

JEGYZŐKÖNYV

Webes adatkezelő környezetek

Féléves feladat

A feladat címe:

Pizzéria kiszálltási részlegének adatbázisa

Készítette: Szabó Martin Attila

Neptunkód: F7M6MG

Dátum: 2025. November.28.

Miskolc, 2025

TARTALOMJEGYZÉK

Tartalomjegyzék

Bevezetés	3
A feladat leírása.....	3
1. Pizzéria kiszállítási részleg XML adatbázis tervezése.....	4
1.1 Az adatbázis ER modell tervezése	4
1.2 Az adatbázis konvertálása XDM modellre.....	5
1.3 Az XDM modell alapján XML dokumentum készítése.....	6
1.4 Az XML dokumentum alapján XMLSchema készítése.....	8
2. XML kezeléshez szükséges DOM implementáció	9
2.1 Adatolvasás (DOMRead)	10
2.2 Adat-lekérdezés (DOMQuery).....	10
2.3 Adatmódosítás (DOMModify).....	11
2.4 Főprogram és Menürendszer (Main)	13
Összegzés.....	14

Bevezetés

A féléves feladat célja egy Pizzéria kiszálltasi részlegének működését támogató adatkezelő rendszer megtervezése és implementálása XML alapokon. A választott témám egy minden nap logisztikai és értékesítési folyamatot modellez, amelyben nyomon követjük a megrendeléseket a vevőktől egészen a kiszállításig.

A feladat leírása

A rendszer célja, hogy átláthatóan tárolja a kiszállításhoz kapcsolódó legfontosabb adatokat. A nyilvántartás kezeli a **vevőket**, akik rendeléseket adhatnak le. Egy vevőhöz több telefonszám (alternatív telefonszám) és összetett címadatak tartoznak, ezzel a feladat követelményének egyik pontját valósítottam meg. A rendszer központi eleme a rendelés, amely összeköti a vevőt a kiszállítást végző futárral és a megvásárolt termékekkel (ez esetben pizzákkal). A **pizzák** különböző méretben és árban érhetők el, egy rendelés pedig több pizzát is tartalmazhat, illetve ugyanabból a pizzából többet is (mennyiségi kezelése- kapcsolatban darabszám). A kiszállítást **futárok** végezik, akikhez a rendszer saját szolgálati járműveket köt, melyet természetesen a vezető visz fel. A járművek műszaki adatait (műszaki érvényesség, üzemanyag) szintén tároljuk a flottakezelés céljából.

Az adatbázis tervezése során figyelembe vettet a valós üzleti igényeket: a futárok és járművek közötti szigorú hozzárendelést, a vevők rugalmas elérhetőségét, valamint a rendelések tételes rögzítését. Az adatbázis tervezését egy ER modellel kezdtem meg, melyben a kapcsolatokat valamint az adott elemek (pl. vevő) tulajdonságait tüntettem fel, az egyszerűbb implementálás érdekében. A megvalósítás során a modern XML technológiákat (XDM modell, XML Schema validáció) alkalmaztam az adatintegritás biztosítására.

1. Pizzéria kiszállítási részleg XML adatbázis tervezése

1.1 Az adatbázis ER modell tervezése

Mint azt korábban leírtam, az adatbázis logikai felépítését ER (Entity-Relationship) modell segítségével terveztem meg. A modell 5 egyedet (entitást) tartalmaz, amelyek lefedik a rendszer teljes működését.

Egyedek és tulajdonságaik:

Vevo: A megrendelők adatait tárolja.

Tulajdonságai: VevoID (elsődleges kulcs - PK), Nev (normál), Email (normál). A Cim tulajdonság összetett (Iranyitoszam, Varos, Utca), a Telefonszam pedig többéértékű tulajdonság, mivel egy vevőnek több elérhetősége is lehet.

Rendeles: A tranzakciók adatait tárolja.

Tulajdonságai: RendelesID (PK), Datum, Vegosszeg, FizetesMod.

Futár: A kiszállító személyzet adatait tárolja.

Tulajdonságai: FutarID (PK), Nev, Telefonszam, SzuletesiDatum.

Jarmu: A kiszállításhoz használt járművek, minden futárnak saját “céges” járműve van.

