Jegyzőkönyv

Webes adatkezelő környezetek

Féléves feladat

XML

Készítette: **Emődi Máté**

Neptunkód: **GT43W5**

Dátum: **2025. december**

Miskolc, 2025

Tartalomjegyzék

Bevezetés

A féléves feladat keretében egy **Étterem Hálózat (Pizzázó) Nyilvántartó Rendszerét** terveztem meg és valósítottam meg XML technológiák segítségével. A választott téma egy valósághű vendéglátóipari modell, amely képes kezelni egy pizzérialánc alapvető adatait, a rendelési folyamatokat, a logisztikai hátteret és a vevői adatbázist.

A projekt célja, hogy bemutassa az XML alapú adatkezelés teljes folyamatát az elméleti tervezéstől (ER és XDM modellek) kezdve a strukturált adattároláson át (XML és XSD) egészen a szoftveres feldolgozásig (Java DOM). A rendszer központi eleme a termékkínálat (**Pizza**) és a vásárlók (**Vevő**) összekapcsolása a rendeléseken keresztül, biztosítva az adatok konzisztenciáját és visszakereshetőségét.

A feladat leírása

A féléves feladat két fő részből tevődik össze: egy adatmodellezési és egy szoftverfejlesztési szakaszból. A cél egy olyan integrált rendszer létrehozása volt, amely megfelel a szigorú tervezési és implementációs követelményeknek.

Az **első szakaszban** a rendszer adatmodelljének megtervezése volt a feladat. A kiírásnak megfelelően legalább 5 egyedet kellett definiálni, amelyek között különböző típusú kapcsolatok (1:1, 1:N, M:N) állnak fenn. A saját témámban 7 egyedet hoztam létre: Beszállító, Futár, Pizzázó, Pizza, Rendelés, Vevő és Bankkártya. Kiemelt követelmény volt a speciális tulajdonságok alkalmazása: a modellben szerepel összetett tulajdonság (pl. a Cím felbontása Város, Utca, Házszám, Irányítószám elemekre) és többértékű tulajdonság is (a Pizza Feltétjei). Az M:N kapcsolatot a Rendelés egyed valósítja meg, amely saját attribútumokkal (Dátum, Darabszám) is rendelkezik.

A tervezés során először az ER modellt készítettem el, amelyet ezt követően XDM modellé konvertáltam. Az XDM modellnél kritikus szempont volt a hierarchikus felépítés kialakítása úgy, hogy az elsődleges kulcsok (PK) és idegen kulcsok (FK) közötti kapcsolatot jelző vonalak ne keresztezzék egymást. Ezt egy logikus, sávos elrendezéssel valósítottam meg. A tervek alapján létrehoztam a validált XML dokumentumot, amelyben minden egyedből legalább két példányt rögzítettem a sokszínűség szemléltetésére. Az adatstruktúra helyességét és a típusosságot (pl. számformátumok, kötelező mezők) egy XML Schema (XSD) fájl biztosítja, amelyben saját típusokat és kulcs-kényszereket0 definiáltam a referenciális integritás megőrzése érdekében.

