Sapientia Erdélyi Magyar Tudományegyetem
Marosvásárhelyi Kar

Felvételi, 2026. szeptember
Szoftverfejlesztés mesterképzés

Javítókulcs

I. Adatbázisok

Uzletek (UzletID, UNev, ManSzemSzam);
RuhaCsop (CsopID, CsNev); // nadrágok, szoknyák, télikabát, stb.
Ruhak (RuhaID, RNev, MertEgys, CsopID);
Raktar (UzletID, RuhaID, MennyiségRaktaron, EladasiAr)
Szallitok (SzallID, SzNev, Helyseg, UtcaSzam);
Szallit (SzallID, RuhaID, Datum, SzallitasiAr);

a) Adjuk meg az Alexis nevű üzletben található ruhák nevét!

```
SELECT DISTINCT r.RNev
FROM Ruhak r
JOIN Raktar ra ON r.RuhaID = ra.RuhaID
JOIN Uzletek u ON ra.UzletID = u.UzletID
WHERE u.UNev = 'Alexis' AND ra.MennyisegRaktaron > 0;
```

b) Adjuk meg azon szállítók nevét, akik csak nadrágot szállítanak!

```
SELECT sz.SzNev
FROM Szallitok sz
JOIN Szallit szal ON sz.SzallID = szal.SzallID
JOIN Ruhak r ON szal.RuhaID = r.RuhaID
JOIN RuhaCsop rc ON r.CsopID = rc.CsopID
where rc.CsNev = 'nadrág'
GROUP BY sz.SzallID, sz.SzNev
HAVING COUNT(DISTINCT rc.CsopID) = 1;
```

c) Adjuk meg a legolcsóbb esőkabátot, melyből van raktáron! (UNev, RNev, MennyiségRaktaron formájában)

```
SELECT u.UNev, r.RNev, ra.MennyisegRaktaron
FROM Uzletek u
JOIN Raktar ra ON u.UzletID = ra.UzletID
JOIN Ruhak r ON ra.RuhaID = r.RuhaID
JOIN RuhaCsop rc ON r.CsopID = rc.CsopID
WHERE rc.CsNev = 'esőkabát' AND ra.MennyisegRaktaron > 0
ORDER BY ra.EladasiAr ASC
```

```
FETCH FIRST 1 ROWS ONLY;
```

d) Adjuk meg a Stefanis nevű üzletben található ruhák értékét csoportokon belül a következő formában: CsNev, Ertek

```
SELECT rc.CsNev, SUM(ra.MennyisegRaktaron * ra.EladasiAr) AS Ertek
FROM RuhaCsop rc
JOIN Ruhak r ON rc.CsopID = r.CsopID
JOIN Raktar ra ON r.RuhaID = ra.RuhaID
JOIN Uzletek u ON ra.UzletID = u.UzletID
WHERE u.UNev = 'Stefanis'
GROUP BY rc.CsNev;
```

e) Adjuk meg azon szállítók nevét, kik pontosan 3 féle ruha csoportot szállítanak!

```
SELECT sz.SzNev
FROM Szallitok sz
JOIN Szallit szal ON sz.SzallID = szal.SzallID
JOIN Ruhak r ON szal.RuhaID = r.RuhaID
GROUP BY sz.SzallID, sz.SzNev
HAVING COUNT(DISTINCT r.CsopID) = 3;
```

f) Adjuk meg minden üzlet esetén annak a ruha csoportnak a nevét, melyből a legnagyobb értékben található ruha az illető üzletben! (UNev, CsNev, Ertek formájában)

```
WITH CsoportErtekek AS (
   SELECT u.UNev, rc.CsNev,
        SUM(ra.MennyisegRaktaron * ra.EladasiAr) AS Ertek,
        RANK() OVER (PARTITION BY u.UzletID ORDER BY SUM(ra.MennyisegRaktaron *
ra.EladasiAr) DESC) as rnk
   FROM Uzletek u
   JOIN Raktar ra ON u.UzletID = ra.UzletID
   JOIN Ruhak r ON ra.RuhaID = r.RuhaID
   JOIN RuhaCsop rc ON r.CsopID = rc.CsopID
   GROUP BY u.UzletID, u.UNev, rc.CsopID, rc.CsNev
)
SELECT UNev, CsNev, Ertek
FROM CsoportErtekek
WHERE rnk = 1;
```

g) Módosítsd meg minden télikabát eladási árát, tedd egyenlővé a legolcsóbb Alexis üzletben levő télikabát árával.

```
UPDATE Raktar
```

```
SET EladasiAr = (
    SELECT MIN(ra.EladasiAr)
    FROM Raktar ra
    JOIN Ruhak r ON ra.RuhaID = r.RuhaID
    JOIN RuhaCsop rc ON r.CsopID = rc.CsopID
    JOIN Uzletek u ON ra.UzletID = u.UzletID
    WHERE u.UNev = 'Alexis' AND rc.CsNev = 'télikabát'
)
WHERE RuhaID IN (
    SELECT r.RuhaID
    FROM Ruhak r
    JOIN RuhaCsop rc ON r.CsopID = rc.CsopID
    WHERE rc.CsNev = 'télikabát'
);
```

h) Törölj minden olyan szállítót, aki több mint 2 éve nem szállított.

```
DELETE FROM Szallitok
WHERE SzallID NOT IN (
    SELECT DISTINCT SzallID
    FROM Szallit
    WHERE Datum >= ADD_MONTHS(SYSDATE, -24));
```

