JEGYZŐKÖNYV

Adatkezelés XML környezetben

Féléves feladat

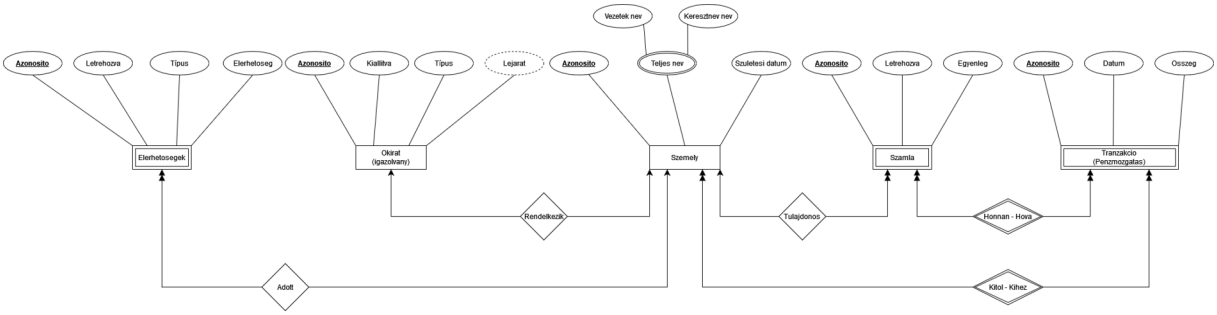
Banki tranzakciós adatbázis

Készítette: **Répási Gábor**
Neptunkód: **D51MXC**

Dátum: **2023.12.04**

Tartalomjegyzék

Bevezetés



1a. feladat

A feladatom egy banki adatbázis menedzselése. Mint minden banknak vannak ügyfelei. Az ügyfelek általában többen vannak, és a számlákból is több van. Ami a hétköznapi életben is előszokott fordulni, hogy egy természetes személy egyszer fordul elő egy banknál, de ennek a személynek lehet több számlája is, ez valósítja meg az **1:N** kapcsolatot.

Szintén egy ügyfélnek kell lennie egy hivatalos elérhetőségnek, amelyen mindig mindenkor utolérhető, ez is megvalósítja az **1:N** kapcsolatot. Opcionális, lehet nulla, egy vagy több is.

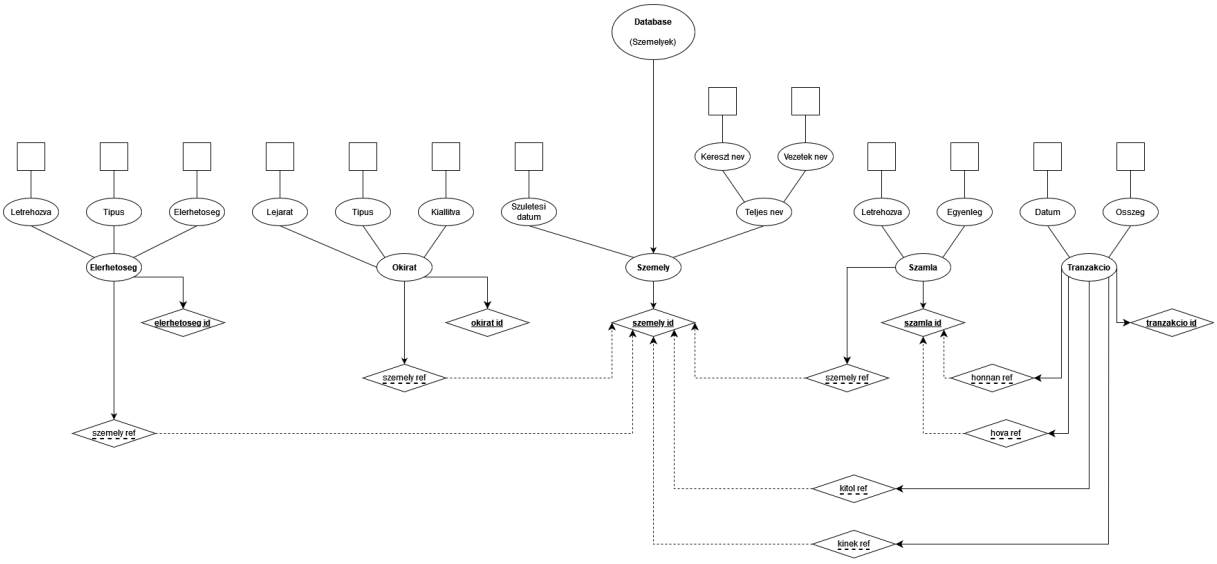
Egy személynek csak egyféle okirata lehet, például nem lehet több személyi igazolványa, csak egy, amely csak egy személyhez tartozhat, ez lesz az **1:1** kapcsolat. A lejárati mező opcionális, ugyanis nem minden okiratnak van lejárati ideje, ez jelöli a rajz is.