Tulajdonságai: Rendszam (PK), Tipus, MuszakiErvenyesseg, Uzemanyag.

Pizza: A terméktörzs. Tulajdonságai: PizzaID (PK), Nev, Ar, Meret.

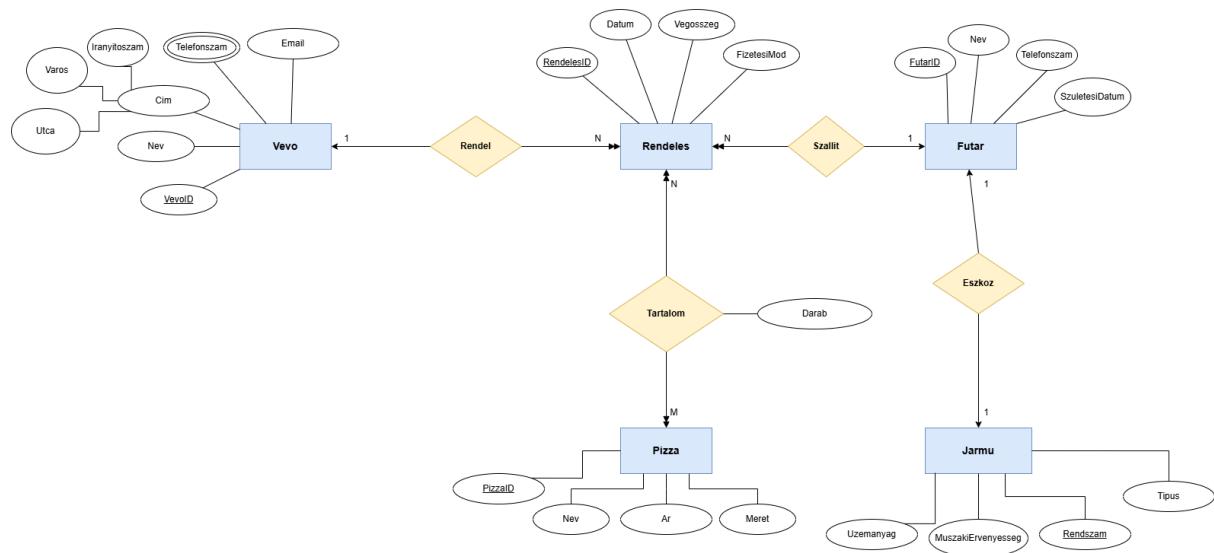
Kapcsolatok: A modellben többféle kapcsolatot valósítottam meg a feladatkiírásnak megfelelően:

1:N (Egy-a-többhöz): Egy vevőnek több rendelése lehet, de egy rendelés egy vevőhöz tartozik. Hasonlóan, egy futár több rendelést vihet.

1:1 (Egy-az-egyhez): Egy futárhoz pontosan egy szolgálati jármű tartozik (és fordítva).

M:N (Több-a-többhöz): A rendeles és a pizza között áll fenn. Egy rendelés több pizzát tartalmazhat, és egy pizza típus több rendelésben szerepelhet. Ennek a kapcsolatnak saját tulajdonsága a darab (mennyiség).

ER modell: Az általam készített ER modell az előzőleg ismertetett szempontok alapján készült el, az alábbi kép ezt prezentálja.



1.2 Az adatbázis konvertálása XDM modellre

Az ER modellt átültettem XDM (XML Data Model) szerkezetre, amely hierarchikus formában ábrázolja az adatokat. A tervezésnél elsődleges szempont volt az átláthatóság és a feladatkiírás azon feltétele, hogy a kapcsolatokat jelölő vonalak ne keresztezzék egymást.

A modell szerkezete: A gyökérelem a pizzeria. Ezen belül helyezkednek el a táblák.

