JEGYZŐKÖNYV

Modern Adatbázis Rendszerek

Féléves feladat

Szobafoglalás

Todo App

Készítette: **Zavarkó Máté**

Neptunkód: **IN3BLK**

Tartalomjegyzék:

Beadandó témája

A beadandó témája egy olyan adatbázis, amely különböző hotelek szobafoglalásait kezeli. Megmutatja, hogy a személyek milyen és mennyi szobát foglaltak le az adott hotelben. Lekérdezhetjük a vevő adatait, ami alapján a számla is készül. Továbbá megtalálható a hotelek adatai és az értékeléseik.

1. Feladat

1a) Az adatbázis ER modell tervezése

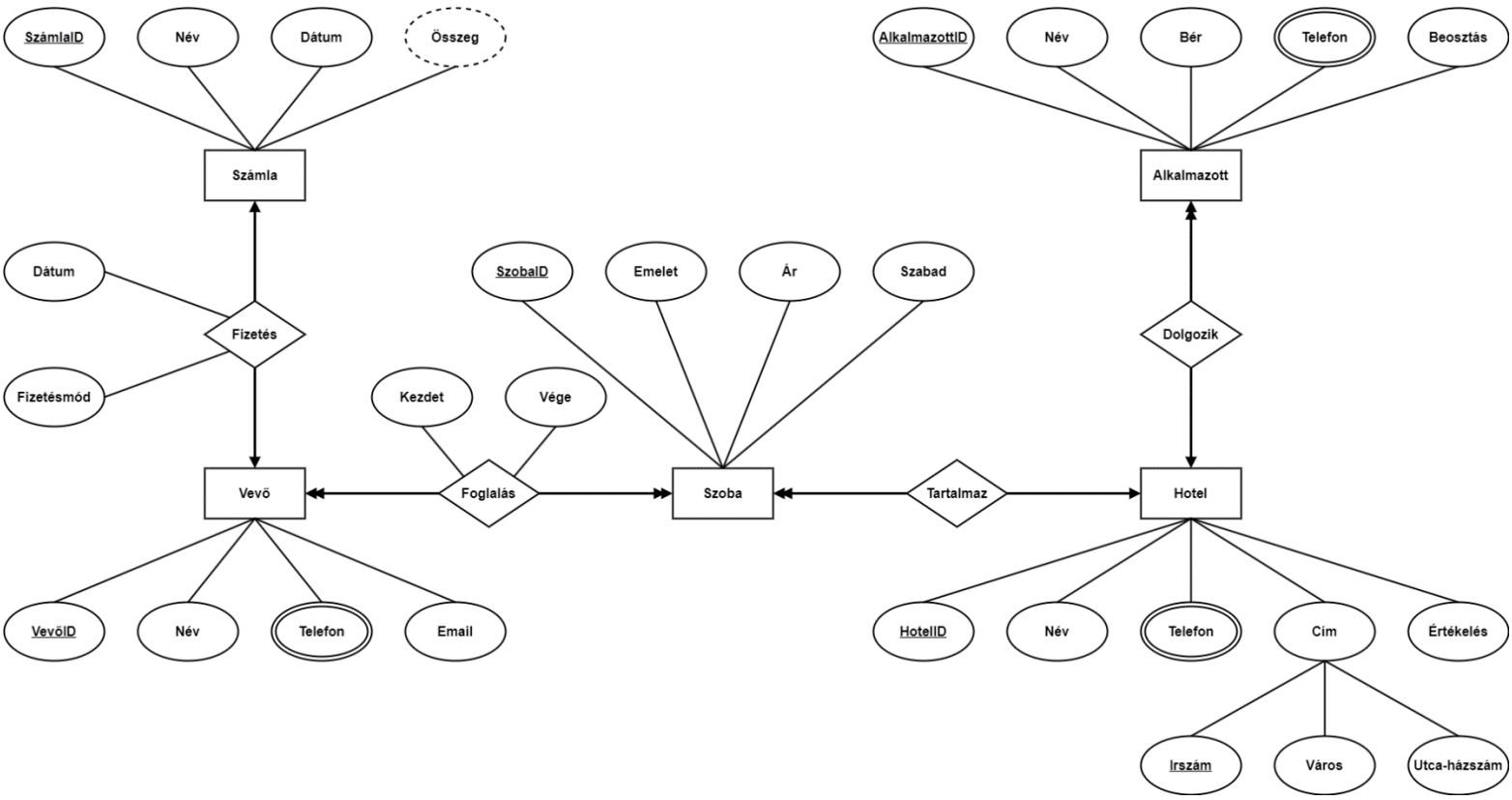
- **Vevő:**
  - *VevőID:* vevő elsődleges kulcsa
  - *Név:* vevő neve
  - *Telefon:* vevő telefonszáma, többértékű tulajdonság
  - *Email:* vevő email címe
- **Számla:**
  - *SzámlaID:* számla elsődleges kulcsa
  - *Név:* vevő neve, akihez tartozik a számla
  - *Dátum:* számla előállításának ideje
  - *Összeg:* fizetendő összeg
- **Szoba:**
  - *SzobaID:* szoba elsődleges kulcsa
  - *Emelet:* melyik emeleten található a szoba
  - *Típus:* szoba ágyainak mennyisége
  - *Ár:* az adott szoba ára egy éjszakára
  - *Szabad:* szoba foglaltsága
- **Hotel:**
  - *HotelID:* hotel elsődleges kulcsa
  - *Név:* hotel neve
  - *Telefon:* hotel telefonszáma, többértékű tulajdonság
  - *Cím:* hotel címe, összetett tulajdonság
  - *Értékelés:* hotel értékelése
- **Alkalmazott:**
  - *AlkalmazottID:* alkalmazott elsődleges kulcsa
  - *Név:* alkalmazott neve
  - *Bér:* alkalmazott bére
  - *Telefon:* alkalmazott telefonszáma
  - *Beosztás:* alkalmazott beosztása

