

JEGYZŐKÖNYV

Webes adatkezelő környezetek

Féléves feladat

Könyvtár Nyilvántartó Rendszer

Készítette: **Küzmös Ízisz Meredisz**

Neptunkód: **O8RVRF**

Dátum: **2025. december**

Miskolc, 2025

Tartalom

Bevezetés	3
A feladat leírása:	3
1. Az XML adatbázis tervezés	4
1.1 Az adatbázis ER modell tervezése.....	4
1.2 Az adatbázis konvertálása XDM modellre	8
1.3 Az XDM modell alapján XML dokumentum készítése	12
1.4 Az XML dokumentum alapján XMLSchema készítése	15
2. DOM feldolgozás JAVA-ban	20
2.1 Adatolvasás.....	20
2.2. Adat-lekérdezés	26
2.3. Adatmódosítás	31

Bevezetés

A féléves feladat célja egy komplex webes adatkezelő rendszer megtervezése és megvalósítása XML technológiák felhasználásával.

A választott téma egy könyvtári nyilvántartó rendszer, amely könyvek, szerzők, kiadók, olvasók és kölcsönzések kezelését teszi lehetővé.

A feladat leírása:

A rendszer két fő részből áll:

1. Az adatbázis tervezése és XML/XSD alapú megvalósítása
 - ER modell készítése az entitások és kapcsolatok definiálásával
 - XDM (XML Data Model) modellre konvertálás
 - Validált XML dokumentum és XSD séma létrehozása
2. DOM API alapú Java program fejlesztése
 - XML dokumentum beolvasása és feldolgozása
 - Lekérdezések végrehajtása
 - Adatok módosítása és mentése

A könyvtári rendszer 5 fő entitást tartalmaz:

- Kiadó,
- Könyv,
- Szerző,
- Kategória
- Olvasó.

A rendszer kezeli az entitások közötti 1:1, 1:N és N:M kapcsolatokat, valamint az összetett (cím) és többértékű (telefonszámok) attribútumokat is. A Székhely_Részletek entitás 1:1 kapcsolatban áll a Kiadó entitással, így biztosítva a részletes címadatok elkülönített tárolását.

1. Az XML adatbázis tervezés

1.1 Az adatbázis ER modell tervezése

Az ER (Entity-Relationship) modell az adatbázis logikai szerkezetét ábrázolja. A könyvtári rendszer 5 egyedet tartalmaz, amelyek között különböző típusú kapcsolatok valósulnak meg.

Egyedek és attribútumaik:

1. KIADÓ

- kiadó_id (PK - elsődleges kulcs, aláhúzva)
- név
- alapítási_év

2. SZÉKHELY_RÉSZLETEK

- székhely_id (PK)
- kiadó_id (FK - Foreign Key)
- város
- utca
- házsám
- irányítószám

3. KÖNYV

- könyv_id (PK)
- cím
- ISBN
- kiadási_év
- oldalszám

4. SZERZŐ

- szerző_id (PK)
- név
- születési_év
- nemzetiség

5. KATEGÓRIA

- kategória_id (PK)
- név
- leírás

6. OLVASÓ

- olvasó_id (PK)
- név
- email
- telefonszámok (többértékű attribútum - dupla ovális)
- cím (összetett attribútum):
 - irányítószám
 - város
 - utca
 - házszám

Kapcsolatok:

A. KIADÓ ↔ SZÉKHELY_RÉSZLETEK: "van" (1:1 kapcsolat)

- - Egy kiadónak pontosan egy székhely részletezése van
- - Egy székhely részletezés pontosan egy kiadóhoz tartozik
- - Ez a kapcsolat biztosítja a részletes címadatok elkülönített tárolását

B. KIADÓ → KÖNYV: "kiadta" (1:N kapcsolat)

- Egy kiadó több könyvet is kiadhat
- Egy könyv csak egy kiadóhoz tartozik

C. KÖNYV ↔ SZERZŐ: "írta" (N:M kapcsolat)

- Egy könyvnek több szerzője lehet
- Egy szerző több könyvet is írhat
- Kapcsolat tulajdonsága: szerep (főszerző, társszerző)

D. KÖNYV ↔ KATEGÓRIA: "tartozik" (N:M kapcsolat)

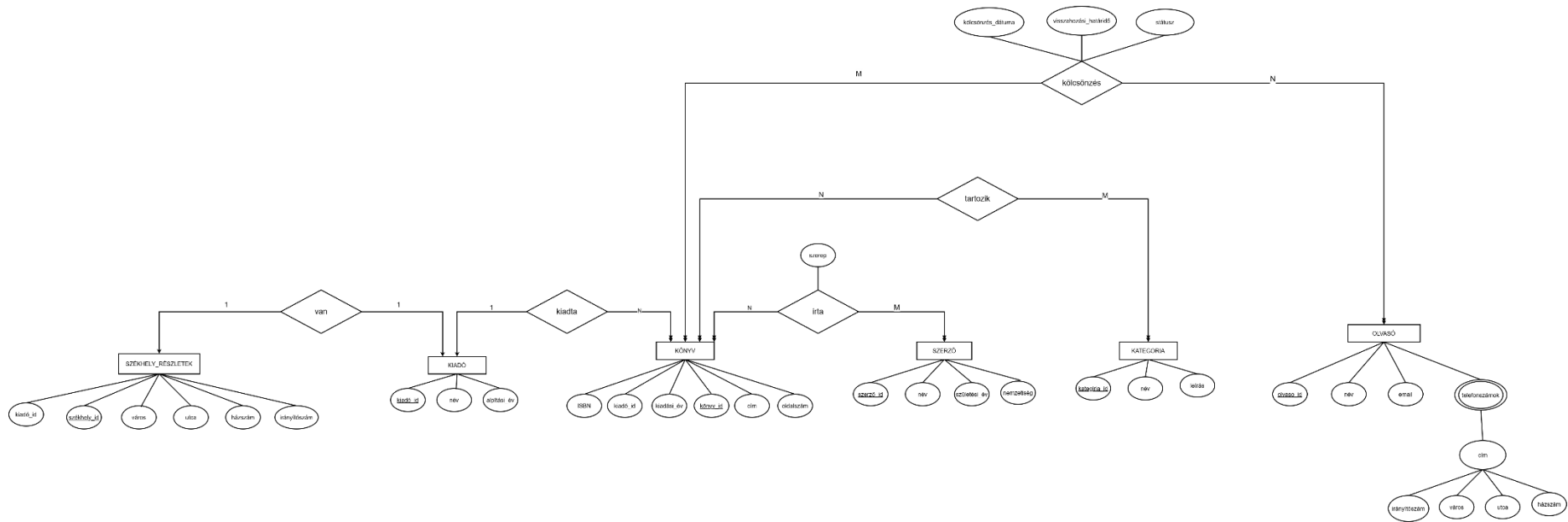
- Egy könyv több kategóriába is tartozhat
- Egy kategóriában több könyv is lehet

