Adatvezérelt rendszerek minta ZH és vizsgafeladatok megoldása

1. Az SQL szabvány milyen tranzakciós izolációs szinteket definiál, és ezek milyen párhuzamos hozzáférési problémákra adnak megoldást?

Az SQL szabvány a 4 gyakori konkurens esetre (piszkos olvasás, nem megismételhető olvasás, elvesztett módosítás, fantom rekordok) nyújt megoldásokat:

- Uncomitted read: mind a 4 probléma előfordulhat.
- Committed read: nincs piszkos olvasás.
- Repeatable read: nincs piszkos olvasás se nem megismételhető olvasás.
- Serializable: a 4 közül egyik probléma sem fordulhat elő.

Ugyanakkor ezek nem oldják meg az összes konkurenciából adódó problémát, az ACID tulajdonságok biztosításához még más módszerek is kellenek (pl. naplózás).

2. Az alábbi problémák megoldására ismertesse, a tanult technológiák, módszerek közül melyiket és hogyan használná. Választását indokolja. Ha mást nem jelöl a feladat, csak hatékony megoldás fogadható el.

a) Egy adatbázis táblában található adatokra olyan megkötéseink vannak, amelyek nem írhatóak le relációs modellben. Hogyan oldaná meg a táblába történő beszúrás esetén a megkötések ellenőrzését robosztus módon?

Pl. trigger felvételével, ami a beszúrásnál és módosításnál ellenőrzi a bevitt adatokat. Vagy egy eljárás felvételével, amit kötelező használni, viszont garantáltan csak érvényes adatokat ír az adatbázisba.

b) Egy újonnan induló fejlesztési projektünkben szeretnénk elérni, hogy az adatbázis lekérdezésekben ne legyenek elgépelések (pl. tábla neve, oszlopok neve). Hogyan lehetne a string alapú lekérdezéseket elkerülni?

- ORM használatával, amik lehetővé teszik a típusbiztos kódírást, illetve egy fejlett IDE képes elírásokat jelezni az osztályok neveiben.
- C# nyelven a programkódban hagyományos szövegalapú lekérdezések helyett LINQ-val.
- Szerver oldali programozással is megoldható, az SQL-ben írt eljárás előre megírható, lefordítható és tesztelhető, ami elírás esetén hibát jelezne, ráadásul az adatbázis ismerete mellett egy SQL-IDE könnyedén jelzi a helytelen tábla/oszlopneveket. A kliensprogramunk ezután a tárolt eljárásokat használná csak.

3. a) Ismertesse röviden a bitmap indexeket (milyen adatokra célszerű definiálni, hogyan használja az Oracle szerver, miért lesz hatékony).

A bitmap index egy 2 dimenziós bit tömb, aminek annyi sora van, ahány rekord a táblában, és annyi oszlopa, ahány különböző érték lehet az indexelt oszlopban. Egy adott sorban és oszlopban

akkor van 1, ha az adott rekord az adott mező adott értékét veszi fel. Olyan oszlopokra használjuk, ahol kevés különböző értéket lehet felvenni. Az olyan lekérdezéseket gyorsítja, ahol néhány oszlop adott értékeire keresünk rekordokat, a várt eredményhalmaz számossága nagy. A bitmap indexben való keresés hatékonyabban implementálható (pl. bitmaszkolással), mint amikor pl. szövegeket kellene összehasonlítanunk sok rekordnál.

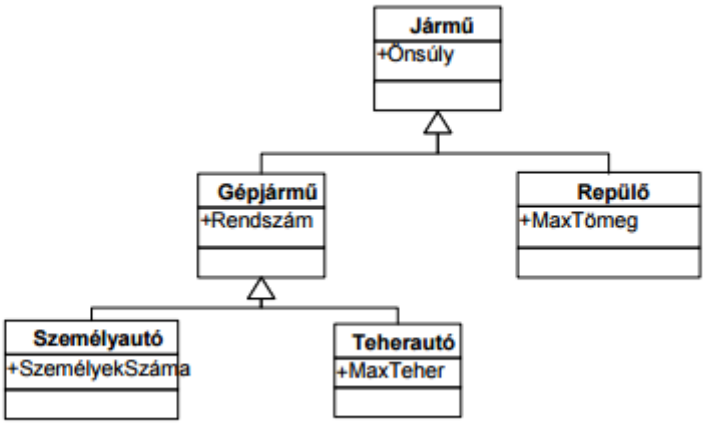
b) Tekintsen egy adatbázis táblát, amiben cégek azonosítóját, nevét és telephelyének megyéjét tároljuk. Mely mezőre érdemes bitmap indexet készíteni? Milyen lekérdezés fogja tudni használni az indexet (szóval vagy SQL kóddal illusztrálja)?

A telephely megyére, mert abból összesen kevés különböző érték lehet (pontosan 19). Az azonosítóra és névre nincs értelme, mert, ha egy adott céget keresünk, az azonosítóra vagy nevére rákeresve csak 1 találat lenne. Erre hatékonyabb egy B* fa, amiben pl. a név kezdőbetűi alapján binárisan tudunk keresni.

```
SELECT * FROM Telephely WHERE Telephely.megye = "Győr-Moson-Sopron";
```

Az index megléte nem módosítja az SQL lekérdezés szintaktikáját, ha a megye oszlopon van bitmap index, a kiértékelés gyorsaságában érezhetünk változást.

