Írta:
KATONA ENDRE

TÉRKÉPI ADATBÁZISOK

Egyetemi tananyag

TYPOTEX

2011

LEKTORÁLTA: Dr. Elek István, ELTE Informatika Kar Térképtudományi és Geoinformatikai Tanszék

TÁMOGATÁS:
Készült a TÁMOP-4.1.2-08/1/A-2009-0008 számú, „Tananyagfejlesztés mérnök informatikus, programtervező informatikus és gazdaságinformatikus képzésekhez” című projekt keretében.

ISBN 978-963-279-509-6

KÉSZÜLT: a Typotex Kiadó gondozásában

FELELŐS VEZETŐ: Votisky Zsuzsa

AZ ELEKTRONIKUS KIADÁST ELŐKÉSZÍTETTE: Faragó Andrea

KULCSSZAVAK:

digitális térkép, térinformatika, geoinformatika, vektoros adatstruktúrák, vektoros algoritmusok, PostGIS, Oracle Spatial

ÖSSZEFOGLALÁS:

A tananyag a térképhez kapcsolódó információs rendszerek fejlesztését hivatott támogatni. Tudományterületét tekintve a *térinformatikához (geoinformatikához)* kapcsolódik, azon belül pedig a *vektoros térinformatikát támogató relációs adatbázisokat* célozza meg.

A térbeli (térképi) adatok kezelésének két alapvető módja kerül bemutatásra: a szétválasztott modell és az integrált modell. A *szétválasztott modell* a térbeli és leíró adatok elkülönült kezelésére és laza kapcsolatára épül, és inkább az elmúlt évek fejlesztési technológiájára jellemző. Ezzel szemben az *integrált modell* lényege, hogy a térbeli és leíró adatokat egy közös relációs adatbázisban tároljuk. Ma már több adatbázis-kezelő is támogatja a térbeli adatok ilyen kezelését. A tananyag a MySQL, PostgreSQL (és annak PostGIS kiterjesztése), valamint az Oracle Spatial lehetőségeit mutatja be.

TARTALOMJEGYZÉK

Bevezetés

Napjainkban egyre több informatikai rendszerhez kapcsolódik digitális térkép, és örvendetesen nő az ingyenesen hozzáférhető szoftverek és térképi adatbázisok köre. A GoogleMaps térképi keresőrendszerét szinte mindenki ismeri és használja. Az elmúlt évtizedben jelentősen fejlődött a kapcsolódó technológia is: több adatbázis-kezelő is támogatja a térbeli (térképi) adatok kezelését, és a megjelenítő szoftverek köre is bővül.

Ez a tananyag a térképhez kapcsolódó információs rendszerek fejlesztését hivatott támogatni. Tudományterületét tekintve a *térinformatikához (geoinformatikához)* kapcsolódik, azon belül pedig a *vektoros térinformatikát támogató relációs adatbázisokat* célozza meg.

A tananyag egy általános térinformatikai alapvetéssel indul, de a 3. fejezetben már a vektoros adatmodellekre és algoritmusokra kerül a hangsúly, a 4. fejezet pedig a relációs adatbázisok alapismereteit foglalja össze.

Ezután következik a tananyag érdemi része: a térbeli (térképi) és leíró adatok kezelésének két alapvető módját mutatjuk be: a szétválasztott modellt (loosely coupled approach, 5. fejezet) és az integrált modellt (integrated approach, 6., 7., 8. fejezetek) [Rigaux és tsai, 2002]. A szétválasztott modell a térbeli és leíró adatok elkülönült kezelésére és laza kapcsolatára épül, és inkább az elmúlt évek fejlesztési technológiájára jellemző. Ezzel szemben az integrált modell lényege, hogy a térbeli és leíró adatokat egy közös relációs adatbázisban tároljuk. Ennek három módja lehetséges: tisztán relációs (6. fejezet), objektum-relációs (7. fejezet) és térbeli adattípusokra épülő megközelítés (8. fejezet).

A záró fejezetek a térbeli adatkezelés két járulékos kérdésével, a térbeli indexeléssel (9. fejezet) és a megjelenítéssel (10. fejezet) foglalkoznak.

A tananyag nem kötelezi el magát egyetlen konkrét adatbázis-kezelő rendszer vagy fejlesztő környezet mellett sem, inkább áttekintést kíván adni a témakörről és ízelítőt az egyes konkrét rendszerekből (lásd a Függeléket). A tényleges fejlesztői munkához elengedhetetlen a megfelelő kézikönyvek tanulmányozása (lásd az Irodalomjegyzéket).

Végül megjegyezzük, hogy a tananyag mintapéldáiban a tábla-, mező- és változónevek – a könnyebb olvashatóság érdekében – ékezetes betűkkel szerepelnek, konkrét programozási környezetben azonban ez esetleg nem megengedett vagy zavarokat okozhat, tehát kerülendő.

1. Raszteres és vektoros képkódolás

Ebben a fejezetben áttekintjük és összehasonlítjuk a térképi információ két fő kódolási módját, a raszteres és vektoros kódolást. Ez utóbbit részletesebben tárgyaljuk, mivel a továbbiakban erre lesz szükségünk.

1.1. Raszteres képkódolás

A képet mátrix formájában tároljuk (1. ábra). Egy mátrixelem szokásos elnevezései: *képpont, pixel, cella*. A kép típusát alapvetően a *bit-per-pixel érték* határozza meg, vagyis az, hogy egy képpont hány bitből áll. Néhány jellemző típus:

- 1 bites pixelek: bináris kép.
- 8 bites pixelek: monochrom kép, 256 szürkeárnyalat.
- 24 bites pixelek: színes kép, ahol a színek a három alapszín (piros, zöld, kék) keverékeként kódoltak, mindegyik színkomponensnél 256 árnyalattal (3*8 bit).
- Multispektrális műholdkép: infravörös és ultraibolya színkomponenseket is tartalmaz, például 7 sávban, sávonként 8-bites (vagy 16-bites) pixelek. A bit-per-pixel érték tehát itt 7*8 (vagy 7*16) bit.

Előfordulhat, hogy a rasztermátrix nem képi információt hordoz (pl. talajtérkép, terepmodell), ilyenkor a pixelenkénti bitek száma is a fentitől tetszőlegesen eltérő lehet.

Felbontás (geometriai): megadja, hogy egy pixel mekkora területnek felel meg a valóságban (pl. 10 x 10 méter).

```
0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 0 0 0
0 1 1 0 0 0 0 0 0 0 0 0
0 1 0 1 0 0 0 0 0 0 0 0
0 1 0 0 1 0 0 0 0 0 0 0
0 1 0 0 0 1 0 0 0 0 0 0
0 1 0 0 0 0 1 0 0 0 0 0
0 1 0 0 0 0 0 1 0 0 0 0
0 1 0 0 0 0 0 0 1 0 0 0
0 1 1 1 1 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0
```

1. ábra: Egy háromszög kontúrja raszteres adatábrázolásban (bináris kép)

Jellemző adatformátumok: TIFF, PCX, BMP, JPEG, stb. Térinformatikai alkalmazásokban leggyakrabban a *TIFF grafikus formátumot* használják. (TIFF = Tagged Image File Format, az 1980-as évek végén kidolgozott raszteres képformátum.) A TIFF fájl felépítése:

- Header (8 byte, részletesebben lásd a Függelékben).
- IFD = Image File Directory: a képet leíró paramétereket tartalmazza tag-ek (magyarul címkék) felsorolásával. A tag-ek többsége opcionális (azaz elhagyható), igen rugalmas képleírást tesznek lehetővé (részletesebben lásd a Függelékben).

– Maga a kép (pixelek sorozata).

GeoTIFF formátum: az 1990-es évek közepén definiált, speciális tag-ekkel bővített, térinformatikai célú TIFF formátum. A GeoTIFF kép olyan programokkal is megjeleníthető, amelyek csak az alap TIFF formátumot ismerik, de ezek természetesen nem tudják értelmezni a speciális címkéket. A geoTIFF formátumot elsősorban *georeferencia* leírására használják: ez lényegében az alkalmazott vetületi rendszer leírását jelenti, amely segítségével az egyes pixeleknek megfelelő vetületi koordináták meghatározhatók.

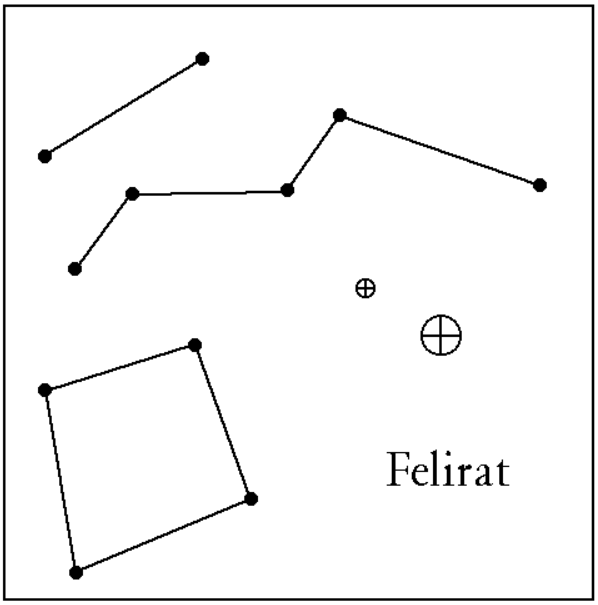
1.2. Vektoros képkódolás

A képet *rajzelemek* halmazaként (rendszereként) tároljuk, az egyes rajzelemeket koordináta-geometriai eszközökkel írjuk le. Ez a mérnöki tervező rendszerek (CAD rendszerek) jellemző adatformátuma. Leggyakoribb rajzelem az úgynevezett *vektor*, vagyis egy (irányított) egyenesszakasz, amelyet végpontjainak koordinátáival adunk meg:

$$LINE\ x_1, y_1, x_2, y_2$$

Például, az 1. ábrán látható háromszög vektoros adatábrázolásban a következő (bal alsó sarok a koordinátarendszer középpontja):

```
LINE  1,1, 1,9
LINE  1,1, 9,1
LINE  1,9, 9,1
```



2. ábra: Vektoros rajzelemek: egyenesszakasz, vonallánc, poligon, blokkok és felirat

Példák további rajzelemekre (2. ábra):
CIRCLE x, y, r: kör, amely (x, y) középponttal és r a sugárral adott.
POLYLINE $x_1, y_1,..., x_n, y_n$*:* vonallánc (más néven töröttvonal, linestring, egyes rendszerekben „ív”), amely a töréspontjainak koordinátáival adott.
POLYGON $x_1, y_1,..., x_n, y_n$*:* töréspontjaival adott alakzat (zárt poligon), ahol $x_{n+1} = x_1$, $y_{n+1} = y_1$.

TEXT x, y, méret, irány, szöveg: felirat a rajzon az (x, y) pontban, adott méretben és irányban. A felirat tartalmát a „szöveg” ASCII jelsorozat adja meg.

1.2.1. A vektoros rajz strukturálása

Rétegekre bontás (fóliázás): rajzelemek csoportosítása jelentésük szerint, valamilyen szempontból. Egy réteg többféle rajzelemtípust is tartalmazhat. Például egy épület alaprajza az alábbi rétegekből állhat:

- falak,
- helyiségek feliratai,
- ajtók és ablakok,
- vízvezetékek és elzárócsapok,
- elektromos vezetékek.

Az egy réteghez tartozó rajzelemek együtt kezelhetők, például közös szín, vonaltípus és vonalvastagság rendelhető hozzájuk. Az egyes rétegek megjelenítése külön-külön ki-bekapcsolható.

Blokk (cell): többször ismétlődő rajzrészlet, jelkulcsi elem kezelésére szolgál, például túristatérképen benzinkút jele (körbe rajzolt T betű), vagy épület homlokzatrajzon ablak, vagy gépészeti rajzon csavar (lásd még a körbe rajzolt kereszt szimbólumot a 2. ábrán). A blokk kezelése két részből áll:

- *blokk definíció:* a blokk egy mintapéldánya, tetszőleges rajzelemek együtteséből áll.
- *blokk hivatkozás,* alakja a következő lehet: (blokknév, x, y, α, zoom), amely a *blokknév* blokk beillesztését írja elő a rajz *(x, y)* koordinátájú pontjára, α elforgatási szöggel és *zoom* nagyítási faktorral. (Általánosabb esetben transzformációs mátrix alkalmazható.)

1.2.2. Pontosságot biztosító eszközök

A vektoros rendszerek nagy pontossággal (általában lebegőpontos számábrázolással) tárolják a koordinátákat. Ha azonban a rajzot képernyőn egérkattintgatással szerkesztjük, a koordináták bevitele szükségképpen pontatlan lesz. A pontosság biztosítására a rendszerek különféle eszközöket biztosítanak.

Numerikus koordináta megadás. Egérkattintás helyett begépeljük a koordinátákat. Ez kétségtelenül kényelmetlen, de garantáltan pontos módszer.

Pontrács vagy négyzetrács megjelenítése, például az egész koordinátájú pontokban. Ez egyrészt tájékozódásra szolgál, másrészt viszont bekapcsolható, hogy csak rácspontra eső pontokban lehessen koordinátát bevinni.

Csatolás (snapping). Ha már meglévő rajzelemhez szeretnénk kapcsolódni, akkor szükséges az aktuálisan bevitt pont ráhúzása a legközelebbi rajzelem megfelelő pontjára. Példák:

- Poligon bezárása (a kezdő és záró pont koordinátáinak pontos megegyezését biztosítani).

– Rajz folytatása már berajzolt szakasz végpontjából.
– T-elágazás (meglévő egyenesszakasz valamely belső pontjára való pontos csatlakozás).

1.2.3. Jellemző adatformátumok

Általában minden vektoros szoftvernek van saját formátuma (AutoCAD: DWG, MicroStation: DGN, stb.). A rendszerfüggetlen (transzfer) formátumok közül a DXF a legismertebb.

A DXF adatformátumot (Drawing eXchange Format) az AutoDesk cég specifikálta és folyamatosan fejleszti. A legtöbb vektoros rendszer tudja importálni, ill. exportálni. Szöveges és bináris változata használatos. Az alábbi szekciókból áll:

– HEADER: változók beállítása (koordinátarendszer, stb.).
– TABLES: vonaltípus, réteg, stb. definíciók.
– BLOCKS: blokk definíciók.
– ENTITIES: rajzelemek felsorolása.

A fájlformátum részletesebb leírása a Függelékben található.

1.3. Összehasonlítás

Raszter	***Vektor***
A látszati kép	A kép struktúrája
Minden raszterponthoz megadja, hogy ott *milyen objektum* van	Minden objektumhoz megadja, hogy az a síkon *hol* van
Pontosság a felbontástól függ	Pontosság a számábrázolástól függ
Nagyításnál durvább lesz	Nagyításnál nem lesz durvább
Transzformáció: lassú, torzulhat	Transzform. gyors, gyakorlatilag nem torzul
Monitoron közvetlenül megjeleníthető	Megjelenítéséhez rajzolóprogram szükséges
Tárolóterület: képmérettől függ	Tárolóterület a rajz bonyolultságától függ

3. ábra: A raszteres és vektoros kódolás összehasonlítása

Tanulságos összevetni a raszteres és vektoros kódolás jellemzőit (3. ábra). Néhány megjegyzés:

– *Alkalmazási terület:* ha az adatforrás kamerakép, akkor a raszteres ábrázolás a kézenfekvő, míg számítógépen szerkesztett tervrajz, térkép esetén a vektoros kódolás a megfelelőbb.
– *Pontosság.* Ha egy raszterkép pontosságát javítani akarjuk, akkor a kép felbontását kell növelni, ami drasztikus tárolóhely növekedéssel járhat. Vektoros esetben a pontosságot a koordináták kódolása határozza meg (16 bites vagy 32 bites, fixpontos vagy lebegőpontos).
– *Transzformáció*. Ha például egy képet 360-szor 1-fokos elforgatással körbeforgatunk, akkor raszterkép esetén a sokszori „átmintázás” miatt számottevő torzulással számolhatunk, míg vektoros esetben a torzulás nem jelentős.

– *Tárolóterület.* A raszteres ábrázolás általában terjedelmesebb, de ha adattömörítést alkalmazunk (például ZIP), akkor ugyanazon kép (mondjuk egy vonalrajz) raszteres és vektoros változatának mérete között már nem lesz nagy különbség.

1.4. Speciális beviteli eszközök

Digitalizáló tábla (tablet): Vektoros beviteli eszköz. Egy elektronikusan vezérelt, A3...A0 méretű táblából, és egy egérhez hasonló pozícionáló eszközből (*tábla kurzor*) áll. A tábla kurzor – az egértől eltérően – abszolút pozíciót érzékelő eszköz: mindig pontosan érzékeli, hogy a tábla mely pontján van, akkor is, ha felemelve helyezzük át.

A digitalizáló tábla manuális vektoros adatbevitelt támogat. A digitalizálandó rajzot a táblára rögzítik, majd a tábla kalibrálásával elérik, hogy a rajz négy sarokpontja a képernyőn látható rajzterület négy sarkának feleljen meg. Ezután a tábla kurzorral manuálisan követik a rajz vonalait, és a vonal végpontoknál gombnyomással viszik be a megfelelő koordinátákat.

Az eljárás hátránya, hogy az adatbevitel pontossága és teljessége csak nehezen ellenőrizhető.

Szkenner (scanner): optikai leolvasó, raszteres adatot állít elő. Főbb típusok:

– *Síkszkenner:* általában A4 (esetleg A3) méretű. A szkennelendő lapot egy üveglapra kell helyezni, amelyet egy levilágító-érzékelő berendezés soronként letapogat. A síkszkenner igen pontos adatbevitelt biztosít, de nagyobb méretű berendezések igen drágák, ezért ritkán használatosak.

– *Dobszkenner:* A1, A0 méretű (valójában csak a szélesség korlátozott, a hosszúság nem). A síkszkennerrel ellentétben itt a levilágító-érzékelő berendezés rögzített, és előtte halad el a digitalizálandó lap. Fényes felületű lapok (fóliák) szkennelése esetén a laptovábbítás egyenetlen lehet, ami a szkennelés pontosságát rontja. Bár a dobszkenner is viszonylag drága berendezés, a térinformatikában ezt használják legáltalánosabban.

Ha a szkenneléssel előállított raszterképet vektorizálni kell, ez a képernyőn hasonlóan végezhető, mint a digitalizáló táblával: raszter háttérképre egérrel rajzoljuk rá a vektoros rajzot. Mivel az operátor a képernyőn egymásra vetítve látja a szkennelt raszteres és az általa létrehozott vektoros rajzot, így a pontosság és teljesség könnyen ellenőrizhető.

1.5. Konverziók

Vektor → raszter: az egyes rajzelemek raszteres képét kell algoritmikusan előállítani, ami viszonylag könnyen megoldható (számítógépes grafika). Vektoros kép monitoron való megjelenítésekor mindig ez történik, mivel a monitor raszteresen dolgozik.

Raszter → vektor: a raszterképen az egyes objektumokat kell felismerni, és megfelelő vektoros kóddal helyettesíteni. A probléma jellemzően szkennelt papírtérképek vektorizálásakor jelentkezik. A lehetséges feldolgozási módok:

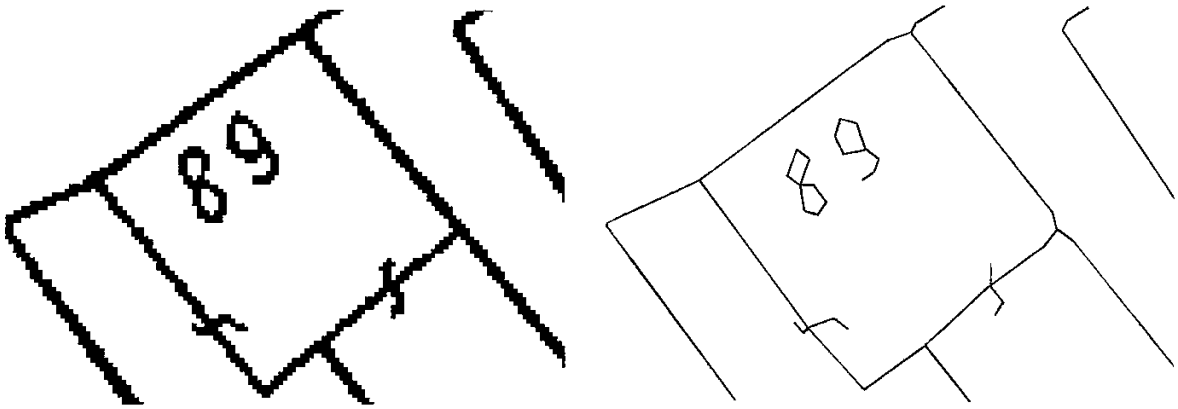
1. *Manuális vektorizálás.* A vektoros rajz előállítását teljes egészében az operátor végzi képernyőn való digitalizálással, amint azt a szkennereknél leírtuk.

2. *Félautomatikus vektorizálás.* A szoftver a képernyőn automatikusan követi a vonalakat és képezi le vektorokra, de elágazásnál megáll, és az operátor irányítására vár: hogyan tovább. Az eljárás például olyankor előnyös, amikor nem kell a teljes rajzot vektorizálni. Az előállított eredmény azonban rendszerint manuális korrigálásra szorul.
3. *Automatikus vektorizálás.* Itt a szoftver operátori beavatkozás nélkül állítja elő a vektoros rajzot. Az eljárás viselkedése általában számos paraméter beállításával szabályozható, a piacon kapható vektorizáló rendszerek több-kevesebb alakfelismerő képességgel is rendelkeznek (szaggatott vonalak felismerése, szimbólumok elkülönítése, stb.). Az előállított vektoros rajz itt is manuális javításra szorul.

Példaként a 4. ábra bal oldalán egy szkennelt kataszteri térkép részletét látjuk (raszter), a jobb oldalon ennek automatikus vonalkövetéssel vektorizált változatát. Ez a vektoros rajz lényegében használhatatlan, mert

– a „89” szám apró vektorok halmazaként jelenik meg (pedig TEXT rajzelemként kellene kódolni, de ehhez karakterfelismerő algoritmus kell),
– a két kis hullámvonal (úgynevezett kapcsolójel) szintén vektorok halmazaként jelenik meg (pedig jelkulcsi elemként, azaz blokk hivatkozásként kellene kódolni, de ehhez alakfelismerő algoritmus kell),
– a vonalrajz T-elágazásainál jellegzetes behúzódások keletkeznek, amelyek megszüntetése szintén egy megfelelően „intelligens” utófeldolgozó algoritmust igényel.

A fenti problémák megoldása alakfelismerő és egyéb mesterséges intelligencia algoritmusok alkalmazását igényli [Katona, 2001].



4. ábra: Automatikus, vonalkövető raszter-vektor konverzió

2. Térképészeti alapfogalmak

Mindenki tudja, mi a térkép, de a rend kedvéért vegyünk egy pontos definíciót a Nemzetközi Térképészeti Szövetség (International Cartographic Association, ICA) meghatározásának megfelelően: *térképnek nevezzük a Föld felszínén, illetve azzal kapcsolatban álló anyagi vagy elvont dolgoknak kicsinyített, általánosított, síkbeli megjelenítését.* Az angol *map* szó nemcsak térképet, de a matematikában leképezést is jelent. Valóban, a térkép is a földfelszín leképezése egy papírlapra vagy képernyőre, meghatározott szabályok szerint.

2.1. Méretarány

A méretarány a térképi távolság és a valós távolság hányadosa. (Ezt a meghatározást a vetületi rendszereknél majd pontosítjuk.) Ha a térkép méretaránya 1:50.000, akkor a térképen 1 mm a valóságban 50.000 mm-nek, azaz 50 méternek felel meg a Föld felszínén.

A „kisméretarányú” és a „nagyméretarányú” jelzők használata gyakran téves vagy félreérthető, ezért fontos tisztázni:

- *Nagyméretarányú* a térkép, ha az 1:m hányados 1:10.000-nél nagyobb (vagyis m < 10.000). A térkép részletgazdag, az egyes objektumok relatíve nagy méretben jelennek meg.
- *Kisméretarányú* a térkép, ha az 1:m hányados értéke 1:10.000 vagy ennél kisebb (tehát m > 10.000). A térkép kevesebb részletet tartalmaz, az egyes objektumok relatíve kisebb méretben jelennek meg.

Térképszelvényen egy összefüggő papírlapon ábrázolt térképrészt értünk. A szelvények továbbosztása általában negyedeléssel történik, például egy 1:4000 méretarányú szelvény által lefedett terület négy 1:2000-es szelvényen ábrázolható.

2.2. Térképtípusok

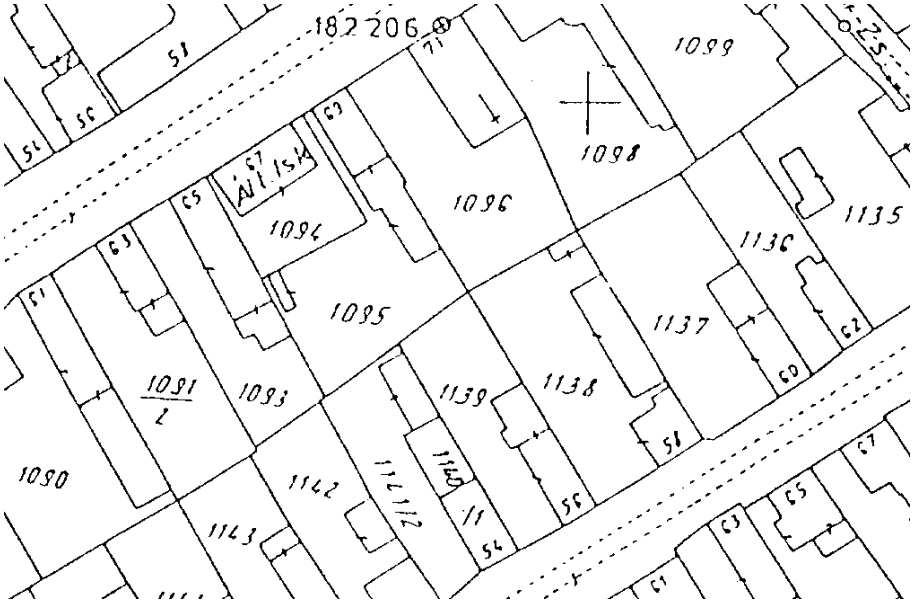
(i) Általános térképek: A földfelszín kiválasztott természetes és mesterséges objektumait ábrázolja (domborzat, vízrajz, út-vasút, települések). Jellemző változatai:

- *földmérési alaptérképek (kataszteri térképek):* 1:500 ... 1:10.000 méretarány (5. ábra). Elsősorban földhivatali ingatlan-nyilvántartásra használják.
- *topográfiai térképek:* 1:10.000 ... 1:300.000 méretarány (6. ábra).
- *földrajzi térképek:* 1:300.000-nél kisebb méretarány, ezekkel a szokásos földrajzi atlaszokban találkozunk.

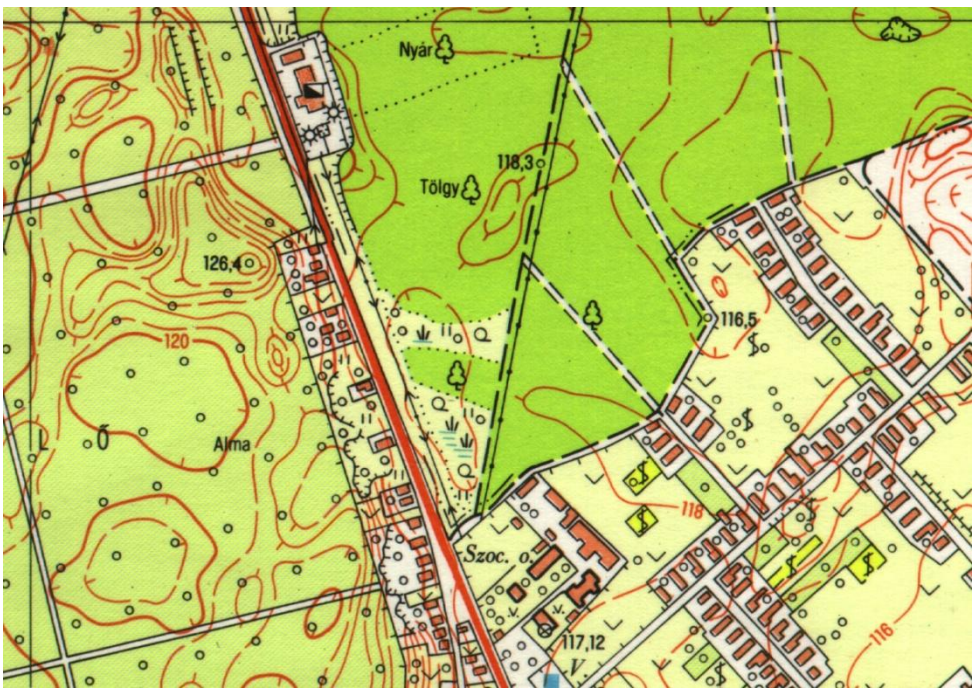
(ii) A *tematikus térkép* valamely téma(csoport) közvetítésére szolgál, mint például közművek, a népesség eloszlása, klimatikus viszonyok, áruforgalmi adatok stb. Általában egy általános térkép egyszerűsített változatára épül rá.

Más felosztás szerint:

- A *vonalas térkép* (vektoros) az objektumokat szimbólumokkal és (határ)vonalakkal ábrázolja, előállítása alapvetően emberi tervezőmunkával történik.
- A *fotótérkép* (raszteres) légifényképek vagy műholdképek alapján készül. A terep jellemzői a fotótérkép alapján önállóan interpretálhatók, bizonyos jellemzők azonosíthatók feliratok elhelyezésével is. Előállításuk viszonylagosan olcsó.



5. ábra: Szkennelt magyar kataszteri térkép részlete (1:2000), az eredeti térkép kézi tusrajz formájában készült. Jellemző objektumok: földrészletek (telkek) helyrajzi számokkal és épületek házszámokkal



6. ábra: Nyomtatott magyar topográfiai térkép részlete (1:10 000)

2.3. Koordináta rendszerek, vetületi rendszerek

Ebben a fejezetben a vetülettan alapfogalmait tekintjük át, és megismerkedünk a térinformatikában leggyakrabban használt vetületi rendszerekkel [Lerner 1989, Stegena 1988, GISjegyzet].

2.3.1. Gömbi geometria

A Föld alakja az ún. *geoid*, amelyet úgy kapunk, hogy a világóceánok közepes szintjét gondolatban a kontinensek alatt is folytatjuk.

A geoid durva közelítéssel gömbnek tekinthető. Finomabb közelítéssel egy, a pólusoknál kissé lapult *forgási ellipszoid*, amelynek egyenlítői átmérője kb. 0.3%-kal nagyobb a sarki átmérőnél. A gömb és a forgási ellipszoid közötti eltérés kb. annyi, mint a földfelszín domborzati változatossága.

Kisebb területek térképezéséhez az úgynevezett *Gauss-gömböt* veszik alapul, amely a földfelszín adott pontjához legjobban simuló gömb.

Egy gömbfelületen sajátos geometriai viszonyok uralkodnak (gömbi geometria), ezt tekintjük át a továbbiakban. A gömb sugarát R-rel jelöljük. Alapfogalmak:

- *Főkör:* a gömb középpontján áthaladó síknak a gömbfelülettel való metszete. Főkörív: a főkör egy szakasza. A főkörök az egyenesek szerepét játsszák a gömbi geometriában. Eltérés a síkgeometriától, hogy nincsenek párhuzamos egyenesek, bármely két gömbi egyenes (főkör) metszi egymást.
- *Főkörív középponti szöge:* a két végpontjából húzott gömbi sugarak által bezárt szög, radiánban mérjük. A főkörív hossza $R*\alpha$.
- *Két pont távolsága:* a pontokon áthaladó főkör rövidebb ívének hossza. (Ez két pont között a legrövidebb út.)
- *Szög:* két gömbi egyenes bezárt szöge, amelyet a síkjaik hajlásszögével mérünk. (Ugyanezt a szögértéket kapjuk, ha felületi görbék hajlásszögeként definiáljuk a szöget.)

