JEGYZŐKÖNYV

Webes adatkezelő környezetek

Féléves feladat

Egy mozi adatkezelő rendszere

Készítette: Losonczi Dominik

Neptunkód: HZ60V1

Dátum: 2025. november

Miskolc, 2025

Tartalomjegyzék

1. Bevezetés

Ez a feladat a "Webes adatkezelő környezetek" nevű tárgyhoz készült féléves feladatként. A webes adatkezelés napjainkban kiemelten fontos, mivel az adatok biztonságos tárolása és feldolgozása alapvető követelmény. Erre a feladatra több nyelv is készült, mint például a GML(Generalized Markup Language) vagy az SGML(Standard Generalized Markup Language), azonban ezek számos ok miatt nem tudtak elterjedni. Erre a feladatnak 1998-ban az XML egy az elődeihez képest egyszerűbb és univerzálisabb nyelv, ami webbarát, így a HTML adatkezelésével közkeletűvé vált.

2. Feladat leírása

Feladatnak egy mozi adatkezelő rendszerét választottam. Azért választottam ezt, mivel a mozik sokrétű és komplex adatkezeléssel rendelkeznek. Nekem csak az alapvető és legkarakterisztikusabb aspektusaival kellett foglalkoznom azért, hogy feldolgozható és bemutatható adatszerkezetet tervezhessek.

A feladatok első felében alapvető adatmodell ábrázolását kellett megvalósítani, mint az ER és az XDM. Ezek segítségével egy XML dokumentumot kellett szerkeszteni, amihez pedig egy XMLSchema dokumentumot társítottam, ezt külső, interneten megtalálható validátorral sikeresen "valid"-nak minősítettem. A feladatok második felében pedig a java-s adatszerkezet létrehozását DOM API segítségével végeztem, ami egyrészről feldolgozta az XML dokumentum adatainak kinyeréséből, másrészt annak konzolra történő kiírásából, esetlegesen adatok megváltoztatásból, amiket egy új file-ba kellett kiírni.

3. Alapvető adatszerkezeti modellek és fileok

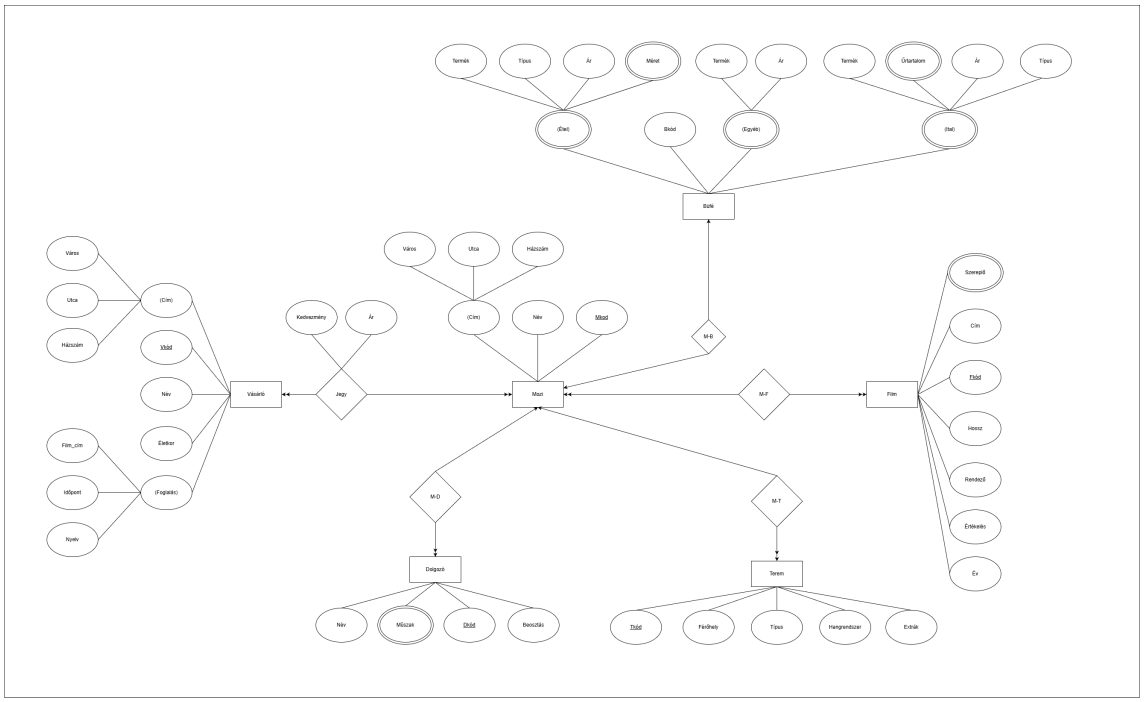
3.1. Az adatbázis ER modell tervezése

Az adatmodellben a "mozi" lett a fő elem, amihez minden más egyed idegenkulccsal csatlakozik. Ennek az egyednek a kulcsa az Mkód, értékei "m?" ahol a "?" bármely szám karakternek feleltethető meg. A tulajdonságai a név (string) és cím, amely a város (string), utca (string) és házszám-ból (integer) áll. A többi egyed a vásárló, dolgozó, terem, film és büfé, mind kapcsolatban állnak a mozival és a vásárlónak még egy olyan kapcsolata is van, aminek tulajdonságai vannak. Ügyeltem a feladat kiíráshoz igazodóan minden kapcsolat típust feltüntetni (1:1, 1:N, N:M), valamint minden tulajdonság típust is használni (normál, kulcs, többértékű, összetett).

A feladatban próbáltam bonyolultabb szerkezeteket is létrehozni, mint a Büfé, aminek 3 tulajdonsága is egyszerre többértékű és összetett. Ennek az adatszerkezete egy egész

feladatéval is felérhetne, viszont ezért maximalizáltam a tulajdonságok számát 4-ben, így még elég részletes és megfelelően kompakt a feladat teljesítéséhez.

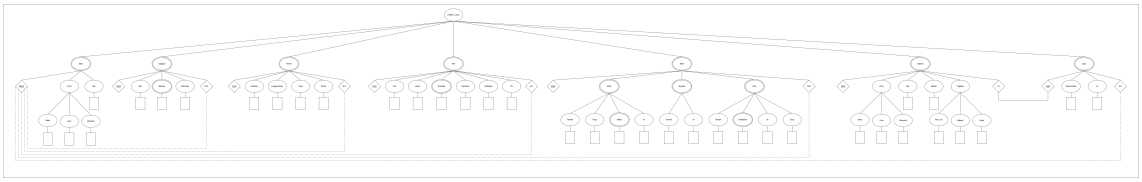
A másik bonyolultabb adatszerkezet a vásárló, és a mozi és vásárló közötti kapcsolat a jegy, ami saját tulajdonsággal rendelkezik, mégpedig kedvezmény (string) ár (integer) tulajdonságokkal.



1. ábra. A mozi ER modellje

3.2. Az adatbázis konvertálása XDM modellre

Ebben a feladatban a már meglévő ER modellt kellett átalakítani egy XDM modellre. Az átalakításhoz létrehoztam egy mesterséges gyökeret "HZ60V1_mozi" néven, és ennek a gyerek egyedei lettek a "mozi", a "dolgozó", a "terem", a "film", a "büfé", a "vásárló" és a jegy. Fontos megjegyezni, hogy itt a kapcsolatokat máshogy kell jelezni, mint az ER modellnél, mivel ott entitások között van kapcsolat az XDM modellnél pedig PK-FK kapcsolat van. Az XDM-nél minden tulajdonságból gyerek elem, a kulcs tulajdonságból pedig attribútum lesz. A modell alapvető felépítése a séma helyett hierarchia.



2. ábra. A mozi XMD modellje

3.3. Az XDM modell alapján XML dokumentum készítése

Az XML struktúrája az XDM szerkezete alapján készült. A gyökérelem "<HZ60V1_mozi>", ez tartalmazza gyerekelemként az összes többi entitást (pl.: <mozi>, <dolgozó>, <terem>), ezeknek saját egyedi azonosító attribútumok van (Mkod, Dkod, Tkod), amik itt már tényleges kulcs értéket kaptak. A kódban található többértékű tulajdonság amik több értékkel is rendelkeznek

Listing 1. A mozi adatszerkezet részlete

```
<büfé Bkod="b1" M_B="m1">
        <étel>
            <termék>Popcorn</termék>
            <típus>Sós</típus>
            <ár>1400</ár>
            <méret>Kicsi</méret>
            <méret>Közepes</méret>
            <méret>Nagy</méret>
        </étel>
        <étel>
            <termék>Popcorn</termék>
            <típus>Vajas</típus>
            <ár>1600</ár>
            <méret>Kicsi</méret>
            <méret>Közepes</méret>
            <méret>Nagy</méret>
        </étel>
        </büfé>
```

Ebben a részletben egy adott büfé több ételt is tárol, és azon belül több méretben is kaphatóak az adott termékek.

3.4. Az XML dokumentum alapján XMLSchema készítése

Az XMLSchema az XML dokumentum alapján készült. Az XSD 3 fő elemre bontható kulcsok, Idegen kulcsok és koplex típusok. A kulcsok részben minden kulcs ami megtalálható az XML fileban fel lett véve. Az idegen kulcs részben felvetettem az idegen kulcsokat és azokat össze kötöttem az elsődleges kulcsokkal. A harmadik részben pedig komplex típusokat hoztam létre. Minden fő elem kapott magának egy külön típust, valamint az összetett elemek is kaptak külön típusokat. A komplex típusok elemekből és ha rendelkeznek vele, akkor attribútumokból állnak. Az elemek a komplex típus gyerek elemit foglalják magukba, amiknek szintén van külön típusaik, akár még több komplex típus.