E. OLVASÓ ↔ KÖNYV: "KÖLCÖNÖZÉS" (N:M kapcsolat)

- Egy olvasó több könyvet is kölcsönözhet
- Egy könyvet több olvasó is kölcsönözhet
- Kapcsolat tulajdonságai:
 - kölcsönzés_dátuma
 - visszahozási_határidő
 - státusz

Az ER diagram Draw.io programmal készült (1. ábra), szabványos jelölésekkel:

- Téglalapok: Egyedek
- Ovális: Attribútumok
- Rombusz: Kapcsolatok
- Dupla ovális: Többértékű attribútum
- Aláhúzás: Kulcs attribútum
- 1, N, M: Kapcsolatok számossága



1. ábra: ER diagram Draw.io programmal

1.2 Az adatbázis konvertálása XDM modellre

Az XDM (XML Data Model) az ER modell XML-hierarchiában megjelenő reprezentációja, amely fa struktúrát követ. A konverzió szabályai alapján az ER modell entitásai és kapcsolatai XML elemekké, a kulcsok pedig rombusz jelölésekkel láthatóvá alakulnak.

KONVERZIÓS SZABÁLYOK:

1. Hierarchikus fa struktúra

- Gyökérelem: könyvtár (dupla ovális)
- Gyűjtőelemek: ovális alakzatok (pl. kiadok, szerzok)
- PK rombusz: A gyűjtőelem és az egyedi elem között (pl. kiadok → ◇ PK → kiado)
- Egyedi elemek: ovális alakzatok (pl. kiado, szerzo)
- Attribútumok: ovális + téglalap alakzatok

2. Jelölési rendszer

- Dupla ovális: Gyökérelem (könyvtár)
- Ovális: Gyűjtő és egyedi elemek
- Rombusz ◇ PK: Primary Key jelölése a gyűjtőelem és egyedi elem között
- Rombusz ◇ FK: Foreign Key kapcsolat jelölése attribútumnál
- Téglalap: Attribútumok szöveges tartalma
- Vonalak: Hierarchia és FK kapcsolatok

3. Primary Key (PK) rombuszok elhelyezése:

- Minden gyűjtőelem és egyedi elem között van PK rombusz
- Példa: kiadok → ◇ PK → kiado
- Ez jelzi, hogy az egyedi elem rendelkezik elsődleges kulccsal

4. Foreign Key (FK) rombuszok és kapcsolatok:

- Az FK attribútum alatt van FK rombusz
- A FK rombuszból vonal megy a hivatkozott elemhez
- Példa: könyv \rightarrow kiadó_id \rightarrow \diamond FK \rightarrow (vonal a kiado elemhez)
- Ez biztosítja a referenciális integritás vizuális megjelenítését

5. 1:N kapcsolatok kezelése:

- Az N oldalon FK attribútum és FK rombusz
- Példa: könyv tartalmaz kiadó_id (FK) \rightarrow \diamond FK \rightarrow kiado elem

6. N:M kapcsolatok kezelése:

- Új kapcsolótábla elem létrehozása
- Tartalmazza mindkét entitás FK-ját FK rombuszokkal
- Plusz a kapcsolat tulajdonságait
- Példa: könyv_szerzo \rightarrow könyv_id \diamond FK + szerző_id \diamond FK

7. Összetett attribútumok:

- Az összetett attribútum ovális elem marad
- Alatta ovális elemekben az egyszerű komponensek
- Komponensek alatt téglalapok a tartalomnak
- Példa: cím (ovális) \rightarrow irányítószám, város, utca, házszám (ovális+téglalap)

8. Többértékű attribútumok:

- Új gyűjtőelem és egyedi elem létrehozása PK rombuszsal

- FK rombusz mutat vissza a szülő elemre
- Példa: olvaso_telefonok \rightarrow \diamond PK \rightarrow olvaso_telefon \rightarrow olvasó_id \diamond FK \rightarrow olvaso

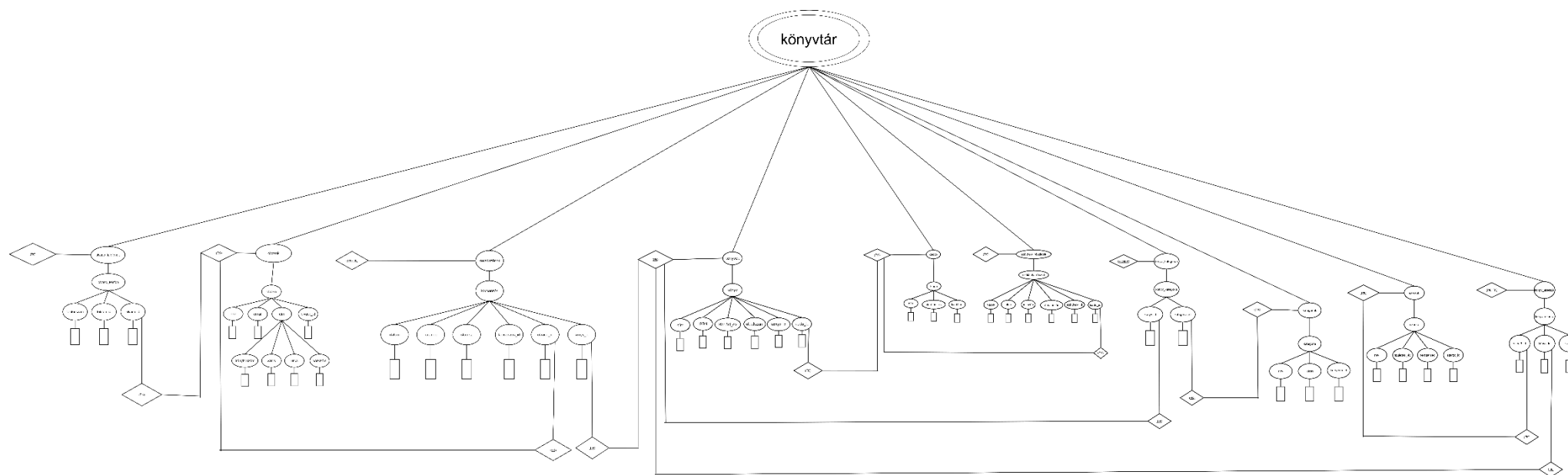
Létrejövő elemek (XML Hierarchy):

