Adatbázis normálformák

Ahhoz, hogy egy relációs adatbázis jól működjön, azaz gyors és precíz keresések megvalósíthatók legyenek benne, fontos, hogy legalább a 3. normálformában (3NF) legyenek az adatok. De mit jelent a normálforma? Matematikailag, halmazelméleti fogalmakkal leírható, de most inkább a felismerését nézzük át és a gyakorlati megvalósításra helyezzük a hangsúlyt.

Nézzük az alábbi adatszerkezetet:

munkatárs neve	**képzettsége(i)**	**születési dátum**
Sándor Boglárka	szoftver tervező Java programozó tesztelő	1990-04-04
Nagy Jenő	dokumentátor	2000-12-21

Kulcsként[1] szolgálhat itt a név és a születési dátum együttese, kisebb cégnél nagy valószínűséggel egyedi lesz ez a kettős (összetett kulcs, illetve composite key). A tábla 0. normálformában van (0NF) mert tartalmaz ismétlődő ismeretet. Ez itt a képzettség, mert nem függ az összetett kulcstól (nem határozza meg a kulcs). Hivatalosan megfogalmazva:

0NF típusú az adat akkor, ha van a táblában olyan másodlagos attribútum, amely a kulcstól funkcionálisan független.

Ebben az esetben a képzettségek csak úgy kereshetőek, hogy egy szöveges adaton belül keresgélünk, de vannak még vadabb megoldások is.

A fenti adatok 1. normálformába hozhatóak (1NF), ha biztosítjuk, hogy minden másodlagos attribútum a kulcstól (összetett kulcs!) funkcionális függésben van, például a következő módon:

munkatárs neve	**képzettség**	**születési dátum**
Sándor Boglárka	szoftver tervező	1990-04-04
Sándor Boglárka	Java programozó	1990-04-04
Sándor Boglárka	tesztelő	1990-04-04
Nagy Jenő	dokumentátor	2000-12-21

Hivatalosan megfogalmazva:

A reláció 1NF típusú, ha minden másodlagos attribútum funkcionálisan függ a kulcstól.

[1] Adatbázis tábla (reláció) esetén a kulcs azt a kiemelt attribútumot jelenti, amely a tábla egyes rekordjait egyértelműen meghatározza (elsődleges kulcs, PK). Ezért a kulcs a táblában mindig egyedi és nem lehet NULL értékű. Használhatunk összetett kulcsot is ebben az esetben kettő vagy több attribútum együtt teszi a rekordot egyedivé. Egy tábla elsődleges kulcsa idegen kulcsként (FK) szerepelhet egy másik táblában, ezzel biztosítva a táblák közötti kapcsolatot.

Igaz, hogy a tábla minden sora egyetlen attribútum értéket tartalmaz, de azért ez messze nem ideális, hiszen redundanciákat tartalmaz! Jó lenne, ha nem lenne benne részleges függés, ami a 2NF típus elérését biztosítja.

Van egy törvényszerűség, amit érdemes tudni:

Ha a reláció kulcsa nem összetett, azaz egyetlen attribútumból áll, akkor automatikusan 2NF típusú! Hozzunk létre egy ideális kulcsot, ami független a tárolt adatoktól, így semmiképpen nem változhat (a név például változhat egy személy élete során!).

A hivatalos megfogalmazás:

A reláció 2NF típusú, ha 1. normálformában van, és minden másodlagos attribútuma a reláció kulcsától teljesen függ.

kulcs	munkatárs neve	képzettség	születési dátum
1	Sándor Boglárka	szoftver tervező	1990-04-04
2	Sándor Boglárka	Java programozó	1990-04-04
3	Sándor Boglárka	tesztelő	1990-04-04
4	Nagy Jenő	dokumentátor	2000-12-21

Ebben a helyzetben is van lehetőség a redundancia megszüntetésére azzal, hogy a relációt szétbontjuk **három** relációra (a kohézió megvalósítása adatbázis szinten!):

munkatárs kulcs	munkatárs neve	születési dátum
1	Sándor Boglárka	1990-04-04
4	Nagy Jenő	2000-12-21

A munkatárs kulcsa és attribútumai

képzettség kulcs	képzettsége
1	szoftver tervező
2	Java programozó
3	tesztelő
4	dokumentátor

A képzettség kulcsa és attribútuma (leírása)

képzettség kulcs	munkatárs kulcs
1	1
2	1
3	1
4	4

Melyik munkatárshoz milyen képzettségek tartoznak?