Listing 2. Az XSD modell részlete

```
<xs:complexType name="vásárlóTipus">
                <xs:sequence>
            <xs:element name="cím" type="címTipus"></xs:element>
                <xs:element name="név" type="xs:string"></
                   xs:element>
            <xs:element name="életkor" type="xs:int"></
               xs:element
                <xs:element name="foglalás" type="foglalásTipus"
                   ></xs:element>
        </xs:sequence>
        <xs:attribute name="Vkod" type="xs:ID" use="required"></
           xs:attribute>
        <xs:attribute name="J_V" type="xs:IDREF" use="required">
           </xs:attribute>
        </xs:complexType>
```

4. Java adatszerkezet megvalósítása DOM API segítségével

4.1. Adatolvasás

A program célja, hogy XML dokumentumból Java objektum-szerűen kinyerje és megjelenítse az adatokat. A feldolgozás az org.w3c.dom könyvtár DOM API-ján keresztül történik, amely az XML fájlokat fa-szerkezetként kezeli, így minden elem és attribútum könnyen elérhető.

Listing 3. Javaban dokumentum létrehozás

```
File xmlFile = new File("HZ60V1_XML.xml");
DocumentBuilderFactory factory = DocumentBuilderFactory.
   newInstance();
DocumentBuilder dBuilder = factory.newDocumentBuilder();
Document doc = dBuilder.parse(xmlFile);
doc.getDocumentElement().normalize();
```

Ez a rész hozza létre magát a DOM objektumot

Listing 4. Javaban mozi objektum létrehozás

```
NodeList mozik = doc.getElementsByTagName("mozi");
for (int i = 0; i < mozik.getLength(); i++) {
    Element mozi = (Element) mozik.item(i);
    System.out.println("Mozi kód: " + mozi.getAttribute("Mkod"))
       ;
    System.out.println("Név: " + mozi.getElementsByTagName("név"
       ).item(0).getTextContent());

    Element cim = (Element) mozi.getElementsByTagName("cím").
       item(0);
    System.out.println("  Város: " + cim.getElementsByTagName("v
       áros").item(0).getTextContent());
    System.out.println("  Utca: " + cim.getElementsByTagName("
       utca").item(0).getTextContent());
}
```

Ez a kódrészlet egy adott ág kezelését írja le. A program feldogozza a mozi elemeket és attribútumokat, majd külön kiírja azokat a konzolra.

4.2. Adat-lekérdezés

Ez a program a DomRead-el szemben csak szimplán kiírja az adatokat, hanem emberileg olvasható és feldolgozható formában és mennyiségben írja ki azt.

Listing 5. Javaban rendezett adatkiírás

```
NodeList mozik = doc.getElementsByTagName("mozi");
for (int i = 0; i < mozik.getLength(); i++) {
    Element mozi = (Element) mozik.item(i);
    String mkod = mozi.getAttribute("Mkod");
    String nev = mozi.getElementsByTagName("név").item(0).
       getTextContent().trim();
    Element cim = (Element) mozi.getElementsByTagName("cím").
       item(0);
    String varos = cim.getElementsByTagName("város").item(0).
       getTextContent();
    String utca = cim.getElementsByTagName("utca").item(0).
       getTextContent();
    String hazszam = cim.getElementsByTagName("házszám").item(0)
       .getTextContent();
    System.out.println("  [" + mkod + "] " + nev + " - " + varos
        + ", " + utca + " " + hazszam);
}
```

Ebben a kódrészletben először kinyer a program minden hasznos elementet és attribútumot, majd azt egy kompakt módon olvashatóan kiírja.

4.3. Adatmódosítás

Egy DomModify program alapvető feladata az adott filerendszerben valamilyen változtatást végrehajtani és egy új fileba elmenteni azokat a változtatásokat.

Listing 6. Javaban adat változtatás

```
Node mozi = doc.getElementsByTagName("mozi").item(0);
Element moziElem = (Element) mozi;
Node nevNode = moziElem.getElementsByTagName("név").item(0);
nevNode.setTextContent("Miskolc Egyetem Mozi");
```

Ebben a kódrészletben a mozi nevének módosítása látható, előszőször elmenti az adatot, majd megváltoztatja. Itt a legelső példány neve módosul "Miskolc Egyetem Mozi"-ra.

Listing 7. Javaban file-ba mentés

```
TransformerFactory transformerFactory = TransformerFactory.
   newInstance();
Transformer transformer = transformerFactory.newTransformer();
transformer.setOutputProperty(OutputKeys.ENCODING, "UTF-8");
transformer.setOutputProperty(OutputKeys.INDENT, "yes");
transformer.setOutputProperty("{http://xml.apache.org/xslt}
   indent-amount", "2");

DOMSource source = new DOMSource(doc);
StreamResult consoleResult = new StreamResult(System.out);
transformer.transform(source, consoleResult);

StreamResult fileResult = new StreamResult(new File("
   HZ60V1_mozi_MOD.xml"));
transformer.transform(source, fileResult);
```

A kód egy "transformer" nevezetű objektum segítségével menti el a megváltoztatott adatokat. A programban a "setOutputProperty()" segítségével állítjuk be a kimeneti file formátumát.