Gyökérelem: könyvtár (dupla ovális)

Gyűjtőelemek (10 db)(2.ábra):

1. kiadok
2. székhely_részletek
3. szerzok
4. kategoriak
5. konyvek
6. olvasok
7. olvaso_telefonok
8. konyv_szerzok
9. konyv_kategoriak
10. kolcsonzesek

Fontos szabály: Az elemeket összekötő vonalak (PK→FK) NEM keresztezik egymást!



2. ábra: Az adatbázis XDM modellje

1.3 Az XDM modell alapján XML dokumentum készítése

Az XML dokumentum az XDM modell alapján készült, amely a könyvtári rendszer konkrét adatait tartalmazza. A dokumentum VS Code szerkesztőben készült, és az XSD séma szerint validált.

Fontosabb jellemzők:

1. Hierarchikus struktúra

- Gyökérelem: <konyvtar>
- Minden egyed külön gyűjtőelemben (<kiadok>, <konyvek>, stb.)

2. Minden többszörösen előforduló elemből legalább 2 példány

- 2 kiadó, 2 szerző, 2 kategória, 2 könyv, 2 olvasó
- 3 telefonszám, 2 könyv-szerző kapcsolat, 3 könyv-kategória kapcsolat
- 2 kölcsönzés

3. Attribútumok és elemek

- PK és FK értékek: attribútumként (pl. kiado_id="K1")
- Adatok: elem szövegtartalomként (pl. <nev>Móra Kiadó</nev>)

4. Összetett attribútum kezelése (cím):

```
<cim>
  <iranyitoszam>4032</iranyitoszam>
  <varos>Debrecen</varos>
  <utca>Kossuth utca</utca>
  <hazszam>15</hazszam>
</cim>
```

5. Többértékű attribútum kezelése (telefonszámok):

- Külön <olvaso_telefon> elemek mindegyik telefonszámhoz

6. Megjegyzések használata:

```
<!-- KIADÓK -->
<!-- SZERZŐK -->
```

stb.

Lényeges KÓD részletek és magyarázat:

xml

```
<?xml version="1.0" encoding="UTF-8"?>
<könyvtar xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="O8RVRF_XMLSchema.xsd">
```

Az XML deklaráció megadja a karakterkódolást (UTF-8) és az XSD séma helyét a validáláshoz.

xml

```
<!-- KIADÓK -->
<kiadok>
    <kiado kiado_id="K1">
        <nev>Móra Kiadó</nev>
        <szekhely>Budapest</szekhely>
        <alapitasi_ev>1950</alapitasi_ev>
    </kiado>

    <kiado kiado_id="K2">
        <nev>Animus Kiadó</nev>
        <szekhely>Debrecen</szekhely>
        <alapitasi_ev>1992</alapitasi_ev>
    </kiado>
</kiadok>
```

A kiadók gyűjtőelemben 2 kiadó példány. A kiadó_id attribútum a PK, a többi adat elem.

xml

```
<!-- OLVASÓK -->
<olvasok>
    <olvaso olvaso_id="O1">
        <nev>Nagy Péter</nev>
        <email>nagy.peter@email.hu</email>
        <cim>
            <iranyitoszam>4032</iranyitoszam>
```

```
        <varos>Debrecen</varos>
        <utca>Kossuth utca</utca>
        <hazszam>15</hazszam>
    </cim>
</olvaso>
</olvasok>
```

Az összetett cím attribútum 4 alárendelt elemként jelenik meg.

xml

```
<!-- OLVASÓ TELEFONOK -->
<olvaso_telefonok>
    <olvaso_telefon telefon_id="T1" olvaso_id="O1">
        <telefonszam>+36 30 123 4567</telefonszam>
    </olvaso_telefon>

    <olvaso_telefon telefon_id="T2" olvaso_id="O1">
        <telefonszam>+36 20 987 6543</telefonszam>
    </olvaso_telefon>
</olvaso_telefonok>
```

Többértékű attribútum kezelése: külön tábla, ahol az olvaso_id FK kapcsolja.

xml

```
<!-- KÖNYV-SZERZŐ kapcsolatok -->
<konyv_szerzok>
    <konyv_szerzo konyv_id="KV1" szerzo_id="SZ1">
        <szerep>főszerző</szerep>
    </konyv_szerzo>
</konyv_szerzok>
'''
```

1.4 Az XML dokumentum alapján XMLSchema készítése

Az XSD (XML Schema Definition) séma az XML dokumentum struktúráját és szabályait definiálja. Saját típusok használatával biztosítja az adatok validitását és konzisztenciáját. A séma key és keyref elemekkel garantálja a referenciális integritást (PK-FK kapcsolatok).

Főbb jellemzők:

1. Saját egyszerű típusok (simpleType):

- ISBNTipus: ISBN formátum ellenőrzése (978-963-11-8765-4)
- EmailTipus: Email formátum validálása
- TelefonTipus: Telefonszám formátum (+36 30 123 4567)
- StatuszTipus: Enum-szerű korlátozás (folyamatban, visszahozva, késésben)

2. Saját összetett típusok (complexType):

- KiadoTipus, KonyvTipus, SzerzoTipus, stb.
- CimTipus: Összetett attribútum 4 alárendelt elemmel

3. Attribútum definíciók:

- use="required": Kötelező attribútum (PK, FK)
- type megadása: string, int, date, vagy saját típus

4. Előfordulási megszorítások:

- minOccurs="1": Legalább 1 példány
- maxOccurs="unbounded": Korlátlan számú példány

5. Sorrend meghatározása:

- `<xs:sequence>`: Pontos sorrend követelése

6. Key és Keyref (Referenciális integritás):

- `<xs:key>`: Primary Key definíció
- `<xs:keyref>`: Foreign Key definíció, hivatkozás egy key-re
- XPath szelektorokkal azonosítja az elemeket