Nem utolsó sorban pedig maga a tranzakciót megvalósító, azt dokumentáló, illetve naplózó bejegyzések, ahol több személyhez is kapcsolódhat, attól függően, hogy kitől és kihez kerül a pénz, valamint melyik számláról melyikre, amely így több-több **N:M** kapcsolat.

A pénzmozgások nyomon követhetőek, ezáltal kimutatásokat is lehet belőle készíteni, amelyet szemléltetni is fogok a második feladat lekérdezési és szűrési feltételével.

Átrajzolás XDM típusú modellre

1b. feladat



(Kép és szerkeszthető terv-sablon mellékelve.)

Az ER modellből megrajzoltam az XDM modell-t.

Tartalmazza a fő „egyedülálló” egyedszerű bejegyzést, amely maga az adatbázis és a neve, amely az én esetemben csak a „database” elnevezést kapott.

A vonalak nem keresztezik egymást.

Az egyedek felett helyezkednek el a tulajdonságaik. Ezt követi az elsődleges kulcsmezőjük a rá következő sorban. A második sortól kezdve pedig az idegenkulcsok, szintén egymást követő sorban, elkülönülten és modell-szerű definiált sorrendben. Az egyedek mutatnak az elsődleges és idege kulcsikra is, az idegenkulcsok mutatnak más egyedek elsődleges kulcsaira.

XML fájl létrehozása

1c. feladat

Létrehoztam az XML fájlt.
Tartalmaz 5 egyedet:

- Személyek
- Elérhetosegek
- Okiratok
- Számlak
- Tranzakciók

Valamint az ehhez szükséges:

- Elsődleges kulcsok definicióját
- Idegenkulcsok definicióját

Részlet a fájl elejéből:

```xml
<?xml version="1.1" encoding="UTF-8"?>
<database xmlns="d51mxc"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="d51mxc XMLSchemaD51MXC.xsd">

    <!-- A személyek fogják használni a bankot -->

    <szemelyek>

    <!-- Első személy -->

        <szemely id="1">
            <teljes_nev>
                <vezetek_nev>Álmos</vezetek_nev>
                <kereszt_nev>Ádám</kereszt_nev>
            </teljes_nev>
            <szuletesi_datum>1999-09-09</szuletesi_datum>
            <elerhetosegek>
                <elerhetoseg idref="1" />
            </elerhetosegek>
        </szemely>
```

XSD fájl létrehozása

1d. feladat

Létrehoztam az XSD-sémafájlt. Létrehoztam a komplex-típusú egyedeket.
Tartalmaz 5 egyedet, amelyeknek mind van egy elsődleges kulcsa, kivétel nélkül.
A az első „Személyek” egyedeken kívül mind tartalmaz legalább egy idegenkulcsot.
A tranzakciókat naplózó „Tranzakciók” egyed 4 idegenkulcsot tartalmaz, amelyből 2-2 egy egyedre mutat.
Összeségében 7 db. Idegenkulcs definíciót tartalmaz.

