**3. Adatbázisbeli szabályok**

iDevice ikon **Bevezetés**

Ebben a fejezetben tisztázzuk a megszorítás fogalmát, mely az adatbázis kezelésére előírt valamely szabály beépítése az adatbázisba. Most készülünk fel arra, hogy az SQL-alapú adatbázisokban hogyan lehet biztosítani az előírt szabályok  és folyamatok érvényesítését.

**3.1. A határozatlan érték**

iDevice ikon

Minden logikai függvény ill. feltétel előtt tisztáznunk kell egy fontos, adattípustól független érték, az ismeretlen érték fogalmát.

Az SQL-alapú adatrendszerek standard adattípusait (számszerűek, karakteresek, logikai, dátumtípusúak) és alapműveleteiket megismertük az Adatbáziskezelés I keretében. Vannak esetek, amikor az adatbázis egyes mezőibe nem tudunk határozott értéket elhelyezni, ezért határozatlan tartalmúnak tekintjük. Szerencsétlen angol kifejezéssel: a mező tartalma NULL-érték (nem létező, ismeretlen érték).

Adattáblák oszlopaira megtilthatjuk az előfordulását.

A NULL-értéket tartalmazó kifejezések eredménye is ismeretlen lesz (kivétel a logikai vagy-művelet, melynek ha az egyik tagja igaz, akkor az eredmény már igaz).

**Példa:**

Adott a köv. tábla egy olyan adatbázis részleteként, melyben a diákok feladataik megoldására kapott pontokat tároljuk, ill. a plusz-pontokat, ha további, merőben más megoldásuk is értékelhető volt.

**MEGOLDÁS {diák\_kód, feladat\_szám, pont\_érték, plusz\_pont}**

Jelen esetben világos, hogy a leírók közül a pont\_érték bizonyára korlátozandó: legalább nulla és legfeljebb egy előírt (másik táblában tárolt) érték. A plusz\_pont pedig értelemszerűen nulla is lehet.

Abban az esetben, ha egy / az összes diák összteljesítményét kellene kiszámolni, akkor a pont\_érték+plusz\_pont aritmetikai összegeket kellene összegezni (pl. SUM függvénnyel az SQL-ben) adott diák / az egész tábla soraiban. Nos, ha valaki nem kapott plusz pontot, mert nem is volt második megoldása, akkor a mezőt nem hagyhatjuk kitöltetlenül, mert az összeg ismeretlenné válna. Tehát a plusz\_pont nem lehet NULL; alapértelmezett értéke pedig lehetne nulla.

**3.2. Megszorítások**

iDevice ikon

A megszorítás fogalma után a megszorítások csoportosítását ismerjük meg. Itt fogjuk megtanulni, hogy az adatbázisban hol és hogyan lehet gondoskodni bizonyos feltételek érvényre juttatásáról.

Az adatbázis kezelésének ellentmondásmentes ügyvitel szerint kell működniük, ezért az adatokra és az eseményekre vonatkozó szabályokat érvényesíteni kell az adatbázisban. Azért, hogy semmilyen adatkezelési esemény ne vezessen az adatok következetlenné válásához, az összes előírást és korlátozást be kell építeni az adatbázisba. A beépítésre került szabályt megszorításnak nevezzük.

A biztonságos adatbáziskezelés tehát a megszorításokkal körülvett adatbázis kezelését jelenti. A szabályok és adatfolyamatok együttesét üzleti logika fedőnéven is emlegetik.

Az SQL nyelv napjainkban már olyan lehetőségekkel rendelkezik, melyek segítségével programozás nélkül érvényesíthetjük az előírt szabályokat, vagyis megszorításokkal is bővíthetjük az adatbázist.

Idén a megszorításokat elegendő verbálisan - ugyanakkor logikailag helyesen -megfogalmazni; a köv. részben próbálkozhatunk mindezek SQL-beli megadásával.

A megszorítások logikai értékű függvények, amelyektől elvárjuk, hogy igazak legyenek. Az olyan módosításokat, amelyek a megszorításokat megsértik, a rendszer visszautasítja.

A megszorítások lényege a hivatkozási épség és az adatösszefüggések felügyelete minden adatkezelés alkalmával.