Gömbi alakzatok:

- *Euler-féle gömbháromszög:* a gömbfelület három pontját összekötő, π-nél kisebb középponti szögű három főkörív által határolt terület. Szögei kisebbek π-nél, szögeinek összege viszont nagyobb π-nél. Felszíne: $F = R^2(\alpha + \beta + \gamma - \pi)$.
- *Gömbkétszög:* két gömbi egyenes által határolt terület. Két főkör a gömböt négy gömbkétszögre osztja. A gömbkétszög felszíne $F = 2R^2\alpha$ (a két főkör α szöge egyértelműen meghatározza). Megjegyezzük, hogy a teljes gömbfelszín $4R^2\pi$.

Földrajzi fogalmak:

- *Északi és déli pólus:* a gömb két kitüntetett, átellenes pontja.
- *Meridián:* a pólusokon áthaladó főkör.

– *Egyenlítő:* a meridiánokra merőleges főkör.
– *Loxodróma:* olyan görbe, amely minden meridiánt azonos szögben metsz. A loxodróma ugyan nem a legrövidebb utat adja két pont között, de ha egy jármű loxodróma pályán halad, akkor az iránytűhöz viszonyított haladási irányát nem kell megváltoztatni.

Gömbi koordinátarendszerek: nem törekednek a gömbfelület síkba való kiterítésére, hanem közvetlenül a gömbfelületet írják le. Két jellemző változat:

1. Geocentrikus: egy pontot az (x, y, z) derékszögű koordinátákkal azonosítunk, ahol a koordinátarendszer origója a Föld középpontja.

2. Földrajzi: egy pontot a (hosszúság, szélesség) koordinátapárral azonosítunk, ahol

– *hosszúság (λ):* a pont meridiánjának a greenwichi kezdő meridiánnal bezárt szöge. Értéke -180° (nyugati hosszúság) és +180° (keleti hosszúság) között változik.
– *szélesség (φ):* a pontból az Egyenlítőre bocsátott merőleges szakasz középponti szöge. Értéke -90° (déli szélesség) és +90° (északi szélesség) között változik.

Meridián: azonos hosszúsági koordinátájú pontok együttese.

Szélességi kör vagy *paralelkör:* azonos szélességi koordinátájú pontok együttese. (Nem főkör, tehát két pont között nem a legrövidebb utat adja!)

2.3.2. Vetületi rendszerek

Vetületi rendszer: egy V: $(\lambda, \varphi) \rightarrow (x, y)$ leképezés, amely a földfelszín minden pontjának a síkbeli Descartes koordinátarendszer egy pontját felelteti meg.

Megjegyezzük, hogy a geodéziában a függőleges koordinátákat jelölik x-szel és a vízszintest y-nal. Mi azonban a matematikai konvenciót alkalmazzuk, vagyis x a vízszintes és y a függőleges koordináta tengely.

A vetületi rendszereket több szempont szerint lehet osztályozni.

A vetítés módja szerint:

– Perspektív vetület: előállítása vetítősugarakkal történik.
– Nem perspektív vetület: nem állítható elő vetítősugarakkal.

A leképezés módja szerint:

– *Síkvetület:* a gömbfelületet közvetlenül síkra képezzük le.
– *Hengervetület:* a gömbfelületet előbb egy hengerfelületre képezzük le, majd azt egy egyenes mentén felhasítva síkba terítjük ki.
– *Kúpvetület:* a gömbfelületet előbb egy kúpfelületre képezzük le, majd azt egy egyenes mentén felhasítva síkba terítjük ki.

Elérendő tulajdonságok (invariánsok) szerint:

- *Területtartó vetület.*
- *Szögtartó vetület.* (A szögtartás navigációs szempontból fontos.)

Hossztartás csak egyes vonalak mentén lehetséges.

A méretarány fogalmának pontosítása. Mivel minden vetületi rendszer torzít, így méretarányon a térképen mért hossz és a vetületi hossz hányadosát értjük, ahol vetületi hosszon a földfelszínnek az adott vetületi rendszer szerinti, kicsinyítés nélküli síkbeli képén mért hossz értendő. A térképen mért távolságokból tehát a méretarány segítségével csak a vetületi távolságokat kapjuk, a valós távolságok meghatározásához az adott vetületi rendszer torzításait is figyelembe kell venni.

Sztereografikus vetület

Perspektív síkvetület: centrális vetítés a gömb egy C pontjából a gömb azon érintősíkjára, amely a vetítési középponttal átellenes P pontban érinti a gömböt. A C pont kivételével a gömb valamennyi pontját egyértelműen leképezi a síkra. Szögtartó és körtartó leképezés, de a C ponton átmenő körök egyenesekre képeződnek le. A P pont közelében a torzítás csekély, a C pont felé haladva rohamosan nő.

Mercator vetület

Szögtartó, nem perspektív hengervetület. Egyenletei:

$$x = \lambda \qquad y = \ln(\operatorname{tg}(\varphi/2 + \pi/4)).$$

A szélességi körök vízszintes, a meridiánok függőleges, a loxodrómák általános helyzetű egyenesekbe mennek át. Nem területtartó, a pólusok felé haladva a területek erősen növekednek.

A Mercator vetület alkalmazásai

GK = Gauss-Krüger vetület: Elsősorban Kelet-Európában használatos. A Föld alakját ellipszoiddal modellezi (ún. Kraszovszkij-féle ellipszoid). Az ellipszoid felszínét 6 fokonként (nagyobb méretarány esetén 3, ill. 2 fokonként) meridiánokkal zónákra (ellipszoid kétszögekre) osztja. Minden egyes zóna esetén egy transzverzális helyzetű elliptikus hengerre Mercator vetítést alkalmaz úgy, hogy a vetítési henger a zóna középmeridiánjánál érinti a felszínt. A vetület szögtartó, és az érintő meridián mentén hossztartó.

UTM = Univerzális Transzverzális Mercator vetület: A GK-hoz hasonló rendszer, a világon általánosan használják. A Föld alakját szintén ellipszoiddal közelíti (ún. Hayford-féle ellipszoid), melynek felszínét 6 fokonként meridiánokkal zónákra osztja. A GK-hoz hasonlóan zónánként transzverzális Mercator vetítést alkalmaz, de úgy, hogy a henger a sarkoknál érinti, egyébként metszi a felszínt. A vetület szögtartó, és a két metsző meridián mentén hossztartó.

Az UTM és GK vetületekhez az egész Földre kiterjedő egységes szelvényezés tartozik. A zónákat szélességi övekre osztják: UTM esetén 8, GK esetén 4 szélességi fokonként. A zónák találkozásánál fellépő elcsúszásokat átfedésekkel küszöbölik ki. (Pl. ha a térképezendő terület két zóna határára esik, akkor a domináns zóna kiterjesztésével térképezik.)

2.3.3. Magyarországi vetületi rendszerek

Budapesti sztereografikus rendszer

A vetítési sík a Gellért-hegy egy meghatározott pontjában érinti a Gauss-gömböt. 127 km sugarú körben 1/10.000-nél kisebb hossztorzulást biztosít. Legnagyobb hossztorzulás Szabolcs-Szatmár megyében lép fel (kb. 4/10.000).

Egységes Országos Vetületi rendszer (EOV)

1975 óta használatos vetület. Olyan vetületi rendszer, amely Magyarország területét egységesen és minimális torzulással kezeli. Az ellipszoidról előbb Gauss-gömbre, majd hengerfelületre vetítenek. A síkbeli koordinátarendszer függőleges tengelye a Gellérthegyen áthaladó meridiánnak, vízszintes tengelye az ország középvonalánál a gellérthegyi meridiánra merőleges gömbi főkörnek felel meg. A vetítési henger erre a főkörre illeszkedik.

Szögtartó vetület. A hossztorzulás az ország egész területén 1/30.000.000 alatt marad. Koordináta egysége 1 méter, a szelvények téglalap alakúak. A koordinátarendszer kezdőpontja a vetítési középponttól 200 km-rel délre, 650 km-rel nyugatra van, így minden koordináta pozitív, és az x, y koordináták sem téveszthetők össze, mert x > 400.000 > y minden esetben teljesül.

EOTR (Egységes Országos Térképezési Rendszer): az EOV-re épülő térképezés. Az országot 83 db 1:100.000 méretarányú szelvény fedi le, ezek továbbosztásával adódnak a nagyobb méretarányú szelvények. A földmérési alaptérképek 1:1000 ill. 1:2000 méretarányban mutatják a beépített területeket, 1:4000 méretarányban lefedik a külterületeket. Szokásos szelvény méret: 50 x 75 cm. Az EOTR topográfiai térképsorozata 1:10.000, 1:25.000 ill. 1:100.000 méretarányú térképeket tartalmaz.

3. Vektoros adatstruktúrák és algoritmusok

3.1. A modellalkotás folyamata

Ahhoz, hogy a létező világ jelenségeit és folyamatait a számítógépre le tudjuk képezni, modellalkotásra van szükség. Ennek három szintjét szokták megkülönböztetni:

Elvi modell: ezen a szinten a számunkra fontos entitásokat, kapcsolatokat és folyamatokat próbáljuk megragadni. Az adatbázisok világában erre a célra szolgál az *egyed-kapcsolat modell*, az információs rendszereknél pedig az *SSADM* (Structured Systems Analysis and Design Method [Bana, 1995]) vagy az UML (Unified Modeling Language [Ullman-Widom, 2008]). Ez a modellezési szint még független a konkrét implementációtól.

Logikai modell: lényegében absztrakt adatstruktúrák modellezési szintje, a modell alapegységeit gyakran *objektum*oknak nevezik. Adatbázisok esetén rendszerint *relációs adatmodellt* használnak (relációsémák, elsődleges kulcsok és külső kulcsok). Gyakran maga a logikai modell is több hierarchiaszintre osztható.

Fizikai modell: a tényleges gépi adatkezelés szintje. Ezt a szintet az egyes alkalmazói szoftverek többé-kevésbé eltakarják a felhasználó elől.

A vektoros térinformatikát továbbiakban a *logikai modell* szintjén vizsgáljuk. Az alábbi fogalmakat használjuk:

- *Térbeli adatbázis* (*spatial database*): részben vagy egészben térbeli (térképi) vonatkozású adatok rendszere. Egy alkalmazás által kezelt valamennyi (térbeli és nem térbeli) adat együttesét nevezzük térbeli adatbázisnak.
- *Fedvény* (angolul *coverage*): tematikusan összetartozó térbeli objektumok együttese. A CAD rendszerbeli rétegfogalom általánosításának tekinthető. Hasonló értelemben használatos a feature class (tulajdonságosztály) fogalma is [Elek, 2006].

A térbeli objektumok általában két fő komponensből állnak:

- *térbeli komponens*, amelyet grafikusan jelenítünk meg (például telek határvonalai),
- *leíró komponens*, amelyet rendszerint táblázatosan jelenítünk meg (például telek adatai).

Ebben a fejezetben a térbeli komponens vektoros kezelését vizsgáljuk, majd a további fejezetekben a két komponens együttes kezelését biztosító adatbázismodelleket tekintjük át.

3.2. Alapvető objektumtípusok

A térbeli objektumokat az alábbi csoportokba sorolhatjuk (7. ábra):

(i) pontszerű (0D) objektum: térbeli helye x, y koordinátával adott. Két típusa van:

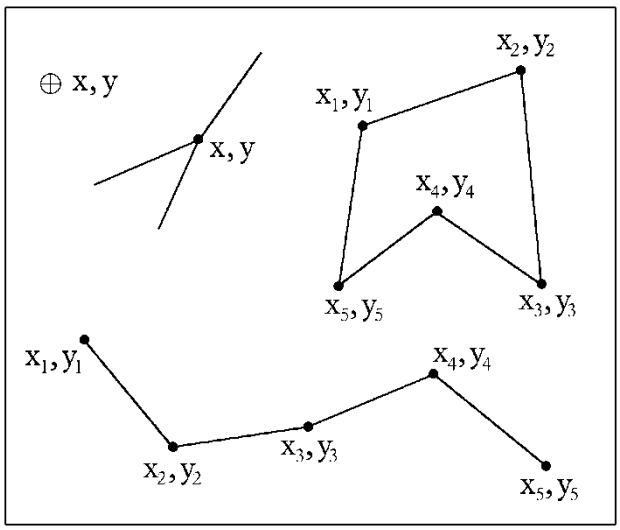
- kisméretű objektum: az adott méretarány mellett túl kicsi a grafikus ábrázoláshoz (például nagy méretarányú térképen lámpaoszlop, vagy kis méretarányú térképen település). Megjelenítése meghatározott jelkulcsi jelöléssel történik.

– csomópont: vonalas objektumok találkozási helyét jelöli (pl. útelágazás). Rendszerint nincs külön grafikus megjelenítése.

(ii) vonalas (1D) objektum: általában vonallánccal adott: x_1, y_1, ..., x_n, y_n (például vasútvonal). Megjelenítése adott színnel és vonaltípussal történik. Vonalon a továbbiakban általában vonalláncot értünk, amely speciális esetként az egyenesszakaszt is magában foglalja.

(iii) területi (2D) objektum: általában poligonnal adott: x_1, y_1, ..., x_n, y_n (például telek). Megjelenítése adott kitöltő mintázattal vagy színnel lehetséges.

Referencia pont: a 2D objektum egy kijelölt belső pontja, amelyhez pl. felirat rendelhető. Konvex poligon esetén a súlypontot célszerű választani, konkáv esetben azonban ez kívül eshet a poligonon.



7. ábra: Vektoros objektumtípusok: jelkulcsi elem és csomópont (0D), vonallánc (1D), poligon (2D).

3.3. A méretarány kérdése

A digitális térkép lényegében egy térbeli adatbázis, amely tetszés szerinti nagyításban térképként megjeleníthető ill. nyomtatható, így a méretarány jelentősége itt megváltozik. Egy digitális térkép méretaránya alapvetően két dolgot határoz meg:

– *mely objektumtípusokat* tartalmaz az adatbázis (pl. kis méretarányú térkép nem tartalmaz épületeket),

– az adott objektumok *milyen adattartalommal tárolódnak* (pl. egy települést kis méretarány esetén pontszerűen (0D objektum), nagyobb méretarány esetén poligonnal (2D objektum) ábrázolunk).

Ez azt jelenti, hogy egy kis méretarányú digitális térképet nincs értelme nagyobb méretarányban nyomtatni, hisz a szükséges adattartalom hiányzik belőle.

Ha egy területet erősen különböző méretarányokban kell kezelni (nagyítás, kicsinyítés), akkor két megoldási lehetőség kínálkozik:

(i) Több különálló, egymásra épülő digitális térképet készítünk. Például egy városi GIS rendszer esetén készíthető egy kis méretarányú áttekintő térkép, amely csak a kerületek körvonalait és a főbb utakat tartalmazza, míg a nagy méretarányú térképen már az épületek körvonalai, házszámok, stb. is látható. A módszer hátránya, hogy ha például egy kerület határa megváltozik, akkor a változást két különálló adatstruktúrán kell átvezetni.

(ii) A méretarány változtatásával fokozatosan ki/bekapcsoljuk az egyes rajzi rétegek megjelenítését. Például egy országos úthálózat esetén kis méretaránynál csak az országos főútvonalak rétegét kapcsoljuk be, majd a méretarány növelésével fokozatosan bekapcsoljuk a másodrendű és harmadrendű utakat, végül a földutakat is.

3.4. Spagetti modell

Ha a digitális térképet vektoros objektumok egyszerű halmazaként kezeljük, akkor *spagetti modell*ről beszélünk. (A kissé ironikus elnevezés arra utal, hogy vonalláncok rendezetlen halmaza egy tál spagettire emlékeztet.) Ilyenkor a vektoros objektumok egymástól függetlenül tárolódnak, közöttük nincs hivatkozási kapcsolat. Ilyen a CAD rendszerek és egyes térinformatikai rendszerek adatstruktúrája.

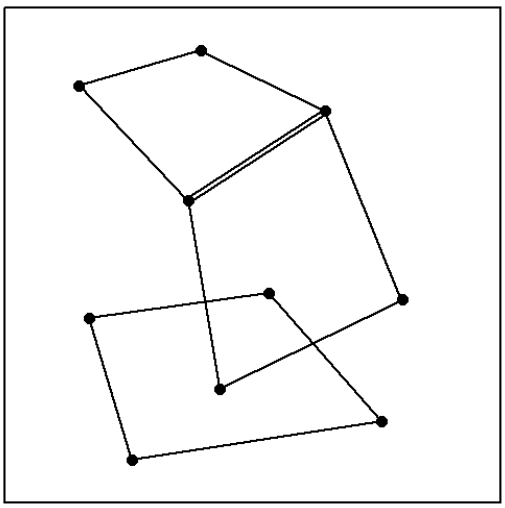
Az adatstruktúra előnye, hogy könnyen kezelhető, az egyes térbeli objektumok egymástól függetlenül módosíthatók. Ezért az előnyért viszont számos hátránnyal kell fizetni:

a) Az egymást metsző vonalak metszéspontjában nem feltétlenül van csomópont (a vonalak ilyenkor „nem tudják”, hogy metszik egymást, lásd 8. ábra).

b) A szomszédos poligonok (például telkek) határvonala kétszer tárolódik, ami egyrészt redundanciát jelent, másrészt módosításkor zavarokat okozhat.

c) Nehéz az adatok integritásának ellenőrzése. Például egy poligonokból álló megyetérképen nem könnyű eldönteni, hogy a poligonok hézag- és átfedésmentesen fedik-e le a területet.

A fenti hátrányokat a topológikus modellek küszöbölik ki.



8. ábra: A spagetti modell

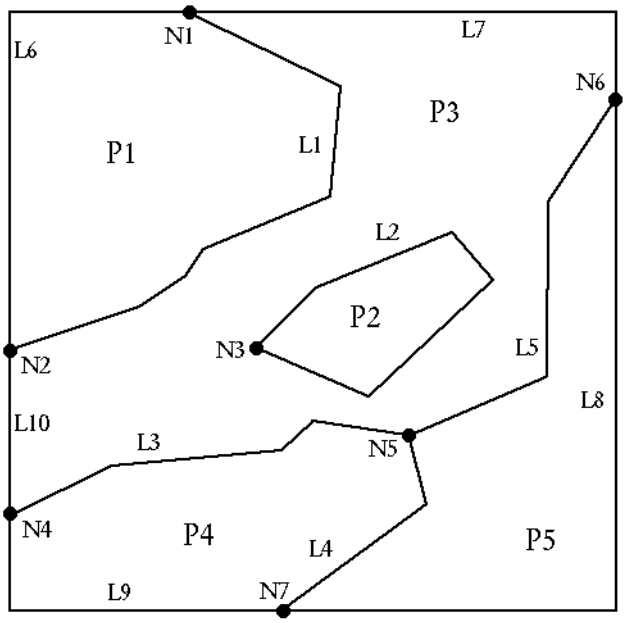
3.5. Topológikus modellek

Ha az adatstruktúra nem csak a rajzi objektumokat, hanem azok térbeli kapcsolódási struktúráját (azaz a topológiát) is tartalmazza, akkor topológikus adatmodellről beszélünk. Az ilyen modelleknél általában minden rajzelemnek egyedi azonosítója (*identifier*, röviden *id*) van, ennek segítségével az egyes rajzelemek egymásra hivatkozhatnak. A topológikus modellek természetes módon biztosítják az adatintegritást, viszont az adatkezelés körülményesebb.

A következőkben három jellegzetes topológikus adatstruktúrát ismertetünk: a tartománytérképet, a hálózatot és a folytonos felületek leírására használható TIN modellt.

3.5.1. Tartománytérkép (folttérkép)

Egy adott területet diszjunkt tartományokkal (foltokkal) hézagmentesen fedünk le (pl. talajtérkép, megyetérkép). Két tartomány határvonalát 1D objektumként, az egyes tartományokat 2D objektumként tároljuk (9. ábra). A határoló vonalláncok nem metszhetik sem önmagukat, sem egymást.



9. ábra: Tartománytérkép. A csomópontokat N_i, a vonalakat L_i, a poligonokat P_i jelöli.

A tartományok szigeteket tartalmazhatnak (lásd P2 a 9. ábrán), amelyek területe nem tartozik a tartományhoz – hiszen a diszjunktság csak így teljesül. A szigetek megkülönböztetett figyelmet igényelnek mind az adatstruktúra, mind az algoritmusok tekintetében.

Alább egy tipikus tartománytérkép adatstruktúrát mutatunk be, ehhez hasonlót használ az ArcInfo rendszer is.

NODE: csomópont tömb, egy rekordjának felépítése:

id	: csomópont azonosító
x, y	: koordináták
[attribútumok]	

LINE: határvonal tömb, egy rekordjának felépítése:

id	: vonal azonosító
node1	: kezdő csomópont azonosítója
node2	: záró csomópont azonosítója
lpoly	: baloldali poligon azonosítója
rpoly	: jobboldali poligon azonosítója
$x_1, y_1,...,x_n, y_n$	: a vonallánc belső töréspontjainak koordinátái
[attribútumok]	

POLYGON: tartomány tömb, egy rekordjának felépítése:

id	: tartomány azonosító
$line_1,..., line_n$	: határvonalak azonosítói
[attribútumok]	

A 9. ábrán látható tartománytérképet leíró adatstruktúra alább látható. Figyeljük meg a sziget leírását!

NODE:

id	*x*	*y*
N1	x_1	y_1
N2	x_2	y_2
N3	x_3	y_3
N4	x_4	y_4
N5	x_5	y_5
N6	x_6	y_6
N7	x_7	y_7

LINE:

id	*node1*	*node2*	*lpoly*	*rpoly*	$x_1,y_1, ..., x_n,y_n$
L1	N1	N2	P3	P1	...
L2	N3	N3	P3	P2	...
L3	N4	N5	P3	P4	...
L4	N5	N7	P5	P4	...
L5	N5	N6	P3	P5	...
L6	N1	N2	P1	P0	...
L7	N1	N6	P0	P3	...
L8	N6	N7	P0	P5	...
L9	N4	N7	P4	P0	...
L10	N2	N4	P3	P0	...

POLYGON:

id	$line_1, ..., line_n$
P1	L1, L6
P2	L2
P3	L1, L7, L5, L3, L10, L2
P4	L3, L4, L9
P5	L4, L5, L8

Megjegyzések:

- Az lpoly és rpoly azonosítók a tartománytérképek hatékony algoritmikus kezelését szolgálják. Pontos jelentésük: az adott határvonal az lpoly és rpoly tartományokat választja el, éspedig ha a node1 kezdőpontból haladunk a node2 záró pont felé, akkor lpoly bal oldalon, rpoly pedig jobb oldalon fekszik.
- Ha a poligon szigete(ke)t tartalmaz, akkor a sziget határvonalait is fel kell venni a POLYGON rekord listájára.

Alább összefoglaljuk az adatstruktúrát, dőlt betűkkel jelölve az egyes komponensek egymásra való hivatkozásait:

NODE: id, x, y
LINE: id, *node1*, *node2*, *lpoly*, *rpoly*, x_1, y_1, ..., x_n, y_n
POLYGON: id, $line_1$, ..., $line_n$

Topológikus modellben az adatok módosítása bonyolultabbá válik. Ha például egy telek poligont ketté kell osztani, akkor

- új NODE-okat kell felvenni,
- a megfelelő határoló LINE-okat ketté kell osztani az új NODE-okra való hivatkozással,
- a POLYGON tömbben a régi telek poligonját törölni kell és két új poligonnal helyettesíteni a megfelelő LINE-okra való hivatkozással,
- végül a határoló LINE-ok lpoly és rpoly hivatkozásait módosítani kell.

3.5.2. Hálózat

0D és 1D típusú objektumok rendszere (pl. úthálózat, folyóhálózat, csőhálózat). Jellemző típusai:

a) Geometriai hálózat: a térbeli viszonyokat pontosan leképezi. Lehetséges felépítése:

NODE: csomópont tömb, egy rekordjának felépítése:

id	: csomópont azonosító
x, y	: koordináták
e_1,...,e_n	: kiinduló élek azonosítói. Az adatkezelés gyorsítását szolgálják, akár el is hagyhatók.

[attribútumok, pl. úthálózat esetén van-e közlekedési lámpa]

LINE: vonallánc tömb, egy rekordjának felépítése:

id	: él azonosító
$node_1$	: kezdő csomópont azonosítója
$node_2$	: záró csomópont azonosítója
x_1, y_1,...,x_n, y_n	: töréspontok koordinátái

[attribútumok, pl. forgalom iránya]

A vonalláncok csomópont nélkül is keresztezhetik egymást (pl. felüljáró). A tartomány-térképnél ez nem volt megengedett.

b) Logikai hálózat: absztrakt gráf, amely csak a kapcsolódási struktúrát tárolja, térbeli információ nélkül. Lehetséges felépítése:

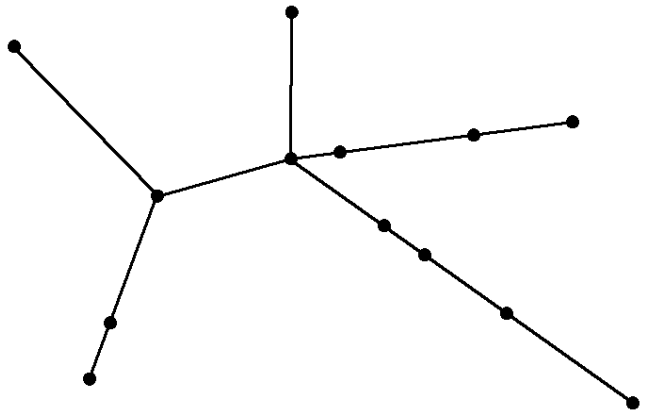
NODE: csomópont tömb, egy rekordjának felépítése:
- id : csomópont azonosító
- $e_1,...,e_n$: kiinduló élek azonosítói
- [attribútumok]

EDGE: él tömb, egy rekordjának felépítése:
- id : él azonosító
- $node_1$: kezdő csomópont azonosítója
- $node_2$: záró csomópont azonosítója
- [attribútumok, pl. az él hossza]

c) Félig geometriai hálózat: a logikai és geometriai hálózat kombinációja: a csomópontok térbeli koordinátákkal adottak, az élek geometriájától viszont eltekintünk. Lehetséges felépítése:

NODE: csomópont tömb, egy rekordjának felépítése:
- id : csomópont azonosító
- x, y : koordináták
- $e_1,...,e_n$: kiinduló élek azonosítói
- [attribútumok]

EDGE: él tömb, egy rekordjának felépítése:
- id : él azonosító
- $node_1$: kezdő csomópont azonosítója
- $node_2$: záró csomópont azonosítója
- [attribútumok, pl. az él tényleges hossza]



10. ábra: Félig geometriai hálózat

Példa félig geometriai hálózatra: vasúthálózat. Például a Szeged-Kiskunfélegyháza szakasz egyetlen él, annak ellenére, hogy a valósághű megjelenítéshez vonallánccal kellene leírni, és a közbülső állomások miatt több élre kellene bontani (10. ábra). Mivel azonban

Szeged és Kiskunfélegyháza között nincs elágazás a vasúti pályán, célszerű elkerülni a több élre bontást. Kérdés, hogyan tartsuk nyilván a közbülső állomásokat és egyéb pályamenti objektumokat (híd, útátjáró, őrház stb.)? Erre megoldás a lineáris címzés módszere.

Lineáris címzés módszere: egy élen belül objektumok azonosítása. A hidakat és egyéb pályamenti létesítményeket külön táblában tartjuk nyilván:

Objektumok (obj_id, megnevezés, edge_id, dist1, dist2)

ahol *obj_id* az objektum azonosítója, *edge_id* az objektumot tartalmazó él, *dist1* az objektum kezdetének, *dist2* a végének az él kezdőpontjától mért távolsága. Ekkor nem szaporodnak el az élek, és nem lassul a feldolgozás, ha az éleken lévő objektumokra nem vagyunk kíváncsiak. Példa objektum-táblára:

obj_id	*megnevezés*	*edge_id*	*dist1*	*dist2*
O1	híd	E3	145	147
O2	útátjáró	E3	192	192
O3	őrház	E4	48	48
O4	vágányjavítás	E5	216	231

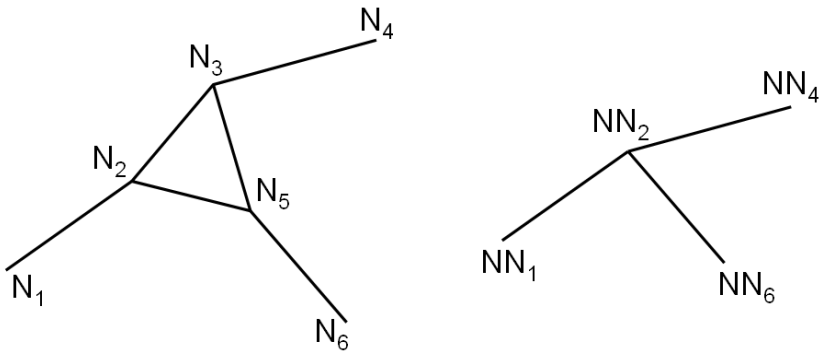
Hasonló módszerrel kezelhető a *postacím geokódolás*. Itt egy utcahálózat adatstruktúrát használunk, ahol minden egyes utcát reprezentáló élhez az utcanév, minhsz, maxhsz attribútumokat csatoljuk (minhsz és maxhsz az utca kezdő- és végpontjához tartozó házszámok). Ha ezek után egy adott utca adott számú házának koordinátáit szeretnénk megkapni, akkor ez az utcának megfelelő élen lineáris interpolációval megtehetjük.

d) Hierarchikus hálózat: több szintű hálózat, amely áll egy teljes részletességű alaphálózatból (1. szint), és ennek leegyszerűsített, absztrakt változataiból (2. szint, stb.). Lehetséges felépítése:

NODE: csomópont tömb, egy rekordjának felépítése:
- id : csomópont azonosító
- x, y : koordináták
- level : a csomópont szintszáma
- parent_node : a magasabb hierarchiaszinten a megfelelő csomópont id-je
- $e_1,...,e_n$: kiinduló élek azonosítói
- [attribútumok]

LINE: vonallánc tömb, egy rekordjának felépítése:
- id : él azonosító
- $node_1$: kezdő csomópont azonosítója
- $node_2$: záró csomópont azonosítója
- parent_line : a magasabb hierarchiaszinten a megfelelő vonallánc id-je
- $x_1, y_1,...,x_n, y_n$: töréspontok koordinátái
- [attribútumok]

Például úthálózat modellezése esetén egy város belső utcahálózatát magasabb szinten eltakarjuk. A 11. ábrán N_2, N_3 és N_5 jelképezi a várost, ezekhez a csomópontokhoz egyaránt az NN_2 parent_node tartozik.



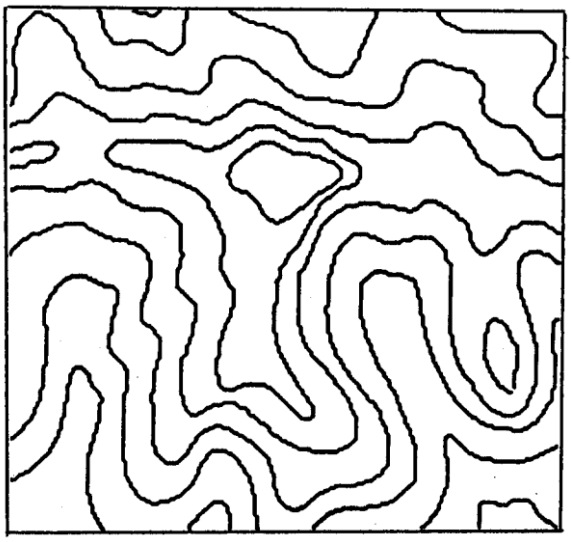
11. ábra: Példa hierarchikus hálózatra. Bal oldalt az 1. szint, jobb oldalt a 2. szint

Hálózatokra sokféle alkalmazás épülhet, alább felsorolunk néhány példát:

- Úthálózat, vasúthálózat: menetrend lekérdezése útkereséssel és átszállások kezelésével.
- Két pont között az optimális útvonal megkeresése adott súlyú és magasságú jármű számára.
- Csővezeték-hálózat: áramlás modellezése.
- Természetes folyóvizek hálózata: árvíz előrejelzés.
- Városi tömegközlekedés optimalizálása (járatok útvonala, sűrűsége, átszállásszám minimalizálása, stb.)

3.5.3. Folytonos felület

Folytonos felületek (pl. domborzat, hőmérséklet eloszlása, stb.) modellezése valamely f(x, y) függvénnyel lehetséges, ahol a függvény értéke az (x, y) koordinátájú pont magasságát, hőmérsékletét, stb. adja. Ilyen folytonos felület vektoros ábrázolására két mód nyílik:



12. ábra: Izovonalas ábrázolás

a). Izovonalas ábrázolás (12. ábra). Izovonalon olyan vonalláncot értünk, amely mentén az f(x, y) függvény értéke állandó (domborzat esetén ez a szintvonal), ezért a vonalhoz attribútumként a megfelelő függvényérték rendelhető. Szokásos ábrázolási módjai:

- vonallánc attribútummal: LINE x_1, y_1, ..., x_n, y_n, f,
- 3D-vonallánc: LINE x_1, y_1, z_1, ..., x_n, y_n, z_n (itt z_1 = ... = z_n, ami redundanciát jelent, ugyanakkor 3D-modellezésnél előnyös lehet ez a megoldás).

b). TIN (= Triangulated Irregular Network): szabálytalan háromszögrács, vagyis sík háromszöglapokkal közelítjük a felszínt (13. ábra). Példaként két lehetséges tárolási módot mutatunk be:

(i) Háromszögenkénti tárolás: a TRIANGLE és NODE tömbökből áll.

A NODE tömb egy rekordjának felépítése:

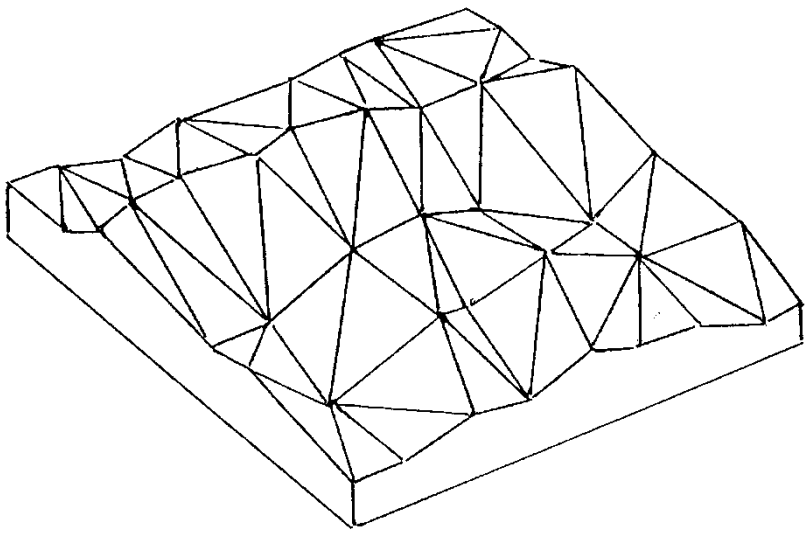
- id: szögpont azonosító száma,
- x, y, z: szögpont koordinátái (z az f(x, y) függvény adott pontbeli értéke).

A TRIANGLE tömb egy rekordjának felépítése:

- id: a háromszög azonosító száma,
- $node_1$, $node_2$, $node_3$: a három szögpont azonosító számai,
- tr_1, tr_2, tr_3: a szomszédos háromszögek azonosító számai.

(ii) Szögpont-szomszédság szerinti tárolás: csak egy NODE tömbből áll, ahol egy rekord tartalma:

- id: szögpont azonosító száma,
- x, y, z: szögpont koordinátái,
- $node_1$, ..., $node_n$: szomszédos (éllel összekötött) szögpontok azonosító számai.



13. ábra: TIN modell

3.6. Spagetti vagy topológikus modell?

A címben feltett kérdés korunk térinformatikájának egyik nagy dilemmája. Elméleti szempontból egyértelmű a topológikus megközelítés fölénye. Gyakorlati szempontból azonban súlyos érvek szólnak az egyszerűbb és könnyebb adatkezelést biztosító spagetti modell mellett.

Tanulságos ebből a szempontból a világ vezető térinformatikai fejlesztőjének, az ESRI-nek (Environmental Systems Research Institute, USA) a tapasztalata. A cég hosszú éveken keresztül a topológikus modell elkötelezettje volt, az ArcInfo rendszerük alapvetően és deklaráltan erre épült. Azután megjelent az ArcInfo „kistestvére", az ArcView, amely csökkentett funkcionalitással és a topológiát feladó, egyszerű és praktikus ún. *shapefile* adatstruktúrával (lásd a Függelékben) átütő sikert hozott. Ez a fejlesztőket arra késztette, hogy új rendszerük, az ArcGIS már lazábban kezelje a topológiát: az alap-adatstruktúra nem topológikus, viszont külön követelményrendszer formájában adják meg a topológiát [Zeiler, 1999].

A tapasztalat tehát az, hogy a spagetti és a topológikus megközelítés egyaránt létjogosultsággal bír [Yeung és Hall, 2007], gyakran „félig topológikus" megoldások formájában ölt testet.

3.7. Vektoros algoritmusok

Ebben a fejezetben alapvető vektoros algoritmusokat mutatunk be [NCGIA 1994, Rigaux és tsai 2002]. Az egyszerűbb algoritmusokat részletesen leírjuk, az összetettebbeknél csak az alapelvekre koncentrálunk.

3.7.1. Egyenesszakaszok metszéspontja

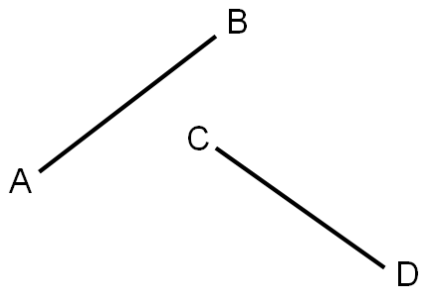
Az A = (a_x, a_y) és B = (b_x, b_y) pontokat összekötő, valamint a C = (c_x, c_y) és D = (d_x, d_y) pontokat összekötő egyenesszakaszok Z = (x, y) metszéspontját számítjuk (14. ábra). Először az A és B, valamint a C és D pontokon áthaladó végtelen egyenesek metszéspontját határozzuk meg. A megoldandó egyenletrendszer:

$$(b_y-a_y)*x - (b_x-a_x)*y = a_x*b_y - a_y*b_x$$
$$(d_y-c_y)*x - (d_x-c_x)*y = c_x*d_y - c_y*d_x$$

Szakaszok metszéspontjának meghatározásakor a fenti egyenletrendszer megoldásaként kapott Z = (x, y)-t csak akkor fogadjuk el, ha A és B közé, ill. C és D közé esik. Ez akkor igaz, ha az alábbi négy feltétel egyidejűleg teljesül:

$$(x - a_x)*(x - b_x) \le 0$$
$$(y - a_y)*(y - b_y) \le 0$$
$$(x - c_x)*(x - d_x) \le 0$$
$$(y - c_y)*(y - d_y) \le 0$$

Az első feltétel azt fejezi ki, hogy x az [a_x, b_x] zárt intervallumba esik, hasonlóan értelmezendő a többi feltétel. Ha az AB szakasz függőleges, akkor az első feltétel olyan esetben is teljesül, amikor Z kívül esik AB-n, ezért szükséges a második feltétel is. Hasonló meggondolásból van szükség mind a négy feltételre.



14. ábra: Egyenesszakaszok metszése.

Ha sok egyenesszakasz metszését kell elvégezni (például, ha spagetti adatstruktúrán topológiát építünk), akkor *n* egyenesszakasz esetén n*(n-1)/2 metszésvizsgálatot kellene végezni. Ugyanakkor a vizsgált egyenesszakaszok gyakran távol vannak egymástól, így eleve nem lehet metszés. Ezért célszerű a számítást úgy gyorsítani, hogy először a vizsgált két egyenesszakasz befoglaló téglalapját határozzuk meg: ha a befoglaló téglalapok diszjunktak, akkor az egyenesszakaszok sem metszhetik egymást. Az AB szakasz P befoglaló téglalapját a bal alsó sarok (p_{x1}, p_{y1}), és a jobb felső sarok (p_{x2}, p_{y2}) koordinátáival adjuk meg, hasonlóan a CD szakasz Q befoglaló téglalapjára (15. ábra).

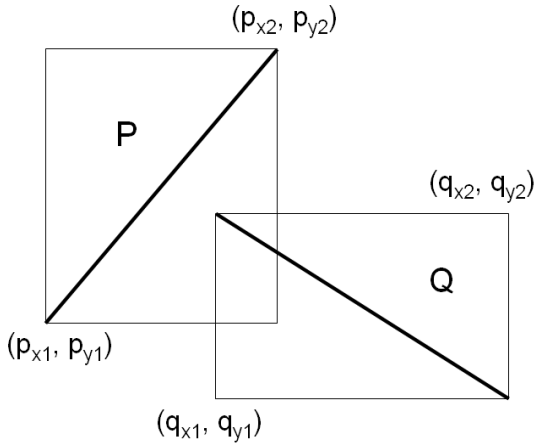
$p_{x1} = \min(a_x, b_x)$ $\quad q_{x1} = \min(c_x, d_x)$
$p_{y1} = \min(a_y, b_y)$ $\quad q_{y1} = \min(c_y, d_y)$
$p_{x2} = \max(a_x, b_x)$ $\quad q_{x2} = \max(c_x, d_x)$
$p_{y2} = \max(a_y, b_y)$ $\quad q_{y2} = \max(c_y, d_y)$

A befoglaló téglalapok akkor és csak akkor diszjunktak, ha

$$p_{x2} < q_{x1} \ \textbf{vagy} \ q_{x2} < p_{x1} \ \textbf{vagy} \ p_{y2} < q_{y1} \ \textbf{vagy} \ q_{y2} < p_{y1}$$

A fenti feltételben $p_{x2} < q_{x1}$ azt fejezi ki, hogy a P téglalap jobb széle balra van a Q téglalap bal szélétől, vagyis a téglalapok diszjunktak – hasonlóan értelmezhető a többi feltétel.

Ha a téglalapok metszik egymást, még nem biztos, hogy a szakaszok is metszik egymást (15. ábra), ilyenkor tehát a metszésvizsgálatot el kell végezni.



15. ábra: Befoglaló téglalapok

3.7.2. Vonalláncok metszése

Ha az L és M vonalláncok n ill. m egyenes szakaszból állnak, akkor n*m metszésvizsgálatot kellene végezni. A számításigény csökkenthető befoglaló téglalapok vizsgálatával:

– Először a teljes vonalak befoglaló téglalapjait határozzuk meg. Az $L = (x_1, y_1,...,x_n, y_n)$ vonal befoglaló téglalapja a $(p_{x1}, p_{y1}, p_{x2}, p_{y2})$ négyessel adható meg, ahol

$p_{x1} = \min(x_1,...,x_n)$
$p_{y1} = \min(y_1,...,y_n)$
$p_{x2} = \max(x_1,...,x_n)$
$p_{y2} = \max(y_1,...,y_n)$

hasonlóan adódik a $(q_{x1}, q_{x2}, q_{y1}, q_{y2})$ befoglaló téglalap.

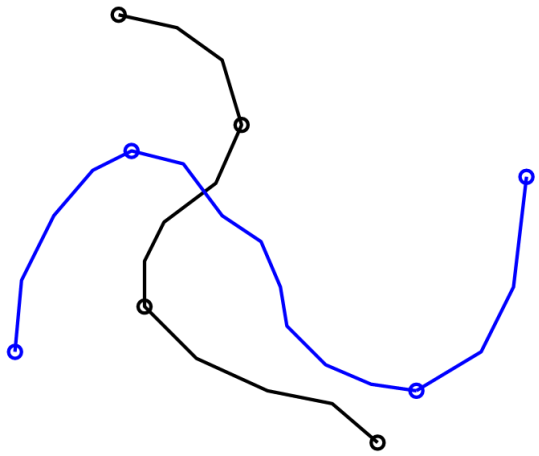
– Megvizsgáljuk, hogy a befoglaló téglalapok metszik-e egymást. Ha ugyanis a téglalapok diszjunktak, akkor a vonalak sem metszhetik egymást. Amint már láttuk, a téglalapok akkor és csak akkor diszjunktak, ha

$$p_{x2} < q_{x1} \ \textbf{vagy} \ q_{x2} < p_{x1} \ \textbf{vagy} \ p_{y2} < q_{y1} \ \textbf{vagy} \ q_{y2} < p_{y1}$$

A számításigény tovább csökkenthető, ha a vonalakat *monoton szakaszokra* osztjuk. Egy $x_i, y_i,...,x_j, y_j$ vonalszakaszt monotonnak nevezünk, ha mind az x, mind az y koordináták monoton nőnek vagy csökkennek (16. ábra). Négyféle monoton szakasz lehetséges:

1. x növekszik és y növekszik,
2. x növekszik és y csökken,
3. x csökken és y növekszik,
4. x csökken és y csökken.

Monoton szakaszok metszésvizsgálata hatékonyabban végezhető: például egy 1. típusú és egy 2. típusú szakasz legfeljebb egyszer metszheti egymást, és a metszéspontot is könnyebb megtalálni a monoton koordinátákon való lépkedéssel.

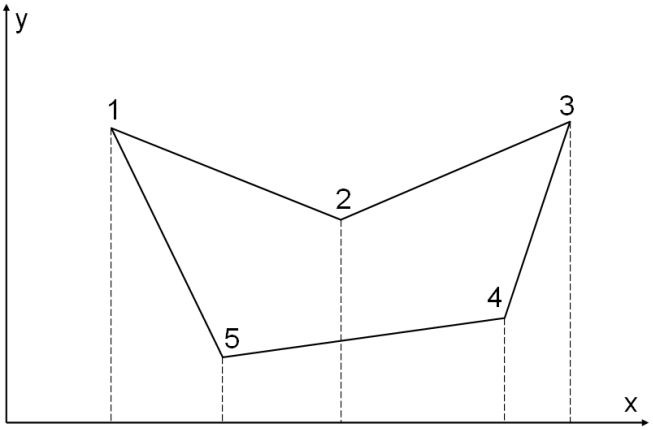


16. ábra: Monoton szakaszok

3.7.3. Poligonok területe

Tekintsük az $x_1, y_1,...,x_n, y_n$ pontok által meghatározott zárt poligont ($x_{n+1} = x_1$, $y_{n+1} = y_1$). Feltételezzük, hogy a poligon önmagát nem metszi.

Bocsássunk függőleges egyenest minden szögpontból az x tengelyre, és számítsuk ki minden élhez az él, a függőleges egyenesek és az x tengely által bezárt trapéz (előjeles) területét (17. ábra): $T_i = (x_{i+1} - x_i) * (y_{i+1} + y_i) / 2$



17. ábra: Poligon területének számítása

Az ábrán jól látható, hogy a fenti formula T_1 és T_2 esetén pozitív, T_3, T_4 és T_5 esetén viszont negatív értéket ad. Ez azt jelenti, hogy a poligon területe éppen az egyes trapézok előjeles területének összegeként adódik:

$$T = \sum_{i=1}^{n} T_i$$

A fenti formula tetszőleges konvex vagy konkáv poligon esetén helyes eredményt ad, feltéve, hogy minden y érték pozitív. Amennyiben negatív y értékek is fellépnek, az y koordinátákra $y_i' = y_i - \min(y_1,..., y_n)$ eltolást kell alkalmazni.

Ha a poligont óramutató járása szerint járjuk körül, akkor pozitív, egyébként negatív terület adódik. A tényleges területérték tehát $|T|$.

Ha sok poligon területét kell egyidejűleg meghatározni, akkor a 3.5.1. alfejezetben ismertetett tartománytérkép adatstruktúra esetén a következő megoldás célszerű:

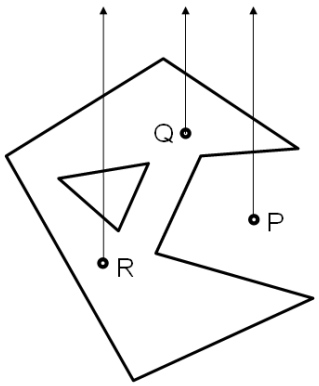
- Minden vonalhoz egy (előjeles) „területértéket” számolunk a fenti formula segítségével.
- Az egyes poligonok területét a határoló vonalak „területének” összegeként kapjuk, ügyelve arra, hogy ha az adott poligon egy vonalnak bal poligonja, akkor a vonalhoz rendelt terület (–1)-szeresével számolunk.

Ez az eljárás helyes eredményt ad lyukak (szigetek) esetén is, sőt akkor is, ha a poligonok vonallistája rendezetlen!

3.7.4. Pont-poligon algoritmus

Feladat: el kell dönteni, hogy egy adott (u, v) pont egy x_1, y_1,..., x_n, y_n poligon belsejében van-e ($x_{n+1} = x_1$, $y_{n+1} = y_1$). A poligon szigetet is tartalmazhat.

Megoldás elve: rajzoljunk a pontból függőlegesen felfelé egy félegyenest! Ha ez páratlan számú helyen metszi a poligont, akkor a pont belül van, egyébként kívül van. Ez akkor is igaz, ha a poligon sziget(ek)et tartalmaz (18. ábra).

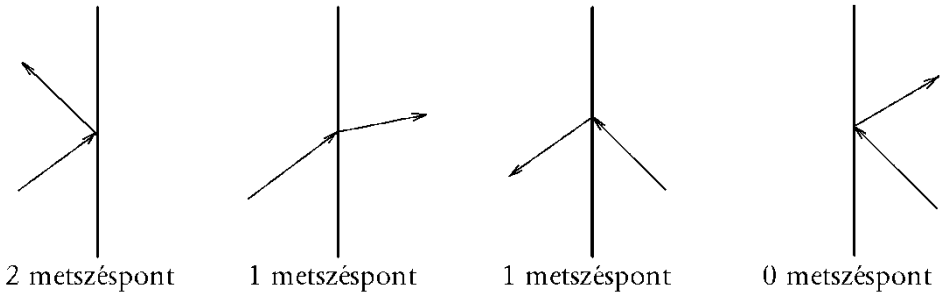


18. ábra: Pont poligonba esésének vizsgálata

Problematikus esetek kezelése:

(i) Ha a poligon egy éle függőleges, akkor nem számítunk metszéspontot (akkor sem, ha az él éppen fedésben van a félegyenessel).

(ii) Ha a félegyenes a poligon egy szögpontján halad át, akkor csak a balról érkező, ill. balra távozó él esetén számítunk metszéspontot (19. ábra).



19. ábra: Lehetséges esetek, ha a félegyenes szögponton halad át

Algoritmus: Ha az alábbi algoritmus lefutása után a 'val' változó értéke 1, akkor a pont belül, ha -1, akkor kívül van, ha 0, akkor a határon van:

```
val := -1
for i=1 to n
if xi+1 != xi then  // nem függőleges él, lásd (i) eset
   if (xi+1-u)*(u-xi) >= 0 then// u ∈ [xi,xi+1]
      if xi+1 != u or xi <= u then   // nem jobbról jön, lásd (ii) eset
         if xi != u or xi+1 <= u then  // nem jobbra megy, lásd (ii) eset
            b := (yi+1-yi)/(xi+1-xi) // iránytangens
            metsz := yi + b*(u-xi)
            if metsz > v then val := val*(-1)
```

```
            if metsz = v then val := 0  // határon van
          end if
        end if
    end if
end if
next i
```

Sok pont, sok poligon vizsgálata

Tartománytérkép esetén nyilvánvaló, hogy egy adott pont egy és csak egy poligonba eshet bele. Tegyük fel, hogy adott Z_1,..., Z_n pontokról el kell dönteni, hogy melyik pont melyik poligonba esik. Az algoritmust a korábban tárgyalt tartománytérkép adatstruktúrán mutatjuk be.

Itt is minden pontból függőlegesen felfelé félegyenest indítunk, és minden félegyenesnek minden vonallal megvizsgáljuk a metszéspontját (metszéspontjait). A Z_1,..., Z_n pontokhoz az m_1,..., m_n és p_1,..., p_n segédtömböket vesszük fel, ahol m_i a legalsó metszéspont y koordinátáját, p_i pedig a metszéspont alatti poligon azonosítóját fogja tartalmazni. Kezdetben minden p_i definiálatlan, m_i értéke „végtelen”.

Az algoritmus az L_1, ..., L_m vonalakon megy végig:

```
For L = L1,...,Lm
  xmin := L minimális x-koordinátája
  xmax := L maximális x-koordinátája
  ymin := L minimális y-koordinátája
  ymax := L maximális y-koordinátája
  for i=1 to n      // végig a pontokon, Zi koordinátái (xi,yi)
    if (xmin ≤ xi ≤ xmax) then
      if (yi < ymax and ymin < mi)
        metsz := Zi-ből induló függőleges félegyenes L-lel való
                 legalsó metszéspontjának y-koordinátája
        if pi=-1 or mi>metsz then
          mi := metsz
          pi := alsó_poligon
        end if
      end if
    end if
  next i
next L
```

A fenti algoritmusban „alsó_poligon” a metszéspont alatti poligon azonosítója, amelyet a vonal lpoly, rpoly paramétereiből tudunk meghatározni attól függően, hogy a vonal balról jobbra vagy jobbról balra halad át a függőleges félegyenesen. Amikor az összes vonallal végeztünk, a Z_i ponthoz a p_i tömbelem tartalmazza a legalsó metszéspont alatti poligon azonosítóját, vagyis azét, amely a pontot tartalmazza. A pontok x koordináta szerinti rendezésével az algoritmus gyorsítható.

3.7.5. Poligonok metszése

Hogyan lehet eldönteni, hogy két poligon, P és Q, metszi-e egymást? Megállapítható, hogy akkor és csak akkor metszik egymást, ha

- van metsző élük, vagy
- egyik tartalmazza a másikat.

Először tehát metsző éleket keresünk. Ha nem találunk, akkor megvizsgáljuk, hogy P

egy tetszőleges pontja Q belsejében van-e (ekkor Q tartalmazza P-t), vagy fordítva.

Következő feladat a metszet poligon(ok) meghatározása. (Két poligon metszete ugyanis poligonhalmaz is lehet, ha legalább az egyik poligon konkáv.) Ha P tartalmazza Q-t, akkor a metszet megegyezik Q-val, és fordítva. A továbbiakban tehát csak azt az esetet kell vizsgálni, amikor P-nek és Q-nak van metsző éle. Az eljárás a következő:

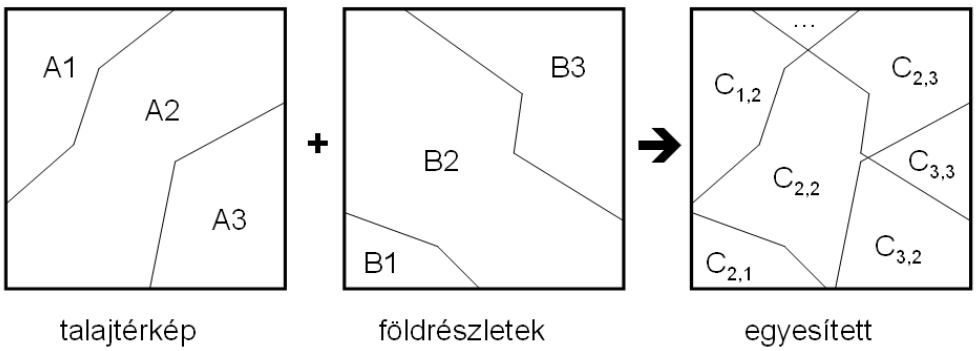
1. Metszéspontok meghatározása, felvételük a poligonok pontlistájára.
2. Szögpontok osztályozása (belső, külső, határ) pont-poligon algoritmussal. Egy szögpont belső, ha a másik poligon belsejében van, külső, ha azon kívül van, és határ, ha a másik poligon határán van.
3. Élek osztályozása (belső, külső, határ). Egy él belső, ha mindkét végpontja belső, vagy egyik belső, másik határ. A külső élek hasonlóan határozhatók meg. Ha viszont egy él mindkét végpontja határon van, akkor az él egy belső pontját kell vizsgálni.
4. Belátható, hogy a metszet-poligon(oka)t a belső és határ élek alkotják.

Két poligon metszetének meghatározása tehát nem is olyan egyszerű feladat. Ezek után meglepő, hogy két poligonhalmaz – nevezetesen két tartománytérkép – metszése viszonylag hatékonyan megoldható, lásd a következő alfejezetet.

3.7.6. Poligon overlay algoritmus

Példa. Tekintsünk két tartománytérképet: egy *talajtérkép*et, amelyen az egyes tartományok adott talajminőségű területeket kódolnak, és egy *kataszteri térkép*et, amely földrészleteket tartalmaz (20. ábra).

Meg kell határozni az egyes földrészletek értékét, amelyet a rá eső különböző talajminőségű területek súlyozott összegével számíthatunk. Például a B_2 földrészlet értéke $a_1*t_{1,2} + a_2*t_{2,2} + a_3*t_{3,2}$, ahol a_i az A_i területhez tartozó talajminőség értéke, $t_{i,j}$ pedig az A_i és B_j metszeteként kapott $C_{i,j}$ tartomány területe.



20. ábra: Példa poligon-overlay műveletre

Ehhez az alábbi feladat megoldása szükséges, amelyet *poligon overlay* (más néven *poligon fedvényezés*) műveletnek neveznek.

Feladat. Adott két, diszjunkt tartományokkal lefedett fedvény:

$$A = A_1 \cup \ldots \cup A_n$$

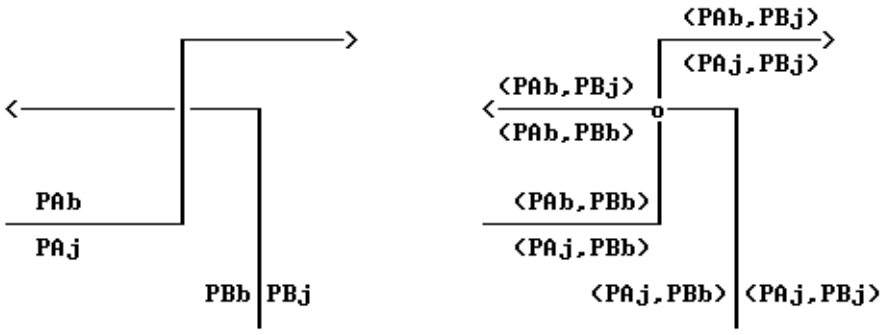
$$B = B_1 \cup \ldots \cup B_m$$

ahol az egyes tartományokhoz rendre az $a_1,\ldots,a_n$ ill. $b_1,\ldots,b_m$ attritútumok tartoznak. A metszet fedvényt kell képezni, vagyis olyan $C = C_1 \cup \ldots \cup C_z$ fedvényt, ahol a lefedő tartományok szintén diszjunktak, és C_k akkor és csak akkor szerepel C-ben, ha van olyan A_i és B_j, hogy a metszetük éppen C_k-val egyenlő. Ekkor C_k-hoz az (a_i, b_j) attribútumpárt kell rendelni.

Algoritmus: Az input és output fedvényeknél egyaránt a tartománytérkép pont-vonal-poligon adatstruktúráját tételezzük fel. Tehát adott az A fedvényt leíró A_{node}, A_{line}, A_{poly} és a B fedvényt leíró B_{node}, B_{line}, B_{poly} adatstruktúrák, ezekből kell létrehozni a megfelelő C_{node}, C_{line}, C_{poly} adatstruktúrát.

Ha az A_i és B_j poligonokat az i, j sorszámokkal azonosítjuk, akkor az A_i és B_j metszésével előálló C-beli poligon azonosítója legyen az (i, j) pár. (A node és line elemek azonosítóinak megadására nincs megkötés.) Az algoritmus lépései:

1. $C_{node} := A_{node} \cup B_{node}$. Ez természetesen a C_{node} tömbnek még nem a végleges állapota.
2. $C_{line} := A_{line} \cup B_{line}$. (Itt az lpoly és rpoly értéke még az A és B fedvényekre vonatkozik.)
3. Képezzük valamennyi A-beli vonal metszéspontjait valamennyi B-beli vonallal. (A vonalak koordináta szerinti rendezésével, és befoglaló téglalapok vizsgálatával elérhető, hogy nem kell minden A-beli vonalat minden B-belivel összevetni.) Minden egyes metszésnél az alábbiakat kell elvégezni:
   - A C_{node} tömbbe felvenni a metszéspontot.
   - A C_{line} megfelelő vonal elemeit helyettesíteni kell a megfelelő részvonalakkal.
   - Minden újonnan keletkezett részvonalhoz meghatározzuk az új lpoly és rpoly értékeket, a metszésben részt vett az A,B-beli vonalak régi lpoly és rpoly értékeinek felhasználásával (21. ábra). Tehát az új bal és jobboldali poligon azonosítók már azelőtt meghatározhatók, hogy az új poligon tömböt létrehoznánk!



21. ábra: A bal és jobboldali poligon azonosítók meghatározása.
Bal oldali ábra: PAb és PAj egy A-beli vonallánc lpoly és rpoly értéke, PBb és PBj egy B-beli vonalláncé. Jobb oldali ábra: a vonalmetszés során meghatározott új poligon azonosítók

4. C_{poly} létrehozása: C_{line}-on végighaladunk, és minden egyes vonalnál lpoly és rpoly értékét vizsgáljuk:

- Ha a megfelelő poligonok még nem szerepeltek C_{poly}-ban, akkor felvesszük azokat.
- C_{poly}-ban a megfelelő poligon vonal-listájára felvesszük az adott vonalra való hivatkozást.

A fenti eljárás eredményeként fokozatosan kitöltődik a C_{poly} tömb.

Hangsúlyozzuk, hogy a fentiekben csak az algoritmus alapötleteit vázoltuk, annak tényleges megvalósításához még számos speciális esetet le kell kezelni.

4. Relációs adatbázisok

Jelen tananyagban a relációs adatbázis-kezelés ismeretét feltételezzük, ebben a fejezetben csupán a terminológia egyértelművé tétele és néhány fontos megjegyzés érdekében foglaljuk össze az alapismereteket.

4.1. A relációs adatmodell

A relációs adatmodell lényege, hogy mindent táblázatok, úgynevezett *adattáblák* segítségével ad meg. Az adattábla (vagy egyszerűen csak *tábla*) sorokból és oszlopokból áll. Egy sorát *rekord*nak nevezzük, amely annyi *mező*ből áll, ahány oszlopa van a táblának. Ha egy mező értéke definiálatlan, azt NULL jelöli (ami nem tévesztendő össze a nulla értékkel). A relációs modellre épülő adatbázis-kezelő rendszereket RDBMS-nek (Relational Database Management System) nevezik.

Alábbiakban a relációs modell pontos matematikai definícióit adjuk.

*Attribútum*nak nevezünk egy tulajdonságot, amelyet a megnevezésével azonosítunk, és *értéktartományt* rendelünk hozzá. A Z attribútum értéktartományát dom(Z) jelöli.

Korlátozás: a relációs adatmodellnél az értéktartomány *csak atomi értékekből állhat,* vagyis elemei nem lehetnek struktúrák, halmazok, stb. Ezt a korlátozást első normálformának (1NF) is nevezik, vagyis egy adatbázis akkor van 1NF-ben, ha az attribútumok értéktartománya csak atomi értékekből áll. (Ezt a korlátozást az objektum-relációs modell oldja fel, lásd a 7. fejezetben.)

Az értéktartomány megadása a gyakorlatban *típus* és *hossz* megadását jelenti esetleges korlátozó feltételekkel. Például egy könyvtári nyilvántartásban a könyvet azonosító *könyvszám* attribútum értéktartománya a legfeljebb 8-jegyű decimális számok halmaza lehet, ahol az első jegy a könyv típusára utal és 1...7 között változhat.

*Relációséma*nak nevezünk egy attribútumhalmazt, amelyhez azonosító nevet rendelünk. (Ahol nem értelemzavaró, relációséma helyett egyszerűen csak *sémát* mondunk.)

