JEGYZŐKÖNYV

Webes adatkezelő környezetek

Féléves feladat

Cipőbolt

Készítette: Kun Dávid
Neptunkód: GEB15I
Dátum: 2025. December

Miskolc, 2025

Tartalomjegyzék

Bevezetés

Az én féléves feladatom témája egy cipőbolt és az ehhez tartozó XML. Azért erre esett a választásom, mivel nagy cipő rajongó vagyok és úgy gondoltam egy cipőboltnak megvan a megfelelő struktúrája, ahhoz, hogy a féléves feladatomat elkészítsem.

A feladatomban 6 egyed található meg. A 6 egyed a következő: Üzlet, Raktár, Kiszállítás, Cipők, Vásárló és végül Rendelés. Ezek az alábbi módon épülnek fel:

Üzlet:

- Ebben az egyedben vannak tárolva az üzlet alapvető adatai. Az egyik legfontosabb egyed a feladatomban.
- Kapcsolat: A Uzlet egyedhez kettő másik egyed is csatlakozik. Az egyik a Vasarlo, ahol egy M:N kapcsolat létezik, a másik pedig a Raktar egyed, ahol 1:1 kapcsolat létezik. Az 1:1 kapcsolatra az a magyarázat, hogy ebben a feladatban az üzletnek egy raktára van és a raktár is csak egy üzlettel dolgozik együtt.
- Ukod: Ez az üzlet elsődleges kulcsa. Ez megtalálható az összes többi egyedben is és azok erre az ukodra hivatkoznak. Ennek az értéke legyen mondjuk “U001”.
- Nev: Az üzlet nevét tartalmazza. Ez legyen most “Balazskicks”
- Elerhetoseg: Az elérhetőség ketté ágazik kettő külön mezőre. Az egyik az emailcím a másik pedig a telefonszám. Ezek az üzlet elérhetőségét tárolják.
- Nyitvatartas: Szintén ketté ágazik kettőfelé. Az egyik a “tol” a másik pedig az “ig” mező. Az előbbi a nyitvatartás kezdetét tárolja, utóbbi pedig a végét.
- Cim: Három felé ágazik el ez a mező. Megtalálható benne, a város, az utca és a házszám.

Raktár:

- Ennek az egyednek a feladata tárolni az üzlethez tartozó raktárnak az adatait. Jelenleg csak egy darab raktár van a feladatban és egy raktárra is lett tervezve.
- Kapcsolat: A Raktar egyed jelenleg kettő másikhoz kapcsolódik. Az első az Uzlet ahol 1:1 kapcsoalt létezik. A másik kapcsolat pedig egy 1:N kapcsolat, ami a Kiszallitas egyedbe kapcsolódik. Itt a kapcsolat típusára az a magyarázat, hogy egy raktárhoz több kiszállítás is tartozhat, viszont egy kiszállítás csak egy raktárhoz kapcsolódhat.
- Rkod: A raktár elsődleges kulcsa. A többi egyed erre hivatkozva tud kommunikálni vele.
- Ukod: Az üzlethez tartozó idegen kulcs.
- Kapacitas: A kapacitás mező tárolja, hogy a raktárnak mekkora a teljes áru befogadó képessége.
- Muszak. Több értékű mező. Itt lehet beállítani, hogy mi az aktuális műszak a raktárban. Ez lehet délelőttös, délutános és akár éjszakás is.
- Hely: Ez a mező tárolja a jelenleg elérhető szabad helyet a raktárban.
- Vezeto: Itt van eltárolva a jelenlegi raktár vezető neve, aki egyben a műszakvezető is.

Kiszállítás:

- A megrendelések kiszállításáért felelős egyed. Tartalmaz minden információt, amelyre a kiszállításhoz van szükség.
- Kkod: A kiszállítás elsődleges kulcsa.
- Ukod: Az üzlethez tartozó idegenkulcs.
- Kamion: Ketté ágazó mező, mely “modell” -re és “rendszám” -ra ágazik el. Az előbbi tartalmazza a kiszállító kamion modelljét, utóbbi pedig annak rendszámát.
- Sofor: A sofőr neve, aki az előbb említett kiszállító kamiont vezeti.
- Telefonszam: A sofőr telefonszámát tartalmazza.
- Cim: A kiszállítás címe. Ez négyfelé ágazik el. Az első az ország, második a város, harmadik az utca és végül negyedikként a házszám.

Cipők:

- Az üzletben elérhető cipők adatait tartalmazza. Minden szükséges adatot tartalmaz ahhoz, hogy egy cipőt pontosan meg lehessen találni. Ez a legfontosasbb egyed az én feladatomban, hiszen eköré épül fel minden.
- Kapcsolat: A cipők egyed a rendelés egyedhez kapcsolódik. A típusa: 1:N. Ez azért van, mivel egy rendeléshez tartozhat több cipő is, viszont egy cipő csak egy rendeléshez kapcsolódhat.
- Ckod: A cipők egyedhez tartozó elsődleges kulcs.
- Ukod: Az üzlethez tartozó idegenkulcs
- Keszlet: Többértékű mező. Tartalmazza, hogy jelenleg hány darab érhető el az adott cipőből.
- Marka: A cipő márkáját tartalmazó mező. Pl: “Nike”.
- Modell: A márkához megfelelő modell nevét tartalmazza. Szóval, ha a márka “Nike” akkor csak ehhez a márkához tartozó modell lehet ebben a mezőben.
- Szin: A cipőhöz tartozó színt tárolja. Több értékű, hiszen egy cipőnek több színe is lehet.
- Meret: Egy adott cípőhöz tartozó méretet tárol. Több értékű, hiszen egy cipő több méretben is elérhető lehet.
- Ar: A cipő árát tartalmazza. Ez egy konstans, tehát, az adott cipő ára mindig ugyanannyi. Nem számít, milyen színű, vagy milyen a mérete, mindig ugyanannyi.

Vásárló:

- Ez az egyed tartalmazza a vásárló adatait. Minden szükséges információ benne van, ahhoz, hogy egy adott vásárlót azonosítani lehessen és felkeresni, ha probléma van a rendeléssel.
- Kapcsolat: A Vasarlo egyed kapcsolódik a Rendeles egyedhez is, illetve az Uzlet egyedhez is. Előbbinél egy 1:N kapcsolat létezik, utóbbinál pedig M:N kapcsolat. Utóbbi esetén ez azért van, mivel egy vásárló több üzletbe is bemehet és egy üzletbe több vásárló is lehet.
- Vkod: A vásárlóhoz tartozó elsődleges kulcs.

- Ukod: Az üzlethez tartozó idegen kulcs.
- Nev: Azt a nevet tartalmazza, amit a vásárló megadott a rendelési folyamat során.
- Cim: A vásárló által megadott szállítási cím. Háromfelé ágazik el, város, utca és házszám.
- Eletkor: Azt az életkort tartalmazza, amit a vásárló megadott a rendelés során.
- Fizetesi mod: Itt van tárolva, hogy a felhasználó milyen fizetési módot választott. Háromfelé ágazik el, ezek a következők: Készpénz, bankkártya és online. Az első kettő utánvétet jelent, míg a harmadik online fizetést. Ez szintén elágazik kétfelé, még pedig: Google Pay és Apple Pay.

Rendelés:

- A rendelés adatait tartalmazó egyed.
- Kapcsolat: A rendelés egyed és a vásárló egyed egy 1: N kapcsolatban állnal. Azért, mivel egy rendelés egy vásárlóhoz tartozhat csak, de egy vásárlóhoz több rendelés is tartozhat, mivel nem szükséges mindent egy rendelésben megvenni. Kapcsolódik még a Cipo egyedhez is, ahol a kapcsolat típusa szintén 1: N.
- Rendeleskod: A rendeléshez tartozó elsődleges kulcs.
- Ukod: Az üzlet idegenkulcsa.
- Cipo: A rendelésben szereplő cipő / cipők adatait tartalmazza.
- Osszeg: A Cipo mező alapján meghatározott összeget tartalmazza.