A hivatkozási épség megőrzése az egyedi kulcsok és külső kulcsok felállításával indul, de arról is gondoskodni kell, mi történjen a kulcsérték módosítása / törlése esetén a külső kulcs értékével. Ezen kívül a rejtett kapcsolatban álló táblák esetén is léteznek olyan hivatkozások, amiket nem lehet külső kulcs megadásával biztosítani.

Az adatösszefüggések közé tartoznak

* az olyan egyszerű feltételek, mint adott mező nagyobb nullánál, vagy a mező nem lehet NULL értékű,
* meg az olyan bonyolult feltételek is, mint adott mező kisebb, mint egy aktuális -, akár több táblából - számított érték.

Valójában már a mezők adattípusának meghatározásával korlátozásokat állítunk fel, de ezeket az alapvető szabályokat nem nevezzük megszorításoknak.

Az olyan egyszerű feltételek, amelyek egy tábla egy sorát érintik, könnyen beépíthető megszorítások. Ilyen esetben a táblába történő beszúrást ill. módosítást utasítja vissza a rendszer, ha a megszorítás nem érvényes.

Az olyan megszorítások, amelyek több sort vagy több táblát érintenek, a szó igazi értelmében összetettek. Ezek beépítése egy táblába helytelen, ui. csak ebbe a táblába való felvitel / módosítás váltja ki a megszorítás ellenőrzését. Ha ilyen megszorítást egyszerűen csak abba az egy táblába építenénk be, melynek valamely mezőjére fogalmaztuk meg, akkor a feltételben érintett másik sor vagy másik tábla sorának esetleges törlése esetén a feltétel elromolhat. Tehát önálló megszorításként érdemes megadni, ha azt szeretnénk, hogy az összes érintett táblában történő karbantartás esetén érvényben maradjon.

A megszorításokat a következő sorrendben célszerű felállítani:

* kulcsok megadása
* tulajdonság-értékekre vonatkozó korlátozások
* önálló megszorítások
* **3.2.1. Az egyedi kulcsok és az elsődleges kulcs megadása**
* iDevice ikon
* Csak egyetlen elsődleges, de akár több egyedi kulcsa lehet a táblának. Elsődlegesnek azt választjuk az egyediek közül, amire máshonnan hivatkozni kell.
* Ezek a kulcsok természetesen nem vehetnek fel ismeretlen értéket (NOT NULL), miután értékükkel csak egyszer fordulhatnak elő saját táblájukban.
* A kulcsok definiálása a konkrét táblába építendő be.

**3.2.2. Az idegen kulcsok definiálása**

iDevice ikon

A hivatkozási épség biztosítása érdekében adjuk meg a külső kulcsokat, miután ezek a kapcsolat hordozói a táblák között. Hiszen egy mező külső kulcsként csak olyan értéket vehet fel, amit kulcsként már tartalmaz. Ugyanakkor egy kulcsértéket addig nem szabad törölni, amíg külső kulcsként van rá hivatkozás.

A külső kulcs szerkezetének természetesen azonosnak kell lennie azon elsődleges kulcséval, melyre hivatkozik; de a kulcsot alkotó oszlopnevek egyezése nincs kikötve. Természetesen lehet több idegen kulcs is egy táblában.

Figyelem: ha az idegen kulcs nem kulcsszerepű (csak leíró), akkor nekünk kell eldönteni, hogy lehet-e nem kitöltött az ügyvitel szerint (ha nem, akkor az értékére megszorítást teszünk NOT NULL-al).

Megjegyzés: általábann elegendő a kapcsolt tábla nevét megadni a külső kulcs mellett, de előfordulhat, hogy a megvalósítás megengedi az elsődleges kulcstól különböző, másik egyedi kulcsra való hivatkozást is.

Arról is gondoskodni kell, hogy a szülő-tábla hivatkozott sorának törlése / módosítása esetén mi legyen a válaszlépés a gyerek-tábla hivatkozó sorában:

* ismeretlen értékkel töltse fel ott is
* alapértelmezett értékkel töltse fel ott is
* ott is törölje ki a hivatkozó sorokat / ott is módosítsa a hivatkozások értékét az új értékre
* ne tegyen semmit

az adatbáziskezelő rendszer.