Jelölések:

- A relációsémát $R(A_1,...,A_n)$ módon szokás jelölni, ahol $A_1,...,A_n$ attribútumok, R pedig a séma neve.
- Használjuk még az R(A) jelölést is, ahol A az $\{A_1,...,A_n\}$ attribútumhalmaz.
- Az R séma A_i attribútumát $R.A_i$-vel jelöljük, ha különböző sémák azonos nevű attribútumait kell megkülönböztetni.

Példa. A könyvek nyilvántartására szolgáló relációséma

Könyv (könyvszám, szerző, cím), ahol az egyes attribútumok értéktartománya:

dom(könyvszám) = 8-jegyű decimális számok halmaza,
dom(szerző) = legfeljebb 30 hosszú karaktersorozatok halmaza,
dom(cím) = legfeljebb 50 hosszú karaktersorozatok halmaza.

Relációnak nevezünk egy $R(A_1,...,A_n)$ séma feletti $T \subseteq dom(A_1)$ X ... X $dom(A_n)$ halmazt. Vagyis, T elemei $(a_1,...,a_n)$ alakúak, ahol $a_i \in dom(A_i)$ (i=1,...,n). A reláció megjelenési formája az *adattábla*, amelynek oszlopai az $A_1,...,A_n$ attribútumoknak, sorai pedig T egyes elemeinek felelnek meg.

Példa. Tekintsük a Könyv (könyvszám, szerző, cím) sémát! Ekkor a dom(könyvszám) X dom(szerző) X dom(cím) halmaz az összes lehetséges *(könyvszám, szerző, cím)* hármast tartalmazza. Ezek közül kiválasztjuk azokat, amelyek a könyvtárban lévő könyveknek felelnek meg, ez lesz a *T* halmaz. Például *T* a következő lehet:

(1121, Sályi, Adatbázisok)
(3655, Radó, Világatlasz)
(2276, Karinthy, Így írtok ti)
(1782, Jókai, Aranyember)

Az adattábla fejlécében a relációsémát szokták megadni, amely azonban nem része a táblának. Példánk esetében:

Könyvszám	*Szerző*	*Cím*
1121	Sályi	Adatbázisok
3655	Radó	Világatlasz
2276	Karinthy	Így írtok ti
1782	Jókai	Aranyember

Mivel a definíció szerint a T reláció egy halmaz, így a relációs modellben *a tábla minden sora különböző*, és a sorokra *semmilyen rendezettséget nem tételezünk fel*. Valójában az adatok gépi tárolása mindig valamilyen sorrendben történik, és a konkrét adatbáziskezelő rendszerek általában megengednek azonos sorokat is. Az elméleti modell és a gyakorlati alkalmazás ezen eltéréseire mindig ügyelni kell.

Több adattábla együttesen alkotja a *relációs adatbázist*, amely egy teljes jelenségkör leírására alkalmas. A könyvtári nyilvántartás egy lehetséges megvalósítását alább láthatjuk. Itt a Könyv táblában adjuk meg az adott könyvet kikölcsönző olvasó számát és a kivétel dátumát. Ha egy könyvet éppen nem kölcsönöztek ki, akkor ezek a mezők definiálatlan (NULL) értékűek, a példában egyszerűen üresen hagytuk ezeket:

A Könyv tábla:

Könyvszám	*Szerző*	*Cím*	*Olvasószám*	*Kivétel*
1121	Sályi	Adatbázisok		
3655	Radó	Világatlasz	122	2005.07.12
2276	Karinthy	Így írtok ti		
1782	Jókai	Aranyember	355	2005.09.23

Az Olvasó tábla:

Olvasószám	*Név*	*Lakcím*
122	Kiss István	Szeged, Virág u. 10.
612	Nagy Ágnes	Szentes, Petőfi út 38.
355	Tóth András	Budapest, Jég u. 3.

A fenti példán jól látható, hogy az *olvasószám* attribútum mindkét táblában szerepel, ezzel kapcsolatot létesít a táblák között. Ez rávilágít a következőre:

A relációs adatmodell lényege, hogy a különböző relációsémák azonos attribútumokat tartalmazhatnak, ezáltal kerülnek kapcsolatba egymással, és így a különálló adattáblák együttese egy szervesen összefüggő adatbázist alkot.

4.2. Kulcsok

Egy *R(A)* relációséma esetén az *A* attribútumhalmaz egy *K* részhalmazát *szuperkulcs*nak nevezzük, ha bármely *R* feletti *T* tábla bármely két sora *K*-n különbözik. Másképpen fogalmazva ez azt jelenti, hogy a szuperkulcsban szereplő attribútumok segítségével egyértelműen azonosíthatók a tábla sorai, de elképzelhető, hogy ehhez kevesebb attribútum is elegendő lenne.

Példa. A Könyv (könyvszám, szerző, cím) sémában a {könyvszám} szuperkulcs, de szuperkulcs például a {könyvszám, szerző} vagy a {könyvszám, cím} attribútumhalmaz is.

Megjegyzendő, hogy a teljes *A* attribútumhalmaz mindig szuperkulcs, hiszen definíció szerint a tábla minden sora különböző.

Az *A* attribútumhalmaz *K* részhalmazát *kulcs*nak nevezzük, ha minimális szuperkulcs, vagyis egyetlen valódi részhalmaza sem szuperkulcs. Ha *K* egyetlen attribútumból áll, akkor *egyszerű*, egyébként *összetett* kulcsról beszélünk.

Megjegyzés: A kulcs nem a tábla tulajdonsága, hanem egy *feltétel előírása* a relációsémára: annak megkövetelése, hogy egy táblában nem lehet két azonos kulcsú sor. A kulcs meghatározása az attribútumok *jelentésének* vizsgálatával lehetséges, és nem egy adott tábla vizsgálatával. Például a fent látható Könyv tábla esetén a *cím* vagy *szerző* attribútumok is minden sorban különbözők, de tudjuk, hogy ez nem garantálható a Könyv tábla mindenkori állapotára.

Ha egy relációsémának több kulcsa is van, egyet kiválasztunk közülük, ez lesz az *elsődleges kulcs (angolul primary key)*. Az elsődleges kulcsot alkotó attribútumokat *aláhúzással* szokás jelölni.

Példa. Az alábbi tábla gépkocsik mozgásának menetlevélszerű nyilvántartását tartalmazza:

Fuvar (gkvez, rendszám, indul, érkezik)

Itt négy összetett kulcs van: {gkvez, indul}, {gkvez, érkezik}, {rendszám, indul}, {rendszám, érkezik}. Ezek közül önkényesen kiválasztunk egyet, ez lesz az elsődleges kulcs:

Fuvar (gkvez, <u>rendszám</u>, <u>indul</u>, érkezik)

Egy relációséma attribútumainak valamely L részhalmaza *külső kulcs (*másnéven *idegen kulcs,* angolul *foreign key)*, ha egy másik séma elsődleges kulcsára hivatkozik. Ez pontosabban azt jelenti, hogy a külső kulcs értéke vagy NULL, vagy a hivatkozott tábla valamely elsődleges kulcs értékével kell, hogy megegyezzen. A külső kulcsot dőlt betűvel, vagy a hivatkozott kulcsra mutató nyíllal jelöljük. A kulcshoz hasonlóan a külső kulcs is *feltétel előírása* a sémákra, és nem az aktuális táblák tulajdonsága.

Ha egy adatbázis valamennyi táblájának sémáját felírjuk a kulcsok és külső kulcsok jelölésével együtt, akkor *relációs adatbázissémát* kapunk.

Példa. A könyvtári nyilvántartás relációs adatbázissémája:

Könyv (könyvszám, szerző, cím, *olvasószám*, kivétel)
Olvasó (olvasószám, név, lakcím)

4.3. Az SQL nyelv alapjai

Az SQL (Structured Query Language, strukturált lekérdező nyelv) a relációs adatbázis-kezelés szabványos nyelve [Gruber, 2003]. Nem algoritmikus nyelv, de algoritmikus nyelvekbe beépíthető (beágyazott SQL). Itt csak a nyelv legfontosabb utasításait tárgyaljuk. A mintautasítások megadásánál a szögletes zárójel elhagyható részt jelent.

4.3.1. Relációsémák definiálása

Relációséma létrehozására a CREATE TABLE utasítás szolgál, amely egyben egy üres táblát is létrehoz a sémához. Az attribútumok definiálása mellett a kulcsok és külső kulcsok megadására is lehetőséget nyújt:

```
CREATE TABLE táblanév
  ( oszlopnév  adattípus  [feltétel],
       ...      ...,
    oszlopnév  adattípus  [feltétel]
    [, táblaFeltételek]
  );
```

Az adattípusok rendszerenként eltérők, néhány jellemző típus:

CHAR(n)	n hosszúságú karaktersorozat
VARCHAR(n)	legfeljebb n hosszúságú karaktersorozat
INTEGER	egész szám (röviden INT)
REAL	valós (lebegőpontos) szám, másnéven FLOAT
DATE	dátum (év, hó, nap)
TIME	idő (óra, perc, másodperc)
CLOB	nagy méretű karakteres adat
BLOB	nagy méretű bináris adat

Feltételek (egy adott oszlopra vonatkoznak):

PRIMARY KEY: elsődleges kulcs
UNIQUE: kulcs
REFERENCES tábla(oszlop): külső kulcs

Táblafeltételek (az egész táblára vonatkoznak):

PRIMARY KEY (oszloplista): elsődleges kulcs
UNIQUE (oszloplista): kulcs
FOREIGN KEY (oszloplista) REFERENCES tábla(oszloplista): külső kulcs

Ha a kulcs vagy külső kulcs több oszlopból áll, akkor csak táblafeltétel formájában adható meg.

Példa. A könyvtári adatbázis definiálása:

```
CREATE TABLE Olvaso
  ( olvasoszam  INTEGER  PRIMARY KEY,
    nev         CHAR(30),
    lakcim      VARCHAR(50)
  );
CREATE TABLE Konyv
  ( konyvszam   INTEGER  PRIMARY KEY,
    szerzo      CHAR(30),
    cim         VARCHAR(80),
    olvasoszam  INTEGER  REFERENCES Olvaso(olvasoszam),
    kivetel     DATE
  );
```

4.3.2. Adattáblák aktualizálása

A táblába *új sor felvétele* az

INSERT INTO táblanév [(oszloplista)] VALUES (értéklista);

utasítással történik. Ha *oszloplista* nem szerepel, akkor valamennyi oszlop értéket kap a CREATE TABLE-ben megadott sorrendben. Egyébként csak az oszloplistában megadott mezők kapnak értéket, a többi mező értéke NULL lesz.

Példa:

```
INSERT INTO Olvaso (nev, olvasoszam) VALUES ("Tóth Aladár", 1111);
```

Sor(ok) módosítása az

UPDATE táblanév SET oszlop = kifejezés, ..., oszlop = kifejezés [WHERE feltétel];

utasítással történik. Az értékadás minden olyan soron végrehajtódik, amely eleget tesz a WHERE feltételnek. Ha WHERE feltétel nem szerepel, akkor az értékadás az összes sorra megtörténik.

Példa:

```
UPDATE Olvaso SET lakcim = „Szeged, Rózsa u. 5."
    WHERE nev = „Kovács József";
```

Sor(ok) törlése a

DELETE FROM táblanév [WHERE feltétel];

utasítással lehetséges. Hatására azok a sorok törlődnek, amelyek eleget tesznek a WHERE feltételnek. Ha a WHERE feltételt elhagyjuk, akkor az összes sor törlődik (de a séma megmarad).

Példa:

```
DELETE FROM Olvaso WHERE nev = „Kovács József";
```

Ha a táblában több Kovács József nevű olvasó volt, akkor mindegyik törlődik.

4.3.3. Lekérdezések

Lekérdezésre a SELECT utasítás szolgál, amely egy vagy több adattáblából egy *eredménytáblát* állít elő. Az eredménytábla a képernyőn listázásra kerül, vagy más módon használható fel. A SELECT utasítás általános alakja a következő:

SELECT [DISTINCT] oszloplista	oszlopok kiválasztása (projekció)
FROM táblanévlista	táblák Descartes-szorzata
[WHERE feltétel]	sorok kiválasztása (szelekció)
[GROUP BY oszloplista	csoportonként összevonás
[HAVING feltétel]]	csoport-szelekció
[ORDER BY oszloplista];	rendezés

Ahol „oszloplista” szerepel, ott általában oszlopkifejezések listáját lehet megadni.

Ha FROM után két táblanevet adunk meg, akkor az egyik tábla minden sorát a másik tábla minden sorával párosítja a rendszer (Descartes szorzat); hasonlóan több tábla esetén.

GROUP BY megadásakor egy csoportba kerülnek azok a sorok, amelyek „oszloplista” értékében megegyeznek, majd minden ilyen csoport egyetlen sorba kerül összevonásra. Az összevonásnál összesítő függvények használhatók (AVG: átlag, SUM: összeg, MIN: minimális érték, MAX: maximális érték, COUNT: elemek száma). Ha GROUP BY nem szerepel, akkor az összesítő függvények a teljes táblára vonatkoznak.

Az egyes alparancsok megadási sorrendje az angol nyelv szabályait követi (lásd fent a mintautasítást), végrehajtási sorrendjük viszont az alábbi:

1. FROM	Descartes-szorzat
2. WHERE	szelekció
3. GROUP BY	csoportonként összevonás
4. HAVING	csoport-szelekció
5. SELECT	projekció
6. ORDER BY	rendezés

A végrehajtási sorrend határozza meg, hogy melyik alparancsban mire lehet hivatkozni. Például GROUP BY után végrehajtott alparancsokban csak összesítő függvény és összesített oszlop adható meg.

Példa. Ábécé sorrendben azon olvasók listája, akik 10-nél több könyvet kölcsönöznek:

```
SELECT nev, Olvaso.olvasoszam
FROM Konyv, Olvaso
WHERE Konyv.olvasoszam=Olvaso.olvasoszam
GROUP BY Olvaso.olvasoszam, nev
HAVING COUNT(*)>10
ORDER BY nev;
```

Az oszlop- és táblanevekhez az AS szócskával alias nevek (másodnevek) adhatók meg, de az AS el is hagyható. A fenti példa alias nevekkel:

```
SELECT nev AS olvasónév, O.olvasoszam AS olvasószám
FROM Konyv AS K, Olvaso AS O
WHERE K.olvasoszam=O.olvasoszam
GROUP BY O.olvasoszam, nev
HAVING COUNT(*)>10
ORDER BY nev;
```

4.3.4. Nézettáblák

A nézettábla *nem tárol adatokat*. Tulajdonképpen egy transzformációs formula, amelyet úgy képzelhetünk el, mint ha ennek segítségével a tárolt táblák adatait látnánk egy speciális szűrőn, „optikán” keresztül. Nézettábla létrehozása:

CREATE VIEW táblanév [(oszloplista)] AS alkérdés;

A SELECT utasítás eredménytáblája alkotja a nézettáblát. „Oszloplista” megadásával a nézettábla oszlopainak új nevet adhatunk. A CREATE VIEW végrehajtásakor a rendszer csak letárolja a nézettábla definícióját, és majd csak a rá való hivatkozáskor generálja a szükséges adatokat. A nézettáblák általában ugyanúgy használhatók, mint a tárolt adattáblák, vagyis ahol egy SQL parancsban táblanév adható meg, ott rendszerint nézettábla neve is szerepelhet.

Példa. Származtatott adatok kezelése. A Raktár (cikkszám, név, egységár, mennyiség) táblából létrehozott nézettábla:

```
CREATE VIEW Készlet (áru, érték) AS
    SELECT név, egységár*mennyiség FROM Raktár;
```

Példa. Adatok elrejtése. A Dolgozó (adószám, név, lakcím, osztálykód, fizetés) táblához létrehozzuk a következő nézettáblát:

```
CREATE VIEW Dolg2 AS
    SELECT adószám, név, lakcím FROM Dolgozó
         WHERE osztálykód='2';
```

Ha a nézettábla tartalmát *módosítjuk*, akkor a módosítás a megfelelő tárolt táblákon hajtódik végre – és természetesen megjelenik a nézettáblában is. Alapelv, hogy egy SQL rendszer *csak akkor engedi meg a nézettábla módosítását, ha a módosítást egyértelműen végre tudja hajtani a tárolt táblákon*. Nem lehet módosítani például a fenti *Készlet* tábla *érték* mezőjét, de a *Dolg2* tábla *lakcím* mezője már gond nélkül módosítható.

5. Szétválasztott modell: térbeli és leíró adatok laza kapcsolata

A térinformatikai rendszerek lényege, hogy a grafikus (térképi) és nem grafikus (leíró) adatokat együtt, integráltan tudják kezelni. Ennek eredményeként például az alábbi lehetőségekhez jutunk:

(i) Rajz → adatbázis lekérdezés: a rajzon grafikus eszközökkel kijelölünk egy rajzelemet vagy rajzelemek egy csoportját, eredményül a kapcsolt leíró adatok listáját kapjuk. (Például adott poligonba eső telkek tulajdonosai.)

(ii) Adatbázis → rajz lekérdezés: SQL-lekérdezéssel kijelölünk egy rekordcsoportot, eredményül a rajzon a kapcsolt rajzelemek eltérő színnel jelennek meg. (Például a 80 évnél idősebb tulajdonosok telkei.)

(iii) Rajz feliratozása az adatbázisból. (Például a térképen megjelenő helyrajzi számok az adatbázisból kerülnek frissítésre.)

A reprezentálandó adatokat tehát két csoportra bontjuk (szétválasztott modell):

- *grafikus adatok:* térképen ábrázolandók,
- *leíró adatok:* táblázatokban tárolhatók.

A szoftverek fejlődése során külön rendszerek jöttek létre a rajzi és táblázatos adatok kezelésére, így általában két független szoftver között kell kapcsolatot teremteni:

- *Grafikus rendszer (Graphics System = GS):* a vektoros rajzot kezeli, például CAD rendszer.
- *Adatbázis-kezelő rendszer (Database Management System = DBMS):* a leíró (táblázatos) adatokat kezeli.

A kapcsolatteremtésről rendszerint a GS gondoskodik, a DBMS a GS-től függetlenül működik.

A grafikus adatokat rétegekbe és rétegcsoportokba szokták rendezni. Példa:

Rétegcsoport Réteg
– alaptérkép: – épületek
  – telkek
  ...
– vízmű: – nyomócső hálózat
  – szennyvíz hálózat
  ...
– elektromos: – magasfeszültségű hálózat
  – transzformátor állomások
  ...

Gyakran többszintű hierarchiát alkalmaznak, például réteg, altéma, téma, teljes térkép. A szétválasztott modellnél *fedvény*nek nevezünk egy réteget vagy rétegcsoportot a hozzá kapcsolódó adattáblával (adattáblákkal) együtt.

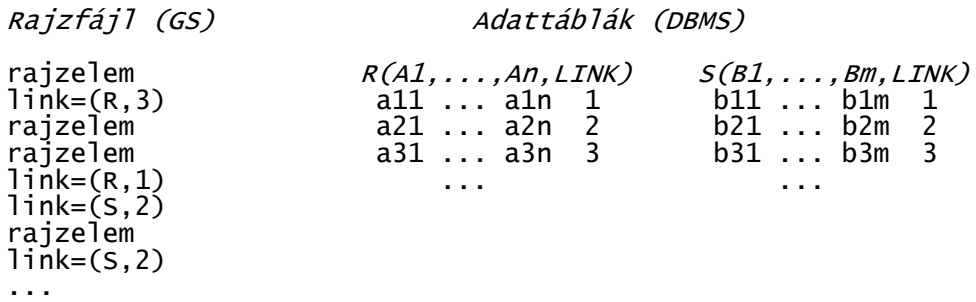
A grafikus és leíró adatok közti kapcsolat rajzelem-adatrekord összekapcsolással valósul meg (általában egy rajzelemhez egy adatrekord kapcsolódik). *Erős kapcsolás*ról beszélünk, ha a kapcsolt rajzelem törlése a megfelelő adatrekord törlését is maga után vonja, *gyenge kapcsolás* esetén ez nem történik meg.

Az alábbiakban két jellemző kapcsolási módot mutatunk be.

5.1. Vektoros rajzelemek bővítése adatbázis linkekkel

Ezt a módszert például a *MicroStation* rendszer alkalmazza [L-Tér, 2001]. A kapcsolás elve:

- A DBMS oldaláról: minden kapcsolandó táblát bővíteni kell egy LINK nevű mezővel, amely kezdetben egy folyamatos rekord sorszámot tartalmazhat (22. ábra). A LINK mezőt a GS írja és olvassa.
- A GS oldaláról: minden rajzelemhez tetszőleges számú adatbázis link csatolható, egy link a kapcsolt tábla azonosítóját és a rekord LINK értékét tartalmazza, egy további bit pedig megmondja, hogy erős vagy gyenge kapcsolásról van szó.



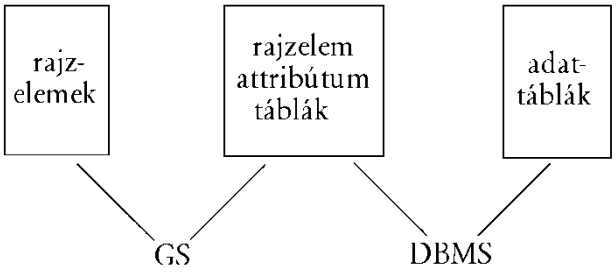
22. ábra: Adatbázis kapcsolat linkeken keresztül.

Ilyen módon sok-a-sokhoz kapcsolat valósul meg, hiszen egy rajzelemhez több rekord, és egy rekordhoz több rajzelem kapcsolható (22. ábra).

5.2. Összekapcsolás rajzelem-azonosítók segítségével

Ezt a módszert az *ArcInfo* rendszer alkalmazza [Zeiler, 1999]. A kapcsolás elve:

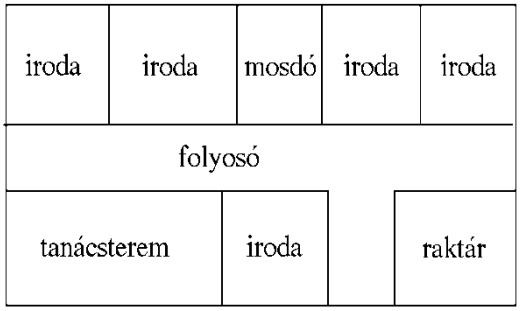
- A GS oldaláról: minden rajzelemnek egyedi azonosítója van. A GS minden rajzelem típushoz egy speciális adattáblát, úgynevezett *rajzelem-attribútumtáblát* generál, amelynek minden egyes rekordja egy rajzelem leírását tartalmazza.
- A DBMS oldaláról: az adatbázis táblái a rajzelem-attribútumtáblákhoz a szokásos relációs módon kapcsolódnak (külső kulcsokkal).



23. ábra: Adatbázis kapcsolat rajzelem-attribútumtáblákon keresztül

Ennél a kapcsolatnál tehát a GS kezeli a rajzfájlt és a rajzelem-attribútumtáblákat, a DBMS pedig a rajzelem-attribútumtáblákat és a további táblákat (23. ábra).

Példa: Tegyük fel, hogy egy vállalati leltárt nyilvántartó adatbázist ki akarunk egészíteni a vállalat irodaépületének tervrajzával (24. ábra), és meg akarjuk jelölni, hogy melyik tárgy melyik helyiségben található.



24. ábra: Irodaépület alaprajza

Az egyes termeket határoló, poligon típusú rajzelemekhez a GS egy Polygon (id, teremid, ter, ker) rajzelem-attribútumtáblát hoz létre, ahol:

- *id:* poligon belső azonosítója (elsődleges kulcs),
- *teremid:* a poligon felhasználói azonosítója (ez is kulcs),
- *ter:* poligon területe,
- *ker:* poligon kerülete.

Az adatbázisban a berendezési tárgyakat egy Leltár (lszám, megnev, érték, dátum, teremid) táblában tartjuk nyilván, melynek mezői:

- *lszám:* a tárgy leltári száma (elsődleges kulcs),
- *megnev:* a tárgy megnevezése,
- *érték:* a tárgy beszerzési értéke,
- *dátum:* a beszerzés dátuma,
- *teremid:* a terem azonosítója, amelyben található (külső kulcs).

A rajzelemek és a Polygon tábla között az *id*, a Polygon és Leltár táblák között a *teremid* mezők képeznek kapcsolatot.

5.3. Példa szétválasztott modellre

Egyszerűsített ingatlan-nyilvántartást veszünk alapul, amely földrészleteket (telkeket), épületeket és tulajdonosaikat kezeli.

Leíró adatok (relációs modell):

Egyedtípusok:
- Tulajdonos (tulazon, név, lakcím)
- Telek (helyrajziszám, terület)
- Épület (postacím, alapterület, szintszám)

Kapcsolatok (feltesszük, hogy egy teleknek több tulajdonosa lehet, épületnek szintén):
- telek-tulajdonos: Telektul (*helyrajziszám, tulazon*, hányad)
- épület-tulajdonos: Épülettul (*postacím, tulazon*, hányad)

Grafikus adatok:

- *Telek* réteg: földrészlet-poligonok, helyrajzi számok a térképen.
- *Épület* réteg: épületpoligonok, házszámok a térképen.
- *Vízrajz* réteg: tavak, vízfolyások.
- *Domborzat* réteg: szintvonalak.

Rajz-adatbázis kapcsolat:

- a telek poligonok és a telek rekordok között,
- az épület poligonok és az épület rekordok között.

Fedvények:

- TELEK fedvény: a *Telek* réteg és a Telek adattábla együttese.
- ÉPÜLET fedvény: az *Épület* réteg és az Épület adattábla együttese.
- VÍZRAJZ fedvény.
- DOMBORZAT fedvény.

A leírtakból látható, hogy információs rendszerünkben vannak olyan adattáblák, amelyekhez nem kapcsolódik grafikus adat (például Tulajdonos), és vannak olyan rajzi rétegek, amelyekhez nem kapcsolódik adattábla (például vízrajz).

5.4. A szétválasztott modell értékelése

A modell alapvető jellemzője, hogy a térbeli és leíró adatokat elkülönítve kezeli. Ebből adódnak az előnyei és hátrányai is.

Előnyök:

- Nagy múltú, fejlett grafikus (GS) és adatbázis-kezelő (DBMS) szoftverek összekapcsolását teszi lehetővé.
- Gyakran a grafikus és leíró adatok külön keletkeznek (rendszerint a leíró adatok már régen adatbázisban vannak, amikor a digitális térképek elkészülnek), így ezek utó-

lagos összekapcsolása természetes megoldás lehet.

– Bizonyos rajzi információk nem kapcsolódnak adatbázishoz (pl. az ingatlan-nyílvántartási példánál a vízrajz és a domborzat).

Hátrányok:

– A térbeli adatok esetén elveszítjük az adatbázis-funkcionalitást (biztonsági mechanizmusok, tranzakció-kezelés, naplózás, adat-rekonstrukció, stb.)
– A grafikus és leíró adatokat elkülönült kezelése miatt a két adatbázis elszakadhat egymástól.

Ez utóbbi elkerülésére

– erős kapcsolás alkalmazható (ahol indokolt),
– mentéskor a grafikus és leíró adatfájlokat mindig együtt kell menteni,
– a grafikus rendszernek időnként ellenőriznie kell a kapcsolatokat (például az eltérésekről hibalistát adhat).

6. Integrált modell: minden adat relációs adatbázisban

A szétválasztott modell hátrányai eltűnnek, ha a grafikus és leíró adatokat egyaránt relációs adatbázisban tároljuk. Ebben a fejezetben az adatbázis-kezelőtől semmilyen különleges támogatást nem várunk el a térbeli adatok kezelésére – ezt a megközelítést ***tisztán relációs modell***nek nevezzük. A térbeli adatstruktúrák topológikus jellegű hivatkozásait külső kulcsokkal valósíthatjuk meg, a változó hosszúságú listák kezelésére CLOB adattípust vagy segédtáblát alkalmazunk.

Ennél a modellnél *fedvény*en a tematikusan összetartozó térbeli adatokat leíró adattáblák együttesét értjük.

Alábbiakban példákon mutatjuk be a lehetséges megoldásokat.

6.1. Félig geometriai hálózat

A relációsémák:

Node (id, x, y, *edges*)
Edge (id, *node1*, *node2*)

Az „edges” attribútum az adott szögpontból kiinduló élek azonosítóinak felsorolását jelenti, ami változó hosszúságú lista. SQL-ben ezt – jobb híján – a CLOB típussal valósítjuk meg: itt karakteres formátumban felsorolhatók az élazonosítók:

```
CREATE TABLE Node
(id INTEGER PRIMARY KEY, x REAL, y REAL, edges CLOB);

CREATE TABLE Edge
(id INTEGER PRIMARY KEY,
 node1 INTEGER REFERENCES Node(id),
 node2 INTEGER REFERENCES Node(id));
```

Az adatbázis feltöltése (egyszerű példaként egy 3 szögpontból és 2 élből álló gráf):

```
INSERT INTO Node VALUES (1, 123, 456, '11 21');
INSERT INTO Node VALUES (2, 234, 567, '11');
INSERT INTO Node VALUES (3, 345, 678, '21');
INSERT INTO Edge VALUES (11, 1, 2);
INSERT INTO Edge VALUES (21, 1, 3);
```
stb.

A fenti megoldás hátránya, hogy a CLOB-beli hivatkozások SQL eszközökkel nem elérhetők, kezelésük külön programot igényel. Az *edges* attribútum azonban csak a hálózat kezelésének gyorsítását szolgálja. Kérdés, hogy ha elhagyjuk, SQL eszközökkel lekérhetők-e egy adott szögpontból kiinduló élek azonosítói. Ezt vizsgáljuk meg az alábbiakban.

A relációsémák:

Node (id, x, y)
Edge (id, *node1, node2*)

A 99-es számú szögpontból kiinduló élek azonosítói a következőképp kérhetők le:

```
SELECT id FROM Edge WHERE node1=99 OR node2=99;
```

Ez a megoldás egyszerű, de lassú. Gyorsítani indexek segítségével lehet:

```
CREATE INDEX n1 ON Edge(node1);
CREATE INDEX n2 ON Edge(node2);
SELECT id FROM Edge WHERE node1=99;
SELECT id FROM Edge WHERE node2=99;
```

6.2. Tartománytérkép spagetti modellben

Ha eltekintünk a topológiától, akkor a tartománytérkép egy poligonhalmazra egyszerűsödik. Például ingatlan nyilvántartás esetén egy Telek (helyrajziszám, terület, poligon) relációsémát SQL-ben a következőképp definiálhatunk:

```
CREATE TABLE Telek
(helyrajziszám INTEGER PRIMARY KEY,
 terület REAL,
 poligon CLOB);
```

Az adatbázis feltöltésére példa:

```
INSERT INTO Telek VALUES (123, 105.6, '11 21, 12 22, 13 23');
```

A fenti megoldás hátránya, hogy a CLOB-beli hivatkozások SQL eszközökkel nem elérhetők, például SELECT utasítással nem tudjuk lekérni egy poligon szögpont-koordinátáit. Megoldás lehet, hogy a koordinátákat külön táblában tároljuk:

Telek (helyrajziszám, terület, poligonID)
PolKoord (*poligonID*, sorszám, x, y)

```
CREATE TABLE Telek
(helyrajziszám INTEGER UNIQUE,
 terület REAL,
 poligonID INTEGER PRIMARY KEY);

CREATE TABLE PolKoord
(poligonID INTEGER REFERENCES Telek(poligonID),
 sorszám INTEGER,
 x REAL, y REAL,
 PRIMARY KEY(poligonID, sorszán));
```

A 987-es helyrajzi számú telek szögpont koordinátáinak lekérése:

```
SELECT x,y FROM Telek,PolKoord
  WHERE helyrajziszám=987 AND
    Telek.poligonID=PolKoord.poligonID
  ORDER BY sorszám;
```

6.3. Tartománytérkép topológikus megvalósítása

A 3.5.1. alfejezetben bemutatott topológikus tartománytérkép adatstruktúrát vesszük alapul, a relációsémákban a külső kulcs jellegű hivatkozásokat dőlt betű jelzi:

Node (id, x, y)
Line (id, *node1, node2, lpoly, rpoly, breakpoints*)
Polygon (id, *lines*)

Mindez SQL-ben:

```
CREATE TABLE Node
 (id INTEGER PRIMARY KEY, x REAL, y REAL);
CREATE TABLE Polygon
 (id INTEGER PRIMARY KEY, lines CLOB);
CREATE TABLE Line
 (id INTEGER PRIMARY KEY,
  node1 INTEGER REFERENCES Node(id),
  node2 INTEGER REFERENCES Node(id)
  lpoly INTEGER REFERENCES Polygon(id),
  rpoly INTEGER REFERENCES Polygon(id)
  breakpoints CLOB);
```

Itt is a CLOB-beli hivatkozások kezelése jelent problémát, például SQL SELECT utasítással nem tudjuk lekérni egy vonallánc vagy poligon szögpont-koordinátáit.