1. Feladat: ER és XDM

1.1 Az adatbázis ER modell tervezése

Az adatbázis felépítése már előre meg lett tervezve, így az ER modell elkészítése nem jelentett nagy problémát. Az ER modell 6 egyedből áll. Ezek a következők: Cipő, Rendelés, Vásárló, Üzlet, Raktár, Kiszállítás. Az egyedek közötti kapcsolatok a következő képpen épülnek fel:
Cipő - Rendelés: 1:N kapcsolat
Rendelés - Vásárló: 1:N kapcsolat
Vásárló - Üzlet: M:N kapcsolat
Üzlet - Raktár: 1:1 kapcsolat
Raktár - Kiszállítás: 1:N kapcsolat

Maga az ER modell:
Ide a linket majd

1.2 Adatbázis konvertálása XDM modellre

Az XMD modellre való konvertálás nem volt bonyolult, mivel az ER modell kiválóan lett megtervezve. Az előadásokon és gyakorlati órákon bemutatott példákból dolgoztam, így még könnyebb volt a konvertálás folyamata.
XDM modellt majd ide

1.3 Az XDM modell alapján XML dokumentum készítése.

Az XML dokumentum elkészítése inkább időigényes volt, mint nehéz. Mivel az adatbázis két legfontosabb egyede az Üzlet, illetve a Cipő, ezért ezeket készítettem el először.

```
<Uzlet ukod="U001">
    <elerhetoseg>
      <emailcim>info@cipobolt.hu</emailcim>
      <telefon>+36301234567</telefon>
    </elerhetoseg>
    <nyitvatartas>
      <tol>08:00</tol>
      <ig>20:00</ig>
    </nyitvatartas>
    <cim>
      <varos>Budapest</varos>
      <utca>Fő utca</utca>
      <hazszam>10</hazszam>
    </cim>
    <nev>KunKicks</nev>
  </Uzlet>
```

Itt látható az XML dokumentumban az Üzlet egyed felépítése. Minden attribútum elkészült az ER és XDM modell alapján az Üzlet egyedhez. Itt az üzlet kódja, vagyis az ukod az “U001” értéket kapta és a továbbiakban is ezt használtam.

```
<Cipok ckod="C998" ukod="U001">
    <Keszlet>50</Keszlet>
    <Marka>Nike</Marka>
    <modell>Air Force</modell>
    <ar>45000</ar>
    <Meret>42</Meret>
    <Szin>Fehér</Szin>
```

```
</Cipok>
```

Itt látható a Cipo egyed felépítése. Itt is minden attribútum úgy lett kialakítva, ahogy az az ER és XDM modellen szerepelt. Jelenleg 4 darab cipő van az adatbázisban, az itt lévő kódrészlet pedig a második cipőhöz tartozik.
Miután elkészült a két legfontosabb egyed, elkészítettem a többi négy egyedet is.

```
<Raktar rkod="R001" ukod="U001">
    <kapacitas>5000</kapacitas>
    <muszak>Délelőtt</muszak>
    <vezeto>Kovács János</vezeto>
    <hely>3000</hely>
</Raktar>
```

Ez jelenleg a raktár kódja az XML fájlban. Be lett állítva egy rkod ami itt az “R001” értéket kapta. Ezen kívül be lett állítva még az ukod is, ami a fentebb említett “U001” értéket kapta.

```
<vasarlo vkod="V555" ukod="U001">
    <nev>Faragó René</nev>
    <cim>
        <varos>Muhi</varos>
        <utca>Fő utca</utca>
        <hazszam>3</hazszam>
    </cim>
    <eletkor>25</eletkor>
    <fizetesi_mod>
        <online>
            <applepay>false</applepay>
            <googlepay>true/googlepay>
        </online>
    </fizetesi_mod>
</vasarlo>
```

Ez itt egy vásárlónak az adatai a Vasarlo egyedben. Ez a vásárló a “V555” vásárlói kódot kapta és itt is szerepel az üzletkód vagyis az “U001”. Jól látható, hogy a fizetési mód is megfelelően lett

elkészítve, pontosan az ER és XDM modellek alapján. Ebben az esetben a vásárló online fizetést választott és a Google Pay opciót.

```
<rendeles rendeleskod="ORD888" ukod="U001">
    <osszeg>45000</osszeg>
    <cipo>C998</cipo>
  </rendeles>
```

A rendelés egyednek, noha nincs sok attribútuma, még egy fontos egyed. Itt a rendeléskód az “ORD888” és az üzletkód ugyanúgy “U001”.

```
<Kiszallitas kkod="K888" ukod="U001">
    <Kamion>
      <modell>Mercedes Sprinter</modell>
      <rendszam>ABC-123</rendszam>
    </Kamion>
    <Sofor>Kiss Béla</Sofor>
    <Telefonszam>+36209999999</Telefonszam>
    <Cim>
      <orszag>Magyarország</orszag>
      <varos>Muhi</varos>
      <utca>Fő utca</utca>
      <hazszam>3</hazszam>
    </Cim>
</Kiszallitas>
```