**3.2.3. Tulajdonság-értékek korlátozása**

iDevice ikon

A tábla definiálásakor a kívánt tulajdonság (mező) adattípusa után gondoskodhatunk az oszlopérték helyességéről; egész egyszerűen logikai feltétel megfogalmazásával.

Például:

NOT NULL  
mező >= 0  
mező IN ('2A', '2C', '3C')  
mező > 0 AND mező < 6  
mező1 <> mező2

Feltételezzük, hogy a hallgatók emlékeznek az Adatbáziskezelés I tantárgy során ismertetett adattípusokra, műveletekre és priorításukra.

Megjegyzés:  
Ugyanitt tehetjük meg, hogy az alapértelmezés szerinti értékről (default-érték) gondoskodunk.

**Figyelem!**

A megadás ilyen elhelyezését legfeljebb sorfeltétel esetén válasszuk, mert csak abban a táblában kerül ellenőrzésre, amelybe beépítettük.

Az adatbáziskezelő rendszerre bízott vizsgálatok hatása az, hogy a beszúrást és módosítást a rendszer utasítsa vissza, ha a feltétel nem áll fenn. Törölni se engedjen, ha a feltétel előírja például a sorban szereplő valamely érték létezését.

Amint összett feltételt kell megfogalmaznunk egy mezőre vonatkozóan, ne a mező táblájának definiálásakor adjuk meg, mert csak ebbe a táblába történő beszúrás / módosítás esetén kerülne ellenőrzésre. A többi érintett sor esetleges törlésével ez a feltétel hamissá válhat.

Például:

mező <=  SUM(másik mező)  
mező IN (másik tábla valamely mezője)

Előfordulhat, hogy a több sort érintő feltétel olyan logikai feltételt tartalmaz, amely törlés hatására nem romolhat el. Ilyen esetben maradhat a megszorítás a táblában.

**3.2.3. Tulajdonság-értékek korlátozása**

iDevice ikon

A tábla definiálásakor a kívánt tulajdonság (mező) adattípusa után gondoskodhatunk az oszlopérték helyességéről; egész egyszerűen logikai feltétel megfogalmazásával.

Például:

NOT NULL  
mező >= 0  
mező IN ('2A', '2C', '3C')  
mező > 0 AND mező < 6  
mező1 <> mező2

Feltételezzük, hogy a hallgatók emlékeznek az Adatbáziskezelés I tantárgy során ismertetett adattípusokra, műveletekre és priorításukra.

Megjegyzés:  
Ugyanitt tehetjük meg, hogy az alapértelmezés szerinti értékről (default-érték) gondoskodunk.

**Figyelem!**

A megadás ilyen elhelyezését legfeljebb sorfeltétel esetén válasszuk, mert csak abban a táblában kerül ellenőrzésre, amelybe beépítettük.

Az adatbáziskezelő rendszerre bízott vizsgálatok hatása az, hogy a beszúrást és módosítást a rendszer utasítsa vissza, ha a feltétel nem áll fenn. Törölni se engedjen, ha a feltétel előírja például a sorban szereplő valamely érték létezését.

Amint összett feltételt kell megfogalmaznunk egy mezőre vonatkozóan, ne a mező táblájának definiálásakor adjuk meg, mert csak ebbe a táblába történő beszúrás / módosítás esetén kerülne ellenőrzésre. A többi érintett sor esetleges törlésével ez a feltétel hamissá válhat.

Például:

mező <=  SUM(másik mező)  
mező IN (másik tábla valamely mezője)

Előfordulhat, hogy a több sort érintő feltétel olyan logikai feltételt tartalmaz, amely törlés hatására nem romolhat el. Ilyen esetben maradhat a megszorítás a táblában.

**3.3. Mintapélda**

iDevice ikon

Életből vett rendszer nagyvonalú leírását lefedő adatbázissal foglalkozunk. Egyszerű ügyvitelt feltételezve megbeszéljük, milyen üzleti logikára épül az adatbázis kezelése, azaz milyen szabályok szerint marad ép az adatbázis. Ekkor jelenthető ki, hogy az adatbázis ellentmondásmentes ügyvitelt biztosít.

