234</jelszo> <!-- Felhasználó jelszava -->
   <profilkep>profil1.jpg</profilkep> <!-- Profilkép fájl neve -->
</felhasznalo>

<!-- Értékelés entitás, amely a felhasználóhoz és sörhöz kapcsolódik -->
<ertekeles feId="1" sId="1">
   <pontszam>5</pontszam> <!-- Értékelés pontszáma -->
   <ertekeles>Kivalo iz, frissito!</ertekeles> <!-- Értékelés szövege -->
```

```
</ertekeles>

<!-- Sör entitás -->
<sor sId="1" fold="1">
    <nev>Golden Ale</nev> <!-- Sör neve -->
    <mufaj>Ale</mufaj> <!-- Sör típusa -->
    <alkohol>5.2</alkohol> <!-- Alkohol tartalom százalékban -->
</sor>
```

FJYXPC_XML.xml file-ra mutató link

1.4 Az XML dokumentum alapján XMLSchema készítése

Amikor az Rbeer XML-ből az XSD sémát készítettem, először a dokumentum szerkezetét elemeztem: milyen elemek ismétlődnek, melyek hordoznak azonosítót, és hogyan kapcsolódnak egymáshoz. Ez alapján egységes fő gyökérelemet (Rbeer) hoztam létre, majd a benne található rekordokat típusokra bontottam. A cél az volt, hogy a séma ne csak a hierarchiát írja le, hanem egyfajta modellként viselkedjen, amely már az XML szinten is szabályozza az adatkezelést.

A komplex struktúrákat xs:complexType-okkal definiáltam, hogy az elemekhez tartozó al-elemek és attribútumok jól kezelhetők legyenek. Ahol egy elem többször is előfordulhat, ott maxOccurs="unbounded"-ot alkalmaztam, ezzel biztosítva, hogy a séma rugalmas legyen a rekordok számának változására. Az entitásokhoz (felhasználó, sör, főzde stb.) kötelező azonosító attribútumokat adtam (feId, sId, fozdeId), így egyértelműen hivatkozhatok rájuk. Ezek az azonosítók xs:ID típussal készültek, hogy a dokumentumon belüli egyediség technikailag is ellenőrizhető legyen.

A kapcsolatok kezeléséhez az XML Schema beépített referencialista-mechanizmusait használtam:

- **xs:key** → egyedi elsődleges kulcs az adott elemgyűjteményen belül
- **xs:keyref** → idegen kulcs, amely másik elem azonosítójára hivatkozik

Ez lehetővé tette, hogy ne csak a struktúra legyen konzisztens, hanem az adatok közötti relációk is érvényesek legyenek. Például a sörértékeléseknél az értékelés feId attribútuma kizárólag olyan felhasználóra mutathat, amely ténylegesen létezik — ezzel adatbázis-szerű integritást tudtam biztosítani XML szinten. Hasonló módon a sörök főzdeId-jére is keyref-et állítottam be, így nem kerülhet be olyan sör, amely nem kapcsolódik valós főzdéhez.

Az adattípusoknál az xs:string, xs:integer, xs:decimal, xs:gYear, xs:date, xs:anyURI típusokat használtam, hogy a tárolt adatok formailag is validálhatók legyenek. A mennyiségi értékekhez (pl. alkoholtartalom, értékelési pontszám) xs:decimal-t alkalmaztam, opcionális restrikciókkal, például minInclusive és maxInclusive határokkal, hogy a megadott értékek ne léphessék túl a logikailag elfogadható kereteket. A dátum alapú mezőknél (pl. sör megjelenési éve vagy értékelés időpontja) xs:gYear és xs:date típusokat használtam, ezzel megelőzve a hibás dátumformátumokból adódó inkonzisztenciát. A sémában igyekeztem különválasztani az egyszerű adattípusokat (xs:simpleType) a komplex struktúráktól, így külön definíció készült például a sör stílusának felsorolására (enum), ami csak előre megadott stíluskódokat fogad el. Ez segített az adatbevitel standardizálásában, mert a szöveges variációkat (pl. „IPA”, „India Pale Ale” vagy „I.P.A.”) egy egységesített kódrendszerrel váltottam ki. A séma így nemcsak a struktúrát definiálja, hanem garantálja az adatok összefüggését

és minőségét is. Az XML dokumentumot az XSD ellen validálva ezután ellenőrizhetem, hogy minden kapcsolat helyes-e, nincs-e „árva” referencia, illetve minden mező értéke megfelel-e a típusnak és a megkötéseknek. Ennek köszönhetően a rendszer jól skálázhatóvá és hosszú távon könnyen karbantarthatóvá vált.

Xsd részlet:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">

<xs:element name="Rbeer">
   <xs:complexType>
      <xs:sequence>
         <!-- Felhasználók -->
         <xs:element name="felhasznalo" type="FelhasznaloTipus"
maxOccurs="unbounded"/>
         <!-- Felhasználó által adott értékelések -->
         <xs:element name="ertekeles" type="ErtekelesTipus" maxOccurs="unbounded"/>
      </xs:sequence>
   </xs:complexType>
   <!-- ====== Primary Key ====== -->
   <xs:key name="felhasznaloKey">
      <xs:selector xpath="felhasznalo"/>
      <xs:field xpath="@feId"/>
   </xs:key>
   <!-- ====== Foreign Key: értékelés -> felhasználó ====== -->
   <xs:keyref name="ertekeles_to_felhasznalo" refer="felhasznaloKey">
      <xs:selector xpath="ertekeles"/>
      <xs:field xpath="@feId"/>
   </xs:keyref>
</xs:element>
<!-- ====== Felhasználó ====== -->
<xs:complexType name="FelhasznaloTipus">
   <xs:sequence>
      <xs:element name="nev" type="xs:string"/>
      <xs:element name="email" type="xs:string"/>
   </xs:sequence>
   <xs:attribute name="feId" type="xs:integer" use="required"/>
</xs:complexType>
<!-- ====== Felhasználó értékelése ====== -->
<xs:complexType name="ErtekelesTipus">
   <xs:sequence>
      <xs:element name="pontszam" type="xs:integer"/>
      <xs:element name="szoveg" type="xs:string"/>
   </xs:sequence>
   <xs:attribute name="feId" type="xs:integer" use="required"/>
</xs:complexType>
```

FJYXPC_XMLSchema.xsd file-ra mutató link

2. A MÁSODIK feladat címe

Project name: FjyxpcDOMParse

Package: fjyxpc.domparse.hu

Class names: (FjyxpcDomRead, FjyxpcDomModify, FjyxpcDomQuery)

2.1 Adatolvasás (FjyxpcDomRead)

Amikor nekiálltam ennek a kódnak, először is az volt a célom, hogy az XML fájl tartalmát át tudjam ültetni egy olvasható, magyar nyelvű formátumba. Tudtam, hogy a feladat lényege az, hogy az XML-ben lévő angol vagy rövidített tagneveket és attribútumokat át kell magyarítani, ezért létrehoztam két térképet (Map), az egyik az elemek neveit, a másik az attribútumokat tárolta.

Statikus elemek definiálása

```
private static final Map<String, String> adatNevek = new HashMap<>();
private static final Map<String, String> kulcsNevek = new HashMap<>();
```

Ezután elkezdtem feltölteni a térképeket konkrét kulcs-érték párokkal. Igyekeztem minden fontos XML elemet lefedni: felhasználó, sör, főzés, főzde, vásárlás és így tovább. Minden elemhez hozzárendeltem a magyar megfelelőjét, például a "mufaj" kulcshoz a "Műfaj" értéket.

Kulcspárok létrehozása

```
adatNevek.put("mufaj", "Műfaj");
adatNevek.put("alkohol", "Alkoholtartalom");
```

A következő lépésben az XML fájl beolvasását valósítottam meg a DocumentBuilder segítségével. Itt nagyon figyeltem, hogy a gyökérelem normalizálva legyen, így könnyebben tudtam feldolgozni a gyermekelemeket is.

Fájl beolvasása

```
DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();

DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();

Document doc = dBuilder.parse(xmlFile);
```

```
doc.getDocumentElement().normalize();
```

Amikor a fájl már be volt olvasva, megnyitottam egy TXT fájlt, ahova a magyarított adatokat kiírtam. Minden elemet és attribútumot külön sorba írtam ki, és ha egy elemnek mélyebb gyermekei voltak, például a <cim> alatt <varos> és <utca>, akkor beljebb húzással jelöltem a hierarchiát.