1.XML adatkezelő rendszer

1.1 Az adatbázis ER modell megtervezése

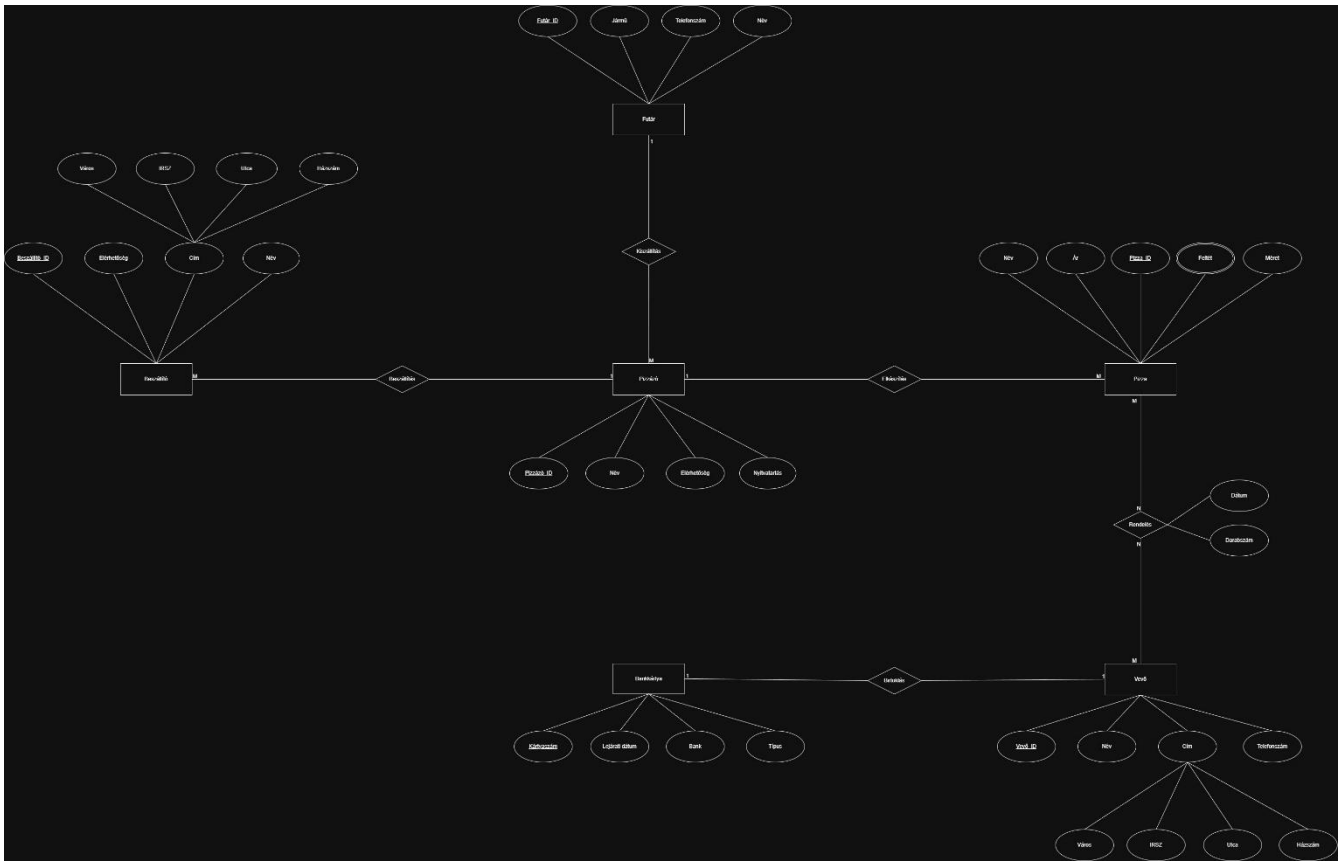
A rendszer adatmodelljének megtervezésekor a valós üzleti folyamatokat vettem alapul. A modell 7 egyedet tartalmaz, amelyek lefedik az étteremhálózat működését. A tervezés során ügyeltem a többféle kapcsolat (1:1, 1:N, M:N) és a speciális tulajdonságok (kulcs, összetett, többértékű) helyes alkalmazására.

Egyedek és tulajdonságaik:

- **Pizzázó:** A rendszer központi eleme. Tulajdonságai: *P_ID (PK)*, Név, Nyitvatartás, Telefonszám.
- **Futár:** A kiszállítást végző dolgozó. Tulajdonságai: F_ID (PK), Név, Jármű, Telefonszám.
- **Beszállító:** Az alapanyagokat biztosító partner. Tulajdonságai: *B_ID (PK)*, Név, Elérhetőség, valamint egy összetett *Cím* tulajdonság.
- **Pizza:** A rendelhető termék. Tulajdonságai: Pizza_ID (PK*)*, Név, Ár, Méret, és a *Feltét*, amely többértékű tulajdonság, hiszen egy pizzán több feltét is lehet.
- **Vevő:** A megrendelő. Tulajdonságai: *V_ID (PK)*, Név, Telefonszám, és szintén összetett *Cím* (Város, Utca, Házszám, IRSZ).
- **Bankkártya:** A fizetési eszköz. Tulajdonságai: *K_ID (PK)*, Lejárat, Bank, Típus.
- **Rendelés:** A Vevő és a Pizza közötti kapcsolatot megvalósító egyed.

Kapcsolatok:

- **1:1 (Egy-az-egyhez):** A *Vevő* és a *Bankkártya* között. Egy vevőhöz egy kártyát rendeltem a modellben.
- **1:N (Egy-a-többhöz):** A *Pizzázó* és a *Futár* között (egy pizzázóhoz több futár tartozik), illetve a *Pizzázó* és a *Beszállító* között.
- **M:N (Több-a-többhöz):** A *Vevő* és a *Pizza* között, amelyet a *Rendelés* egyed valósít meg. Fontos követelmény volt, hogy ennek a kapcsolatnak is legyen saját tulajdonsága: ezek a *Dátum* és a *Darabszám*.



1. ábra Az adatbázis ER modellje

1.2 Az adatbázis konvertálása XDM modellre

Az ER modell alapján elkészítettem a hierarchikus XDM modellt (XML Data Model). A konvertálás során a legfontosabb szempont az volt, hogy a vizuális megjelenítés átlátható legyen, és az elemeket összekötő vonalak (PK, FK kulcsok) NE keresztezzék egymást.

Megvalósítás menete: A gyökérelem (GT43W5_Etterem) alatt a "szomszédos elrendezés" elvét követtem. A logikailag összetartozó egyedeket egymás mellé helyeztem a fastruktúrában, így a hivatkozások (Idegen kulcsok) csak a közvetlen szomszédokra vagy közeli ágakra mutatnak.

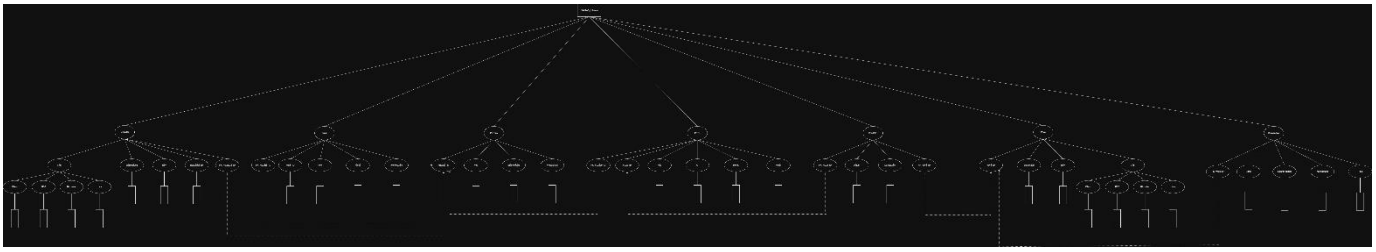
A bal oldalon a *Beszállító* és *Futár* elemek találhatók, amelyek a *Pizzázóra* hivatkoznak.

A *Pizzázó* alatt helyezkedik el a *Pizza*.

A struktúra jobb oldalán a *Vevő* és a hozzá tartozó *Bankkártya* található.

Középen, "hídként" szerepel a *Rendelés*, amely balra a *Pizzára* (Pizza_FK), jobbra pedig a *Vevőre* (Vevo_FK) hivatkozik.

Ezzel az elrendezéssel sikerült elérni a teljesen kereszteződésmentes vonalvezetést. A modellt szerkesztőprogrammal rajzoltam meg, szabványos fás jelöléssel.