A gond tulajdonképpen abból adódik, hogy a relációsémák *breakpoints* és *lines* attribútumai változó hosszúságú listákat tartalmaznak (azaz a sémák nincsenek 1. normálformában). Ha normalizáljuk az adatbázist, a következőt kapjuk:

Node (id, x, y)
Line (id, *node1, node2, lpoly, rpoly*)
LineCoord (*id*, sorszám, x, y)
Polygon (id, sorszám, *line*)

Itt egy vonallánchoz a LineCoord táblában annyi sor tartozik, ahány töréspontja van a vonallláncnak. Most már le tudjuk kérdezni egy adott – mondjuk, a 987-es számú – vonallánc szögpont koordinátáit:

```
SELECT x, y FROM Node WHERE id=(SELECT node1 FROM Line WHERE id=987);
SELECT x, y FROM LineCoord WHERE id=987 ORDER BY sorszám;
SELECT x, y FROM Node WHERE id=(SELECT node2 FROM Line WHERE id=987);
```

Poligon szögpontjainak lekérdezése hasonlóan történhet, de még bonyolultabb lenne.

6.4. Félig topológikus megoldások

Most az egyszerűsítés érdekében próbáljunk enyhíteni a topológikus követelményeken! Példaként tekintsünk egy földrészlet-nyilvántartást, ahol tartománytérkép helyett egy

Telek (helyrajziszám, terület, x_1, y_1, ..., x_n, y_n)

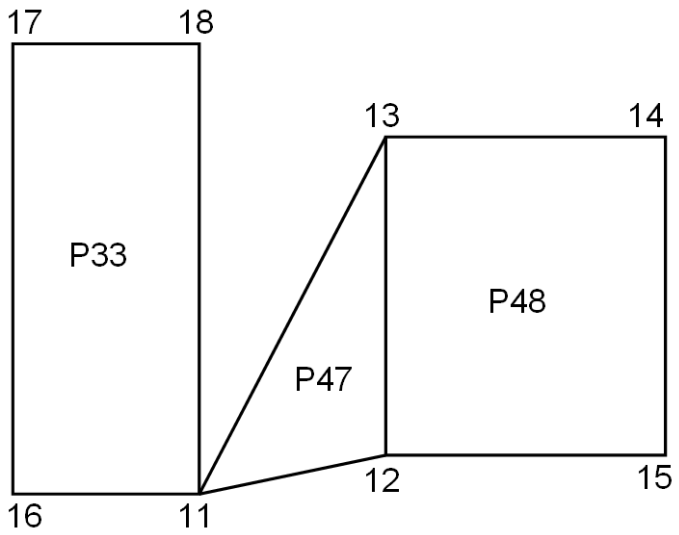
sémából indulunk ki. Itt x_1, y_1, ..., x_n, y_n egy poligon szögpontjainak listáját jelenti. Egy lehetséges relációs megvalósítás:

Telek (helyrajziszám, terület, poligonid)

Poligon (*poligonid*, sorszám, *pontid*)

Pont (pontid, x, y)

Nézzünk egy konkrét példát (25. ábra):



25. ábra: Három telek

TELEK adattábla:

Hrsz	*Terület*	*Poligonid*
1121	250	47
3655	400	48
2276	1300	33

PONT adattábla:

Pontid	*X*	*Y*
11	220	110
12	310	115
13	307	250
14	442	250
15	436	105
16	156	110
17	150	244
18	220	238

POLIGON adattábla:

Poligonid	*Sorszám*	*Pontid*
47	1	11
47	2	12
47	3	13
48	1	12
48	2	13
48	3	14
48	4	15
33	1	11
33	2	16
33	3	17
33	4	18

A fenti megoldás topológikus abban a tekintetben, hogy a csomópontokat egyszeresen tárolja, viszont a határvonalakat nem kezeli külön entitásként, így például a 12–13 él kétszer tárolódik.

Most kérdezzük le a 3655 helyrajzi számú telek határvonal-koordinátái:

```
SELECT sorszám, x, y
FROM Telek, Poligon, Pont
  WHERE helyrajziszám = 3655 AND
  Telek.poligonid = Poligon.poligonid AND
```

```
  Poligon.pontid = Pont.pontid
  ORDER BY sorszám;
```

Látjuk, hogy tisztán SQL eszközökkel elérhetővé váltak a koordináták. A térkép kirajzolásához a szükséges adatok kinyerése azonban nehézkes: a Poligon táblán kell végigmenni (poligon-záródásra figyelni kell), minden él kétszer kerül kirajzolásra.

Másik megközelítés: a poligont nem szögpontok sorozatával, hanem élek sorozatával adjuk meg:

Telek (helyrajziszám, terület, poligonid)
Poligon (poligonid, sorszám, vonalid)
Vonal (vonalid, pont1, pont2)
Pont (pontid, x, y)

Térkép kirajzolásához szükséges adatok kinyerése:

```
SELECT P1.x, P1.y, P2.x, P2.y
  FROM Vonal, Pont P1, Pont P2
  WHERE Vonal.pont1 = P1.pontid AND Vonal.pont2 = P2.pontid;
```

Harmadik megközelítés: a szögpont-koordinátákat a vonalaknál tároljuk (ami a szögpontok többszörös tárolását, vagyis újabb redundanciát jelent):

Telek (helyrajziszám, terület, poligonid)
Poligon (poligonid, sorszám, vonalid)
Vonal (vonalid, x1, y1, x2, y2)

Térkép kirajzolásához szükséges adatok kinyerése viszont rendkívül egyszerű:

```
SELECT * FROM Vonal;
```

Láttuk tehát, hogy sokféle megközelítés lehetséges sajátos előnyökkel és hátrányokkal. A konkrét alkalmazással szemben támasztott igények dönthetik el, hogy melyik megoldást válasszuk.

6.5. Összefoglalás

Az integrált modell *előnye*, hogy a grafikus és leíró adatokat egy közös adatbázisban tároljuk, így azok nem szakadhatnak el egymástól, és a teljes adatbázis-funkcionalitás érvényesül valamennyi adatra.

Hátrány viszont, hogy a térbeli adatok kezelése problémássá válik: lekérdezéskor ismernünk kell a térbeli adatok pontos tárolási struktúráját (például a fenti telek nyílvántartásnál a bemutatott megoldások teljesen eltérő adatkezelést igényelnek), továbbá, az adatkezelés hatékonyabbá tételéhez több-kevesebb redundancia bevitelére lehet szükség.

Az integrált modell továbbfejlesztése két irányban lehetséges:

- objektum-relációs adatbázis-kezelő alkalmazása,
- térbeli adattípusok bevezetése.

Ezeket az irányokat vizsgáljuk meg a következő fejezetekben.

7. Integrált modell: objektum-relációs adatbázis

Az objektum-relációs adatbázis-kezelés lényege, hogy a relációs modellt objektum-orientált elemekkel bővítjük. A reláció azonban továbbra is alapfogalom, tehát nem tiszta objektum-orientáltságról van szó (belül valójában minden relációsan működik, erre épül rá egy objektum-orientált réteg).

Az objektum-relációs adatbázis-kezelő rendszereket ORDBMS-nek (Object-Relational DBMS) nevezik, ilyen például az Oracle és a PostgreSQL.

7.1. Az objektum-relációs modell

A főbb bővítések:

- Kollekciótípusok (komplex attribútumok). Szemben a relációs modellel, egy attribútum értéke nem csak atomi lehet, hanem komplex objektum, például halmaz (azonos típusú elemek együttese), tömb (azonos típusú elemek rendezett sorozata), struktúra (különféle típusú elemek együttese), illetve ezek kombinációja.
- Beágyazott tábla (nested table): a kollekciótípus speciális esete, amikor a relációséma egy attribútuma maga is relációséma, azaz struktúrahalmazt reprezentál.
- Metódusok: műveletek a definiált új objektumtípusokhoz.
- Hivatkozások: kapcsolat megvalósítása külső kulcs helyett.

Kétféle objektumot szoktak megkülönböztetni:

- Sorobjektum: a tábla egy sorát tekintjük objektumnak (ez valójában egy struktúra). A könyvtári példát alapul véve, az Olvasó (olvasószám, név, lakcím) táblában minden egyes olvasó egy objektum. Sorobjektumok azonosítására szolgál az objektum-azonosító (OID), amely tartalmilag azonos sorokat is megkülönböztet.
- Oszlopobjektum: a relációséma egy attribútumát tekintjük objektumnak. Például az Olvasó (olvasószám, név, lakcím) séma lakcím attribútumát (irányítószám, helység, utca, házszám) struktúraként kezeljük.

Az alábbi jelöléseket vezetjük be:

- Ha a relációséma egy attribútuma struktúra, azt R(a1, a2, s(b1, b2, b3), a4) módon jelöljük, például Olvasó (olvszám, név, lakcím(irszám, helység, utca, hsz)).
- Ha a relációséma egy attribútuma halmaz vagy tömb, azt nagybetűvel jelöljük: R(a1, a2, A3, a4). Például, ha egy könyvnek több szerzője lehet: Könyv (könyvszám, Szerző, cím).
- Beágyazott tábla: R1(a1, a2, R2(b1, b2, b3), a4). Például, ha egy olvasónak több lakcíme lehet (26. ábra): Olvasó (olvszám, név, Lakcím(irszám, helység, utca, hsz)). A lakcímet itt nagybetűvel jelöljük.

Olvszám	Név	Lakcím
122	Kiss	IrSzám: 8423 | Helység: Győr | Utca: Fő u. | Hsz: 123.
612	Nagy	IrSzám: 6721 | Helység: Szeged | Utca: Virág u. | Hsz: 10. IrSzám: 7581 | Helység: Pécs | Utca: Kő u. | Hsz: 7.
355	Tóth	IrSzám: 8420 | Helység: Győr | Utca: Jég u. | Hsz: 18.

26. ábra: Példa beágyazott táblára

7.2. Az SQL objektum-relációs lehetőségei

Az objektum-relációs lehetőségeket az SQL3 (más néven SQL:1999) szabvány rögzíti. A könnyebb kipróbálhatóság érdekében azonban ebben a fejezetben az Oracle rendszer szintaxisát követjük, amely egyébként alig tér el az SQL3 szabványtól [Loney, 2006].

7.2.1. Új adattípus deklarálása

Ami a legfontosabb az objektum-relációs modellben: a felhasználó új adattípust (objektumtípust) definiálhat (más néven ADT = Abstract Data Type = UDT = User Defined Type). Az utasítás szintaxisa:

CREATE TYPE típusnév AS OBJECT
(attribútumok
metódusok
);

Példaként tekintsük az Olvasó (olvszám, név, lakcím(irszám, helység, utca, hsz)) relációsémát: Ennek SQL megvalósítása:

```
CREATE TYPE CímTípus AS OBJECT
( irszám  NUMBER,
  helység CHAR(20),
  utca    CHAR(20),
  házszám NUMBER,
);
CREATE TABLE Olvasó
( olvasószám NUMBER,
  név CHAR(30),
  lakcím CímTípus
);
```

Itt a címTípust oszlopobjektumként használjuk. Minden objektumtípushoz implicite definiálódik egy konstruktor metódus, amelynek neve megegyezik az objektumtípuséval, pa-

raméterei megegyeznek az attribútumokkal, visszaadott értéke az új objektum. Beszúrás konstruktor metódussal:

```
INSERT INTO Olvasó VALUES
  (123, 'Nagy Péter',
   CímTípus(6720,'Szeged','Kő u.',3)
  );
```

Most kérjük le a pécsi olvasók névsorát (alias név használata kötelező):

```
SELECT Olv.név
FROM Olvasó Olv
WHERE Olv.lakcím.helység = 'Pécs';
```

Egymásba ágyazott típusokra példa:

Olvasó (olvasószám, személy(név, lakcím(irszám, helység, utca, hsz))):

```
CREATE TYPE CímT AS OBJECT
( irszám  NUMBER,
  helység CHAR(20),
  utca    CHAR(20),
  házszám NUMBER
);
CREATE TYPE SzemélyT AS OBJECT
( név CHAR(30), lakcím CímT );

CREATE TABLE Olvasó
( olvasószám NUMBER,
  személy SzemélyT
);
```

Beszúrás konstruktor függvénnyel:

```
INSERT INTO Olvasó VALUES
  (123, SzemélyT('Nagy Péter',
   CímT(6720,'Szeged','Kő u.',3))
  );
```

Lekérdezés:

```
SELECT Olv.olvasószám
FROM Olvasó Olv
WHERE Olv.személy.lakcím.helység = 'Pécs';
```

7.2.2. Metódusok létrehozása

Típus deklarálása metódussal:

CREATE TYPE típusnév AS OBJECT
(attribútumok
MEMBER FUNCTION név(paraméterek) RETURN típus
);

Metódus(ok) létrehozása:

CREATE TYPE BODY típusnév AS metódusok megadása;

Az egyes metódusok megadása:

MEMBER FUNCTION név (paraméterek) RETURN típus IS programblokk;

Tekintsünk egy példát! A lakcím típushoz megadunk egy cím_hun és egy cím_eng metódust, amelyek magyar, illetve angol konvenció szerinti lakcím stringet képeznek:

```
CREATE TYPE CímT AS OBJECT
( irszám  NUMBER,
  helység CHAR(20),
  utca    CHAR(20),
  házszám NUMBER,
   MEMBER FUNCTION cím_hun RETURN CHAR(50),
   MEMBER FUNCTION cím_eng RETURN CHAR(50)
);
CREATE TYPE BODY CímT AS
MEMBER FUNCTION cím_hun RETURN CHAR IS
  BEGIN
    RETURN( TO_CHAR(irszám) || helység ||
  ', '|| utca || TO_CHAR(házszám) );
  END;
MEMBER FUNCTION cím_eng RETURN CHAR IS
  BEGIN
    RETURN( utca || TO_CHAR(házszám) ||', '
      || helység || TO_CHAR(irszám) );
  END;
```

7.2.3. Objektumtáblák

Objektumtáblán sorobjektumok halmazát értjük. Létrehozása:

CREATE TYPE típus AS OBJECT (attribútumok);
CREATE TABLE tábla OF típus;

Az objektumtáblát kétféleképp kezelhetjük: egyoszlopos táblaként, vagy többoszlopos táblaként (mintha relációs tábla lenne). Példa objektumtáblára

```
CREATE TYPE OlvasóT AS OBJECT
( olvasószám NUMBER,
  név CHAR(30),
  lakcím CHAR(50)
);
CREATE TABLE Olvasó OF OlvasóT;
```

Itt az OlvasóT típust sorobjektumként használjuk.

Beszúrás egyoszlopos táblaként, konstruktorral:

```
INSERT INTO Olvasó VALUES
( OlvasóT(112,'Kiss','8423 Győr Fő u.123') );
```

Beszúrás többoszlopos táblaként:

```
INSERT INTO Olvasó VALUES
(112,'Kiss','8423 Győr Fő u.123');
```

Lekérdezés egyoszlopos táblaként:

```
SELECT VALUE(Olv) FROM Olvasó Olv
  WHERE Olv.olvasószám=112;
```

Lekérdezés többoszlopos táblaként:

```
SELECT név, lakcím FROM Olvasó
  WHERE olvasószám=112;
```

7.2.4. Beágyazott táblák (nested tables)

Először egy objektumtípust, majd erre egy táblatípust definiálunk:

CREATE TYPE típus AS OBJECT (attribútumok);
CREATE TYPE táblatípus AS TABLE OF típus;

Példa beágyazott táblára: Olvasó (olvszám, név, Lakcím(irszám, helység, utca, hsz))

```
CREATE TYPE CímT AS OBJECT
( irszám  NUMBER,
  helység CHAR(20),
  utca    CHAR(20),
  házszám NUMBER
);
CREATE TYPE CímekT AS TABLE OF CímT;
CREATE TABLE Olvasó
( olvasószám NUMBER,
  név CHAR(30),
  lakcím CímekT
) NESTED TABLE lakcím STORE AS lakcímTábla;
```

Amint látjuk, az utasítás végén meg kell adni egy táblanevet (esetünkben lakcímTábla), amelyet a rendszer a beágyazott tábla tárolására használ. Beszúrás konstruktor-függvényekkel:

```
INSERT INTO Olvasó VALUES
(612, 'Nagy Éva', CímekT(
  CímT(6721,'Szeged','Virág u.',10),
  CímT(7581,'Pécs','Kő u.',7)
));
```

Adott olvasóhoz új lakcím felvétele a TABLE függvénnyel lehetséges:

```
INSERT INTO
  TABLE( SELECT lakcím FROM Olvasó WHERE olvasószám=612 )
  VALUES( CímT(1111,'Budapest','Hárfa u.',19) );
```

7.2.5. Dinamikus tömbök

Dinamikus tömb absztrakt adattípusként hozható létre a VARRAY (variable length array) kulcsszóval. Mérete tetszőleges, de maximális mérete megadandó.

Példaként vegyük a Könyv (könyvszám, Szerző, cím) relációsémát, ahol egy könyvnek több, legfeljebb 10 szerzője lehet:

```
CREATE TYPE SzerzőT AS VARRAY(10) OF VARCHAR2(20);
CREATE TABLE Könyv
( könyvszám NUMBER,
  szerző SzerzőT,
  cím CHAR(50)
);
```

Beszúrás:

```
INSERT INTO Könyv VALUES
  (1234, SzerzőT('Sályi','Szelezsán'), 'Adatbázisok');
```

Lekérdezéskor a szerzők felsorolva jelennek meg.

```
SELECT szerző, cím FROM Könyv;
```

7.2.6. Altípusok, öröklés

Tekintsük egy oktatási intézmény helyiségeit nyilvántartó adatbázist. Az egyes helyiségeket a tartalmazó épület azonosítójával és az azon belüli ajtószámmal azonosítjuk, további attribútumok a helyiség neve és alapterülete.

A *helyiség* egyed altípusa a *tanterem,* attribútumai az ülőhelyek száma, a tábla és vetítő típusa, de emellett örökli a főtípus attribútumait is. SQL-ben a főtípus deklarációjában NOT FINAL figyelmeztet arra, hogy altípusok lesznek, az altípus deklarációjában pedig UNDER kulcsszóval kell megadni a főtípust:

```
CREATE TYPE helyiség AS OBJECT
( épület CHAR(10),
  ajtószám NUMBER(4),
  név VARCHAR2(20),
  terület NUMBER,
  PRIMARY KEY (épület, ajtószám)
) NOT FINAL;
CREATE TYPE tanterem UNDER helyiség
( férőhely NUMBER(4),
  tábla VARCHAR2(20),
  vetítő VARCHAR2(20)
);
```

A Helyiségek objektumtábla vegyesen tartalmazhat fő- és altípusokat:

```
CREATE TABLE Helyiségek OF helyiség;
```

Beszúrás konstruktor függvényekkel:

```
INSERT INTO Helyiségek VALUES (helyiség('Irinyi', 123, 'Raktár', 25));
INSERT INTO Helyiségek VALUES (tanterem('Bolyai', 205, 'Riesz', 51,
   92, 'normál tábla', 'vetítővászon'));
```

7.3. Térbeli adatok kezelése objektum-relációs adatbázisban

A következőkben példákat adunk arra, hogyan lehet felhasználni az objektum-relációs lehetőségeket térbeli adatstruktúrák megadására.

7.3.1. Félig geometriai hálózat

A relációsémák:

Node (id, x, y, *edges*)
Edge (id, node1, node2)

Az „edges” attribútum az adott szögpontból kiinduló élek azonosítóinak felsorolását jelenti, amit dinamikus tömbbel valósítunk meg:

```
CREATE TYPE EdgesTípus AS VARRAY(100) OF INTEGER;
CREATE TABLE Node
(id INTEGER PRIMARY KEY, x REAL, y REAL, edges EdgesTípus);
CREATE TABLE Edge
(id INTEGER PRIMARY KEY,
 node1 INTEGER REFERENCES Node(id),
 node2 INTEGER REFERENCES Node(id));
```

A dinamikus tömb könnyebben kezelhető, mint a CLOB (amit a tisztán relációs modellnél alkalmaztunk), továbbá metódusok készítésével is támogatható a hálózat kezelése.

7.3.2. Tartománytérkép spagetti modellben

Most is ingatlan nyilvántartást veszünk alapul, a poligonok koordinátáit beágyazott táblában tároljuk: Telek (helyrajziszám, terület, Poligon(num, x, y))

```
CREATE TYPE KoordTipus AS OBJECT (num NUMBER, x REAL, y REAL);
CREATE TYPE KoordTabla AS TABLE OF KoordTipus;
CREATE TABLE Telek
(helyrajziszám INTEGER PRIMARY KEY,
 terület REAL,
 poligon KoordTabla)
 NESTED TABLE poligon STORE AS KoordTab;
```

Az adatbázisba egy rekord feltöltése:

```
INSERT INTO Telek VALUES
(987, 101.6, KoordTabla(
  KoordTipus(1, 11,21),
  KoordTipus(2, 32,42),
  KoordTipus(3, 53,63))
);
```

Egy telek koordinátáinak lekéréséhez a beágyazott táblát a TABLE függvény segítségével érjük el:

```
SELECT x,y
  FROM TABLE( SELECT poligon FROM Telek WHERE helyrajziszám=987 )
  ORDER BY num;
```

7.3.3. Tartománytérkép topológikus megvalósítása

Tekintsük a már ismert relációsémákat:

Node (id, x, y)
Line (id, *node1, node2, lpoly, rpoly, breakpoints*)
Polygon (id, *lines*)

SQL megvalósításban a *breakpoints* leírására beágyazott táblát, a *lines* kezelésére pedig dinamikus tömböt használunk. A beágyazott tábla *num* attribútuma egy sorszám, amely a koordináták helyes sorrendjét biztosítja:

```
CREATE TABLE Node
 (id INTEGER PRIMARY KEY, x REAL, y REAL);
CREATE TYPE LinesTipus AS VARRAY(100) OF INTEGER;
CREATE TABLE Polygon
 (id INTEGER PRIMARY KEY, lines LinesTipus);
CREATE TYPE KoordTipus AS OBJECT (num NUMBER, x REAL, y REAL);
CREATE TYPE KoordTabla AS TABLE OF KoordTipus;
CREATE TABLE Line
 (id INTEGER PRIMARY KEY,
  node1 INTEGER REFERENCES Node(id),
  node2 INTEGER REFERENCES Node(id),
  lpoly INTEGER REFERENCES Polygon(id),
  rpoly INTEGER REFERENCES Polygon(id),
  breakpoints KoordTabla) NESTED TABLE breakpoints STORE AS KoordTab;
```

A 987-es számú vonallánc szögpont koordinátáinak lekérése:

```
SELECT x,y FROM Node WHERE
  id=(SELECT node1 FROM Line WHERE id=987);
SELECT x,y FROM
  TABLE(SELECT breakpoints FROM Line
  WHERE id=987) ORDER BY num;
SELECT x,y FROM Node WHERE
  id=(SELECT node2 FROM Line WHERE id=987);
```

7.4. Összefoglalás

Az objektum-relációs modell lényegesen jobb lehetőségeket biztosít a térbeli adatok kezelésére, azonban a VARRAY-beli és beágyazott táblabeli hivatkozások használata még mindig kissé nehézkes, és a megoldások hatékonysága is kérdéses [Vasas, 2010].

Ezért több adatbázis-kezelő is beépített térbeli adattípusokat és függvényeket biztosít a térképi adatok hatékony kezelésére, a következő fejezet ezt az irányt vizsgálja meg.

8. Integrált modell: térbeli adattípusok

Ma már számos adatbázis-kezelő rendszer biztosít beépített térbeli adattípusokat a megfelelő kezelő függvényekkel és SQL támogatással. Ebben a fejezetben először a témakör szabványosítását célzó OGC modellt tekintjük át, majd mintapéldákon keresztül bemutatjuk annak használatát. Az ettől többé-kevésbé eltérő konkrét megvalósításokat a Függelék tartalmazza (MySQL, PostgreSQL, PostGIS, Oracle Spatial).

8.1. Az OGC modell

Az Open Geospatial Consortium (korábbi nevén OpenGIS Consortium, röviden OGC) számos cég és intézmény együttműködésével létrejött szervezet, amely a térbeli adatkezelés elméleti és gyakorlati kérdéseinek tisztázását és szabványosítását tűzte ki célul. Az elmúlt évek során számos specifikációt dolgoztak ki, amelyek a honlapjukról letölthetők [OGC].

Először az OGC elvi modelljét tekintjük át, amely a térbeli objektumokat – objektumorientált szemlélettel – osztályokba sorolja.

8.1.1. Osztályhierarchia

Minden osztályhoz attribútumok és metódusok (függvények) tartoznak. Az osztályok öröklési hierarchiát alkotnak, vagyis az alárendelt osztályok öröklik a felettes osztály attribútumait és metódusait, amelyeket saját attribútumokkal és metódusokkal egészíthetnek ki. Egy osztály nem példányosítható, ha absztrakt osztály, vagyis konkrét objektumpéldányt nem tartalmaz. Célja csupán az, hogy attribútumait és metódusait örököljék az alosztályok.

Az OGC osztályhierarchia a következő:

Alakzat (Geometry) (nem példányosítható)
- Pont (Point): x, y koordinátákkal adott
- Görbe (Curve) (nem példányosítható)
  - Vonallánc (LineString): x_1, y_1,..., x_n, y_n koordinátákkal adott
    - Vonal (Line): x_1, y_1, x_2, y_2 koordinátákkal adott
    - Egyszerű zárt vonallánc (LinearRing)
- Felület (Surface) (nem példányosítható)
  - Poligon (Polygon): x_1, y_1,..., x_n, y_n koordinátákkal adott
- Kollekció (GeometryCollection)
  - Ponthalmaz (MultiPoint)
  - Görbehalmaz (MultiCurve) (nem példányosítható)
    - Vonallánchalmaz (MultiLineString)
  - Felülethalmaz (MultiSurface) (nem példányosítható)
    - Poligonhalmaz (MultiPolygon)

Az egyes osztályokról további információkat adunk meg az alábbiakban.

Minden térbeli objektumnak definiálva van a belseje (interior), külseje (exterior) és határa (boundary). Minden alakzat topológiailag zárt, vagyis a határát tartalmazza.

Az *Alakzat (Geometry)* ősosztály attribútumai (többek között):

- **SRID** = Spatial Reference Identifier: koordinátarendszer azonosító. (Két azonos koordinátájú objektum távolsága lehet nemnulla, ha más SRID-vel rendelkeznek!)
- **MBR** (Minimum Bounding Rectangle), or Envelope: befoglaló téglalap.
- **Dimenzió.** Lehetséges értékei: „üres alakzat", 0D (hossza és területe egyaránt 0), 1D (hossza > 0, területe = 0), 2D (területe > 0).

Pont (Point): határa az üres halmaz. Példák: kis méretarányú térképen település, várostérképen buszmegálló.

Görbe (Curve): határa a két végpontja. (Zárt görbe határa üres.) A görbe *egyszerű,* ha önmagát nem metszi.

Vonallánc (LineString): Példák: folyó, út, csővezeték.

Egyszerű zárt vonallánc (LinearRing): kezdő és végpontja megegyezik, önmagát nem metszi.

Felület (Surface): összefüggő felület, amely lyukakat tartalmazhat. Határa a külső és belső határ együtt.

Poligon (Polygon): lyukakat tartalmazhat. Határa egyszerű zárt vonalláncokból áll, amelyek nem metszhetik (csak diszkrét pontokban érinthetik) egymást. Egy külső határa és nulla vagy több belső határa van. Példák: megye, erdő.

Kollekció (GeometryCollection): tetszőleges alakzatok együttese, amelyekhez közös koordinátarendszer tartozik.

Ponthalmaz (MultiPoint): 0D alakzatnak tekintendő. Egy ponthalmaz egyszerű, ha minden pontja különböző. Példák: településhalmaz, városban jegyárusító helyek.

Görbehalmaz (MultiCurve): 1D alakzatnak tekintendő. Egyszerű, ha minden eleme egyszerű és nem metszik egymást. Zárt, ha minden eleme zárt.

Vonallánchalmaz (MultiLineString): Példák: folyóhálózat, úthálózat.

Felülethalmaz (MultiSurface): Elemei nem metszhetik egymást, de véges sok pontban érinthetik egymást.

Poligonhalmaz (MultiPolygon): Elemei nem metszhetik egymást. Példák: erdőfoltok, tórendszer.

8.1.2. Adatformátumok

Az OGC specifikálta a térbeli alakzatok megadási módjait SQL-ben: Alapvetően kétféle megadási mód lehetséges:

- WKT formátum (Well-Known Text): szöveges formátum.
- WKB formátum (Well-Known Binary): bináris formátum.

Specifikálták továbbá az EWKT és EWKB (extended WKT és WKB) formátumokat, amelyek 3 és több dimenziós kiterjesztetések.

Számunkra a WKT formátum a legfontosabb, a szintaxist alább egyszerű példákon mutatjuk be.

Pont:
```
POINT(15 20)
```
Vonallánc:
```
LINESTRING(0 0, 10 10, 20 25, 50 60)
```
Poligon:
```
POLYGON(0 0,10 0,10 10,0 10,0 0)
```
Poligon lyukkal. Előbb a külső határt adjuk meg következhet egy vagy több belső határ (lyuk) megadása:
```
POLYGON((0 0,10 0,10 10,0 10,0 0), (5 5,7 5,7 7,5 7, 5 5))
```
Ponthalmaz:
```
MULTIPOINT(0 0, 20 20, 60 60)
```
Vonallánchalmaz:
```
MULTILINESTRING((10 10, 20 20),(15 15, 30 15))
```
Poligonhalmaz:
```
MULTIPOLYGON((2 2, 2 4, 4 4, 2 2),
((0 0,10 0,10 10,0 10,0 0), ((5 5,7 5,7 7,5 7, 5 5)))
```
Kollekció:
```
GEOMETRYCOLLECTION(POINT(10 10), POINT(30 30), LINESTRING(15 15, 20 20))
```

A WKB formátumot az adatbázis-kezelők BLOB értékként kezelik. A részleteket nem tárgyaljuk, csupán egy példát adunk az adatábrázolás jellegének bemutatására. `POINT(1 1)` leírása hexadecimálisan:

```
0101000000000000000000F03F000000000000F03F
```

amelynek felépítése:

- Bájtsorrend (little-endian vagy big-endian), a példában: 01
- WKB típus (Point, LineString, stb.), a példában: 01000000
- X (double): 000000000000F03F
- Y (double): 000000000000F03F

8.1.3. Függvények

Az OGC számos függvényt specifikál, amelyek SQL lekérdezésekben a térbeli adatok kényelmes kezelését teszik lehetővé. Alább bemutatjuk a fontosabbakat.

Konverziós függvények

Jelölje „geometry” a rendszer belső formátumát, amelyben a térbeli adatokat tárolja. Ekkor az alábbi konverziós függvények állnak rendelkezésre:

Konverzió belső formátumról:

WKB = asBinary (geometry)
WKT = asText (geometry)

Konverzió belső formátumra:

geometry = GeomFromWKB (WKB, SRID)
geometry = GeomFromText (WKT, SRID)

Térbeli műveletek

Length (Curve): a görbe hosszát adja vissza. A gyakorlatban rendszerint vonalláncra alkalmazzuk.

Area (Surface): a felület területét adja vissza. A gyakorlatban poligonokra alkalmazzuk.

Distance (Geometry, Geometry): a két alakzat távolságát adja vissza.

Within (Geometry, Geometry): A visszatérési érték 1 (TRUE), ha az első alakzatot tartalmazza a második, 0 (FALSE), ha nem tartalmazza, és -1 (UNKNOWN), ha valamelyik argumentum definiálatlan (NULL).