A relációs adatbázisok már jól működnek, ha elérjük a 3. normálformát, amelynek a meghatározása a következő:

A reláció 3NF típusú, ha a 2. normálformát már elérte, és nincs olyan másodlagos attribútuma, amely tranzitív módon valamely kulcstól függ.

Ennek megértéséhez egy bonyolultabb példát kell előszednünk.

Nézzünk egy áruházat, ahol különböző árucikkeket tudunk megrendelni.A rendeléseket egy RENDELÉS relációban tároljuk, ahol a következő attribútumok felvitele történik. Az egyszerűség kedvéért most egy adott rendelésen egyetlen árucikk szerepel:

RENDELÉS {*rendelés_szám*, *vevőkód*, vevő_neve, vevő_címe, vásárlás_kelte, szállítási_határidő, *cikkszám*, árucikk_neve, egységár, rendelt_mennyiség, fizetendő}

A tábla erősen redundáns, mert több attribútumunk is van, amely nem teljesen függ a kulcstól. Felismerhetjük, hogy több attribútum kulcsszerű viselkedést mutat, mert meghatároznak további attribútumokat. Ez viszont nem engedhető meg egy relációs adatbázisban! Többlépésben megszüntethejük a redundanciát, de itt most a legfontosabb a tranzitív függőség felismerése és megszüntetés, mert enélkül nem érhetjük el a 3. normálformát.

A tranzitív függőség a táblában:

{ *rendelés_szám* }-> { *vevőkód* } -> { vevő_neve, vevő_címe }

Azaz a rendelés száma meghatározza a vevőkódot, az pedig a vevő nevét, címét. Végül négy táblához jutunk, a következő attribútumokkal:

ÁRUCIKK { *cikkszám*, árucikk_neve, egységár }

TÉTEL { *rendelés_szám*, *cikkszám*, rendelt_mennyiség }

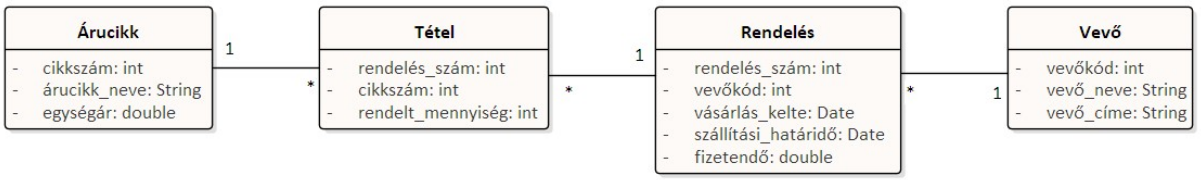
RENDELÉS { *rendelés_szám*, *vevőkód*, vásárlás_kelte, szállítási_határidő, fizetendő }

VEVŐ { *vevőkód*, vevő_neve, vevő_címe }

Az első attribútum minden esetben az elsődleges kulcs (PK), a többi dőlt betűvel szedett attribútum pedig idegen kulcs (FK). Azt is észrevehetjük, hogy most már akárhány tételt felvehetünk egy rendelésre, hiszen a kulcsokon keresztül egyértelmű, hogy melyik megrendeléshez tartozik. A tétel esetében speciális a kulcshelyzet, mert egy tipikus sok-sok kapcsolatot oldunk fel azzal, hogy összekapcsoljuk egy relációban (táblában) az árucikkeket és a rendeléseket.

Lényegében a következő Object Relational Mapping történt az OO adatszerkezet és az adatbázis táblák között:

Kapcsolódó osztályok



A megfelelő relációk/táblák: