Az adatbázis-szemlélet alapvetései

Az alábbiakban rövid elméleti összefoglaló következik olyasmiről, amiről az az ember benyomása, hogy egyszerű dolgok fölösleges bonyolítása, formalizmus, ami nemhogy javítaná, rontja a tisztánlátási esélyeinket. Mégis, ha arra gondolunk, hogy a relációs adatbázisok "felfedezése" és bevezetése a számítógépes adatnyilvántartások fejlődésének elmúlt harminc éves történetében milyen óriási jelentőséggel bír, akkor el kell fogadnunk azt, hogy aki egy kicsit komolyabb szinten szeretne foglalkozni ezzel a témakörrel, annak érdemes visszatekinteni egy kicsit a kiindulóponthoz: hogyan is gondolta ezt Edward F. Codd a 1970-ben?

1970-ben jelent meg Codd cikke, amelyben a relációs adatbázis-kezelőkre vonatkozóan 13 szabályt fogalmaz meg, 0-tól 12-ig számozva:

0. Az adatbázis-kezelő legyen képes minden adatbázis-műveletet reláció-műveletekre alapozva elvégezni.

1. Információs szabály
Az adatbázisra vonatkozó összes információt (beleértve a táblaneveket, mezőneveket) explicit módon táblázatokban kell tárolni.
2. Garantált hozzáférés
A relációs adatbázis minden adata garantáltan legyen hozzáférhető táblák kombinációi, elsődleges kulcsok és (vagy) oszlopnevek segítségével.
3. A NULL értékek egzakt kezelése
Az adatbázis-kezelőben legyen tökéletesen megoldva a NULL értékek (ismeretlen, vagy nem létező adatok) kezelése, úgy, hogy a NULL érték kezelése ne legyen összemosva az alapértelmezett értékek kezelésével.
4. Aktív, online relációs katalógus
Mivel az adatbázis szerkezeti leírását (logikai felépítésének adatait) is táblákban tároljuk, a szerkezetről az adatbázis-kezelő utasításaival tetszőleges információt meg lehessen kapni.
5. Programnyelv, ami minden részletre kiterjed
Legalább egy olyan átfogó programnyelvet kell kifejleszteni, amely tartalmaz megfelelő eszközöket az adatok definiálására, az adatokkal való manipulációkra, az adatok helyességének ellenőrzésére, a hozzáférési jogosultságok kezelésére, a tranzakciók lebonyolítására. A nyelv szintaktikai szabályai legyenek egyértelműen meghatározva.
6. Virtuális adathalmazok
Az adatbázis-kezelő rendszer legyen képes minden, a fizikai adathalmazból elvileg előállítható (virtuális) adathalmaz előállítására, karbantartására.
7. Halmaz-szintű beszúrás, módosítás, törlés
Az adatbázis-kezelőben legyen halmaz-szintű beszúrási, törlési és módosítási lehetőség. (Nem csak az adatok visszakeresését, hanem a felsorolt műveleteket is halmazműveletekre kell visszavezetni)
8.Fizikai adatfüggetlenség
Az alkalmazói programok és az ad hoc adatlekérések szintjén ne legyen érzékelhető, ha az adatok fizikai elérési módja, tárolási technikája módosul.
9. Logikai adatfüggetlenség
Az alkalmazói programokat (amennyire ez lehetséges) és az ad hoc adatlekéréseket ne befolyásolja a táblák szerkezetében történő változás.
10. Integritási függetlenség
Az adatbázis-kezelő nyelve legyen alkalmas az integritási szabályok definiálására. A szabályok legyenek tárolhatók, és a rendszer követelje meg a betartásukat
11. Adatelosztási függetlenség
Legyen lehetőség az adatbázis megosztására, a párhuzamos hozzáférésre.
12. Kiforgathatatlanság
Meg kell akadályozni, hogy az alacsony szintű nyelveken írt programok megsértsék az integritási szabályokat.