Kapcsolatok:

- **Vevő és Számla (Fizetés)**
  Vevő és a *Számla* között egy-egy (1:1) kapcsolat van, mivel egy vevőhöz egy számla, illetve egy számla egy vevőhöz tartozik. Hozzátartozói a *Dátum*, hogy mikor történt a fizetés, és a *Fizetésmód*, hogy hogyan fizetett a vevő.
- **Vevő és Szoba (Foglalás)**
  Vevő és a *Szoba* között több-több (N:M) kapcsolat van, hiszen egy vevő kivehet több szobát és egy szobát több ember is kivehet. Hozzátartozói a *Kezde*t és a *Vége,* amik mutatják, hogy mettől meddig foglalta le a vevő a szobát.
- **Szoba és Hotel (Tartalmaz)**
  Szoba és *Hotel* között egy-több (1:N) kapcsolat van, ugyanis egy szoba egy hotelhez tartozik, de egy hotelhez több szoba is tartozik.
- **Hotel és Alkalmazott (Dolgozik)**

  Hotel és *Alkalmazott* egy-több (1:N) kapcsolat van, mivel egy hotelnek lehet több alkalmazottja, viszont egy alkalmazott egy hotelben dolgozhat.

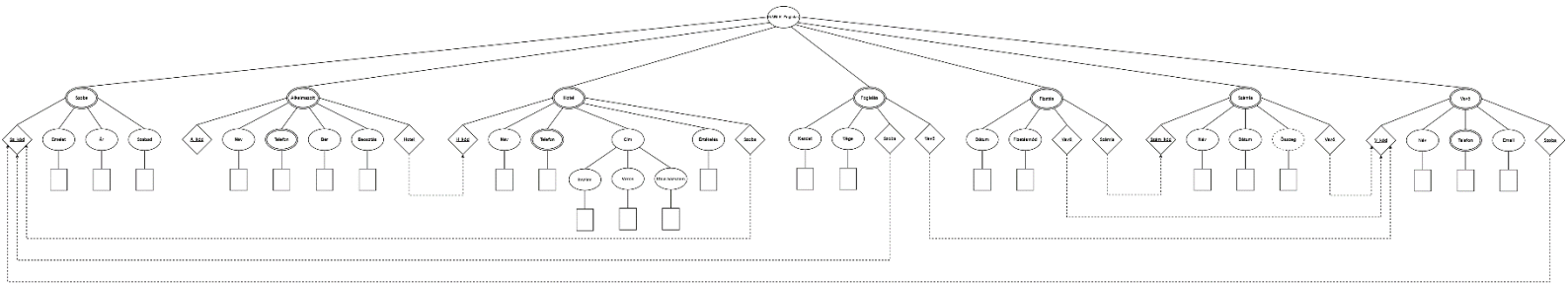
ER Modell:



1b) Az adatbázis konvertálása XDM modellre

Az XDM modell használatakor háromféle jelölést alkalmazhatunk. Az elemeket ellipszis ábrázolja, minden egyedből és a tulajdonságokból elem lesz. Az attribútumokat rombusz jelöli, melyek a kulcs tulajdonságokból erednek. A szöveget, amely az XML dokumentumban megjelenik, téglalap ábrázolja. Azok az elemek, amelyek többször is előfordulhatnak, dupla ellipszissel vannak jelölve. Az idegenkulcsok és a kulcsok közötti kapcsolatot szaggatott vonalas nyíl jelzi.

XDM Modell:



1c) Az XDM modell alapján XML dokumentum készítése

Az XML dokumentumot az XDM modell alapján elkészítettem, kezdve a root (gyökér) elemmel, amely az " IN3BLK_Foglalas ". Létrehoztam 3-3 példányt a gyermek elemekből, melyek attribútumai tartalmazzák a kulcsokat és idegenkulcsokat is. Ezt követően ezekhez az elemekhez létrehoztam a további gyermek elemeket is.

1d) Az XML dokumentum alapján XMLSchema készítése

Ezután elkészítem az XML-ben meghatározott típusokat és kulcsokat megszabó sémát, kigyűjtöm az egyszerű típusokat, elemeket, és ezek megszabásait beállítom. Ezt követően definiálom a saját, komplex típusaimat, ezekre is alkalmazom a megszabásokat, az egyszerű típusok felhasználásával. Ezen lépések után a gyökérelemtől kiindulva felépítem az XML struktúrát, meghatározom az elsődleges kulcsokat, valamint ezekhez az elsődleges kulcsokhoz tartozó idegenkulcsokat. Emellett az 1:1 kapcsolat megvalósításához használom a "Unique"-t is.

2. Feladat

2a) Adatolvasás

Ez a Java program az XML állomány tartalmának beolvasását és megjelenítését végzi el a DOM (Document Object Model) API segítségével. A program első lépésként betölti a megadott XML fájlt, majd normalizálja a dokumentumot, hogy az elemek egységesen kezelhetők legyenek. A gyökérelem és annak attribútumai kiírásra kerülnek a konzolra és egy kimeneti fájlba is. Ezután a program külön-külön végigmegy az XML-ben található főbb elemtípusokon, mint például a szobák, alkalmazottak, hotelek, számlák, vevők, foglalások és fizetések. Ezeket az elemeket rekurzívan dolgozza fel: minden elem nevét, attribútumait, valamint, ha van, a szöveges tartalmát kiírja megfelelő behúzással. A feldolgozott adatokat egy új fájlba is menti, ezzel biztosítva, hogy a dokumentum jól olvasható, formázott formában is elérhető legyen.

2b) Adatmódosítás

A fájl beolvasása és normalizálása után különböző típusú elemekre keres rá – például alkalmazottakra, hotelekre, vevőkre, foglalásokra és fizetésekre –, majd ezek közül meghatározott index alapján kiválasztott elemek egyes mezőit módosítja. Konkrétan egy alkalmazott bérét, egy hotel telefonszámát, egy vevő email címét, egy foglalás végdátumát, valamint egy fizetés módját írja át új értékekre. A módosításokat követően a program kiírja a teljes módosított XML dokumentumot a konzolra, így az eredmények azonnal láthatók.