Végül itt látjató a kiszállítás egyedre egy példa. Itt a kiszállítási kódban szereplő szám, ami jelenleg “888” megegyezik a rendelésben szerepő számmal, ami szintén “888”. Ez a könnyebb azonosítás érdekében készült. A cím teljesen megyezeik azzal a címmel, ami az ehhez kapcsolódó vásárlónál szerepel.

1.4 Az XML dokumentum alapján XMLSchema készítése

Az elkészített XML Schema egy szigorúan típusos, moduláris felépítésű adatmodellt valósít meg, amely támogatja az adatok validálását, a redundancia csökkentését, és képes lekezelni a cipőbolt hálózat összetett, hierarchikus adatkapcsolatait. Az XML Schema-t is, mint ahogy eddig mindent, az előadáson és gyakorlati órákon bemutatott példák alapján készítettem el.

A séma nevesített komplex típusokat használ. Ahelyett, hogy mindent egyben ömlesztve írtam volna le, külön definiáltam a típusokat.

```
<xs:element name="CipoBoltAdatbazis">
    <xs:complexType>
      <xs:sequence>
       <xs:element name="Uzlet" type="UzletType" maxOccurs="unbounded"/>
        <xs:element name="Raktar" type="RaktarType" maxOccurs="unbounded"/>
        <xs:element name="Cipok" type="CipokType" maxOccurs="unbounded"/>
        <xs:element name="vasarlo" type="VasarloType" maxOccurs="unbounded"/>
        <xs:element name="rendeles" type="RendelesType" maxOccurs="unbounded"/>
        <xs:element name="Kiszallitas" type="KiszallitasType" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

Ez átláthatóbbá teszi a kódot, és lehetővé teszi a karbantartást. Ha később változik a "Vásárló" szerkezete, csak a VasarloType-ot kell módosítani.

A sémában látható a kód-újrafelhasználás elve. Példa: Létrehoztunk egy CimTypeSimple nevű típust. Ezt használja a Vásárló és az Üzlet is. Így nem kellett kétszer leírni ugyanazt a város-utca-házszám szerkezetet. Ez csökkenti a redundanciát.

```
<xs:complexType name="CimTypeSimple">
    <xs:sequence>
      <xs:element name="varos" type="xs:string"/>
      <xs:element name="utca" type="xs:string"/>
      <xs:element name="hazszam" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
```

A séma nem kezel mindent egyszerű szövegként, hanem szigorú adattípusokat kényszerít ki.

```
<xs:sequence>
      <xs:element name="osszeg" type="xs:integer"/>
      <xs:element name="cipo" type="xs:string"/>
</xs:sequence>
```

Ez biztosítja az adatintegritást, vagyis azt, hogy az adatok "tiszták" és feldolgozhatóak legyenek.

A séma szabályozza, hogy egy elemből hány darab fordulhat elő.

```
<xs:element name="Uzlet" type="UzletType" maxOccurs="unbounded"/>
<xs:element name="Raktar" type="RaktarType" maxOccurs="unbounded"/>
<xs:element name="Cipok" type="CipokType" maxOccurs="unbounded"/>
<xs:element name="vasarlo" type="VasarloType" maxOccurs="unbounded"/>
```

maxOccurs="unbounded": A gyökérelem alatt lévő csoportoknál (pl. Cipok, Rendeles) ez azt jelenti, hogy korlátlan számú cipő vagy rendelés lehet az adatbázisban.

```
<xs:element name="applepay" type="xs:string"minOccurs="0"/>

<xs:element name="googlepay" type="xs:string" minOccurs="0"/>
```

minOccurs="0": A fizetési módoknál (pl. online, applepay) látható. Ez azt jelenti, hogy ezek az adatok opcionálisak. Nem kötelező minden vásárlónak ApplePay-t használnia, így az elem elhagyható.

Bár az XML hierarchikus, ez a modell relációs adatbázis-szerű kapcsolatokat valósít meg attribútumokon keresztül. Minden fő elemnek van egy egyedi azonosítója (pl. ukod - Üzlet kód, ckod - Cipő kód).