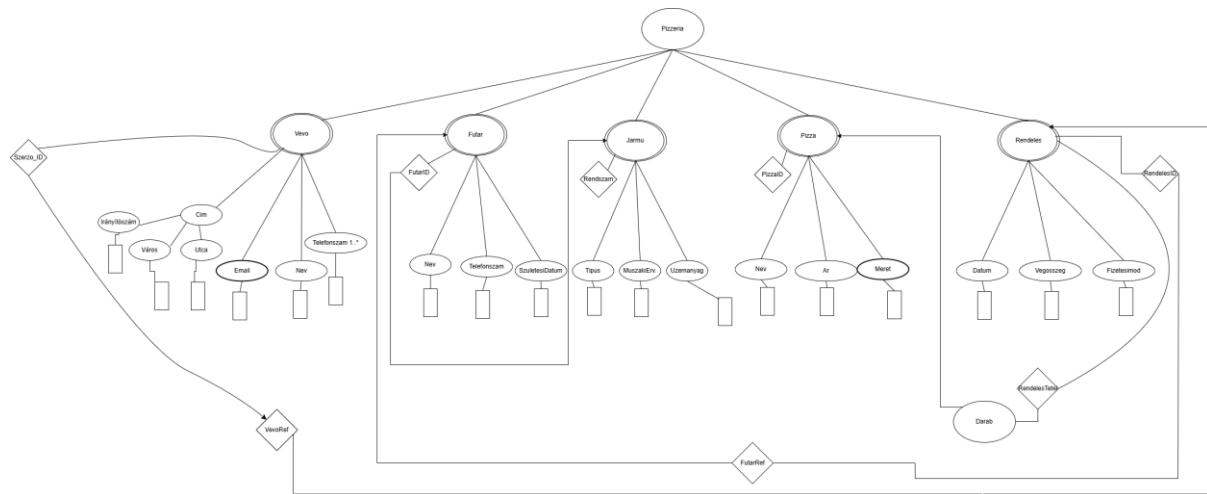
Az elrendezés során a vevo entitást bal oldalra, a futar és jarmu entitásokat jobb oldalra helyeztem.

Középen helyezkedik el a rendeles, amely így keresztezés nélkül tud hivatkozni (idegen kulccsal) mind a vevőre, mind a futárra.

A Pizza entitás a rendeles alatt kapott helyet, így a rendelés tételei fentről lefelé mutató nyíllal hivatkozhatnak rá.

A kapcsolatokat Idegen Kulcsok (FK) segítségével definiáltam, amelyek mindenkor az Elsődleges Kulcsra (PK) mutatnak.

Az alábbi ábrán az általam készített XDM modell látható, a feladatkiírás kritériumának (nyilak ne keresztezzék egymást) megfelelően.



1.3 Az XDM modell alapján XML dokumentum készítése

Az XDM modell alapján elkészítettem az adatbázishoz tartozó XML állományt. A kód valid az XML Sémára nézve. minden többszörösen előforduló elemből (pl. vevo, rendeles) legalább 2-3 példányt hoztam létre a demonstráció érdekében.

Kód részlet (Vevő entitás összetett és többértékű tulajdonságokkal):

```

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

    <Vovo VovoID="1">
        <Nev>Kovács János</Nev>
        <Cim>
            <Irányítószám>3515</Irányítószám>
            <Varos>Miskolc</Varos>
            <Utca>Egyetemváros út 1.</Utca>
        </Cim>
        <Telefonszám>06301112222</Telefonszám>
        <Telefonszám>06203334444</Telefonszám>
        <Email>kovacs.janos@uni-miskolc.hu</Email>
    </Vovo>

```

A több-a-többhöz kapcsolatot (rendeles-pizza) a rendelésen belül beágyazott RendelesTetel elemekkel oldottam meg, amelyek attribútumként hivatkoznak a pizzára (PizzaRef) és tartalmazzák a mennyiséget (kapcsolatokban darab).

Kód részlet (M:N kapcsolat megvalósítása):

```

<Rendeles RendelesID="1">
    <VovoRef>1</VovoRef>
    <FutarRef>1</FutarRef>
    <Datum>2025-10-15</Datum>
    <Vegosszeg>5700</Vegosszeg>
    <FizetesMod>Bankkártya</FizetesMod>
    <RendelesTetel PizzaRef="1" Darab="1"/>
    <RendelesTetel PizzaRef="2" Darab="1"/>
</Rendeles>

<Rendeles RendelesID="2">
    <VovoRef>2</VovoRef>
    <FutarRef>2</FutarRef>
    <Datum>2025-10-16</Datum>
    <Vegosszeg>5000</Vegosszeg>
    <FizetesMod>Készpénz</FizetesMod>
    <RendelesTetel PizzaRef="1" Darab="2"/>
</Rendeles>

```

Természetesen a kapcsolatok megvalósítása mellett a rendelések ugyan úgy tartalmaznak minden adatot, melyre szükségük van. A VovoRef és a FutarRef a rendeléshez kötött vevő és futár azonosítójára (VovoID és FutarID-re) hivatkozik.