Intersects (Geometry, Geometry): A visszatérési érték 1 (TRUE), ha a két alakzat metszete nemüres, 0 (FALSE), ha üres, és -1 (UNKNOWN), ha valamelyik argumentum definiálatlan (NULL).

Intersection (Geometry, Geometry): alakzatot ad vissza, amely a két argumentum metszete.

Buffer (Geometry, distance): alakzatot ad vissza, amely az argumentumként megadott alakzat „distance" távolsággal való kiterjesztése (övezetképzés).

8.2. Térbeli lekérdezések

A térbeli adattípusok gyakorlati alkalmazását két mintaadatbázison mutatjuk be. A leírásnál az OGC specifikációt követjük, az egyes konkrét adatbázis-kezelők ettől többé-kevésbé eltérő megoldásokat alkalmaznak (lásd a Függeléket).

8.2.1. Első mintaadatbázis

Tekintsük az alábbi relációsémákat, amelyek egy földrészlet fedvényt, egy talajtérkép fedvényt és egy kút nyilvántartást kódolnak, mindegyiknél „geom" jelöli a térbeli attribútumot. A példa érdekessége, hogy a relációsémák között nincs hagyományos értelemben vett (külső kulcs) kapcsolat, a térbeli attribútumok révén mégis sokféleképp összekapcsolhatók, amint azt látni fogjuk:

Telek (helyrajziszám, terület, geom)
Talaj (talajId, talajnév, érték, geom)
Kút (id, típus, geom)

Alább megadjuk az SQL deklarációkat: Mivel adott minőségű talaj több foltban helyezkedhet el, ezért ott MultiPolygon típust alkalmazunk:

```
CREATE TABLE Telek
( helyrajziszám CHAR(10) PRIMARY KEY,
  terület       INTEGER,
  geom          POLYGON
);
CREATE TABLE Talaj
( talajId  CHAR(5) PRIMARY KEY,
  talajnév VARCHAR(20),
  érték    INTEGER,
  geom     MULTIPOLYGON
);
CREATE TABLE Kút
( id    CHAR(10) PRIMARY KEY,
```

```
  típus VARCHAR(20),
  geom  POINT
);
```

Az adatbázis feltöltésének bemutatásához táblánként egy-egy mintapéldát adunk. A 14-es számú koordinátarendszerrel dolgozunk:

```
INSERT INTO Telek
  VALUES ( '1234', 1400, GEOMFROMTEXT('POLYGON(1 2,1 5,6 5)',14) );
INSERT INTO Talaj
  VALUES ( '112','Homok',18,
  GEOMFROMTEXT('MULTIPOLYGON((1 2,1 10,6 5),(11 2,9 8,3 5))',14) );
INSERT INTO Kút
  VALUES ( 'A42','Artézi',GEOMFROMTEXT('POINT(12 72)',14) );
```

Lekérdezések

Az első lekérdezés azon telkek helyrajzi számát listázza, amelyeknél hibás területérték szerepel az adatbázisban:

```
SELECT helyrajziszám FROM Telek WHERE AREA(geom) <> terület;
```

Az alábbi lekérdezés azon telkek térbeli adatait válogatja le (például megjelenítéshez), amelyek területe kisebb egy adott értéknél:

```
SELECT geom FROM Telek WHERE terület < 200;
```

A következő lekérdezés térbeli összekapcsolást (spatial join) hajt végre: azon telkek helyrajzi számát listázza, amelyeken nyilvántartott kút van:

```
SELECT helyrajziszám FROM Telek, Kút
  WHERE WITHIN (Kút.geom, Telek.geom);
```

Végül egy összetettebb lekérdezés: először egy nézettábla segítségével lényegében poligon-overlay műveletet (3.7.6. alfejezet) hajtunk végre a *Telek* és *Talaj* fedvények között, amelynek eredményeként megkapjuk az egyes metszet tartományok földértékét:

```
CREATE VIEW Metszet(hrsz, tid, földérték) AS
  SELECT helyrajziszám, talajId, érték*AREA(INTERSECTION(Te.geo,Ta.geo))
  FROM Telek AS Te, Talaj AS Ta
  WHERE INTERSECTS(Te.geo,Ta.geo);
```

Ezután már egyszerű összegzéssel számítható az egyes telkek értéke:

```
SELECT hrsz, SUM(földérték) FROM Metszet GROUP BY hrsz;
```

8.2.2. Második mintaadatbázis

Tekintsük a következő adatbázist:

Megye (megyekód, megyenév, geom)
Út (útszám, útnév, úttípus)
Szakasz (szkód, sznév, geom)
ÚtSzakasz (*szkód, útszám*, sorszám)

Az adatbázis értelmezése: egy út több szakaszból áll, és különböző utaknak lehetnek közös szakaszai. Ez azt jelenti, hogy az utak és szakaszok között sok-a-sokhoz kapcsolat van, ezt valósítja meg az ÚtSzakasz tábla. A „sorszám” attribútum azt adja meg, hogy adott szakasz az adott útnak hányadik szakasza. A megfelelő SQL deklarációk:

```
CREATE TABLE Megye
( megyekód INTEGER PRIMARY KEY,
  megyenév VARCHAR(20),
  geom POLYGON
);
CREATE TABLE Út
( útszám INTEGER PRIMARY KEY,
  útnév CHAR(8),
  úttípus CHAR(2)
);
CREATE TABLE Szakasz
( szkód INTEGER PRIMARY KEY,
  sznév VARCHAR(40),
  geom LINESTRING
);
CREATE TABLE ÚtSzakasz
( szkód INTEGER REFERENCES Szakasz(szkód),
  útszám INTEGER REFERENCES Út(útszám),
  sorszám INTEGER,
  PRIMARY KEY(útszám,szkód)
);
```

Lekérdezések

Adott út teljes hossza:

```
SELECT SUM(LENGTH(Szakasz.geom)) FROM Út, Szakasz, ÚtSzakasz
WHERE ÚtSzakasz.útszám = Út.útszám AND
  ÚtSzakasz.szkód = Szakasz.szkód AND útnév = 'E11';
```

Adott megye területét metsző utak listája:

```
SELECT DISTINCT útnév FROM Megye, Út, Szakasz, Útszakasz
WHERE ÚtSzakasz.útszám = Út.útszám AND ÚtSzakasz.szkód = Szakasz.szkód
  AND megyenév = 'Csongrád' AND INTERSECTS(Szakasz.geom,Megye.geom);
```

Hibakeresés: egymást metsző megyék listája:

```
SELECT M1.megyenév, M2.megyenév FROM Megye M1, Megye M2
WHERE INTERSECTS(M1.geom, M2.geom) AND M1.megyekód < M2.megyekód;
```

Adott téglalapot metsző megyék listája:

```
SELECT * FROM Megye WHERE INTERSECTS(geom, POLYGON(1 2, 2 2, 2 1));
```

8.3. Topológia térbeli adattípusokkal

A térbeli adattípusok kétségtelenül az adatkezelés legkényelmesebb módját biztosítják, ugyanakkor lényegében spagetti modellt valósítanak meg. Egy földrészlet-nyilvántartás esetén például nehéz ellenőrizni, hogy a telek poligonok hézag- és átfedésmentesen fedik-e le a területet. Alább megvizsgáljuk, hogyan lehet topológiát kezelni térbeli adattípusok esetén.

Tekintsünk egy tartománytérképet:

Node (id, geom)
Line (id, *node1*, *node2*, *lpoly*, *rpoly*, geom)
Polygon (id, *lines*)

Mindez SQL-ben:

```
CREATE TABLE Node
 (id INTEGER PRIMARY KEY, geom POINT);
CREATE TYPE LinesTipus AS VARRAY(100) OF INTEGER;
CREATE TABLE Polygon
 (id INTEGER PRIMARY KEY, lines LinesTipus);
CREATE TABLE Line
 (id INTEGER PRIMARY KEY,
  node1 INTEGER REFERENCES Node(id),
  node2 INTEGER REFERENCES Node(id),
  lpoly INTEGER REFERENCES Polygon(id),
  rpoly INTEGER REFERENCES Polygon(id),
  geom LINESTRING);
```

Ha összevetjük a fenti leírást a tartománytérkép objektum-relációs megvalósításával (7.3.2. alfejezet), akkor láthatjuk, hogy a térbeli adattípusok egyszerűsítették a megoldást: a csomópontokat POINT, a vonalakat LINESTRING típussal deklaráltuk, egyedül a poligonoknál nem volt célszerű térbeli adattípus alkalmazása.

Tegyük fel, hogy ezt a tartománytérképet földrészletek (telkek) és vízrajz (folyók, tavak, patakok) leírására használjuk, ahol a vízrajz határvonala minden esetben egyben telekhatár is. (Ha lenne egy patak mindkét oldalán elhelyezkedő telek, ezt két résztelekként modellezzük.) Ez a megoldás lehetővé teszi, hogy a telkek és a vízrajz geometriáját egy közös tartománytérképpel írjuk le: Tekintsük az alábbi relációsémákat

Telek (réteg, id, helyrajziszám)
Vízrajz (réteg, id, név)

A „réteg” attribútum azért kell, hogy a telek- és vízrajz-azonosítókat egymástól meg tudjuk különböztetni. (Ha például minden telek az 1-es és minden folyó a 4-es rétegben van, akkor nem fog gondot okozni, ha azonos id értékű telek és folyó van.) Ezek után csak az van hátra, hogy a tartománytérképet a Telek és Vízrajz táblákhoz kapcsoljuk. Ehhez egy

Relation (réteg, id, topoTípus, topoId)

táblát használunk, ahol topoTípus jelentése: 1=node, 2=line, 3=poligon. Tartalma például:

réteg	*id*	*topoTípus*	*topoId*
1	23	3	234
1	24	3	567
2	75	2	121
2	75	2	217
2	75	2	321
3	43	3	789

stb.

Vagyis, a 23-as számú telket a 234-es poligonhoz kapcsolja, a 24-est az 567-eshez, a 75-ös számú patakot a 121, 217 és 321 számú vonalláncokhoz, végül a 43-as számú tavat a 789-es poligonhoz rendeli.

Példaként kérdezzük le a Bükkös patak teljes hosszát:

```
SELECT SUM(LENGTH(Line.geom)) FROM Vízrajz, Relation, Line
  WHERE Vízrajz.réteg=Relation.réteg AND Vízrajz.id=Relation.id AND
  Relation.topoId=Line.id AND Vízrajz.név="Bükkös";
```

Megjegyezzük, hogy a fentiekhez hasonló megoldással biztosít topológiát az Oracle Spatial [Kothuri és tsai, 2007, Appendix C].

8.4. Az integrált modell értékelése

Napjainkban már egyértelmű, hogy az integrált modell jelenti a térképi információs rendszerek fejlesztésének korszerű irányát. Az előnyök meggyőzőek (egységes adatkezelés és adatbázis-funkcionalitás), a hátrányok (lassú adatelérés és megjelenítés) pedig fokozatosan eltűnnek az új fejlesztések nyomán (lásd az Indexelés és Megjelenítés fejezeteket).

Az integrált modellt három változatban tárgyaltuk.

- Tisztán relációs megközelítés (6. fejezet).
- Objektum-relációs megközelítés (7. fejezet).
- Beépített térbeli adattípusok használata (8. fejezet).

Mindhárom esetben több példát adtunk, amelyek bemutatták a modellezés lehetséges változatait. Noha kétségtelenül a beépített térbeli adattípusok használata a legcélszerűbb megoldás, korábbi ismereteink sem voltak hiábavalók, mert

- szükségessé válhat a beépített adattípusok kiegészítése saját adatstruktúrákkal és topológiával;
- a térbeli adattípusok gyakran objektum-relációs környezetben jelennek meg (például Oracle).

Az eddigi tárgyalás során igyekeztünk az általánosság szintjén maradni, a konkrét adatbázis-kezelők térbeli támogatását a Függelék tekinti át.

9. Indexelés

Amikor térképet megjelenítünk, vagy térképen keresünk, sok ezer (vagy millió) adat közül kell a megfelelőket kiválasztani, az adatbázisból gyorsan kiolvasni, és a képernyőn kirajzolni. Mindennek nagyítás, kicsinyítés és gördítés (scrollozás) esetén is hatékonyan kell működnie. A relációs adatbázis-kezelő rendszereket eredetileg nem ilyen üzemmódra tervezték, a térbeli adatok hatékony kezelése tehát speciális támogatást igényel: ez a térbeli indexelés.

Ebben a fejezetben először a hagyományos, „egy dimenziós' indexelést mutatjuk be, utána a térbeli, „két vagy több dimenziós" indexelés változatait tekintjük át.

9.1. Hagyományos adatbázis indexek

Az index nem része a relációs modellnek, hanem kiegészítő adatstruktúra, amelyet adott táblához lehet generálni. Fő céljai:

- *Keresések gyorsítása*. Ha például egy könyvtári nyilvántartásban adott olvasószámnak megfelelő rekordot keressük, ehhez ne kelljen valamennyi rekordot végignézni.
- *Rendezés*. Listázáskor illetve feldolgozáskor gyakran szeretnénk valamilyen szempont szerint rendezve kezelni a rekordokat (például az Olvasó táblát név szerint ábécé rendben), függetlenül a fizikai adattárolás sorrendjétől.

Az indexet a tábla attribútumainak valamely L részhalmazához generáljuk, ezt *indexkulcs*nak nevezzük. Az indexet is táblaként lehet elképzelni, amelynek első oszlopa az indexkulcsot, a második a megfelelő rekord sorszámát (a gyakorlatban inkább a rekord fizikai címét a merevlemezen) tartalmazza (27. ábra). Az indextábla indexkulcs szerint rendezett, segítségével az eredeti tábla sorai is rendezve kezelhetők, ha az indexen végighaladva mindig a megfelelő sorszámú rekordot olvassuk ki az eredeti táblából.

Könyvszám	*Szerző*	*Cím*	*Olvasószám*	*Kivétel*
1121	Sályi	Adatbázisok		
2276	Karinthy	Így írtok ti		
3655	Radó	Világatlasz	122	2010.07.12
1782	Jókai	Aranyember	355	2010.09.23

Szerző	*Sorszám*
Jókai	4
Karinthy	2
Radó	3
Sályi	1

Cím	*Sorszám*
Adatbázisok	1
Aranyember	4
Így irtok ti	2
Világatlasz	3

27. ábra: A Könyv (könyvszám, szerző, cím, olvasószám, kivétel) táblához létrehozott szerző szerinti, ill. cím szerinti index szemléltetése

Az index konkrét megvalósítása DBMS-enként változik. Az indextábla általában úgynevezett *B-fa* (B = balanced = kiegyensúlyozott) struktúrában kerül tárolásra, amely a bináris keresőfa általánosítása. Tulajdonságai:

- egy csomópontnak kettőnél több (akár 50-100) gyermeke lehet,
- minden módosítás után kiegyensúlyozott marad (így az adatelérés a legrosszabb esetben is csak logaritmikus időt igényel).

A B-fát általában mágneslemezen tárolják (kivéve a gyökér csomópontot, amely tartósan a memóriában lehet). Egy csomópont *egy lemezblokkot* foglal el, ezért akár száz gyermekre mutató pointert is tartalmazhat. A keresés ritkán mélyebb 3 szintnél. Mivel a keresés idejében a lemezolvasás a meghatározó, így a gyakorlatban konstans keresési idővel számolhatunk.

Index létrehozása viszonylag lassú, hiszen ekkor végig kell menni a teljes táblán. A folyamatot úgy képzelhetjük el, hogy az i-edik rekordhoz egy (z_i, i) párt generálnak, ahol z_i az L indexkulcs értéke az adott rekordban, i pedig a rekord fizikai sorszáma, és ezt a (z_i, i) párt fűzik fel a B-fára.

Index használata.

- Az elkészült indexben L adott értékéhez (például a 2276 könyvszámhoz) gyorsan előkereshető a megfelelő rekord fizikai címe.
- A tábla rendezett listázásához a B-fát kell bejárni.
- Ha a táblába új rekordot veszünk fel, ez általában a tábla végére kerül, egyidejűleg a (z_i, i) pár beszúrásra kerül az indexbe.
- Ha rekordot törlünk a táblából, a megfelelő indexbejegyzés törlődik, de a táblában a rekord helye üresen marad, így a rekordok fizikai címei nem változnak meg.

Egy táblához *egyszerre több index* is létrehozható, például a könyveket indexelhetjük könyvszám, szerző és cím szerint is (27. ábra). A rekordokat a képernyőn mindig aszerint látjuk rendezve, hogy a lekérdezésnél melyik mező szerinti rendezettséget kérjük. Ilyenkor automatikusan a megfelelő index lép működésbe, miközben a rekordok fizikai sorrendje mindvégig változatlan marad.

Index létrehozása SQL-ben a

CREATE [UNIQUE] INDEX indexnév ON tábla(oszloplista);

utasítással lehetséges, amely a megadott tábla felsorolt oszlopaira, mint indexkulcsra generál indexet. Ha UNIQUE szerepel, akkor a tábla nem tartalmazhat két azonos indexkulcsú rekordot.

Példa:

```
CREATE INDEX szerind ON Könyv(szerző);
```

Adjuk ki most a

```
SELECT * FROM Könyv WHERE szerző='Jókai';
```

lekérdezést! Ha nem lenne index, a DBMS-nek minden rekordra meg kellene vizsgálnia a szerző='Jókai' feltétel teljesülését. A szerző szerinti index segítségével azonban gyorskereséssel megtalálja Jókai könyveit, és a többi rekorddal nem kell foglalkoznia.

9.2. Térbeli indexek

Térképi alkalmazásoknál gyakran van szükség adott térbeli feltételnek eleget tevő (például megjelenítésnél adott téglalapba eső, vagy snappingnél adott környezetbe eső) objektumok kiválasztására. Ilyenkor az összes objektum végigellenőrzése nyilván elfogadhatatlanul lassú lenne. A térbeli indexelés célja, hogy az ellenőrzendő objektumok számát nagyságrendekkel csökkentse. A kiválasztás általában két lépésből áll:

- *Előszűrés*: a szóba jöhető objektumok kiválasztása térbeli index segítségével.
- *Kiválasztás:* a feltételnek eleget tevő objektumok kiválasztása egyenkénti ellenőrzéssel.

Ha például összesen 100 000 térbeli objektum van, és ebből kell 20-at kiválasztani, akkor előszűréssel kiválasztunk – mondjuk – 100-at (ez gyors eljárás), majd ezekből kiválasztunk 20-at egyenkénti ellenőrzéssel (ez lassú, de csak 100 elemet kell vizsgálni, és nem százezret, ami 1000-szeres gyorsítást jelent).

Térbeli keresés hagyományos indexszel is gyorsítható. Tekintsük a Pont (id, x, y) táblát, ahol az (a, b) pont 10 sugarú környezetében keresünk:

```
CREATE INDEX xKoord ON Pont(x);
```

Az előszűrést megvalósító lekérdezés:

```
SELECT * FROM Pont WHERE x BETWEEN a-10 AND a+10;
```

Itt azonban csak egy dimenzió szerint indexelünk, a kiválasztás során egy függőleges sávban minden elemet ellenőrizni kell. Próbálkozzunk két indexszel:

```
CREATE INDEX xKoord ON Pont(x);
CREATE INDEX yKoord ON Pont(y);
SELECT * FROM Pont
  WHERE x BETWEEN a-10 AND a+10 AND y BETWEEN b-10 AND b+10;
```

Ha belegondolunk, belátható, hogy a DBMS vagy az egyik, vagy a másik index szerint gyorsít, de a kettőt egyszerre nem tudja használni. Érdemi gyorsításhoz tehát speciális térbeli indexek kellenek.

A térbeli indexeket két fő csoportra oszthatjuk [Rigaux és tsai, 2002]:

- *Térvezérelt (space-driven) indexek:* a tér felosztása az adatoktól független. Ilyen lesz például a négyzetrács index és a négyesfa, amint látni fogjuk. Ezek nem kiegyensúlyozott indexek, vagyis a keresés ideje helyről helyre változhat.
- *Adatvezérelt (data-driven) indexek:* a tér felosztása az adatoktól függ. Ilyen az R-fa, amely a B-fa többdimenziós általánosításának tekinthető. Az R-fa kiegyensúlyozott, vagyis a keresés ideje minden esetben ugyanannyi.

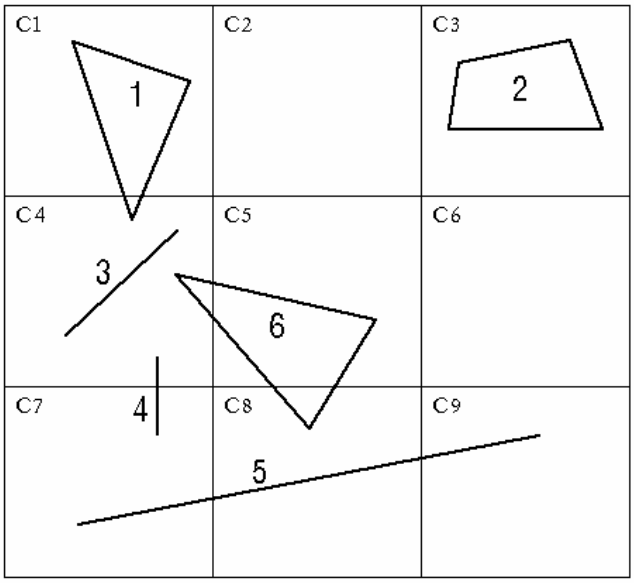
Az indexelési módok tárgyalásánál feltesszük, hogy minden térbeli objektumnak egyedi azonosítója van.

9.2.1. Négyzetrács index (grid index)

A teljes térkép területét n x m négyzetből álló ráccsal fedjük le (a 28. ábrán n = m = 3). Minden négyzethez egy indexlistát rendelünk, amely az adott négyzetbe – részben vagy egészben – beleeső objektumok indexeit tartalmazza (29. ábra). Ha egy objektum több négyzetben is szerepel, akkor szükségképpen több indexlistán fog szerepelni.

Gépi adatstruktúra szintjén egy kétsoros tömböt alkalmazhatunk, ahol az első sor a listaelemeket, a második sor pedig a láncoló „next" pointereket tartalmazza. A (30. ábrán a kilenc lista az 1., ..., 9. oszlopokban kezdődik. A listák végét –1 jelzi.

Ha például térkép megjelenítésekor adott téglalapba eső rajzelemeket keressük a grid index segítségével, akkor először ellenőrizni kell, hogy a téglalap mely grid négyzeteket tartalmazza vagy metszi (előszűrés), és csak az ezeknek megfelelő indexlistákon kell végigmenni (kiválasztás). A térkép kirajzolása annál gyorsabb lesz, minél kisebb kivágatot akarunk megjeleníteni.



28. ábra: Négyzetrács indexelés

C1 C2 C3 C4 C5 C6 C7 C8 C9
R1 R2 R1 R6 R4 R5 R5
R3 R5 R6
R4
R6

29. ábra. Indexlisták a fenti négyzetrács indexhez. Az i-edik objektum azonosítóját Ri jelöli

1	2	3	4	5	6	7	8	9	10	11	12	13	14
R1	-1	R2	R1	R6	-1	R4	R5	R5	R3	R4	R5	R6	R6
-1	-1	-1	10	-1	-1	12	14	-1	11	13	-1	-1	-1

30. ábra: A gépi adatstruktúra

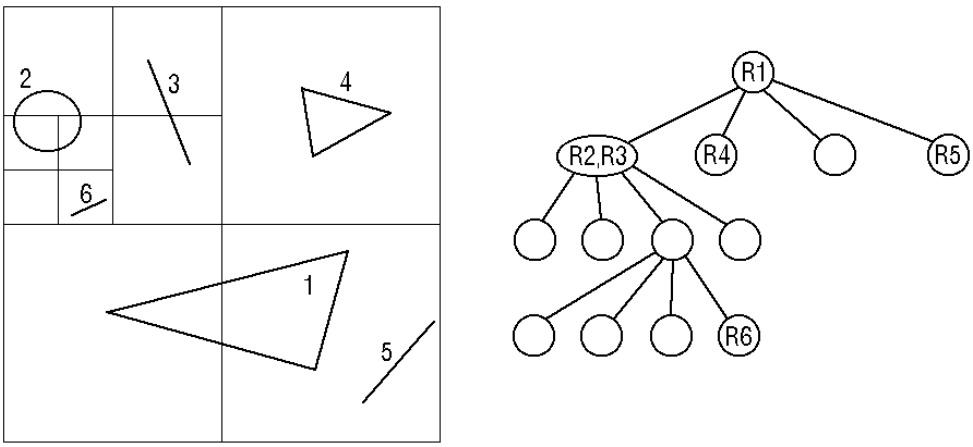
Megjegyzések:

- Amikor azt vizsgáljuk, hogy egy objektum beleesik-e egy grid négyzetbe, elegendő az objektum befoglaló téglalapját vizsgálni. Így ugyan előfordulhat, hogy egy objektum olyan négyzet listájára is felkerül, amelybe valójában nem esik bele (például a 28. ábrán szereplő 6. alakzatot fel kellene venni a C7 négyzet listájára is). Ez azonban nem okoz gondot, mivel az indexelés csak előszűrést végez, az utána következő ellenőrzésnél az ilyen objektumok kiesnek.
- A rajz módosításakor nem szükséges a listákról törölni a törölt/módosított objektumokra való hivatkozásokat, elegendő csak az új/módosított objektumokat felvenni. Az így bennmaradó hibás hivatkozások az előszűrés utáni ellenőrzésnél kiesnek.

9.2.2. Négyesfa index (quadtree index)

Alapelv: a teljes rajzterületet alkotó téglalapot négy egyenlő részre osztjuk, majd az egyes negyedeket tovább negyedeljük, stb. Így egy fastruktúra keletkezik, amelynek gyökere a teljes rajzterületet reprezentálja, szögpontjai pedig a negyedeléssel kapott egyes szegmenseket.

A négyesfa sokféleképpen felépíthető, pl. attól függően, hogy pontszerű vagy területi objektumok tárolására kívánjuk használni [Samet, 1989]. Itt csak egy jellemző megoldást mutatunk be. Minden objektumot a négyesfa egy (és csak egy) szögpontjához rendeljük: ahhoz a szögponthoz, amelyhez tartozó szegmensbe az objektum befoglaló téglalapja teljes egészében belefér, de annak egyik negyedébe sem fér már bele (31. ábra).



31. ábra: Négyesfa index.

Adatstruktúra: a négyesfa Node(n_1, n_2, n_3, n_4, objektumlista) felépítésű elemek sorozata lehet, ahol

- n_1,..., n_4: a leszármazott node-okra mutató pointerek,
- *objektumlista:* az adott szögponthoz tartozó objektumok azonosítói.

Négyesfára objektum felvételekor az új objektumot először a gyökérre helyezzük. Ha valamelyik negyedben elfér, akkor egy szinttel süllyesztjük, stb. Téglalap alakú terület ki-

rajzolásnál gyökérből indulunk, de csak azokra a leszármazottakra megyünk tovább, amelyek metszik a téglalapot.

A négyesfa index itt leírt változata redundanciamentes, vagyis minden objektum csak egyszer szerepel rajta.

9.2.3. R-fa index

Az R-fa (régiófa) a B-fa adaptációja több dimenzióra. Nem számokat, hanem téglalapokat rendez. Kiegyensúlyozott fa. N-dimenzióra is működik, de 2D-re tárgyaljuk.

A fa minden szögpontjának egy lemezblokk felel meg. Egy szögpontnak annyi leszármazottja van, ahány index-bejegyzés elfér egy lemezblokkon (ez sok lehet, akár 100 is). A fa mélysége általában nem több 3-4 szintnél.

Az R-fa felépítése:

Index elem: (téglalap, pnt), ahol pnt egy gyermek szögpontra mutató pointer, a téglalap pedig a gyermek szögponthoz rendelt összes téglalap minimális befoglaló téglalapja (amely tehát tartalmazza az adott szögponthoz tartozó teljes részfát). A közbülső szögpontok index elemek halmazát tartalmazzák.

Adatelem: (téglalap, id), amely az id azonosítójú objektumra hivatkozik a befoglaló téglalapjával. Ha az objektum pont, akkor a téglalap is ponttá zsugorodik. A levél szögpontok adatelemek halmazát tartalmazzák.

Az R-fa rangja r, ha egy lemezblokkon legfeljebb 2r indexbejegyzés fér el. Az indexbejegyzések minimális száma m, ahol $0 < m \leq r$. m értéke a konkrét R-fa-kezelő algoritmustól függ, egy szokásos érték például $m = 0.8r$. A gyökér legalább 2 bejegyzést tartalmaz. Egy objektum csak egy levélen szerepelhet.

Pont szerinti lekérdezés

Adott pontot tartalmazó objektumokat keresésünk. A gyökértől lefelé indulunk. Ha egy gyermek téglalapja tartalmazza a pontot, akkor a megfelelő részfában keresünk tovább. Ha több gyermek is tartalmazza a pontot, akkor mindegyik részfát végig kell nézni. Ezért a keresési idő logaritmikusnál több is lehet, a legrosszabb esetben a teljes fát be kell járni! Ha egy levél téglalapja tartalmazza a pontot, akkor meg kell vizsgálni, hogy maga az objektum is tartalmazza-e. A gyakorlatban általában logaritmikus idővel számolhatunk.

Téglalap szerinti lekérdezések

Adott téglalapba eső objektumok lekérdezése esetén úgy járunk el, mint a pont szerinti lekérdezésnél, de mindig azt vizsgáljuk, hogy a keresési téglalap metszi-e az adott szögpont téglalapját. A levelek szintjén azt vizsgáljuk, hogy a keresési téglalap tartalmazza-e az adott levél téglalapját.

Adott téglalapot metsző objektumok lekérdezése úgy történik, mint az előző esetben, de a levelek szintjén is metszést vizsgálunk. Ha a kereső téglalap metszi egy objektum befoglaló téglalapját, akkor ellenőrzendő, hogy magát az objektumot is metszi-e.

Ponthoz legközelebbi objektum keresése

Kis négyzettel vesszük körül a pontot, és ezt metsző téglalapokat keresünk. Ha egyet sem találunk, a kereső négyzet méretét növeljük mindaddig, amíg nem találunk metsző téglalapot. (Valójában négyzet helyett kört kellene venni, de négyzettel könnyebb számolni.)

Beszúrás

A gyökértől süllyesztjük a (R, id) elemet, ahol R az id-vel azonosított objektum befoglaló téglalapja. Olyan leszármazottat keresünk, amelynek téglalapjába R belefér. Ha több leszármazott téglalapjába is belefér (átfedő téglalapok), akkor a bal szélső szögpontnál folytatjuk a beillesztést. Ha egyik leszármazott téglalapjába sem fér bele R, akkor azt a leszármazottat választjuk, amelynek téglalapja a legkevesebb nagyítást igényli (területi értelemben) ahhoz, hogy lefedje R-t. Levél szinten felvesszük az új objektumot, és ha kell, növeljük a levél téglalapját, és visszamenőleg az ősök téglalapjait is.

Ha a levélen már nincs hely az új objektumnak, akkor kettévágjuk a levelet. A kettévágásnál arra kell törekedni, hogy minimalizáljuk a téglalapok átfedését, mert az többszörös keresést okozhat. Ha a kettévágás miatt a szülő szögpont is betelik, akkor azt is kettévágjuk, és szükség esetén ezt a műveletet továbbvisszük az ősök felé. Ha a gyökér kettévágása is szükséges, akkor a fa szintszáma eggyel nő.

Törlés

Az objektum törlése után a levél és az ősök téglalapjait zsugorítani kellhet. Ha egy levéllap m-nél kevesebb bejegyzést tartalmaz, akkor törlésre kerül, és az objektumokat újra be kell szúrni az R-fába.

R-fa változatok

R*-fa: A kényszerített újra beszúrás elvét alkalmazza: ha egy szögpont betelik, az elemek kb. 30%-át újra beszúrjuk a fába, ezzel csökkentve a szögpont-kettéosztás valószínűségét.

R+-fa: A téglalapok adott szinten nem metszik egymást, ezért a keresési idő mindig logaritmikus. Ennek viszont ára van: az új elem beszúrásra kerül minden olyan részfába, amelynek téglalapját metszi. Ezért az R+-fa jelentősen nagyobb lehet, mint ugyanarra az adathalmazra felépített R-fa. A részleteket nem tárgyaljuk.