```
<xs:attribute name="rkod" type="xs:string" use="required"/>
<xs:attribute name="ukod" type="xs:string"/>
```

A use="required" attribútum biztosítja, hogy ezeket az azonosítókat kötelező megadni, így minden rekord egyértelműen azonosítható. Látható, hogy az ukod (Üzlet kód) szinte mindenhol megjelenik "idegen kulcsként", ami azt jelzi, hogy melyik üzlethez tartozik az adott raktár vagy rendelés.

2. DomParse feladatok

2.1 Adatolvasás

Ahhoz, hogy az adatolvasás működjön, vagyis a DomRead, először is importálni kell a megfelelő könyvtárakat a programhoz.

```
import java.io.File;
import java.io.PrintWriter;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.Document;
import org.w3c.dom.NamedNodeMap;
```

```
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
```

Ezek a könyvtárak, az XML kezelését, az XML beolvasását, illetve a feladat végén az eredmény fájlba írását teszik lehetővé.
Ezután be kell olvasni az adott XML fájlt, valamint inicializálni kell azt a fájlt, amelybe az eredményt szeretnénk majd, hogy kerüljön. Erre az alábbi kódrész biztosít lehetőséget:

```
File xmlFile = new File("GEB15IXML.xml");
        writer = new PrintWriter("GEB15IXML.txt", "UTF-8");
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        DocumentBuilder dBuilder = factory.newDocumentBuilder();
        Document doc = dBuilder.parse(xmlFile);
        doc.getDocumentElement().normalize();
```

Miután beolvastuk a fájlt, ki is kell írni azt a konzolra. Ezt a következő kódrészlet teszi lehetővé:

```
NodeList rootChildren = doc.getDocumentElement().getChildNodes();
        for (int i = 0; i < rootChildren.getLength(); i++) {
           Node node = rootChildren.item(i);
           if (node.getNodeType() == Node.ELEMENT_NODE) {
              writeToBoth("---------------------------------------", writer);
              writeToBoth(node.getNodeName().toUpperCase(), writer);
              writeToBoth("---------------------------------------", writer);
              printCleanData(node, "", writer);
              writeToBoth("", writer); // Üres sor
           }
        }
```

Ez a kód először is lekéri az XML-ből a gyerek node-okat, amelynek a számát beteszi egy for ciklusba. Ebben a ciklusban egy if feltétel ellenőrzi, hogy node-ok nevét, majd pedig ki írja a konzolra és a fájlba. A kódsor végén még rekurzió is történik, ugyanis a kód, újra meghívja magát, hogy minden adatot kiolvasson az XML-ből.
Azonban a tényleg munka nem itt történik, hanem egy másik osztályban, ugyanis objektum orientáltan lett elkészítve a kód.

```
   private static void printCleanData(Node node, String indent, PrintWriter writer) {
```

```
    if (node.hasAttributes()) {
        NamedNodeMap attributes = node.getAttributes();
        for (int j = 0; j < attributes.getLength(); j++) {
            Node attr = attributes.item(j);
            writeToBoth(indent + attr.getNodeName() + ": " + attr.getNodeValue(), writer);
        }
    }
    NodeList children = node.getChildNodes();

    for (int i = 0; i < children.getLength(); i++) {
        Node child = children.item(i);

        if (child.getNodeType() == Node.ELEMENT_NODE) {
            if (hasChildElements(child)) {
                writeToBoth(indent + child.getNodeName() + ":", writer);
                printCleanData(child, indent + "  ", writer);
            } else {
                String text = child.getTextContent().trim();
                writeToBoth(indent + child.getNodeName() + ": " + text, writer);
            }
        }
```

A kód ezen része ellenőrzi le, hogy az adott node-nak vannak-e tulajdonságai és ha vannak, akkor azokat kiírja és eltárolja. Ezután pedig tovább is adja az adokat a kód további részeinek a folyamatokhoz.

