Jegyzőkönyv

Webes adatkezelő környezetek

Féléves feladat

Könyvesbolt

Készítette: Orosz Péter
Neptun kód: WO02D7
Dátum: 2025. December

Tartalomjegyzék

Bevezetés

A modern webes alkalmazások egyik alapvető feladata az adatok hatékony kezelése és tárolása. Az XML (Extensible Markup Language) széles körben elterjedt formátum strukturált adatok leírására, cseréjére és validálására, különösen olyan rendszerekben, ahol a platformfüggetlenség és az adatok áttekinthetősége kiemelt szempont. A jelen jegyzőkönyv célja bemutatni egy könyvesbolt adatkezelő rendszerének tervezését és megvalósítását XML alapokon, az adatmodellezéstől kezdve az XML dokumentum és séma elkészítésén át a Java nyelvű feldolgozásig. A dokumentum lépésről lépésre ismerteti az adatbázis tervezésének folyamatát, az XML-re optimalizált modell kialakítását, valamint az adatok kezelését DOM Parser segítségével.

A feladat leírása

A féléves feladat célja egy könyvesbolt adatainak modellezése, tárolása és feldolgozása XML technológiák alkalmazásával. A feladat során első lépésként meg kell tervezni az adatbázis ER modelljét, amely tartalmazza a könyvesbolt működéséhez szükséges főbb entitásokat (vevők, profilok, rendelések, rendelési tételek, könyvek, szerzők, kiadók) és azok kapcsolatait. Az ER modell alapján el kell készíteni az XML-re optimalizált XDM modellt, majd ennek megfelelően létre kell hozni egy mintaként szolgáló XML dokumentumot, amely tartalmazza a könyvesbolt adatait. Az XML dokumentum szerkezetének validálásához XMLSchema-t kell készíteni, amely meghatározza az adatok típusát, kötelező mezőit és kapcsolatait. Végül Java nyelven, DOM Parser segítségével kell megvalósítani az adatok beolvasását, feldolgozását és táblázatos megjelenítését, demonstrálva az XML alapú adatkezelés gyakorlati alkalmazását.