Integritás, konzisztencia

Codd meghatározásaiban többször szerepel az "integritás" szó. A szó jelentése: sértetlenség, épség jelent. Az adatok integritása azt jelenti, hogy az adatbázisban tárolt adatok helyesek, a valós rendszernek megfelelnek. Az adatbázis-kezelő programoknak kell olyan eszközrendszerrel rendelkezni, amelyek biztosítják az adatok helyességét és épségét-integritását.

Az adatintegritási szabályok közül hármat emelünk ki:

Tartomány-integritás
Az adatmezők értékei egy egyed meghatározott tulajdonságát adják meg. Minden mezőre meg kell határozni azt a szóbajöhető értékek halmazát, vagyis meg kell adni a mező értékkészletét. Az értékkészlet angolul: domain (tartomány), innen ered a tartomány-integritás kifejezés. A tartomány-integritás azt jelenti, hogy mező értéke csak az adott tulajdonsághoz tartozó érték-tartományból kerülhet ki. Nem teljesül az tartomány-integritás, ha például az adatbázis-kezelő megengedi, hogy a dolgozó fizetéseként negatív számot vigyünk be.

Azonosítási integritás
Az egyedeknek mindig azonosíthatóaknak kell lenniük. Az elsődleges kulcsként megjelölt mező (vagy mezőkombináció) értéke minden rekordban különböző kell, hogy legyen, és nem lehet NULL érték, adatbevitelnél kötelező kitölteni. (A NULL érték egy speciális kód, arra utal, hogy a mező értékét nem töltötték ki. Ez nem azonos azzal, hogy például egy numerikus érték helyett valaki 0-át ír, vagy egy szöveg helyére üres sztringet).

Hivatkozási integritás
Külön ki szokták emelni a hivatkozási integritás a fontosságát. A hivatkozási integritás megőrzése azt jelenti, hogy a kapcsolatokat meghatározó adatok sértetlenségét biztosítjuk. Sérül a hivatkozások integritása, ha például valamely táblából olyan rekordot törlünk, amelyre egy másik tábla hivatkozik.

Konzisztencia
A "konzisztens" szó jelentése: kiegyensúlyozott, ellentmondás-mentes, következetes. Az, hogy az adatbázis konzisztens az azt jelenti, hogy az adatbázis logikai szerkezete ép, a tárolt adatok megfelelőlek, az adatbázis nem tartalmaz fölösleges redundanciát, (így nem tartalmaz egymásnak ellentmondó adatokat). Az adatbázis konzisztenciáját az integritási szabályok biztosítják. Codd elképzelése szerint az adatbázis-kezelőnek biztosítania kell azt, hogy ezeket a szabályokat is tároljuk, és a programnyelvnek kell tartalmaznia olyan eszközöket, amelyekkel a szabályok betartása ellenőrizhető.

Codd tiszta és logikus követelményeit nem volt egyszerű betartani. Mindenesetre ezeknek az elveknek a megvalósítása kapcsán született meg az SQL nyelv. Egy programnyelv csak akkor tehető alkalmassá ilyen erős követelmények kiszolgálására, ha jó az adatmodell. (Ez a szemlélet a gyökere általában a mai programozási módszertannak is: az objektumorientált programozás nem az algoritmust, hanem az adatstruktúrát helyezi a középpontba.). A normalizálás az adatszerkezet átvizsgálásának algoritmusa. Lehet, hogy egy éles eszű tervező ösztönösen jó modellt alkot, de bonyolult, többszáz egyedtípust tartalmazó adatbázis esetében jobb ha ő is igénybe veszi a gépet. Ez pedig csak akkor lehetséges, ha a módszerekre gépies recepteket, algoritmusokat tudunk adni.