Lényeges KÓD részletek és magyarázat:

xml

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

Az XSD séma deklarációja, amely az XML Schema névteret használja.

xml

```
<!-- Gyökér elem: könyvtar -->
<xs:element name="könyvtar">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="kiadok" type="KiadokTipus"/>
      <xs:element name="szerzok" type="SzerzokTipus"/>
      <xs:element name="kategoriak" type="KategoriakTipus"/>
      <xs:element name="konyvek" type="KonyvekTipus"/>
      <xs:element name="olvasok" type="OlvasokTipus"/>
      <xs:element name="olvaso_telefonok" type="OlvasoTelefonokTipus"/>
      <xs:element name="konyv_szerzok" type="KonyvSzerzokTipus"/>
      <xs:element name="konyv_kategoriak" type="KonyvKategoriakTipus"/>
      <xs:element name="kolcsonzesek" type="KolcsonzesekTipus"/>
    </xs:sequence>
  </xs:complexType>
```

A gyökérelem definíciója, amely meghatározza az XML dokumentum felépítését. A sequence biztosítja az elemek pontos sorrendjét.

xml

```
<!-- Saját típus: ISBN formátum -->
<xs:simpleType name="ISBNTipus">
  <xs:restriction base="xs:string">
    <xs:pattern value="\d{3}-\d{3}-\d{2}-\d{4}-\d"/>
  </xs:restriction>
</xs:simpleType>
```

Saját egyszerű típus, amely reguláris kifejezéssel ellenőrzi az ISBN formátumot (pl. 978-963-11-8765-4). A pattern megszorítás csak az adott mintának megfelelő értékeket fogad el.

xml

```
<!-- Saját típus: Email formátum -->  
<xs:simpleType name="EmailTípus">  
    <xs:restriction base="xs:string">  
        <xs:pattern value="[^@]+@[^@]+\.[^@]+" />  
    </xs:restriction>  
</xs:simpleType>
```

Email cím validálása reguláris kifejezéssel. Biztosítja, hogy „@” és „.” karakter szerepeljen a megfelelő helyen.

xml

```
<!-- Saját típus: Státusz (enum-szerű) -->
<xs:simpleType name="StatuszTípus">
  <xs:restriction base="xs:string">
    <xs:enumeration value="folyamatban"/>
    <xs:enumeration value="visszahozva"/>
    <xs:enumeration value="késésben"/>
  </xs:restriction>
</xs:simpleType>
```

Enum-szerű típus, amely csak 3 konkrét értéket engedélyez. Az enumeration megszorítás felsorolja az érvényes lehetőségeket.

xml

```
<!-- KIADÓ típus -->
<xs:complexType name="KiadoTipus">
  <xs:sequence>
    <xs:element name="nev" type="xs:string"/>
    <xs:element name="szekhely" type="xs:string"/>
    <xs:element name="alapitasi_ev" type="xs:int"/>
  </xs:sequence>
  <xs:attribute name="kiado_id" type="xs:string" use="required"/>
</xs:complexType>
```

Összetett típus a kiadó entitáshoz. A sequence meghatározza az elemek sorrendjét. Az attribútum definíció use="required" jelzi, hogy a kiadó_id kötelező (PK).

xml

```
<!-- Saját összetett típus: Cím -->
<xs:complexType name="CimTipus">
  <xs:sequence>
    <xs:element name="iranyitoszam" type="xs:string"/>
    <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>
```

Összetett attribútum kezelése: a cím 4 alárendelt elemből áll, amelyek mind kötelezőek.

xml

```
<!-- KÖNYV típus -->
<xs:complexType name="KonyvTipus">
  <xs:sequence>
    <xs:element name="cim" type="xs:string"/>
    <xs:element name="ISBN" type="ISBNTipus"/>
    <xs:element name="kiadasi_ev" type="xs:int"/>
    <xs:element name="oldalszam" type="xs:int"/>
  </xs:sequence>
  <xs:attribute name="konyv_id" type="xs:string" use="required"/>
  <xs:attribute name="kiado_id" type="xs:string" use="required"/>
</xs:complexType>
```

A KÖNYV típus használja a saját ISBNTipus-t az ISBN elem validálásához. Két kötelező attribútum: konyv_id (PK) és kiado_id (FK az 1:N kapcsolathoz).

xml

```
<!-- KEY definíció - Kiadó PK -->
<xs:key name="kiadoKey">
  <xs:selector xpath="kiadok/kiado"/>
  <xs:field xpath="@kiado_id"/>
</xs:key>
```

A key elem definiálja a kiadó elsődleges kulcsát. A selector meghatározza, hogy melyik elemeken alkalmazzuk a kulcsot (XPath: kiadok/kiado), a field pedig megadja a kulcs attribútumot (@kiado_id). Ez biztosítja, hogy minden kiado_id egyedi legyen.

xml

```
<!-- KEYREF definíció - Könyv -> Kiadó FK -->
<xs:keyref name="konyvKiadoRef" refer="kiadoKey">
  <xs:selector xpath="konyvek/konyv"/>
  <xs:field xpath="@kiado_id"/>
</xs:keyref>
```

A keyref elem definiálja az idegen kulcsot. A refer="kiadoKey" hivatkozik a korábban definiált kiadoKey-re. A selector meghatározza, hogy melyik elemeken alkalmazzuk (konyvek/konyv), a field pedig az FK attribútumot (@kiado_id). Ez biztosítja, hogy minden könyv kiado_id-ja létező kiadóra hivatkozzon - vagyis garantálja a referenciális integritást.

xml

```
<!-- KÖLCSÖNZÉS típus -->
<xs:complexType name="KolcsonzesTipus">
  <xs:sequence>
    <xs:element name="kolcsonzes_datuma" type="xs:date"/>
    <xs:element name="visszahozasi_hatarido" type="xs:date"/>
    <xs:element name="statusz" type="StatuszTipus"/>
  </xs:sequence>
  <xs:attribute name="kolcsonzes_id" type="xs:string" use="required"/>
  <xs:attribute name="olvaso_id" type="xs:string" use="required"/>
  <xs:attribute name="konyv_id" type="xs:string" use="required"/>
</xs:complexType>
```

N:M kapcsolat megvalósítása: 3 attribútum (PK és 2 FK), valamint 3 elem tulajdonságként. A státusz elem a StatuszTipus-t használja.

Key és Keyref Összefoglalás:

Az XSD sémában összesen 5 key és 8 keyref van definiálva:

Key-k (Primary Keys):

1. kiadoKey - Kiadó PK
2. szerzoKey - Szerző PK
3. kategoriaKey - Kategória PK
4. konyvKey - Könyv PK
5. olvasoKey - Olvasó PK

Keyref-ek (Foreign Keys):

1. konyvKiadoRef - Könyv → Kiadó
2. konyvSzerzoKonyvRef - Könyv_Szerző → Könyv

3. `konyvSzerzoSzerzoRef` - `Könyv_Szerző` → `Szerző`
4. `konyvKategoriaKonyvRef` - `Könyv_Kategória` → `Könyv`
5. `konyvKategoriaKategoriaRef` - `Könyv_Kategória` → `Kategória`
6. `olvasoTelefonRef` - `Olvasó_Telefon` → `Olvasó`
7. `kolcsonzesOlvasoRef` - `Kölcsönzés` → `Olvasó`
8. `kolcsonzesKonyvRef` - `Kölcsönzés` → `Könyv`

Ez a struktúra biztosítja, hogy minden FK hivatkozás valóban létező PK-ra mutasson, így az adatok integritása garantált az XML szinten is, nem csak az adatbázis szinten.

2. DOM feldolgozás JAVA-ban

Project name: `O8RVRFDOMParse`

Package: `o8rvrf.domparse.hu`

Class names: `O8RVRFDomRead`, `O8RVRFDomQuery`, `O8RVRFDom`

Modify XML dokumentum: `O8RVRF_XML.xml`

A második feladat célja az XML dokumentum programozott feldolgozása Java nyelven, DOM (Document Object Model) API használatával. A DOM API lehetővé teszi az XML dokumentum fa struktúraként történő kezelését, ahol minden elem, attribútum és szövegtartalom egy-egy csomópont (node).

A program 3 fő osztályból áll, amelyek különböző műveleteket valósítanak meg:

1. `DomRead`: Teljes XML beolvasás, minden node típus feldolgozása és statisztika
2. `DomQuery`: Célzott lekérdezések végrehajtása
3. `DomModify`: XML módosítás és mentés

Fejlesztői környezet:

- IDE: IntelliJ IDEA
- JDK: OpenJDK 22
- XML Parser: `javax.xml.parsers` (beépített)
- Build Tool: Natív Java

2.1 Adatolvasás

Fájlnév: `O8RVRFDomRead.java`

A `DomRead` osztály feladata az XML dokumentum teljes beolvasása és a tartalom strukturált megjelenítése a konzolon. A program rekurzív bejárással dolgozza fel a dokumentumfát,

kiírva minden elemet, attribútumot és különböző node típusokat. Az osztály statisztikát is készít a DOM fa szerkezetéről.

Főbb működési lépések:

1. DocumentBuilder létrehozása

- DocumentBuilderFactory inicializálása
- XML parser konfigurálása

2. XML fájl beolvasása és parse-olása

- File objektum létrehozása az XML elérési úttal
- Document objektum generálása

3. Dokumentum normalizálása

- Whitespace kezelés
- Szöveges node-ok tisztítása

4. Rekurzív bejárás minden node típussal

- ELEMENT_NODE: XML elemek
- TEXT_NODE: Szöveges tartalom
- COMMENT_NODE: Megjegyzések (<!-- ... -->)
- CDATA_SECTION_NODE: CDATA szekciók
- PROCESSING_INSTRUCTION_NODE: Feldolgozási utasítások
- Egyéb node típusok logolása

5. Konzolra írás jelölésekkel

- Nyitó és záró tag-ek
- Attribútumok név="érték" formában
- [TEXT] - Szövegtartalom
- [COMMENT] Megjegyzések
- [CDATA] - CDATA szekciók
- [PI] - Processing instructions

6. DOM fa statisztika

- Elemek száma
- Szöveges node-ok száma
- Megjegyzések száma
- Attribútumok száma

Lényeges KÓD részletek és magyarázat:

java

```
// XML fájl elérési útja
File xmlFile = new File("src/o8rvrf/domparse/hu/O8RVRF_XML.xml");

// DocumentBuilderFactory és DocumentBuilder létrehozása
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();

// XML dokumentum beolvasása
Document document = builder.parse(xmlFile);

// Dokumentum normalizálása
document.getDocumentElement().normalize();
```

Az első lépés a DocumentBuilder objektum létrehozása a Factory tervezési minta használatával. A parse() metódus az XML fájlt beolvassa és DOM fát épít belőle. A normalize() eltávolítja a felesleges whitespace node-okat és összevonja a szomszédos TEXT node-okat.

java

```
private static void printNode(Node node, String indent) {

    // ELEMENT típusú node
    if (node.getNodeType() == Node.ELEMENT_NODE) {
        Element element = (Element) node;

        // Elem neve kiírása
        System.out.print(indent + "<" + element.getNodeName());

        // Attribútumok kiírása
        NamedNodeMap attributes = element.getAttributes();
```

```

        if (attributes.getLength() > 0) {
            for (int i = 0; i < attributes.getLength(); i++) {
                Node attr = attributes.item(i);
                System.out.print(" " + attr.getNodeName() +
                                "=\"" + attr.getNodeValue() + "\"");
            }
        }
        System.out.println(">");

        // Gyermek node-ok rekurzív bejárása
        NodeList children = node.getChildNodes();
        for (int i = 0; i < children.getLength(); i++) {
            printNode(children.item(i), indent + " ");
        }

        // Záró tag
        System.out.println(indent + "</" + element.getNodeName() + ">");
    }
}

```

A rekurzív printNode metódus feldolgozza a DOM fát. Az Element node esetén kiírja a nyitó tag-et az attribútumokkal, majd rekurzívan meghívja magát minden gyermek node-ra. Végül kiírja a záró tag-et. Az indent paraméter biztosítja a hierarchikus megjelenítést behúzással.

java

```

// TEXT típusú node - szöveges tartalom
else if (node.getNodeType() == Node.TEXT_NODE) {
    String content = node.getNodeValue().trim();
    if (!content.isEmpty()) {
        System.out.println(indent + "[TEXT] " + content);
    }
}

// COMMENT típusú node - XML megjegyzések
else if (node.getNodeType() == Node.COMMENT_NODE) {
    String comment = node.getNodeValue().trim();
    System.out.println(indent + "[COMMENT] <!-- " + comment + " -->");
}

// CDATA típusú node - CDATA szekciók
else if (node.getNodeType() == Node.CDATA_SECTION_NODE) {

```

```

        String cdata = node.getNodeValue();
        System.out.println(indent + "[CDATA] <![CDATA[" + cdata + "]]>");
    }
}

```

A különböző node típusok kezelése egyedi jelölésekkel. A TEXT node esetén a getNodeValue() visszaadja a szövegtartalmat, amit trim()-elünk (whitespace eltávolítása). Csak a nem üres szövegeket írjuk ki [TEXT] jelöléssel. A COMMENT node-ok [COMMENT] jelöléssel láthatóak, így könnyen azonosíthatók az XML megjegyzések. A CDATA szekciók [CDATA] jelöléssel kerülnek kiírásra.

java

```

/**
 * DOM fa statisztika - node típusok számlálása
 */
private static void printStatistics(Node root) {
    int[] counts = new int[4]; // [elemek, szövegek, megjegyzések, attribútumok]
    countNodes(root, counts);

    System.out.println("Elemek száma: " + counts[0]);
    System.out.println("Szöveges node-ok száma: " + counts[1]);
    System.out.println("Megjegyzések száma: " + counts[2]);
    System.out.println("Attribútumok száma: " + counts[3]);
}

private static int[] countNodes(Node node, int[] counts) {
    if (node.getNodeType() == Node.ELEMENT_NODE) {
        counts[0]++;
        NamedNodeMap attrs = node.getAttributes();
        if (attrs != null) {
            counts[3] += attrs.getLength();
        }
    } else if (node.getNodeType() == Node.TEXT_NODE &&
        !node.getNodeValue().trim().isEmpty()) {
        counts[1]++;
    } else if (node.getNodeType() == Node.COMMENT_NODE) {
        counts[2]++;
    }

    NodeList children = node.getChildNodes();
    for (int i = 0; i < children.getLength(); i++) {
        countNodes(children.item(i), counts);
    }
}

```

```

    return counts;
}
...

```

*A statisztika modul rekurzívan végigmegy a teljes DOM fán és számolja a különböző node típusokat. Az int tömb 4 számlálót tartalmaz: elemek, szöveges node-ok, megjegyzések és attribútumok. Az Element node esetén az attribútumokat is megszámloljuk a getAttributes() metódussal. Ez a statisztika segít megérteni az XML dokumentum komplexitását és szerkezetét. *

****KIMENET RÉSZLET:****

```
...
```

=== XML Dokumentum Olvasása ===

Gyökér elem: konyvtar

--- Teljes XML tartalom ---

<konyvtar>

[COMMENT] <!-- KIADÓK -->

<kiadok>

<kiado kiado_id="K1">

<nev>

[TEXT] Móra Kiadó

</nev>

<szekhely>

[TEXT] Budapest

</szekhely>

...

=== DOM Fa Statisztika ===

Elemek száma: 79

Szöveges node-ok száma: 47

Megjegyzések száma: 9

Attribútumok száma: 34

A kimenet jól mutatja a különböző node típusok jelölését: [TEXT] a szöveges tartalomhoz, [COMMENT] a megjegyzésekhez. A statisztika összefoglalja a DOM fa méretét: 79 elem, 47 szöveges tartalom, 9 megjegyzés és 34 attribútum.

2.2. Adat-lekérdezés

Fájlnév: O8RVRFDomQuery.java

A DomQuery osztály célzott lekérdezéseket hajt végre az XML dokumentumon.

A lekérdezések különböző feltételek alapján keresnek adatokat, hasonlóan egy adatbázis SELECT műveleteihez.

A program 5 különböző lekérdezést valósít meg:

1. Összes kiadó listázása

- `getElementsByTagName()` használata
- Név és székhely kiírása

2. Könyv részletes adatai ID alapján

- Attribútum alapú keresés
- Összes kapcsolódó adat kigyűjtése

3. Adott kiadó összes könyve

- FK alapú szűrés
- Kapcsolt adatok megjelenítése

4. Olvasó teljes adatlapja

- Összetett attribútum (cím) kezelése
- Többértékű attribútum (telefonszámok) összegyűjtése

5. Folyamatban lévő kölcsönzések

- Státusz alapú szűrés
- Kapcsolt entitások adatainak megjelenítése

Fontos: A lekérdezéseknél NEM használunk XPath kifejezéseket, csak DOM API metódusokat!

Lényeges KÓD részletek és magyarázat:

java

```
// 1. Lekérdezés: Összes kiadó  
private static void query1_OsszesKiado() {
```

```

NodeList kiadok = document.getElementsByTagName("kiado");

for (int i = 0; i < kiadok.getLength(); i++) {
    Element kiado = (Element) kiadok.item(i);
    String id = kiado.getAttribute("kiado_id");
    String nev = kiado.getElementsByTagName("nev")
        .item(0).getTextContent();
    String szekhely = kiado.getElementsByTagName("szekhely")
        .item(0).getTextContent();

    System.out.println("Kiadó ID: " + id);
    System.out.println("  Név: " + nev);
    System.out.println("  Székhely: " + szekhely);
}
}

```

A `getElementsByTagName()` visszaad minden "kiado" elemet `NodeList` formában. A ciklus végigmegy az összes elemen, és mindegyikből kinyeri az id attribútumot (`getAttribute`), valamint a név és székhely elemek tartalmát (`getTextContent`).

java

```

// 2. Lekérdezés: Könyv részletei ID alapján
private static void query2_KonyvReszletei(String konyvId) {
    NodeList konyvek = document.getElementsByTagName("konyv");

    for (int i = 0; i < konyvek.getLength(); i++) {
        Element konyv = (Element) konyvek.item(i);
        String id = konyv.getAttribute("konyv_id");

        if (id.equals(konyvId)) {
            String cim = konyv.getElementsByTagName("cim")
                .item(0).getTextContent();
            String isbn = konyv.getElementsByTagName("ISBN")
                .item(0).getTextContent();

            // ... további adatok

```

```

        System.out.println("Cím: " + cim);
        System.out.println("ISBN: " + isbn);
        return; // Megtaláltuk, kilépünk
    }
}

```

Ez egy FK (Foreign Key) alapú szűrés, ahol a könyv táblában lévő kiadó_id attribútum alapján keressük meg az összes kapcsolódó könyvet.

java

//3. Lekérdezés: Adott kiadó összes könyve

```

private static void query3_KiadoKonyvei(String kiadoId) {
    System.out.println("-- 3. LEKÉRDEZÉS: Kiadó könyvei (Kiadó ID: "
        + kiadoId + ") ---");
    NodeList konyvek = document.getElementsByTagName("konyv");
    int szamlalo = 0;

    for (int i = 0; i < konyvek.getLength(); i++) {
        Element konyv = (Element) konyvek.item(i);
        String kid = konyv.getAttribute("kiado_id");

        if (kid.equals(kiadoId)) {
            szamlalo++;
            String cim = konyv.getElementsByTagName("cim")
                .item(0).getTextContent();
            String konyvId = konyv.getAttribute("konyv_id");

            System.out.println("  " + szamlalo + ". " + cim
                + " (ID: " + konyvId + ")");
        }
    }

    if (szamlalo == 0) {
        System.out.println("  Nincs könyv ettől a kiadótól!");
    }

    System.out.println();
}

```

```
}
```

ID alapú keresés: végigmegyünk az összes könyvön, és összehasonlítjuk az id attribútumot a keresett értékkel (equals). Ha megtaláltuk, kigyűjtjük az összes adatot és return-nel kilépünk.

java

```
// 4. Lekérdezés: Olvasó adatai összetett és többértékű
attribútumokkal

private static void query4_OlvasoAdatai(String olvasoId) {
    NodeList olvasok = document.getElementsByTagName("olvaso");

    for (int i = 0; i < olvasok.getLength(); i++) {
        Element olvaso = (Element) olvasok.item(i);
        String id = olvaso.getAttribute("olvaso_id");

        if (id.equals(olvasoId)) {
            // Összetett attribútum kezelése
            Element cimElem = (Element) olvaso.getElementsByTagName("cim")
                .item(0);
            String irszam = cimElem.getElementsByTagName("iranyitoszam")
                .item(0).getTextContent();
            String varos = cimElem.getElementsByTagName("varos")
                .item(0).getTextContent();

            // ...

            System.out.println("Cím: " + irszam + " " + varos +
                "...");

            // Többértékű attribútum: telefonszámok
            NodeList telefonok = document.getElementsByTagName("olvaso_telefon");
            for (int j = 0; j < telefonok.getLength(); j++) {
                Element telefon = (Element) telefonok.item(j);
                String oid = telefon.getAttribute("olvaso_id");
                if (oid.equals(olvasoId)) {
                    String szam = telefon.getElementsByTagName("telefonszam")
                        .item(0).getTextContent();
                    System.out.println("    - " + szam);
                }
            }
        }
    }
}
```

```

        }
    }
}
}
}

```

Összetett attribútum: először megkeressük a <cim> elemet, majd ebből gyűjtjük ki az alárendelt elemeket (irányítószám, város, stb.). Többértékű attribútum: külön táblában van (olvaso_telefon), ezért egy másik ciklussal végigmegyünk, és az olvaso_id FK alapján szűrjük ki az adott olvasóhoz tartozó telefonokat.

java

```

// 5. Lekérdezés: Státusz alapú szűrés
private static void query5_FolyamatbanLevoKolcsonzesek() {
    NodeList kolcsonzesek = document.getElementsByTagName("kolcsonzes");

    for (int i = 0; i < kolcsonzesek.getLength(); i++) {
        Element kolcsonzes = (Element) kolcsonzesek.item(i);
        String statusz = kolcsonzes.getElementsByTagName("statusz")
            .item(0).getTextContent();

        if (statusz.trim().equals("folyamatban")) {
            // Adatok kiírása
        }
    }
}
}
...

```

***Feltételes lekérdezés:** a státusz elem tartalmát összehasonlítjuk a keresett értékkel.

A `trim()` eltávolítja a felesleges whitespace karaktereket.*

****Kimenet részlet:****

...

=== XML Dokumentum Lekérdezések ===

--- 1. LEKÉRDEZÉS: Összes kiadó ---

Kiadó ID: K1

Név: Móra Kiadó

Székhely: Budapest

--- 3. LEKÉRDEZÉS: Kiadó könyvei (Kiadó ID: K1) ---

1. A fekete város (ID: KV1)

2. Szent Péter esernyője (ID: KV2)

--- 4. LEKÉRDEZÉS: Olvasó adatai (ID: O1) ---

Név: Nagy Péter

Email: nagy.peter@email.hu

Cím: 4032 Debrecen, Kossuth utca 15

Telefonszámok:

- +36 30 123 4567

- +36 20 987 6543

2.3. Adatmódosítás

Fájlnév: O8RVRFDomModify.java

A DomModify osztály az XML dokumentum módosítását valósítja meg. A program különböző művelet típusokat támogat: új elemek hozzáadása, meglévő elemek módosítása, elemek törlése, majd a módosított dokumentum mentése új fájlba.

A program 5 különböző lekérdezést valósít meg:

1. Új elem hozzáadása (kiadó)

- `createElement()` használata
- `setAttribute()` és `setTextContent()`
- `appendChild()` a DOM fába illesztéshez

2. Meglévő elem módosítása (könyv oldalszáma)

- Elem keresése ID alapján
- Szövegtartalom cseréje

3. Új komplex elem hozzáadása (olvasó összetett attribútummal)

- Többszintű elem hierarchia építése
- Összetett attribútum (cím) kezelése

4. Attribútum módosítása (kölcsonzés státusza)

- Elem tartalmának frissítése

5. Elem törlése (kölcsonzés)

- `removeChild()` használata
- Parent node keresése

6. Módosított XML mentése

- Transformer használata
- Formázott kimenet (indent)

Lényeges KÓD részletek és magyarázat:

java

```
// 1. Módosítás: Új elem hozzáadása
private static void modify1_UjKiadoHozzaadasa() {
    // Szülő elem megkeresése
    NodeList kiadokList = document.getElementsByTagName("kiadok");
    Element kiadok = (Element) kiadokList.item(0);

    // Új elem létrehozása
    Element ujKiado = document.createElement("kiado");
    ujKiado.setAttribute("kiado_id", "K3");

    // Gyermek elemek létrehozása és hozzáadása
    Element nev = document.createElement("nev");
    nev.setTextContent("Helikon Kiadó");
    ujKiado.appendChild(nev);

    Element szekhely = document.createElement("szekhely");
    szekhely.setTextContent("Budapest");
    ujKiado.appendChild(szekhely);
}
```

```

        // Új elem hozzáadása a szülőhöz
        kiadok.appendChild(ujKiado);
    }

```

*Új elem hozzáadása lépései:

1. Megkeressük a szülő elemet (kiadok)
2. createElement() létrehoz egy új elemet, de még nincs a fában
3. setAttribute() beállítja az attribútumokat
4. settextContent() beállítja a szövegtartalmat a gyermek elemeknek
5. appendChild() hozzáfűzi a gyermek elemeket a szülőhöz
6. Végül az új kiadót is appendChild()-dal hozzáadjuk a kiadok listához*

java

```

// 2. Módosítás: Elem tartalmának módosítása
private static void modify2_KonyvAdatModositasa(String konyvId,
                                                String ujOldalszam)
{
    NodeList konyvek = document.getElementsByTagName("konyv");

    for (int i = 0; i < konyvek.getLength(); i++) {
        Element konyv = (Element) konyvek.item(i);
        String id = konyv.getAttribute("konyv_id");

        if (id.equals(konyvId)) {
            // Elem keresése és módosítása
            Element oldalszam = (Element) konyv
                .getElementsByTagName("oldalszam").item(0);
            String regiErtek = oldalszam.getTextContent();
            oldalszam.setTextContent(ujOldalszam);

            System.out.println("Régi: " + regiErtek);
            System.out.println("Új: " + ujOldalszam);
            return;
        }
    }
}

```

*Elem módosítása:

1. Megkeressük a módosítandó elemet ID alapján
2. `getElementsByTagName()`-mel kigyűjtjük a módosítandó gyermek elemet (oldalszam)
3. `getTextContent()` lekéri a régi értéket (log céljából)
4. `setTextContent()` beállítja az új értéket A DOM automatikusan frissül, de még nem mentettük fájlba!*

java

```
// 3. Módosítás: Összetett elem hozzáadása
private static void modify3_UjOlvasoHozzaadasa() {
    Element ujOlvaso = document.createElement("olvaso");
    ujOlvaso.setAttribute("olvaso_id", "03");

    // ...név, email...

    // Összetett attribútum: cím
    Element cim = document.createElement("cim");

    Element irányitoszam = document.createElement("iranyitoszam");
    irányitoszam.setTextContent("4024");
    cim.appendChild(írányitoszam);

    Element varos = document.createElement("varos");
    varos.setTextContent("Debrecen");
    cim.appendChild(varos);

    // ... utca, házszám ...

    ujOlvaso.appendChild(cim);

    // Hozzáadás a szülőhöz
    olvasok.appendChild(ujOlvaso);
}
```

*Összetett attribútum létrehozása:

1. Létrehozzuk a főelemet (cím)
2. Létrehozzuk a gyermek elemeket (irányítószám, város, stb.)
3. A gyermekeket hozzáadjuk a cím elemhez
4. A cím elemet hozzáadjuk az olvasóhoz Hierarchia: olvasó > cím > irányítószám, város, utca, házszám*

java

// 5. Módosítás: Elem törlése

```
private static void modify5_KolcsonzesTorlese(String kolcsonzesId) {
    NodeList kolcsonzesek = document.getElementsByTagName("kolcsonzes");

    for (int i = 0; i < kolcsonzesek.getLength(); i++) {
        Element kolcsonzes = (Element) kolcsonzesek.item(i);
        String id = kolcsonzes.getAttribute("kolcsonzes_id");

        if (id.equals(kolcsonzesId)) {
            // Szülő elem megkeresése és gyermek eltávolítása
            Node parent = kolcsonzes.getParentNode();
            parent.removeChild(kolcsonzes);

            System.out.println("✓ Törölve: " + kolcsonzesId);
            return;
        }
    }
}
```

*Elem törlése:

1. Megkeressük a törlendő elemet ID alapján
2. getParentNode() megadja a szülő node-ot
3. removeChild() eltávolítja a gyermek node-ot a szülőből A DOM frissül, de a fájl még nem!*

java

// Módosított XML mentése

```
private static void saveModifiedXML(String filename)
```

```

        throws TransformerException {
TransformerFactory transformerFactory = TransformerFactory.newInstance();
Transformer transformer = transformerFactory.newTransformer();

// Formázás beállítása
transformer.setOutputProperty(OutputKeys.INDENT, "yes");
transformer.setOutputProperty(
    "{http://xml.apache.org/xslt}indent-amount", "2");
transformer.setOutputProperty(OutputKeys.ENCODING, "UTF-8");

// Mentés
DOMSource source = new DOMSource(document);
StreamResult result = new StreamResult(new File(filename));

transformer.transform(source, result);
}
...

```

*XML mentése fájlba:

1. TransformerFactory létrehozza a Transformer objektumot
2. `setOutputProperty()` beállítja a formázást:
 - INDENT: behúzás használata (yes)
 - indent-amount: 2 szóköz behúzás
 - ENCODING: UTF-8 karakterkódolás
3. DOMSource becsomagolja a Document objektumot
4. StreamResult meghatározza a kimeneti fájlt
5. `transform()` végrehajtja a mentést

Az eredmény egy szépen formázott, olvasható XML fájl!*

****KIMENET:****

...

=== XML Dokumentum Módosítása ===

--- 1. MÓDOSÍTÁS: Új kiadó hozzáadása ---

✓ Új kiadó hozzáadva: Helikon Kiadó (K3)

--- 2. MÓDOSÍTÁS: Könyv oldalszámának módosítása ---

✓ Könyv ID: KV1

Régi oldalszám: 320

Új oldalszám: 350

--- 3. MÓDOSÍTÁS: Új olvasó hozzáadása ---

✓ Új olvasó hozzáadva: Tóth Eszter (O3)

--- 4. MÓDOSÍTÁS: Kölcsönzés státuszának módosítása ---

✓ Kölcsönzés ID: KOLCS1

Régi státusz: folyamatban

Új státusz: visszahozva

--- 5. MÓDOSÍTÁS: Kölcsönzés törlése ---

✓ Kölcsönzés törölve: KOLCS2

✓ Minden módosítás sikeresen végrehajtva!

✓ Módosított XML mentve: O8RVRF_XML_MODIFIED.xml