1 Az adatbázis tervezése és megvalósítása XML-ben

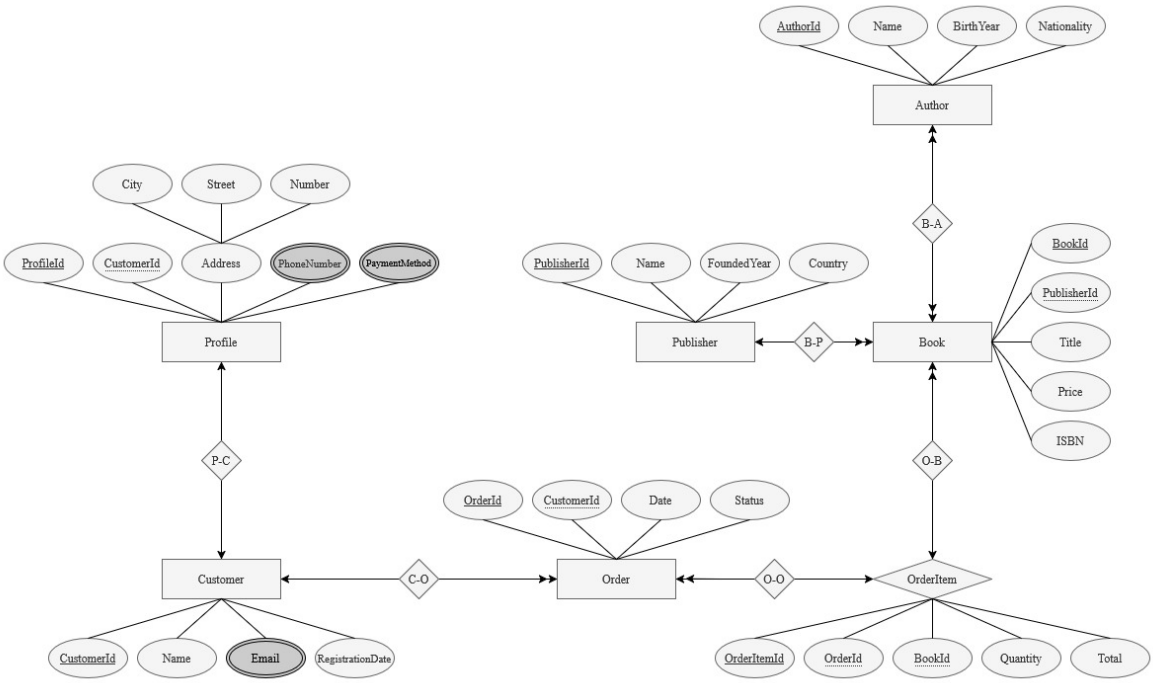
1.1 Az adatbázis ER modell tervezése

Az adatbázis szerkezetét az ER diagram szemlélteti (lásd az 1. ábrát), amelyben az entitások, attribútumok és kapcsolatok kerültek megtervezésre. Az ER modell célja, hogy áttekinthetően ábrázolja az adatbázis logikai felépítését, a főbb adatcsoportokat és azok összefüggéseit.

Az ER modell tervezése során először azokat a főbb adatcsoportokat (entitásokat) azonosítottam, amelyek a könyváruház működéséhez szükségesek: vevők (Customer), profilok (Profile), rendelések (Order), rendelési tételek (OrderItem), könyvek (Book), szerzők (Author) és kiadók (Publisher). Ezeket a valós üzleti folyamatok alapján választottam ki, figyelembe véve, hogy milyen információkat kell tárolni és milyen kapcsolatokat kell kezelni közöttük.

Az entitásokhoz hozzárendeltem a szükséges attribútumokat, például a vevőhöz nevet, email címet és regisztrációs dátumot, a könyvhöz címet, árat és ISBN-t. A kapcsolatok meghatározásánál ügyeltem arra, hogy a valós kapcsolatok (pl. egy vevő több rendelést is leadhat, egy könyvet több szerző is írhat) helyesen jelenjenek meg az adatmodellben. Az összetett kapcsolatokhoz (pl. Book-Author) kapcsolótáblát (B-A) terveztem.

Az ER diagram elkészítéséhez először papíron vázlatot készítettem, majd digitálisan, diagramkészítő eszközzel rajzoltam meg a végleges változatot, amelyet az alábbi ábrán mutatok be.



1. ábra. Az adatbázis ER diagramja

Entitások és attribútumaik

Entitás	Főkulcs	Idegen kulcs(ok)	Attribútum(ok) (típus)
Profile	ProfileId	CustomerId	Address: City (egyszeres) Address: Street (egyszeres) Address: Number (egyszeres) PhoneNumber (többszörös) PaymentMethod (többszörös)
Customer	CustomerId	–	Name (egyszeres) Email (többszörös) RegistrationDate (egyszeres)
Order	OrderId	CustomerId	Date (egyszeres) Status (egyszeres)
OrderItem	OrderItemId	OrderId, BookId	Quantity (egyszeres) Total (egyszeres)
Book	BookId	PublisherId	Title (egyszeres) Price (egyszeres) ISBN (egyszeres)
Author	AuthorId	–	Name (egyszeres) BirthYear (egyszeres) Nationality (egyszeres)
B-A	–	BookId, AuthorId	–
Publisher	PublisherId	–	Name (egyszeres) FoundedYear (egyszeres) Country (egyszeres)

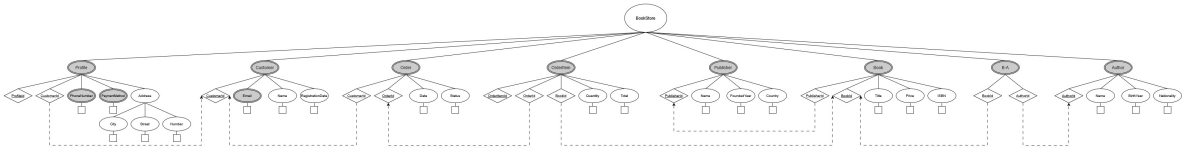
1.2 Az adatbázis konvertálása XDM modellre