10. Megjelenítés

Nehéz elképzelni egy térképi adatbázis alkalmazást képernyőre rajzolt térképek nélkül. Természetesen próbálkozhatunk C vagy Java nyelven írt megjelenítő program készítésével, amely ODBC, illetve JDBC interfészen keresztül éri el az adatbázist. Az igényes és hatékony megjelenítés azonban nem egyszerű feladat.

10.1. A megjelenítés problémái

10.1.1. Hatékonyság

A felhasználó számára akkor megfelelő egy térképi információs rendszer, ha gyorsan tud a térképen navigálni: nagyítani, kicsinyítni és gördíteni a képet minden irányban. Ha a megjelenítendő adatokat mindig a relációs adatbázisból kellene kiolvasni, ez meglehetősen lassú lenne – hiszen a relációs modellt nem térkép megjelenítésre tervezték. Mit lehet tenni? Tekintsük át a lehetőségeket!

Térbeli index alkalmazása. Az előző fejezetben láttuk, hogy térbeli index segítségével gyorsan elérhetők az aktuális kivágatba eső adatok, nem szükséges a teljes adatbázisban keresgélni.

Gyorsítótárak (cache) használata. Mivel a felhasználó gyakran adott területen belül mozog, fölösleges lenne a már egyszer lekért adatokat újra meg újra lekérni. Kétféle gyorsítótár lehetséges:

- *Adat-gyorsítótár:* a korábban lekért vektoros geometriai adatokat tárolja.
- *Kép-gyorsítótár: a vektoros adatokból generált raszteres képernyőképeket tárolja.*

10.1.2. Grafika

Szép, áttekinthető térképet rajzolni nem egyszerű feladat [Zentai, 2000], és intelligens grafikát igényel. Példaként néhány problémás dolog:

- Utak, utcák kirajzolása kis méretaránynál szimpla, nagyobb méretaránynál kettős vonallal.
- Útelágazások, felüljárók, alagutak stb. korrekt kirajzolása.
- Speciális jelkulcsi elemek: például vasúti sín.
- Feliratok: például település neve a település pontja mellé, alá vagy fölé kerüljön, településnevek ne csússzanak össze.
- Utcanevek kiírása, például kanyarodó utcák esetén.
- Útszám címkék elhelyezése.
- Területet kitöltő jelkulcsi elemek, például erdő, mocsár stb.
- Méretarány változtatás kezelése nagyításkor és kicsinyítéskor.

A fenti problémák megoldása jól tanulmányozható például a Google térképrendszerén [GoogleMaps]. Kérdés azonban, hogy saját alkalmazás fejlesztése esetén milyen lehetőségeink vannak. A továbbiakban a MapServer nevű, nyílt forráskódú szoftvert és Oracle MapViewert mutatjuk be.

10.2. UMN MapServer

Az UMN MapServer [Kropla, 2005] [MapServer] egy nyílt forrású fejlesztő környezet, mely képes internetes térinformatikai alkalmazások felépítésére. A rendszert eredetileg a University of Minnesota (UMN) ForNet nevű projektje keretén belül fejlesztették ki. A MapServer számos adatformátumot képes kezelni, néhány fontosabb:

- MapInfo Files
- Oracle Spatial
- Geography Markup Language (GML)
- TIGER Files
- ESRI Binary Coverages (ADF)
- ESRI ArcSDE (SDE)
- MicroStation Design Files (DGN)
- MySQL MYGIS Format

Egy úgynevezett Mapfile definiálja a kapcsolatot az objektumok között, megmutatja a MapServernek, honnan olvashatja ki az adatokat, és megmondja, hogyan kell azokat megjeleníteni. Alapegysége a layer, ami adat (attribútumok és geometria) és stílus kombinációja.

A Mapfile hierarchikus szerkezetben tárolja az egyes objektumokat: a Map objektum a gyökér, és minden egyéb ez alatt helyezkedik el. Az objektumok rendre END kulcsszóval végződnek.

A MapServer telepítésével együtt általában egy Apache webszerver is telepítésre kerül. A MapServer projektet CGI változók és template fájlok segítségével lehet vezérelni [Szrnka, 2008].

10.3. Oracle MapViewer

Az Oracle MapViewer [Kothuri és tsai, 2007] egy szerveroldali komponens, főbb részei a következők:

Térkép-renderelő „engine”: Feladata az adatbázisban tárolt információkból képernyőn megjeleníthető (raszteres) térkép-részletek generálása. Többféle formátumban tud gyártani: GIF, JPEG, SVG, PNG.

Térkép definíciók: Adatbázisban tárolt (XML) adatok. Ezek az adatok írják le, hogy mi legyen a térképen, és az hogyan nézzen ki (vonalak vastagsága, színe, stb.).

Programozói interfészek: Segítségükkel elérhetőek a MapViewer szolgáltatásai. A támogatott interfészek a következők: Java, PL/SQL, JavaScript, és XML.

Oracle MapBuilder: Grafikus segédeszköz, mely a térkép definíciók létrehozását, és kezelését könnyíti meg. Ezen kívül számos hasznos funkciója van, például shapefile-ok importálása, metaadatok importálása és exportálása, stb.

Az Oracle MapViewer letölthető az Oracle honlapjáról, és használható az OTN Developer Licence keretén belül.

10.3.1. Stílusok

Mielőtt kirajzolnánk bármilyen alakzatot a képernyőre, meg kell adni a MapViewernek, hogy hogyan tegye ezt. Erre valók a stílusok. Stílust létrehozhatunk pontszerű objektumhoz, vonalakhoz, illetve területtel rendelkező objektumokhoz, vagy szöveges elemekhez.

Area stílus. Ez a stílus olyan területtel rendelkező objektumokhoz használható, melyek kitöltéséhez valamilyen mintára van szükség. Ilyen például kertek, temetők jelölése a településtérképeken. Meg kell adni a kitöltési mintaként használt képet, és a kitöltés szegélyét.

Color stílus. Hasonló az előzőhöz, de itt a kitöltés nem valamilyen minta, hanem egyszerű szín. Létrehozáskor meg kell adni a kitöltés színét, a szegélyvonal színét és vastagságát.

Line stílus. Vonalláncok megjelenítéséhez alkalmazható. Négyféle jellemző állítható be:

- General (általános jellemzők): vonalvastagság, szín, vonalak végződése (lekerekített, vagy szögletes).
- Center line (középvonal): vastagság, szín, vonalszaggatás megadható, illetve lehetőség van „kerítés” megadására is (a középvonalra merőleges, rövid egyenesszakaszok)
- Wing line (oldalvonalak): vastagság, szín és vonalszaggatás adható meg.
- Marker pattern: vonalon megjelenő ismétlődő szimbólum.

Marker stílus. Szimbólumok megadása pont típusú objektumokhoz. Szimbólumként megadható kép; általunk létrehozott vektoros objektum (kör, csillag, stb.), és lehetőség van True Type betűtípusok szimbólumként való használatára is.

Text stílus. Megadhatók szöveges stíluselemek, melyek utcák, települések feliratozásához használhatók.

Advenced styles. Bonyolultabb stílusfajta, segítségével diagramok tehetők a térképre, illetve egy adott értéktől függően más-más objektumot lehet kirajzolni.

10.3.2. Témák

A témák (vagy rétegek) összetartozó alakzatokat jelenítenek meg. Egy téma különböző típusú objektumokat is megjeleníthet (például vízrajzi réteg esetén poligonokat és vonalláncokat egyszerre), és egy táblához több témát is meg lehet adni (például mindenféle út egy közös táblában van tárolva, ekkor készíthető egy olyan téma, mely a csak a főutakat tartalmazza). A téma megadása 4 lépésben történik:

1. lépés: Meg kell adni a téma nevét, rövid leírását, a táblát, melyre a téma épül, és a tábla azon oszlopát, mely a megjelenítendő geometriai adatokat tartalmazza.

2. lépés: Meg kell adnunk azt a stílust, melyet a megjelenítéshez használ a téma (a kiválasztott stílust és a megjelenítendő alakzatok típusát egyeztetni kell).

3. lépés (opcionális): Felirat megadása. Ki kell választani a tábla azon oszlopát, amelyet a térképen feliratként kívánunk használni, valamint a felirathoz tartozó szöveges stílust is.

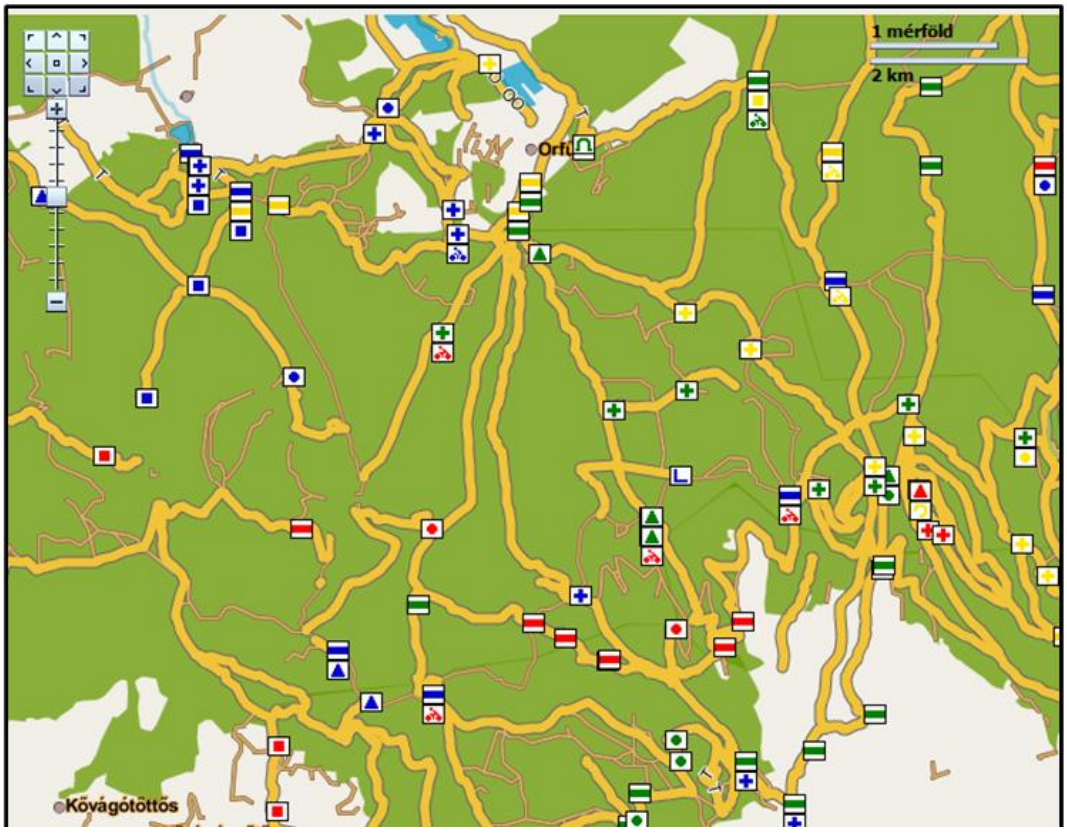
4. lépés (opcionális). Szűrési feltételeket lehet megadni, ezáltal csak a feltételeknek megfelelő sorokat fogja figyelembe venni a téma (például csak főutakat az utak táblából).

10.3.3. Térképek

A térképek a témákból épülnek fel. Meg kell adni a térkép nevét, majd ki kell választani azokat a témákat, amelyeket szeretnénk megjeleníteni a térképen. Fontos a témák sorrendje: a listában lentebb levő témát fogja később kirajzolni a MapViewer; az itt található alakzatok tehát fedni fogják az összes olyan alakzatot, melyek az épp kirajzolt téma feletti témához tartoznak.

Az is beállítható, hogy az egyes témák milyen méretarány tartományban jelenjenek meg. Ehhez „Min. scale” és „Max. scale” értékeket kell megadni. „Scale mode = RATIO” beállítás mellett a már megszokott térképarányok (pl. 1:100 000) használhatók. A téma akkor fog megjelenni, ha a zoom szint meghaladta a Min. scale értéket, és eltűnik, ha meghaladja a Max. scale értéket. A méretarányok megadásakor az 1 értéket nem kell megadni, csak a második értéket (emiatt a „Min. scale”-ben szereplő értéknek mindig nagyobbnak kell lennie, mint a „Max. scale”-ben szereplő érték!).

Illusztrációként egy alkalmazás [Dózsa, 2010] képernyőképét mutatja a 32. ábra.



32. ábra: Oracle Spatial és MapViewer segítségével készült, túristautakat ábrázoló alkalmazás [Dózsa, 2010]

A stílusok, témák és térképek definíciói adattáblákban tárolódnak, maguk a definíciók CLOB oszlopban XML szintaxis szerint kerülnek megadásra. Ezek direkt felhasználói módosítása azonban nem célszerű, helyette a MapBuilder eszköz használata javasolt, amely grafikus felületen teszi lehetővé a stílusok, témák és térképek kényelmes megadását.

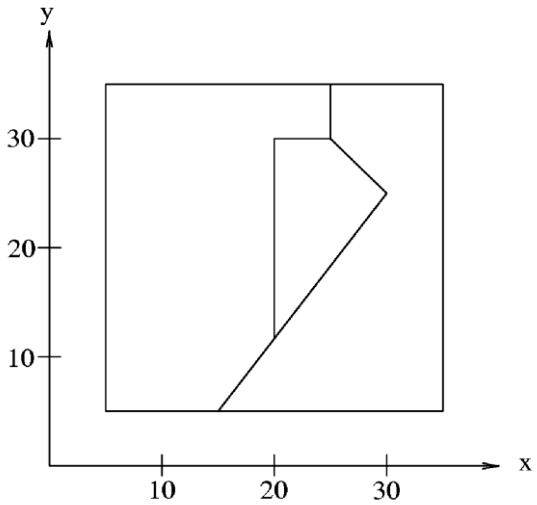
11. Gyakorló kérdések és feladatok

1. Jellemezze a raszteres és vektoros adatábrázolást, valamint az oda-vissza konverziót az előnyök, hátrányok, nehézségek szempontjából!

2. Milyen szekciókból áll a DXF fájl, melyik mit tartalmaz? Adjon példát vonallánc megadására!

3. Határozza meg a kis- és nagyméretarányú térkép fogalmát! Adjon példát mindkettőre!

4. A 33. ábra szerinti tartománytérképhez írja fel a megfelelő Node-Line-Polygon adatstruktúrát! (A rajzterületen kívüli poligont jelölje P_0-lal!)



33. ábra: Tartománytérkép.

5. Milyen algoritmusoknál és hogyan hasznosíthatjuk a vektoros tartománytérkép adatstruktúra lpoly és rpoly attribútumait?

6. Mit nevezünk lineáris címzés módszerének a hálózat adatstruktúránál? Adjon rá példát!

7. Vonalláncok (töröttvonalak) metszésvizsgálata milyen módszerekkel gyorsítható?

8. Definiálja pontosan a Geometry, MultiLineString és LinearRing fogalmakat az OGC modellben!

9. Mit jelent a WKT és WKB az OGC modellben? Adjon példát poligon (lyukkal), ponthalmaz és kollekció WKT-típusú megadására!

10. Jellemezze és hasonlítsa össze a MySQL, a PostreSQL és az Oracle Spatial térbeli adatkezelését, példán bemutatva!

11. Tekintsük az alábbi sémákat:

Utca (utcaId, név)
Utcaszakasz (szakaszId, *utcaId*, sorszám, vonallánc)
Írja fel a fenti sémák definícióját SQL-ben
a) tisztán relációs modellben CLOB nélkül,
b) objektum-relációs modellben VARRAY segítségével,
c) objektum-relációs modellben beágyazott tábla segítségével!
Kérdezze le a Virág utca teljes hosszának vonallánc koordinátáit mindhárom esetben!

12. Tekintsük az alábbi sémákat:
Megye (megyekód, megyenév, poligon)
Baleset (id, dátum, jelleg, x, y),
ahol x, y a közlekedési baleset helyszínének koordinátái.
Írja fel a fenti sémák definícióját SQL-ben
a) tisztán relációs modellben CLOB nélkül,
b) objektum-relációs modellben beágyazott tábla segítségével,
c) térbeli adattípusok segítségével!
A c) esetben kérjen le baleseti statisztikát is (megyenév, balesetszám) felépítésben!

11.1. Feladatok megoldása

1. Számítógépen szerkesztett rajz esetén a vektoros, míg fénykép esetén a raszteres kódolás a természetes. A raszterkép kezelése egyszerűbb (például monitoron közvetlenül megjeleníthető), a vektoros ábrázolás viszont pontosabb. A vektor-raszter konverzió számítógépes grafikai algoritmusokkal korrekten megoldható, a raszter-vektor konverzió viszont alakfelismerést igénylő mesterséges intelligencia feladat, melynek megoldása csak korlátozottan lehetséges.

2. A DXF fájl szekciói: HEADER (változók beállítása), TABLES (vonaltípusok, rétegek megadása), BLOCKS (blokk definíciók), ENTITIES: (rajzelemek leírása). Vonallánc megadására példa:

```
  0
POLYLINE
  0
VERTEX
  10
123.1
  20
456.2
  0
VERTEX
  10
232.0
  20
454.0
  0
SEQEND
```

3. Nagyméretarányú egy térkép, ha méretaránya 1:10 000-nél nagyobb (például 1:2000), egyébként kisméretarányú (például 1:50 000). Példák: kisméretarányú: földrajzi térkép, nagyméretarányú: kataszteri térkép.

4. Az adatstruktúra:

NODE:	*id*	*x*	*y*
	N1	15	5
	N2	20	10
	N3	25	30
	N4	25	35

LINE:	*id*	*node1*	*node2*	*lpoly*	*rpoly*	$x_1,y_1, \ldots, x_n,y_n$
	L1	N1	N2	P1	P3	
	L2	N3	N4	P1	P3	
	L3	N2	N3	P1	P2	20,30
	L4	N2	N3	P2	P3	30,25
	L5	N1	N4	P0	P1	5,5, 5,35
	L6	N1	N4	P3	P0	35,5, 35,35

POLYGON:	*id*	$line_1, \ldots, line_n$
	P1	L1, L5, L2, L3
	P2	L3, L4
	P3	L1, L4, L2, L6

5. Három algoritmusnál hasznosítjuk az lpoly, rpoly attribútumokat:

   a) *Ha sok poligon területét* kell egyidejűleg meghatározni, akkor minden vonalhoz egy előjeles „területértéket” számolunk, és az egyes poligonok területét a határoló vonalak „területének” összegeként kapjuk. Ha az adott poligon egy vonalnak bal poligonja (lpoly), akkor a vonalhoz rendelt terület (–1)-szeresével számolunk, ha jobb poligonja (rpoly), akkor a vonalhoz rendelt területtel.
   b) *Ha sok pontnak sok poligonba esését vizsgáljuk*, akkor a függőleges félegyeneseknek a vonalláncokkal való metszéspontjait határozzuk meg, és a metszéspont alatti poligon azonosítóját a vonal lpoly, rpoly paramétereiből tudjuk meghatározni.
   c) A poligon overlay algoritmusnál minden újonnan keletkezett részvonalhoz meghatározzuk az új lpoly és rpoly értékeket, a metszésben részt vett vonalak régi lpoly és rpoly értékeinek felhasználásával: az új bal és jobboldali poligon azonosítók már azelőtt meghatározhatók, hogy az új poligon tömböt létrehoznánk.

6. A lineáris címzés módszere egy hálózat élei mentén elhelyezkedő objektumok hatékony nyilvántartására szolgál. A lényeg: az éleket nem bontjuk részélekre, hanem egy külön Objektumok (obj_id, megnevezés, edge_id, dist1, dist2) táblában tároljuk az él menti objektumokat. Példa lehet egy autóút-hálózat esetén az út menti parkolók, benzinkutak, vendéglátó egységek, hidak, útjavítások nyilvántartása.

7. Vonalláncok metszésvizsgálata befoglaló téglalapok alkalmazásával gyorsítható: ha két vonallánc befoglaló téglalapjai diszjunktak, akkor biztosan nem metszik egymást. További gyorsítás érhető el térbeli index alkalmazásával: egymástól távol eső vonalláncok esetén még a befoglaló téglalapok vizsgálata sem szükséges. Monoton szakaszokra bontással is gyorsíthatunk: egy monoton növő és egy monoton csökkenő

szakasz legfeljebb egyszer metszheti egymást, és a metszéspontot is könnyebb megtalálni.

8. A definíciók:
Geometry: ősosztály, nem példányosítható. Minden térbeli objektumtípus ebből származtatható. Legfontosabb attribútumai az SRID (koordinátarendszer azonosító). MBB (befoglaló téglalap) és a dimenzió.
MultiLineString: vonalláncok halmaza.
LinearRing: egyszerű zárt vonallánc: kezdő és végpontja megegyezik, önmagát nem metszi.

9. WKT formátum (Well-Known Text): szöveges formátum. WKB formátum (Well-Known Binary): bináris formátum.

```
POLYGON((0 0,10 0,10 10,0 10,0 0), (5 5,7 5,7 7,5 7, 5 5))
MULTIPOINT(0 0, 20 20, 60 60)
GEOMETRYCOLLECTION(POINT(30 30), LINESTRING(15 15, 20 20))
```

10. Adatbázis-kezelők térbeli adatkezelésének összehasonlítása:
MySQL: az OGC specifikációt követi, de maga az adatbázis-kezelő nem objektum-relációs. Példa poligon megadására: `POLYGON(0 0,10 0,10 10,0 10,0 0)`
PostgreSQL: objektum-relációs rendszer, viszont a térbeli adattípusokra az OGC-től erősen eltérő megoldásokat alkalmaz. Példa poligon megadására:

```
((0, 0), (10, 0), (10, 10), (0, 10)).
```

Oracle Spatial: objektum-relációs rendszer, amely egyetlen univerzális térbeli adattípust használ (SDO_GEOMETRY). Példa poligon megadására:

```
SDO_GEOMETRY(2003, NULL, NULL,
SDO_ELEM_INFO_ARRAY(1,1003,1),
SDO_ORDINATE_ARRAY(0,0, 10,0, 10,10, 0,10, 0,0))
```

11. A relációsémák definíciója és a lekérdezés:
a) tisztán relációs modell:

```
CREATE TABLE Utca
( utcaId INTEGER PRIMARY KEY,
  név VARCHAR(30)
);
CREATE TABLE Utcaszakasz
( szakaszId INTEGER PRIMARY KEY,
  utcaId INTEGER REFERENCES Utca(utcaId),
  sorszám INTEGER
);
CREATE TABLE Vonal
( szakaszId INTEGER REFERENCES Utcaszakasz(szakaszId),
  sorsz INTEGER,
  x REAL, y REAL,
  PRIMARY KEY (szakaszId, sorsz)
);
SELECT x, y FROM Utca, Utcaszakasz, Vonal
  WHERE Utca.utcaId=Utcaszakasz.utcaId
    AND Utcaszakasz.szakaszId=Vonal.szakaszId AND név='Virág'
  ORDER BY sorszám, sorsz;
```

b) objektum-relációs modell, VARRAY:

```
CREATE TABLE Utca
( utcaId INTEGER PRIMARY KEY,
  név VARCHAR(30)
);
CREATE TYPE Vonaltip AS VARRAY(100) OF REAL;
CREATE TABLE Utcaszakasz
( szakaszId INTEGER PRIMARY KEY,
  utcaId INTEGER REFERENCES Utca(utcaId),
  sorszám INTEGER,
  vonallánc Vonaltip
);
SELECT vonallánc FROM Utca, Utcaszakasz
  WHERE Utca.utcaId=Utcaszakasz.utcaId AND név='Virág'
  ORDER BY sorszám;
```

c) objektum-relációs modell, beágyazott táblával:

```
CREATE TABLE Utca
( utcaId INTEGER PRIMARY KEY,
  név VARCHAR(30)
);
CREATE TYPE KoordTipus AS OBJECT (num NUMBER, x REAL, y REAL);
CREATE TYPE KoordTabla AS TABLE OF KoordTipus;
CREATE TABLE Utcaszakasz
( szakaszId INTEGER PRIMARY KEY,
  utcaId INTEGER REFERENCES Utca(utcaId),
  sorszám INTEGER,
  vonallánc KoordTabla
) NESTED TABLE vonallánc STORE AS KoordTab;

SELECT vonallánc FROM Utca, Utcaszakasz
  WHERE Utca.utcaId=Utcaszakasz.utcaId AND név='Virág'
  ORDER BY sorszám;
```

A fenti lekérdezés tárolási sorrendben listázza a beágyazott tábla sorait. A 'num' szerinti rendezettség garantálása csak beágyazott SQL-lel oldható meg.

12. A relációsémák definíciója:
    a) tisztán relációs modellben:

```
CREATE TABLE Megye
( megyekód INTEGER PRIMARY KEY,
  megyenév VARCHAR(20),
  poligonID INTEGER UNIQUE
);
CREATE TABLE PolKoord
( poligonID INTEGER REFERENCES Telek(poligonID),
  sorszám INTEGER,
  x REAL, y REAL,
  PRIMARY KEY(poligonID, sorszán)
);
CREATE TABLE Baleset
( id INTEGER PRIMARY KEY,
  dátum DATE,
  jelleg CHAR(30),
  x REAL, y REAL
);
```

b) objektum-relációs modellben beágyazott tábla segítségével:

```
CREATE TYPE KoordTipus AS OBJECT (num NUMBER, x REAL, y REAL);
CREATE TYPE KoordTabla AS TABLE OF KoordTipus;
CREATE TABLE Megye
( megyekód INTEGER PRIMARY KEY,
  megyenév VARCHAR(20),
  poligon KoordTabla
);
CREATE TABLE Baleset
( id INTEGER PRIMARY KEY,
  dátum DATE,
  jelleg CHAR(30),
  x REAL, y REAL
);
```

c) térbeli adattípusok segítségével:

```
CREATE TABLE Megye
( megyekód INTEGER PRIMARY KEY,
  megyenév VARCHAR(20),
  poligon POLYGON
);
CREATE TABLE Baleset
( id INTEGER PRIMARY KEY,
  dátum DATE,
  jelleg CHAR(30),
  hely POINT
);
```

A lekérdezés:

```
SELECT megyenév, COUNT(*) AS balesetszám
  FROM Megye, Baleset
  WHERE WITHIN(Baleset.hely, Megye.poligon)
  GROUP BY megyenév;
```

Függelék

F1. Grafikus formátumok

F1.1. A TIFF grafikus formátum

TIFF = Tagged Image File Format, az 1980-as évek végén kidolgozott raszteres képformátum. A kép leírására tag-eket (címkéket) használ, ami nagyfokú rugalmasságot biztosít. A 6.0 verzió teljes angol nyelvű leírása letölthető a [TIFF] webhelyről, itt csak a legfontosabb jellemzőket tárgyaljuk. A TIFF fájl felépítése:

- header,
- IFD, amely a képet leíró tag-eket tartalmazza,
- maga a kép (pixelek sorozata).

Az IFD általában megelőzi a képet, de ez nem szükségszerű. Egy TIFF fájl több IFD-t is tartalmazhat.

A header 8 byte-ból áll, felépítése:

- 0-1. byte: bájtsorrend, két lehetséges értéke:

a) Ascii 'II' = hexa 4949: SHORT, LONG adatokban a kisebb című byte a kisebb helyértékű. Ezt a konvenciót „little endian”-nak nevezik, a PC processzorok így működnek.

b) Ascii 'MM' = hexa 4D4D: SHORT, LONG adatokban a kisebb című byte a nagyobb helyértékű. Ezt a konvenciót „big endian”-nak nevezik, így működnek a Sun és Motorola processzorok.

- 2-3. byte: verziószám, általában 42 (decimális).
- 4-7. byte: az első IFD kezdőpointere (byte sorszám a fájl kezdetétől számítva).

Image File Directory (IFD):

- 0-1. byte: tag-ek száma
- tag-ek felsorolása

Egy tag felépítése (12 byte):

- 0-1. byte: tag azonosítószám
- 2-3. byte: tag típusa (1 = BYTE, 2 = ASCII, 3 = SHORT, 4 = LONG, 5 = RATIONAL)
- 4-7. byte: hossz (hány db fenti típusú érték van)
- 8-11. byte: a tag értéke (ha elfér 4 byte-on), vagy az érték(ek) kezdőpointere (byte sorszám a fájl kezdetétől számítva).

Fontosabb tag típusok:

megnevezés	*azonosító*	*típus*	*hossz*
kép szélesség	256	SHORT vagy LONG	1
kép magasság	257	SHORT vagy LONG	1
bit-per-érték	258	SHORT	érték-per-pixel
tömörítésmód	259	SHORT	1
(1: tömörítetlen, 2...6: különféle tömörítő eljárások)			
foto-interpr.	262	SHORT	1
(0: fehéret jelent a 0 érték, 1: feketét jelent a 0 érték)			
kép kezdőpoint.	273	SHORT vagy LONG	sávok száma
érték-per-pixel	277	SHORT	1
sáv sorok száma	278	SHORT vagy LONG	1

Példa:

```
Header: 'II', 42, 8
IFD:    6, 256, 4, 1, 2000
           257, 4, 1, 3000
           258, 3, 1, 1
           259, 3, 1, 1
           262, 3, 1, 1
           273, 4, 1, 110
képpontok sorfolytonosan (8 pixel bájtonként)
```

F1.2. A GeoTIFF formátum

Az 1990-es évek közepén definiált, speciális tag-ekkel bővített, térinformatikai célú TIFF formátum. A GeoTIFF kép olyan programokkal is megjeleníthető, amelyek csak az alap TIFF formátumot ismerik, de ezek természetesen nem tudják értelmezni a speciális címkéket.

Sok új privát címke bevezetése helyett egy speciális címkét, a 34735 azonosítójú *GeoKeyDirectoryTag*-et definiálták. Ezen keresztül lehet elérni az ún. kulcsokat (geoKeys), amelyek a kép térinformatikai leírását adják.

A geoTIFF formátumot elsősorban *georeferencia* leírására használják: ez lényegében az alkalmazott vetületi rendszer leírását jelenti, amely segítségével az egyes pixeleknek megfelelő vetületi koordináták meghatározhatók.

F1.3. A DXF formátum

Az AutoCAD rendszer fejlesztője, az AutoDesk cég által specifikált vektoros adatcsere formátum (DXF = Drawing eXchange File format). Specifikációja az AutoCAD verzióit követi, a teljes angol nyelvű dokumentációk letölthetők a [DXF] webhelyről. Szöveges és bináris változata van, a szövegeset tárgyaljuk. A fájl kétsoros csoportokból áll, egy csoport felépítése:

- csoportkód,
- érték.

Fontosabb csoportkódok:

- 0: fájl elválasztó (vagy elem, táblabejegyzés kezdete)
- 1: egy elem szöveg értéke
- 2: szekciónév, blokknév, attribútum címke, stb.
- 6: vonaltípus név (pl. CONTINUOUS)
- 9: változónév azonosító (a headerben használatos)
- 10, 11, ..., 18: X koordináta
- 20, 21, ..., 28: Y koordináta
- 30, 31, ..., 38: Z koordináta (kétdimenziós rajz esetén nem használatos)
- 40, 41, ..., 48: lebegőpontos érték

A DXF fájl az alábbi szekciókból áll:

- HEADER: változók beállítása (koordinátarendszer, stb.)
- TABLES: többek között vonaltípusok, rétegek megadására szolgál.
- BLOCKS: blokk definíciókat tartalmaz.
- ENTITIES: rajzelemek leírása.