A fájl teljes tartalma a feltöltött F7M6MG_XML.xml fájlban található, a többi entitással együtt megírva.

1.4 Az XML dokumentum alapján XMLSchema készítése

Az adatok integritásának és típusosságának biztosítására elkészítettem a F7M6MG_XMLSchema.xsd sémát. A séma definiálja a megengedett elemeket, attribútumokat és azok típusait.

Saját típusok definiálása: Készítettem egyszerű és összetett saját típusokat a kód újrafelhasználhatósága és a validáció érdekében.

cimTipus (ComplexType): A címek egységes szerkezetét (irányítószám, város, utca) írja le.

```
<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:sequence>
</xs:complexType>
```

arTipus (SimpleType): Az árak és összegek validálására szolgál, amely xs:integer alapú, de egy megszorítást tartalmaz, miszerint az érték nem lehet negatív (minInclusive = 0).

```
<xs:simpleType name="arTipus">
    <xs:restriction base="xs:integer">
        <xs:minInclusive value="0"/>
    </xs:restriction>
</xs:simpleType>
```

Kulcsok és hivatkozások: Az adatbázis konzisztenciáját az xs:key (elsődleges kulcs) és xs:keyref (idegen kulcs) elemekkel biztosítottam. Ez garantálja, hogy nem lehet olyan rendelést rögzíteni, amely nem létező vevőre vagy futárra hivatkozik.

```
<xs:key name="PizzaKey">
  <xs:selector xpath="Pizza"/>
  <xs:field xpath="@PizzaID"/>
</xs:key>

<xs:key name="RendelesKey">
  <xs:selector xpath="Rendeles"/>
  <xs:field xpath="@RendelesID"/>
</xs:key>

<xs:unique name="JarmuRendszamUnique">
  <xs:selector xpath="Jarmu"/>
  <xs:field xpath="@Rendszam"/>
</xs:unique>

<xs:keyref name="RendelesVovoRef" refer="VovoKey">
  <xs:selector xpath="Rendeles/VovoRef"/>
  <xs:field xpath=". />
</xs:keyref>

<xs:keyref name="RendelesFutarRef" refer="FutarKey">
  <xs:selector xpath="Rendeles/FutarRef"/>
  <xs:field xpath=". />
</xs:keyref>
```

A validálás során a séma ellenőrzi az XML dokumentum szerkezetét és adattartalmát, típushelyességet és kapcsolatokat, valamint például az egyedi arTipusnal még egy extra feltétel is meg lett adva az ellenőrzésnek. A teljes séma a repositoryba feltöltött F7M6MG XMLSchema.xsd fájlból található.

2. XML kezeléshez szükséges DOM implementáció

A feladat második részében egy Javaban íródott konzolos alkalmazást készítettem, amely a DOM szabvány segítségével dolgozza fel az XML adatbázist. A program képes az XML fájl beolvasására, strukturált megjelenítésére, specifikus adatok lekérdezésére és az adatok módosítására.

A projekt adatai:

- **Project name:** F7M6MGDOMParse
 - **Package:** f7m6mg.domparse.hu
 - **Class names:**
 - Main (Központi menürendsze)

- F7M6MGDOMRead (Adatolvasás és mentés)
- F7M6MGDOMQuery (Adatlekérdezés)
- F7M6MGDOMModify (Adatmódosítás)

2.1 Adatolvasás (DOMRead)

Az adatolvasásért a F7M6MGDOMRead osztály felel. A program a javax.xml.parsers csomag osztályait használja az XML fájl beolvasásához. A beolvasás után a teljes dokumentum szerkezetét rekurzív módon járjuk be, és formázva írjuk ki a konzolra.

