JEGYZŐKÖNYV

Adatkezelés XML környezetben

Féléves feladat

Emberek Az Iskolában

Készítette: **Pogonyi Ábel Kürt**

Neptunkód: **TR6FKP**

Dátum: **2024.11.12.**

Tartalomjegyzék

Bevezetés 3

Feladat leírás 3

1. XML Modell 3

1.1 Az adatbázis ER modell tervezése 3

1.2 Az adatbázis konvertálása XDM modellre 4

1.3 Az XDM modell alapján XML dokumentum készítése 4

1.4 Az XML dokumentum alapján XMLSchema készítése 4

2. Java feldolgozás 5

2.1 Adatolvasás 5

2.2 Adatírás 6

2.3 Adatlekérdezés 6

2.4 Adatmódosítás 7

Bevezetés

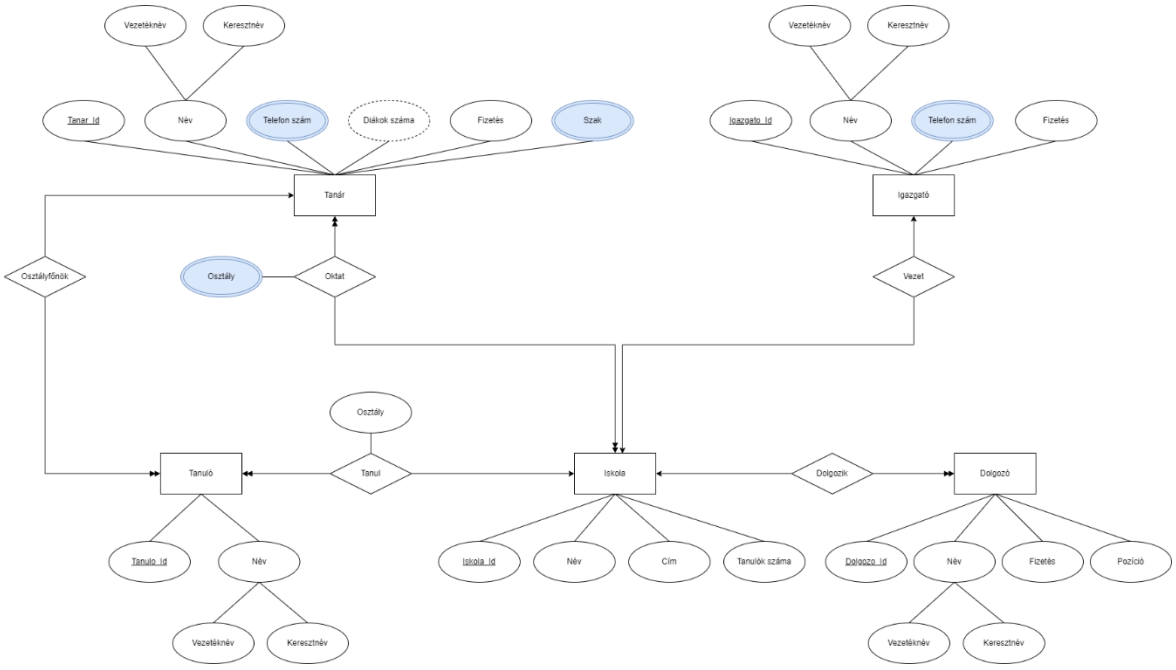
Én egy iskola nyilvántartási rendszert találtam ki, ahol fel lehet venni iskolák alapvető adatait és a személyeket, akik az iskolába járnak valami oknál kifolyólag. A modell nem teljes körű és csak egy minta megvalósítás, ami nagyon sok mindenre nem tér ki.

Feladat leírás

A modell fő célpontja az iskolában valamilyen módon járó emberek. Őket 4 csoportra bontottam: Igazgatók, Dolgozók, Tanárok és Diákok. Minden csoport és az iskola külön, külön egy-egy egyed. A modell leírja az egyes csoportok közötti kapcsolatot, illetve információkat tud tárolni egyes példányokról, ezzel megkülönböztetve őket. Az iskolának van címe és neve. Az embereknek van nevük. Akik munkát végeznek van fizetésük. Akik szellemi munkát végeznek lehet telefonszámuk. A tanároknak vannak osztályai és szakjaik. A diákoknak meg van egy osztályuk és osztályfőnökük.