Gyerek elemek kiírása

```
if (subElem.hasChildNodes() && hasElementChild(subElem)) {
    printLine(at(subElem.getNodeName()) + ":", writer);
    // mélyebb elemek kiírása
}
```

Az attribútumok esetében először mindig leellenőriztem, hogy létezik-e magyar megfelelőjük a kulcsNevek térképen, ha nem, akkor az eredeti nevüket használtam. Így biztosítottam, hogy semmi se vesszen el és minden könnyen olvasható legyen.

Kimenet magyarosítása

```
String magyarKulcs = kulcsNevek.getOrDefault(attr.getNodeName(), attr.getNodeName());
printLine(magyarKulcs + ": " + attr.getNodeValue(), writer);
```

Egész végig próbáltam a kódot logikusan felépíteni, hogy könnyen követhető legyen, és a kiírt fájl is tiszta, rendezett legyen. Nagyon élveztem a kihívást, mert így egyszerre kellett figyelnem a Java nyelvi részletekre, a fájlkezelésre és az XML struktúrájára. Az egész folyamat során folyamatosan ellenőriztem, hogy minden tag és attribútum a megfelelő magyar névvel jelenjen meg, így a végső TXT fájl átlátható és könnyen értelmezhető lett.

FjyxpcDomRead.java file-ra mutató link

2.2. Adat-lekérdezés (FjyxpcDomQuery)

Amikor nekifogtam ennek a programnak, az volt a célom, hogy az XML-ből beolvasott adatokat úgy kapcsoljam össze, hogy egy felhasználóhoz tartozó értékelt söröket minden további információval együtt meg tudjam jeleníteni. A DOM-alapú XML feldolgozás mellett döntöttem, mert így az egész dokumentumhoz hozzáférek, és könnyen ki tudom nyerni az elemeket és attribútumokat. Első lépésként betöltöttem az XML fájlt és létrehoztam a DOM fastruktúrát.

Fájl beolvasása

```
File inputFile = new File("FJYXPC_XMLTask/FJYXPC_XML.xml");
DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
Document doc = dBuilder.parse(inputFile);
doc.getDocumentElement().normalize();
```

Ezután elkezdtem kinyerni a sörök adatait. A sör objektumoknál ID-vel és névvel dolgoztam, hiszen később ezek alapján kapcsoltam össze az értékeléseket. A sör Elementet magát is eltároltam, mert az attribútumok (például forgalmazó vagy címke ID) később innen érhetők el.

Sörök feldolgozása:

```
NodeList sorLista = doc.getElementsByTagName("sor");
for (int i = 0; i < sorLista.getLength(); i++) {
    Element s = (Element) sorLista.item(i);
    String sId = s.getAttribute("sId");
    String nev = getTextContentOfChild(s, "nev");
    sorNevek.put(sId, nev != null ? nev : "Ismeretlen sör");
    sorElements.put(sId, s);
}
```

A következő részben a címkéket dolgoztam fel. Minden címkéhez különböző részadatok tartoztak: kalória, összetevők, IBU stb. Ezeket összefűztem egy egybefüggő szöveggé, hogy később könnyen megjeleníthető legyen.

Címke összefűzése:

```
String desc = "";
if (kaloria != null) desc += "Kalória: " + kaloria;
if (osszetevok != null) desc += ", Összetevők: " + osszetevok;
if (nettoT != null) desc += ", Netto: " + nettoT;
if (ibu != null) desc += ", IBU: " + ibu;
```

Ezután a forgalmazókat és a felhasználókat emeltem ki az XML-ből. Itt az azonosító és a név volt a legfontosabb, ezekhez később csak lekérések történtek. Ügyeltem arra, hogy ha valami hiányzott, akkor alapértelmezett értéket kapjon.

A legösszetettebb lépés az értékelések összekapcsolása volt. Minden értékelést egy saját objektumba mentettem („ErtekeltSor”), amelybe beírtam a pontszámot, a sör nevét, valamint a hozzá tartozó forgalmazó és címkeinformációkat.

Értékelt sör kódrészlet

```
ErtekeltSor sorObj = new ErtekeltSor(
    sId,
    sorNev,
    pontszam != null ? pontszam : "0"
);

Element sorElem = sorElements.get(sId);
String foId = sorElem.getAttribute("foId");
sorObj.forgalmazoNev = forgalmazok.getOrDefault(foId, "Nincs forgalmazó adat");

String cId = sorElem.getAttribute("cId");
if (cId == null || cId.isEmpty())
    cId = sId;

sorObj.cimkeId = cId;
sorObj.cimkeLeiras = cimkeLeirasok.getOrDefault(cId, "Nincs címke adat (" + cId + ")");
```

Mivel a felhasználókat azonosító alapján raktam össze, minden értékelt sört a megfelelő felhasználó listájához adtam hozzá. Így egy Map-ben létrejött a kapcsolat: felhasználó ID → értékelt sörök listája. Ez biztosította, hogy egymás után tudjam megjeleníteni azokat a tételeket, amelyek egy személyhez tartoznak.