Normál formák, normalizálás

Funkcionális függőség
Ezt a fogalmat először Codd vezette be, amikor megfogalmazta a relációs adatbázisok elméleti alapjait. Az eredeti angol kifejezés: "functional dependency". A "functional" angol szó egyik jelentése: "működéshez tartozó". A "függőség", arra utal, hogy egy dolog valamilyen módon függ egy másik dologtól, az előbbi egy bizonyos szabály szerint meghatározza az utóbbit. Például a négyzet oldala meghatározza a négyzet területét. Hasonló függőségi viszonyt fedezhetünk fel az adatbázis relációiban is, ha arra gondolunk, hogy az egyedi kulcs egyértelműen meghatározza az egyed többi tulajdonságát. A "funkcionális függőség" kifejezés arra utal, hogy a vizsgált rendszer működését meghatározó, abból fakadó függőségi kapcsolatokról lesz szó. Az, hogy

például a személyi számtól "függ" egy ember minden további személyi adata, az országos lakónyilvántartási rendszer működését számos ponton meghatározó tény.

Azt mondjuk, hogy a B tulajdonság funkcionálisan függ az A tulajdonságtól, ha az A egyértelműen meghatározza a B-t.

A személyi szám egyértelműen meghatározza pl. a személy nevét. Ez azt jelenti, hogy egy adott személyi számhoz egy adott pillanatban csak egy név tartozhat. Fordítva ez nem feltétlenül igaz. Azt nem követeljük meg, hogy egy névhez, mindig ugyanaz a személyi szám tartozzon, hiszen számtalan Tóth Katalin van az országban, és más-más a személyi számmal. (A funkcionális függőség nem egy-egy értelmű kapcsolatot jelent).

Az előbbi példánál maradva: **a személyi szám** tulajdonságtól funkcionálisan függ a **személy neve** tulajdonság, mert ha egy táblázat két különböző sorában ugyanaz a személyi szám szerepel, akkor azokban a sorokban a személy neve is azonos. Ugyanez igaz az autó rendszámára és a kocsi típusára is.

A definíció, pontosítva:
Azt mondjuk, hogy a (B1, B2, ...Bn) tulajdonsághalmaz funkcionálisan függ az (A1, A2, ...Am) tulajdonsághalmaztól, ha minden olyan esetben, amikor a reláció két sorában a (A1, A2, ...Am) tulajonságok értékei megegyegyeznek, akkor ezekben a sorokban a (B1, B2, ...Bn) tulajdonságok értékei is megegyeznek. Jelölése: (A1, A2, ...Am)-> (B1, B2, ...Bn).

Első normál forma (1NF)
Egy munkahelyi adatbázis Dolgozó táblájában szeretnénk tárolni a dolgozók adatait. A Dolgozó reláció sémája:

Dolgozó (törzsszám, név, végzettség, beosztás)

A tervezés során kiderül, hogy az adatokat nem lehet a szokott módon egy táblázatba rendezni, mivel egy dolgozónak több végzettsége is lehet:

Törzsszám	Név	Végzettség	Beosztás
112	Varga Ferenc	mérlegképes könyvelő közgazdász	gazdasági igazgató
113	Tóth Irma	közgazdász mérnök rendszerszervező	irodavezető

Ez a táblázat nem reláció, ui. a Descartes-féle szorzat elemei minden tulajdonsághalmazból (így a végzettségek közül is) csak egyet tartalmazhatnak. A táblázatot úgy alakíthatjuk át relációvá, hogy az egyes végzettségek mellett a dolgozó többi tulajdonságát megismételjük. Így a táblázat már eleget tesz a relációkkal szemben az előző leckében megfogalmazott követelményeknek.

Dolgozó (törzsszám, végzettség, név, beosztás)

Törzsszám	Végzettség	Név	Beosztás
112	mérlegképes könyvelő	Varga Ferenc	gazdasági igazgató
112	közgazdász	Varga Ferenc	gazdasági igazgató
113	közgazdász	Tóth Irma	irodavezető
113	mérnök	Tóth Irma	irodavezető
113	rendszerszervező	Tóth Irma	irodavezető

A reláció azonosítója a törzsszám és a végzettség együtt. Az azonosító a többi tulajdonságot egyértelműen meghatározza.
Akkor mondjuk, hogy egy reláció első normál formában van, ha a reláció minden másodlagos tulajdonsága funkcionálisan függ az azonosítótól.