2. ábra Az adatbázis XDM modellje

1.3 Az XDM modell alapján XML dokumentum készítése

A tervek véglegesítése után létrehoztam a validált XML állományt. A dokumentum felépítése szigorúan követi az XDM hierarchiát. A feladatkiírásnak megfelelően minden többszörösen előforduló elemből (Pizzázó, Rendelés, Vevő, stb.) legalább két példányt rögzítettem, hogy a lista-szerkezet jól látható legyen.

Kód felépítése és magyarázata: Az XML gyökéreleme a GT43W5_Etterem. Ezen belül sequence sorrendben helyezkednek el az egyedek. Az azonosítókat (PK) attribútumként (pl. P_ID="1"), a tulajdonságokat gyermekelemként (pl. <Nev>) tároltam. A kapcsolatokat idegen kulcs (FK) elemekkel valósítottam meg. Például a Rendelésnél látható, hogyan kapcsolódik össze a Pizza és a Vevő:

```
<Rendeles R_ID="500">
        <Datum>2025-11-27</Datum>
        <Darabszam>2</Darabszam>
        <Pizza_FK>30</Pizza_FK>
        <Vevo_FK>100</Vevo_FK>
</Rendeles>
```

1.4 Az XML dokumentum alapján XMLSchema készítése

Az adatok integritását és a szerkezeti szabályokat egy XML Schema (XSD) fájlban definiáltam. Itt határoztam meg a kötelező elemeket, az adattípusokat (pl. xs:int, xs:date) és a kulcsokat.

Saját típus használata: A kódismétlés elkerülése és a tisztább struktúra érdekében létrehoztam egy saját komplex típust sajatCimTipus néven. Mivel a *Beszállító* és a *Vevő* is rendelkezik Címmel (Város, Utca, Házszám, IRSZ), ezt a típust mindkét helyen fel tudtam használni.

```
<xs:complexType name="sajatCimTipus">
  <xs:sequence>
    <xs:element name="Varos" type="xs:string"/>
    <xs:element name="IRSZ" type="xs:string"/>
    <xs:element name="Utca" type="xs:string"/>
    <xs:element name="Hazszam" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Kulcsok és hivatkozások (Key/Keyref): Az xs:key elemekkel biztosítottam, hogy az azonosítók egyediek legyenek az egész dokumentumban. Az xs:keyref elemekkel pedig a referenciális integritást védem: a validátor hibát jelez, ha a Futár olyan Pizzázó ID-ra hivatkozik (Pizzazo_FK), amely nem létezik a Pizzazo elemek között.

2. DOM alapú adatkezelés (Java)

A feladat második részében egy Java nyelven írt alkalmazást készítettem, amely a **DOM** szabvány segítségével dolgozza fel az előzőleg elkészített XML állományt. A megoldás során nem használtam külső könyvtárakat, kizárólag a Java beépített javax.xml.parsers és org.w3c.dom csomagjait.

2.1 Adatolvasás

Az adatolvasásért felelős osztály (GT43W5DomRead) feladata a statikus XML fájl betöltése a memóriába, a fa-struktúra bejárása, és a tartalom strukturált megjelenítése.

Megvalósítás: A program első lépésként létrehoz egy DocumentBuilderFactory és DocumentBuilder példányt, amelyek segítségével beolvassa az XML fájlt. A beolvasás után lefuttattam a normalize() metódust, hogy az esetleges formázási hibákat (felesleges szóközök, sortörések) kiküszöböljem. A feldolgozás során a getDocumentElement().getChildNodes() metódussal kértem le a gyökérelem gyermekeit. Egy ciklus segítségével végigiteráltam a csomópontokon és csak azokat dolgoztam fel, amelyek típusa ELEMENT_NODE. A kiírásnál ügyeltem a blokk formátumra: külön kezeltem a fő elemeket, attribútumaikat és a beágyazott gyermekelemeket. Végül a Transformer osztály segítségével a memóriában lévő fát kimentettem egy új fájlba (GT43W5_XML_Read_Output.xml).

Lényeges kód részlet (Beolvasás és gyökérelem lekérése):