Megvalósítás menete:

1. DocumentBuilderFactory és DocumentBuilder inicializálása.
2. Az XML fájl parse-olása és normalizálása.
3. A printDocument metódus segítségével a csomópontok típusának vizsgálata és kiírása.
4. A beolvasott adatstruktúra mentése egy új fájlba (F7M6MGXML_Saved.xml) a Transformer osztály segítségével.

Kód részlet: Ez a részlet mutatja be, hogyan írjuk ki a tag-eket és attribútumokat.

```
private static void printDocument(Node node, String indent) {
    // Csomópont típusának ellenőrzése
    if (node.getNodeType() == Node.ELEMENT_NODE) {
        Element element = (Element) node;

        // Elem neve és attribútumai
        System.out.print(indent + "<" + element.getnodeName());

        // Attribútumok kiírása, ha vannak
        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.getAttributeValue() + "\"");
            }
        }
        System.out.print(">");

        // Gyerek csomópontok feldolgozása
        NodeList children = element.getChildNodes();
        boolean hasElementChildren = false;
```

2.2 Adat-lekérdezés (DOMQuery)

Az adatbázisból történő információyerést a F7M6MGDOMQuery osztály végzi. A feladatkiírásnak megfelelően **XPath kifejezések használata nélkül**, tisztán a DOM API metódusaival (getElementsByTagName, item, getAttribute, getTextContent) valósítottam meg a lekérdezéseket.

A program 4 különböző lekérdezést futtat:

1. **Vevők listázása:** Kilistázza az összes vevő nevét és lakcímét (irányítószám, város, utca összefűzésével).
2. **Pizzák árlistája:** Megjeleníti az étlapot, azaz a pizzák nevét, árát és méretét.
3. **Futárok és járműveik:** Ez a lekérdezés összekapcsolja a Futár és a Jármű entitásokat. minden futárhoz megkeresi a hozzá tartozó szolgálati járművet.
4. **Rendelések statisztikája:** Összeszámolja a rendszerben lévő rendeléseket és kiszámolja azok összértékét.

Kód részlet (Futárok és járműveik):

```
NodeList futarList = doc.getElementsByTagName("Futar");
NodeList jarmuList = doc.getElementsByTagName("Jarmu");

for (int i = 0; i < futarList.getLength(); i++) {
    Element futar = (Element) futarList.item(i);
    String futarID = futar.getAttribute("FutarID");
    String nev = getElementTextContent(futar, "Nev");
    String telefon = getElementTextContent(futar, "Telefonszam");

    System.out.println("Futár ID: " + futarID);
    System.out.println(" Név: " + nev);
    System.out.println(" Telefon: " + telefon);

    // Járművek keresése, amelyek ehhez a futárhoz tartoznak
    for (int j = 0; j < jarmuList.getLength(); j++) {
        Element jarmu = (Element) jarmuList.item(j);
        String tulajdonos = getElementTextContent(jarmu, "TulajdonosFutar");

        if (tulajdonos.equals(futarID)) {
            String rendszam = jarmu.getAttribute("Rendszam");
            String tipus = getElementTextContent(jarmu, "Tipus");
            String uzemanyag = getElementTextContent(jarmu, "Uzemanyag");

            System.out.println(" Jármű: " + tipus + " (" + rendszam + ")");
            System.out.println(" Üzemanyag: " + uzemanyag);
        }
    }
    System.out.println();
}
```

2.3 Adatmódosítás (DOMModify)

Az XML adatbázis tartalmának módosítását a F7M6MGDOMModify osztály végzi. A program bemutatja új elemek létrehozását, meglévő adatok módosítását és a változtatások mentését.

A program 4 módosítást hajt végre:

1. **Új pizza felvétele:** Egy teljesen új pizza elemet hoz létre (ID: 4, Magyaros), beállítja a gyerekelemeit (Nev, Ar, Meret), majd hozzáfüzi a gyökérelemhez (appendChild).
2. **Ármódosítás:** Megkeresi a "Margherita" pizzát (PizzaID="1") és az árát 2500-ról 2800 Ft-ra módosítja a setTextContent metódussal.
3. **Új Vevő rögzítése:** Létrehoz egy komplex szerkezetű új Vevo elemet (ID: 4, Szabó Anna), amely tartalmaz összetett Cim elemet is.
4. **Fizetési mód frissítése:** A 2-es azonosítójú rendelés fizetési módját "Készpénz"-ről "Online átutalás"-ra módosítja.