A Dolgozó reláció első normál formában van. Azt azonban látnunk kell, hogy a reláció rendundanciát tartalmaz, ebben a formájában nem lehet végleges megoldás.

Teljes függőség, részleges függőség
Azt mondjuk, hogy a B tulajdonság funkcionális teljesen függ A=(A1, A2, ...Am) tulajdonsághalmaztóll, ha az A tulajdonsághalmaz funkcionálisan meghatározza B-t, de A-nak nincs olyan részhalmaza, amelyre ugyanez igaz volna. Ha B tulajdonságot az A valamely részhalmaza is meghatározza, akkor azt mondjuk, hogy B funkcionális részlegesen függ az A tulajdonsághalmaztól.

Második normál forma (2NF)
Akkor mondjuk, hogy egy reláció második normál formában van, ha első normál formában van, és minden másodlagos tulajdonság teljesen függ a elsődleges kulcstól.
A Dolgozó relációban az elsődleges kulcs egy része, a törzsszám, egyértelműen meghatározza a dolgozó nevét és beosztását. Az utóbbi tulajdonságok tehát részlegesen függnek az azonosítótól. A reláció tehát nincs 2NF-ben. Emeljük ki az azonosítónak azt a részét, amely a részleges függésben lévő tulajdonságokat meghatározza, és vegyük mellé az említett tulajdonságokat.

Ez egy projekció:

Törzsszám	Név	Beosztás
112	Varga Ferenc	gazdasági igazgató
112	Varga Ferenc	gazdasági igazgató
113	Tóth Irma	irodavezető
113	Tóth Irma	irodavezető
113	Tóth Irma	irodavezető

Töröljük az azonos sorokat:

Törzsszám	Név	Beosztás
112	Varga Ferenc	gazdasági igazgató
113	Tóth Irma	irodavezető

Az új Dolgozó reláció elsődleges kulcsa már csak a törzsszám, és így, magától értetődően 2NF-ben van. (Ez minden olyan relációra igaz, amelyben az azonosító egyetlen mezőből áll.)
Az előző Dolgozó relációból most emeljük ki egy másik táblába azokat a tulajdonságokat, melyek a teljes elsődleges kulcstól (törzsszám, végzettség) függnek. Ez ismét egy projekció:

Törzsszám	Végzettség
112	mérlegképes könyvelő
112	közgazdász
113	közgazdász
113	mérnök
113	rendszerszervező

Ebben a táblában az azonosító továbbra is együtt a törzsszám és a végzettség. (Láthatjuk, hogy a két tulajdonság között egyik irányban sincs függőség.) Abban az esetben, amikor a teljes tábla az elsődleges kulcs, nincs másodlagos tulajdonság, és így nyilvánvalóan teljesül a 2NF.

Tranzitív függőség
Akkor mondjuk, hogy a A, B, C tulajdonságok tranzítív függőségben vannak, ha A funkcionálisan meghatározza B-t, B pedig C-t. Látni fogjuk, hogy ha a tranzitív függőség a relációban redundanciát és ebből adódóan anomáliákat okoz.

Harmadik normál forma (3NF)
Legyen a számla egyedtípus sémája a következő:

Számla (számlaszám, vevő_kód, vevő_neve, vevő_címe, dátum, számla_összeg)

Látszólag semmi probléma nincs az adatszerkezettel, ezek az adatok a számlán valóban szerepelni szoktak. A számlaszám egyedi kulcs, meghatározza a vevőt és a többi tulajdonságot. A reláció 2NF-ben van, részleges függőséget nem is tartalmazhat, hiszen az azonosító egyetlen mező.