Az ER diagram alapján elkészült az XDM modell (lásd a 2. ábrát), amely az adatszerkezetet XML-re optimalizált formában mutatja be. Az XDM modell segíti az XML dokumentum struktúrájának megtervezését, figyelembe véve az adatok hierarchiáját és összefüggéseit.

Az XDM modell elkészítésekor az ER diagram entitásait és kapcsolatait kellett úgy átalakítanom, hogy azok megfeleljenek az XML hierarchikus szerkezetének. Első lépésként meghatároztam, hogy mely entitások lesznek főbb gyökérelemek (pl. Customer, Book), és melyek lesznek beágyazott vagy hivatkozott elemek (pl. Order, OrderItem).

Az ER modell relációs szerkezetét át kellett alakítani úgy, hogy az XML-ben a kapcsolatok vagy beágyazással, vagy attribútumként történő hivatkozással jelenjenek meg. Például a rendelési tételeket (OrderItem) az adott rendelés (Order) alá helyeztem, és a könyvekre, szerzőkre hivatkozásokat attribútumként vagy külön elemként jelenítettem meg.

A modell készítése során törekedtem arra, hogy az adatok logikusan, könnyen feldolgozható módon jelenjenek meg az XML-ben, és a későbbi validálás is egyszerű legyen. A végleges XDM modellt szintén diagramkészítő eszközzel rajzoltam meg.



2. ábra. Az adatbázis XDM modellje

1.3 Az XDM modell alapján XML dokumentum készítése

Az XDM modellből kiindulva elkészült az XML dokumentum (lásd: `WO02D7_XML.xml`), amely az adatok tényleges tárolását biztosítja. Az XML fájlban az adatok a modellnek megfelelően hierarchikusan, jól strukturáltan jelennek meg.

Az XML dokumentum elkészítéséhez először az XDM modell szerkezetét követtem, és meghatároztam a főbb elemeket és azok attribútumait. Az egyes entitásokhoz tartozó adatokat mintaként töltöttem ki, ügyelve arra, hogy minden kötelező mező szerepeljen, és a kapcsolatok (pl. rendelés és rendelési tételek, könyvek és szerzők) is megjelenjenek.

Az XML szerkesztése során figyeltem arra, hogy a dokumentum jól olvasható, áttekinthető legyen, és megfeleljen az XDM modellben megadott hierarchiának. A dokumentumot Visual Studio Code-al készítettem el, majd ellenőriztem, hogy szintaktikailag helyes legyen.

```
<?xml version="1.0" encoding="UTF-8"?>
<BookStore>
  <Customer CustomerId="C001">
    <Name>John Doe</Name>
    <Email>john.doe@example.com</Email>
    <RegistrationDate>2022-01-15</RegistrationDate>
  </Customer>
  ...
</BookStore>
```

Az XML dokumentum szerkezete lehetővé teszi több vevő, profil, rendelés, szerző, kiadó és könyv kezelését, valamint az adatok közötti kapcsolatok megjelenítését.

1.4 Az XML dokumentum alapján XMLSchema készítése

Az XML dokumentum szerkezetének validálásához elkészült az XMLSchema (lásd: `WO02D7_XMLSchema.xsd`), amely meghatározza az elemek típusait, kötelező és opcionális mezőket, valamint az adatok közötti kapcsolatokat. Az XMLSchema biztosítja, hogy az XML dokumentum megfeleljen a kívánt szerkezeti és tartalmi követelményeknek.

Az XMLSchema készítése során az XML dokumentum szerkezetét vettem alapul, és minden főbb elemhez (pl. Customer, Book, Order) létrehoztam a megfelelő komplex típusokat. Meghatároztam, hogy mely elemek kötelezőek, melyek ismétlődhetnek (pl. több email cím), és definiáltam az attribútumokat is.