2c) Adatlekérdezés

Az adatok lekérdezését megvalósító program különféle lekérdezéseket hajt végre egy XML dokumentumon. A dokumentum beolvasása és normalizálása után öt különböző lekérdezés kerül végrehajtásra:

Az első lekérdezés kiolvassa az 1-es azonosítóval rendelkező hotel adatait, mint például a nevét, telefonszámát, címét és értékelését. A második lekérdezés során azon alkalmazottak neve jelenik meg, akiknek a bére meghaladja a 300 000 forintot. A harmadik lekérdezésben a Royal Hotelhez tartozó vevők adatai kerülnek listázásra. A negyedik lekérdezés Kiss Anna szálláshoz kapcsolódó számlázási adatait gyűjti össze, például a foglalás időszakát, a szoba emeletét, valamint a fizetendő összeget. Az utolsó lekérdezés pedig azoknak a vevőknek a nevét és a fizetés dátumát írja ki, akik bankkártyával fizettek. Mindezek a lekérdezések DOM segítségével, a megfelelő csomópontok bejárásával és a szükséges attribútumok, illetve al-elemek szűrésével valósulnak meg.

2d) Adatírás

Az XML fájl létrehozásához DOM-alapú megközelítést alkalmaztam. Az osztály egy teljes struktúrájú XML dokumentumot generál, amely az *IN3BLK* sémának megfelelő adatokat tartalmaz. Az osztályban létrehozott főbb elemek: Szoba, Alkalmazott, Hotel, Szamla, Vevo, Foglalas és Fizetes. Minden elemhez több példány is hozzáadásra kerül, a hozzájuk tartozó attribútumokkal és alárendelt elemekkel együtt. A dokumentum végül fájlba íródik a Transformer osztály segítségével.

3. Feladat

Az elkészített Todo App egy egyszerű, webes felületen elérhető feladatkezelő, amely lehetővé teszi a felhasználók számára új feladatok hozzáadását, meglévők státuszának (kész/nem kész) módosítását, valamint feladatok törlését. Az alkalmazás kliens-oldali és szerver-oldali részekből áll, ahol a backend egy Node.js alapú API-t biztosít a műveletekhez, míg a frontend egy tiszta HTML, CSS és JavaScript alapú felhasználói felületet kínál.

3a) Technológiák

A projekt fejlesztése során az alábbi technológiákat használtam:

- Frontend:
  - HTML5
  - CSS3
  - JavaScript
- Backend:
  - Node.js
  - Express.js
  - MongoDB adatbázis

3b) Frontend

Feladat hozzáadása:

Egy űrlap két inputmezőből áll (Task title, Task description), valamint egy Add gombból. A gomb megnyomására az új feladat a backendre kerül mentésre, majd automatikusan frissül a megjelenített feladatlista.

Feladatlista:
A létrehozott feladatok listaként jelennek meg. Minden listaelem tartalmazza:

- A feladat címét (title)
- Leírását (description)
- Létrehozásának dátumát (createdAt)
- Egy pipálható négyzetet a feladat állapotának módosítására (completed)
- Egy törlés gombot a feladat végleges eltávolítására

3c) Backend

A szerver oldalon egy egyszerű REST API szolgálja ki a frontend kéréseit:

- **GET /tasks**:
  Visszaadja az összes feladatot.
- **POST /tasks**:
  Új feladatot hoz létre, amely a MongoDB adatbázisba kerül mentésre.
- **PATCH /tasks/:id**:
  Egy adott feladat készültségi státuszát módosítja.
- **DELETE /tasks/:id**:
  Egy adott feladatot töröl az adatbázisból.

3d) Működés folyamata

A felhasználó megad egy címet és opcionálisan leírást, majd rákattint az Add gombra.

A frontend POST kérést küld a backend API-nak, amely elmenti az új feladatot az adatbázisba.

Ezután a frontend automatikusan újratölti a feladatlistát a GET végpont meghívásával.

A meglévő feladatok mellett egy checkbox található: ha a felhasználó átállítja (kipipálja vagy visszaveszi) a státuszt, egy PATCH kérés megy ki a backendhez.

A Delete gombbal a felhasználó véglegesen törölheti a feladatot, ami egy DELETE kérésen keresztül történik.

Az oldal minden művelet után frissíti az aktuális állapotot a felhasználó felé.