Header: a változónevek $ jellel kezdődnek. Például az alábbi változók használhatók (a változónév után a változó értékének csoportkódját adjuk meg, alkalmazását lásd alább a minta DXF fájlban):

$ACADVER 1: AutoCAD verziószám, például az AC1006 érték 10-es verziónak felel meg.
$EXTMIN 10, 20, 30: rajzterjedelem bal alsó sarokpontja
$EXTMAX 10, 20, 30: rajzterjedelem jobb felső sarokpontja
$LIMMIN 10, 20: rajzhatár bal alsó sarokpontja
$LIMMAX 10, 20: rajzhatár jobb felső sarokpontja
$TEXTSIZE 40: alapértelmezett szövegmagasság

Entities: többek között az alábbi rajzelemek használhatók:
LINE 10, 20, 30 (kezdőpont), 11, 21, 31 (végpont)
CIRCLE 10, 20, 30 (középpont), 40 (sugár)
TEXT 10, 20, 30 (beillesztési pont), 40 (magasság), 1 (szöveg)
POLYLINE: vonallánc kezdete
SEQEND: vonallánc vége
VERTEX 10, 20, 30: vonallánc töréspontja
(a POLYLINE és SEQEND között tetszőleges számú töréspont adható meg)

Minta DXF fájl (demo.dxf):

```
  0
SECTION
  2
HEADER
  9
$ACADVER
  1
AC1006
  9
$EXTMIN
 10
111.134400
 20
22.550900
  9
$EXTMAX
 10
112.414500
 20
23.709600
  9
$LIMMIN
 10
110.134400
 20
21.550900
  9
$LIMMAX
 10
113.414500
 20
22.709600
  9
$TEXTSIZE
 40
0.2
  0
ENDSEC
  0
SECTION
  2
TABLES
  0
ENDSEC
  0
SECTION
  2
BLOCKS
  0
ENDSEC
  0
SECTION
  2
ENTITIES
  0
LINE
  8
0
 62
1
  6
CONTINUOUS
 10
111.134400
 20
23.491500
 11
112.414500
 21
23.709600
  0
POLYLINE
  8
0
  6
CONTINUOUS
 66
     1
 70
     0
  0
VERTEX
  8
0
  6
CONTINUOUS
 10
111.556500
 20
22.796300
  0
VERTEX
  8
0
  6
CONTINUOUS
 10
112.332700
 20
23.368800
  0
VERTEX
  8
0
  6
CONTINUOUS
 10
112.387200
 20
22.591800
  0
VERTEX
  8
0
  6
CONTINUOUS
 10
111.570200
 20
22.550900
  0
SEQEND
  8
0
  0
ENDSEC
  0
EOF
```

F1.4. Az ArcView shapefile

A shapefile a népszerű ArcView térinformatikai rendszer adatformátuma. Valójában három fájlból áll:

*.SHP: térbeli objektumok leírását tartalmazó fájl.
*.SHX: indexfájl, a térbeli objektumok adatainak gyors elérését támogatja.
*.DBF: dBase formátumú adattábla, az egyes térbeli objektumokhoz tartozó leíró adatokat tartalmazza.

A három fájl alapneve meg kell, hogy egyezzen, csak a kiterjesztésben különböznek. A három fájl adatelemeinek darabszáma és sorrendje szigorúan megegyezik. Alább részletezzük az egyes fájlok felépítését.

Az SHP fájl felépítése

– 100 byte header. Többek között a térbeli objektumok típusát tartalmazza. Lehetséges főbb típusok: pont, ponthalmaz, vonallánc, poligon. Egy SHP fájl azonos típusú objektumokat kell, hogy tartalmazzon. (Később ez a korlátozás feloldásra kerülhet.)
– *n* db változó hosszúságú rekord.

Egy rekord felépítése:

– 8 byte rekord header. 0-3. byte: rekord sorszám (1-től kezdve), 4-7. byte: rekordhossz 16-bites szavakban.
– rekord tartalom: 0-3. byte: alakzat típusa, utána az alakzat leírása. Az alakzat típusa null is lehet (definiálatlan alakzat, amelynek esetleg később adjuk meg a geometriáját). Például pont leírása: 0-3. byte: 1 (pont típus), 4-11. byte: X koordináta (double), 12-19. byte: Y koordináta (double). A vonallánc és a poligon több komponensből állhat, leírásuk jóval összetettebb, itt nem részletezzük.

Az SHX fájl felépítése

– 100 byte header
– *n* db 8-byte-os rekord

Az *i*-edik rekord felépítése:

– offset: az SHP fájlbeli i-edik objektum leírásának kezdőcíme a fájl kezdetétől 16-bites szavakban számítva.
– hossz: megegyezik az SHP fájl objektum-headerében tárolt hossz értékkel.

A DBF fájl felépítése

Szabványos dBase adatstruktúra, egy *n* sorból és *m* oszlopból álló relációs adattáblát tartalmaz. Komponensei:

dBase header (teljes hossz 32 + *m**32 + 1, ahol *m* a mezők száma):

0. byte: verziószám
1-3. byte: dátum
4-7. byte: rekordok száma (long)
8-9. byte: fejrész hossza byte-ban (short)
10-11. byte: rekordhossz byte-ban (short), töröltségi jelzéssel együtt
12-31. byte: fenntartott
32. byte-tól: mezőleíró tömb (m*32 byte)
utolsó byte: mezőterminátor 0Dh

A mezőleíró tömb egy elemének fontosabb komponensei:

0-10. byte: mezőnév (ASCII nullával kiegészítve).
11. byte: mezőtípus: C = karakter, N = numerikus, L = logikai, D = dátum, M = memo.
16. byte: mezőhossz binárisan.

*Adatrekordok (*n *darab).* Egy rekord felépítése: a kezdő byte töröltség jelzést tartalmaz (2Ah = törölt, szóköz = nem), utána a rekord mezői tömören.

Fájl vége jel: ASCII 26 (1Ah)

Az ArcView shapefile formátum részletesebb leírását a következő dokumentum tartalmazza: ESRI Shapefile Technical Description, an ESRI white paper, July 1998 (letölthető az [ESRI] honlapról).

F2. A MySQL térbeli adatkezelése

A MySQL népszerű, nyílt forráskódú adatbázis-kezelő rendszer [MySQL]. Térbeli adatkezelése alapvetően az OGC specifikációt követi. Az SRID koordinátarendszer azonosító alapértelmezés szerint -1 (definiálatlan). Az alábbi térbeli adattípusok használhatók:

GEOMETRY: tetszőleges típus
POINT
LINESTRING
POLYGON
MULTIPOINT
MULTILINESTRING
MULTIPOLYGON
GEOMETRYCOLLECTION: tetszőleges objektumok halmaza

F2.1. Példa térbeli oszlop használatára

```
CREATE TABLE Utca
( id INTEGER,
  név VARCHAR(30),
  geom LINESTRING
);
```

Az Utca tábla feltöltése:

```
INSERT INTO Utca (id, név, geom)
VALUES (3, 'Kő utca', GeomFromText('LINESTRING(101 200,191 242)'));
```

Feltöltés változó segítségével:

```
SET @g = 'LINESTRING(101 200,191 242)';
INSERT INTO Utca (id, név, geom)
VALUES (3, 'Kő utca', GeomFromText(@g));
```

Lekérdezés:

```
SELECT id, AsText(geom) FROM Utca
WHERE name='Kő utca';
```

Eredmény:

```
LINESTRING(101 200,191 242)
```

F2.2. Példák függvények használatára

A függvények hibás argumentum esetén NULL értékkel térnek vissza.

Dimension (geom). Visszaadott érték: #1 (üres), 0, 1 vagy 2. Példa:

```
SELECT Dimension(geom) FROM Utca
    WHERE name='Kő utca';
```

Eredmény: 1

Envelope (geom). Visszaadott érték: befoglaló téglalap (Minimum Bounding Rectangle = MBR), poligon formájában. Példa:

```
SELECT AsText(Envelope(geom))
      FROM Utca WHERE name='Virág utca';
```

Eredmény: `POLYGON((1 1,2 1,2 2,1 2,1 1))`

GeometryType (geom). Visszaadott érték: típus. Példa:

```
SELECT GeometryType(geom) FROM Utca WHERE name='Kő utca';
```

Eredmény: LINESTRING

GLength (linestring). Visszaadott érték: a vonallánc hossza (double). Példa:

```
SELECT GLength(GeomFromText ('LineString(1 2,3 2,6 2)'));
```

Eredmény: 5

NumPoints (linestring). Visszaadott érték: vonallánc szögpontjainak száma. Példa:

```
SELECT NumPoints(GeomFromText ('LineString(1 2,3 2,6 2)'));
```

Eredmény: 3

Area (polygon). Visszaadott érték: a poligon területe (double). Példa:

```
SET @poly = 'Polygon((0 0,0 3,3 0,0 0),(1 1,1 2,2 1,1 1))';
SELECT Area(GeomFromText(@poly));
```

Eredmény: 4

NumInteriorRings (polygon). Visszaadott érték: poligonbeli lyukak száma. Példa:

```
SET @poly ='Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1))';
SELECT NumInteriorRings(GeomFromText(@poly));
```

Eredmény: 1

MBRContains (g1,g2). Visszaadott érték: 1, ha g1 befoglaló téglalapja tartalmazza g2-t, 0 egyébként. Példa:

```
SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
SET @g2 = GeomFromText('Point(1 1)');
SELECT MBRContains(@g1,@g2), MBRContains(@g2,@g1);
```

Eredmény: 1, 0

F3. A PostgreSQL térbeli adatkezelése

A PostgreSQL nyílt forráskódú, objektum-relációs adatbázis-kezelő rendszer [Postgres]. Korábbi változata Postgres (Post Ingres) néven futott, fejlesztő: University of California at Berkeley, Computer Science Department. A térbeli adatok kezelését az OGC specifikációtól eltérő módon valósítja meg.

F3.1. Térbeli adattípusok

Típusnév	*Írásmód*	*Megjegyzés*
POINT	(x, y)	Pont
LINE	((x1, y1), (x2, y2))	Végtelen egyenes
LSEG	((x1, y1), (x2, y2))	Egyenesszakasz
BOX	((x1, y1), (x2, y2))	Téglalap (szemközti sarkok koordinátái)
PATH	((x1, y1),..., (xn, yn))	Zárt vonallánc: (xn, yn) összekötve (x1, y1)-gyel
	[(x1, y1),..., (xn, yn)]	Nyílt vonallánc
POLYGON	((x1, y1),..., (xn, yn))	Poligon
CIRCLE	< (x, y), r >	Kör (középpont, sugár)

Megjegyzések:
– ((x1, y1), (x2, y2)) helyett (x1, y1), (x2, y2) vagy x1, y1, x2, y2 írásmód is alkalmazható.
– < (x, y), r > helyett ((x, y), r) vagy (x, y), r vagy x, y, r írásmód is alkalmazható.

Típuskonstruktorok:

box'((10,5),(16,25))'	téglalap
lseg'((10,5),(16,25))'	egyenesszakasz
'((10,5),(16,25))'::box	téglalap
'((10,5),(16,25))'::lseg	egyenesszakasz

F3.2. Térbeli műveletek

A PostgreSQL a térbeli adattípusokra igen sokféle műveletet definiál, itt csak néhány fontosabbat mutatunk be.

Egyenesszakaszok metszéspontja: lseg # lseg → point
Példa: ’((1,-1),(-1,1))’ # ’((1,1),(-1,-1))’
Vonallánc és alakzat metszése: lseg ?# geo → boole
Példa: lseg’((-1,0),(1,0))’ ?# box’((-2,-2),(2,2))’
Poligonok metszése: polygon && polygon → boole
Példa: ’((1,1),(1,2),(2,1))’&&’((1,1),(2,2),(2,1))’
Poligon vagy vonallánc szögpontjainak száma: # polygon → integer
Példa: #’((1,0),(0,1),(-1,0))’

Pont eleme-e egy alakzatnak: point @ geo → boole
Példa: point'(1,1)' @ circle'((0,0),2)'
Geo1-hez legközelebbi pont geo2-n: geo1 ## geo2 → point
Példa: point'(0,0)' ## lseg'((2,0),(0,2))'

F3.3. Térbeli függvények

A nagyszámú térbeli függvény közül néhány fontosabb:

Terület: area(geom) → double
Példa: area(box'((0,0),(1,1))')
Hossz: length(geom) → double
Példa: length(path'((-1,0),(1,0))')
Középpont: center(geom) → point
Példa: center(box'((0,0),(1,2))')
Téglalap magassága: height(box) → double
Példa: height(box'((0,0),(1,1))')
Téglalap szélessége: width(box) → double
Példa: width(box'((0,0),(1,1))')

Konverziós függvények

Pontpárból téglalap: box(point, point) → box
Példa: box(point'(0,0)',point'(1,1)')
Téglalapból poligon: polygon(box) → polygon
Példa: polygon(box'((0,0),(1,1))')
Poligon befoglaló téglalapja: box(polygon) → box
Példa: box(polygon'((0,0),(1,1),(2,0))')
Vonalláncból poligon: polygon(path) → polygon
Példa: polygon(path'((0,0),(1,1),(2,0))')
Poligonból vonallánc: path(polygon) → path
Példa: path(polygon'((0,0),(1,1),(2,0))')

F3.4. Példák

Alábbiakban a 8.2.2. alfejezetben definiált mintaadatbázison mutatjuk be a PostgteSQL térbeli adatkezelését.

Adatbázis feltöltése

```
INSERT INTO Megye(megyeKód,megyeNév,geo)
  VALUES (12,'Csongrád','((1,2),(1,5),(6,5))');
INSERT INTO Szakasz(szKód,szNév,geo)
  VALUES (112,'B24','((10,15),(16,5))');
```

...stb.

Lekérdezések

Adott megye területét metsző utak listája:

```
SELECT DISTINCT útNév
FROM Megye, Út, Szakasz, Útszakasz
WHERE ÚtSzakasz.útSzám = Út.útSzám AND
  ÚtSzakasz.szkód = Szakasz.szkód AND
  megyeNév = 'Csongrád' AND Szakasz.geo ?# Megye.geo;
```

Hibakeresés: egymást metsző megyék listája:

```
SELECT M1.megyeNév, M2.megyeNév
FROM Megye M1, Megye M2
WHERE M1.geo && M2.geo AND M1.megyeKód < M2.megyeKód;
```

Adott téglalapot metsző megyék listája:

```
SELECT * FROM Megye WHERE
'((1,1),(1,2),(2,2),(2,1))'::polygon && geo;
```

F4. A PostGIS térbeli adatkezelése

A PostGIS a PostgreSQL térbeli kiterjesztése, az OGC specifikációt követi. Komolyabb térinformatikai alkalmazásokhoz szükséges (például MapServer, lásd 10.2. fejezet). A térbeli adatok leírására a PostGIS metaadat-táblákat használ (ezek részei az OGC specifikációnak, de például a MySQL nem alkalmazza őket):

- SPATIAL_REF_SYS: vetületi rendszerek leírására,
- GEOMETRY_COLUMNS: táblák térbeli oszlopainak leírására.

Ez utóbbi táblára szükségünk lesz, ezért definícióját alább megadjuk:

```
CREATE TABLE Geometry_Columns
( f_table_catalog VARCHAR(256) NOT NULL,
  f_table_schema VARCHAR(256) NOT NULL,
  f_table_name VARCHAR(256) NOT NULL,
  f_geometry_column VARCHAR(256) NOT NULL,
  coord_dimension INTEGER NOT NULL,
  srid INTEGER NOT NULL,
  type VARCHAR(30) NOT NULL
);
```

Az egyes attribútumok jelentése:

f_table_catalog, f_table_schema: katalógus információk.
f_table_name: a térbeli oszlopot tartalmazó tábla neve.
f_geometry_column: a térbeli oszlop neve.
coord_dimension: 2, 3 vagy 4.
srid: koordináta-rendszer azonosító.
type: Ha a térbeli oszlop minden eleme azonos típusú, akkor POINT, LINESTRING, POLYGON, MULTIPOINT, MULTILINESTRING, MULTIPOLYGON vagy GEOMETRYCOLLECTION. Vegyes típusú oszlop esetén GEOMETRY.

F4.1. Térbeli oszlop használata

Először definiáljuk a táblát térbeli oszlop nélkül, például:

```
CREATE TABLE Utca
( id INTEGER,
  név VARCHAR(30)
);
```

Térbeli oszlop hozzáadása:

```
SELECT AddGeometryColumn('Utca','geom',-1, 'LINESTRING',2);
```

Ezzel új sort veszünk fel a Geometry_Columns táblába, ahol rendre az f_table_name, f_geometry_column, srid, type és coord_dimension mezőknek adunk értéket. A többi mező az alapértelmezett értéket kapja.

Az Utca tábla feltöltése:

```
INSERT INTO Utca (id, név, geom)
VALUES (3, 'Kő utca', GeomFromText('LINESTRING(101 200,191 242)', -1));
```

Lekérdezés:

```
SELECT id, AsText(geom) FROM Utca WHERE name='Kő utca';
```

A lekérdezés eredménye:

```
id | geom
---+----------------------------
 3 | LINESTRING(101 200,191 242)
```

F5. Az Oracle Spatial térbeli adatkezelése

Az Oracle adatbázis-kezelő rendszer Spatial nevű kiegészítő modulja támogatja a térbeli adatok kezelését [Oracle Spatial]. A megvalósítás objektum-relációs elven alapul, vagyis a 7.2. fejezetben tárgyalt SQL eszközöket használja. AZ OGC specifikációtól eltérően egyetlen univerzális objektumtípust használ SDO_GEOMETRY néven (SDO a Spatial Data Option rövidítése). Egy térbeli adatot tartalmazó táblának legalább két oszlopa van:

- Elsődleges kulcs,
- SDO_GEOMETRY típusú oszlop, amely vegyesen tartalmazhat különféle elemtípusokat.

Megjegyezzük, hogy 3D és 4D koordináták is megengedettek, de a függvények többsége csak 2D-re működik.

F5.1. SDO_GEOMETRY definíciója

Mivel az SDO_GEOMETRY univerzális típus, vagyis valamennyi térbeli adattípus leírását lehetővé teszi, így felépítése meglehetősen bonyolult:

```
CREATE TYPE sdo_point_type AS OBJECT
  (X NUMBER, Y NUMBER, Z NUMBER);
CREATE TYPE sdo_elem_info_array AS VARRAY (1048576) of NUMBER;
CREATE TYPE sdo_ordinate_array AS VARRAY (1048576) of NUMBER;
```

```
CREATE TYPE SDO_GEOMETRY AS OBJECT
( SDO_GTYPE NUMBER,
  SDO_SRID NUMBER,
  SDO_POINT SDO_POINT_TYPE,
  SDO_ELEM_INFO SDO_ELEM_INFO_ARRAY,
  SDO_ORDINATES SDO_ORDINATE_ARRAY
);
```

Alábbiakban az egyes komponenseket részletezzük.

SDO_GTYPE: Az alakzat típusát adja meg 4 jegyű, „`dltt`” alakú szám formájában, ahol

d = dimenzió (2, 3, 4)

l = 0 ha nem LRS alakzat. (LRS = linear referencing system = lineáris címzési rendszer, lásd 3.5.2. alfejezetben. l = 3 vagy 4 adja meg, hogy melyik dimenzió tartalmazza a lineáris távolságot.)

tt = alakzattípus (00...07). Lehetséges értékei:

00 = ismeretlen

01 = pont

02 = vonallánc, amely íves szakaszokat is tartalmazhat

03 = poligon, esetleg lyukakkal (először a külső, azután a belső határvonalat kell megadni.)

04 = kollekció: tetszőleges elemek halmaza.

05 = ponthalmaz.

06 = vonallánc halmaz.

07 = poligonhalmaz (diszjunkt poligonok).

Példa: SDO_GTYPE = 2003 = 2D poligon, mivel

d = dimenzió = 2.

l = 0: nem LRS alakzat.

tt = 03 = poligon.

Példa: SDO_GTYPE = 3302 = LRS vonallánc:

d = dimenzió = 3 (valójában 2D alakzatról van szó, a harmadik koordináta az LRS-hez kell).

l = 3: LRS alakzat, ahol a harmadik koordináta adja az adott pont lineáris távolságát a vonallánc kezdőpontjától.

tt = 02 = vonallánc.

Egy tábla egy oszlopában csak azonos dimenziójú alakzatok lehetnek. Az egyes komponenseket visszaadó függvények: Get_Dims, Get_LRS_Dim, Get_Gtype.

SDO_SRID: Koordinátarendszer (Spatial Reference System) azonosítója: az SDO_COORD_REF_SYS tábla SRID oszlopára hivatkozik. Egy tábla egy oszlopában csak azonos SDO_SRID értékek szerepelhetnek. SDO_SRID értéke NULL is lehet (nem definiálunk koordinátarendszert).

SDO_POINT: Pont alakzat megadására szolgál, ekkor az SDO_ELEM_INFO és SDO_ORDINATES attribútumok NULL értékűek. (Nem pont alakzat esetén az SDO_POINT értéke közömbös.)

SDO_ELEM_INFO: A koordináták értelmezésére szolgál. Számhármasokból áll, minden számhármas egy alkotóelem leírását tartalmazza:

- SDO_STARTING_OFFSET: az alkotóelemhez tartozó első koordináta sorszáma az SDO_ORDINATES tömbben (1-től számozva).
- SDO_ETYPE: az alkotóelem típusa (lásd alább).
- SDO_INTERPRETATION. Ha SDO_ETYPE összetett elem, akkor a hármasok száma, ha nem, akkor a koordináták értelmezése (lásd alább).

ETYPE	INTERP.	Jelentés
1	1	Pont
1	n	Ponthalmaz
2	1	Vonallánc egyenesszakaszokkal
2	2	Vonallánc körívekkel
1003	1	Külső poligon
2003	1	Belső poligon
1003	2	Külső poligon körívekkel
2003	2	Belső poligon körívekkel
1003	3	Külső téglalap
2003	3	Belső téglalap
1003	4	Külső kör
2003	4	Belső kör
4	n	Vegyes vonallánc (egyenesek, körívek)
1005	n	Külső vegyes poligon (egyenesek, körívek)
2005	n	Belső vegyes poligon (egyenesek, körívek)

SDO_ORDINATES: Koordináták felsorolása, nem tartalmazhat NULL értéket.

Szabályok

Poligon megadása esetén az első pont és az utolsó pont meg kell, hogy egyezzen.
Példa: 4 szögpontú poligon: (X1, Y1, X2, Y2, X3, Y3, X4, Y4, X1, Y1)
Külső poligon: óramutató járásával szemben körüljárva.
Példa: háromszög: (1,1, 8,1, 1,5, 1,1)
Belső poligon: óramutató járása szerint körüljárva.
Példa: háromszög: (1,1, 1,5, 8,1, 1,1)
Körív megadása: kezdőpont, belső pont, végpont.
Körívlánc megadása: csatlakozó pontot nem ismételjük.
Példa: (1,1, 1,5, 5,8, 10,10, 12,10)
Kör megadása: három kerületi ponttal.
Téglalap megadása: bal alsó és jobb felső sarok koordinátáival.
Példa: (1,2, 10,20)

Példák

1. példa: pont

```
SDO_GTYPE = 2001.      2D, nem LRS, pont
SDO_SRID = NULL.
```

```
SDO_POINT = (10,20).
SDO_ELEM_INFO = NULL.
SDO_ORDINATES = NULL.
```

2. példa: pont felvétele SQL-ben

```
CREATE TABLE Város
( kód NUMBER PRIMARY KEY,
  név VARCHAR2(20),
  hely SDO_GEOMETRY );
INSERT INTO Város VALUES(152,'Pécs',
  SDO_GEOMETRY(2001, NULL, SDO_POINT_TYPE(1235, 4178, NULL),
  NULL, NULL));
```

3. példa: téglalap

```
SDO_GTYPE = 2003    2D, nem LRS, poligon
SDO_SRID = NULL.
SDO_POINT = NULL.
SDO_ELEM_INFO = (1,1003,3)  Külső téglalap
SDO_ORDINATES = (1,1, 5,7)  Bal alsó, jobb felső
```

4. példa: poligon lyukkal

```
SDO_GTYPE = 2003    2D, nem LRS, poligon
SDO_SRID = NULL.
SDO_POINT = NULL.
SDO_ELEM_INFO = (1,1003,1, 19,2003,1)
```

Megjegyzés: 1003: külső poligon, 2003: belső poligon, ez utóbbi a 19. pozíción kezdődik, első pontja (7,5).

```
SDO_ORDINATES = (2,4, 4,3, 10,3, 13,5, 13,9, 11,13, 5,13, 2,11, 2,4,
7,5, 7,10, 10,10, 10,5, 7,5).
```

5. példa: vegyes vonallánc

```
SDO_GTYPE = 2002.      2D vonallánc
SDO_SRID = NULL.
SDO_POINT = NULL.
SDO_ELEM_INFO = (1,4,2, 1,2,1, 3,2,2)
```

Magyarázat: (1,4,2): vegyes vonallánc 2 komponenssel, amely komponensek a következők. (1,2,1): vonallánc egy vagy több egyenesszakasszal, amely a következő pointerig (3) tart. (3,2,2): vonallánc egy vagy több körívvel, amely a tömb végéig tart. A csatlakozó pontot (10,14) nem ismételjük.

```
SDO_ORDINATES = (10,10, 10,14, 6,10, 14,10).
```

6. példa: poligon felvétele SQL-ben

```
CREATE TABLE Telek
( helyrajziszám CHAR(10) PRIMARY KEY,
  terület INTEGER,
  geom SDO_GEOMETRY );
INSERT INTO Telek VALUES
(123, 1200, SDO_GEOMETRY(2003, NULL, NULL,
   SDO_ELEM_INFO_ARRAY(1,1003,1),
   SDO_ORDINATE_ARRAY(2,4, 10,3, 13,5, 13,9, 5,13, 2,4))
);
```

F5.2. Függvények

A számos függvény és operátor közül itt csak kettőt mutatunk be.

SDO_DISTANCE (geom1, geom2, tolerance [, params])

A geom1 és geom2 alakzatok minimális távolságát (azaz legközelebbi pontjaik távolságát) adja vissza.

tolerance: kerekítési hibák elkerülésére szolgáló tolerancia paraméter, szokásos értéke 0,5. (Ha például a két alakzat távolsága kisebb, mint tolerance, akkor a távolság nulla lesz.)

params: a távolság mértékegységének megadására szolgál. Elhagyható, ekkor a koordinátarendszerhez megadott mértékegység érvényes.

RELATE (geom1, mask, geom2, tolerance)

Két alakzat között fennálló geometriai kapcsolatot határoz meg, ahol mask a meghatározandó kapcsolatot leíró karakterlánc, értékei a következők lehetnek:

1) DETERMINE: ebben az esetben a függvény határozza meg a kapcsolat típusát, amelyet stringként ad vissza (például 'INSIDE').

2) A következők egyike: INSIDE, COVEREDBY, COVERS, CONTAINS, EQUAL, OVERLAPBDYDISJOINT, OVERLAPBDYINTERSECT, ON, TOUCH. A függvény értéke a maszkként megadott kapcsolat típusa (ha teljesül), vagy 'FALSE' (ha nem teljesül).

3) ANYINTERACT: a visszaadott érték 'TRUE', ha bármely a 2) pontban leírt tulajdonság fennáll, 'FALSE' egyébként.

4) DISJOINT: a visszaadott érték 'TRUE', ha 2) pontban felsorolt tulajdonságok egyike sem áll fenn, 'FALSE' egyébként.

F5.3. Lekérdezések

Tekintsük a 8.2.1. alfejezet mintaadatbázisának alábbi két sémáját:

Telek (helyrajziszám, terület, geom)
Kút (id, típus, geom)

A megfelelő SQL deklarációk:

```
CREATE TABLE Telek
( helyrajziszám CHAR(10) PRIMARY KEY,
  terület       INTEGER,
  geom          SDO_GEOMETRY
);
CREATE TABLE Kút
( id    CHAR(10) PRIMARY KEY,
  típus VARCHAR2(20),
  geom  SDO_GEOMETRY
);
```

Kérjünk le listát távolság szerint növekvő sorrendben azon kutakról, amelyek az 1234 helyrajzi számú telektől legfeljebb 100 koordinátaegységnyire vannak:

```
SELECT id, típus, SDO_DISTANCE(Telek.geom, Kút.geom, 0.5) AS távolság
  FROM Telek, Kút
  WHERE Telek.helyrajziszám=1234 AND
  SDO_GEOM.SDO_DISTANCE (Telek.geom, Kút.geom, 0.5) < 100
  ORDER BY távolság;
```

Kérjünk le listát azon telkekről, amelyeken artézi kút van:

```
SELECT helyrajziszám FROM Telek, Kút
  WHERE Kút.típus='Artézi' AND
  SDO_GEOM.RELATE (Kút.geom, 'INSIDE', Telek.geom, 0.5) = 'INSIDE';
```

Irodalomjegyzék

Bana István: *Az SSADM rendszerszervezési módszertan.* LSI Oktatóközpont, 1995.

Dózsa Olivér: Térképi adatok kezelése és megjelenítése az Oracle Spatial használatával. Diplomamunka, SZTE, 2010.

Elek István: *Bevezetés a geoinformatikába.* ELTE Eötvös Kiadó, 2006.

Gruber M.: *SQL A-Z.* Kiskapu kiadó, 2003.

Katona E.: *Automatikus térkép interpretáció.* PhD értekezés, Szegedi Tudományegyetem (2001).

Klinghammer I., Papp-Váry Á.: *Földünk tükre a térkép.* Gondolat Kiadó, 1983

Kothuri, R. V., A. Godfrind, E. Beinat: *Pro Oracle Spatial for Oracle Database 11g.* Apress, 2007.

Kropla, B.: *Beginning MapServer – Open Source GIS Development*. Apress, 2005.

Lerner János: *Térképészeti alapismeretek.* ELTE jegyzet, 1989.

Loney, K.: *Oracle database 10g teljes referencia.* Panem, 2006. (1448 oldal)

NCGIA Core Curriculum, magyar fordítás (az eredeti kiadás szerkesztői M. F. Goodchild és K. K. Kemp). EFE FFFK, Székesfehérvár. (1994)

Rigaux, Ph., Scholl, M., Voisard, A.: *Spatial databases, with application to GIS.* Morgan Kaufmann Publishers, San Francisco, 2002.

Samet H.: *Design and Analysis of Spatial Data Structures*. Addison Wesley, 1989.

Stegena Lajos: *Vetülettan*. Tankönyvkiadó, Budapest, 1988.

Szrnka Éva: PostgreSQL adatbázisban tárolt térképi információ megjelenítése UMN MapServerrel. Diplomamunka, SZTE, 2008.

L-Tér Stúdió Kft: *Térinformatika lépésről lépésre (MicroStation/J, MS Geographics, MS Descartes)*. 2001

Ullman J. D., Widom J.: *Adatbázis rendszerek – Alapvetés.* Második, átdolgozott kiadás, Panem, 2008.

Vasas Szabolcs: Objektum-relációs eszközök hatékonyságának vizsgálata Oracle környezetben. Szakdolgozat, SZTE, 2010.

Yeung, A. K. W., G. B. Hall: *Spatial Database Systems: Design, Implementation and Project Management.* Springer (2007)

Zentai L.: *Számítógépes térképészet.* ELTE Eötvös Kiadó, 2000.

Zeiler, M.: *Modeling Our World. The ESRI's Guide to Geodatabase Design.* ESRI Press, 1999.

Webes információforrások

[DXF] DXF file format specification:
http://usa.autodesk.com/adsk/servlet/item?siteID=123112&id=12272454&linkID=10809853

[ESRI] Environmental Systems Research Institute: http://www.esri.com

[GISjegyzet] Belényesi Márta – Kristóf Dániel – Magyari Julianna: *Térinformatika elméleti jegyzet.* Szent István Egyetem, Mezőgazdaság- és Környezettudományi Kar, Gödöllő, 2008.
http://www.kti.szie.hu/TTTT/letoltes/terinformatika/elmeleti_jegyzet.pdf

[GoogleMaps] A Google világtérkép-keresőrendszere: http://maps.google.com

[OGC] Open Geospatial Consortium: http://www.opengeospatial.org

[MapServer] Az UMN MapServer honlapja: http://mapserver.org

[MySQL] MySQL 5.1 Reference Manual: http://dev.mysql.com/doc/

[Oracle Spatial] Oracle Spatial Developer's Guide
http://www.oracle.com/technology/products/spatial/spatial_doc_index.html

[Postgres] PostgreSQL 8.3.4 Documentation: http://www.postgresql.org/docs/

[TIFF] TIFF 6.0 file format specification (1992):
http://partners.adobe.com/public/developer/en/tiff/TIFF6.pdf

Ábrák jegyzéke