Az adatok helyességének biztosítására egyedi típusokat (simpleType) is létrehoztam, például az email címekhez reguláris kifejezést használtam, hogy csak érvényes formátumú email címek legyenek elfogadva. Hasonló módon jártam el a telefonszámok, fizetési módok és státuszok esetében is, ahol szükséges volt.

Az XMLSchema-t többször teszteltem az XML dokumentumra, hogy minden szerkezeti és tartalmi követelmény teljesüljön, és a validáció hibamentes legyen.

```
<xs:element name="Customer" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Name" type="xs:string"/>
      <xs:element name="Email" type="EmailType" maxOccurs="unbounded"/>
      <xs:element name="RegistrationDate" type="xs:date"/>
    </xs:sequence>
    <xs:attribute name="CustomerId" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>

<xs:simpleType name="EmailType">
  <xs:restriction base="xs:string">
    <xs:pattern value="[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}"/>
  </xs:restriction>
</xs:simpleType>
```

Az XMLSchema részleteiben megtalálhatók az egyedi típusdefiníciók, mint például az email címek, telefonszámok, fizetési módok és státuszok mintái, amelyek biztosítják az adatok helyességét és konzisztenciáját.

2 A DOM Parser használata Java-ban

2.1 Adatolvasás

Az XML-ből történő adatolvasás a DOM Parser segítségével lépésről lépésre történik. Az alábbiakban bemutatom a folyamatot a legfontosabb kódrészletekkel és azok jelentőségével.

1. **Az XML dokumentum betöltése és feldolgozása**

   Először a program betölti az XML fájlt, majd DOM objektummá alakítja azt:

   ```
   File file = new File("WO02D7_XML.xml");
   DocumentBuilderFactory documentBuilderFactory = DocumentBuilderFactory.newInstance();
   DocumentBuilder documentBuilder = documentBuilderFactory.newDocumentBuilder();
   Document document = documentBuilder.parse(file);
   document.getDocumentElement().normalize();
   ```

   Ezek a sorok biztosítják, hogy az XML dokumentum beolvasásra és feldolgozásra kerüljön. A `normalize()` hívás eltávolítja a felesleges whitespace karaktereket, így az adatok feldolgozása egyszerűbb.

2. **A gyökérelem és gyermekelemek bejárása**

   A program lekéri a gyökérelemet, majd végigiterál a közvetlen gyermekelemeken:

   ```
   Element root = document.getDocumentElement();
   NodeList children = root.getChildNodes();

   for (int i = 0; i < children.getLength(); i++) {
       Node child = children.item(i);
       if (child.getNodeType() == Node.ELEMENT_NODE) {
           Element childElement = (Element) child;
           // feldolgozás...
       }
   }
   ```

   Ez a szerkezet biztosítja, hogy csak az elemtípusú (tag) node-okat dolgozzuk fel, így az XML szerkezetének minden fő entitása (pl. Customer, Profile, Order) külön-külön kezelhető.

3. **Adatok kinyerése és táblázatos megjelenítése**

   Minden entitás esetén a program először egyszer jeleníti meg a táblázat fejlécét, majd minden rekordot egy sorban ír ki. Például a vevők (Customer) esetén:

```
if (!customerHeader) {
    System.out.println("\n=== Customer Table ===");
    System.out.printf("| %-12s | %-20s | %-60s | %-16s |%n", "CustomerId", "Name", "Emails", "RegistrationDate");
    customerHeader = true;
}
String customerId = childElement.getAttribute("CustomerId");
String name = childElement.getElementsByTagName("Name").item(0).getTextContent();
NodeList emails = childElement.getElementsByTagName("Email");
List<String> emailList = new ArrayList<>();
for (int j = 0; j < emails.getLength(); j++) {
    emailList.add(emails.item(j).getTextContent());
}
String registrationDate = childElement.getElementsByTagName("RegistrationDate").item(0).getTextContent();
System.out.printf("| %-12s | %-20s | %-60s | %-16s |%n",
    customerId,
    name,
    String.join(", ", emailList),
    registrationDate
);
```

   Itt látható, hogy az attribútumokat (`getAttribute`) és az al-elemek szövegét (`getElementsByTagName(...).item(0).getTextContent()`) külön-külön olvassuk ki. Az email címek többszörös előfordulása miatt egy listába gyűjtjük őket, majd összefűzzük.

4. **Többszörös előfordulású elemek kezelése**

   Az olyan mezőknél, amelyek többször is előfordulhatnak (pl. Email, PhoneNumber, PaymentMethod), a program minden előfordulást kigyűjt egy listába:

```
NodeList emails = childElement.getElementsByTagName("Email");
List<String> emailList = new ArrayList<>();
for (int j = 0; j < emails.getLength(); j++) {
    emailList.add(emails.item(j).getTextContent());
}
```

   Ez a megközelítés biztosítja, hogy minden adat megjelenjen a táblázatban, függetlenül attól, hány példányban szerepel az adott elem.

5. **Formázott, táblázatos megjelenítés**

   Az adatok kiírása minden entitás esetén egységes, jól olvasható, táblázatos formában történik, például:

```
System.out.printf("| %-12s | %-20s | %-60s | %-16s |%n", ...);
```

   Ez a formázás segíti az adatok áttekinthetőségét és a különböző entitások gyors összehasonlítását.

Az adatolvasás során tehát a DOM Parser segítségével strukturáltan, táblázatosan jelennek meg az XML-ben tárolt adatok, minden entitás típusra külön-külön, a Java program logikáját követve.

2.2 Adat-lekérdezés

Az XML dokumentumban tárolt adatokra különböző lekérdezéseket valósítottam meg Java nyelven, DOM Parser segítségével. Ezek a lekérdezések a `WO02D7DomQuery.java` állományban külön metódusokként szerepelnek, és a program futásakor mind a konzolra, mind egy kimeneti fájlba (`WO02D7_DomQueryOutput.txt`) kiírásra kerülnek. Az alábbiakban bemutatom a lekérdezések működését és tényleges eredményét.

1. **Vevők, akiknek egynél több email címük van**

   A `CustomersWithMoreThanOneEmail` metódus végigiterál az összes `Customer` elemen, megszámolja az egyes vevőkhöz tartozó `Email` elemeket, és csak azokat írja ki, akiknek több email címük is van. Az eredmény:

   ```
   Customers with more than one email:
   Customer ID: C001 has 2 emails.
   Customer ID: C002 has 2 emails.
   Customer ID: C003 has 2 emails.
   ```

2. **Könyvek, amelyek ára meghalad egy adott értéket, kiadói adatokkal**

   A `BooksWithPriceGreaterThan` metódus minden `Book` elem árát (`Price`) ellenőrzi, és ha az meghaladja a megadott küszöböt (itt: 10.0), akkor kiírja a könyv címét, árát, valamint a kapcsolódó kiadó nevét, alapítási évét és országát. Az aktuális kimenet:

   ```
   Books with price greater than 10.0:
   Book Title: The Great Gatsby, Price: 10.99, Publisher: Scribner, Founded: 1846, Country: USA
   Book Title: To Kill a Mockingbird, Price: 12.99, Publisher: HarperCollins, Founded: 1989, Country: USA
   Book Title: The Catcher in the Rye, Price: 11.99, Publisher: Penguin Books, Founded: 1935, Country: UK
   Book Title: The Hobbit, Price: 14.99, Publisher: HarperCollins, Founded: 1989, Country: USA
   ```

3. **Rendelések adott státusszal, vevő és dátum adatokkal**

   Az `OrdersWithStatus` metódus az összes `Order` elem közül csak azokat választja ki, amelyek státusza (pl. `Shipped`) megegyezik a keresett értékkel. A lekérdezés kiírja a rendelés azonosítóját, státuszát, dátumát, valamint a kapcsolódó vevő nevét és első email címét:

   ```
   Orders with status 'Shipped':
   Order ID: O001, Status: Shipped, Date: 2022-05-01, Customer: John Doe, Email: john.doe@example.com
   Order ID: O005, Status: Shipped, Date: 2022-09-10, Customer: Jane Smith, Email: jane.smith@example.com
   ```

4. **Szerzők, akik egynél több könyvet írtak**

   Az `AuthorsWithMoreThanOneBooks` metódus a `B-A` kapcsolóelemek alapján megszámolja, hogy egy szerző hány könyvhöz kapcsolódik, és csak azokat írja ki, akik legalább kettőhöz. Az aktuális eredmény:

   ```
   Authors with more than one book:
   Author: George Orwell has 3 books.
   Author: Harper Lee has 3 books.
   ```

A lekérdezések eredményei jól szemléltetik, hogy a DOM Parser segítségével nemcsak az adatok beolvasása, hanem összetettebb szűrések és kapcsolatok kezelése is könnyen megvalósítható Java nyelven.

2.3 Adat-módosítás

Az XML dokumentumban tárolt adatok módosítását szintén Java nyelven, DOM Parser segítségével valósítottam meg. Az adat-módosító műveletek a `WO02D7DOMModify.java` állományban külön metódusokként szerepelnek, és a program futásakor mind a konzolra, mind egy kimeneti fájlba (`WO02D7_DOMModifyOutput.txt`) kiírásra kerülnek. Az alábbiakban bemutatom a legfontosabb módosítási műveleteket és azok működését.

1. **Új vevő hozzáadása**

   Az `AddNewCustomer` metódus egy új `Customer` elemet hoz létre, amelyhez automatikusan generál egyedi `CustomerId`-t, valamint egy hozzá tartozó `Profile` elemet is. A metódus paraméterként kapja meg a vevő nevét és email címét, majd az XML dokumentum végéhez illeszti az új elemeket. A regisztráció dátuma automatikusan a mai nap lesz.

   ```
   printAndWrite(writer, AddNewCustomer(document, "Aurelia Starling", "aurelia.starling@example.com"));
   ```

2. **Könyv árának módosítása**

   Az `UpdateBookPrice` metódus megkeresi a megadott `BookId`-val rendelkező könyvet, majd módosítja annak `Price` elemét az új értékre. Ha a könyv vagy az ár elem nem található, a metódus hibaüzenetet ad vissza.

   ```
   printAndWrite(writer, UpdateBookPrice(document, "B001", 29.99));
   ```

3. **Szerző és kapcsolódó könyvek törlése**

   A `RemoveAuthor` metódus eltávolítja a megadott `AuthorId`-val rendelkező szerzőt, valamint az összes olyan `B-A` kapcsolóelemet, amely ehhez a szerzőhöz tartozik. Azok a könyvek, amelyekhez a törölt szerzőn kívül már nem tartozik más szerző, szintén törlésre kerülnek az XML-ből.

   ```
   printAndWrite(writer, RemoveAuthor(document, "A002"));
   ```

4. **Új email cím hozzáadása vevőhöz**

   Az `AddNewCustomerEmail` metódus megkeresi a megadott `CustomerId`-val rendelkező vevőt, majd hozzáad egy új `Email` elemet a megadott email címmel. A metódus visszajelzést ad a művelet sikerességéről vagy sikertelenségéről.

   ```
   printAndWrite(writer, AddNewCustomerEmail(document, "C001", "john.new@example.com"));
   ```

Az adat-módosítási műveletek során a DOM Parser lehetővé teszi az XML dokumentum szerkezetének dinamikus módosítását, új elemek beszúrását, meglévő elemek módosítását vagy törlését. A módosítások után a program a változtatásokat mind a konzolra, mind egy kimeneti fájlba írja, így a felhasználó azonnal visszajelzést kap az elvégzett műveletekről.