A program végül a konzolra írta ki a végeredményt. Minden felhasználót külön blokkban jelenített meg, majd alatta sorban a hozzá tartozó söröket és azok adatait. A kiíratási rész számomra azért volt fontos, mert ezen keresztül láttam, mennyire jól sikerült az adatkapcsolások logikája.

Az egész folyamat során arra figyeltem, hogy a program ne omoljon össze akkor sem, ha hiányos az XML. Ha valamilyen adat nem szerepelt a dokumentumban, a kód védőértékeket alkalmazott. Így a feldolgozás stabil maradt, és a kapott eredmény is jól áttekinthető lett.

FjyxpcDomQuery.java file-ra mutató link

2.3. Adatmódosítás (FjyxpcDomModify)

Először is, úgy kezdtem, hogy meghatároztam, melyik XML attribútumokhoz szeretném magyar neveket rendelni a könnyebb olvashatóság érdekében. Ehhez létrehoztam egy Map-et, ahol a kulcs az eredeti attribútum neve, az érték pedig a magyar megfelelője. Ez nagyon praktikus volt, mert később minden attribútumhoz könnyen hozzá tudtam férni a magyar nevet használva.

Kulcsok párosítása

```
private static final Map<String, String> kulcsnevek = new HashMap<>();
...
   kulcsnevek.put("feld", "Felhasználó azonosító");
   kulcsnevek.put("sld", "Sör azonosító");
```

Ezután arra gondoltam, hogy nem minden XML elemet szeretnék kiírni a konzolra, csak bizonyosokat, ezért létrehoztam egy Set-et a megjelenítendő elemek nevével. Így egyszerűen tudtam szűrni a dokumentum elemeit.

Kiírt elemek meghatározása

```
private static final Set<String> megjelenitendo = new HashSet<>(Arrays.asList(
      "felhasznalo",
      "vasarlas",
      "sor",
      "fozes"
));
```

A kód következő részében az XML fájlt beolvastam és normalizáltam. Ez fontos volt, hogy az egész dokumentumot egységes struktúrában kezelhessem, így könnyebben tudtam módosításokat végezni a csomópontokon.

A módosításokat lépésenként végeztem. Először az első felhasználó első vásárlásának árát állítottam át 999-re. Ehhez lekértem az összes <vasarlas> elemet, és a megfelelő uId-t ellenőriztem.

Modosítás példa:

```
NodeList vasarlasLista = doc.getElementsByTagName("vasarlas");
for (int i = 0; i < vasarlasLista.getLength(); i++) {
   Element vas = (Element) vasarlasLista.item(i);
   if ("1".equals(vas.getAttribute("uId"))) {
      Node ar = vas.getElementsByTagName("ar").item(0);
      if (ar != null) {
         ar.setTextContent("999");
      }
      break;
   }
}
```

Ezután a sör értékelését és műfaját módosítottam, majd a főzés főzőmesterét. Mindig először lekértem a megfelelő elemeket getElementsByTagName-nel, ellenőriztem az attribútumokat, és ha egyezett, átírtam a kívánt értékre.

A kiírásnál szerettem volna, hogy a magyar kulcsok jelenjenek meg a konzolon, ezért a NamedNodeMap segítségével végigmentem az attribútumokon, és a Map-ből vettem a magyar megfelelőket. A gyermekelemeket is kiírtam, és ha egy elemnek további gyermekelemei voltak, akkor azokat is szépen, behúzva jelenítettem meg.

Kiiratás

```
NamedNodeMap attrs = elem.getAttributes();
for (int a = 0; a < attrs.getLength(); a++) {
   Node attr = attrs.item(a);
   String kulcs = kulcsnevek.getOrDefault(attr.getNodeName(), attr.getNodeName());
   System.out.println(kulcs + ": " + attr.getNodeValue());
```

```
}
```

FjyxpcDomModify.java file-ra mutató link