II. Objektumelvű programozás és szoftvertervezés

1. Példányszintű vs. osztályszintű adattagok

- Példányszintű (instance): minden objektumnak saját másolata van belőle.
- Osztályszintű (static): egyetlen közös érték, amin az összes példány osztozik, az osztályhoz tartozik.

```
public class Diak {
    private static int letszam = 0;   // osztályszintű: összes diák száma
    private String nev;               // példányszintű: saját név

    Diak(String nev) {
        this.nev = nev;
        letszam++;            // minden létrehozásnál nő a közös számláló
    }
    //…
}
```

2. Egy-a-sokhoz kapcsolat

```
+-----------+  1        *  +---------+
```

```
| Konyvtar  |------------->| Konyv  |

+-----------+              +--------+
```

```
public class Konyv {
    private String cim;
    private Konyv(String cim) { this.cim = cim; }
    //…
}

public class Konyvtar {
    private List<Konyv> konyvek = new ArrayList<>();
    public void addKonyv(Konyv k) { konyvek.add(k); }
    //…
}
```

Magyarázat: A Konyvtar osztály egy gyűjteményt (List<Konyv>) tárol referenciaként, így egyetlen Konyvtar objektum tetszőleges számú Konyv objektumot tartalmazhat — ez valósítja meg az egy-a-sokhoz kapcsolatot.

3. Egyszeresen láncolt lista

```
class Node {
    int adat;
    Node next;
    Node(int adat) {
        this.adat = adat;
        this.next = null;
    }
}

public class LinkedList {
    private Node head;

    public LinkedList() {
        head = null;                    // üres lista
    }

    public void insertFront(int ertek) { // beszúrás a fej elé
        Node uj = new Node(ertek);
        uj.next = head;
        head = uj;
    }

    public void print() {               // kiíratás
        Node cur = head;
        while (cur != null) {
            System.out.print(cur.adat + " ");
            cur = cur.next;
        }
```

```
        System.out.println();
    }

    public boolean search(int ertek) {   // keresés
        Node cur = head;
        while (cur != null) {
            if (cur.adat == ertek) return true;
            cur = cur.next;
        }
        return false;
    }
}
```

4. Nyílt-zárt elv (Open-Closed Principle)

Jelentése: egy osztály legyen **nyitott a bővítésre**, de **zárt a módosításra** — új funkció hozzáadásakor a meglévő, már tesztelt kódot nem kelljen megváltoztatni; ezt absztrakcióval (interfész/öröklés) érjük el.

```
interface Alakzat {
    double terulet();
}

class Kor implements Alakzat {
    double r;
    Kor(double r) { this.r = r; }
    public double terulet() { return Math.PI * r * r; }
}

class TeruletSzamito {
    double osszTerulet(List<Alakzat> alakzatok) {
        double sum = 0;
        for (Alakzat a : alakzatok){
            sum += a.terulet();
        }
        return sum;
    }
}
```

Magyarázat: ha új alakzatot (pl. Negyzet) kell hozzáadni, csak egy új osztályt kell írni, amely megvalósítja az Alakzat interfészt — a TeruletSzamito osztályt **nem kell módosítani**. Ez a nyílt-zárt elv: nyitott az új alakzatokra (bővítés), de zárt a meglévő logika átírására (módosítás).

III. Programozás és programozási nyelvek

```
void koktelrendezes(int *x, int n)
{
    int bal = 0;
    int jobb = n - 1;
    int i, cserehely, temp;

    while (bal < jobb)
    {
        cserehely = bal;

        for (i = bal; i < jobb; i++)
        {
            if (x[i] > x[i + 1])
            {
                temp = x[i];
                x[i] = x[i + 1];
                x[i + 1] = temp;

                cserehely = i;
            }
        }

        jobb = cserehely;

        if (bal >= jobb)
            return;

        cserehely = jobb;

        for (i = jobb; i > bal; i--)
        {
            if (x[i - 1] > x[i])
            {
                temp = x[i - 1];
                x[i - 1] = x[i];
                x[i] = temp;

                cserehely = i;
            }
        }

        bal = cserehely;
    }
}
```

Legjobb eset: $\Theta(n)$, mert rendezett tömb esetén az első menet után csere hiányában leáll.

Legrosszabb eset: $\Theta(n^2)$, mert lineárisan sok, lineáris hosszúságú menet válhat szükségessé.