```
File xmlFile = new File("GT43W5_XML.xml");

DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

DocumentBuilder builder = factory.newDocumentBuilder();

Document doc = builder.parse(xmlFile);

doc.getDocumentElement().normalize();

System.out.println("Gyökér elem: " + doc.getDocumentElement().getNodeName());
```

2.2 Adat-lekérdezés

Az adatlekérdezést megvalósító osztály (GT43W5DOMQuery) célja, hogy információkat nyerjen ki az XML adatbázisból anélkül, hogy XPath kifejezéseket használna. A programban 4 különböző típusú lekérdezést valósítottam meg.

Megvalósítás: A lekérdezésekhez a doc.getElementsByTagName("ElemNeve") metódust használtam, amely visszaadja az adott nevű elemek listáját (NodeList). Ezt a listát ciklussal jártam be, és a feltételek vizsgálatával pl. if (varos.equals("Miskolc")) szűrtem ki a kívánt adatokat. Különlegesség a 4. lekérdezés, ahol két külön listát (Rendelések és Pizzák) kapcsoltam össze programozottan az idegen kulcsok (Pizza_FK) alapján, szimulálva ezzel az SQL-ből ismert JOIN műveletet.

Elvégzett lekérdezések:

- **Egyszerű listázás:** Az összes Pizzázó nevének kiírása.
- **Szűrés idegen kulcsra:** Azok a futárok, akik az 1-es ID-jú pizzázóhoz tartoznak.
- **Szűrés összetett elemre:** Azok a vevők, akiknek a címe (beágyazott elem) Miskolcon van.
- **Kapcsolt lekérdezés (JOIN):** A rendelések listázása úgy, hogy a Pizza ID helyett a Pizza neve jelenjen meg.

Lényeges kód részlet (Kapcsolt táblás lekérdezés logikája):

```
// Belső ciklus a Pizza nevének kikeresésére ID alapján
for(int j=0; j<pizzas.getLength(); j++) {
  Element pizza = (Element) pizzas.item(j);
  // Ha a Pizza ID-ja megegyezik a Rendelés FK-jával
  if(pizza.getAttribute("Pizza_ID").equals(pizzaFK)) {
    pizzaNeve = pizza.getElementsByTagName("Nev").item(0).getTextContent();
    break;
  }
}
```

2.3 Adatmódosítás

A módosító osztály (GT43W5DOMModify) demonstrálja, hogyan lehet a DOM fa tartalmát és szerkezetét megváltoztatni, majd az eredményt perzisztensen tárolni. A program 4 különböző módosítást végez.

Megvalósítás: A módosításhoz először megkerestem a célelemet (pl. ID alapján), majd a setTextContent() metódussal írtam felül az értékét. Strukturális módosításnál (új elem hozzáadása) a doc.createElement("UjElem") metódussal hoztam létre az új csomópontot, és az appendChild() metódussal fűztem hozzá a meglévő szülőhöz. A műveletek végén a TransformerFactory segítségével, formázott kimenettel (indentálással) mentettem el a változásokat a GT43W5_XML_Modify_Output.xml fájlba.

Elvégzett módosítások:

- **Árváltozás:** A 30-as ID-jú pizza árát 3500-ra növeltem.
- **Szövegcsere:** A 10-es ID-jú beszállító nevét átírtam.
- **Adatjavítás:** A 20-as ID-jú futár járművét "Autó"-ra módosítottam.
- **Struktúra bővítés:** Az 500-as rendeléshez hozzáadtam egy új <Surgos>IGEN</Surgos> elemet.

Lényeges kód részlet (Új elem hozzáadása):

```
if (rendeles.getAttribute("R_ID").equals("500")) {
  // Új elem létrehozása és hozzáadása a fához
  Element ujElem = doc.createElement("Surgos");
  ujElem.setTextContent("IGEN");
  rendeles.appendChild(ujElem);
}
```