Részlet az XSD-fájl elejéből:

```
1   <?xml version="1.1" encoding="UTF-8"?>
2   <xs:schema targetNamespace="d51mxc" elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
3
4       <!-- Minden személynek van egy opcionális elérhetősége -->
5
6     <xs:complexType name="elerhetosegek">
7       <xs:sequence>
8         <xs:element type="xs:string" name="letrehozva"/>
9         <xs:element type="xs:string" name="tipus"/>
10        <xs:element type="xs:string" name="elerhetoseg"/>
11      </xs:sequence>
12      <xs:attribute type="xs:string" name="idref" />
13      <xs:attribute type="xs:string" name="id" />
14    </xs:complexType>
```

Részlet az XSD-fájl idegenkulcs definícióiból:

```
111     <!-- Itt kezdődnek az idegen kulcsok kapcsolatainak a felépítése (7 reláció) -->
112
113         <xs:keyref name="elerhetosegRef" refer="szemelyId">
114           <xs:selector xpath="./szemelyek/szemely/elerhetosegek/elerhetoseg"/>
115           <xs:field xpath="@idref" />
116         </xs:keyref>
117
118         <xs:unique name="okiratRef" refer="szemelyId">
119           <xs:selector xpath="./okiratok/okirat/szemelyek/szemely"/>
120           <xs:field xpath="@idref" />
121         </xs:unique>
122
123         <xs:keyref name="szamlaRef" refer="szemelyId">
124           <xs:selector xpath="./szamlak/szamla/szemelyek/szemely"/>
125           <xs:field xpath="@idref" />
```

DOMRead

2a. feladat

A „DOMReadD51mxc.java” fájl nem nyitja meg közvetlen az XML fájlt, hanem átveszi a DOMQuery-től. Ez csak a fő funkciókat tartalmazza, elvégzi az adatkinyerést, majd visszaadja a query-nek a lekérdezett adatokat a „printAdat” függvénnyel.

```java
// Listázófüggvény
private static String getTextContent(Element parentElement, String childTagName) {
    NodeList nodeList = parentElement.getElementsByTagName(childTagName);
    if (nodeList.getLength() > 0) {
        return nodeList.item(0).getTextContent();
    } else {
        return "N/A";
    }
}
```

```java
// Szülő objektum lekérdezése
Element rootElement = document.getDocumentElement();

// Gyermek objektum(ok) lekérdezése
NodeList dataList = rootElement.getElementsByTagName(expression);

// Kilistázom a lekérdezett objektumokat
for (int i = 0; i < dataList.getLength(); i++) {
    Element databaseElement = (Element) dataList.item(i);
    String title = getTextContent(databaseElement, node);
    if(title != "N/A")
    {
        System.out.println(node + ": " + title);
    }

}
```

DOMModify

2b. feladat

A DomModifyD51mxc.java megnyitja az XML-fájlt és abból felépít a DOM fát, ebben végez módosításokat majd mentésre átadja a DOMWrite-nak amely az előzőkben leírtak szerint menti, majd kilistázza.

```
// Törlök egy elemet
removeElement(document, expression:"/database/szemelyek/szemely[@id=1]", elementName:"szuletesi_datum");

// Hozzáadok egy elemet
Node nodeAdd = document.createElement("szuletesi_datum");
nodeAdd.appendChild(document.createTextNode("1999.09.19"));
addElement(document, expression:"/database/szemelyek/szemely[@id=1]", nodeAdd);

// Létrehozom a szülő elemet
Node nev = document.createElement("teljes_nev");

Node adat1 = document.createElement("vezetek_nev");
adat1.appendChild(document.createTextNode("Átok"));

Node adat2 = document.createElement("kereszt_nev");
adat2.appendChild(document.createTextNode("Áron"));

// Összefűzöm a gyermek elemeket
nev.appendChild(adat1);
nev.appendChild(adat2);

// Felülírom a meglévőt
replaceElement(document, expression:"/database/szemelyek/szemely[@id=1]", nev);
```

```
<teljes_nev>
<vezetek_nev>Átok</vezetek_nev>
<kereszt_nev>Áron</kereszt_nev>
</teljes_nev>
```

Részlet a fa strúktúra listázásból (behúzásnak 4 db. szóköz karaktert állítottam be):

```
    <tranzakcio id="3">
        <szemelyek>
            <szemely idref="3"/>
            <szemely idref="1"/>
        </szemelyek>
        <szamlak>
            <szamla idref="3"/>
            <szamla idref="1"/>
        </szamlak>
                    <datum>2023-11-30</datum>
                    <osszeg>1000</osszeg>
    </tranzakcio>
    <tranzakcio id="4">
        <szemelyek>
            <szemely idref="2"/>
            <szemely idref="1"/>
        </szemelyek>
        <szamlak>
            <szamla idref="2"/>
            <szamla idref="1"/>
        </szamlak>
                    <datum>2023-11-30</datum>
                    <osszeg>1000</osszeg>
    </tranzakcio>
</tranzakciok>
```