Mégis, ha ezt a sémát feltöltjük adatokkal, nyilvánvalóvá válik a redundancia: egy vevő adatai annyiszor fognak szerepelni az adatbázisban, ahányszor a vevő vásárol. Nyilvánvaló, hogy megint két különböző egyedtípus - a számla és a vevő - adatait próbáltuk meg egy sémában tárolni.

A reláció tranzitív függőséget tartalmaz: a számlaszám meghatározza a vevő kódját, a vevő kódja pedig a vevő nevét, és címét. A megoldást most is a séma felbontása adja:

Számla (számlaszám, vevő_kód, dátum, számla_összeg)
Vevő (vevő_kód, vevő_neve, vevő_címe)

Akkor mondjuk, hogy egy reláció harmadik normál formában van, ha második normál formában van, és nem tartalmaz tranzitív függőséget.

A normál formák ismerete a hasznos lehet az anomáliáktól mentes adatbázis tervezéséhez. Egy bizonyos gyakorlat után nem lesz nehéz olyan sémákat tervezni, amelyek eleget tesznek az első három normál forma követelményének. Magasabb fokú normál formák is léteznek, de mivel a modul célja nem az adatbázisok elméletének, hanem az SQL nyelv alapjainak bemutatása, azokra nem térünk ki.

ETK (egyed, tulajdonság, kapcsolat) adatmodell
A fentiek alapján érthető, hogy miért nevezik a relációs adatmodellt gyakran ETK adatmodellnek. Az egyedtípusok sémájának kialakítása során az elsődleges és idegen kulcsok meghatározásával gyakorlatilag meghatározzuk az egyedek közötti kapcsolatokat is. A kapcsolatokat közvetlenül nem írjuk le az adatbázis sémájában, de azokat az egyedtípusok sémája meghatározza. Az interaktív felülettel rendelkező relációs adatbázis-kezelők (például Access-ben) lehetőséget nyújtanak arra, hogy a sémákban rejlő kapcsolatokat rögzítsük. A kapcsolatok deklarálása lehetővé teszi, hogy a rendszer ellenőrizze, hogy a felhasználó által végrehajtott adatmanipulációk közben a kapcsolatok ne sérüljenek. (Például ne törölhessünk olyan könyvet a könyvtári állományból, amely ki van kölcsönözve.)

Az adatbázis fogalmának egy lehetséges definíciója is ezen alapul: az adatbázis logikailag összefüggő adatok rendszere, amelyeket kapcsolataikkal együtt tárolunk.

Adatbázis tervezés

Az adatbázis tervezés első lépéseként meg kell vizsgálni az adott rendszert, az esetlegesen érvényben lévő nyilvántartást, és pontosan meg kell fogalmazni azt, hogy milyen szolgáltatásokat várunk az adatbázistól. Ez a munkafázis a felmérés és a helyzetelemzés.

A második fázis az adatbázis logikai modelljének megtervezése. Ebben a munkafázisban meg kell határozni az adatbázis egyedtípusait, el kell dönteni, hogy a vizsgált rendszerben az egyedtípusoknak mely tulajdonságai fontosak számunkra, mit érdemes tárolni. Az adatszerkezeteket úgy kell meghatározni, hogy az adatbázisba ne kerülhessenek be lehetetlen adatok. Ehhez az kell, hogy már ezen a szinten pontosan definiáljuk az egyes tulajdonságok értékkészletét.

Az adatbázis hatékonysága szempontjából nagyon fontos, hogy az egyes egyedtípusokhoz jól válasszuk meg az elsődleges kulcsot. A sémák kialakítása után meg kell határozni a relációk közötti kapcsolatokat, majd el kell végezni a normalizálást.

A harmadik lépésben az adatbázis fizikai megvalósítása következik. A megvalósítás módja az adott hardver és szoftver környezet függvénye.