1. XML Modell

1.1 Az adatbázis ER modell tervezése

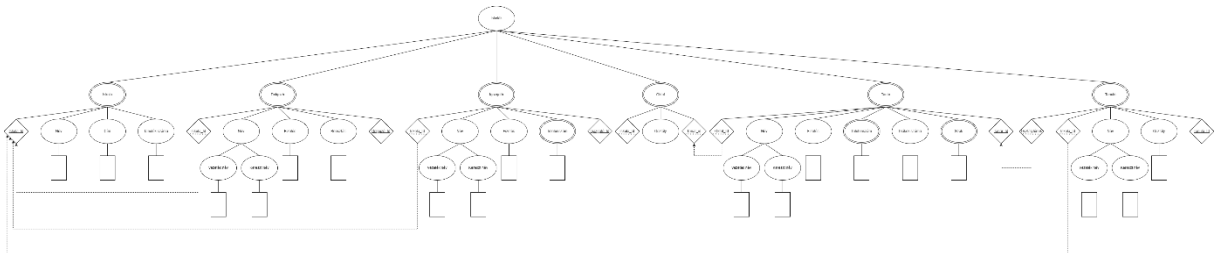


1. ábra ER modell

Minden egyed az iskolához köthető. Egy iskolának egy igazgatója van, ezért a köztük lévő kapcsolat típusa 1:1. A diák és a dolgozó is csak egy iskolában járhat, de egy iskolába több tanuló és dolgozó is van, ezért az ő kapcsolatuk az iskolával N:1. Egy tanár lehet több tanulónak is az osztályfőnöke, de egy diáknak csak egy osztályfőnöke lehet ebben a modellben, ezért a

tanár diák kapcsolat 1:N. Jelenlegi válságos helyzetben egy tanár több iskolában is taníthat és egy iskolában több tanár is van, ezért az ő kapcsolatuk N:M. A telefonszám, a szak és a tanár osztályai azok a mezők, ahol több adat is előfordulhat, ezen kívül az összes többi adat vagy egyszerű, vagy egyszerűen összetett.

1.2 Az adatbázis konvertálása XDM modellre



2. ábra XDM modell

Az XDM modell nagyban hasonlít az ER modellre. Legnagyobb különbsége abban mutatkozik, hogy a tanár-iskola N:M kapcsolatot egy újabb egyedben definiáljuk, az oktat egyedben. Ez az egyed tartalmazza az iskola és tanár idegen kulcsát, illetve azt az osztályt, amelyiket oktatja. Ezen kívül a modell már tartalmazza az idegen kulcsokat is.

1.3 Az XDM modell alapján XML dokumentum készítése

Az XML dokumentum elkészítése igazán nem volt nehéz az XDM modell alapján. Példányokat kellet létrehozni és őket feltölteni valamilyen adatokkal. Sajnos nincs ilyen irányú kreativitásom, ezért a teszt adatokból hiányzik a kreativitás szikrája, de a célnak megfelelőek. Az adatok létrehozásánál figyelni kellet a megszorításokra, például a telefonszám formátumára, illetve arra, hogy melyik elemből hány fordulhat elő. Itt a telefonszám és a szakok voltak azok, ahol több példány is előfordulhat, illetve akár egyetlen egy se. Továbbá arra, hogy az adatokat milyen formátumban és milyen mennyiséget vesznek fel, mert az XMLSchemában ezeket le kell írni.

1.4 Az XML dokumentum alapján XMLSchema készítése

Ez a része már érdekesebb volt. Minden egyed saját típust kapott. Bár a feladat leírásban benne volt, hogy legyen minimum három példány, én mégsem raktam bele ezt a megszorítást, ezért egy-egy egyedből korlátlan példány lehet és minimum egy. Azután a kulcsok és idegen kulcsok lettek definiálva:

```
1. <xs:key name="iskola_kulcs">
2.             <xs:selector xpath="iskola"/>
3.             <xs:field xpath="@iskola_id"/>
4. </xs:key>
```

Külön típust kapott az emberek neve, amely tartalmazza a vezeték és kereszt nevüket:

```
1. <xs:complexType name="nevTipus">
2.     <xs:sequence>
3.         <xs:element name="vezeteknev" type="xs:string"/>
4.         <xs:element name="keresztnev" type="xs:string"/>
5.     </xs:sequence>
6. </xs:complexType>
7.
```