DOMQuery

2c. feladat

A DomQueryD51mxc.java megnyitja az XML-fájlt és abból felépít a DOM fát. Majd feldolgozásra továbbítja a DOMRead osztálynak aki visszaadja kilistázott formában. Így működik a DOMRead és DOMQuery kéz a kézben.

```
public class DomQueryD51mxc
{
    Run | Debug
    public static void main(String[] args) {
        try {

            // Beolvasom az adatbázisfájlta projekt "resource" mappájából
            URL url = DomQueryD51mxc.class.getClassLoader().getResource("XMLD51MXC.xml");

            // Megnyitom a fájlt
            File file = new File(url.toURI());
            FileInputStream fileInputStream = new FileInputStream(file);

            // Használom a szerkesztő könyvtárat
            DocumentBuilderFactory documentBuilderFactory = DocumentBuilderFactory.newInstance();
            DocumentBuilder documentBuilder = documentBuilderFactory.newDocumentBuilder();
            Document document = documentBuilder.parse(fileInputStream);

            // Normalizálom a fájlom (előtte utána üres karektereket levágom stb...)
            document.normalize();
```

Az 5 db. Lekérdezésem eredménye:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
Kilistázom a neveket:
teljes_nev:
                                Álmos
                                Ádám

teljes_nev:
                                Balga
                                Béla

teljes_nev:
                                Cukor
                                Cecil

Kilistázom az elérhetőségeket:
eleres: +36 30 123 45 67
eleres: +36 70 987 65 43
eleres: Budapest 1111 Vizsgázó utca 4.
Kilistázom a számlák egyenlegeit:
egyenleg: 500000
egyenleg: 700000
egyenleg: 5000
Kilistázom a tranzakciókat:
osszeg: 2000
osszeg: 1000
osszeg: 1000
osszeg: 1000
Kilistázom az igazolványokat:
tipus: Személyi igazolvány
tipus: Jogosítvány
tipus: Személyi igazolvány
```

DOMWrite

2d. feladat

A DOMWriteD51mxc.java a DOMRead-hez hasonlóan funkciókkal bővíti ki a hozzá tartozó DOMModify osztályt. Feladata a módosítás utáni fájlba mentés és kiíratás: XMLD51MXC_uj.xml

```java
public static void write(Document document) throws TransformerException {

    document.normalizeDocument();

    TransformerFactory transformerFactory = TransformerFactory.newInstance();
    Transformer transformer = transformerFactory.newTransformer();
    DOMSource source = new DOMSource(document);
    StreamResult result = new StreamResult(new File("XMLD51MXC_uj.xml"));
    transformer.setOutputProperty(OutputKeys.INDENT, "yes");
    transformer.transform(source, result);

    System.out.println("Az XML fájl sikeresen kiíródott!");
}
```

Adatok kilistázására való funkció (fa struktúrába):

```java
public static void printNode(Source source, int depth) {
    try {
        TransformerFactory transformerFactory = TransformerFactory.newInstance();
        Transformer transformer = transformerFactory.newTransformer();

        transformer.setOutputProperty(OutputKeys.OMIT_XML_DECLARATION, "yes");
        transformer.setOutputProperty(OutputKeys.INDENT, "yes");

        // DOMSource átkonvertálása karakterlánccá
        StringWriter writer = new StringWriter();
        transformer.transform(source, new StreamResult(writer));
        String xmlString = writer.toString();

        System.out.println("XML fa struktúra:");

        // XML karakterlánc kiíratása fa struktúrába
        String[] lines = xmlString.split("\n");
        for (String line : lines) {
            for (int i = 0; i < depth; i++) {
                System.out.print("    "); // TAB
            }
            System.out.println(line);
        }

    } catch (Exception e) {
        e.printStackTrace();
    }
}
```