```
private static void writeToBoth(String text, PrintWriter writer) {
    System.out.println(text); // Konzol
    if (writer != null) {
        writer.println(text); // Fájl
    }
}
```

Az alábbi segédosztály felel azért pedig, hogy egy sor kóddal egyszerre tudjunk a konzolra is kiírni, illetve a fájlba is írni.

A teljes kódot tartalmazó fájl itt érhető el: linket ide

2.2 Adatlekérdezés

Az adatlekérdezésnek is a legfontosabb lépése, a megfelelő könyvtárak importálása. Az alábbi könyvtárakat importáltam, hogy a kód megfelelően működjön, tudja kezelni az XML fájlokat és hogy a lekérdezések működjenek.

```
import java.io.File;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
```

Ezt követően kell betölteni magát az XML fájlt. Ennek a megvalósítása nem egy komplikált folyamat, csupán az alábbi néhány sor szükséges hozzá:

```
File xmlFile = new File("GEB15IXML.xml");
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        DocumentBuilder dBuilder = factory.newDocumentBuilder();
        Document doc = dBuilder.parse(xmlFile);
        doc.getDocumentElement().normalize();
```

Ezután következik az első lekérdezés. Én, az első lekérdezésemnek azt választottam, hogy “Mikor van nyitva az üzlet?”.

```
System.out.println("\n1. Mikor van nyitva az üzlet?");

        // Megkeressük a 'nyitvatartas' elemeket (feltételezzük, hogy 1 van)
        NodeList nyitvaList = doc.getElementsByTagName("nyitvatartas");

        if (nyitvaList.getLength() > 0) {
           Element nyitvaElem = (Element) nyitvaList.item(0);

           String tol = nyitvaElem.getElementsByTagName("tol").item(0).getTextContent();
           String ig = nyitvaElem.getElementsByTagName("ig").item(0).getTextContent();

           System.out.println("   Nyitvatartás: " + tol + " - " + ig);
        }
```

Először is a kód, kiírja a konzolra a kérdést, a jobb átláthatóság érdekében. Ezután a “nyitvaList” nevű változóba eltárolásra kerül minden olyan node, amely tartalmazza, azt, hogy “nyitvatartas”. Ezt követően pedig egy szimpla, if feltétellel ellenőrizve van, hogy van-e ilyen node, majd pedig lekéri az XML fájlból a node-hoz tartozó adatokat, ami itt a “tol” és “ig”. A következő sorban pedig ezeket az adatokat kiírja a konzolra.

A következő lekérdezésemnek azt választottam, hogy “Milyen cipő van jelenleg?”. Ennek megvalósítása nagyon hasonlóan történik az első lekérdezéshez.

```
NodeList cipoList = doc.getElementsByTagName("Cipok");

        for (int i = 0; i < cipoList.getLength(); i++) {
            Node node = cipoList.item(i);
            if (node.getNodeType() == Node.ELEMENT_NODE) {
                Element elem = (Element) node;

                String marka = elem.getElementsByTagName("Marka").item(0).getTextContent();
                String modell = elem.getElementsByTagName("modell").item(0).getTextContent();
                String ar = elem.getElementsByTagName("ar").item(0).getTextContent();

                System.out.println("----------------------------");
                System.out.println("   Márka: " + marka);
                System.out.println("   Modell: " + modell);
                System.out.println("   Ár: " + ar + " Ft");
            }
        }
```

Itt ugyanúgy, létre van hozva egy list, ami itt a “cipoList” névre hallgat, ami eltárolja a “Cipok” névvel ellátott attribútumokat. Ezt követően egy if ciklus ellenőrzi, hogy van-e bármennyi ilyen attribútum-e. Abban az esetben, ha van, akkor pedig lekéri az XML fájlból az adatokat és formázottan kiírja a konzolra.

Mivel a harmadik lekérdezés felépítése is ilyen, ezért következőnek a negyedik lekérdezést említeném meg. Itt a lekérdezés az volt, hogy “Van-e Faragó René nevű vásárló?”. Itt a lekérdezés formája, ugyanúgy megegyezik a többivel, azonban itt a végén van egy csavar. Mivel a lekérdezésem az az, hogy van-e ilyen nevű vásárló, ezért a kódtól el van várva, hogy egy “Igen” van egy “Nem” választ adjon.

Ez a következő képpen van megvalósítva:

```
String aktualisNev = elem.getElementsByTagName("nev").item(0).getTextContent();