Illetve a telefonszám, amin egy regex megszorítás van:

```
1. <xs:simpleType name="telefonszamTipus">
2.     <xs:restriction base="xs:string">
3.        <xs:pattern value="\+36/\d{2} \d{3}-\d{4}"/>
4.     </xs:restriction>
5. </xs:simpleType>
```

Ezeken kívül még a fizetésre volt egy megszorítás, hogy nullától nem lehet kisebb:

```
1. <xs:simpleType name="fizetesTipus">
2.     <xs:restriction base="xs:integer">
3.         <xs:minInclusive value="0"/>
4.     </xs:restriction>
5. </xs:simpleType>
6.
```

Nagyjából ezekből a típusokból épülnek fel az egyedek, attribútumokkal kiegészítve. Továbbá sorrend és számosság megszorítások vannak még benne.

2. Java feldolgozás

2.1 Adatolvasás

Az adatolvasás eléggé bruteforce metódussal készült, mivel ismerjük az XML és XMLSchema dokumentumokat, ezért ismerjük a felépítést és nem kell dinamikusan lekérdeznünk, ehelyett használhatunk konstans lekérdezéseket. A szerkezet ugyan az minden egyednél. Lekérdezzük az összes példányt ugyanazzal a taggel. Majd ezt a listát feldolgozzuk. A feldolgozás menete is ugyan az, lekérjük az attribútumokat és a gyerek elemeket, majd kiírjuk konzolra a kinyert adatokat. ennél a feladatnál az érdekesebb része az a kivételkezelés, például, ha egy elem nem kötelező, akkor lehet, hogy nincs benne ezért kell null kezelés. Emellett a lista kezelés, ha egy elem többször is előfordulhat, azt hogyan kell kezelni. Ilyen elemek a telefonszámok, a szakok és az Oktatban az osztályok. Őket egy külön metódus íratja ki, ami megkapja a NodeListet és azon végig iterálva készít egyetlen stringet, amit majd ki tudok íratni:

```
1. static String getListString(NodeList list, String name){
2.     String result = "";
3.     for (int i = 0; i < list.getLength(); i++){
4.         result += name + list.item(i).getTextContent() + "\n";
5.     }
6.     return result;
7. }
```

Ezen kívül még a név kiíratására csináltam egy külön metódust mivel az több egyedben is előfordul:

```
1. static String getName(Node nevNode){
2.     Element nev = (Element) nevNode;
3.     Node node1 = nev.getElementsByTagName("keresztnev").item(0);
4.     String keresztnev = node1.getTextContent();
5.     Node node2 = nev.getElementsByTagName("vezeteknev").item(0);
6.     String vezeteknev = node2.getTextContent();
7.     return "Név vezetéknév: " + vezeteknev + "\nNév keresztnév: " + keresztnev;
8. }
9.
```

2.2 Adatírás

Az adatírás az adatolvasáshoz hasonlóan egy igazán céltudatos. 3 lépésből áll egy példány létrehozása.

1. Attribútumok megadása
2. Gyerek elemek létrehozása
3. Gyerek elemek feltöltése adattal

Gyerek elem létrehozása szöveg tartalommal:

```
1. private static Node createTextElement(Document doc, String name, String value){
2.     Element node = doc.createElement(name);
3.     node.appendChild(doc.createTextNode(value));
4.     return node;
5. }
```

Többször előforduló elemek létrehozása:

```
1.     private static void appendArray(Document doc, Element parent,String name,
String[] array){
2.         for (String element : array) {
3.             parent.appendChild(createTextElement(doc, name, element));
4.         }
5.     }
```

Ezután már csak a megfelelő tag nevekkel és adatokkal létrehozzuk a különböző példányokat, amiket hozzáadunk a gyökérelemhez.

2.3 Adatlekérdezés

Az érdekesebb része itt kezdődött a feladatnak. 4 lekérdezést csináltam:

1. Az iskolák nevei és az igazgatói.

   Elsőnek lekérdeztem az összes iskolát majd azok adatait kigyűjtöttem, majd lekérdeztem az összes igazgatót és abból kiválasztottam azt amelyiknek az iskola idegen kulcsa megegyezett az iskola kulcsával. Ezt megcsináltam az összes iskolára.