A módosítások végén a program a teljes, frissített DOM fát elmenti a F7M6MGXML_Modified.xml fájlba. Ezeket természetesen meg lehetne írni olyan formában is, hogy a felhasználó válassza ki melyik ID-jű elemet szeretné átírni, illetve melyik attribútumát, viszont erre a feladat nem ad konkrét utasítást.

Kód részlet (Új pizza létrehozása és hozzáadása):

```
private static void modify1_AddNewPizza(Document doc) {
    System.out.println("--- 1. MÓDOSÍTÁS: Új pizza hozzáadása ---\n");

    // Gyökér elem lekérése
    Element root = doc.getDocumentElement();

    // Új Pizza elem létrehozása
    Element ujPizza = doc.createElement("Pizza");
    ujPizza.setAttribute("PizzaID", "4");

    // Pizza adatok hozzáadása
    Element nev = doc.createElement("Nev");
    nev.setTextContent("Carbonara");
    ujPizza.appendChild(nev);

    Element ar = doc.createElement("Ar");
    ar.setTextContent("3500");
    ujPizza.appendChild(ar);

    Element meret = doc.createElement("Meret");
    meret.setTextContent("32");
    ujPizza.appendChild(meret);

    // Pizza elem hozzáadása a dokumentumhoz
    root.appendChild(ujPizza);

    System.out.println("Új pizza hozzáadva:");
    System.out.println("  Pizza ID: 4");
    System.out.println("  Név: Carbonara");
    System.out.println("  Ár: 3500 Ft");
    System.out.println("  Méret: 32 cm\n");
}
```

2.4 Főprogram és Menürendszer (Main)

A szoftver belépési pontja és felhasználói felülete a Main osztály. Ez az osztály fogja össze az előzőekben ismertetett funkciókat (olvasás, lekérdezés, módosítás) egy egységes, konzolos menürendszerbe.

A megvalósítás egy végtelen ciklusban (while) fut, amely minden művelet után visszatér a főmenübe, amíg a felhasználó a kilépés opciót nem választja.

A főprogram funkciói:

Menü megjelenítése: Egy strukturált, szöveges felületet rajzol ki a konzolra, felsorolva az elérhető opciókat (1-4).

Opció beolvasás: A Scanner osztály segítségével olvassa be a felhasználó választását, és lekezeli az esetleges hibás beviteleket (pl. nem számot adnak meg).

Vezérlés: Egy switch-case szerkezet segítségével hívja meg a megfelelő osztályok main metódusait (runDOMRead, runDOMQuery, runDOMModify).

Összesített futtatás: A 4-es opció választása esetén egymás után, automatikusan lefuttatja minden három részfeladatot (Beolvasás -> Lekérdezés -> Módosítás), szüneteket beiktatva a jobb olvashatóság érdekében.

Összegzés

A munka a logikai tervezéssel indult, ahol az ER és XDM modellek segítségével megterveztem az adatbázist, különös figyelmet fordítva a bonyolultabb kapcsolatok (M:N) és adatszerkezetek (összetett, többétekű tulajdonságok) helyes ábrázolására. Ezt követte a fizikai megvalósítás, ahol a létrehozott XML adatbázis integritását és típusosságát egy saját típusokat és kulcsokat tartalmazó XML Schema (XSD) biztosítja.

A rendszer elérését egy Java nyelven írt, DOM szabványra épülő alkalmazással valósítottam meg. A program képes az XML állomány fastruktúrájának beolvasására, specifikus üzleti kérdések megválaszolására (lekérdezések), valamint az adatok módosítására és mentésére. A program modularitása és a konzolos menürendszer biztosítja a könnyű kezelhetőséget és az átláthatóságot.

A rendszer tesztelése során meggyőződtem arról, hogy az adatbázis validálása sikeres, a Java alkalmazás pedig hibamentesen kezeli a kapcsolatokat és az adatmanipulációs műveleteket.