            // Összehasonlítjuk
            if (aktualisNev.equals(keresettNev)) {
                megvan = true;
                // Ha megvan, kiléphetünk a ciklusból (break), nem kell tovább keresni
                break;
            }
        }
    }

    if (megvan) {
        System.out.println("   Igen, van ilyen nevű vásárló!");
    } else {
        System.out.println("   Nincs " + keresettNev + " nevű vásárló az adatbázisban.");
```

A kód, mielőtt még befejezné a lekérdezést, egy if statement használatával ellenőrzi, hogy az "aktualisNev" és a "keresettNev" megegyezik-e. Ha nem, akkor egy negatív választ ad, hogy egyezik, akkor pedig egy pozitív választ ad.
A teljes kódot tartalmazó fájl itt érhető el: linket

2.3 Adatmódosítás

Ennél a feladatnál is a legfontosabb, hogy a megfelelő könyvtárakat importáljuk. Ehhez az adatmódosítás feladathoz az alábbi könyvtárakat használtam:

```
import java.io.File;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
```

```
import org.w3c.dom.NodeList;
```

Ezek a könyvtárak tartalmaznak minden metódust, ami XML beolvasására, kezelésére, módosítására és mentésére kell.

Legelső lépésként beolvastam az XML fájlt, hogy azt tudjam használni. Ezt az alábbi módon tettem meg:

```
File inputFile = new File("GEB15IXML.xml");

        DocumentBuilderFactory docFactory = DocumentBuilderFactory.newInstance();
        DocumentBuilder docBuilder = docFactory.newDocumentBuilder();

        Document doc = docBuilder.parse(inputFile);
```

Az első módosításomnak azt választottam, hogy a legelső cipőnek az adatit módosítsa át a kód egy másik cipő adataira. Ehhez az alábbi kódot használtam:

```
Node cipok = doc.getElementsByTagName("Cipok").item(0);
        NodeList list = cipok.getChildNodes();

        for (int temp = 0; temp < list.getLength(); temp++) {
           Node node = list.item(temp);
           if (node.getNodeType() == Node.ELEMENT_NODE) {
              Element eElement = (Element) node;
              if ("Marka".equals(eElement.getNodeName())) eElement.setTextContent("Nike");
              if  ("modell".equals(eElement.getNodeName()))  eElement.setTextContent("Air  Force
Supreme");
              if ("Szin".equals(eElement.getNodeName())) eElement.setTextContent("fehér");
              if ("Meret".equals(eElement.getNodeName())) eElement.setTextContent("42");
              if ("Keszlet".equals(eElement.getNodeName())) eElement.setTextContent("67");
           }
        }
```

Először a kód eltárolja a “cipok” változóban azt a node-ot, aminek a neve “Cipok” és ebből is kizárólag azt, amelynek az indexe nulla. Ezt követően egy “list” nevű változóban el vannak tárolva a child node-ok. Egy for ciklus ezután végig megy ezeken a child node-okon és a cikluson belül egy if statement ellenőrzi, hogy valósak-e a node-ok. Ha igen, akkor pedig if állítások sorozata következik, amelyek ellenörzik, hogy a “Cipok” node attribútuma, az a megfelelő értékekkel

rendelkezik-e, ahhoz, hogy a módosítás végrehajtódjon. Ha megfelelőek, akkor pedig kicseréli őket, a kívánt értékre.

Mivel a többi lekérdezés is ugyanilyen alapokra épül, ezért a fájlba írást szeretném kifejteni. Ennek megvalósításáért az alábbi kódrészlet felel:

```
TransformerFactory transformerFactory = TransformerFactory.newInstance();
        Transformer transformer = transformerFactory.newTransformer();
        DOMSource source = new DOMSource(doc);
        StreamResult result = new StreamResult(new File("GEB15IXML1.xml"));
        transformer.transform(source, result);
        System.out.println("Minden módosítás sikeresen végrehajtva!");
        System.out.println("Eredmény mentve: GEB15IXML1.xml");
```

Az egész művelet legfontosabb része a TransformerFactory illetve a StreamResult. Ez a két metódus felel a fájlba írásért, illetve azért, hogy a kimeneti fájl, az XML formátumban legyen. Egy másik fontos lépés még, hogy a fájl neve be legyen állítva, amit itt “GEB15IXML1.xml” lett.

A teljes kódot tartalmazó fájl itt érhető el: linket ide