Tervezzük meg egy internetes áruház adatmodelljét!
Az internetes áruházban a vevők megrendeléseket adnak fel. A megrendelés tartalmazza a kiválasztott áruk listáját, a vevő adatait. Az eladó a megrendelést fogadja, visszaigazolja, majd utánvéttel elküldi a megrendelőnek. A csomag tartalmazza a számlát, amit a vevő a csomag átvételekor egyenlít ki. Az üzlet egyetlen raktárban tárolja az árukészletet.

Az adatbázis megtervezésének következő lépése az egyedtípusok meghatározása.

Áru(áru_azonosító, megnevezés, m_egység, mennyiség, eladási ár)
Vevő(vevő_azonosító, jelszó, e-mail, vevő_név, város, utca, irsz, telefon)

Megrendelés
A vevő kiválasztja az árut, és beleteszi a kosárba. Az áru adatai egy rendelési tételt alkotnak. A megrendelés több tételből áll, akkor tárolódik, amikor a vevő minden árut kiválasztott. A megrendelés tehát a következő részekből áll:

Megrendelés(vevő_azonosító, dátum, szállítási_határidő)
Tétel1(áru_azonosító, mennyiség), Tétel2(áru_azonosító, mennyiség), ... Tételn(áru_azonosító, mennyiség)
Megrendelésláb(Érték_összesen)

A fentiek alapján világos, hogy a Megrendelés legalább két egyedtípust tartalmaz. A megrendelés alján szereplő összeget nem kell tárolni, hiszen a tételeken szereplő összegekből elvileg bármikor kiszámítható.
Vegyük észre, hogy a Megrendelés egyedtípusban nincs megfelelő egyedi azonosító, hiszen egy vevő egy adott napon többször is vásárolhat. Célszerű egy egyedi kódot generálni, annál is inkább, mert a jelenlegi sémában a tételek azonosítása sem megoldott. A Tétel egyedtípus egyszerű szerkezetű, csak éppen sehonnan nem derül ki, hogy melyik megrendeléshez tartozik. Módosítsuk a sémákat:

Áru(áru_azonosító, megnevezés, m_egység, készlet, eladási ár)
Vevő(vevő_azonosító, jelszó, e-mail, vevő_név, város, utca, irsz, telefon)
Megrendelés(megrend_kód, vevő_azonosító, dátum, szállítási_dátum)
Tétel(megrend_kód, áru_azonosító, mennyiség, kedvezmény)

Az utolsó mezőt azért kell feltétlenül betenni a Tétel rekordba, mert a boltban gyakori a kedvezmény (ez egy %), és akkor a tétel értékét (Áru.eladási_ár* Tétel.mennyiség) a megadott százalékkal csökkenteni kell.
A számlázással egyelőre nem foglalkozunk, az adatmodell kiegészítése a Számla egyedtípussal az Olvasó feladat lesz.

Az egyedtípusok közötti kapcsolatok:
Vevő---Megrendelés: 1:N Kapcsolómező: vevő_azonosító
Megrendelés---Tétel: 1:N Kapcsolómező: megrend_kód

Tétel---Áru: N:1

A Vevő és az Áru egyedtípus között N:M kapcsolat van, hiszen egy vevő több árut vásárol, és egy árut több vevő vásárolhat meg. Az N:M kapcsolatot azonban a Megrendelés és a Tétel egyedtípusokon keresztül 1:N típusú kapcsolatokra bontottuk fel. Közvetve minden egyedtípus kapcsolatban áll egymással. A tervezett relációk mindegyike harmadik normál formában van, minden relációban van elsődleges kulcs, részleges és tranzitív függőségeket nem tartalmaznak. Ezzel az adatbázis logikai terve elkészült.

A szoftverrendszerek belső minősége szoros összefüggésben van a fejlesztésükkel és fenntartásukkal kapcsolatos költségekkel. Léteznek olyan statisztikai és dinamikus teszteléshez hasonló eszközök és technikák, amelyek a szoftverrendszerek gyors és alapos elemzését teszik lehetővé.