**3.3.1. Oktatás**

iDevice ikon

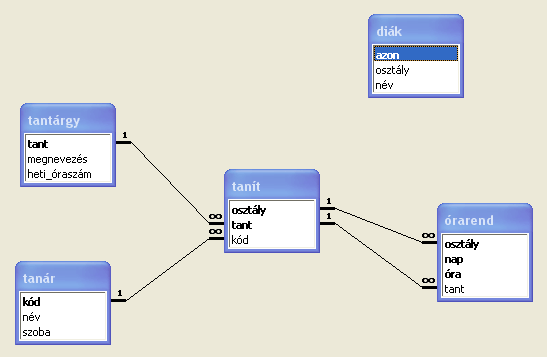
**A rendszer ismertetése**

Egy iskola folyó tanévében a diákok és tanárok viszonyát tartjuk nyilván értelemszerű törzsadatokkal az órarend kialakításáig. Az egyes tantárgyakat osztályoknak tanítják a tanárok, egy osztályba sok diák jár. Egy tanár több tárgyat is taníthat, és fordítva. Ugyanannak az osztálynak több tárgyat, illetve egy tárgyat több osztálynak is taníthat ugyanaz a tanár. A nap és óra sorszámozott típusú (1-5: a napok, 1-8: az órák), a tantárgy kódja beszédes (pl. MAT4: negyedikes matematika).

Kellenek a törzsek, a tanítás ténye és maga az órarend.

**DIÁK** {azonosító, név, osztály, ...}  
**TANÁR** {kód, név, szoba}  
**TANTÁRGY** {tant, megnevezés, heti\_óraszám}  
**TANÍT** {osztály, tant, kód}  
**ÓRAREND** {osztály, nap, óra, tant}

**Kapcsolatok**



15. ábra

*Gyakorlás céljából*[*töltse le az Access-adatbázist*](http://centroszet.hu/tananyag/adatbazis_alap/oktatas.mdb)*!*

*Figyelje meg, hogy az Access nem egy vonallal húzza be a kapcsolatot az összetett külső kulcs felől!*

Azonnal látjuk, hogy ebben az adatbázis-kezdeményben nincs osztály-törzs, és az egyes évfolyamok tematikája nincs tárolva. Ha az iskolatitkár fegyelmezetten indítja a tanévet, és azt kell berögzíteni, akkor ez az adatbázis megfelelő.

**Követelmények:**

* egy osztályba sok diák járhat
* egy tanár több osztályt taníthat
* egy tanár több tárgyat taníthat
* egy tanár egy osztálynak több tárgyat taníthat
* egy tanár egy tárgyat több osztálynak taníthat
* az órarend mind az osztály, mind a tanár szempontjából ütközésmentes kell legyen
* a heti óraszám korlátozása

**Fontos megjegyzés:**

Az azonosító felépítése szervezői feladat, de tudnunk kell, hogy egy olyan beszédes azonosító kiosztása, amely függ az osztálytól, nem jó (pl. osztály+sorszám). Ilyen esetben azoknak, akik nem bukotak meg, évente új azonosítót kellene osztani, ami rossz, mert kényelmetlen. Ilyen esetben lehet a beiratkozás éve+sorszám, vagy egy abszolut sorszám az azonosító.

A követelmények pontosítása után a következő szabályok beépítése kívánatos:

1. létező tantárgyat csak létező tanár taníthat
2. tanítani csak olyan osztálynak lehet, amelybe legalább 1 diák jár
3. egy osztálynak egy tárgyat egy tanár tanítja az adott tanévben
4. egy osztálynak adott nap adott órájában egy tárgyat taníthatnak
5. az órarend a tanár szempontjából is ütközésmentes (azaz a tárgyat tanító tanár egyszerre nem taníthat több helyen)
6. adott osztálynak adott tárgyat a tárgy heti óraszámában feltüntetett alkalommal kell szerepeltetni az órarendben
7. a tárgy azonosítójának felépítése feleljen meg az előírásnak
8. minden tantárgynak a heti óraszáma 1 és 10 közötti érték lehet
9. minden osztálynak legyen egy osztályfőnöke, de minden tanár legfeljebb egy osztálynak lehessen a főnöke

Nem megoldott viszont:

* a párhuzamos osztályok tanmenetének fixálása

Ezért itt két dolog nem lerendezett:

* az osztály értékének szabályozása az adatbevitelnél.

Ez a diák esetében lehetséges (pl. csak számjegy és betűkarakter a megengedett, vagy felsorolt konkrét osztályok közül való választás), de a tanít-tábla esetében azt kell leszabályozni, hogy csak a DIÁKban előforduló osztályhoz lehessen sort felvinni. Semmi garancia nincs arra, hogy minden osztályhoz valóban felvittek legalább 1 diákot.

* az évfolyam leválasztása az osztályból.

Az egyes évfolyamok egyforma tematikája nincs tárolva, tehát előfordulhat, hogy a TANÍT ellenőrzése után kimarad az egyik évfolyam minden párhuzamos osztályából valamelyik tárgy.

Tehát csak valóban fegyelmezett előmunkálatok után rögzített tanít-tábla esetén maradhat meg ez az adatbázis.

Mivel képzésünkben erőn felül (de megtérülően) foglalkozunk az adatbázistervezéssel, a 2. problémát (évfolyam-tematika) meg sem oldjuk. Az 1. probléma pedig nemcsak az osztálytörzs létrehozásával küszöbölhető ki, ha végiggondoljuk a rendszer ügyvitelét. Valószínűleg egy önálló funkcióként, nem pedig az órarendkészítési feladatok szintjén veszik fel az új diákokat (beiratkozás, új osztályok indítása). Bizonyára a tanévzárás is ilyen, hiszen akkor legalább a nem bukottakat tovább kell léptetni (osztályukban az évfolyam számát megnövelni). Tehát bízhatunk abban, hogy az összes diák az aktuális osztályával szerepel az adatbázisban ezen ügyviteli funkció idején. Az órarendkészítés tehát a tanévindítási funkció alatt elhelyezhető feladat, melynek ugyancsak van lezárása, ami után már nem módosítható.

**Megszorítások**

Jelen adatbázis-kezdemény kulcsainak és külső kulcsainak beépítése automatikus ellenőrzéseket állít be az adatbázisban, de bonyolult szabályok figyelését csak önálló megszorítások megadásával érünk el.

Nézzük, hogyan gondoskodhatunk a fenti 9 szabály betartásáról:

1. a TANÍT külső kulcsa a tant és a kód
2. a TANÍT osztály előfordul a DIÁK osztályai között
3. a TANÍT kulcsa osztály+tant
4. az ÓRAREND kulcsa osztály+nap+óra
5. naponta, óránként és kódonként 0 vagy 1 sor van az ÓRARENDben
6. osztályonként és tantárgyanként legfeljebb annyi sor van az ÓRARENDben, ahány a tárgy heti óraszáma
7. a tantárgy azonosítójának formátuma 4 karakteren: betű, betű, betű, számjegy, és a számjegy korlátozható
8. pl. 0< heti óraszám < 11
9. osztályfőnöki óra bevezetésével oldjuk meg a problémát

A kulcsok és külső kulcsok megadását {1),3),4)} egy táblaszerkezetben már ismerjük, SQL-megvalósítását a következő részben tanuljuk, mint ahogy a többi egyszerű vagy összetett megszorítás SQL-megfogalmazását is.

A 8) egy egyszerű feltétel a TANTÁRGY heti óraszámára

A 7) egyszerű, mezőre vonatkozó feltétel, mely a TANTÁRGY-bővítésnél kell, hogy érvényesüljön. Az összes többi tábla tant-mezője külső kulcs, tehát csak olyan lehet, ami a törzsben létezik.

Ezzel szemben a 2) önálló szabály kell legyen, ui. a TANÍT osztálya nem külső kulcs. Ha csak a TANÍT bővítés esetén ellenőriznénk, hogy az osztály csak olyan lehet, ami a DIÁKban létezik, akkor a DIÁKból való törlésekkel előfordulhat, hogy a feltétel elromlik; szemben a külső kulcsos hivatkozások megőrzésének biztosításával.

Az 5) biztosítása jó helyen lesz az ÓRAREND táblában, ui. csak annak bővítése / módosítása esetén kell ellenőrizni. Törléskor nem romolhat el a feltétel. Figyelem: jelen esetben az, hogy az ÓRAREND külső kulcsa az osztály+tant, önmagában nem elég a megszorításhoz, viszont szükséges a feltétel megfogalmazásához. Megj. ha a TANÍTból törlünk, akkor az itteni külső kulcs miatt lesz annak feltétele, hogy az innen is törlődjön.

A 6) beállítása gondoskodik minden ÓRAREND-bővítés/módosítás esetén arról, hogy ne ütemezzenek le több alkalmat, mint az előírt heti óraszám. Az órarendkészítés hosszabb folyamat, az ilyen ügyviteli feladat ellátása nem a menüpontból való kilépéssel fejeződik be, tehát az órarendkészítés lezárásának (őnálló funkció) feltétele lehet csak, hogy minden osztály összes tantárgya pontosan annyi sorban forduljon elő, ahány a heti óraszáma.

Mivel általában heti egy, órarendben elhelyezett osztályfőnöki órára van igény, az osztályfőnök szerepe {9)} ebben az adatbázisban ún. osztályfőnöki óra tantárggyal megoldható. Nem a tanárban kell tárolni, hogy ő melyik osztálynak a főnöke (beleértve, hogy ez lehet nem kitöltött érték), mégcsak nem is egy lehetséges osztálytörzsben, hogy ki az osztály osztályfőnöke, hanem tantárgyként kell felfogni. Legyen az OFO rövidnév az osztályfőnöki óra azonosítója, melyet a TANÍTban tárolva az osztályfőnökség tényét rögzítjük. Külön kell az ÓRAREND-bővítéskor ellenőrizni, hogy az adott tanár csak egy osztálynak „taníthassa" ezt a „tárgyat". (Néhány szabály fennállása ellenőrizhető a lekérdezések futtatásával.)

Ezek után nem nehéz megválaszolni az olyan kérdéseket, hogy mi annak a feltétele, hogy valamely táblába felvegyünk / valamely táblában módosítsunk / valamely táblából töröljünk egy sort. (A hivatkozási épség ellenőrzésére mód van a csatolt adatbázisban.)

**Rejtett kapcsolatok tárgyalása**

A diák nem kapcsolódik tisztán másik táblához.

Az osztály csak egy metszet-tulajdonság, de jelen esetben egy lekérdezés alkalmával a DIÁK és a TANÍT osztályon keresztül való összekapcsolása jelentéssel bír: minden diák minden tárgya szerepel majd benne azzal a tanárral, aki neki azt tanítja. (ld. SQL-lekérdezés)

Ha az évvégi osztályzatokat tárolnánk, akkor az egyes diákok tárgyankénti osztályzatát kellene „lerakni" pl. egy bizonyítvány-táblában. Ennek kulcsa csak akkor lehet az azonosító és tant*,* ha az előző évfolyamokat nem kell látni, vagyis a bukottak jegyei nem mondanak ellent a kulcsnak.

Ilyen bizonyítvány-tábla érdekes viszonyban áll a tanít-táblával:

**BIZONYÍTVÁNY** {azonosító, tant, jegy}  
**TANÍT** {osztály, tant, kód}

Itt a tant metszet-tulajdonságra alapozott összekapcsolás nem hordozna jelentést, hiszen a tárgyat tanuló összes diák „összeállna" a tárgyat tanuló összes osztállyal, akkor is, ha ő nem oda jár. Ekkor azt kell észrevenni, hogy a DIÁK-BIZONYÍTVÁNY helyes összekapcsolása áll kapcsolatban a TANÍT táblával (ez viszont nem jelölhető semmilyen eszközben). Lekérdezésnél kell észnél lenni, hogy a 3 tábla csak együtt alkalmas az összekapcsolásra (ld. SQL-lekérdezés).

Tehát a Bizonyítvány sorainak felvitele alkalmával nem elegendő ellenőrizni, hogy létező diák és létező tantárgy mellé egy 1-5 közötti jegyet rögzítsünk, hanem azt is, hogy a diák azt a tárgyat tanulta-e (tanították-e neki az osztályában).