4. Adott az alábbi osztályhierarchia. Képezze le általános sémára (metadata driven). Mutassa be a létrejövő táblákat (értelmezze a tábla szerepét és tartalmát) és mutasson példa rekordokat egy személyautó példányra minden érintett táblában! Ha szükséges egészítse ki az osztályokat a megfelelő shadow információkkal!



Kell összesen 5 tábla: Class, Inheritance, Attribute, Attribute Type és Value. Az első 4 tábla az osztályok szerkezetének általános leírására való. Új rekordok felvételével tetszőleges osztályok felvehetők tetszőleges attribútumokkal, és köztük bármilyen öröklés megvalósítható. Az utolsó, Value tábla a konkrét példányok, objektumok értékeit tárolja. Ha egy osztály alapján példányosítunk egy objektumot, a Value táblába be kell írni minden értéket, ami azt az új objektumot jellemzi.
A lenti táblákban minden osztály szerkezete és kapcsolata, illetve két személyautó és egy repülő adatai láthatók.

Class	
CID	**Name**
1	Jármű
2	Gépjármű
3	Repülő
4	Személyautó
5	Teherautó

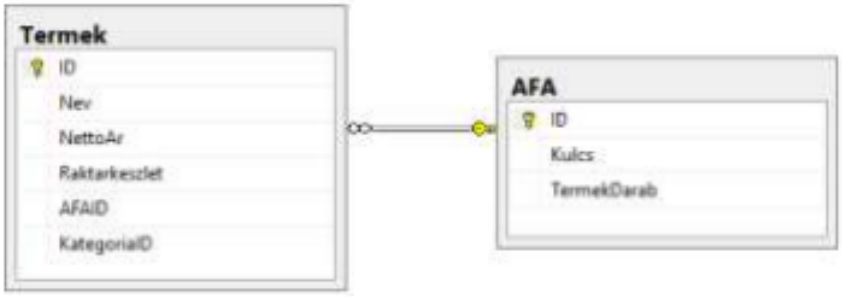
Inheritance	
Super	**Sub**
1	2
1	3
2	4
2	5

Attribute		
AID	**CID(FK)**	**ATID(FK)**
11	1	21
12	2	22
13	3	23
14	4	24
15	5	25

Attribute Type	
ATID	**Desc.**
21	Önsúly
22	Rendszám
23	MaxTömeg
24	SzemélyekSzáma
25	MaxTeher

Value		
OID	**AID(FK)**	**Value**
101	11	800 kg
101	12	EAC-196
101	14	4
102	11	2000 kg
102	12	JKO-337
102	14	8
201	11	15000 kg
201	13	1000 kg

5. Adott az alábbi adatmodell MSSQL Serveren. Az AFA tábla TermekDarab oszlopában az adott áfakulcshoz tartozó termékek darabszámát (ahány féle termék tartozik az áfához) szeretnénk tárolni. Készítsen triggert, ami karbantartja az AFA.TermekDarab mezőt!



A Termék táblára kell triggert készíteni, és ha változik (insert, delete, update) frissíteni kell az Afa tábla megfelelő (adott áfakulcs) TermekDarab oszlopát. A hatékonyság növelése érdekében összeillesztjük a táblákat, hogy tényleg csak ott legyen módosítás, ahol változás volt az adatokon.

```
CREATE TRIGGER AfaKarbantartas
    ON Termek
    FOR INSERT, UPDATE, DELETE
AS BEGIN
    UPDATE Afa
    SET TermekDarab =
        ISNULL(TermekDarab, 0)
        + ISNULL(ValtozasInserted, 0)
        - ISNULL(ValtozasDeleted, 0)
```

```
    FROM Afa a
        LEFT JOIN
            (SELECT afaid, COUNT(RaktarKeszlet) AS ValtozasInserted
            FROM inserted
            GROUP BY afaid) i
        ON a.ID = i.afaid
        LEFT JOIN
            (SELECT afaid, COUNT(RaktarKeszlet) AS ValtozasDeleted
            FROM deleted
            GROUP BY afaid) d
        ON a.ID = d.afaid
END;
```

6. Tekintse a fenti adatbázis modellt és annak Entity Framework leképzését (a kulcsok mentén automatikusan generált navigation property-kkel). Írjon C# nyelven Linq-to-Entities technológiával kódrészletet, amely törli azon Áfa kategóriákat, amelyhez nem tartozik termék.

Ha a kategóriához nem tartozik termék, a megfelelő navigation property listájában nem lesz egy elem se. Először keressük ki az ilyen tulajdonságú kategóriákat, majd töröljük mindegyiket.

```
using (var db = new DatabaseContext())
{
    var list = from a in db.Afa
               where !a.Termekek.Any()
               select a;
    db.Afa.RemoveRange(list);
    db.SaveChanges();
}
```