   Ebben ez a segédfüggvény segített, ami a tagnév és id-k alapján visszaadja az egyezést:

```
 1. public static Node matchId(Document doc,String tagName, String idName, String id){
 2.     NodeList nodes = doc.getElementsByTagName(tagName);
 3.     for (int i = 0; i < nodes.getLength(); i++) {
 4.         Node node = nodes.item(i);
 5.         if (node.getNodeType() == Node.ELEMENT_NODE) {
 6.             Element elem = (Element) node;
 7.             if (elem.getAttribute(idName).equals(id)) {
 8.                 return node;
 9.             }
10.         }
11.     }
12.     return null;
13. }
```

2. Egy tanár és az általa tanított tantárgyak.

   Ez egy egyszerű lekérdezés volt, ez a többszörösen előfordulható elemek kiírását célozza meg. Lekérdeztem a tanárt, majd kinyertem belőle az adatokat és megjelenítettem a konzolon.

```
1. System.out.println("Tanár neve: " + getFullName(tanar) + ", tantárgyak: " +
String.join(", ", szakok));
```

3. Egy tanárhoz tartozó diákok

   Ez az első két feladat kombinációja, elsőnek lekérdeztem a tanárt, majd az összes hozzátartozó diákot, utána megjelenítettem konzolon. Itt már az előbb említett id keresés bővített változatát használom, ahol már egy NodeList-et ad vissza minden egyező elemmel.

```
 1. public static List<Node> matchIdArray(Document doc,String tagName, String idName,
String id){
 2.     NodeList nodes = doc.getElementsByTagName(tagName);
 3.     List<Node> resultList = new ArrayList<>();
 4.     for (int i = 0; i < nodes.getLength(); i++) {
 5.         Node node = nodes.item(i);
 6.         if (node.getNodeType() == Node.ELEMENT_NODE) {
 7.             Element elem = (Element) node;
 8.             if (elem.getAttribute(idName).equals(id)) {
 9.                 resultList.add(node);
10.             }
11.         }
12.     }
13.     return resultList;
14. }
```

4. Egy iskolában lévő összes ember

   Itt már szinte semmi újdonság nincsen az id keresés alapján megkeresem az összes embert, aki az adott iskolához tartozik.

2.4 Adatmódosítás

Itt is 4 feladatot csináltam. Viszont itt nem volt a fájl elmentve, ezért a változások elvesztek a program futásának végén, de minden feladat után a konzolon megjelenítem az XML dokumentum jelenlegi állapottát, ezzel a metódussal:

```
 1. public static void printXML(Document doc) throws Exception{
 2.     TransformerFactory transformerFactory = TransformerFactory.newInstance();
 3.     Transformer transformer = transformerFactory.newTransformer(new StreamSource(new
File("./XMLTaskTR6FKP/DOMParseTR6FKP/pretty.xsl")));
 4.
 5.     DOMSource source = new DOMSource(doc);
 6.     StreamResult console = new StreamResult(System.console().writer());
 7.
 8.     transformer.transform(source, console);
 9. }
```

Ez a metódus egy xsl fájlt használ a megjelenítés megformálásához. Az xsl fájlt ChatGPT írta még azelőtt, hogy tanultuk volna.

1. Dolgozó áthelyezése más iskolába.

   Ez nem volt nehéz csak az iskola idegenkulcsát kellet megváltoztatni.

2. Tanár hozzáadása egy iskolához

   Itt létre kellet hozni egy új ’oktat’ egyedet, azt feltölteni a megfelelő adatokkal és jó helyre beszúrni. Itt megkerestem az első oktat elemet és elé szúrtam be, hogy egy helyen legyenek az azonos egyed példányai.

3. Fizetés módosítása

   Ez is egyszerű volt, itt csak a szöveg értékét kellett megváltoztatni a példánynak.

4. Telefonszám hozzáadása

   Ez egy kicsit érdekesebb volt, mert nem biztos hogy egy dolgozónak van telefonszáma, ezért a pozíciót hogy hova kell beszúrni nem volt triviális. Mivel fizetése biztos hogy van, ezért megkerestem azt és az azután jövő element elé szúrtam be a telefonszámot, így ha még eddig nem is volt, most jó helyen lett berakva.